Reduced CNN Model for Rotation-Invariant Classification

Rosemberg Rodriguez Salas

Université Gustave Eiffel

École doctorale nº 532 :
Mathematiques et Sciences et Technologies
de l'information et de la communication (MSTIC)

THÈSE

pour obtenir le grade de docteur délivré par

UNIVERSITÉ GUSTAVE EIFFEL

présentée par

Rosemberg Rodríguez Salas

le 5 Juillet 2021

Reduced CNN Model for Rotation-Invariant Classification

Jury

Mme Fan Yang, Professeur	Univ de Bourgogne	Examinatrice
M. Alejandro Castillo Atoche, Professeur	Univ Autónoma de Yucatán	Rapporteur
M. Nicolas Loménie, Associate Professeur	Univ Paris Descartes	Rapporteur
Mme Eva Dokladalova, Professeur	Univ Gustave Eiffel	Directrice de thèse
M. Petr Dokladal, Chargé de recherche	MINES Paris	Co-directeur de thèse

LIGM, Univ Gustave Eiffel
CNRS, ESIEE Paris, F-77454 Marne-la-Vallée

Acknowledgments

There are several people to thank who took part in this project and thesis writing. Without them, I might not have finished this Ph.D., and to whom I am greatly indebted.

Firstly, I would like to express my gratitude towards my director **Eva Dokladalova** and co-director, **Petr Dokladal**, for their continuous support and understanding. Their patience, motivation, and knowledge guided me from almost no knowledge of the field to finish this thesis. Moreover, while there existed discussions and different points of view, they always guided me with their best interests in mind and challenging me to achieve new levels. I will always be grateful for teaching me about what science is about and the spirit of the investigation.

Also, I would like to thank the jury board members **Fan Yang**, **Alejandro Castillo Atoche**, and **Nicolas Loménie**. Their valuable suggestions, time, and interest in my work were essential to improve and detail this thesis manuscript. Also, I thank them for their challenging questions, which intended to widen my research perspective.

Next, I would like to thank every person that was not directly on the stage but was there for me behind the scenes to motivate me, support me, and having the patience to endure with me in the hardest epochs. I genuinely think this work is thanks to you, and I would not be here if they had not accompanied me. Each one of you knows how much time you were there for me. This achievement is all yours.

Specifically, I would like to thank **Paula Salas Castillo**, my mother, who pushed me to take this opportunity in a new country with a different language and a challenging topic. She gave me enough peace of mind to focus on my research while knowing she was alone in my country. I want to dedicate this work and this achievement to you for your continuous support and faith in me even in times when I lost faith in myself.

Finally, I would like to sincerely thank **University Gustave Eiffel**, which gave me this opportunity. They always gave me the necessary research and personal resources to develop this work and finish this thesis manuscript. I hope that they continue to support Mexican students and help them to achieve their objectives.

Abstract

During the last years, the number of computer-vision-based industrial, automotive, and surveillance applications has grown exponentially. This trend is fostered by the success of Convolutional Neural Networks (CNNs) in this field. This success is related to their performances and also a notable advances in their efficient implementations. Nevertheless, their computing requirements remain very high and sometimes limiting. This is mainly due to the number of trainable parameters needed to provide generalization capabilities on complex data.

In this work, we specifically target the problem of rotation-invariant classification. The main objective is to reduce the size of a models capable of rotation-invariant classification without sacrificing accuracy and result quality. To do so, we propose to focus on the direct integration of the invariance capability to the frequent image transformation by rotation in the neural network architecture. We propose an original CNN architecture the first layer of which consists of a constrained collection of filters. These filters are rotated copies of each other, ordered by the angle of rotation, and form together a roto-translation space. A subsequent translating classifier is then used to perform the classification. The position of the maximum likelihood indicates the rotation angle of the input. We evaluate several filter families such as those from the scattering transform, steerable filters, and Gabor filters. We present experimental results on how the proposed methodological approach allows endowing backbone CNNs with rotation-invariant properties. We also provide a proof of trainability of such architecture.

We evaluate the results on simple and complex datasets by using custom network and one of the well-known, state-of-the-art, backbone network ResNet. We demonstrate that the proposed rotation invariance framework allows achieving state-of-the-art error rate results on simple datasets as MNIST (2.05% vs. 6.00% previously, when trained on up-right examples) and outperformed previous approaches on complex datasets as CIFAR-10 (21.50% vs. 55.88% previously) while reducing the number of trainable parameters from the current state-of-the-art methods by more than 50%. Furthermore, these results are obtained without data-augmentation techniques and with a novel angular prediction capabilities.

Thus, the presented results open the possibility to use these networks in resource-constrained devices such as embedded platforms and smartphones. Also, it opens the opportunity to use this concept in applications where the rotation-invariance can play its role but where one often encounters relatively limited datasets as aerial imagery, food recognition, and face recognition.

Résumé

Au cours des dernières années, le nombre d'applications industrielles, automobiles et de surveillance basées sur la vision par ordinateur a augmenté de façon exponentielle. Cette tendance est favorisée par le succès des réseaux de neurones convolutifs (CNN) dans ce domaine. Ce succès est lié à leurs performances et aussi à des avancées notables dans leurs implémentations efficaces. Néanmoins, leurs besoins informatiques restent très élevés et parfois limitants. Cela est principalement dû au nombre de paramètres entraînables nécessaires pour fournir des capacités de généralisation sur des données complexes.

Dans ce travail, nous ciblons spécifiquement le problème de la classification invariante par rotation. Nous proposons une architecture CNN originale dont la première couche est constituée d'un ensemble contraint de filtres. Ces filtres sont des copies orientées les uns des autres, sont ordonnés par l'angle de rotation et forment ensemble un espace roto-translationel. Un classifieur effectuant une translation est ensuite utilisé pour effectuer la classification. La position du maximum de probabilité indique l'angle de rotation de l'entrée. Nous évaluons plusieurs familles de filtres telles que celles utilisés dans la transformée scattering, des filtres orientables et des filtres Gabor. Nous présentons des résultats expérimentaux sur la façon dont l'approche méthodologique proposée permet de doter les architectures "backbone" de propriétés invariantes par rotation. Nous fournissons également une preuve de l'entraînablité d'une telle architecture.

L'objectif principal est de réduire la taille d'un modèle de classification invariante par rotation sans sacrifier la précision et la qualité des résultats. Pour ce faire, nous proposons de nous concentrer sur l'intégration directe de la capacité d'invariance à la transformation de l'image par rotation dans l'architecture du réseau neuronal.

Nous évaluons les résultats sur des ensembles de données simples et complexes en utilisant un réseau personnalisé et l'un des réseaux dorsaux bien connus et à la pointe de la technologie ResNet. Nous démontrons que le cadre d'invariance de rotation proposé permet d'obtenir des résultats de taux d'erreur de pointe sur des ensembles de données simples comme MNIST (2,05 % contre 6,00 % précédemment, quand formés sur des exemples en haut à droite) et surpassé les approches précédentes sur des ensembles de données complexes comme CIFAR-10 (21,50 % contre 55,88 % précédemment) tout en réduisant de plus de 50 % le nombre de paramètres pouvant être entraînés à partir des méthodes de pointe actuelles. De plus, ces résultats sont obtenus sans techniques d'augmentation des données et avec de nouvelles capacités de prédiction angulaire.

Ainsi, les résultats présentés ouvrent la possibilité d'utiliser ces réseaux dans des dispositifs aux ressources limitées tels que les plates-formes embarquées et les smartphones. En outre, cela ouvre l'opportunité d'utiliser ce concept dans des applications où l'invariance de rotation peut jouer son rôle mais où l'on rencontre souvent des ensembles de données relativement limités comme l'imagerie aérienne, la reconnaissance alimentaire et la reconnaissance faciale.

Table of contents

List of acronyms

AGV	Automated Guided Vehicles
ANN	Artificial Neural Network
AI	Artificial Intelligence
CIFAR	Canadian Institute For Advanced Research
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep learning
DSP	Digital Signal Process
FPGA	Field Programmable Gate Arrays
FPS	Frames per Second
GB	Gigabyte
GCN	Gabor Convolutional Networks
GPU	Graphics Processing Unit
IR	Intermediate Representation
KM-CNN	Kernel Mapping Convolutional Neural Networks
MNIST	Modified National Institute of Standards and Technology
NCS	Neural Computer Stick
NPU	Neural Processing Unit
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PloU	Pixels-Intersection-over-Union
RAM	Random Access Memory
ReLu	Rectified Linear Unit
RGB	Red Green Blue
RIN	Rotation Invariant Network
RRT	Randomly Rotated Training
SBC	Single Board Computer
SFCNN	Steerable Filter Convolutional Neural Network
SIMD	Single instruction multiple data
SoC	System on Chip
SSD	Single-shot detectors
TPU	Tensor Processor Unit
UAV	Unmanned Aerial Vehicles
URT	Upright Training
USB	Universal Serial Bus
VC	Vapnik-Chervonekis
VFN	Vector Field Networks
VGG	Visual Geometry Group
VPU	Visual Processing Unit
VRAM	Video random access memory

Chapter 1

Introduction

1.1 Computer Vision Technological Trends

In the last years, the number of computer vision-based applications has grown exponentially. Today, there should exist more than 1 billion cameras around the globe (Lin and Purnell [2019]) used for security surveillance in the major cities. Equally, the estimated number of hand-held camera equipment (i.e., smartphones and similar) goes up to 3.5 billion devices in the world (O'Dea [2020]). If we consider also computer vision-aided robots, autonomous and remote sensing systems, we expect that the number of devices with a camera to go up to more than 40 billion cameras in the world in 2022 (Fig. 1.1).

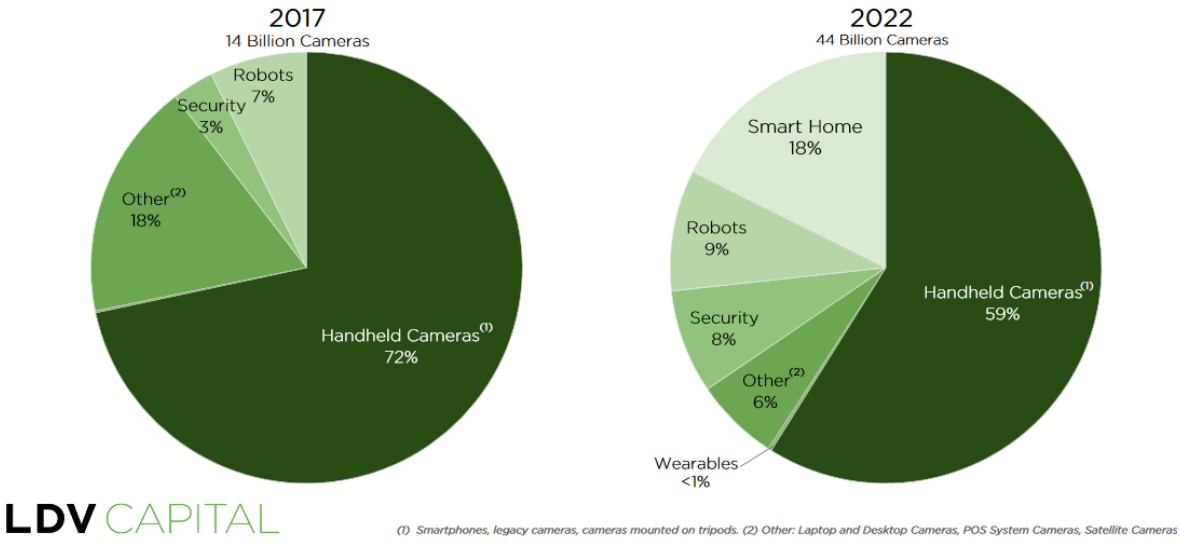


Figure 1.1 – Number of total cameras predicted for 2022 (Capital [2017]).

It is, of course, impossible to process images of the applications, mentioned briefly above, manually. Therefore, automated image analysis and understanding must be used. As a prime example of this automation, we can cite one of the most advanced surveillance networks in China that can process images from 170 million cameras around the city of Guiyang to find a person in less than 7 minutes thanks to face recognition (Liu [2017]). There are also applications for automatic detection of objects such as abandoned luggage in public spaces (Dahi et al.

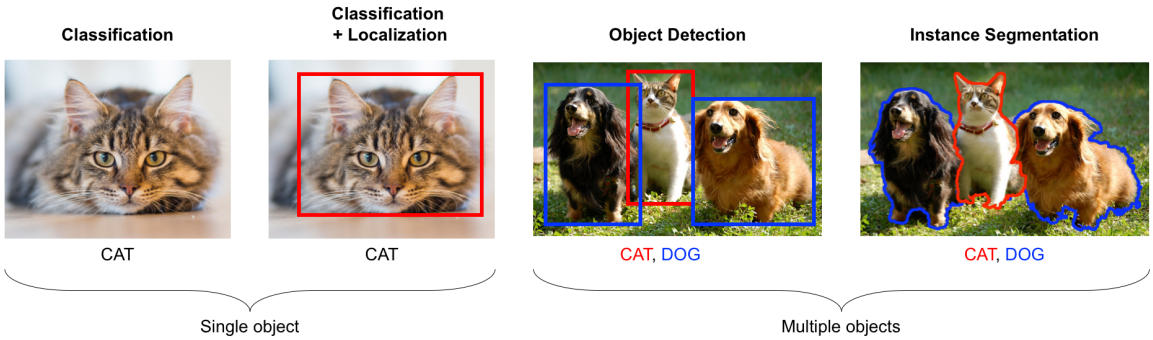


Figure 1.2 – Image recognition tasks (image by Shanmugamani [2018]).

[2017]), and there are many other possible illustrations such as remote sensing (Ren et al. [2018]), autonomous cars (Chen et al. [2016]), etc.

These automated methods are based on sequences of often complex computer vision algorithms allowing the understanding of image content such as image enhancement, motion detection, segmentation, pattern analysis, classification, and others.

The most frequently encountered tasks in understanding the content of images are classification, detection, and segmentation (Fig. 1.2). Image classification consists of assigning a label from a determined number of existing classes to an input image. Image detection is the process of finding the coordinates of given objects in the input image. Finally, image segmentation is the operation of grouping image pixels into regions according to predefined criteria. Usually, combining one or more of these methods has to be used to design real-life applications.

For a decade, the computer vision tasks mentioned above have been solved by artificial neural networks (ANNs) (Alzubaidi et al. [2021]). The ANNs derive their basic principle from the computational model of the artificial neuron first published in McCulloch and Pitts [1943]. One of the significant advantages of these networks is the generalization capability which allows obtaining correct results even with examples not included in the training stage.

Unfortunately, when applying these networks to the image processing domain, the number of neurons can increase rapidly. For example, the recent and popular DALL-E model has more than 12 billion parameters (Ramesh et al. [2021a]). DALL-E represents an extreme case, however, the size of the ANN model, in terms of the number of parameters to be learned, remains a major concern in the field. We discuss this problem in more detail in section 1.2.

Despite the problem of model size, and thanks to the technological advancement of computational resources, neural networks are recording widespread success throughout the field. This is due in particular to the introduction of convolutional neural networks (CNN) (LeCun et al. [1989]). In contrast to a classical perceptron, the CNNs have the advantage of being able to extract features directly from the input data. This feature extraction reduces the dimensionality of the problem and becomes more easily solved by the classifier. As a result, the CNNs have become the standard for all image processing problems. In the CNN family, especially the so-called deep learning models achieve unmatched

performance (Hinton [2009]). On the other hand, if these new models allow us to extract higher level features (Deng and Yu [2014]), they also increase the size of the networks and require high computing effort.

This chapter presents the existing considerations to implement deep learning networks in different hardware such as embedded devices, hand-held devices, and high-end servers. We present a catalog of applications and their target hardware to highlight the accuracy and bottlenecks found in these applications. Then, we outline the principles behind neural networks and visit a brief history of their evolution through the latest years. The objective of these three different points of view on deep learning utilization is to illustrate the context of this research work. In section 1.3 we present a proposal of a mechanism on how to reduce the size of deep learning models while preserving the accuracy. We exploit the rotation invariance mechanism, and we discuss that it enables new possibilities for embedded devices and small database applications.Finally, we present the contributions of this work and the manuscript organization.

1.1.1 Deployment and Computing Considerations of Deep Learning Models

Although the CNNs, and in particular deep learning have outperformed conventional computer vision approaches, they do have some drawbacks. We have already mentioned the size of the model. As we also stated, it is common to see deep learning models reaching millions of parameters. While these large networks can be implemented relatively easily in dedicated hardware such as GPUs and GPU clusters, it is always a challenge to implement them in devices with more constrained resources without losing the quality of the result.

At the same time, the demand for their on-board installations is proportional to the use of consumer electronics: for example, smart cameras for surveillance, drones, smartphones.

To address this challenge, hardware manufacturers, software developers and researchers have launched a joint effort to enable neural networks to function in such on-board devices and portable devices.

It should be noted that the embedding of deep learning models near the sensors has certain advantages known today under the term edge computing (Shi et al. [2016]). Its advantages are well known, we could mention some such as security issues for facial recognition applications, bandwidth reduction costs for sending multiple images to server or network latency issues for applications. at critical time (Shi and Dustdar [2016]).

In the following paragraphs, we present different considerations for deploying deep learning models on different families of hardware platforms. We also recall the various techniques used by the scientific community to reduce the cost of computing neural networks and allow their implementation on on-board devices. To complete, we present an overview of the sample applications for each platform family. The main objective is to highlight the hardware and software efforts that allow deep learning networks to operate in devices with limited resources and to better situate the contribution of this thesis thereafter.

Hardware Platforms and Deep Learning Applications

Embedded devices. In the category of embedded peripherals, we can essentially classify single board computers (SBC) like the Raspberry (Pi Pi [2021]), low power GPU-based embedded platforms like NVIDIA's Jetson Nano (NVIDIA [2021a]), USB accelerators like Google Coral products (Google [2020]) and Field Programmable Gate Arrays (FPGAs) boards (Xilinx [2021]).

Some of these devices are not initially intended for DL applications. The tendency is to equip these platforms with acceleration supports for DL models. The main technological barrier is the number of resources available on each board. Commonly, the board clock frequency is usually limited for the processor to limit the energy consumption of the board.

For example, the latest version of Raspberry Pi, Raspberry 4, includes a Quad-core Cortex-A72 (ARM v8) processor and can go up to 8 GB of RAM. Without any dedicated hardware support, a Raspberry 4 can run the MobileNetv1 (Allan [2019a]) neural network for object detection at four frames per second (fps) on average. Note that the Raspberry GPU is not used in deep learning applications because it is not compatible with the libraries most used for deep learning. Compared to the previously introduced Jetson Nano, the inference time is almost five times higher (Süzen et al. [2020]), with Rasperry making inference in 173 seconds versus 32 seconds on the DeepFashion2(5k) dataset (Ge et al. [2019]).

Another example of the SBC is the Jetson Nano. This board contains a 128-core MaxwellGPU and a 1.43GHz quad-core ARM A57 processor. Although still limited in resources, this card can run MobileNetv1 at 16 fps at a resolution of 300 x 300 pixels (Allan [2019a]). Several applications can benefit from this frame rate despite the low resolution compared to high-end devices. To mention some, portable devices for sign language identification (Zhou et al. [2020]) and in-field apple detection to estimate production yield (Mazzia et al. [2020]). Other solutions from NVIDIA are the Jetson TX2i for industrial applications and Xavier, which offers more resources than its Nano counterpart (NVIDIA [2021b]).

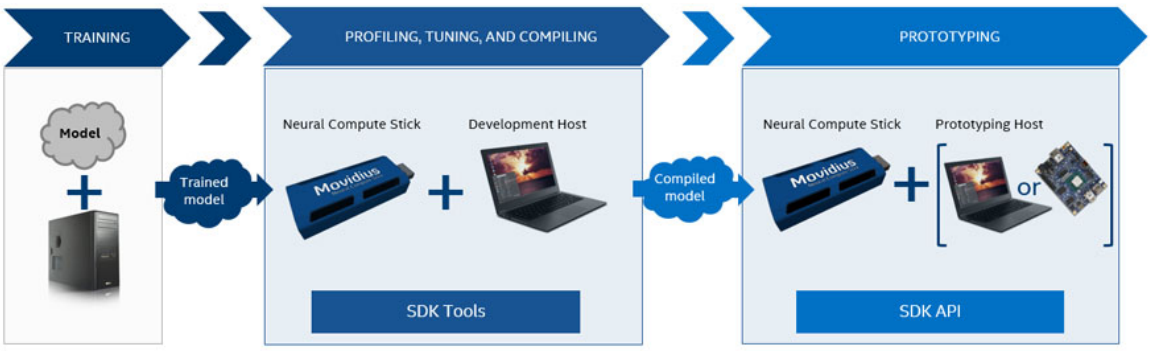


Figure 1.3 – Typical workflow for development with the Intel Neural Computer Stick (NCS) (Intel [2019]).

To enhance embedded boards capacity, Intel Neural Computer Stick (NCS) has recently obtained attention from the deep learning community (Intel [2019]). Inside, we can find the Movidius Myriad X processor, featuring 16 processing

cores SHAVE processors (Rivas-Gomez et al. [2018]) and a dedicated neural computer engine (an on-chip hardware block specifically designed to run deep neural networks at high speed and low power without compromising accuracy). It comes as a USB accelerator and is compatible with Caffe, Tensorflow, mxnet, ONNX, PyTorch, and PaddlePaddle frameworks. We can observe the typical workflow in Figure 1.3. First, the user trains a model using Tensorflow or Caffe. Then, the model is optimized using the OpenVINO toolkit (Intel [2021b]) provided by Intel to generate an Intermediate Representation (IR) of the model. Finally, this model is loaded in the NCS for inference. When used on a Raspberry Pi 4 (with USB 3.0 capability), the Intel NCS obtains 80.4 ms of inference time on the MobileNet v1 SSD 0.75 depth model (Allan [2019b]) (Intel [2021a]). These results are obtained with an image input size of 300 x 300 pixels. Similar results with 96.4 ms of inference time per image are reported (Reuther et al. [2019]) with MobileNet v2 on single-shot detectors (SSD) trained with the COCO dataset (Lin et al. [2014]) with images of size 640×480 pixels.

We can also mention the Coral board USB accelerator from Google. This USB accelerator contains a Tensor Processor Unit (TPU), which can be used as a co-processor to reduce the inference time of deep learning models in hardware-limited devices. When used with a USB 3.0 capable board, the Coral achieves 18.2 ms of inference time on the MobileNet v1 SSD 0.75 depth model (Allan [2019a]).

Finally, the FPGA-based embedded boards recently have gained attention. These devices are oriented towards high parallelization techniques. We can find several image processing application where algorithms such as the Scale Invariant Feature Transform (SIFT) is optimized to run optimally with a low-memory footprint (30 FPS for 320 x 240 pixels input image)(Bonato et al. [2008]). To reduce computing costs, some implementations calculate Gaussian kernels offline for use in Gaussian filters and reduce the number of logic elements used in operation (Li et al. [2018b]). In this sense, CNNs could be benefited from a parallel calculation of the kernels, activations and take advantage of the current optimization techniques. Recently, we can find efforts of deploying CNN in FPGAs with encouraging results (96.5% accuracy on the MNIST dataset) where CNNs outperform conventional ultra-high-speed object detection algorithms. The main limitation of this approach is that the existing CNN models have a large number of parameters and are difficult to deploy in FPGAs with limited on-chip memory resources (Li et al. [2019]).

To conclude, we can observe lower frames per second (fps) when using higher resolutions or when multiple objects have to be classified or detected in the image. To obtain reasonable fps, we usually have to sacrifice generalization. Also, one limiting factor comes with the high number of trainable parameters of CNNs. This phenomenon has been recently reduced by completing the embedded board with a dedicated USB accelerator. However, this is not enough, and some effort is needed to reduce the CNN size to fit in these devices.

Hand-held devices. We call hand-held devices to all modern consumer electronic systems as smartphones, smart cameras, or other. They can be considered as embedded systems but since its utilisation is specific, we consider them as a separated category. Usually, hand-held devices are not optimized for deep lear-

ning applications, and they present poor inference and speed performance, when these devices do not have GPUs, as shown in the literature (Maeda et al. [2018]).

Compared to simple statistical methods previously deployed on smartphones (Do and Gatica-Perez [2014]; McManus et al. [2013]), deep learning models required huge computational resources and thus running them on the small CPU of these devices was nearly infeasible from both the performance and power efficiency perspective. The first attempts to accelerate deep learning models on mobile GPUs and DSPs were made in 2015 by Qualcomm, ARM and other SoC vendors, though at the beginning mainly by adapting deep learning models to the existing hardware. Specialized Artificial Intelligence (AI) silicon started to appear in mobile SoCs with the release of the Snapdragon 820/835 with the Hexagon V6 68x DSP series optimized for AI inference, the Kirin 970 with a dedicated NPU unit designed by Cambricon, the Exynos 8895 with a separate Vision Processing Unit, MediaTek Helio P60 with AI Processing Unit, and the Google Pixel 2 with a standalone Pixel Visual Core. (Ignatov et al. [2019])

For example, Qualcomm is relying on its AI Engine (consisting of the Hexagon DSP, Adreno GPU and Kryo CPU cores) for the acceleration of AI inference (Deng et al. [2020]). In all Qualcomm SoCs supporting Android NNAPI, the Adreno GPU is used for floating-point deep learning models, while the Hexagon DSP is responsible for quantized inference. It should be noted that though the Hexagon 68x/69x chips are still marketed as DSPs, their architecture was optimized for deep learning workloads and they include dedicated AI silicon such as tensor accelerator units, thus not being that different from NPUs and TPUs proposed by other vendors. This processor can usually be found in Samsung, Asus, and Xiaomi's most recent devices. For reference, a Snapdragon 855 processor with a GPU accelerator can run an inference with MobileNet v2 in 24 ms (41 fps) with input resolution of 224 x 224 pixels (Ignatov et al. [2019]).

Most of the SoCs presented during the past 12 months show a performance equivalent to or higher than that of entry-level CUDA-enabled desktop GPUs and high-end CPUs. The 4th generation of mobile AI silicon yields even better results (Ignatov et al. [2019]).

We can observe that the main consequence of the limited resources of these devices is the reduced generalization capabilities and the relatively low accuracy of the implemented CNNs. In the end, it is necessary to have smaller and efficient neural networks that avoid these consequences for hand-held device applications.

High-end servers. The last category is high-end devices usually found in large server rooms or clusters of GPUs. The main application of this hardware is the training of the neural networks that can go long as days or weeks. While having this huge amount of processing power benefits neural networks in general, it comes with an elevated energy cost. In fact, one NVIDIA Titan Xp can consume up to 250 Watts and usually we can find several of them in a single server.

Several high-end clusters are used for training when developing neural network applications. The trend converges towards augmenting the datasets when training the network. It is not rare to see neural network models trained on millions of examples and transformation of the examples. The main consequence of this is that the model size increases to even millions of parameters. While this ap-

proach has proven good results with accuracy up to 99% on classification tasks, the truth is that these models can hardly be deployed on commonly used devices.

Usually, high-end servers have an enormous energy budget. It is not rare to find some clusters training networks for days or even weeks. The users do not deal with time or energy constraints when designing CNNs. In the latest years, green computing has gained attention from the community to solve the energy conservation point of view. Nevertheless, one of the most commonly used techniques is data augmentation that increases training time up to 600% (O'Gara and McGuinness [2019]) as a byproduct of the network size. Hence, network size reduction could also benefit these servers to reduce the necessary energy to train a network.

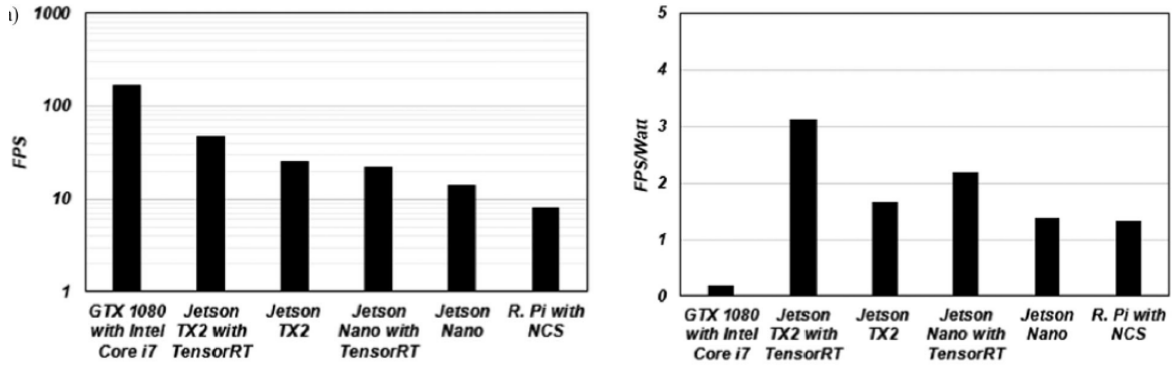
To complete the high-end server usage, we can also mention the inference on these high-end devices. Popular trademarks like Airbnb or Facebook use neural networks to serve their users daily (Haldar et al. [2019]). The prediction capabilities of the neural networks and the classification and detection capabilities of CNNs serve millions of people daily on these and other brands. In this sense, our work is not directly related to these types of hardware stations. Nevertheless, a small size network would benefit these trademarks to deploy several instances of the reduced model to serve more users with less memory utilization.

In conclusion, we can observe that high-end servers are used mostly for training networks and as backend inference services. In this work, we are interested in embedded and hand-held applications. However, it is important to note that high-end servers can also benefit from reducing neural network model size. This reduction can benefit them in reduced energy costs (training in less time) and computational effort (less inference time workload).

To complete the hardware platforms analysis we can cite the work from Arabi et al. [2020]. In their work the authors propose a comparative study between edge devices and compare them to a desktop GPU. Edge devices are commonly known as the embedded devices which are the closest to the sensors on products and transfer to a server the pertinent information of the application. They compare three edge computing platforms for inference at the edge: NVIDIA Jetson TX2, NVIDIA Jetson Nano, and Raspberry Pi 3 with an Intel Neural Computer Stick (NCS) (Movidius Platform).

Comparison of inference speed for six different hardware options is presented in Figure 1.4a. Arabi et al. [2020] also investigates inference efficiency. Inference efficiency is measured by dividing inference speed by power consumption, namely fps/Watt. The Jetson TX2 at maximum performance consumed 15W of power, the Jetson Nano 10 W of power, the R. Pi 3B+ and Intel NCS 6 W, and desktop PC (with a GTX 1080 Ti GPU and Intel Core i7 CPU) is estimated to consume almost 850 W. Figure 1.4b summarizes the inference efficiency of these hardware platforms. While the desktop PC offers better fps it is straightforward to notice that is not as energy efficient as embedded platforms.

In conclusion, we can observe efforts from hardware manufacturers to offer alternatives and hardware compatible with deep learning applications. In the last few years, we have observed deep learning applications moving from high-energy consuming hardware to embedded devices. Despite these efforts, the accuracy and performance (frames per second) are significantly lower than their high-end



(a) Inference speed

(b) Inference efficiency

Figure 1.4 – Performance comparison of hardware for inference on CNNs (Arabi et al. [2020]).

counterpart. In this sense, there also exist software efforts to reduce the size of the neural networks. In the next paragraphs, we outline some of the existing software techniques to achieve lightweight networks that manipulate the inner weights, such as pruning and quantization.

Size Optimization Techniques for Deep Learning

The following section outlines the software efforts to obtain lightweight networks by modifying the internal values of the network after training. While these techniques are usually applied when the neural network is deployed in hardware constrained devices (i.e., limited parallelism, absence of GPU, missing floating-point dedicated hardware), the trend converges in using them as part of the neural network workflow on every platform.

Quantization and Pruning. In the search for lower computational cost, we can find several neural compression methods have appeared in the literature. Among them, the most commonly used techniques are quantization (Chen et al. [2021]) and pruning (Liang et al. [2021]).

Often, deep learning networks contain redundant parameters that have no impact on the network but still count in the computational cost (Denil et al. [2013]). Removing these redundant parameters by pruning results in smaller networks. This means that by removing unnecessary elements in the neural network structure, the number of parameters is reduced with no significant accuracy drop up to some extent.

Another approach to optimize the size of the deep learning models resides in quantization techniques (Young et al. [2020]). In these techniques, the number of bits used to represent parameters is reduced, allowing a reduction in the network size. It is often done to increase the parallelism from the single instruction, multiple data (SIMD) processes or to avoid the usage of floating point computing. The number of bits reduction is usually done after training when preparing the network to be deployed in the inference platform. While the last advances on quantization techniques (Chen et al. [2021]) result in a good tradeoff between network compression and accuracy, they still have limitations, such as the

introduction of significant losses when used in classification CNNs (Guo [2018]).

In this work, we mainly focus on the design and training of the neural network before any optimization technique (quantization, pruning) is applied. In this sense, we study the reduction of convolutional neural networks taking advantage of the internal properties of the operations and the usage of these properties to foster information from the network. A reduction of the neural network before the optimization techniques would result in a smaller and faster network to be deployed in the target hardware for inference.

Deep Learning Applications

To complete this section, we present a range of applications ordered by the hardware platform. The main motivation is to highlight the impact of hardware constraints on the execution of convolutional neural network models with respect to their size.

Embedded devices applications. We classify the embedded devices applications as the ones using Single Board Computers (SBC) that might have an NVIDIA GPU (Jetson TX1, Jetson TX2, Xavier, Raspberry Pi) or not (Sabre board IMX based platform(NXP [2021])). Also, there exist specialized hardware-oriented to image processing such as the Visual Processing Units (VPU) (Movidius USB stick and PCI boards). Usually, these kinds of devices are conceived to find a tradeoff between computational resources and energy consumption. These dedicated hardware platforms allow the end-to-end applications to exist in the field. While Raspberry Pi has a GPU, regarding the CNN applications, the instruction set is not compatible with the most commonly used libraries, so it is limited to CPU usage in most of the applications; hence we consider it in this study as a CPU solution.

Main applications of embedded devices include the ones that need local-processing of the image, low weight and low power consumption. The embedded devices applications cover numerous commercial and industrial domains such as security cameras, agriculture, medical wearable devices, telemedicine, automated guided vehicles (AGV), driving assistants, and drone obstacle detection.

Sa et al. [2017] presents an application for classifying weeds from valuable plants to be used in farming. They use a drone that scans over different areas containing weeds, crops, and a mixture of both. Then they segment the images for each one of the three classes. The drone collects images at 1 Hz and transmits them to a ground station. Compared to a Titan X GPU (one of the top GPUs present on the cluster server mainframes), the NVIDIA Jetson TX2 is 3.6 times slower, but it consumes 17.8 times less power lower. Their network is based on a modified version of the VGG16 network with 14.7 M parameters. This network size allows the model to run in the embedded device. Nevertheless, they achieve 80% accuracy (F1 score) and focus mostly on lowering the energy consumption of the drone.

Ardiyanto et al. [2017] present a medical application as medical assistance for doctors. In their work, the proposed system assesses the severity of the *diabetic retinopathy* from the retina images. The model size of their network is 7.8MB compared to 42 MB of the ResNet20. For each image, their network takes

3.8ms to infer a result on GTX 1080 for images of 320 x 240 pixels. On the NVIDIA Jetson TK1 platform, their network takes 41ms for inference, losing about 5% of accuracy due to the hardware limitations. They could not deploy and test the ResNet model to the embedded platform because of the size limitations. In this sense, this work demonstrates the challenges of deploying bigger models to platforms with limited memory. Also, this work demonstrates the possibility of creating custom networks that can efficiently use the resources and obtain lightweight models capable of running in embedded systems.

Manderson et al. [2018] present a robot controller capable of route optimization for coral reef classification. The robot navigates near coral reefs avoiding collisions, and estimates an efficient route through coral-free areas. The decisions of the network are used to change the orientations of the underwater robot. The network architecture is a modification of the ResNet18 with fewer layers and different activation functions with 10 M parameters. These changes are proposed to increment the computational efficiency of the network. On NVIDIA Jetson TX2, they process images at 10 fps with 41% accuracy. In this sense, we can observe that the network selection is limited by the hardware constraints as computational efficiency and memory size.

Bura et al. [2018] presents a system to help drivers in finding a parking location and bill them automatically based on the exact usage of parking space. This system uses a Raspberry Pi and an NVIDIA Jetson TX2. They use ground and top-view cameras connected to both boards. The Raspberry Pi is used to recognize the license plate, and the NVIDIA Jetson TX2 tracks the vehicle till it is parked in the parking lot. To classify if the parking lot is empty or occupied, they chose to use a custom network instead of the AlexNet network (61 M parameters.) They introduce a lightweight neural network with one convolution layer and three dense layers (approximately 150K parameters). Also, they use Tiny-YOLO (15 M parameters) for the tracking and a lightweight. Their custom network takes 7.11 ms to classify each slot with 99.51% accuracy. This measure means that for a parking lot with 50 slots, it takes up to 355.5 ms to classify the entire lot. With these results and approach, we can observe that the trend converges towards custom networks o low-size networks to be deployed on embedded devices. The network selection is based mainly on the number of parameters, and when they are higher than expected, then a custom network is designed.

Madaan et al. [2017] present a technique to detect wires oriented to unmanned aerial vehicles (UAV) based on a monocular camera. For this problem, the hardware target system is an NVIDIA Jetson TX2. Their work presents a survey with 30 different neural network architectures changing the number of channels and dilation factors. These architectures size goes between 325K and 1.1 M parameters. The network with a higher number of parameters achieves the best results (0.75 AP score). Nevertheless, the biggest network was also the slowest one with two frames per second (fps). Consequently, as their motivation is the faster approach, they decided to use a model with 84K parameters that obtained 4 fps on the Jetson TX2. In this sense, we can observe that network size influences which network to use when searching for the best performance.

In conclusion, we can observe that the main limitation of the presented models is the size of the network. When the network is transported from develop-

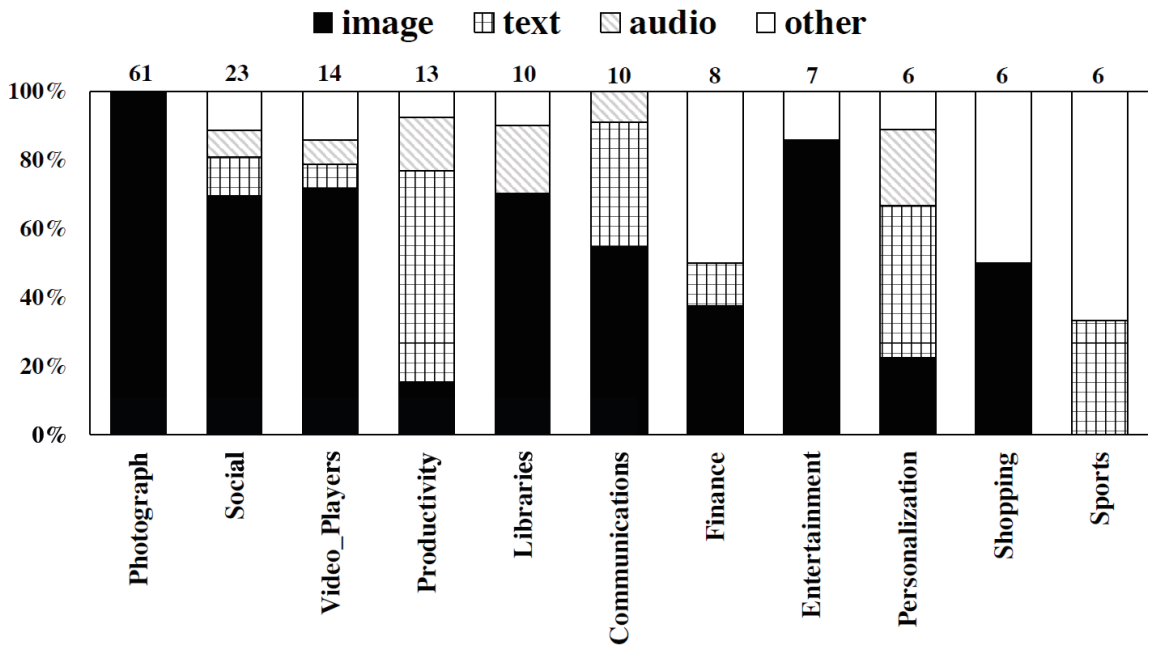


Figure 1.5 – Distribution of Deep Learning smartphones based applications over categories defined by Google Play. Numbers on top: Number of applications in the corresponding categories. (Xu et al. [2019]).

ment to the embedded board, it needs to be reduced using optimization techniques. In this sense, the most significant impact on the accuracy is the reduced memory available to deploy the model. Also, in some applications, the size of the network is correlated to the fps performance. Therefore, reducing the size of the network before the usage of optimization techniques would benefit these applications. Also, if the network is small enough, it could be deployed without going through size optimization techniques on the field.

Hand-held devices applications. As introduced previously, the latest hardware inside portable phones consists of a System on Chip (SoC) processor unit that includes both CPU and GPU in one silicon dice (Qualcomm [2020]). Handheld devices based deep learning techniques have already been applied with success in many fields and with different information sources. In the latest years, the number of applications has increased exponentially compared to the ones available several years ago (Lane and Georgiev [2015]).

One of the main limitations to develop applications for more fields on this type of hardware is the existing tradeoff between hardware resources and network size. Also, the CNN-based applications share resources with other applications as the device's purpose is not limited to the CNN task. Simultaneously, the sensors on the portable terminal represent an advantage to acquire several types of signals and process them. The main drawback is that these applications have to operate in unconstrained environments, noisy data sources, and often with limited battery. In the scope of this manuscript, this becomes an area of opportunity that deep learning applications can benefit from. The main challenge is to overcome the memory restrictions and battery consumption.

Two main approaches exist to deploy deep learning applications on the mobile: (1) the server approach and (2) the standalone approach. The first one uses

the hand-held device as a sensor terminal where the data is sent to a cloud-based server containing the deep learning model, which after processing the data, sends back the results to the device. The second one uses the deep learning model directly on the mobile device. It can run without any connectivity and may produce faster results but has to cope with the limited resources available on the device. We mainly focus on the second one (2) as we study that efficient usage of the filters and network optimization would result in smaller networks able to run on smartphones.

In a survey done by Xu et al. [2019], we can find that image processing is the most popular usage of deep learning on smartphones (Fig. 1.5). Specifically, the applications-oriented towards beauty and face detection usually found in photo editing and camera applications to improve images are the most widely used. In addition, applications like Adobe Scan use text recognition models to scan documents from the phone. Usually, these proprietary applications do not disclose the model accuracy or performance, but we can assume they use state-of-the-art CNNs inside the network architecture. We can observe that in the hand-held devices category, deep learning-based applications span over several categories: medical, security, communications, and others.

To mention some, in the medical field Velasco et al. [2019] propose a smartphone based skin disease classification. For this, they use Mobilenet (5 M parameters) and transfer learning techniques. As a result, they achieve 94.4% of accuracy and classify seven different skin diseases. They obtain these results on a smartphone with an Android operative system to be used in the field and remote areas. The selection of the Mobilenet network is based on the small size of the network.

Chen et al. [2018] present a class of extremely efficient CNN models, MobileFaceNets. These models use less than 1M parameters, and they are specifically tailored for high-accuracy real-time face verification on mobile and embedded devices. They achieve 92.59% of accuracy in the MegaFace dataset (Kemelmacher-Shlizerman et al. [2016]) with an inference speed of 23 ms on a Qualcomm Snapdragon 820 CPU of a mobile phone. In their paper, we can observe several mobile implementations oriented towards face verification, and most of them with less than 2M of parameters. These results mean that there exists an effort to reduce the size of the network while keeping a good accuracy. Also, we can find a tradeoff between the size of the network and the accuracy of the model.

Even the most popular companies, such as Facebook, find it challenging to deploy deep learning models on smartphones (Wu et al. [2019]). In their efforts, they have studied several applications and target hardware reaching some conclusions. First, while some smartphones have a GPU, it is not much faster than the CPU in the smartphone (Gao et al. [2015]). This is mainly because mobile GPUs were not designed to process the same high-resolution graphics rendering that computer oriented GPUs are. Next, they conclude that it is important to maximize accuracy when it comes to smartphones while keeping model sizes reasonable. Facebook focuses on model architecture optimization to identify highly accurate models while minimizing the trainable parameters (Wu et al. [2019]). Finally, we can conclude that model size is one of the most important factors in selecting a network model to find a good balance between accuracy,

performance and network size.

High-end applications on dedicated servers. Lastly, high-end applications running in a cluster of GPUs or dedicated servers are the ones that can use more resources. Usually, this type of hardware is used in the training stages of the networks, but we can also find clusters serving hundreds or thousands of users on the web and real-time applications.

The most used application of high-end GPUs is for training networks. Normally, the bigger networks can train for days, even for weeks (Redmon et al. [2016], Ramesh et al. [2021b]). The resources are usually available for some selected users and are mostly used in research and industrial environments. It has been demonstrated that the size of the networks impacts the training time for the network. This is evident when data augmentation is used as the server needs to process several transformations of the training and images and usually train on thousand of images.

We are primarily motivated in the two first platfom types: embedded devices and hand-held devices. In this work, we pay attention to how providing the generalization features of the network impacts the size of the model. In addition, we are interested in analyzing how the invariance to deformations (mainly rotation invariance) property can be used to reduce the size of the model. The main consequence of this a reduced size of the predictor hence network reduction size. This reduction acquires importance mostly in limited-resource devices, as explained previously.

In conclusion, we observe a joint effort between upgrading hardware capabilities and optimizing computational software requirements. The main consequence of these efforts is the multiple techniques on the software and hardware side.

The current deep learning workflow includes first training the model in an unconstrained environment (computer with a GPU or cluster server). Then, the model is optimized by pruning the weights or reducing the data format to avoid floating-point operations. Finally, the optimized model is deployed to the embedded device. Several times the model has competitive accuracy when trained and tested in the unconstrained environment. Nevertheless, when moved to a constrained environment (embedded board with limited memory and processing power), the accuracy is reduced significantly.

In this work, we focus on the reasoning that a thoughtful neural network design can reduce the size of the network before any optimization strategy and use all the capabilities offered by the hardware fabricants. In the next section, we visit a brief history of neural networks to understand and find possible design opportunities that we can use to reduce deep learning network size.

1.2 Artificial Neural Networks

In the last ten years, artificial neural networks have increased their popularity. One reason for neural network's success is the generalization capacities that allow them to work with previously not seen examples. In traditional programming, a programmer creates a model that computes the input to obtain some

results (Fig. 1.6a). Instead, neural networks are trained with a pair of inputs and desired outputs (where the number of examples is proportional to the complexity of the problem) and generate a set of model parameters (Fig. 1.6b).

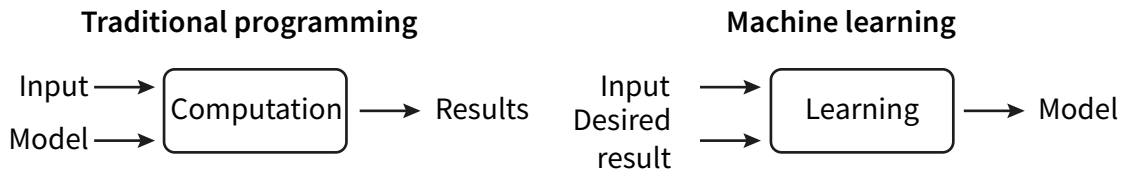


Figure 1.6 – Traditional programming versus machine learning (neural networks) computation.

This process is known as training the network. After it, the neural network becomes a black box capable of generalizing for small input changes while keeping higher accuracy rates on non-previously seen examples of the training set.

The neural network (a.k.a model) is trained iteratively on the training set; each iteration containing all examples is known as an epoch. Currently, networks can be trained from hundreds to thousands of epochs determined by the complexity of the data. This training of a neural network is possible because of two main elements: artificial neurons and the back-propagation learning algorithm.

A neuron is the basic element of a neural network, is composed of an input, an output, weight, and bias (Fig. 1.7). The weight (ω_m) is a mathematical value that controls the signal strength of the connection between the inputs. In other words, the weight value controls the influence that the inputs (x_m) will have on the output (y). The bias value (b) allows the neuron to represent more complex data as it shifts the transfer function along the input axis. Similar to the linear function $y = a + bx$ where the term a allows the line to shift over the axis, the bias value of the neuron allows to shift the output value outside of the origin. Lastly, the activation function ($\varphi(\cdot)$) allows the neuron to have a non-linear transformation of the inputs (when the activation is not linear i.e., sigmoid, ReLU function).

The weights and bias are trainable values that are updated in the back-propagation process at the training step. The way that the neurons are connected determines the type of neural network layer. A fully connected layer (a.k.a dense layer) connects each of the inputs to each one of the neurons present on the layer and each neuron's output is connected to the output of the layer.

Back-propagation refers to the propagation of the error from the output to the input of the model and was first presented by Rumelhart et al. [1986]. This algorithm is widely used for training neural networks and calculates the gradient of the error function concerning the neuron. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule (Goodfellow et al. [2016]).

The concept of neurons and neural networks is not recent. McCulloch and Pitts [1943] suggested the first neuron model able to compute logical operations. It was a neuron with two binary inputs and one binary output. If the threshold

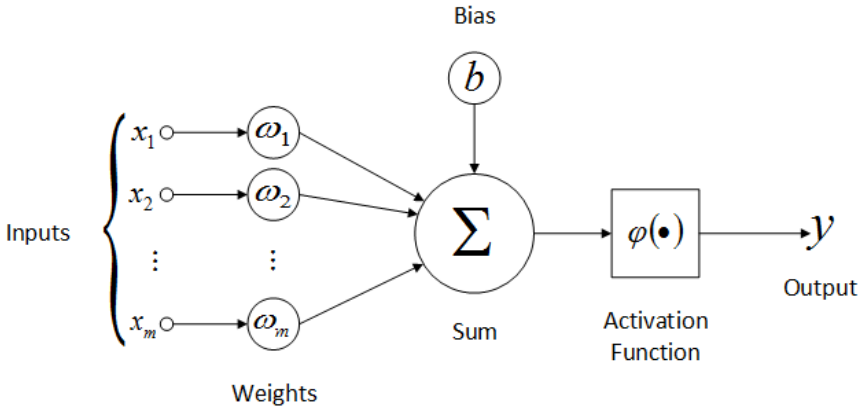


Figure 1.7 – Mathematical model of an artificial neuron (McCulloch and Pitts [1943]]).

was met the neuron was activated. Then, Rosenblatt [1958] developed the perceptron. Instead of two binary outputs, the perceptron has an arbitrary number of weighted inputs. The value of the inputs (weights) can be different for each input. The perceptron can linearly separate an input space. Training the perceptron means adjusting the weights and biases such that the input space is correctly divided hence the classes separated. The learning rule was simple enough, if the output is correct then the weights are not changed, if the output is wrong the input vector is added or substracted from the weights (Fig. 1.8).

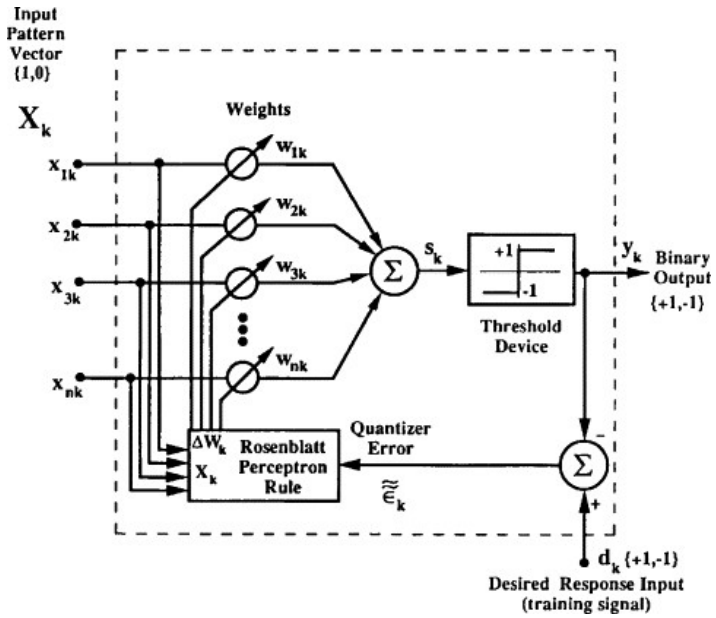


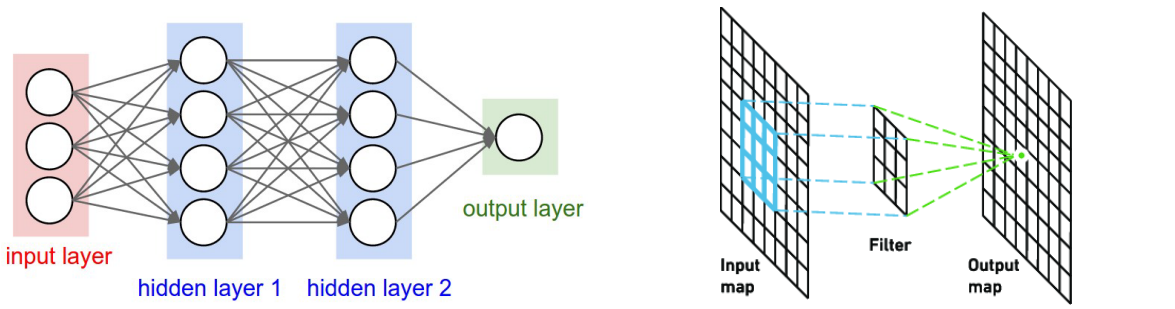
Figure 1.8 – Perceptron and the adaptive threshold element that allowed adapting weights (Rosenblatt [1958]).

To obtain non-linear responses from the network (to produce nonlinear decision boundaries) the next step was to add a layer of neurons. This multilayer perceptron allows the network to have non-linear boundaries. In this sense, instead of the input being connected directly to the output via one or several neurons now is connected to an intermediate layer of neurons. As this layer does not constitute the input or the output it is hidden from the outside, hence the hid-

den layer. Hornik et al. [1989] establishes that standard multilayer feedforward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function. In this sense, a multilayer perceptron is a universal approximator among continuous functions. Unfortunately, while this established the bases of neural networks it does not clarifies the number of neurons in the hidden layer that are needed to obtain this universal approximator. As the number of neurons increases, the training time could become large.

1.2.1 Convolutional Neural Networks

For image data, the first approaches used a set of fully connected layers (one or more hidden layers Fig. 1.9a). The main disadvantage of these approaches was the number of parameters requiring a neuron associated with each input image pixel. The number of parameters grew up quickly following the size of the images. Furthermore, its main disadvantage was the difficulty in predicting features that vary in position over the image. If the object appeared in different positions of the image, a different set of weights was activated. In the following paragraphs, we will study the historical development of the convolutional neural networks until the present.



(a) Dense network (Byun [2020])

(b) Convolutional Layer (Yakura et al. [2018])

Figure 1.9 – **Layer comparison.** (a) Dense Network with two dense layers. (b) Convolutional layer with one filter.

The first work on modern CNNs was proposed by LeCun et al. [1998] in which they demonstrate that a CNN model that aggregates simpler features into progressively more complicated features can be succesfully used for handwritten character recognition (Fig. 1.10). The proposed network LeNet was smaller compared to multilayer perceptrons and presented several advantages over them. Instead of having fixed neurons attention to pixels, a convolutional window slides over the image (Fig. 1.9b). In this sense, it provides a translation invariance to spatial translation over the image while dense networks not. Also, as the weights on the sliding window are reused when scanning the image, the size of the network becomes significantly smaller.

In addition to their accurate models, CNNs tend to be computationally efficient, both because they require fewer parameters than fully-connected architectures and because convolutions are easy to parallelize across GPU cores.

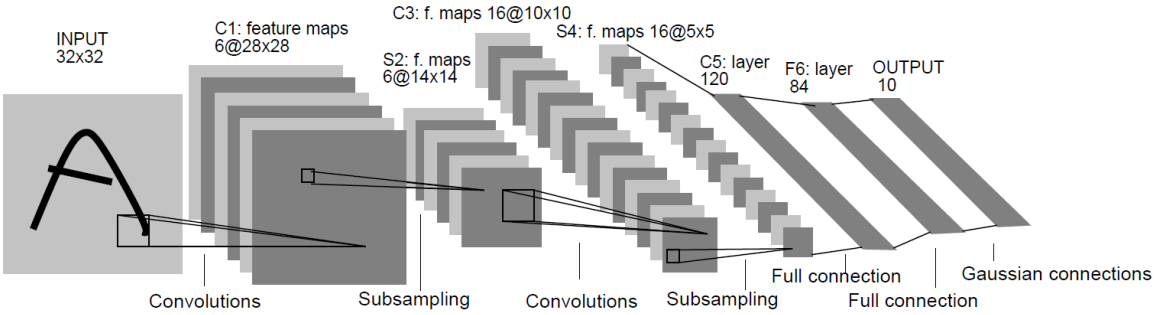


Figure 1.10 – Architecture of LeNet-5: Convolutional Neural Network for digits recognition. (LeCun et al. [1998]).

Consequently, the applications include several fields including one-dimensional sequences as audio, text, and time series analysis.

Usually, CNNs are shaped by one or more convolution layers. While a dense layer connects every input and output to the neurons, a convolutional layer consists of a small fixed-size window (the trend is 3 or 5 pixels) that scans the input image and convolves the input with the convolution kernel. The sliding window acts as a filter in which each pixel corresponds to a learnable weight of the network. Naturally, the convolution layer becomes spatial invariant as the same window scans the image along the two image axis. The main consequence of this is that the network can find the required features it needs despite the position of the feature in the input image. CNNs systematize this idea of spatial invariance and exploits it to learn useful representations with fewer parameters.

The next surge in popularity for CNNs was in 2012 when a CNN called Alexnet outperformed state of the art results in the ImageNet challenge (Fig. 1.11). Krizhevsky et al. [2012] work showed that the new hardware technologies (GPUs) and the availability of large sets of data could enable the researchers to create complex CNNs to solve computer visions taks that were previously hard to tackle. The main contributions of their work were: the use of rectified linear units (ReLu) as neuron activation functions, the usage of the dropout technique to avoid overfitting the model, the inclusion of maxpooling operation instead of the commonly used average pooling and the use of GPUs (NVIDIA GTX 580) to reduce the training time.

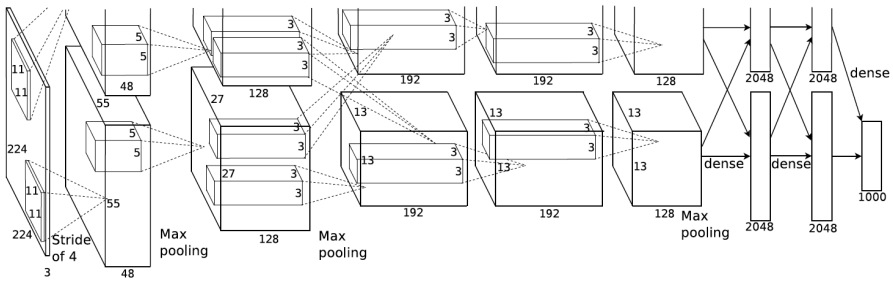
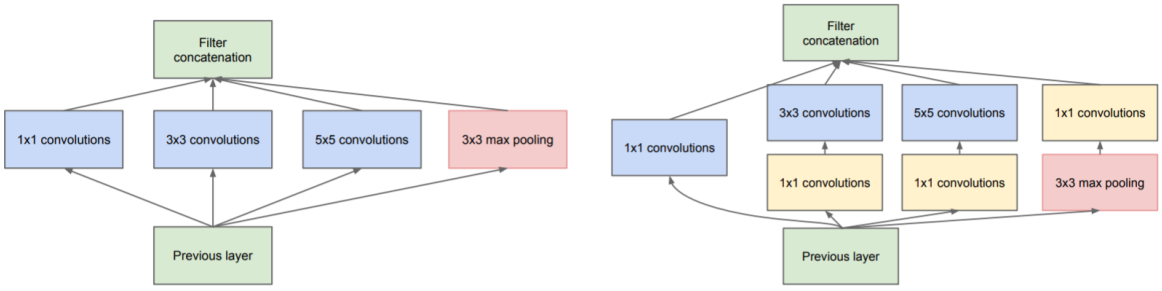


Figure 1.11 – Alexnet architecture (image from the original paper) (Krizhevsky et al. [2012]).

Another improvement are VGG networks from Oxford Simonyan and Zisser-

man [2015]. Instead of using large filters (Alexnet used 9 x 9 and 11 x 11 filters) they used much smaller filters of size 3 x 3. The main advantage and contribution of VGG networks was the notion of combining several 3 x 3 filters in sequence to emulate the effect of larger receptive fields. These notions have been used in more recent architectures. The number of parameters massively increased up to 144 Millions of trainable parameters. The training of this network was difficult, and it is described in the paper how it had to be split into smaller networks with layers added one by one.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 1.12 – Inception module used in GoogLeNet. (a) First approach without dimension reduction (b) 1 x 1 convolutions are used to reduce the number of operations.(Szegedy et al. [2015]).

Bigger companies as Google started to be interested in efficient and large architectures powered by their servers. Their first effort was to reduce the computational requirements of deep neural networks. The main product of their work was the inception module on the GoogLeNet network Szegedy et al. [2015].

The inception module is the combination of convolutional filters of different sizes. The main contribution was the usage of 1 x 1 convolutional blocks to reduce the number of features before the parallel block (Fig. 1.12). This bottleneck allowed to reduce the number of features, and operations, so the inference time was lower. The descriptive potential of the layer was not lost although doing less operations. The next step for them was to regularize the model. For this the introduced a Batch Normalized Inception (Ioffe and Szegedy [2015]), in which the Batch-normalization operation computes the standard deviation of the feature maps and normalizes the values at the output. The training of the next layers was easier as the layer does not need to learn any offset of the input data.

Deep learning models started to appear in the literature, albeit with several constraints due to different challenges such as vanishing gradient. After several layers, the gradient would become zero ; hence the network would not converge as a result. To solve this He et al. [2016] propose Deep residual networks to ease the training of deeper networks. The idea was simple, bypass the input to the subsequent layers. Instead of bypassing a single layer like in previous works (Sermanet and LeCun [2011]), they bypass after two convolutional layers. Also, they used the same 1 x 1 bottleneck similar to the used in the inception networks to reduce the number of features at each layer (Fig. 1.13).

After the advent of the residual networks, the effort for efficient and smaller networks started to appear. Howard et al. [2017] presented a novel approach for mobile and embedded devices. MobileNets were based on a streamlined ar-

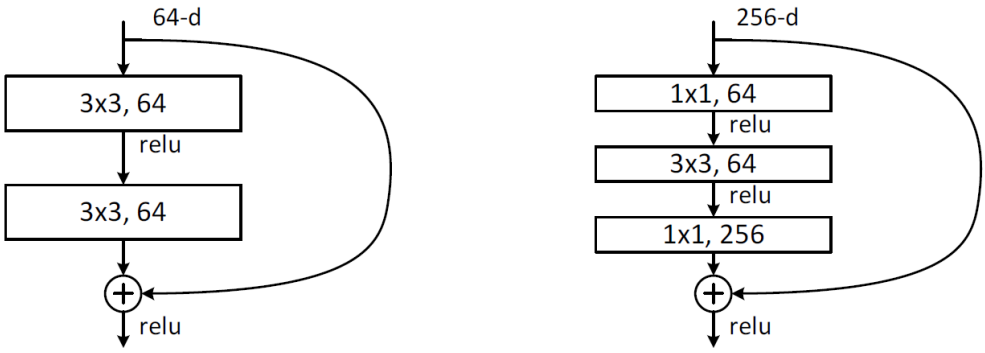


Figure 1.13 – A deeper residual function for ImageNet. Left: a building block from ResNet-34. Right: a bottleneck building block used in ResNet-50/101/152 (He et al. [2016]).

chitecture that uses depthwise separable convolutions to build lightweight deep neural networks. Their main novelty was the inclusion of two simple global hyperparameters that efficiently tradeoff between latency and accuracy. These hyperparameters allow the model builder to choose the suitable sized model for their application based on the problem constraints. In this sense, the network was lightweight (5M parameters with 1.0 depth hyperparameter) compared to its competitors, such as ResNet50 (26 M parameters).

Recently, EfficientNet (Tan and Le [2019]) has become the center of attention. The authors propose a scaling method that uniformly scales the dimensions of the neural networks using a single compound coefficient. CNNs are usually designed with certain dimensions: depth (number of layers), width (number of filters), and resolution (input image size). EfficientNet presents a compound coefficient that uniformly scales them (Fig. 1.14). This technique allows the authors to produce a model that provides higher accuracy than the existing convolutional networks (up to the date of this writing) and a considerable reduction in model size.

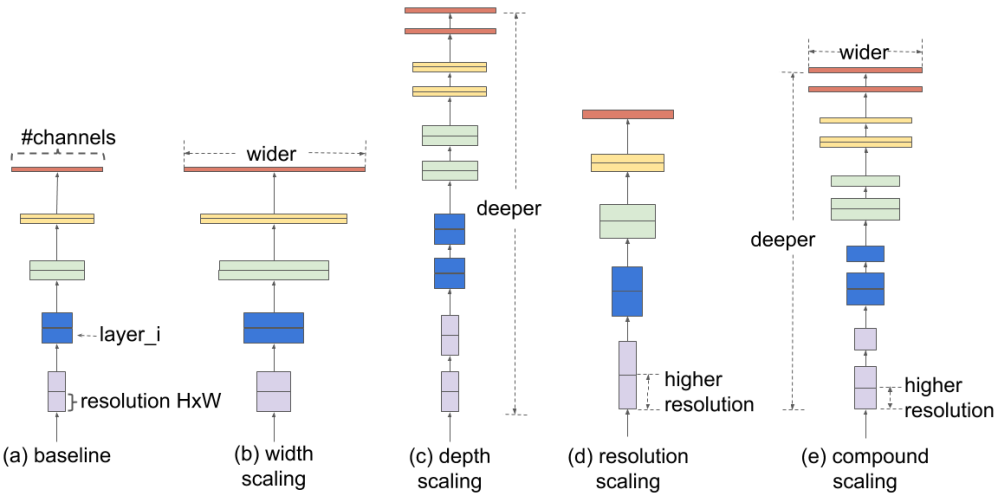
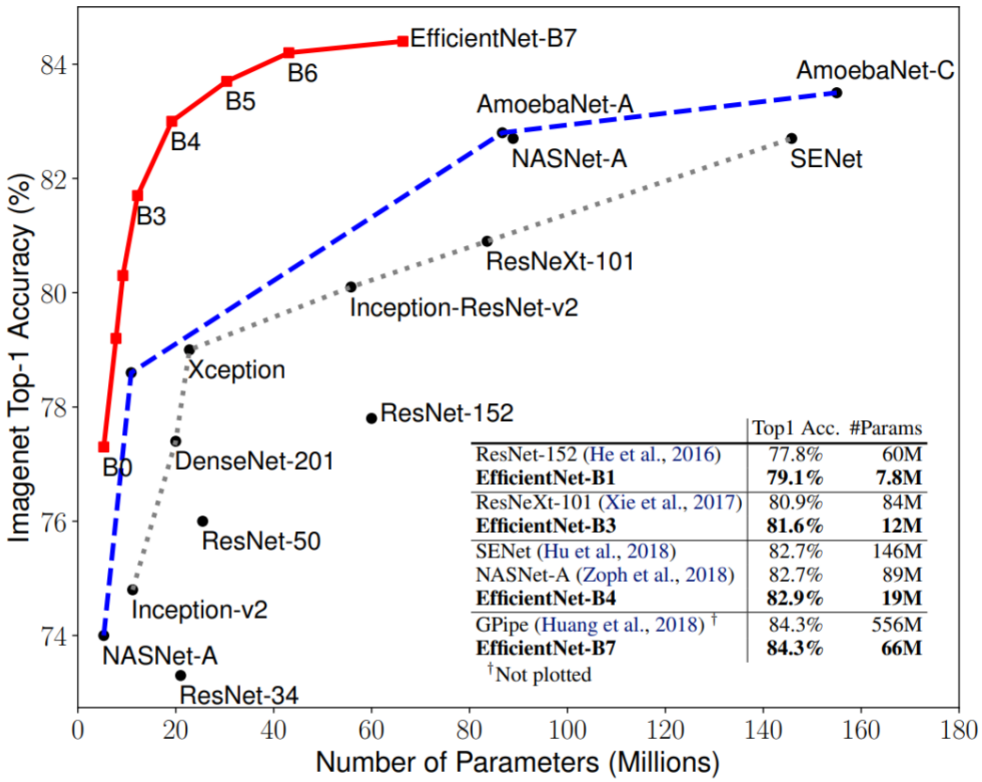


Figure 1.14 – Comparison of different scaling methods. Unlike conventional scaling methods (b - d) that arbitrary scale a single dimension of the network, EfficientNet compound scaling method (e) uniformly scales up all dimensions in a principled way (Tan and Le [2019]).

Figure 1.15 summarizes some of the existing approaches performance on the ImageNet (Russakovsky et al. [2015]) classification challenge. We can observe the correlation between network size and accuracy with models up to 150 M parameters. In conclusion, we can observe that the global tendency is to increase the network model size as long as resources (memory size, computing size) are available. Usually, this is done by scaling the network size (depth or width) and, in some cases, the size of the input images. Nevertheless, while the historical trend converges toward bigger and deeper networks, it disqualifies embedded devices as target hardware.

Thus, in this work, we study the rotation invariance property of the networks to reduce their size and computational requirements. We study this property, avoiding data augmentation techniques that usually make the network predictor bigger to become invariant to the rotations.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 1.15 – Model size versus ImageNet accuracy (Image from: Tan and Le [2019]).

1.2.2 Rotation invariance

This section presents the rotation invariance concept, the motivation behind studying it, and how we will use it to reduce the size of the neural networks.

Classically, the CNN deal with various image transformations by using data augmentation (Dyk and Meng [2001]). This means training the network with a dataset enriched by rotated, mirrored, sheared and scaled original images. In this way, each class becomes a super-class containing all possible deformations of itself. The main consequence is that the classifier itself is forced to become invariant to all these deformations. One consequence is that the model's size

increases, and the second one is a higher probability of overfitting. Finally, such large models penalize the computational performances of the application. The model's size may even become an eliminating factor in mobile and embedded applications.

On image classification tasks, neural networks have outperformed state-of-the-art techniques achieving high values as 99% of accuracy. The main challenge to obtain these results is the uncontrolled conditions of the image acquisition. Often, these images are obtained in different light conditions, sizes, and rotations. Therefore, one of the main objectives of this work is to endow the neural networks with the robustness of classification with respect to the rotation of the input image. Also, we can add applications in which the estimation of the object's rotation angle like pose estimation in robotics hands, automatic food cashiers, and pose detection in printed circuit boards (PCB).

One possible axis to study is presented in several works (Freeman and Adelson [1991]; Weiler et al. [2018]; Worrall et al. [2017]) where it has been demonstrated that the filters contained in the first layers of the network capture low-level features (e.g., edges in different orientations) and the complexity of the filters increases with the depth of the network (e.g., a car wheel, dog-ear) (Figure 1.16). These first-layer filters are usually rotated copies of themselves. It means that a set of oriented features could be used as the first layer avoiding redundant orientations. These works are further discussed in chapter 2.

Recently, several authors proposed obtaining rotation invariance by mapping the input pattern into some particular space (e.g., oriented feature space) (Wiersma et al. [2020]). Nonetheless, all these approaches still require including rotated samples in the training dataset to achieve a reasonable error rate. The second problem – the angular prediction – remains unsolved unless an angle-labeled dataset is available. However, most of the classification datasets do not include the angle information.

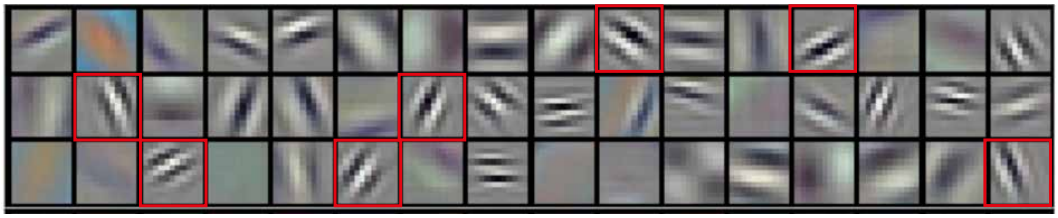
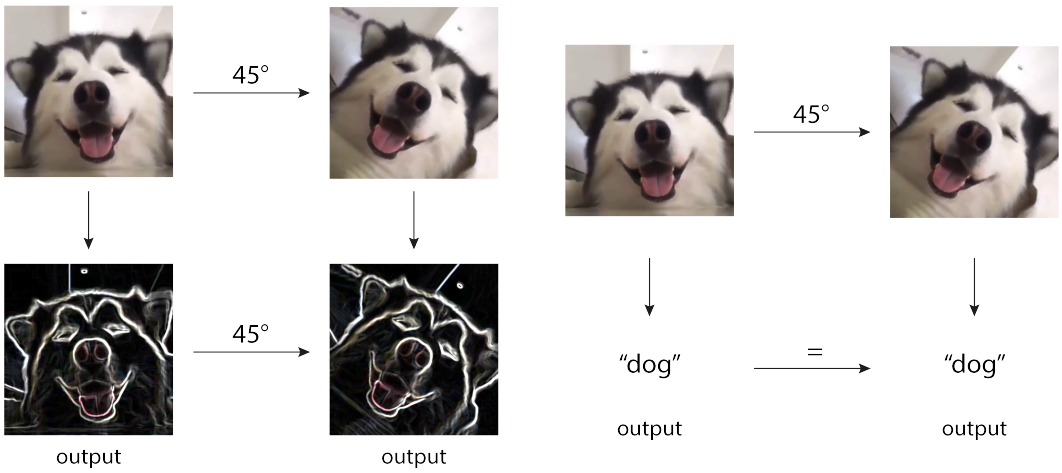


Figure 1.16 – The first layer of AlexNet architecture (Krizhevsky et al. [2012]). Red boxes show an example of rotated versions of almost the same filters.

In this work, we focus on the deformations concerning the in-plane rotation of the objects in the image. There exist several properties of the images that correspond the the relation between the input image and an output processed image. It is important to recall three of them: equivariance, invariance and covariance. Equivariance makes reference to a transformation in which the output results follow the same behavior when the inputs has changed (Figure 1.17(a)). Invariances refers to an output remaining constant under a transformation of the input Figure (Figure 1.17(b)). Last, the covariance refers to the output changing as a direct function of an input change. We can recall the terms equivariance,



(a) Equivariance (b) Invariance

Figure 1.17 – Equivariance and Invariance examples (Kaggle [2014]).

invariance and covariance of a function $f(\cdot)$ with respect to a transformation $g(\cdot)$, as defined in Marcos et al. [2017]:

— Equivariance: $f(g(\cdot)) = g(f(\cdot))$. A transformation in the input results in the same change at the output.

— Invariance: $f(g(\cdot)) = f(\cdot)$. Output remains constant under a transformation of the input.

— Covariance: $f(g(\cdot)) = g'(f(\cdot))$. The output changes as a direct function of a change in the input.

where g' is another transformation, which is itself a function of $g(\cdot)$. With the above definitions, equivariance and invariance are special cases of covariance. Hence, the classification of a randomnly rotated object is invariant to rotation, and prediction of the angle of the rotation is covariant with the rotation.

To conclude, the CNN represent the state of the art for classification tasks. It is well known that one of the main CNN properties is the translation equivariance of the feature representation learned in the first convolutional layers. On the contrary, the rotation invariance of the classification has only recently drawn attention and the literature rapidly grew abundant. Since several years, the trend converges towards encoding the rotation equivariance directly in the CNN architectures. To do that, a large variety of approaches exists. These approaches will be further discussed on Chapter 2.

1.3 Thesis Contributions

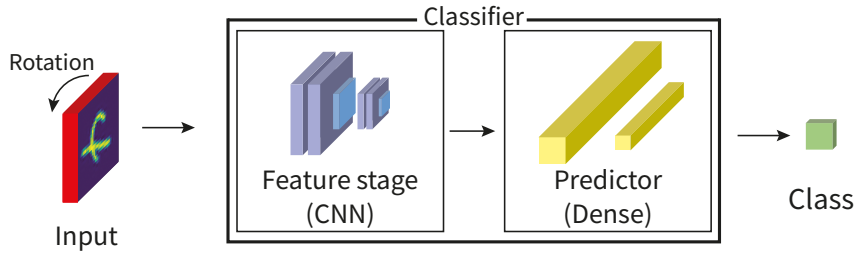
As mentioned previously, deep learning applications have become the standard for visual understanding tasks. In this sense, they can be found in several devices, from high-end servers up to embedded devices. While there are high efforts by hardware manufactures and software developers to reduce the size and computational effort required to run neural networks, there is still a gap between the performance on the different devices. In this sense, reducing these

models would benefit the possible implementations in devices with low-power processors and limited memory. In this work, we approximate this effort by studying the invariance to deformations, specifically the rotation invariance in a novel approach (Fig. 1.18).

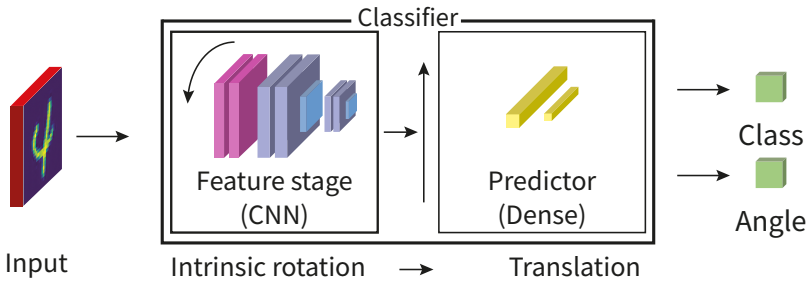
While naturally convolutional layers are spatially invariant, they lack rotation invariance properties. Currently, the trend converges towards endowing them with such properties by presenting rotated samples of the images in the dataset (Fig. 1.18a). The main consequence of feeding the network with these rotated samples is the increase in the predictor size to become invariant to the transformations. For example, in the case of rotations, the predictor learns invariance up to 360° or up to some $d\phi$ and be applied $360°/d\phi$ times.

In this work, we present a novel approach to obtain rotation invart CNNs. Instead of rotating the input image (data augmentation technique), we propose a CNN feature stage with intrinsic rotation properties and a small predictor translating over the rotated copies of the input.

We propose to obtain intrinsic rotation using a set of ordered oriented filters that express the angular relationships between the elements of the object in the input image (Fig. 1.18b in pink). The design is based on the principle of oriented feature space proposed by Chen et al. [2000]. Our novelty is re-orienting the oriented features to the horizontal reference to obtain a roto-translational feature space. The main consequence of this alignment is that the feature space linearly translates following the rotation of the input image.



(a) Classical CNN organization with data augmentation at the input



(b) Proposed construction of rotation invariant CNN with reduced size

Figure 1.18 – Contribution of the thesis.

Then we add a predictor that iterates (translates) over the roto-translation feature space and outputs a prediction for each translation. In this sense, the predictor keeps its reduced size as it does not need to become invariant to multiple

orientations. Furthermore, as a byproduct of the design, we obtain information on the angle of the rotation.

As a consequence of the design, the network becomes rotation invariant. We test rotation invariance by training the network with upright-oriented examples and validating with randomly orientated examples. This network can also be trained with augmented examples (random rotations), improving the results obtained with classic data augmentation.

In this manuscript, we first provide the proof of concept of stated ideas. We use a face recognition problem to illustrate the utilization of the ordered filter set on a challenging problem recognized for the sensitivity for the image deformations. Also, we illustrate results in an embedded board (NVIDIA Jetson TX1) to evaluate the performance and experience of the proposed network in real-life hardware constraints.

Once we prove this hypothesis, we use the intuition behind the roto-translational feature space to construct a custom layer based on oriented features. Also, we design a translating predictor based on 3D convolutions and dense layers. With these layers, we propose several alternatives to obtain rotation-invariant networks.

We apply this knowledge to simple and complex datasets reaching state-of-the-art results and outperforming the classical approaches while keeping the size of the network minimal. Also, we present the class and angular prediction results of the translating predictor. Furthermore, we validate the network trainability mathematically and present a general methodology for other networks.

1.3.1 Thesis Structure

In this section, we introduce the general content of the subsequent chapters. To facilitate this, we outline the relations between different contributions in Fig. 1.19.

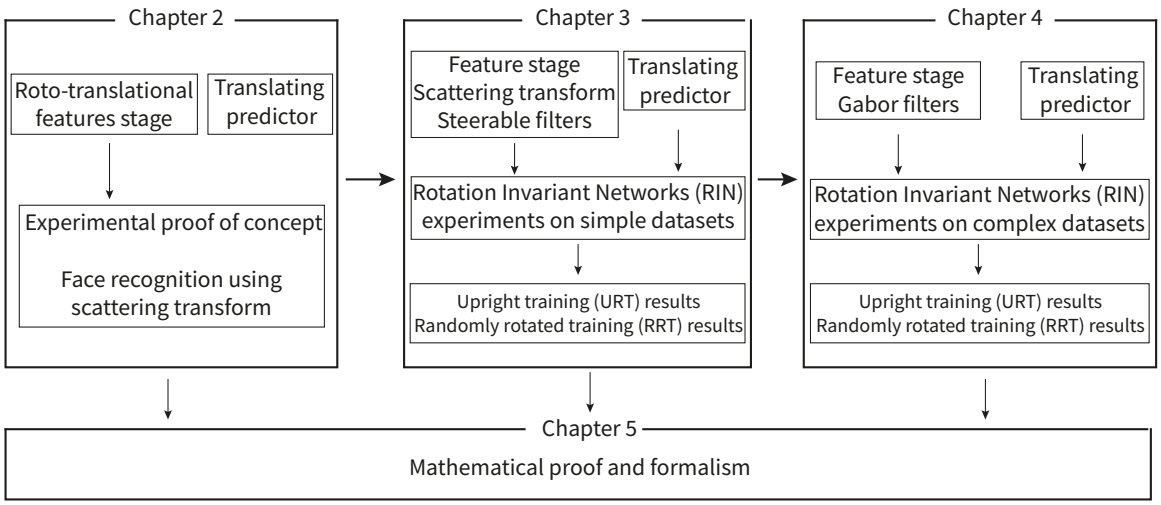


Figure 1.19 – Manuscript structure organization

Chapter 2 introduces the concept of roto-translational feature space, which is one of our main contributions. First, we construct the roto-translational feature

space using two consecutive operations: oriented features space and features re-orientation. The first one decomposes the input image into several oriented components ordered by increasing magnitude. Then, the second one re-orients the features to the horizontal reference. With this, we get a roto-translational feature space that translates over the depth axis following the rotation of the input image. Then, we prove the possibility of training a CNN using such feature space on a known challenging task on face recognition. Finally, we introduce the concept of the translating feature space, which is applied to the roto-translational feature space to obtain rotation invariant prediction. As a byproduct of the design, we also obtain the angle orientation of the input image.

Chapter 3 presents the first experiments using the proposed Rotation Invariant Networks (RIN). First, we discuss the inner components of the feature stage and propose two alternatives to generate the oriented feature space: scattering transform and steerable filters. Next, we present the properties of each alternative, propose an architecture based on each approach and discuss their advantages and disadvantages. Finally, we outline the obtained results on the MNIST when trained on upright oriented (URT) and randomly oriented training (RRT) examples.

Chapter 4 introduces our experiments with complex datasets. This chapter presents a new alternative for the oriented feature space: Gabor filters and the possibility of using state-of-the-art backbone networks as feature extractors. First, we discuss the properties and parameters that become trainable for the network. Then, we test these filters on the MNIST dataset to compare with the previous filter alternatives (scattering, steerable). Next, we introduce ResNet as a feature extractor in the proposed network to solve the challenging classification task on complex datasets. Finally, we present the results of our proposals on the CIFAR-10 dataset with upright training and randomly rotated training.

Chapter 5 outlines the mathematical proof and formalisms. After several experiments on different datasets, we converge to mathematical methodology that describes the behavior of the rotation invariant networks. First, we outline the properties of the feature stage. Also, we describe the properties and the proof of the translating features over the translating feature space. Then, we present the prediction model properties and definitions. Furthermore, we prove the trainability of the translating predictor. Finally, we conclude this chapter by discussing the Vapnik-Chervonenkis dimension of the model to confirm the minimal size of the translating predictor.

Chapter 6 presents conclusions and perspectives on future works. We discuss several perspectives and possible improvements axis for the rotation invariant networks. First, we outline and discuss the main contributions of our work. There we recall the properties and design of the roto-translational feature space and translating predictor. Then, we discuss the possible axis of improvement for future works.

Chapter 2

Roto-translational Feature Space

By construction, the convolution layers are translation invariant (Kayhan and Gemert [2020]). However, the convolutions are not naturally invariant to other transformations such as rotations. As mentioned previously, in this work, we focus on in-plane rotational deformations, which form the basis of the contribution of this thesis.

As outlined in section 1.3, we propose a CNN feature stage with intrinsic rotation properties and a predictor translating over the rotated copies of the input. In this sense, the predictor becomes smaller than its invariant counterpart because it does not need to become rotationally invariant.

To achieve this, we need to decompose the input image into several oriented components to obtain the image edges' angular information. The historically first to propose an oriented space is Chen et al. [2000]. The authors propose generating a multiple orientation representation by adding an extra axis to the image. This method allows the effective segmentation of lines at intersections as local regions where lines occur at multiple orientations (Fig. 2.1).

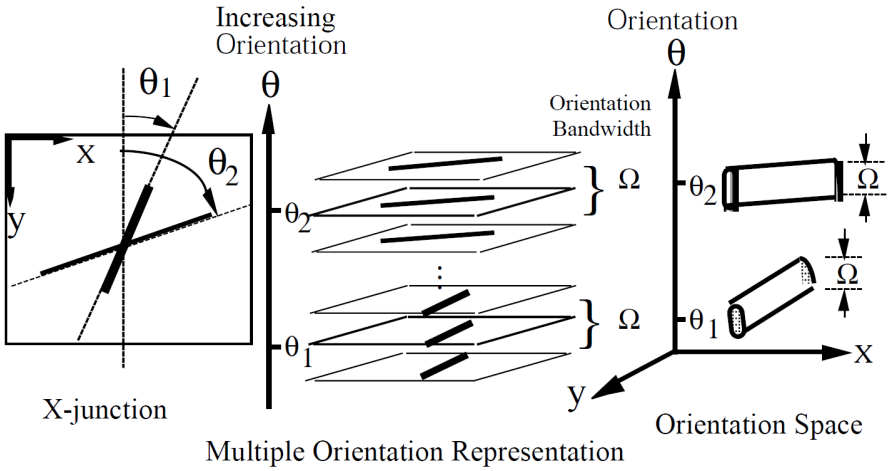


Figure 2.1 – Outline of the concept of orientation space filtering.

Compared to Chen et al. [2000] work, our novelty is to compensate the multiple orientation representation to obtain components aligned to the horizontal reference. These aligned components have a translation behavior over the additional feature axis that follows the rotation of the input. This concept is deeply

discussed and demonstrated in section 2.2. We follow this discussion by presenting a classification problem using oriented feature space components based on the scattering transform network.

In this experiment, we use a classical computer vision problem: face recognition. We use this problem to confirm that it is possible to obtain classification convergence on a convolutional neural network while using oriented components as filters on the first layer. We base our research on previous approaches and extend the obtained results to an embedded platform to demonstrate the possibility of deployment on constrained devices.

In section 2.3 we introduce the concept of translating predictor. The translating predictor scans each translation generated by the roto-translational feature space and outputs a probability for each translation. In this sense, we obtain a probabilistic distribution where the maximum value corresponds to the class and the angle of the input image.

To complete, in the following paragraphs, we present an in-depth study of the state-of-the-art techniques used on rotation invariant classification tasks. We discuss their advantages, disadvantages, and contributions.

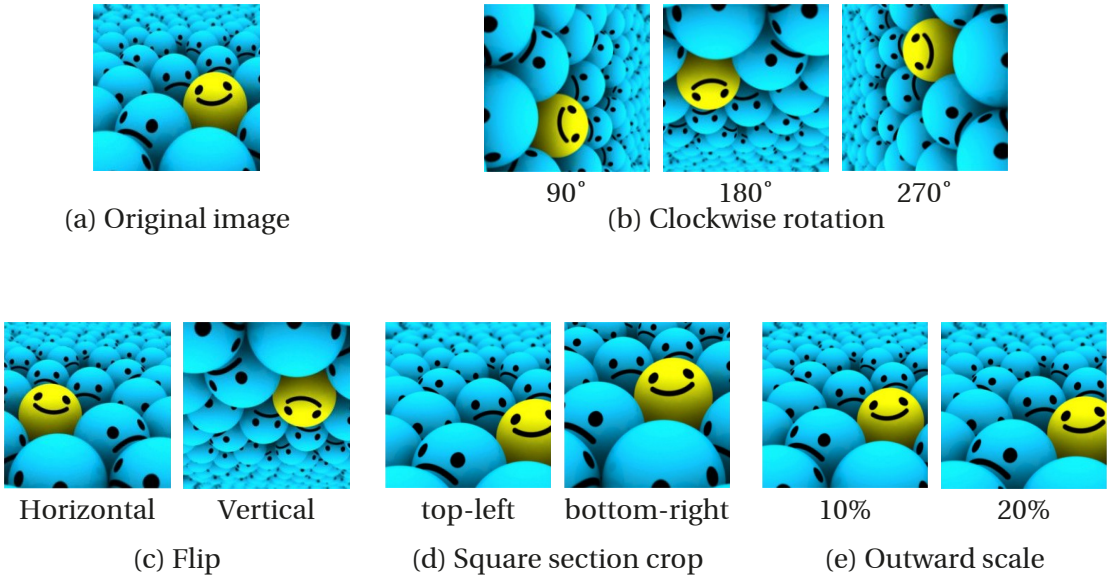
2.1 Existing Approaches to Obtain Rotation Invariant Classification

In general, the literature uses randomly oriented samples containing data set as MNIST-rot (Worrall [2017]) to evaluate the different CNN architectures. However, in order to provide an unbiased evaluation we argue that the rotation-invariant properties can be only illustrated by training a network with upright oriented examples and then testing on a randomly oriented ones. Only this demonstrates that the network effectively learns to classify rotated samples it has never seen during the training. At the end of this section, we present a summary table that condenses the existing methods and the used training approach. In addition, the table (2.1) summarizes the properties of the selected state-of-the-art methods in terms of the number of orientations, the model size, and the capability of predicting the angle of the input rotation.

The literature contains numerous references on endowing the CNNs with rotation invariance or equivariance. Two main strategies exist: i) data augmentation or ii) modification of the network architecture.

The historically first to explore the data augmentation method was the general statistical community. Data augmentation schemes were used to make simulation feasible and straightforward, while auxiliary variables were adopted to improve the speed of iterative simulation Dyk and Meng [2001]. Then, the notion was introduced to the convolutional network applications by Simard et al. [2003]. Their work describes how one of the most critical practices to train a CNN is getting a training set as large as possible. To obtain this, he proposed expanding the training set by adding a new form of distorted data (different transformations of the examples with the same label). Several years later, the technique was used on AlexNet (Krizhevsky et al. [2012]), and from there, it became one of the trends to increase the accuracy of neural network models.

The basic principle is to enrich the training data with samples undergoing transformations. This technique is well known and it results in learning generalized (but larger) models without requiring modification of the network architecture. The major disadvantage of this approach is the need to increase the computing capacity for learning while increasing the risk of overfitting Weiler et al. [2018].



(a) Original image (b) Clockwise rotation (c) Flip (d) Square section crop (e) Outward scale

Figure 2.2 – Data augmentation techniques.

Usually, the CNNs use data augmentation techniques to endow the network with rotational invariance properties (Dyk and Meng [2001]). Data augmentation consists of training the CNN with a dataset enriched by rotated, flipped, cropped and scaled example images (Fig. 2.2). This augmentation can be seen as each class becoming a super-class containing all the possible deformations itself done by the data augmentation process. While most of the data augmentation techniques have demonstrated to increase the accuracy of the models this comes with some drawbacks.

To mention some drawbacks, the CNN training time is increased considerably so it can learn and capture all of these deformations of the input data, and comes with a higher probability of overfitting. This is normally noted as an increased training time of the same magnitude of the number of transformations added to the input data. Some studies conclude by having found that the training time can increasing up to 600% when using data-augmentation against the non-augmented model (O'Gara and McGuinness [2019]).

One of the main drawbacks in the rotation context is the loss of the rotation equivariance data when trained. When the predictor becomes invariant, the angular information is lost. It is the product of the predictor itself learning to classify the orientations of the class as a single super-class. Instead, the rotation information (as separate sub-classes) should be kept until the very last layer allowing the network to foster the rotation equivariance and predict the angle with the same number of resources.

Since several years, in the effort to avoid the previous drawbacks the trend

converges towards encoding the rotation equivariance directly in the CNN architectures. To do that, a large variety of approaches exists. We can divide these approaches following the domain of the features and how they are processed: continuous rotation angle sampling (for 360 rotation equivariance) and discrete angle sampling for rotating filters.

2.1.1 Continuous Angular Sampling

The papers referenced in this section allow moving from discrete orientations to continuous sampling. The principle has been introduced by the harmonic networks (or H-Nets) in Worrall et al. [2017]. H-Nets exhibit equivariance to patch-wise translation and continuous 360-rotation. It achieves this behavior by replacing the regular CNN filters with circular harmonics, returning a maximal response, and orientation for every receptive field patch.

H-nets hard-bake 360°-rotation equivariance into their feature representation, by constraining the convolutional filters of a CNN to be from the family of circular harmonics. If f is the feature mapping of a standar convolutional layer, then 360°-rotational equivariance can be hard-baked in by restricting the filters to be from a circular harmonic family. They assume that the image patch is only able to rotate locally about the origin of the filter. This means that the cross-correlation response is a scalar function of input image patch rotation (Fig. 2.3). The rotation order of feature maps and filters sum upon cross-correlation, so to achieve a given output rotation order, they must obey the equivariance condition.

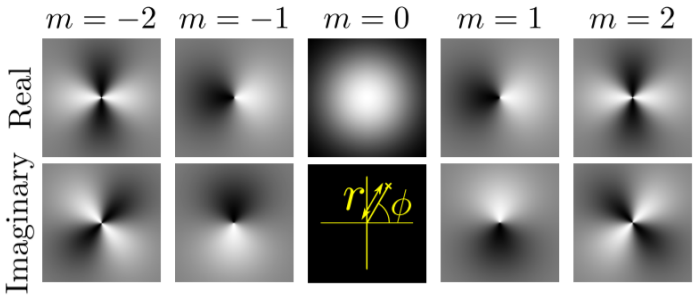


Figure 2.3 – **Harmonic Networks (Worrall et al. [2017]).** Real and imaginary parts of the complex Gaussian Filter. Cross-correlation, of a feature map of rotation order n with one of these filters of rotation order m, results in a feature map of rotation order $m+n$. Note that then negative rotation order filters have flipped imaginary parts compared to the positive orders.

One drawback of this approach is that combining complex features, with phases, which rotate at different frequencies, leads to entanglement of the responses. In fact, at every feature map, the equivariance condition must be verified. It is needed to avoid arriving at the same feature map along two different paths, with different summed rotation orders. They resolve this problem by enforcing the equivariance condition at every feature map. They solve this by creating separate streams of constant rotation order responses running through the network. These streams contain multiple layers of feature maps separated by rotation order, zero cross-correlations and nonlinearities. Moving between streams, they use cross-correlations of rotation order equal to the difference between

those two streams. At the end, when multiple response converge at a feature map, they sum the responses of the same rotation order.

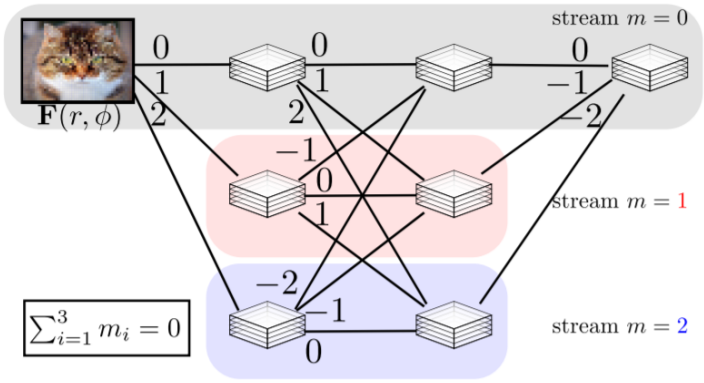


Figure 2.4 – **Harmonic Networks (Worrall et al. [2017]).** An example of a 2 hidden layer H-Net with $m = 0$ output, input-output left-to-right. Each horizontal stream represents a series of feature maps of constant rotation order. The edges represent cross-correlations and are numbered with the rotation order of the corresponding filter. The sum of rotation orders along any path of consecutive edges through the network must equal M = 0, to maintain disentanglement of rotation orders.

Some approaches exist in the literature that base their filters on the H-nets theory. We can cite the Spherical CNN Cohen et al. [2018] and Icosahedral CNN Cohen et al. [2019]. The first one uses a spherical cross-correlation, which is expressive and rotation-equivariant. Also this network allow training with upright position samples only; however, the computational cost remains very high.

Icosahedral CNN Cohen et al. [2019] proposes an improvement to the Spherical CNN. Mainly, the authors pay attention to the reduction of computational cost. They proceed by sub-sampling the sphere in a single 2D convolution call and make it scalable. A similar approach is represented by CubeNet Worrall and Brostow [2018]. It uses a 3D CNN group convolution that permutes the output as a function of the input while keeping time 2× slower than non-group CNNs. However, they need the rotational data augmentation because of the lack of angular selectivity.

2.1.2 Discrete Angular Sampling

The main approach of this group is to generate an oriented filter bank by using some filter rotation technique (e.g., steerable filters Freeman and Adelson [1991]). Usually, the network generates during the learning process a set of mother wavelets and then rotates them to generate oriented responses. Most of the works mentioned in this subsection are based on Freeman's work so it is convenient to show the basic methodology on how they generate the oriented filter bank.

Freeman and Adelson [1991] present a methodology to rotate filters using a set of two base filters. They describe the term "steerable filter" as a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of "basis filters" (Fig. 2.5). They show two and three dimensional cases and

demonstrate the steerable properties and how many basis filters are needed to steer a given filter. The steerable filters are generated as a linear sum of rotated versions of itself.

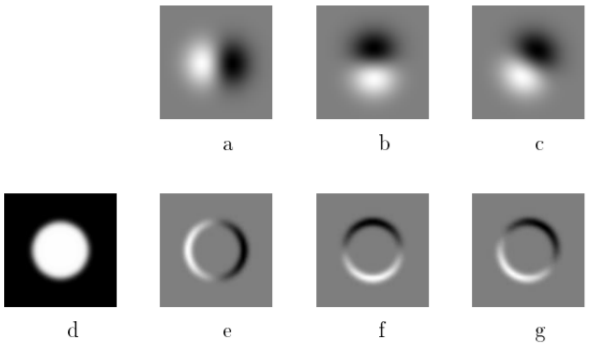


Figure 2.5 – **Steerable filters (Freeman and Adelson [1991]).** Example of steerable filters: (a) $G_1^{0^\circ}$ first derivative with respect to x (horizontal) of a Gaussian; (b) $G_1^{90^\circ}$, which is $G_1^{0^\circ}$ rotated by 90°. From a linear combination of these two filters, one can create G_1^{θ} which is an arbitrary rotation of the first derivative of a Gaussian. The same linear combinations used to synthesize G_1^{θ} from the basis filters will also synthesize the response of an image to G_1^{θ} from the response of the image to the basis filters; (d) image of a circular disk; (e) $G_1^{0^\circ}$ convolved with the disk (d).

The two-dimensional case can be easily demonstrated using Gaussian filters. Consider the two-dimensional, Gaussian function G written in Cartesian coordinates x and y:

$$G(x, y) = e^{-(x^2+y^2)} \tag{2.1}$$

where they set scaling and normalization constants to 1 for convenience.
The first x derivative of a Gaussian is

$$G_1^{0^\circ} = \frac{\partial}{\partial x} e^{-(x^2+y^2)} = -2xe^{-(x^2+y^2)} \tag{2.2}$$

where the notation 0° means a derivative in the horizontal direction (∂x).
The same function, rotated 90° (derivative in vertical ∂y), is:

$$G_1^{90^\circ} = \frac{\partial}{\partial y} e^{-(x^2+y^2)} = -2ye^{-(x^2+y^2)} \tag{2.3}$$

It is straightforward to show that G_1 filter at an arbitrary orientation θ can be synthesized by taking a linear combination of $G_1^{0^\circ}$ and $G_1^{0^\circ}$:

$$G_1^{\theta} = cos(\theta)G_1^{0^\circ} + sin(\theta)G_1^{0^\circ} \tag{2.4}$$

Since $G_1^{0^\circ}$ and $G_1^{90^\circ}$ span the set of G_1^{θ} filters, they become basis filters for G_1^{θ}. The $cos(\theta)$ and $sin(\theta)$ terms are the corresponding interpolation functions for those basis filters. Because convolution is a linear operation an image can be filtered at any arbitrary orientation using the linear combination of the images filtered by the basis filters.

Following this methodology a discrete number of filters can be generated in several orientations and conform the first layer of the neural network. Most of the works in this section find a tradeoff between the number of orientations of these filters and the number of parameters of the networks. This means that increasing

the number of orientations usually results in a higher number of parameters and increased training time.

One of the first proposal has been published by Zhou et al. [2017]. The authors propose the Oriented Response Networks in which they have Active Rotating Filters. These filters rotate during the convolution and produce maps with location and orientation explicitly encoded. The main drawback is high number of parameters to learn. Next, we can cite Rotation Invariant Local Binary Networks Zhang et al. [2018]. The authors introduce the Local Binary Orientation Module. It can be inserted into a traditional CNN. This layer contains Local Binary Convolutional and Active Rotating Filters.

A significant contribution is recent work of Weiler et al. [2018]. Their SFCNN comprises a set of complex filters and rotates them by phase manipulation. However, these filters have learned weights (magnitude of the activation changes but shape remains constant) and present results for a classification task on only the rotated MNIST dataset. While the SFCNN work is very similar to one of the approaches used in this thesis we use a single real mother wavelet rotated using the steerable filters technique Freeman and Adelson [1991]. Also, we let the filters learn weights, shape and size parameters. This allows the filters to change shape and size when trained. Also, we provide angular prediction capability to our network relying on the relative position of the filters in the network. Furthermore, SFCNN uses a generalization of He's weight initialization to improve their results while our network is independent of weight initialization.

Recently, the idea of learning steerable filters inspired the introduction of layers based on the Gabor filter bank Luan et al. [2018]. Also, they present interesting results using ResNet as backbone (instead of convolutional predictor) of their network for Natural Image Classification tasks. Whereas the obtained accuracy is very competitive, the authors present the results only for four different orientations, and the number of parameters increases to almost 2 million parameters.

Another example of discrete filter banks is TI-pooling (Laptev et al. [2016]). The authors use rotated versions of the same image as the input letting the network choose the right orientation thanks to the integration of parallel siamese architectures and new TI-POOLING operator.

The scattering transform-based network architecture presented at the begining of the Chapter 3 is also present in the state of the art. Sifre and Mallat introduced the scattering transform-based networks in Sifre and Mallat [2013]. They use scattering transforms to generate rotation-equivariant feature extractors. This introduces the concept of rotating the inner filters, and it allows obtaining a tradeoff between the classification accuracy and the network size.

Another approach based on wavelet scattering networks has recently appeared (Saydjari and Finkbeiner [2021]). In this work, the authors use scattering networks, which are CNNs with fixed filters and weights. They introduce a fast-to-compute rotationally equivariant wavelet scattering network (EqWS). They obtain competitive results on the MNIST dataset when trained on upright oriented examples (92.12% accuracy) and trained on randomly oriented examples (92.10%). Nevertheless, a considerable drawback of their approach is that the wavelets are not trainable. In this sense, a small change in the data distribution

would significantly impact the network accuracy.

In conclusion, discrete angle sampling approaches represent the state of the art approaches when dealing with rotation invariant networks. One of the main disadvantages of the method is calculating the complex filters increasing the network's training time and size. The general approach is to have learnable weights, but there are only few paper proposing to learn the shape and size parameters. Also, according to our knowledge, there is a real lack of networks allowing to predict the rotation angle.

Several papers even propose a theoretical generalization of the above-mentioned approaches. One can find a good transformation invariance/equivariance theoretical aspects in Weiler et al. [2018] or Pan Zhong [2019]. Many complementary works rely on this theory to propose alternative solutions. We can cite Cireşan et al. [2012] and Multi-Column Deep Neural Networks, they train a model for each one of the possible transformations followed by a global averaging pooling. In Gens and Domingos [2014], the authors propose a generalization of convnets that forms a component of feature transformation maps over arbitrary symmetry groups. In the last, the Spatial Transformer Network (Jaderberg et al. [2015]) applies spatial transformations to the feature maps. To complete, we can also cite Pan Zhong [2019] discussing deeply the theoretical aspects of invariance/equivariance transformations.

To solve the problem of rotation equivariance, we can also introduce the transformation invariance locally, Marcos et al. [2017] introduce a CNN architecture encoding rotation equivariance, called Rotation Equivariant Vector Field Networks (RotEqNet). The network applies one filter at different orientations and extracts a vector field feature map, encoding the maximum activation in terms of magnitude and angle.

Zhou et al. [2017] propose the Oriented Response Networks in which they have Active Rotating Filters. These filters rotate during the convolution and produce maps with location and orientation explicitly encoded. The main limitation of these approaches is a high number of parameters to learn. We can also cite Rotation Invariant Local Binary Networks (Zhang et al. [2018]). The authors present the Local Binary Orientation Module that can be inserted into a traditional CNN. This layer contains Local Binary Convolutional and Active Rotating Filters.

Yet another strategy is represented by deep symmetry networks. In Dieleman et al. [2016], they introduce four operations (roll, stack, pool, slice) that can be inserted into neural network models as layers, making their models partially equivariant to rotations. They also integrate parameter sharing across different orientations, obtaining smaller models.

Recently, Li et al. [2018a] presented the Deep Rotation Equivariant Network. Their network is based on ResNet34 (He et al. [2015]) and specific upsampling and projection layers encoding the rotation and reflection symmetry of dermoscopy images. However, they do not achieve a complete rotation equivariance.

In RotDCF (Gao et al. [2019]), the authors propose a decomposition of the filters in group equivariant CNNs; they show benefits in the reduction of the parameters and computational complexity. They also illustrate how the decomposition of the convolution filter across the 2D space leads to an implicit regularization of the filters, and improves the robustness of the learned representations.

However, they only present results on rotated validation up to 60°while training on upright examples.

Other interesting approaches could be also cited. Follmann and Bottger [2018] present Rotationally-Invariant Convolution Module with rotational convolutions and rotational pooling layers. They achieve rotational invariance by rotating the filters and then back-rotating the generated feature map by the negative angle of the filter. We compare our results directly with them on the CIFAR-10 dataset when trained on upright and validated on rotated samples. The Polar Transformer Network (Esteves et al. [2018]) transforms the input to polar coordinates with the origin learned as the centroid of a single channel.

Yet another work exists that is similar to our approach (Zhou et al. [2019]). In their work, the authors propose obtaining rotation invariance via matching criterion and the kernel-mapping CNN (KM-CNN). Their proposal is to apply rotation transformations to the convolution kernels using shifting pixels instead of rotating them with a bilinear transformation. Using this method, they obtain 9.7% accuracy on the MNIST dataset when trained on upright examples and validated on random (multiples of 10°) orientations. Unfortunately, the proposed method to rotate the convolution kernels is limited to 45° increments. In this sense, the angular prediction is limited to ±45°.

Finally, the most recent advances go towards the union of techniques as reinforcement learning with rotation equivariance (Basu et al. [2021]). In their work, the authors show that the inclusion of equivariance improves neural network performance for classification. Furthermore, they use deep Q-learning to search (in a reduced solution space) for an equivariant representation of different transformations. Learning this representation by reinforced learning avoids the problem of finding the adequate balance between the number of features and a large number of symmetries on equivariant networks with a fixed number of parameters. They obtain competitive results on the MNIST dataset when trained on random orientations (87.2% accuracy). Nevertheless, as they use reinforcement learning, the network is trained several times with different parameters to let the algorithm learn the representation. In this sense, the training becomes exhaustive and complicated.

We can conclude that it is inducibly proven that the models transforming the input image are considerably larger than the ones based on the rotation invariance (or equivariance) built in the network. The current state of the art CNN achieve the rotation invariance by average pooling from the output of the network but lose the angular information of the input. Only a few of them can extract and predict the angular transformation of the input. To complete, the Table 2.1 contains a summary of the existing methods selected in function of the number of parameters and network capabilities.

As a conclusion of our study of the state of the art we can observe that rotational-invariance has been solved in several ways but finding several drawbacks for each approach. We demonstrate in this work that an efficient usage of the internal network filters and an adequate managment of the rotational equivariant properties can result in an improved accuracy for rotated imaged and the capability of predicting the input angle. Other means to improve the accuracy when classifying rotated objects include the usage of custom loss techniques

such as Pixels-Intersection-over-Union (PIoU) (Chen et al. [2020a]). The idea is that an oriented bounded box – a one that is aligned with the rotated object – presents less overlap with the background in complex environments.

Also, we conclude that it is possible to obtain a class invariant inner mapping where the translation of the filters is equivariant with the rotation of the examples. Also, the model can acquire the capability to learn the angle of the input without any angular label on the dataset. As a consequence of reusing the translation of the filters the network can remain small in learnable parameters and be oriented to embedded devices platforms.

Table 2.1 – State of the art in the domain of rotation-invariant object classification: summary of main properties. Legend: Size : Model size when applied to MNIST dataset; Upright - upright oriented examples only; n.c. - non communicated.

		Model		Output		Training approach	
Name	Dataset	Size	Tested rotations	Classification	Angle	Upright	Randomly rotated
TI-Pooling Laptev et al. [2016]	MNIST-rot	n.c.	24	yes	no	no	yes
Harmonic networks Worrall et al. [2017]	MNIST-rot	33k	Continuous	yes	no	no	yes
Spherical CNN Cohen et al. [2018]	MNIST/MNIST-rot	68k	Continuous	yes	no	yes	yes
SFCNNs Weiler et al. [2018]	MNIST-rot	n.c.	24	yes	no	yes	yes
GCNs Luan et al. [2018]	MNIST-rot CIFAR10	1.86M	4	yes	no	no	yes
Icosahedral CNN Cohen et al. [2019]	MNIST/MNIST-rot	n.c.	n.c.	yes	no	yes	yes
RotEqNet Marcos et al. [2017]	MNIST/MNIST-rot	100k	17	yes	no	yes	yes
RI-LBCNNs Zhang et al. [2018]	MNIST-rot	390k	8	yes	no	no	yes
ORN-8 Zhou et al. [2017]	MNIST/MNIST-rot CIFAR10	969k	8	yes	no	yes	yes
Rot.-Inv. Conv. Follmann and Bottger [2018]	MNIST/MNIST-rot CIFAR10/CIFAR10-rot	11M	8	yes	no	yes	yes
EqWS Saydjari and Finkbeiner [2021]	MNIST-rot	n.c.	6	yes	yes	yes	yes
KM-CNN Zhou et al. [2019]	MNIST/MNIST-rot	n.c.	7	yes	yes	yes	yes
RIN (scattering + CNN) Rodriguez Salas et al. [2019]	MNIST/MNIST-rot	7k	16	yes	yes	yes	yes
RIN (steerable + CNN) Rodriguez Salas et al. [2021a]	MNIST/MNIST-rot CIFAR10/CIFAR10-rot	42k	16	yes	yes	yes	yes

2.2 Roto-translational Feature Space

One of the main contributions of this thesis work is the study and proposal of a roto-translational feature space. A roto-translational feature space consists of a collection of features that translate following the rotation of an object on an image. In this sense, we can say that the features covariate linearly with the input's rotation. One of the main challenges is to obtain feature translation while preserving the angular relationship between the features.

First, an input image (Fig. 2.6a) on upright position (no rotation of the object in the image) is decomposed in a set of oriented features (Fig. 2.6b). These features then are a set of oriented features ordered by increasing angular magnitude. As the set of features contain rotation and relative positition of the filters information we need to produce a feature space that contains only the relative position of the filters translating with the input rotation. For this we compensate the rotation component of the features by rotating them with the negative angle that produces them (Fig. 2.6c).

Then, following this intuition we can observe that if the input image is rotated by some angle (Fig. 2.6d) and the rotation is compensated the feature space translates by a magnitude that is linearly dependent of the input angle and the number of oriented features present in the feature space (Fig. 2.6f). In the example case of Fig. 2.6 we can observe that the same feature present in the place $i = 0$ (with the input on upright position) moves one space over the feature space to $i = 1$ when the input is rotated. Finally, this means that the feature space translates over the depth axis following the input rotation.

Having this translation is one of the most important attributes needed to obtain rotation invariance. The translation over the feature space means that when translated over the axis a predictor could scan each translation and access to the rotation information of the input despite being trained on upright examples only. These two operations (roto-translation and translating predictor) can then produce rotation invariant networks when trained on upright datasets.

The intuition behind the roto-translation feature space then can be described mathematically as follows. In general, we propose a roto-translational feature space that is created by the oriented decomposition of the input image in several filters.

Let $\mathbf{x}$ be an example of the dataset $\mathbf{x} \in \mathbb{R}^{m \times n}$ with m and n being the width and height of the input image, and let g be a filter acting as an oriented edge detector with periodicity 2π. Let ρ_φ be a transformation rotating the support by the angle φ. Then $\rho_\varphi g = g^\varphi$, with $g{=}g^{0^o}$, be such a filter oriented along the angle φ.

The product $\mathbf{x} * g^\varphi$ extracts the oriented components of $\mathbf{x}$ that are oriented in φ. We have a set of orientations $\varphi_i = 2\pi i/\mathrm{N}$ for $i = 0, \ldots, \mathrm{N}$, with $\mathrm{N} \in \mathbb{Z}^+$, and $d\varphi = 2\pi/\mathrm{N}$. The ordered set $[\mathbf{x} * g^{\varphi_i}]$ contains all oriented components of $\mathbf{x}$.

Is straightforward to see that the input image $\mathbf{x}$ of size [m, n] is decomposed in N oriented components of the same size generating a feature space with size [m, n, N] (Fig. 2.6b). This can be observed in Figure 2.6b where each feature is an oriented component in the angle φ_i.

Second, let ρ_φ be a transformation rotating the support by the angle φ. Then

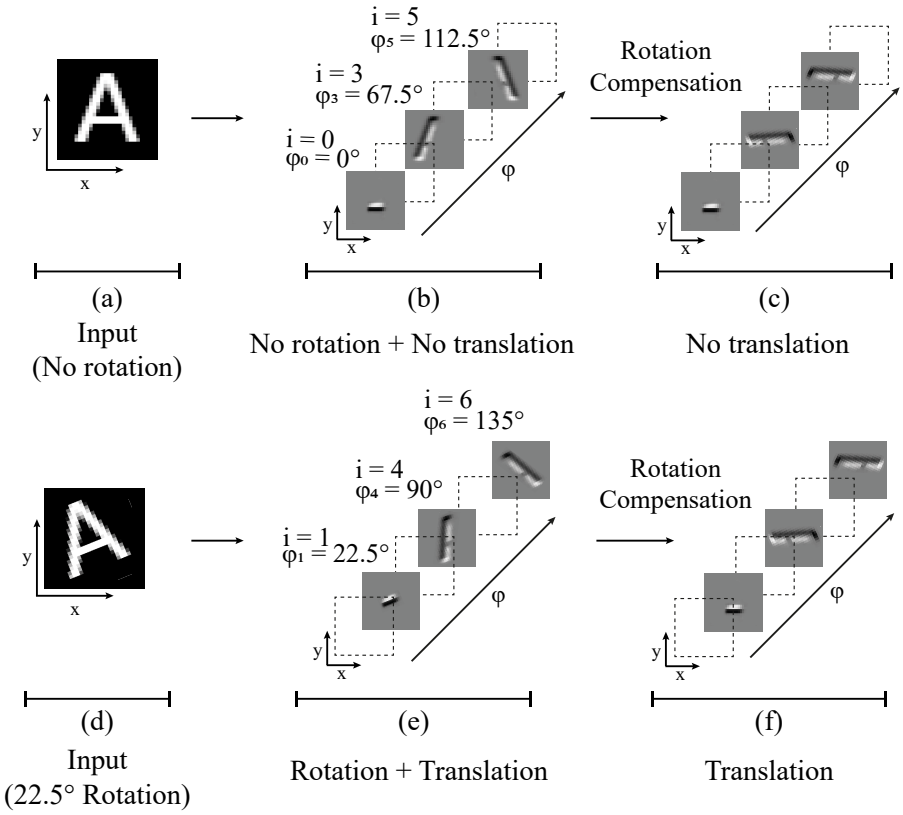


Figure 2.6 – Roto-translational feature space.

we compensate the rotation by rotating the support $-\varphi_i$ degrees. This re-orientation results in all the feature components to be aligned to the referential orientation φ_0 (Figure 2.6c).

To validate this, we experimented with different approaches and feature representations. We first started with a 1-D representation to make an easier understanding of the roto-translational properties of the feature space, then we analyze the main drawbacks of this approach and implement a 3-D representation of this feature space. Finally, we study how the translating predictor is applied to this feature representations and the expected results of this contribution.

2.2.1 1-D Feature Representation

The first tested approach was to transform the oriented feature representation into a 1D version of itself by flattening it. This operation means transforming the 3D dimensional space from $m \times n \times \mathrm{N}$ to a 1D dimensional space of size $\mathrm{V} = m \times n \times \mathrm{N}$ (Fig. 2.7). For example, for an oriented feature space of $14 \times 14 \times 16$ we obtain a 1D array of $\mathrm{V} = 3,136$. One of the main properties of this feature space is the translation behavior when the input is rotated by a magnitude of $d\varphi$. When the input image is rotated by $d\varphi$ a translation occurs over the feature space V in a magnitude equal to $m \times n$ (Fig. 2.8). For the example case this would mean a translation to the right (when input image is rotated clockwise) of magnitude 196.

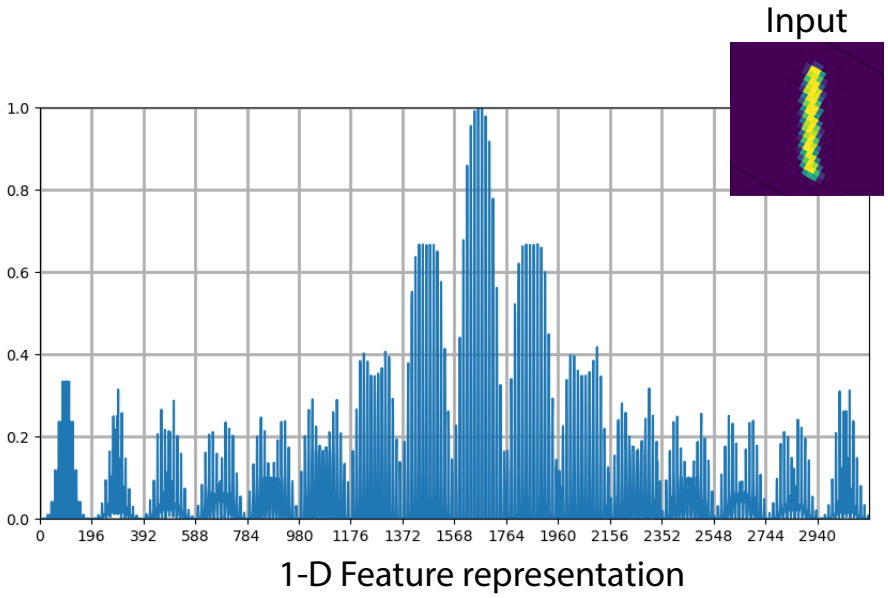


Figure 2.7 – Flattened version of the feature space using scattering transform features.

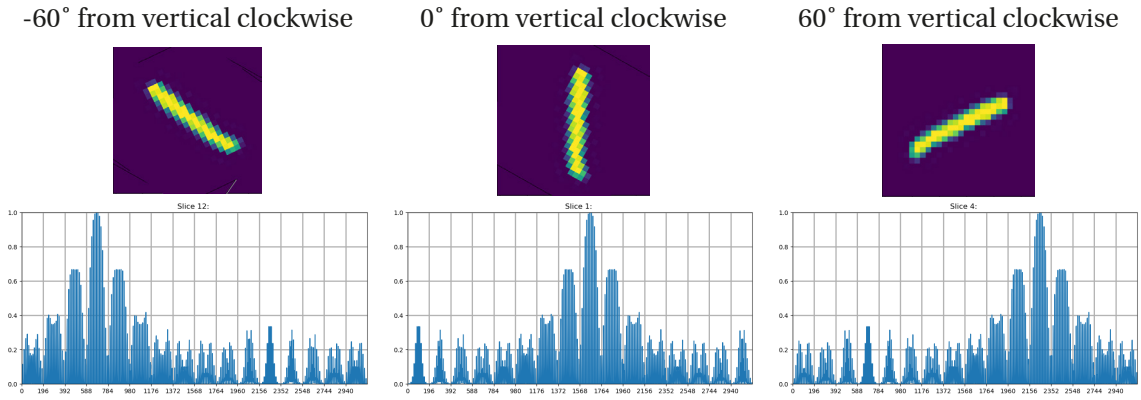


Figure 2.8 – The output contains each one of the translations in one full vector.

The second property of the flattened feature space V is the capability of capturing the internal properties of the input image. The angular distance between the edges present in the image are mapped to a linear distance inside the feature space. Let the letter X be the input image as an example. The letter X contains two angles between the edges one of 69° and other of 112° ; these two angles are mapped then into the feature space V as a distance of $6d\varphi$ and $10d\varphi$ respectively. For N = 16 we obtain $d\varphi = 11.25$ and by multiplying this by the linear distance ($6d\varphi$ = 67.50°and $10d\varphi = 112.5°$) we recover the existing angle between the edges of the input image (Figure 2.9).

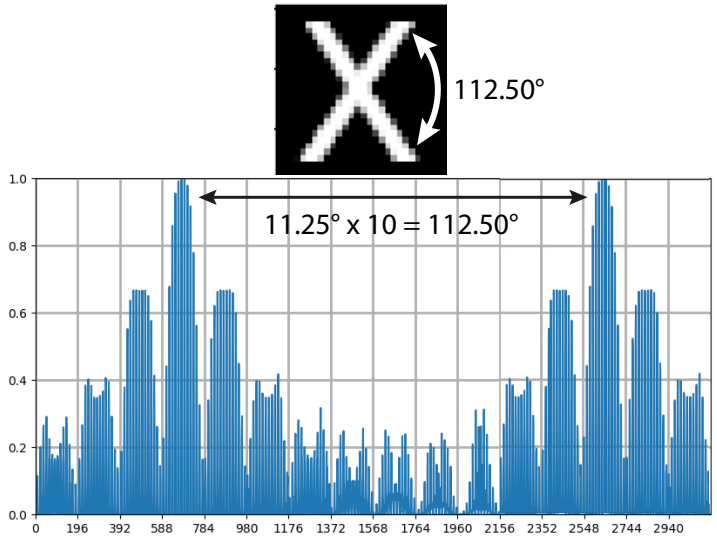
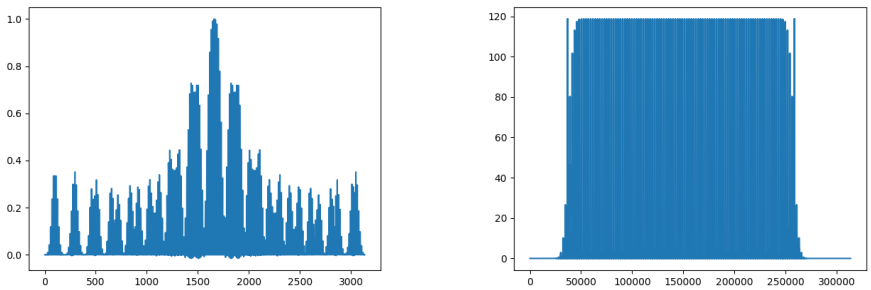


Figure 2.9 – Angular distance between the edges is mapped as a linear distance of the responses on the plot.

Scanning Order

One of the important points to obtain a roto-translational feature space is the scanning order of the activation maps. Usually, to transform a 2-D image to a 1-D vector the processor concatenates each row or column of the image into a single vector. As observed in Fig. 2.6c the features are aligned to the vertical reference to compensate the rotation of the input. To scan this features we verify that the scanning orientation is the same as the reference to which we aligned the features.



(a) Correct (b) Incorrect

Figure 2.10 – Scanning order output when scanned correctly and incorrectly.

The main consequence of an incorrect scan order is the lack of roto-translational properties in the resulting flattened feature space (Figure 2.10). The best way to avoid losing the roto-translational properties is to scan each one of the filters in the same angle φ that generates them (Figure 2.11). One straightforward way to achieve this is by re-indexing the images using a custom dense layer. The re-indexing process was done by generating the rotation indexes using a bilinear transform of the image and connecting the input pixel to the mapped output pixel following this bilinear transformation. This process is equivalent to re-orient the image to the vertical reference. For the re-orientation algorithm, we tested two alternatives nearest neighbor and bilinear. The second one delivered better outputs with less noise and the best quality.

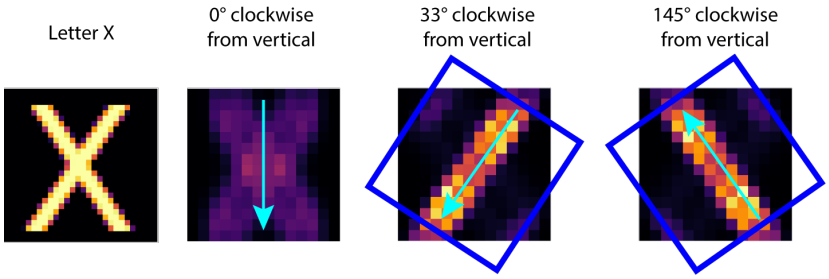


Figure 2.11 – Implemented scan order.

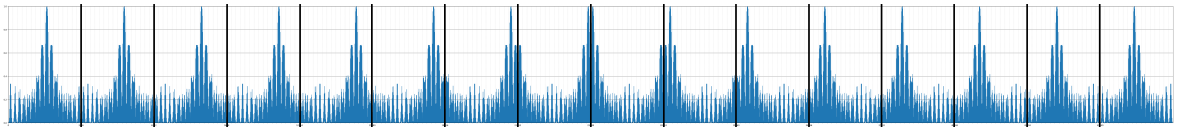


Figure 2.12 – The output contains the vectors of all translantions.

The custom dense layer re-indexes each pixel to a selected output pixel on the output vector, one of the advantages of this operations is that there is no training weights needed for the layer and is considered a frozen weights layer on the training stage. The main drawback of this implementation is that the number of parameters grows rapidly for image size and number of discrete orientations N.

As we can observe in Figure 2.12, the output vector contains the flattened version of the oriented feature space and the N possible discrete translations concatenated in one full orb. When we refer to a full translation orb we refer to the one dimensional matrix that contains all translations. Observe how the values translate to the right in a cyclic way. This vector (full orb) can be used by a scanning window of a 1D convolutional predictor to output a higher probability in the orientation corresponding to the upright position. While having the all the possible discrete translations in one vector can be useful and makes a straightforward neural network implementation it comes with a high number of parameters hence bigger memory requirements.

The number of non-trainable parameters of this layer can be calculated by the number of inputs multiplied by the number of outputs. For an image of size $14x14$ with N = 16 the input size is 3,136 and the output size is the flattened version of this space multiplied for each one of the possible translations of the

feature space (3,136 x 16 = 50,176). To obtain the number of non-trainable parameters we multiply the input size by the output size ($3,136 \times 50,176 = 157,351,936$). There several consideration to take into account about this huge amount of parameters. First, these paramaters are static and are not updated on training stage. Also, the main function of these layer is to re-orient the input features to obtain the correct scan order. Furthermore, as each angle position is independent this process could be paralellized. At the end, these considerations were enough factor to search for other solutions and test different representations such as the 3-D Feature representation explained in the next subsection.

2.2.2 3-D Feature Representation

The first 1D approach worked as proof of concept of the roto-translational approach. One of the main drawbacks of the 1D approach is that the convolution window becomes very large compared with the state of the art approaches (3,136 for a 14 x 14 image with N = 16 vs state of the art window of 3 to 5). The straightforward solution to this problem is to use a collection of filters (14 x 14 x 16) in the 3D form, and apply a 3D convolution with the size of the input image (14 x 14) and moving over the padded axis (31) with a stride = 1.

For an easier understanding of the features we use a planar representation of the 3D spaces (Fig. 2.13). As commonly used in deep learning, a 3D collection of features can be represented in 2D form with each feature aranged in the same plane. This representation allows to study each one of the feature individually and plot the relationship between them in a visual way. Each feature on fig. 2.13(b) represents a slice of the 3D feature space. In general, we use this representation for features and filters that are contained in a 3D space.

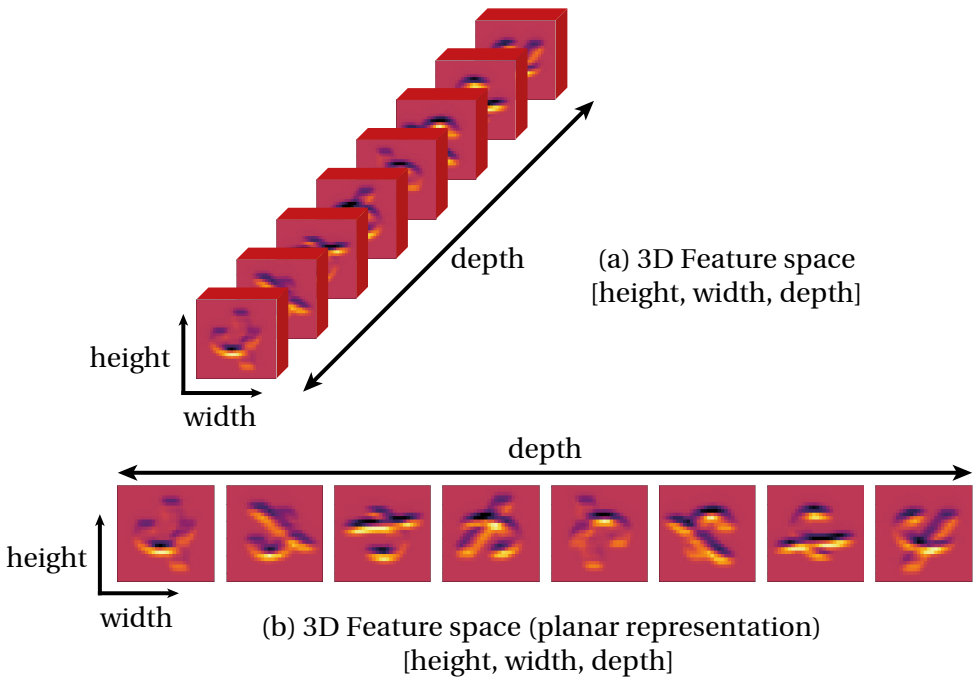
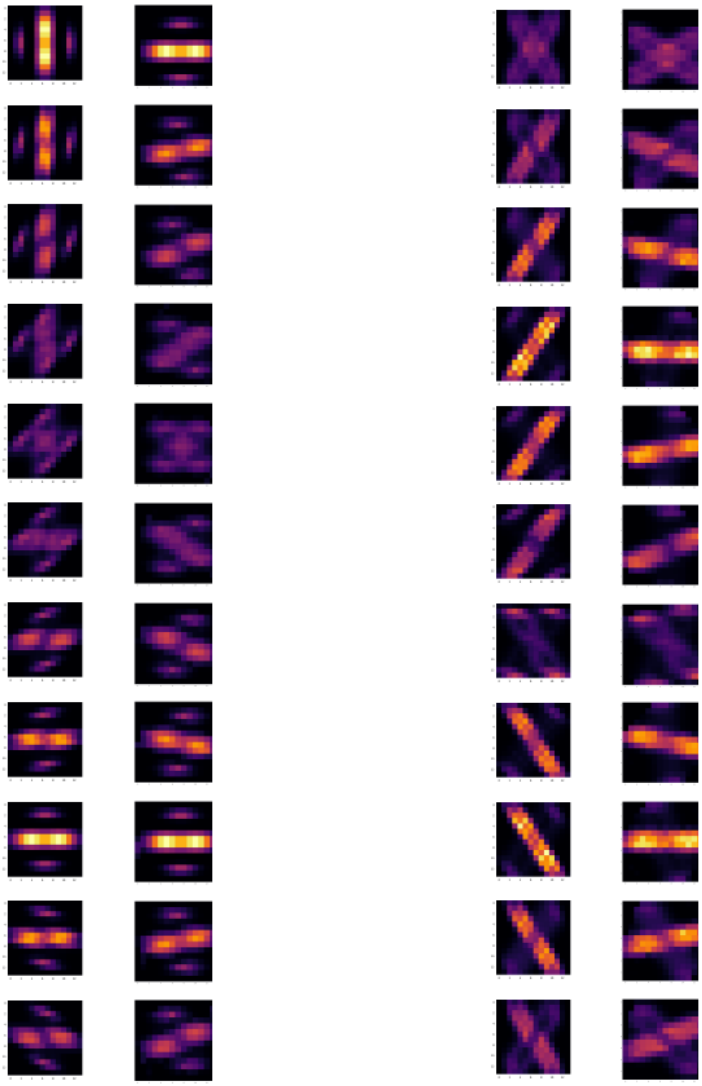


Figure 2.13 – Different representations of a 3D feature space of a hand-written number 4.

3D Re-orientation

Analog to the 1D approach the scan order is needed to obtain the expected roto-translational behavior. An straightforward way to obtain this scan order is to rotate the result image (after convolving with the oriented filter) by the negative angle of the oriented filter. In Figure 2.14 we can observe how these images are re-oriented from the oriented response to an horizontal aligned response. Figure 2.14 shows a Plus sign (a) and letter X (b) oriented response in first column, then in second column we observe how the response is oriented to the horizontal reference, notice how the higher energy activation is completely aligned to the horizontal reference.



(a) Plus sign
First column: Scattering transform
Second column: Re-oriented output

(b) Letter X
First column: Scattering transform
Second column: Re-oriented output

Figure 2.14 – Feature re-orientation step for horizontal scanning order.

The output of the orientation to the horizontal reference generates and oriented feature space that contains the same properties than the 1D counterpart. The angular distance in the edges of the input image is then mapped to a linear distance between the filter elements. The roto-translational property is present

in the sense that a rotation change in the input image translates proportionally the aligned feature space. For example, in Figure 2.15 we can observe a two places translation over the feature space for a rotation of $2d\varphi$.

In the next section, we test the discrimination capabilities of this collection of oriented filters on a convolutional neural network. For this test, we select a more complex task such as face recognition. Using a complex task allows us to observe what accuracy we obtain with these oriented filters as the first layer without invariance to rotation.

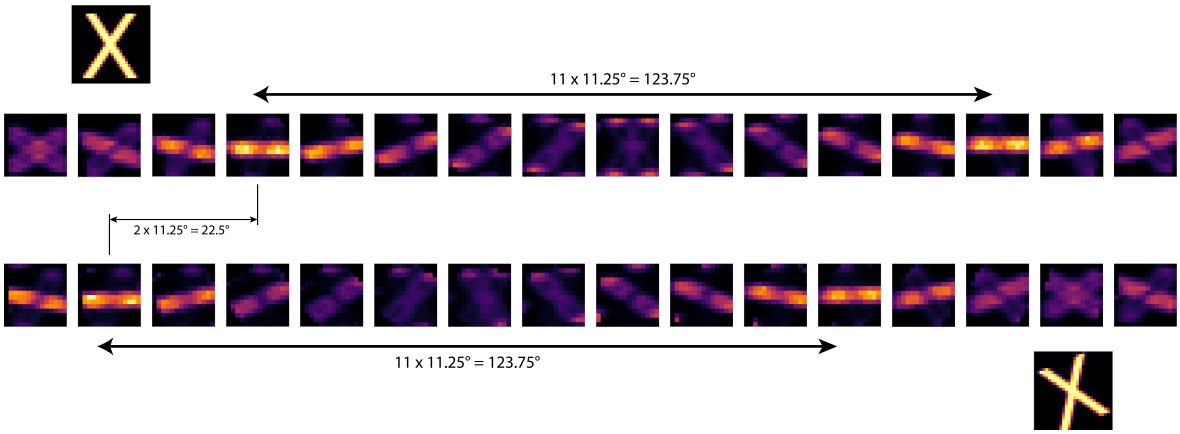


Figure 2.15 – Translation can be observed over the 3D space, angular information is preserved as a linear distance over the filters.

2.2.3 Face Recognition Using an Oriented Feature Space

Face recognition refers to the task of labeling a face against a database of faces. Usually, big companies train CNNs with several thousands of labeled examples to solve the face recognition problem (Kumar et al. [2009], Huang et al. [2007], Liu et al. [2015]). The network training goes up to several hours on specialized hardware to achieve state-of-the-art results (Coleman et al. [2017]).

Several works on the state-of-the-art exist about the face recognition problem (Taigman et al. [2014], Schroff et al. [2015], Deng et al. [2019]). Usually, they work with upright-oriented faces or use data augmentation for robustness against deformations. Also, most of them focus on the complete pipeline: face detection, alignment, feature representation, and classification. In this experiment, we focus on the last two tasks, feature representation, and classification.

The typical workflow includes two branches (Fig. 2.16). The first branch extracts the features from the input frames (video or images). The second branch obtains features from a face database (gallery). In the end, a classifier compares both features and outputs a predicted similarity index result.

To follow this workflow, we based our first studies on the previous internship study of Steeve Sivanantham on facial classification with reduced datasets (Sivanantham et al. [2017]). The authors propose a CNN for face recognition in two stages: An invariant scattering convolution network based on scattering transform (Bruna and Mallat [2013a]) to obtain the features from the input image and a siamese network to make the prediction (Bromley et al. [1994]).

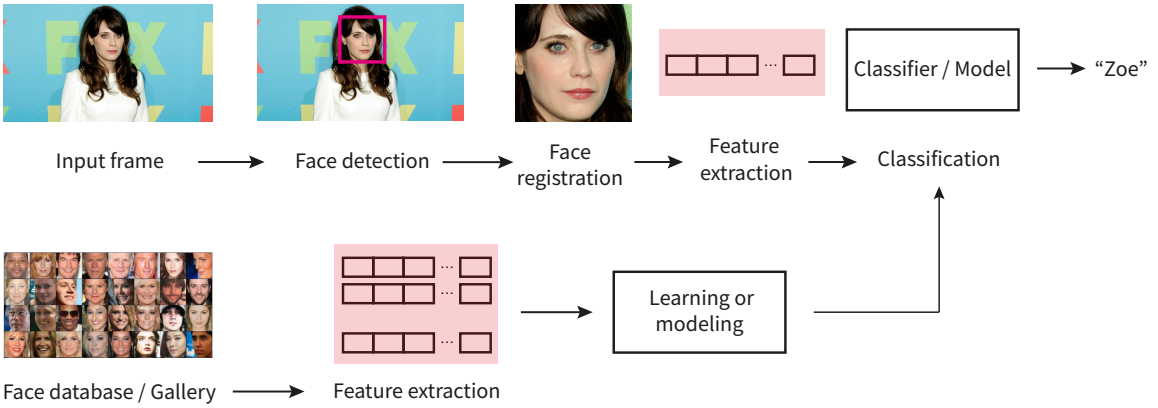


Figure 2.16 – Face recognition workflow.

The scattering transform is an orthogonal transform based on wavelets. A wavelet scattering network computes a translation-invariant image representation stable to deformations and preserves high-frequency information. In this sense, it decomposes the input image in several oriented components arranged by increasing magnitude. We outline this transformation characteristics on Chapter 3.

A scattering convolution network is formed then by the union of the scattering transform output and a CNN. After the oriented decomposition of the input, a series of convolutional layers and maxpooling operators are connected. A hidden dense layer connects the convolutional feature extraction stage and outputs the results in the one-out-of-many format (Fig. 2.17). We show and discuss the scattering convolution network results applied to a face recognition problem on section 2.2.4.

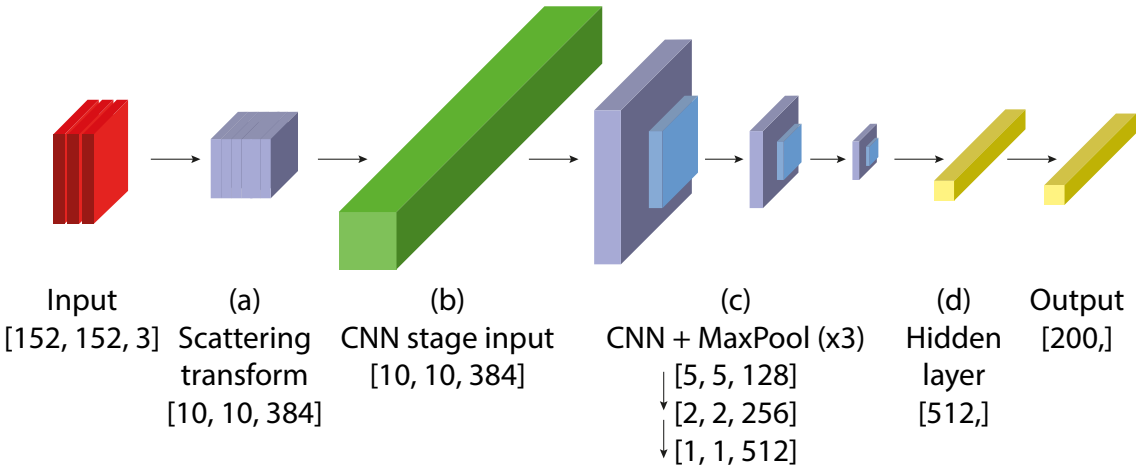


Figure 2.17 – Scattering convolutional network.

For the next step, we implement a siamese network using the previously trained scattering transform network. Obtaining a favorable result (accuracy convergence) would mean that the network can train, despite using oriented filters as the first later of the convolutional neural network.

A siamese neural network is a type of architecture that contains two identical subnetworks (branches) (Bromley et al. [1994]) (Fig. 2.18). Identical subnetworks mean that the configuration, connections, and weights of the networks

are the same for both subnetworks. Each subnetwork will then have an input (a face image) and output a feature vector. The siamese neural network then outputs the similarity between the feature vectors in the form of distance from each class (Bromley et al. [1994]).

The siamese neural network architecture allows one-shot learning, meaning that after trained one image is enough to allow the network to recognize this image in the future. As the siamese neural network has two inputs (two copies of the same network), the first input is a ground-truth database image of the face we want to classify. The second input contains the images that will be tested similar or not to the first one. These images do not need to be part of the training dataset that trained the scattering network (Koch et al. [2015]).

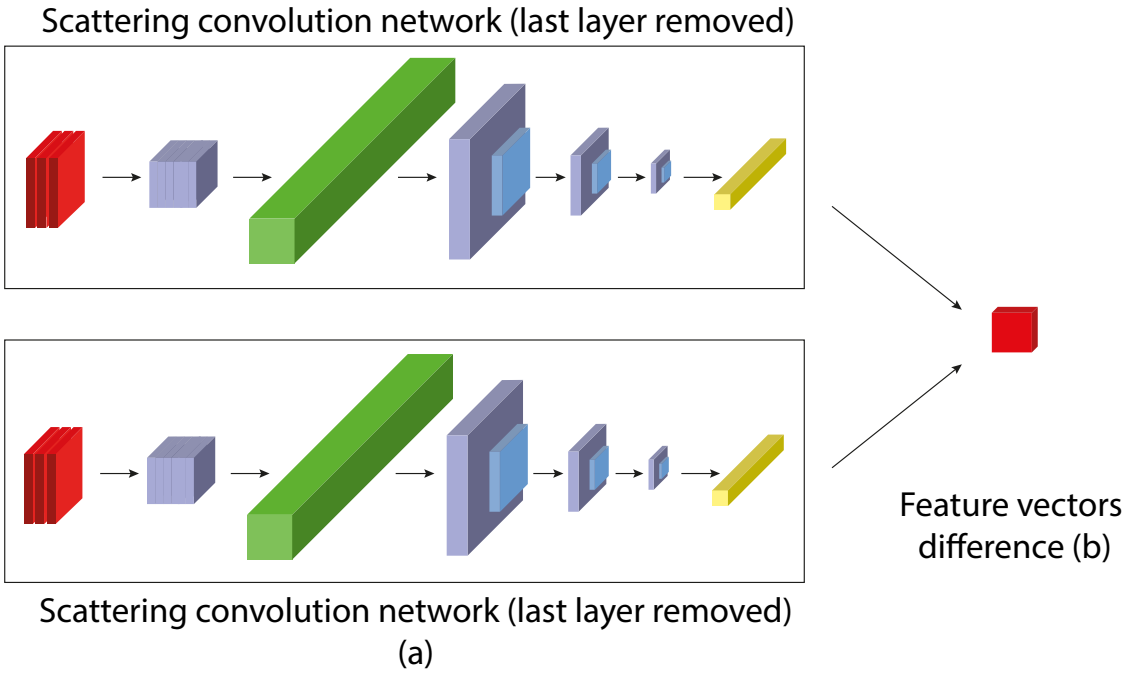


Figure 2.18 – Scattering siamese network architecture.

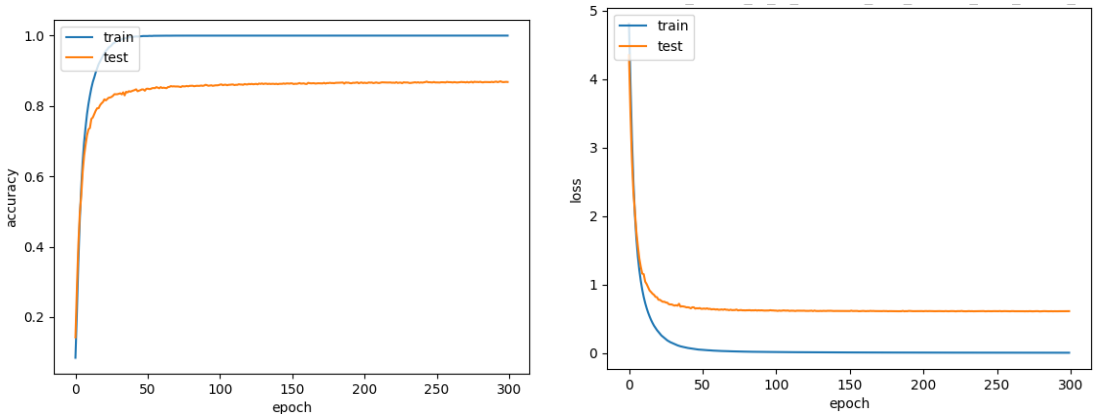
In the following subsection, we show the experimental setup and results of these models applied to the face recognition problem. We start by outlining the used dataset for training and validation. Then we show the results on the scattering neural network and siamese network. Finally, we implement the siamese model on an NVIDIA Jetson TX1 board and discuss its performance.

2.2.4 Experimental Setup and Results

To test the presented model architectures, we selected the Labeled Faces in the Wild (LFW) dataset (Huang et al. [2007]). This dataset contains 58,000 photos with 200 IDs. The face from the input images is cropped to 152 x 152 pixels in a pre-processing step. Hence, the dataset contains faces with eyes and mouth in almost the same position and orientation.

The first network to test is the scattering transform network (Fig. 2.17). For this, we first process the dataset and extract the scattering features in a pre-computed step. We store these features, and then we train the network using

these features. The network has in total 2.2 M parameters. We trained the network for 300 hundred epochs on an NVIDIA Titan Xp, with each epoch taking about 15 seconds to train (not including the scattering transform time). After training, we reached a test set accuracy of 85 % with this model (Fig. 2.19).



(a) Accuracy (b) Loss

Figure 2.19 – Scattering network accuracy and loss for LFW dataset. Training values in blue, validation values in orange.

The second model to test is the siamese neural network (Fig. 2.18). Once we trained the scattering model, it is duplicated and trained in pairs of faces. To achieve this, we make a second instance of the scattering neural network and delete the last layer for both (containing the output classes). This process means that the output of each branch network is a feature vector encoding the input image.

The principle consists of presenting a set of two faces, each one to one of the network branches. Each branch then will output a feature vector based on the features of the input image. Then, we calculate the difference between these two vectors. In the end, this value indicates the similarity between the two faces, and if the result is low, then the two presented images are probably the same person.

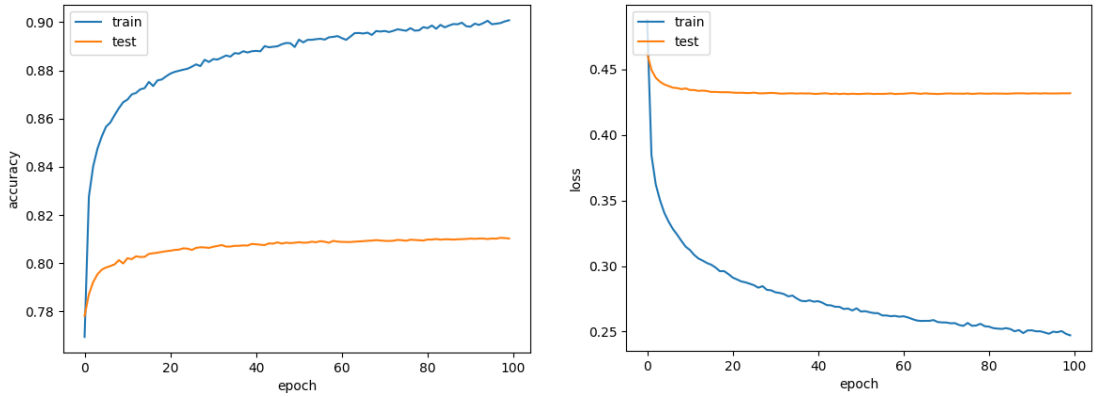
By following this methodology, the user can add more faces to the newly generated dataset and compare them with a live feed from a camera. There is no need to re-train the network each time the user adds an image as the network recognizes the similarity between the two input faces.

To generate the dataset first, we generated pairs of faces that belong to the same person and labeled them as identical. Then, we generated pairs that belong to different people. Finally, we trained the network for 100 epochs using these pairs of labeled face images.

The siamese network contains 2.4M of trainable parameters. While the number of parameters is roughly similar to the scattering transform network since several of them are shared between the two replicas, it requires twice the computation (to process each of the images on each branch). This increase in computational effort becomes a significant drawback on embedded devices with limited resources.

The scattering-siamese network achieved 80% of accuracy after hyper-parameter fine-tuning. Part of this fine-tuning step was to select the cropping size of the face (how close the face appears in the image). Despite the hyperparameter

tuning, different crop sizes and different numbers of filters, the result capped at 80%(Fig. 2.20).



(a) Accuracy (b) Loss

Figure 2.20 – Scattering-siamese accuracy and loss for LFW dataset.

While the results of these experiments do not seem encouraging (80% accuracy), they answer the initial hypothesis about the possibility of using oriented wavelets in classification problems. This proof of concept means that we can obtain classification results based on our main contribution of the feature space's intrinsic rotations. These results open the possibility of using these ordered oriented features as a roto-translational feature space which with a well-designed predictor can endow with rotation invariance properties the convolutional neural networks.

Finally, to test the network functionality on devices with limited resources, we deployed the network to an NVIDIA Jetson TX1 board. We used a test page with three faces to test the siamese neural network and presented it via webcam live stream to the network (Fig. 2.21a). Then, we added the same three-person faces to the siamese database (which feeds the first network branch) (Fig. 2.21b). We then feed the network's second branch from camera images of a webcam connected to the board. It is important to note that any of these person's faces were present on the training dataset.

With the Jetson TX1 board limited resources, the network runs at 4 fps on average and can classify each one of the faces correctly (Fig. 2.21). During the tests, there were several false positives and wrong classifications. On the results window (Fig. 2.21c), we can notice the values for the similarity between images to be highly variant between frames, and in some tests, not classifying adequately the face.

In the next section, we introduce the properties and behavior of the translating predictor. This predictor uses the recently studied roto-translational feature space to obtain a value for each translation contained in the space. We discuss how we apply it to the 1-D and 3-D feature space implementation and obtain the probabilistic distribution containing class and angle information.

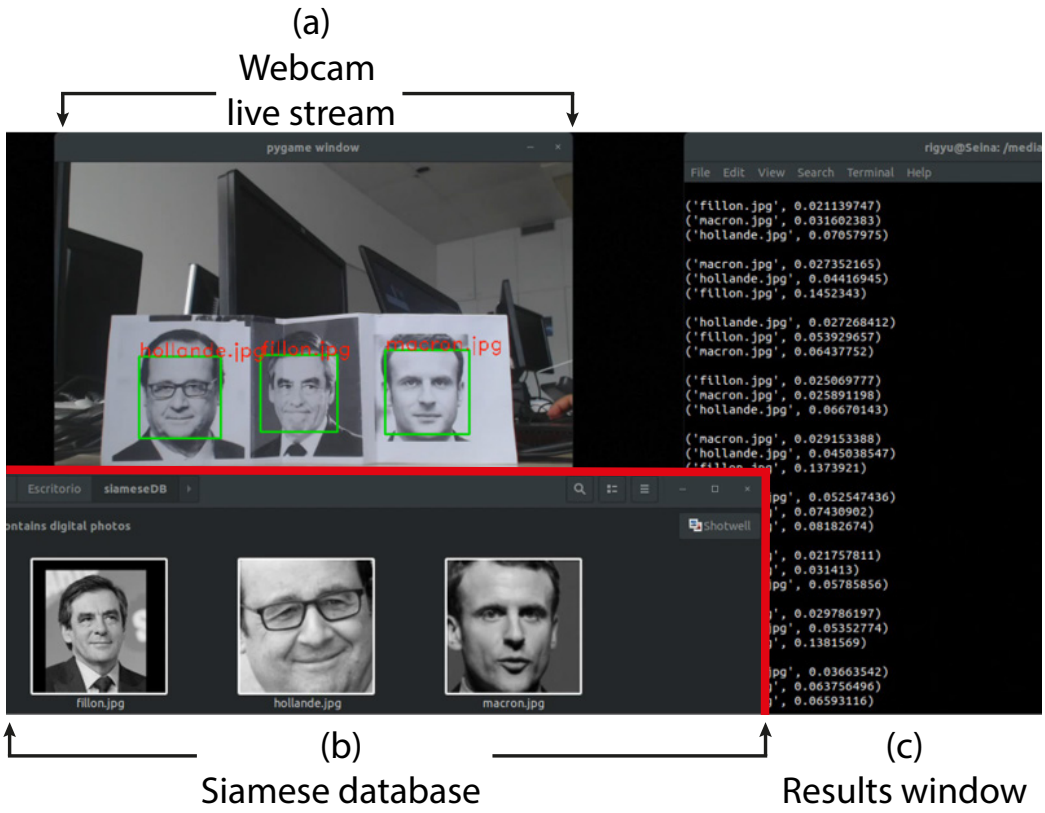


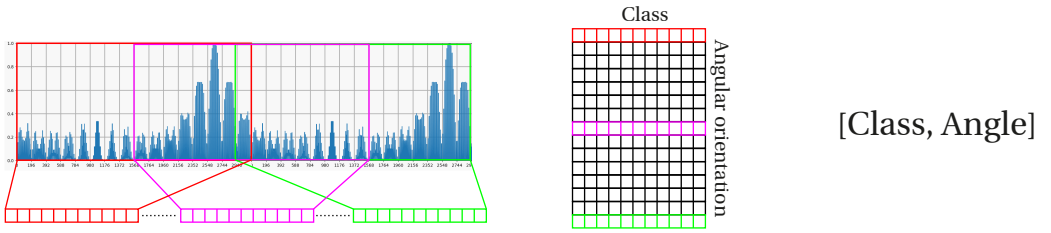
Figure 2.21 – Siamese network for Face Classification on NVIDIA Jetson TX1.

2.3 Translating Predictor

As demonstrated in the previous section, the feature space is now a roto-translational feature space. This means that the feature space translates proportionally to the rotation of the input image. In both cases, 1D and 3D, the feature space contain all the possible discrete translations corresponding to the sampling of the input. The next step is to apply a predictor that scans the roto-translational feature space and predicts at each translation. The result of these predictions will contain a maximum value where the trained orientation is contained (usually upright position), a similar value in the neighbors of the correct orientation due to the predictor tolerances, and a low value in the other orientations. Training the predictor to be angularly selective is of importance, so it can discern correctly between the trained orientation and produce a near-zero value in the other orientations.

To obtain this behavior we propose using a 1-D dense-layer predictor translating along the axis and performing a convolution of the input and the weights. The input is cyclic with periodic boundary condition. This cyclic convolutional predictor with the appropriate size and strides can scan the feature space cyclically and output a result for each one of the possible translations of the space. Unfortunately, at the time of this writing, a straightforward implementation of a cyclic convolution that allows flexibility on different platforms and easy deployment was not found. An alternative approach to circumvent this issue is to pad the feature space by itself and apply a convolutional predictor over this pad-

ded feature space. As shown in Fig. 2.22, this padded feature space contains the translating feature space, and to the predictor window, this feature space translates on each one of the strides.



(a) Oriented predictor scan (b) Oriented classes matrix (c) Prediction output

Figure 2.22 – Predictor applied to 1D feature space. (a) Sliding window observes a translation of the feature space. (b) The obtained prediction of each translation is stacked in a 2D table. (c) A maxpooling operation over columns and rows results in the predicted class and angle.

Applying this convolution window over the padded feature space is equivalent to scan each translation individually. Each one of the predictions is then stacked in a single oriented classes matrix. As the predictor's output is in the one-out-of-many format, it is equivalent to a vector of size K where K is the number of classes in the training set. Consequently, the columns will contain the class information. The number of rows is then determined by the number of translations contained in the feature space that is the number of sampled orientations on the input image. As one of the translations belongs to the trained orientation, only one of the rows of the table contains a maximum value corresponding to this predicted translation.

It is straightforward to obtain the class and angle using a maxpooling operation over the rows and columns to obtain the class and angle of the input. To obtain the angle, the distance from the upright oriented row and the predicted row is calculated and multiplied by $d\Phi$.

For the 3D approach, we follow the same methodology as its 1D counterpart. We have a 3D feature space in the shape of [height, width, N]. As the first layer decomposes the input using oriented filters, the result of each oriented filter is saved in the depth of the 3D feature space. The same concept of the 1D approach is then applied, the feature space is padded by itself, and then a 3D convolutional predictor is applied. This behavior is analog to a cyclic 3D convolutional predictor behavior Fig. 2.23.

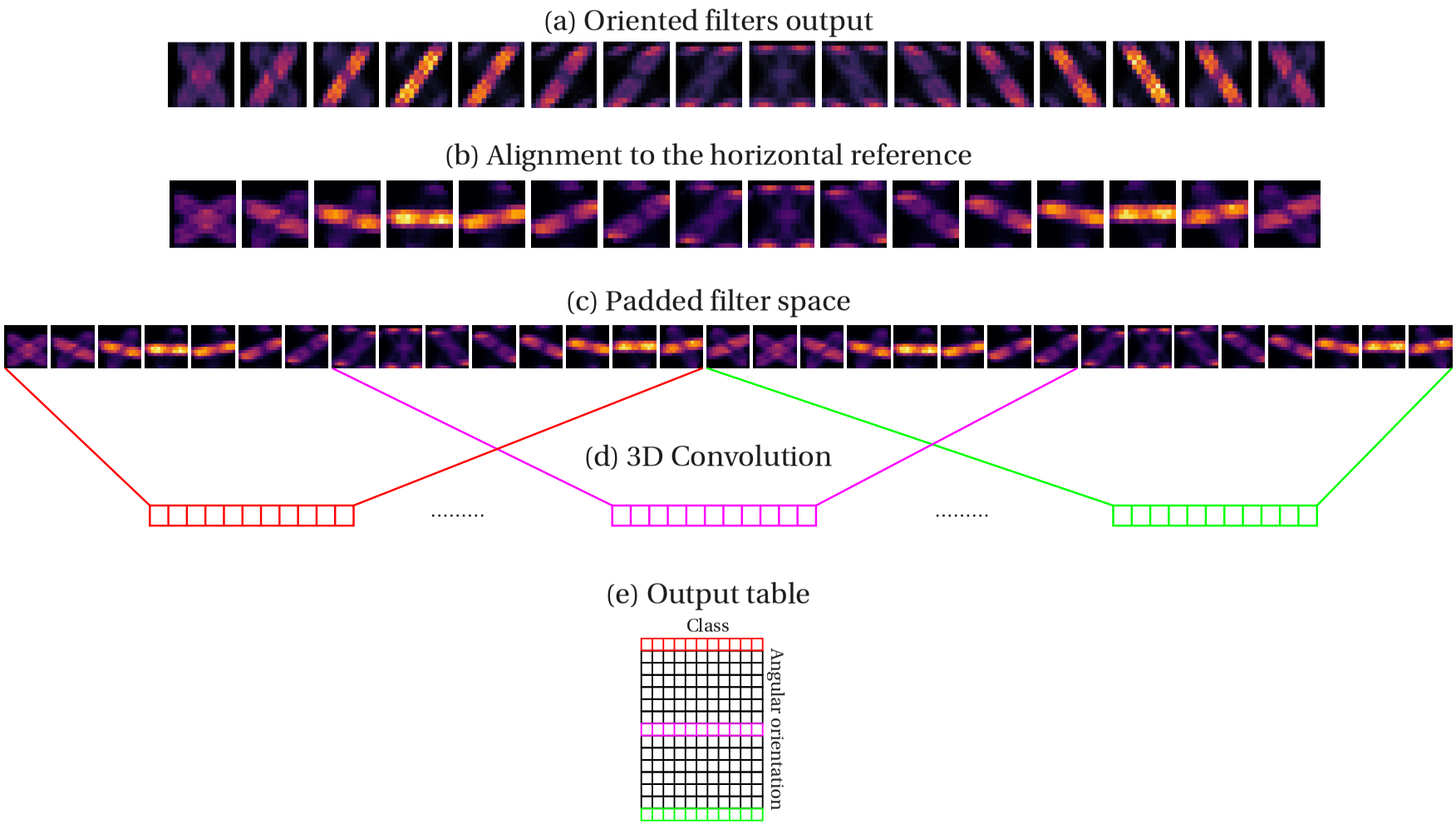
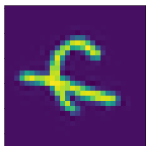


Figure 2.23 – 3D predictor scan approach (in 2D representation for clarity). (a) Feature space output from the oriented filters ; (b) Each filter is aligned to the horizontal reference ; (c) Padded filter space ; (d) A 3D convolution is applied to the padded filter space to emulate the behavior of a cyclic convolution ; (e) Analog to the 1D approach each result is saved in a 2D array that contains class and angular orientation.

Each one of the predictions is then stacked in an probability distribution (P) (Fig. 2.23e). For a roto-translational feature space that contains four dimensions [width, height, N, features], a 3D convolution is needed to scan each translation. We code this as a 3D convolution of size [5, 5, N] followed by a fully connected (dense) layer. Each result then becomes independent of the other. The translating predictor becomes angular selective in the sense that it learns to output a high probability for the translation that corresponds to the almost un-rotated version of the input.

We can observe the angular selectivity property of the predictor in the output table of the model before applying the globalmaxpooling operation. Recall that this output table is a probability distribution containing the class information over the columns and angular orientation in the rows. The output of this predictor stage is a probability distribution in the form of $[N, K]$ with N the number of orientations, and K the number of classes (Fig. 2.24). The highest row value corresponds to the angular information and the maximum column corresponding to the class.

Input image



Classes

Orientation index	0	1	2	3	4	5	6	7	8	9
0	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
1	0.	0.	0.	0.	0.01	0.	0.	0.	0.	0.
2	0.	0.	0.	0.	0.85	0.	0.	0.	0.	0.
3	0.	0.	0.	0.	0.15	0.	0.	0.	0.	0.
4	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
5	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
6	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
7	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
8	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
9	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
10	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
11	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
12	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
13	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
14	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
15	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.

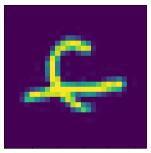
Maximum value located at - Row: 2, Column: 4

Figure 2.24 – Output table for an input with a rotated number 4 from the MNIST dataset. $[N = 16, K = 10]$

Given the angular sampling $d\varphi$ then it suffices for the dense layer classifier to be rotation invariant up to $\pm d\varphi/2$. A finer angular sampling $d\varphi$ will require a smaller rotation invariance from the classifier, hence a smaller model. Notice that

a self-organizing behavior appears in a continuous mapping of the rotation angle to the probability distribution P. This behavior is a consequence of the design of the network and the training. First, the predictors are ordered at an increasing angle. Second, during the training, an example pattern, rotated in an angle not necessarily precisely equal to an entire multiple of $d\varphi$, is presented to the network on training. In this sense, it makes the classifier become rotation-invariant to misalignments up to $\pm d\varphi/2$. When a pattern is presented to the network during the training, then one row of the architecture will predict a maximum class probability. However, also its immediate neighbors will predict the same class with a somewhat smaller probability of the same class (Fig. 2.24). Continuous mapping of the rotation progressively appears during the training. This behavior is similar to that of the Kohonen self-organizing maps Kohonen [1990] without being explicitly fostered algorithmically.

Input image



Classes

Orientation index

	0	1	2	3	4	5	6	7	8	9
0	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
1	0.	0.	0.	0.	0.72	0.	0.	0.	0.	0.
2	0.	0.	0.	0.	0.28	0.	0.	0.	0.	0.
3	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
4	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
5	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
6	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
7	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
8	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
9	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
10	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
11	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
12	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
13	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
14	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
15	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.

Maximum value located at - Row: 1, Column: 4

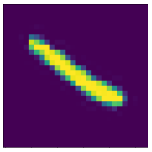
Figure 2.25 – Output table for an input with a rotated number 4 from the MNIST dataset. Compared to Fig. 2.24 the input image has rotated $d\varphi$ degrees. [$N = 16, K = 10$]

Rotation invariance can easily be observed on the output table P when the input image is rotated by some angle. Notice that there exists a linear correlation between $d\varphi$ and the magnitude of the angle. This means that for an input rotation of $d\varphi$ the maximum value location moves one row (Fig. 2.25).

It is important to recall that the training stage of the network is angle blind. At

the training stage, a globalmaxpooling operation is applied to table P. The output of this operation contains a probability distribution in the one-out-of-many format with the size equal to the number of classes K. Hence, the network is not directed towards selecting a specific row for the upright position of the image.

Input image



Classes

Orientation index	0	1	2	3	4	5	6	7	8	9
0	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
1	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
2	0.	0.03	0.	0.	0.	0.	0.	0.	0.	0.
3	0.	0.44	0.	0.	0.	0.	0.	0.01	0.	0.
4	0.	0.06	0.	0.	0.	0.	0.	0.	0.	0.
5	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
6	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
7	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
8	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
9	0.	0.01	0.	0.	0.	0.	0.	0.	0.	0.
10	0.	0.07	0.	0.	0.	0.	0.	0.	0.	0.
11	0.	0.32	0.	0.	0.	0.	0.	0.	0.	0.
12	0.	0.03	0.	0.	0.	0.	0.	0.	0.	0.
13	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
14	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.
15	0.	0.	0.	0.	0.	0.	0.	0.	0.	0.

Maximum value located at - Row: 3, Column: 1

Figure 2.26 – Output table for an input with a rotated number 1 from the MNIST dataset. [$N = 16, K = 10$]

As explained previously, the network self-organizes to map each angle to one specific row continuously. The starting row is not directed in training. It means that the network randomly selects a row that corresponds to the upright position as the starting point for the continuous mapping of the angles. In this sense, let R_{ur} be the index row of the probability distribution P when predicting an upright oriented class. While selected randomly by the network, this value is constant per class (i.e., for class 4, the upright row is always 13).

Also, let $R_{predict}$ be the index row predicted by the network from an input image with an unknown orientation. To calculate the predicted angle then we use the upright row R_{ur}, predicted row $R_{predict}$ and $d\varphi$. For example, for $R_{predict} = 1$ (Fig. 2.25), $R_{ur} = 13$ and $d\varphi = 22.5°$ we calculate the predicted angle by $\theta = 360° - |R_{ur} - R_{predict}| d\varphi$. In this case, the predicted angle is 90°.

An interesting observation occurs when the evaluated class is symmetrical. For example, in some written forms of the number 1, it is identical if observed

upright or upside down. After several experiments, we observed that the network splits the maximum probability between the two possible observations of the number 1 (Fig. 2.26). A maximum value appears at row 3 and a similar value at row 11.

The linear distance between the maximal values is eight, which corresponds exactly to half of the angular sampling of N = 16. Transforming this linear distance to degrees would result in $8d\varphi = 180°$. This result means that the network correctly recognizes the two possible orientations of the number 1, successfully predicting the class and the angle despite having two correct answers.

In this sense, if a perfectly isometric class (it is classified equally in more than one orientation, e.g., number 1) is presented to the network that does not variate with the angle, the network output will contain similar maximum values over the column that corresponds to the class. So, there could exist a mix between rotation invariant objects and variant ones in the training set, and the network would learn to identify them correctly.

In the next section, we discuss the translating predictor training. Also, we describe the intuition behind the backpropagation algorithm and how the predictor weights are reinforced for the upright orientation.

2.3.1 Training of the Translating Predictor

As introduced in the previous paragraphs, the translation predictor consists of a 3-D convolutional layer (for the 3-D feature space) that scans each translation of the oriented feature space (Fig. 2.23). Following this convolutional layer is a hidden dense layer that acts as a predictor for the classifier and then outputs layer with as many neurons as classes (K).

The predictor then scans each translation to output a predicted result for the class. We can view this scanning predictor as a set of identical copies (they share weights), each with attention to each translation of the feature space. In this sense, they can be all instantiated and their execution parallelized as there is no data dependency between them. We will see later (Chapter 5) that these parallel predictor are small in size as they do not need to become invariant to the rotation of the input. At the end, the network stores the results of each translation prediction in a probability distribution table (P).

On the forward pass prediction, the maxpooling operation selects the maximum value of the table over an axis. The maximum row outputs the predicted angle, and the maximum column the predicted class. For the training stage, we do not provide the network with angular labels.

For the training stage we use the categorical cross entropy loss function. This function is commonly used on classification problems. During this training stage, the maxpooling operation is active by selecting the classifier that has predicted the maximum likelihood of some class. The classifier that generated the maximum row is then selected, and the network uses this branch during the backpropagation step. We can observe this on Fig. 2.27 where the colored arrows reperesent the backpropagation branches from the output towards the input.

The prediction is reinforced over the branch that corresponds to the upright oriented features by the backpropagation algorithm (Fig. 2.27 green arrow). This

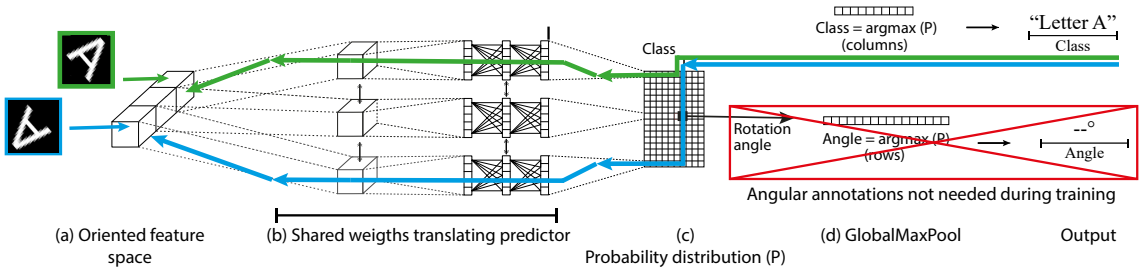


Figure 2.27 – Error backpropagation during training. Only one of the coloured backpropagation path is active at a time. The path indicated by the other colour is used when the same example is eventually presented to the input with a different orientation.

training setup does not reinforce the other positions, just the upright-oriented features. During the training, the same example is eventually presented to the network again (albeit at another rotation). Then, the classifier predicts the class (at some other position, though), and the prediction of this class will be reinforced further during the backpropagation on the corresponding branch (Fig. 2.27 blue arrow).

The main consequence of this process is that the predictor becomes angularly selective. It becomes reinforced as an upright-oriented classifier and is not reinforced in any other orientations. Thus, the predictor can remain small and does not need to grow large to encompass all rotated copies inside the model compared to the larger predictors product of using augmented training.

With this methodology, it is possible to train the network in both ways: using a dataset of upright-oriented examples or randomly oriented examples. For the first, the classifier becomes angularly selective to the upright position pattern on training. If validated on a randomly oriented example, it will find the upright pattern in some translation position over the oriented feature space. In this sense, the classifier becomes invariant to rotations while keeping its reduced size. The network can also be trained on randomly rotated examples keeping the same rotation invariant properties.

To complete these statements, we present the mathematical proof of trainability of the network in Chapter 5. In this mathematical methodology, we outline the characteristics of the translation predictor and its trainability characteristics. Also, we outline the congruence between the proof Chapter (5) and the experiments presented in this document.

2.4 Conclusions

In this chapter, we presented an approach that allows a Convolutional Neural Network input to be transformed using a series of oriented filters. To obtain a roto-translational feature space, it is important to respect the scanning order of the images and the alignment of the resulting wavelets to obtain a translation on the feature space. Also, we introduced the concept of the translating predictor applied to the roto-translational feature space.

Obtaining a roto-translational feature space is important to this work because

it can preserve the angular properties of the input image as the distance between the filters. This property opens the possibility to explore different approaches that can be applied to this roto-translational feature space to predict the input's angle using the embedded angular properties.

For the feature space we presented two different approaches using 1D and 3D features. While the presented 1D approach presents several drawbacks as having the full translation orb and convolution with a big filter window, it was the first approach and aided in understanding the roto-translational properties of the feature space. On the other hand, the 3D approach presented several advantages when paired with a 3D convolution predictor. The main advantage of this 3D approach is the low-memory footprint needed to capture the angular interaction of the filter space and the straightforward implementation of the predictor over the translations. The rotation invariant property is acquired while preserving the number of trainable parameters and without data augmentation techniques.

As the feature space contains the information of the translated input, it was demonstrated that it contains all the possible discrete translations up to the angular sampling. A subsequent predictor can scan over these translations and obtain a prediction for each one. As the feature space translates with the input angle, the angle can be predicted by knowing the position of the best prediction of the network.

This predictor can be implemented in two different ways depending on the application requirements. It can be a set of identical copies (shared weights) of the predictor applied to each translation in parallel. Alternatively, it can be a single predictor that serially scans each translation. In this sense, the first approach would be faster as the operation is done in parallel. The second results in smaller networks as the predictor is not replicated as many times as translations exist over the feature space.

Also, we proved that it is possible to use the oriented feature space as the first layer of a convolutional neural network on a complex task such as face recognition. In this sense, the next step is to test the aligned feature space with the translating predictor and verify its capabilities. In the following chapter, we test this on simple datasets and propose to replace the scattering transform with steerable filters to obtain the oriented feature space.

Chapter 3

Rotation Invariant Networks on Simple Datasets

The most common method to develop new concepts and test ideas with Convolutional Neural Networks (CNNs) is to start on simple datasets. We refer to simple datasets as the type of dataset that does not contain several light variations, different color conditions, or several different input transformations. Thus, to validate the roto-translational feature space presented in Chapter 2, we propose the test on simple datasets.

One of the most used simple datasets in the literature is the MNIST dataset (LeCun and Cortes [2010]). We classify this dataset as simple because it presents a single object in the image (a digit) without out-of-plane rotations and does not contain textures. The validation on this dataset also allows describing the roto-translational feature space properties and compare the network results in accuracy terms. Furthermore, it allows experimenting with different alternatives for the oriented feature space.

As explained previously (chapter 1), one of the main contributions of this work is endowing with intrinsic rotations the feature stage of the network. First, we generate an oriented feature space. Then, we re-orient these features to the horizontal reference. Finally, a series of convolutions and maxpooling operations are applied to the re-oriented feature space. In this sense, these convolutions and maxpooling operations act as feature extractors of the network (Fig. 3.1).

In this chapter, we propose using the scattering transform and steerable filters to generate the oriented feature space. Then, we use them as the first layer of the proposed network architectures. Later, in chapter 4 we introduce Gabor filters as an alternative to generating the oriented features. Furthermore, in chapter 4 we propose different alternatives for the feature extraction part of the feature stage (Fig. 3.1).

In general, these operations allow us to construct a Rotation Invariant Network (RIN). We refer to the different alternatives using the convention *RIN (oriented feature space + feature extraction)* to differentiate the used feature stage internal components during the rest of this manuscript. For example, for a rotation-invariant network using steerable filters as the oriented feature space and a custom convolution feature extraction, we use the notation RIN (steerable + CNN). For the sake of brevity, we do not include the commonly used opera-

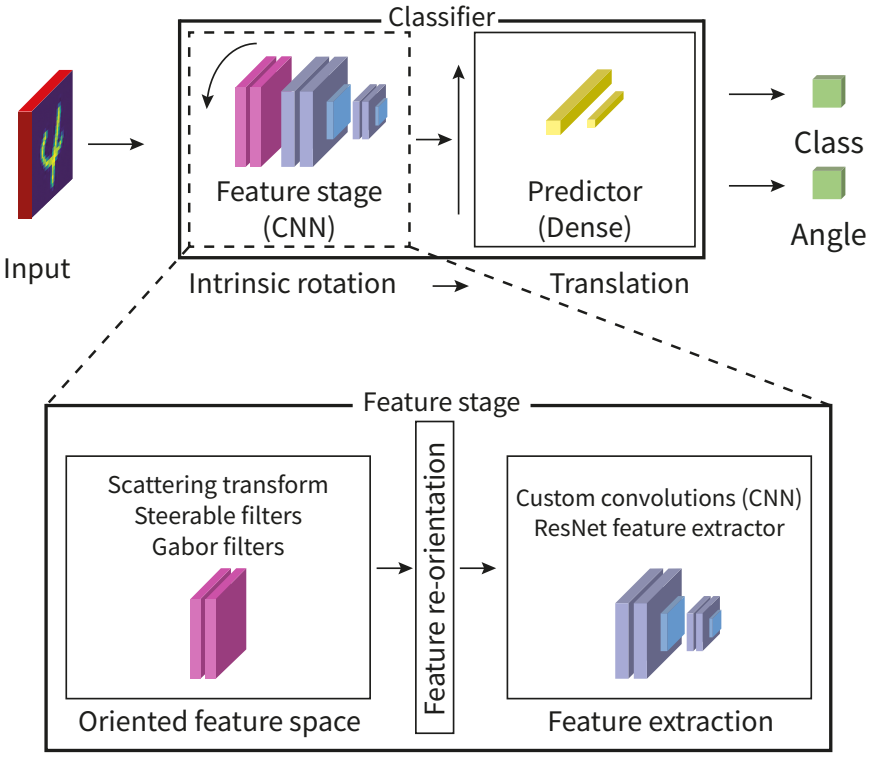


Figure 3.1 – Feature stage internal components.

tions, such as the feature re-orientation, in this notation as they are present in all the network architectures.

To complete this chapter, we present results for RIN (scattering + CNN) and RIN (steerable + CNN) on the MNIST dataset. Also, we present how we obtain the angle prediction as a byproduct of the architecture design.

3.1 MNIST Dataset and Variations

One of the several datasets used in state-of-the-art to validate rotation invariant properties of the neural networks is the MNIST dataset. The MNIST database of handwritten digits is divided into 60,000 training examples and 10,000 test examples (Figure 3.2). The digits have been size-normalized and centered in a fixed-size image of 28 x 28 pixels LeCun and Cortes [2010]. Usually, rotation invariance is tested by rotating the examples in the training and test set ; we argue this is equivalent to data augmentation and should be avoided. Nevertheless, we propose training with randomly oriented examples that allows us to compare directly with the state-of-the-art techniques. Also, we train the network with the original training set in the upright position and test the rotation invariance with randomly rotated test examples. This training and testing methodology allows to truly prove the rotation invariance properties by testing in orientations not seen by the network in the training phase.

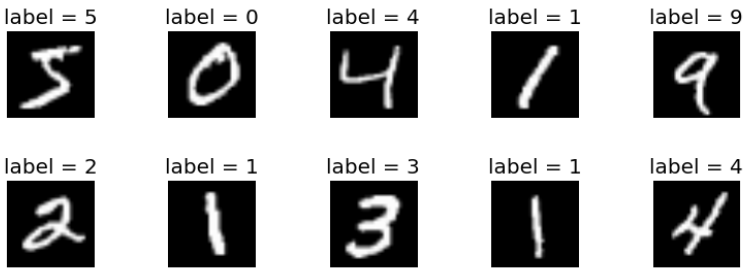


Figure 3.2 – MNIST dataset examples

3.1.1 Training Strategies

The dataset variations can then be divided in different approaches mostly based on the training sets variations. As we believe that rotation invariance properties are best evaluated on randomly rotated samples the validation set contains for all cases samples rotated by a random angle between $[0, 2\pi]$.

Upright Training (URT) For the sake of simplicity we define upright as the original orientation of the dataset. This means that for upright trainig set we use the original samples of the MNIST dataset. We keep the training set in the original upright position and the validation set is rotated by a random angle between $[0, 2\pi]$.

Rotated MNIST Dataset (RRT) (MNIST-rot) Rotated MNIST (MNIST-rot) is usually used by the state of the art approaches. To allow a direct comparison with the literature we present tests on this dataset. This dataset contains 12,000 training samples, and 58,000 testing samples randomly rotated between $[0, 2\pi]$.

In this chapter, we explore the possibility of endowing a Convolutional Neural Network with Rotation Invariant properties by adding a first layer with roto-translational properties. This first layer decomposes the input image in several oriented components ordered by orientation. We present two possible transformations: scattering transform and steerable filters. In Chapter 4 we also present the usage of Gabor filters as an alternative for the oriented decomposition of the first layer.

3.2 Scattering Transform

The scattering transform is one of the multiple existing forms of signal representation for classification, it builds an invariant and stable signal information along multiple paths. Bruna and Mallat (Bruna and Mallat [2013a]) developed a Convolutional Network using the scattering transformation that computes a translation invariant image presentation, which is stable to deformations and preserves high frequency information for classification. As mentioned in their work the scattering network can provide the very first layer of a Deep Convolutional Network.

Using the real part of the Morlet wavelet (Morlet et al. [1982]) described in Bruna and Mallat [2013a] we transform the input image into a set of oriented wavelets (Fig. 3.3). This transformation has a period of 180° and outputs

higher energy when the wavelet orientation is colinear with the edges of the input (33.75° and 146.75° for the letter X).

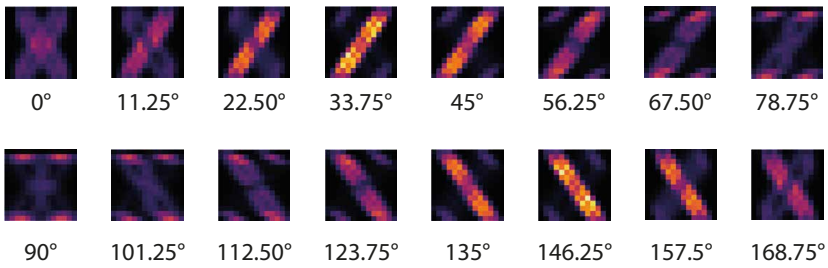


Figure 3.3 – Oriented-wavelet feature space for an input letter X. Product of the scattering transform with parameters M = 2, J = 1, L= 16.

One of the main properties of this oriented-wavelet feature space is the capability of mapping the angular separation between edges to a linear distance between the elements of the space. For example, the letter X contains a 68° degree and a 112° degree between the strokes, this angles are mapped as a linear distance of 6 and 10 respectively over the feature space. This characteristic allows the further layers of a convolutional neural network to use this as a feature on the classification task.

Another property of this transformation is that it presents roto-translational characteristics over the oriented-wavelet feature space. It means that there is a translation over the feature space that is covariant to the input rotation. This translation is proportional to the angular sampling and the magnitude of the input rotation. Using oriented features to generate it produces a feature space that contains rotation and translation components. Therefore, to obtain a translation feature space, we compensate the rotation component by negatively rotating each space element by the angle that generates it (Fig. 3.4). We refer further to this process as angular compensation (re-orientation) of the feature space.

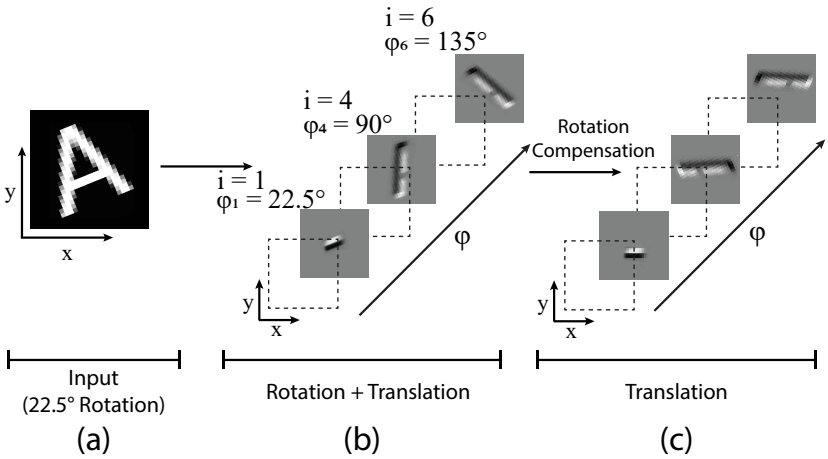


Figure 3.4 – Compensation of the rotation (re-orientation of the features). a) The input image with 22.5° of rotation, b) The feature space contains rotation and translation components, c) After compensating the rotation a translation feature space is obtained.

As a result of the oriented decomposition and the feature space's angular compensation, we obtain a translating feature space that contains all the possible discrete rotations of the input encoded as translations. As explained in

section 2.3, a cyclic convolutional predictor scans each translation and makes a prediction. To emulate the behavior of a cyclic convolutional predictor, we pad the feature space by itself, and then we apply a 3D convolution operation over the padded feature space. This 3D convolution operation is then a translating predictor that scans each one of the translations contained in the translating feature stage. The predictor outputs a higher probability in the translation corresponding to the upright orientation (as trained with upright examples) and its neighbours. It is important to consider that this is a shared weight predictor that adjusts its prediction capabilities in the way that the upright prediction is enhanced and the non-upright positions are learned to be ignored. This capability of the predictor to have a high output value only at translation corresponding to the upright position brings the possibility to obtain the angular rotation of the input by knowing the place in which the maximum output is located.

3.2.1 Scattering Wavelet

A wavelet transform computes the convolutions with dilated and rotated wavelets. Wavelets are localized waveforms and are thus stable to deformations, as opposed to Fourier sinusoidal waves. However, convolutions are translation covariant, not invariant (Bruna and Mallat [2013a]). The scattering transform builds nonlinear invariants from wavelet coefficients, with modulus and averaging pooling functions (Fig. 3.5).

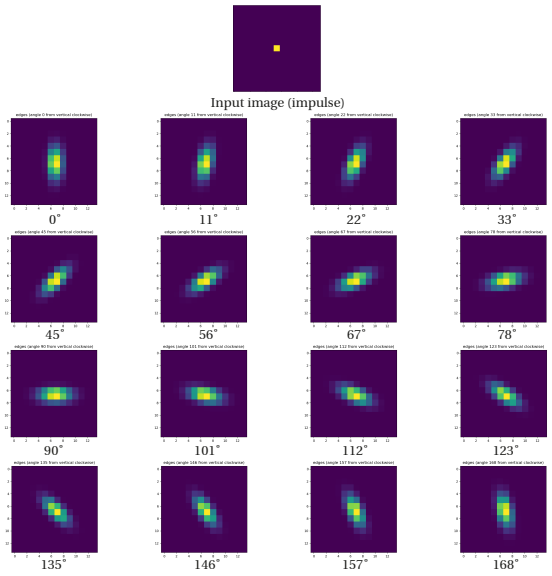


Figure 3.5 – Scattering wavelet response to impulse response. Oriented responses for each filter can be observed. (The angles below the figure are clockwise orientation with respect to the vertical.)

Scattering transforms build invariant, stable and informative representations through a non-linear, unitary transform, which delocalizes signal information into scattering decomposition paths. They are computed with a cascade of wavelet modulus operators, and correspond to a convolutional network where filter coefficients are given by a wavelet operator (Bruna and Mallat [2013a]).

Due to their invariance and stability properties, scattering operators linearize deformations (Bruna and Mallat [2011]). This linearization property can be

exploited to build linear generative classifiers in the scattering domain. When applied to stationary textures, scattering transforms provide new texture descriptors, incorporating high order moments which can discriminate non-Gaussian properties. As a result, state-of-the-art classification results are obtained on hand-written digit recognition and texture classification.

3.2.2 Rotation Invariant Network Based on Scattering Wavelets

Previously introduced wavelet transforms decompose the input image with a family of wavelets. The resulting representation outputs high-frequency components of the signal. The main drawback is that the resulting representation becomes translation covariant. Shifting the image also shifts the wavelet response. This behavior makes difficult the classification between translated images using only wavelet operators.

Bruna and Mallat [2013b] present the invariant scattering convolution networks to address the covariance problem. A wavelet scattering network computes a translation-invariant image representation, stable to deformations, and preserves high-frequency information for classification. It is based on the Wavelet scattering transform, which builds a signal representation with a redundant dictionary of Morlet wavelets.

As introduced by Bruna and Mallat [2013b], a scattering representation consists of order 0, 1, and 2 coefficients, composed of wavelets in different sequences. Multiple wavelets capture high-frequency structures of the image. In their work, the authors outline that paths up to length 2 are enough to extract almost all the frequency information of the data. This property is useful at the classification of texture images where an invariant representation against deformations such as translation and rotation is needed.

In comparison, in this work, we use the real part of the first order of the scattering transform to obtain an oriented representation of the input image. Then, we apply a series of convolutional layers and a translation predictor to obtain class and angular information. Compared to previous works (chapter 2), we use the equivariance to predict the angle of the input image.

As a result, we propose RIN (scattering + CNN), which can be observed in Figure 3.6.

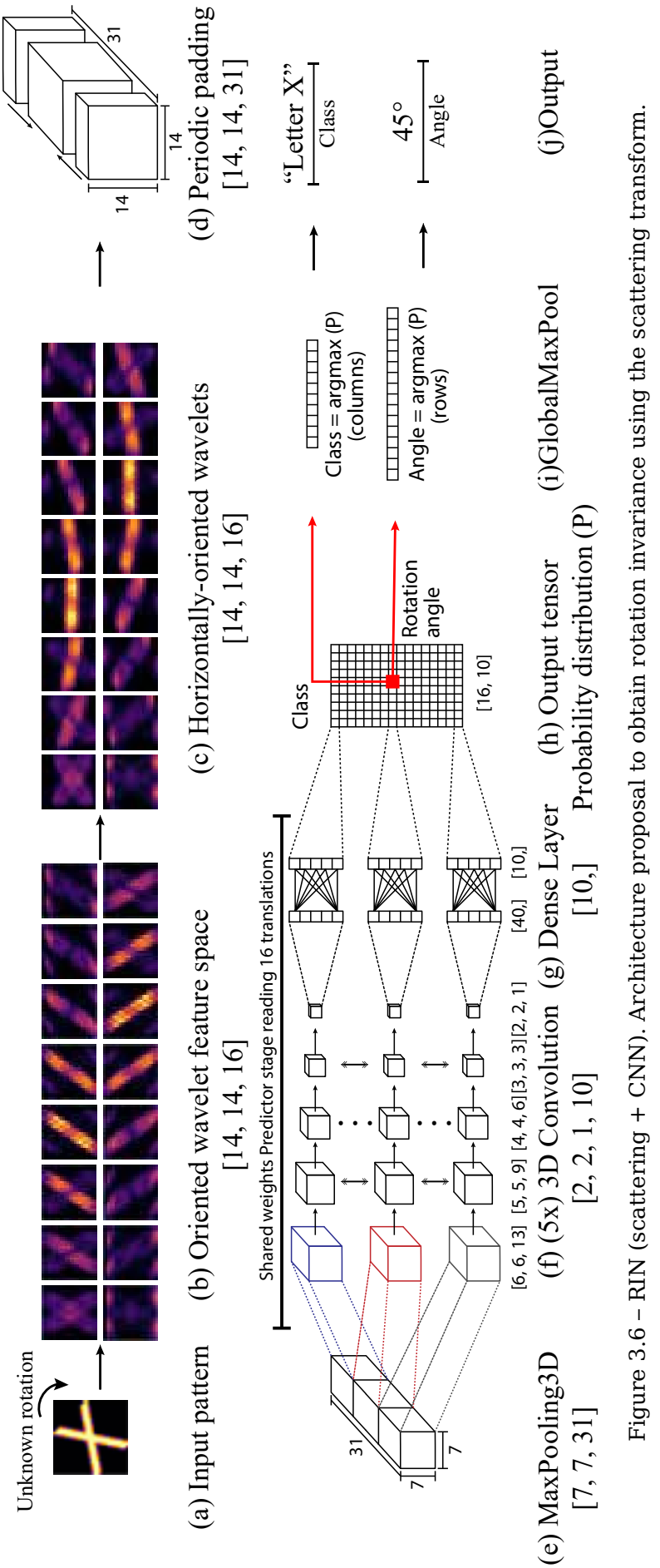


Figure 3.6 – RIN (scattering + CNN). Architecture proposal to obtain rotation invariance using the scattering transform.

In the feature stage, the input image (Figure 3.6a) is first transformed using the scattering operation and outputs an oriented wavelet feature space (Figure 3.6b). Then, we re-orient the feature space using a bilinear rotation. This rotation compensates each feature by the angle that generated it allowing us to have all the features aligned to the same horizontal reference (Figure 3.6c).

Then, we pad the feature space by itself (Figure 3.6d) and apply to the padded space a 3D convolutional predictor. Recall that, as explained in the previous section, this mimics the behavior of a cyclic convolution predictor.

The prediction stage starts with a maxpooling operation (Figure 3.6e) that allows refining the features obtained in the previous stage. The translating 3D predictor is formed by a series of 5 3D convolutions (Figure 3.6f). Following these series of 3D convolutions, a dense Layer (Figure 3.6g) transforms the extracted predictions into a one-out-of-many format with ten outputs (there are ten classes). The shared predictor stage outputs a one-out-of-many vector prediction for each translation contained in the wavelet feature stage. This prediction contains a higher value in the translation that corresponds to the upright orientation of the input image. The output predictions are condensed in a 2D output tensor (Figure 3.6h) containing as many rows as translations and as many columns as classes. This output tensor is then a probability distribution over two axes and contains one maximum value containing the predicted class (column index) and predicted rotation (row index). A globalmaxpooling operation (Figure 3.6i) applied to the row, and the column axis outputs the class and rotation prediction.

We discuss the results of this architecture applied to the MNIST dataset in section 3.4.

3.3 Steerable Filters

The second approach to obtain a roto-translational feature space is based on steerable filters (RIN (steerable + CNN)). Oriented filters have been deeply studied by Freeman and Adelson (Freeman and Adelson [1991]). They define the term *steerable filter* as a class of filters in which a filter of arbitrary orientation is synthesized as a linear combination of a set of *basis filters* (Figure 3.7).

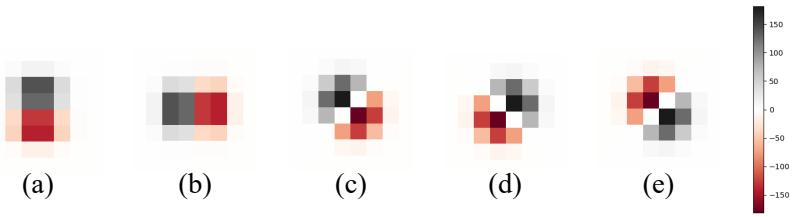


Figure 3.7 – Steerable filters. (a) s^{0° first derivative with respect to x (horizontal) of a Gaussian (b) s^{90° which is s^{0° rotated by 90°; (c) 45°, (d) 135°, (e) 225°, formed by the linear combination of (a) and (b).

The two-dimensional case of steerable filter found in Freeman's methodology can be used as a first layer of the Rotation Invariant Network (RIN). Consider the two-dimensional, Gaussian function S written in Cartesian coordinates x and

y:

$$S(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \tag{3.1}$$

Substituting $l = \frac{1}{2\sigma^2}$ in the Eq. 3.1 we can make a generalization of this filter. Let the parameters α and β be the shape parameters on the exponent and l the scaling parameter.

$$S(x, y, l, \alpha, \beta) = \frac{l}{\pi} e^{-l(\alpha x^2 + \beta y^2)} \tag{3.2}$$

Following Freeman's (Freeman and Adelson [1991]) methodology we calculate the first-order directional derivative of Eq. 3.2 in the x direction. Let the derivative of S be denoted by s.

$$s(x, y, l, \alpha, \beta)^{0^\circ} = \frac{\partial}{\partial x} \frac{l}{\pi} e^{-l(\alpha x^2 + \beta y^2)} = \frac{-2\alpha l^2 x}{\pi} e^{-l(\alpha x^2 + \beta y^2)} \tag{3.3}$$

Then the same function in y direction.

$$s(x, y, l, \alpha, \beta)^{90^\circ} = \frac{\partial}{\partial y} \frac{l}{\pi} e^{-l(\alpha x^2 + \beta y^2)} = \frac{-2\beta l^2 y}{\pi} e^{-l(\alpha x^2 + \beta y^2)} \tag{3.4}$$

A filter with any arbitrary orientation φ can be calculated by the linear combination of s^{0° and s^{90° using:

$$s^{\varphi} = cos(\varphi) s^{0^\circ} + sin(\varphi) s^{90^\circ} \tag{3.5}$$

s^{0° and s^{90° are the *basis filters* and the terms (l, α, β) are shape parameter of the filter.

Following this methodology we create an ensemble of stterable filters ordered with the increasing orientation angle φ_i. These parameters will be optimized by training to fit the data.

3.3.1 Learning Steerable Filters

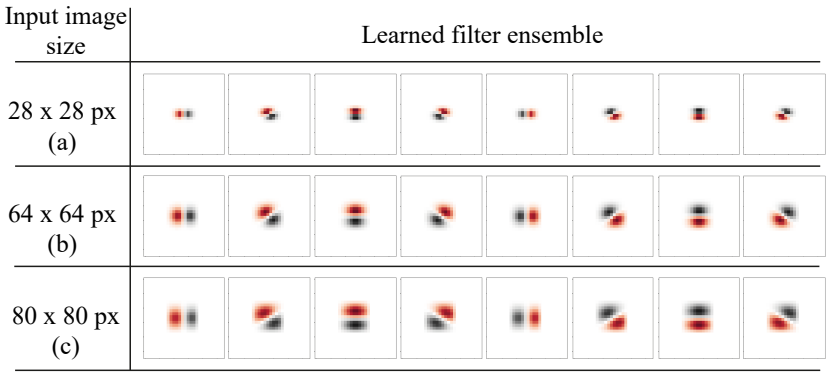


Figure 3.8 – Learned steerable filters for different size of images. The filter increases in width and height following the input image.

RIN (steerable + CNN) uses a single basis filter as first layer to obtain an oriented feature space. The number of basis filters can be increased but the tests did not show any improvement on the accuracy when using more than one. This

is certainly due to the simplicity of the MNIST dataset consisting of exclusively of pencil strokes with a constant width that a single optimized filter can be activated with.

To test the trainability of the filters, we observed the change of the scaling parameter l that affects the size of the filter. First, we trained the network with three different input image sizes: 28 x 28, 64 x 64, and 80 x 80 pixels, then we plot the steerable filter that corresponds to the best accuracy. As observed in Figure 3.8, the steerable size changed following the size of the input image. This change in the size means that the network can learn and adapt the basis filter for the input images. Also, for the biggest case (80 x 80 px), we observed that the filter support crops the basis filter. The filter support can be increased to avoid being cropped when using larger images.

To complete, we can explain the training mechanism that optimizes the parameter l. First, we generate a kernel window using the steerable equations presented previously. This kernel window then is dependent on the values l, α, β. α and β are then the shape (height and width) parameters of the filter. After several experiments, we observed that these two values converged in equal values. This convergence is mainly due to the filter needing to be circular ($\alpha \approx \beta$) to allow the steering of the filter. Consequently, we only kept l as a parameter.

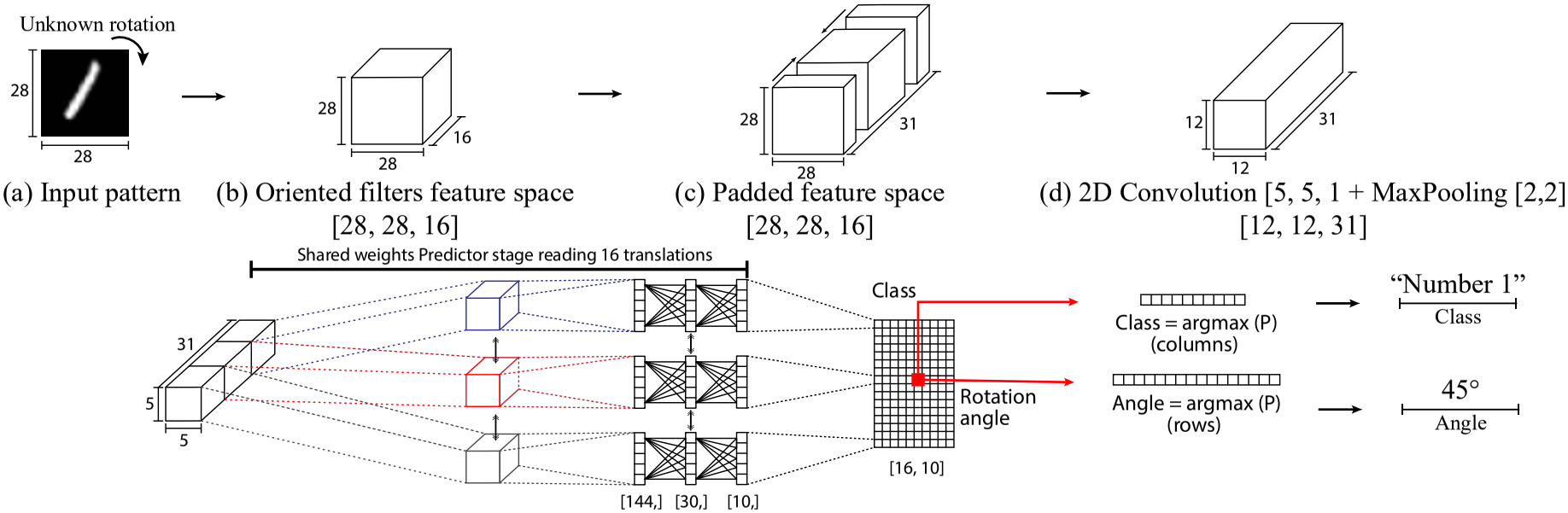
The kernel window produced above is then dependent on l, and we can select the support size of the window as a parameter. The next step was to rotate the filter using eq. 3.5. We generate as many kernel windows as $d\Phi$, each rotated by $d\Phi/360°$, and then we encoded them as convolutional filters. We then created a custom convolutional layer that convolves the input image with these filters containing the rotated steerable filters.

The custom convolution layer then outputs the activation of the input image convolved with the oriented filters generated with the steerable equations. To train this custom layer, we used the Layer class from the Tensorflow platform. The main advantage of this class is the automatic differentiation that automatically updates the parameter l.

3.3.2 Rotation Invariant Network Based on Steerable Filters

Similar to RIN (scattering + CNN), RIN (steerable + CNN) contains a feature stage and a predictor stage. The feature stage is responsible of building the roto-translational feature space (Figure 3.9a-d). The predictor stage contains the 3D convolutional predictor scanning over the feature space and generates a probability distribution output (Figure 3.9e-j).

Analog to RIN (scattering + CNN), the steerable filters in the oriented ensemble are angularly compensated by the angle that generates each filter. Also, the oriented filter feature space (Figure 3.9b) is padded by itself (Figure 3.9c) to obtain a space that contains each one of the possible discrete translations of the input pattern. A series of convolutional predictors are applied to the padded feature space (Figure 3.9d-e). These convolutional predictors contain convolutional and maxpooling operations. Also, these predictors are applied to the height and width dimensions and preserve the angular dimension of the feature space.



(a) Input pattern

(b) Oriented filters feature space [28, 28, 16]

(c) Padded feature space [28, 28, 16]

(d) 2D Convolution [5, 5, 1 + MaxPooling [2,2] [12, 12, 31]

(e) 2D Convolution [5, 5, 1] MaxPooling [2, 2] [5, 5, 31]

(f) 3D Convolution [3, 3, 16] [3, 3, 16]

(g) Dense Layer [10,]

(h) Output tensor Probability distribution (P)

(i) GlobalMaxPool

(j) Output

Figure 3.9 – RIN (steerable + CNN). Architecture using the steerable filters to generate the oriented features. Oriented filters and re-rientation are contained in the first layer of the network.

A 3D convolution (Figure 3.9f) applied to this space is equivalent to applying a cyclic convolutional predictor over the oriented filter feature space. Recall that this 3D convolution predictor reads each one of the possible translations of the feature space and outputs a higher probability for the translation that contains the upright orientation using a dense layer predictor (Figure 3.9g) ; these results are stored on a probability distribution of size (number of classes K, orientations N) (Figure 3.9h). Next, a globalmaxpool (Figure 3.9i) is applied to the probability distribution on the row and the columns to obtain the class and angle index.

3.4 Results

We evaluate the rotation invariant properties of the previously proposed networks on three variations of the MNIST dataset. The angular sampling is $\Phi = 16$ for the main tests and a particular test with varying Φ. The training is done by using the original class labels in the one-out-of-many format, angular labels are not provided as part of the training phase.

For the upright training (URT) we train the network as usual state of the art approaches, this means training the network with the original upright 60,000 examples. To validate the rotation invariant properties of the network we rotate the validation set of 10,000 examples by an unknown random angle between [0, 2π].

Randomly rotated training (RRT) using MNIST-rot is meant to test the non-supervised angular prediction. This means to train on randomly oriented examples and make the prediction without angular labeling available in the training stage. All the training and validation examples of the original MNIST are rotated by an unknown random angle between [0, 2π].

Usually the rotated MNIST dataset (MNIST-rot) (Worrall [2017]) is used by several state of the art approaches. We present results with this modification of the MNIST dataset to allow a direct comparison with the current approaches. This dataset contains 12,000 training and 58,000 testings examples randomly rotated between [0, 2π].

3.4.1 Rotation Angle Prediction

As a second product of the methodology, the network can predict the angle of the input image without angular labels on the training dataset. To achieve this, we obtain the angular information from the probability distribution (P) (section 2.3). To recall, the predictor stores the prediction of each translation of the feature space in this table. The predictor outputs a high probability of the class when the examples is in the upright position (when trained on upright examples) and near-zero values elsewhere.

Due to slight variations of the training data (not all training examples are in perfect upright orientation), the predictor becomes tolerant to some angular misorientation. We refer to tolerance as classifying the object correctly despite some angular misorientations. As a consequence of the predictor tolerance, higher probability values also appear on the adjacent orientations of the upright position up to $\pm d\Phi$.

In Figure 3.10, we can observe how the probability distribution (P) behaves with different orientations of the same example (number 7). The red cells represent the maximum values of the table and the white near-zero values. When the input image is rotated, we observe the maximum values of the table translating from up to down. Each translation corresponds to an angle increment of $d\Phi$.

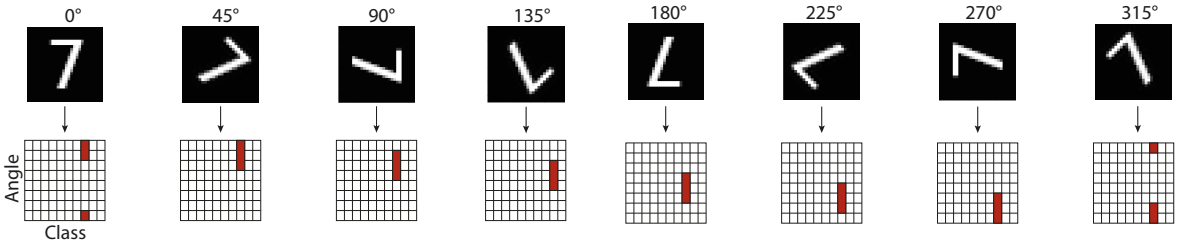


Figure 3.10 – Equivariant translation with respect to the input rotation. In red higher values of the table.

The tolerance acquired by the predictor as a product of the misorientations of the dataset allows the network to map the rotation of the input (when trained with randomly oriented samples) to a periodic space that assigns one row to each possible discrete angle of the network. This property can be seen as a self-organizing behavior mapping consecutive angular values as consecutive rows in the table. The output space has the same behavior when trained with upright oriented and randomly rotated datasets. This leads to generating a linear relationship from the consecutive angles without any reference existing on the angular rotation input space.

On the datasets that do not contain an upright reference (e.g., for the plankton dataset (Orenstein et al. [2015]), the upright position does not exist), the network randomly selects a row corresponding to a virtual right orientation. This random selection means that the network selects a random baseline for each class as virtual upright orientation and then maps the consecutive angular values from this base.

One possible application for this is the automatic reorientation of randomly oriented samples (Figure 3.11). In this application, the network would select a random row for each class's virtual upright orientation. Then, the rotation angle can be calculated by the modulo of this randomly chosen baseline upright orientation. It is important to notice that while the network randomly selects the virtual upright position row, this position is constant per class. This property allows to re-orient the examples in this class to this virtual position.

3.4.2 Rotation Invariant Classification

Rotation-invariance is a property that describes the capability of the network to obtain the correct class prediction despite the orientation of the example. First, we test this property by training the network with upright examples (RRT) and validating on randomly rotated samples (MNIST-rot). Then, we compare the results of state of the art architectures with the MNIST-rot dataset. At the end,

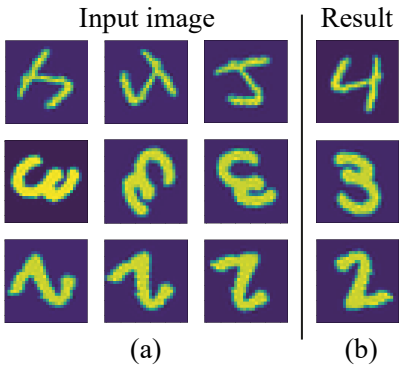


Figure 3.11 – Automatic reorientation of the MNIST randomly oriented examples usign RIN (steerable + CNN).

we outline some global properties of the network and behavior over different inputs.

Angular sampling tests. One of the particular parameters we test is the number of tested rotations N. We change the value of N from 2 to 24 to show the correlation between N and accuracy (Fig. 3.12). Having $N > 4$ (as most of the state-of-the-art architectures) is helpful for the model's accuracy. Also, we can observe that the accuracy does not longer significantly increases for values over $N = 14$ – both the upright and the randomly oriented training present this behavior.

Fig. 3.12 shows that the number of rotations N do not has an impact when trained on RRT. This phenomenon is mostly due to RRT being similar to data augmentation and the predictor observing different orientations of each example. In the same Fig. 3.12 we observe that the number of rotations heavily affects the accuracy when trained on URT. This behavior can be explained as the predictor observing upright-oriented examples and not having enough angular filters to describe the angular information between edges. For example, when $N = 2$ there exist 2 filters and 2 possible translations of the feature space with $d\Phi = 90°$.

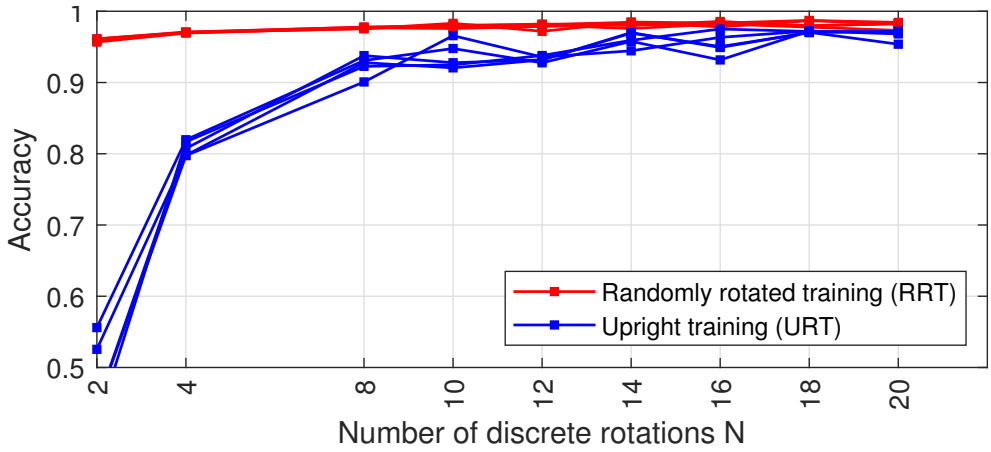


Figure 3.12 – Effect of N on accuracy for URT and RRT. With upright training the prediction accuracy slightly decreases on lower N values.

Training parameter l test. The l parameter of the basis filters defines the width of the edge detector filter. Lower values of l indicate a wider edge detector.

Fig. 3.13 shows the correlation between the size of the input image and the l value. For larger input images, the value of l decreases, and the filters grow larger. The change in the filters can be observed in Fig. 3.8. This proves the ability of the network to adapt the edge detector to the size of the input. This behavior is consistent for different values of N demonstrating the training capability of this parameter.

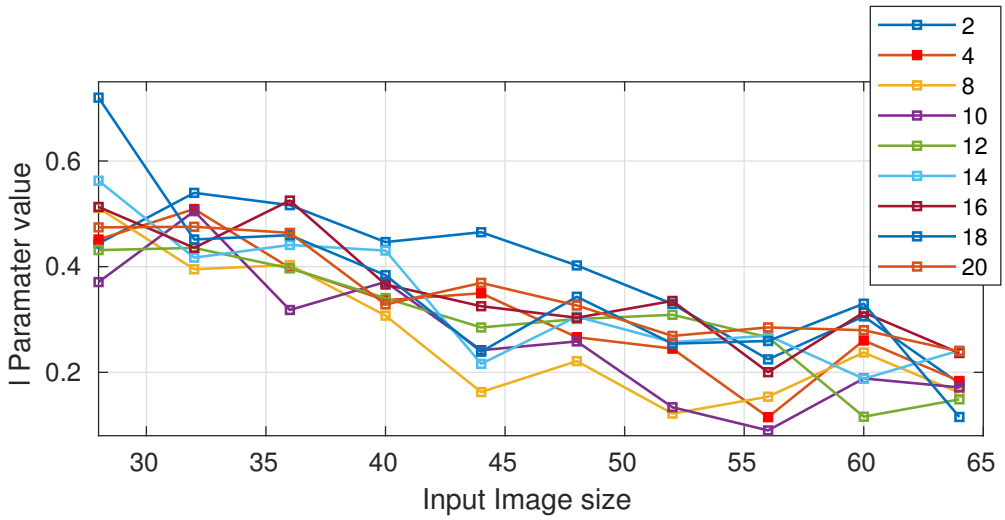


Figure 3.13 – Effect of N on learned value l. Learned parameter l value is inversely proportional to the size of the input pattern. This behavior is consistent for N = 2 to 20.

Comparison with the State of the Art

Table 3.1 – Obtained error rate with URT

Method	Error rate	# parameters
ORN-8(ORPooling)Zhou et al. [2017]	16.67%	397k
ORN-8(ORAlign)Zhou et al. [2017]	16.24%	969k
(RP_RF_1) Follmann and Bottger [2018]	19.85%	130k
(RP_RF_1_32)* Follmann and Bottger [2018]	12.20%	1M
RotDCF (60 degrees) Gao et al. [2019]	17.64%	760k
Spherical CNN Cohen et al. [2018]	6.00%	68k
Icosahedral CNN Cohen et al. [2019]	30.01%	n.c.
RI-LBCNNs Shin and Yun [2019]	25.77%	390k
RIN (scattering + CNN) Rodriguez Salas et al. [2019]	17.21%	7k
RIN (steerable + CNN) Rodriguez Salas et al. [2021a]	2.05%	42k

RIN (scattering + CNN) reaches state-of-the-art results while RIN (steerable + CNN) outperforms previous works compared to the communicated error rate. For all the reported results, we used 16 sampled orientations. For the training with upright position (URT) (Table 3.1), (scattering + CNN) has a higher error rate while having the lowest number of trainable parameters of all approaches.

Also, for URT, RIN (steerable + CNN) contains a higher number of parameters but outperforms the existing state-of-the-art approaches. It is important to mention that most of the state-of-the-art approaches shown on the table use other techniques to improve their accuracy. While Spherical CNN achieves a low error rate, it comes with high complexity and processing power to obtain these results. Furthermore, other state-of-the-art works achieve rotation invariance by losing the equivariance property. Thus, they can not make an unsupervised angular prediction.

Table 3.2 – Obtained error rate with MNIST-rot

Method	**Error rate**	**# parameters**
Harmonic Networks Worrall et al. [2017]	1.69%	33k
TI-Pooling Laptev et al. [2016]	1.26%	n.c.
R. Eq. VFN Marcos et al. [2017]	1.09%	100k
ORN Zhou et al. [2017]	1.37%	397k
SFCNNs Weiler et al. [2018]	0.714%	n.c.
RP_RF_1* Follmann and Bottger [2018]	3.51%	130k
RI-LBCNNs Shin and Yun [2019]	1.36%	390k
GCNs Luan et al. [2018]	1.10%	1.86M
RIN (scattering + CNN) Rodriguez Salas et al. [2019]	2.69%	7k
RIN (steerable + CNN) Rodriguez Salas et al. [2021a]	0.93%	42k

For the training with rotMNIST (Table 3.2), the results reach state of the art error rate. In this type of training RIN (scattering + CNN) error rate improves with respect to the URT training; also, it remains as the lowest number of parameters approach. The RIN (steerable + CNN) reaches less than 1% error rate while keeping a lower number of parameters. The closest approach Rotation Equivariant VFN uses more than the double of parameters and GCNs almost 2 Million parameters to achieve similar results than the presented in this work.

All of the results obtained for RIN (steerable + CNN) use a single learnable basis filter rotated N time. Also, it preserves the equivariant properties of the filters allowing the network to predict the angular rotation of the input. In this sense, this is the first approach to our knowledge that uses the roto-translational properties of the space to make an angular prediction of the input image.

3.5 Conclusion

In this chapter, we presented two convolutional architectures based on the scattering transform (RIN (scattering + CNN)) and one based on steerable filters (RIN (steerable + CNN)). The main objective of these architectures is to tackle the rotation invariant and angular prediction that exists on several deep learning applications. To achieve this objective, we use the roto-translational feature space that transforms into translation the rotation of the input image and a shared weight translational predictor that scans every possible discrete translation (hence discrete rotation) of the input image.

We tested both networks on variations of the MNIST dataset achieving state-of-the-art results for the randomly rotated training and outperforming state-of-the-art accuracy and number of parameters.

First, we showed results using RIN (scattering + CNN). This approach reached state-of-the-art results. Its main drawback was the lack of trainability on its filters. The main consequence is a high error rate when trained with upright examples and tested on random rotated examples.

Also, we presented RIN (steerable + CNN) with trainable basis filters as the first layer. This approach tries to compensate for the lack of trainability of the previous one by using filters that adapt to the input data size characteristics. In this chapter, we demonstrated the trainability of these filters and how their size changes following the input image size. While this approach outperformed state-of-the-art techniques, it comes with one major drawback.

Steerable filters have low flexibility as, by definition, they need to be circular. Consequently, the network can not train the filter to have better angular or frequency selectivity limiting with this the adaptation to the input data. To solve this, in the next chapter, we propose using Gabor filters that have increased flexibility, allowing them to be angular and frequency selective.

Chapter 4

Rotation Invariant Networks on Complex Datasets

In the previous chapter, we presented a rotation-invariant architecture (RIN) that reaches state-of-the-art results on the MNIST dataset but with the advantage of being trained only on upright oriented and validated on randomly rotated samples. This architecture used steerable filters to obtain a roto-translational feature space followed by a shared weight predictor.

RIN (steerable + CNN) architecture reached state-of-the-art accuracy on this simple dataset because it comprises simple images containing an almost identical pencil stroke width to write the numbers. In this sense, using one basis filter was enough to obtain these results. One of the main limitations of steerable filters is the circular shape of the filter. To rotate, the width and height of the filter must be identical. Consequently, the angular selectivity of the filter is limited (section 3.3.1).

To improve the descriptive potential of the filters in RIN (steerable + CNN), we propose the usage of Gabor filters. Gabor filters are linear filters generally used to discriminate textures with different frequencies. 2D Gabor filters discriminate a specific frequency in a specific orientation (Daugman [1985]) based on the Gabor wavelet (Gabor [1946]). Hence, we can use them as filters to obtain a roto-translational feature space with better angular and frequency selectivity than the steerable filters.

This chapter introduces the Gabor layer based on trainable Gabor filters, indicates the parameters we make trainable, and shows their usage on classification tasks. We propose a CNN architecture using Gabor filters to generate a roto-translational feature space RIN (Gabor + CNN).

First, we test RIN (Gabor + CNN) on the MNIST dataset to compare them directly with our previous approaches RIN (scattering + CNN) and RIN (steerable + CNN). Then, we apply them to the CIFAR dataset that contains complex features. Finally, we propose an alternative to the CNN inside the feature extraction. Recall that after the roto-translational space, we plug several convolutions to act as feature extractors. To differentiate between alternatives, we refer to the previously presented convolutional feature extractor as RIN (Gabor + CNN) and the alternatives as RIN (Gabor + alternative).

4.1 Gabor Filters

Gabor filters have been used extensively in the literature in numerous applications (e.g., character recognition (Singh et al. [2012]), facial features extraction (Zadeh et al. [2019]), fingerprint recognition (Lee and Wang [1999]), object detection (Jain et al. [1997])). The most important properties are related to invariance to illumination, rotation, scale, and translation. Commonly, researchers and developers used these filters in banks arranged by spectral and temporal modulation frequencies (Schädler et al. [2012]). These filter banks are usually designed not to overlap frequencies, scales, or orientations.

In this work, we focus on four parameters of the Gabor filters: frequency (F), rotation (θ), frequency selectivity ajdustment (σ_x) and angular selectivity adjustment (σ_y). In this section, we highlight how we use these parameters to generate an oriented feature space. Then, we describe the parameters of these filters and the oriented feature space obtaining.

A 2D Gabor filter can be defined as an oriented sine (imaginary part of the filter) or cosine (real part) modulated by a 2D Gaussian function. The shape of the Gabor function is defined by the frequency, the orientation of the sinusoid and the scale of the Gaussian function. It offers a higher degree of liberty compared to steerable filters, especially in the sense that the angular and frequency selectivity can be learned independently. With this methodology we can then generate a set of rotating wavelets that can act as filters.

Following Gabor's filters methodology (Gabor [1946]), consider the rotation matrix in 2D space applied to a kernel pixel (x, y):

$$\begin{bmatrix} cos\varphi & -sin\varphi \\ sin\varphi & cos\varphi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos(\varphi) - ysin(\varphi) \\ ycos(\varphi) + xsin(\varphi) \end{bmatrix} \tag{4.1}$$

Then, the new location for the pixel is $(x_\varphi,\ y_\varphi)$:

$$\begin{cases} x_\varphi = xcos(\varphi) - ysin(\varphi) \\ y_\varphi = xsin(\varphi) + ycos(\varphi) \end{cases} \tag{4.2}$$

To adjust the frequency and angular selectivity, we divide this location by the magnitude σ_x and σ_y, respectively. Then, r_1 becomes the frequency selectivity adjustment value and r_2 angular selectivity adjustment value:

$$r_1 = \frac{x_\varphi}{\sigma_x};\ \ r_2 = \frac{y_\varphi}{\sigma_y} \tag{4.3}$$

These two parameters can then be applied to the Gabor function (G) and obtain the real (G_re) and imaginary (G_i) components as:

$$G_{re}(\varphi, \sigma_x, \sigma_y, F, x, y) = e^{\frac{r1^2+r2^2}{2}} cos(2\pi F x_\varphi) \tag{4.4}$$

$$G_i(\varphi, \sigma_x, \sigma_y, F, x, y) = e^{\frac{r1^2+r2^2}{2}} sin(2\pi F x_\varphi) \tag{4.5}$$

To generate the kernel windows then we use the (Eq. 4.4) and (Eq. 4.5). Where $(x,\ y)$ correspond to the pixel position on the kernel window. Parameters $(\varphi, \sigma_x, \sigma_y, F)$ are then variables that affect the filter shape (Fig. 4.1).

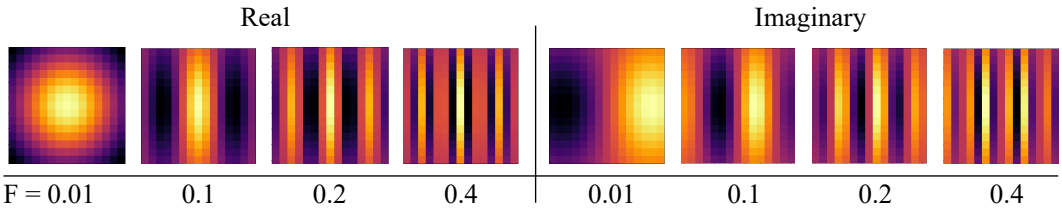


Figure 4.1 – Gabor filters examples. Real and Imaginary parts of the Gabor filter ensemble for different frequencies using Eq. 4.4 and Eq. 4.5 with $\sigma_x = 2, \sigma_y = 2, \varphi = 0$

Recall that the main objective is to use these equations to generate a roto-translational feature space (chapter 2). First, we create an oriented filter ensemble. Then, we re-orient the features to the horizontal reference. After this, we have a roto-translational feature space to be used by the translating predictor.

Creating a roto-translational feature space based on Gabor filters is then straightforward. First, we generate a set of basis filters (real and imaginary) using Eq. 4.4 and Eq. 4.5. Examples of these filters can be observed in Fig. 4.1, and the parameters are discussed in the next section. The generated filters have a vertical orientation ($\varphi = 0$). Then, we make the parameters $\sigma_x, \sigma_y, \mathrm{F}$ trainable parameters to be updated during the training stage of the network. Making these parameters trainable follows the same methodology explained in section 3.3.1 but with Gabor equations instead of steerable ones.

It is important to note that the values $\sigma_x, \sigma_y, \mathrm{F}$ are independent for real and the imaginary filters. While, by definition, Gabor filters imaginary and real parts are linearly dependent, we use them separated. It means that the real filter can have a different frequency, angular selectivity, or frequency selectivity than the imaginary. The motivation behind this choice comes from the intuition that a higher number of filters usually results in higher accuracy. Consequently, having these filters (real and imaginary) separated gives them the freedom to better adapt to the features. Furthermore, we did several experiments using the same $\sigma_x, \sigma_y, \mathrm{F}$ values for both basis filters (real and imaginary). These experiments showed an increased accuracy when the values were independent than using the same for both.

After generating the basis filters, we create a list of orientations φ_i where $i = 0, \ldots, \mathrm{N}$. This operation generates two oriented filter ensembles ($\mathrm{G}_{re}^{\varphi_i}$, $\mathrm{G}_{i}^{\varphi_i}$) with as many filters as orientations N.

Finally, these oriented filter ensembles become the kernels of the convolution operation. When convolved with the input, they generate an oriented decomposition of the input image. Then, they are re-oriented to the horizontal reference to obtain the roto-translational feature space. These two operations, convolution with the Gabor trainable filters and re-orientation of the oriented components, become the Gabor layer. This layer accepts as hyperparameters the size of the kernel support, the number of orientations N, number of basis filters (although in this work, we use a single pair of basis filters), and activation.

We present experiments and results of the Gabor layer as a roto-translational feature space generator (in the first layer of RIN (Gabor + CNN)) in the sections 4.2 and 4.3. To better understand the Gabor filter shape parameters ($\varphi, \sigma_x, \sigma_y, \mathrm{F}$)

and how each one affects the filter shape, we outline them in the following subsection.

4.1.1 Gabor Filter Parameter Description

As outlined previously, parameters $\varphi, \sigma_x, \sigma_y, \mathrm{F}$ change the Gabor filter shape. We use the first one (φ) to generate the N orientations of the oriented filter ensemble. The last three parameters shape the filter and become trainable parameters of the Gabor layer.

This chapter highlights how these parameters affect the filter and the expected behavior in training and inference for each one. Also, we present figures to understand the changing shapes of the filters visually.

Frequency: The frequency parameter allows the filter to adapt to different existing frequencies in the input image (Fig. 4.2). For example, on datasets containing low-frequency information such as pencil strokes or lines, the Gabor filter would converge towards a low-frequency shape ($\mathrm{F} \approx 0.1$). On the contrary, if the images contain high-frequency components (i.e., textures), the frequency converges towards high-frequency values ($\mathrm{F} \to .5$). In this sense, once trained, the filter can discriminate features based on the frequency by filtering them, or keeping them for the subsequent layers.

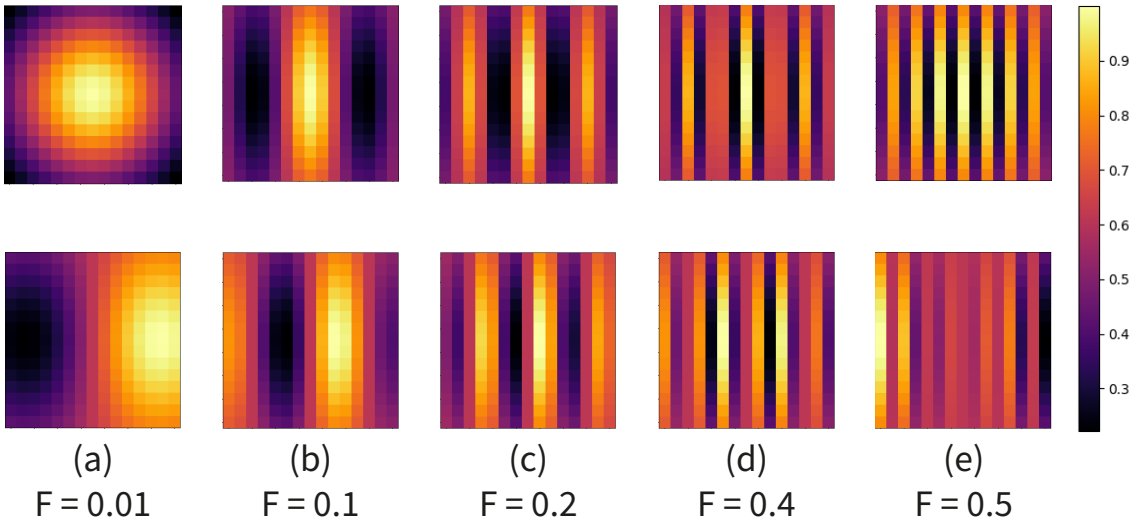


Figure 4.2 – Gabor filters for different frequency (F) values ($\theta = 0, \sigma_x = 2, \sigma_y = 2$). First row real part of the filter; second row imaginary part of the filter.

Frequency selectivity σ_x: This parameter changes the horizontal aperture of the filters. Hence, it allows to accept or deny more frequencies based on the wide of the filter. Notice in Fig. 4.3 how a higher value allows more period of the filter to be included into the kernel, narrowing thus the filter pass band.

Angular selectivity σ_y: This parameter changes the vertical aperture of the filter (Fig. 4.4). Thus, it becomes thinner while keeping frequency information. It learns to discriminate features based on the width of the features. For example, images such as pencil strokes train it to select the width of the pencil stroke.

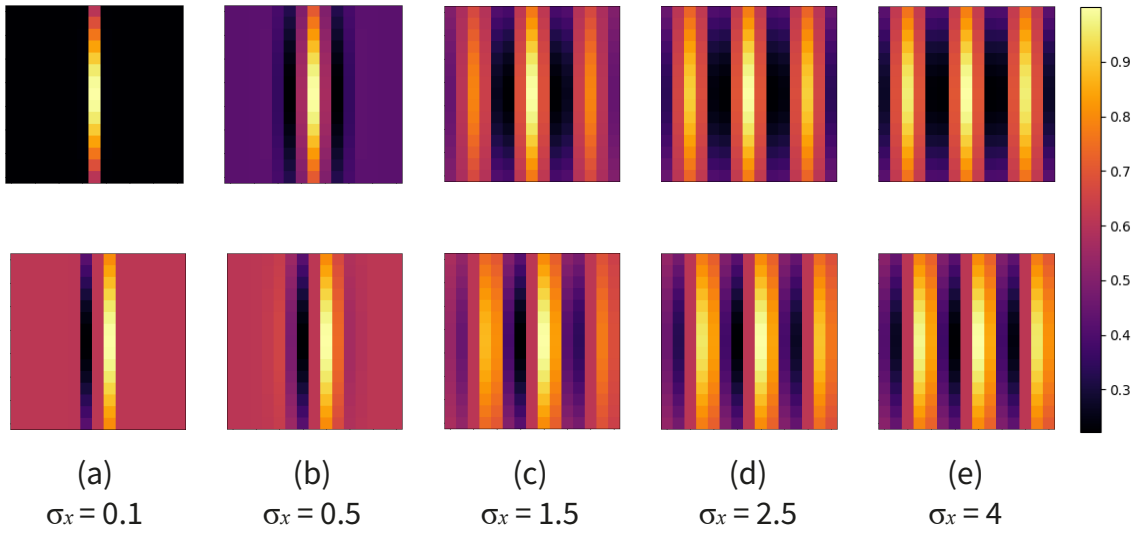


Figure 4.3 – Gabor filters for different (σ_x) values ($\varphi = 0, F = 0.2, \sigma_y = 2$). First row real part of the filter; second row imaginary part of the filter.

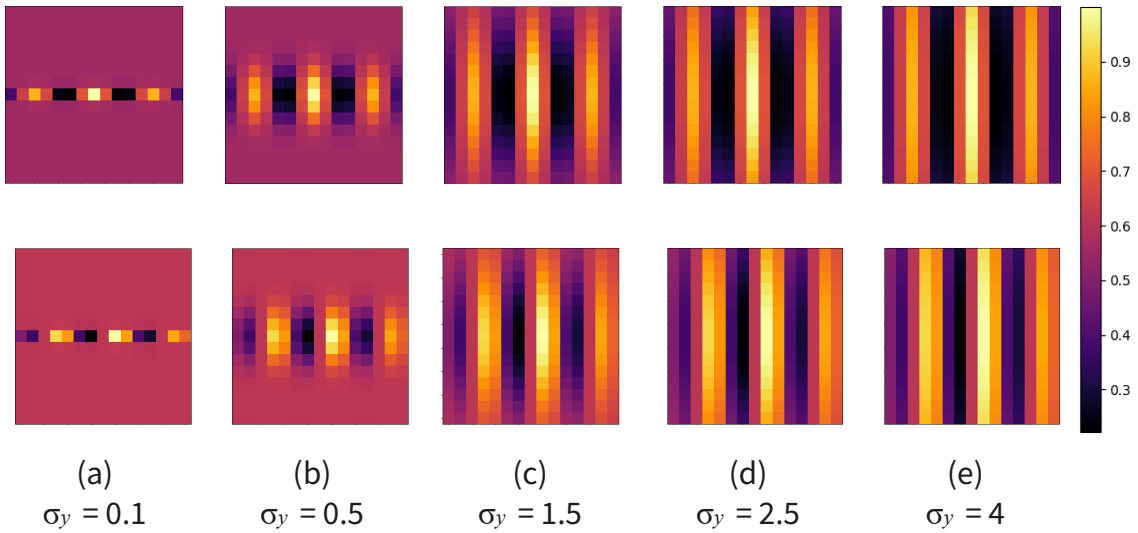


Figure 4.4 – Gabor filters for different (σ_y) values ($\varphi = 0, F = 0.2, \sigma_x = 2$). First row real part of the filter; second row imaginary part of the filter.

Rotation φ: Finally, the parameter φ allows the filter to be rotated to a fixed orientation (Fig. 4.5). The network does not train this parameter. Instead, it is used as a rotation parameter to obtain the oriented feature space. As explained previously, an index list of $i = 0, \ldots, \mathrm{N}$ generates an angle set φ_i with a constant rotation increment. Then, φ_i selects the number of orientations and angular orientations of the filter. Recall that N is a hyperparameter of the Gabor layer.

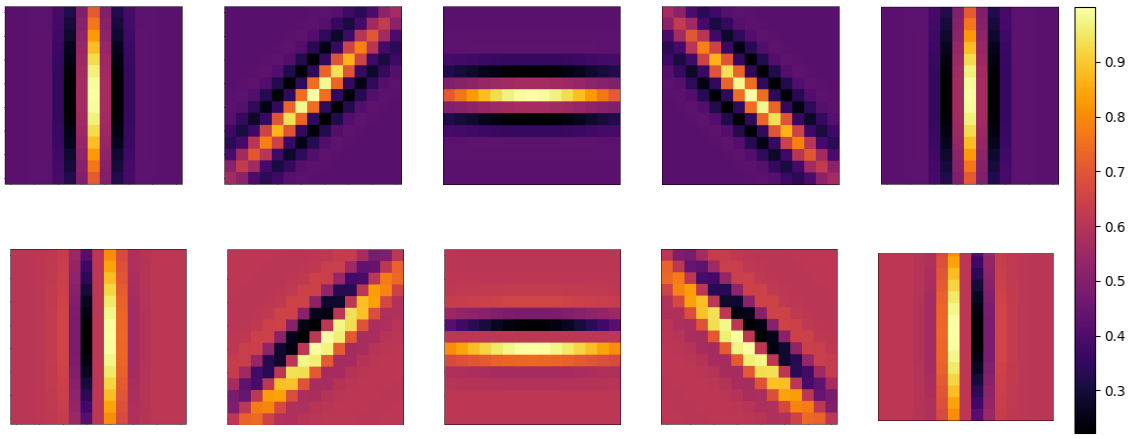


Figure 4.5 – Gabor filters for different (φ) values ($\mathrm{F} = 0.2, \sigma_x = 2, \sigma_y = 2$). First row real part of the filter; second row imaginary part of the filter.

To conclude, we can observe that these filters are more flexible than our previous approaches (scattering, steerable). Gabor filters can adapt their frequency, frequency selectivity, and angular selectivity to the training dataset. To directly compare these proposed trainable Gabor filters with our previous approaches, we test the Gabor layer as a roto-translational feature space generator in the RIN (Gabor + CNN) architecture using the MNIST dataset.

4.2 Gabor Filters as Feature Extraction on Simple Datasets

As explained previously, Gabor filters have several advantages compared to our previous filters used to generate the oriented decomposition of the input. These filters can learn the image frequency of the input image. Also, they can learn to become angular and frequency selective to discriminate the oriented features. In the previous section, we introduced the Gabor layer as the concatenation of two operations: the convolution operation (between the input and the oriented filters) and the re-orientation of the feature space.

Again, we start with the MNIST dataset as the most straightforward problem to validate the Gabor layer as a roto-translational feature space generator. We insist on using the MNIST to validate the layer because it also allows us to directly

compare with our previous proposals RIN (scattering + CNN) and RIN (steerable + CNN)

In this sense, RIN (Gabor + CNN) is similar to the previously introduced steerable network. We use the Gabor layer as a substitute for the oriented filters space presented on the rotation invariant network based on steerable filters (section 3.3.2). It means that the Gabor layer becomes an oriented representation mapping layer that outputs the roto-translational feature space (Fig. 4.6a).

Recall that the Gabor layer contains an oriented feature ensemble composed of real and imaginary filters. It means that for a kernel size of (width, height) and 16 orientations (N = 16), the filter shape is [m, n, 16, 2] when using one (complex) basis filter. MNIST input image dimension is [28, 28, 1].

To obtain the roto-translational feature space, we first convolve the real part of the oriented filter ensemble with the input (input: [28, 28, 1], filters: [m, n, 16, 1]) and let the N = 16 orientations act as filters of the convolution. Then, we calculate the convolution of the input image with the imaginary part of the oriented filter ensemble. Consequently, we have two ensembles of shape [m, n, 16, 1] due to the operations. Then, we concatenate (on the last axis) the two ensembles (real and imaginary) to obtain a shape of [16, 28, 28, 2]. Finally, the second operation of the layer re-orient the features to the horizontal reference and outputs a roto-translational feature space.

The following layer then pads the feature space by itself (Fig. 4.6b). As explained in the previous steerable approach, this padding, in conjunction with a 3D convolutional predictor, emulates the behavior of a cyclic convolution.

We then add a convolutional feature extraction stage (Fig. 4.6c). These convolutions are 2D convolutions, and we apply them to the second and third axis of the feature space. It means that the network preserves the first axis information, thus keeping the translation information of the feature space (Fig. 4.6d). The real and imaginary components of the feature space (last axis [16, 28, 28, 2]) are then input filters to the convolution operation.

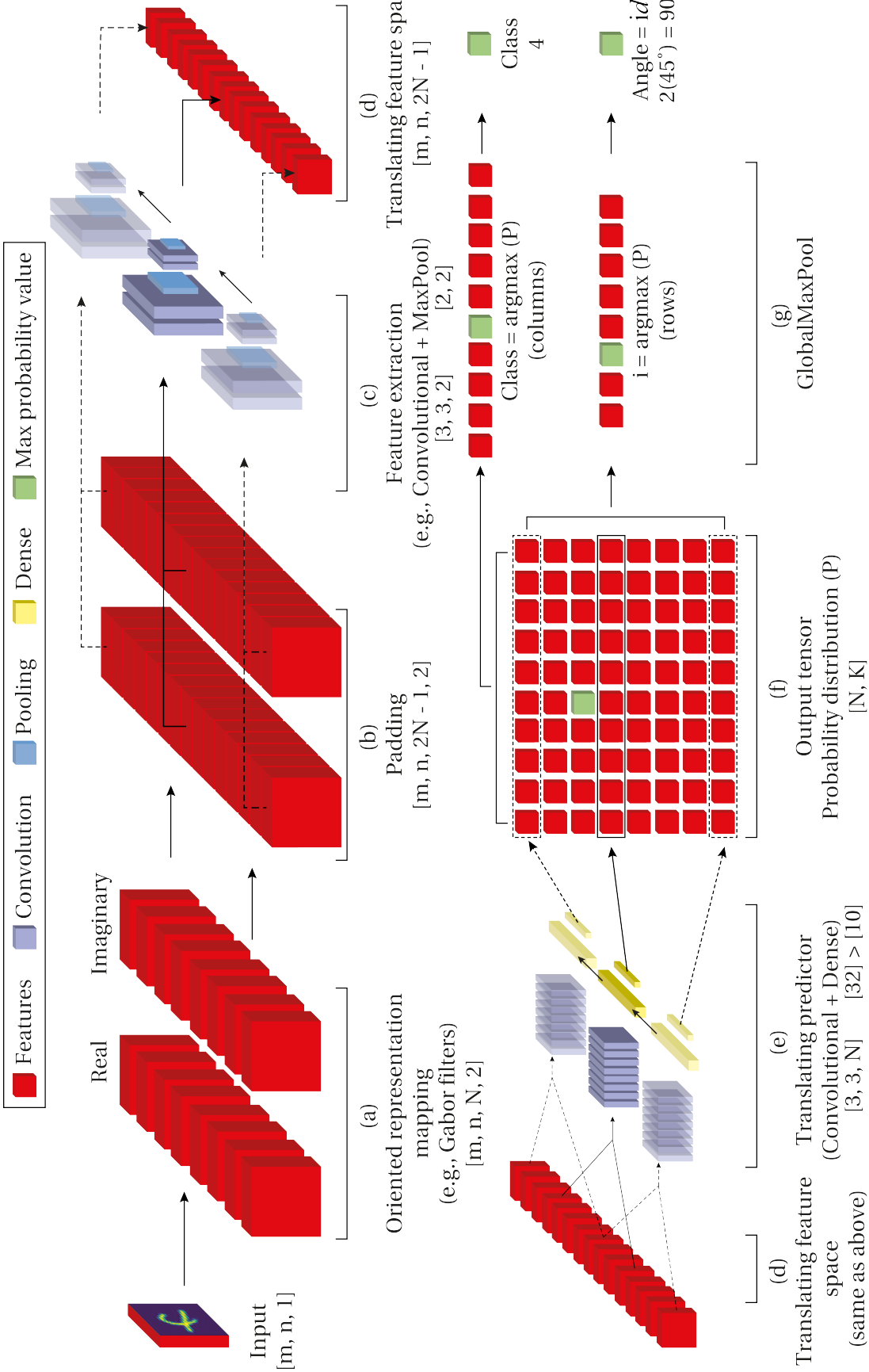


Figure 4.6 – RIN (Gabor + CNN). Proposed architecture using Gabor layer to obtain roto-translating features and a CNN feature extractor.

The predictor stage of the network is identical to RIN (steerable + CNN) approach. Recall that (chapter 3) we apply a 3D convolutional translating predictor (multiple instances of the same predictor with shared weights) to the roto-translational feature space to output a prediction for each translation of the feature space (Fig. 4.6e). The network stores each prediction in a probability distribution (Fig. 4.6f). Finally, a globalmaxpool operation applied to the columns and rows outputs the predicted class and angle information (Fig. 4.6g).

In the next section, we use this architecture on the MNIST dataset and compare the results with the previous implementations.

4.2.1 Results

To test rotational-invariance properties, we trained the network with upright samples (all the samples in the same orientation) and validated with randomly rotated samples of the MNIST dataset. We trained the network for 200 epochs using the Adam optimizer on a Titan Xp GPU with 16 GB VRAM; unless noted, the number of orientations is $N = 16$.

RIN (steerable + CNN) and RIN (Gabor + CNN) achieved state-of-the-art results when trained on upright examples and validated on rotated examples 4.1. If we compare the error rate of both, they outperform the previous state-of-the-art results. The nearest result is the work of Cohen et al. [2018] which is a particular case of a network that works on 3D spherical images with a high computational cost.

Table 4.1 – Obtained error rate on the MNIST dataset (train: upright / validation: rotated)

Method	Error rate	# parameters
ORN-8(ORPooling) Zhou et al. [2017]	16.67%	397k
ORN-8(ORAlign) Zhou et al. [2017]	16.24%	969k
RotInv Conv. (RP_RF_1) Follmann and Bottger [2018]	19.85%	130k
RotInv Conv. (RP_RF_1_32)* Follmann and Bottger [2018]	12.20%	1M
RotDCF (60 degrees) Gao et al. [2019]	17.64%	760k
Spherical CNN Cohen et al. [2018]	6.00%	68k
Icosahedral CNN Cohen et al. [2019]	30.01%	n.c.
RI-LBCNNs Shin and Yun [2019]	25.77%	390k
RIN (scattering + CNN) Rodriguez Salas et al. [2019]	17.21%	7k
RIN (steerable + CNN) Rodriguez Salas et al. [2021a]	2.05%	42k
RIN (Gabor + CNN) Rodriguez Salas et al. [2021b]	1.71%	9k

Concerning our contributions, we can discuss different points. Our first approach (RIN (scattering + CNN)), presents a low number of trainable parameters while having the worst error rate of the three. It is mostly due to the static nature of the scattering filters. The RIN (Gabor + CNN) approach presented in this chapter is better than the RIN (steerable + CNN) approach in terms of error rate but barely by 0.3% while requiring by far fewer parameters (42k vs 9k).

As observed in Table 4.1, RIN (Gabor + CNN) has a lower number of parameters while achieving higher accuracy; this is mainly due to the adaptability

of the filters (frequency, angularly selective, and frequency selective) compared to RIN (scattering + CNN) and RIN (steerable + CNN). Recall that steerable filters need to preserve the circular shape to keep their steerability properties compared to the Gabor filters that have better angular and frequency selectivity (section 4.1.1). Our results demonstrate the capability of the Gabor layer to generate a roto-translational feature space and use a translation predictor over this feature space to obtain rotation invariance and angular prediction properties to neural networks. Furthermore, the number of parameters has been kept lower than 10K. Compared to the RIN (steerable + CNN) approach, RIN (Gabor + CNN) trainable parameters reduction is almost five times.

4.3 Gabor Filters as Feature Extraction on Complex Datasets

The previous section validated the Gabor layer to generate a roto-translational feature space followed by a convolutional feature extractor and the translation predictor (3D convolution + dense layer). The Gabor layer approach outperformed the state-of-the-art results while reducing the number of trainable parameters significantly.

The next step is to test RIN (Gabor + steerable) in datasets that contain complex features. One commonly used dataset for this is the CIFAR-10 from the Canadian Institute For Advanced Research. The small-sized image examples from this dataset allow us to test the network capabilities on image sizes similar to the ones of the MNIST. Furthermore, literature has used this dataset in the context of upright training and randomly rotated validation.

CIFAR-10 (Figure 4.7) contains 60,000 examples separated into ten classes with the size of 32 x 32 pixels. While the MNIST dataset contains gray examples (one input channel), CIFAR-10 contains color images with three channels (RGB). The classes included in this dataset are mutually exclusive (e.g., there is no overlap between automobiles and trucks).

Usually, state-of-the-art approaches use this dataset in its original orientation. To test rotation invariance, we propose the usage of randomly oriented examples in the validation. It means training the network with the examples in their original orientation (upright position) and validating with randomly oriented examples. This training approach is analogous to the one used in the previous chapter with the MNIST dataset.

As observed in Figure 4.7, the images contain objects on different perspectives, rotations, and sizes. One of the main challenges is the in-plane and out-of-plane rotations. For example, the airplane class contains airplane images from different perspectives like flying with a ground perspective or landing on the track with a frontal or aerial perspective. Also, the small size of the images (32 x 32 pixels) makes the detection challenging when the objects are on a rich background or not in focus.

For the experiments with the CIFAR-10 dataset, we train the network with the original orientation of the examples and validate on rotated ones. Usually, when rotating images, we can find boundary artifacts. One of these artifacts

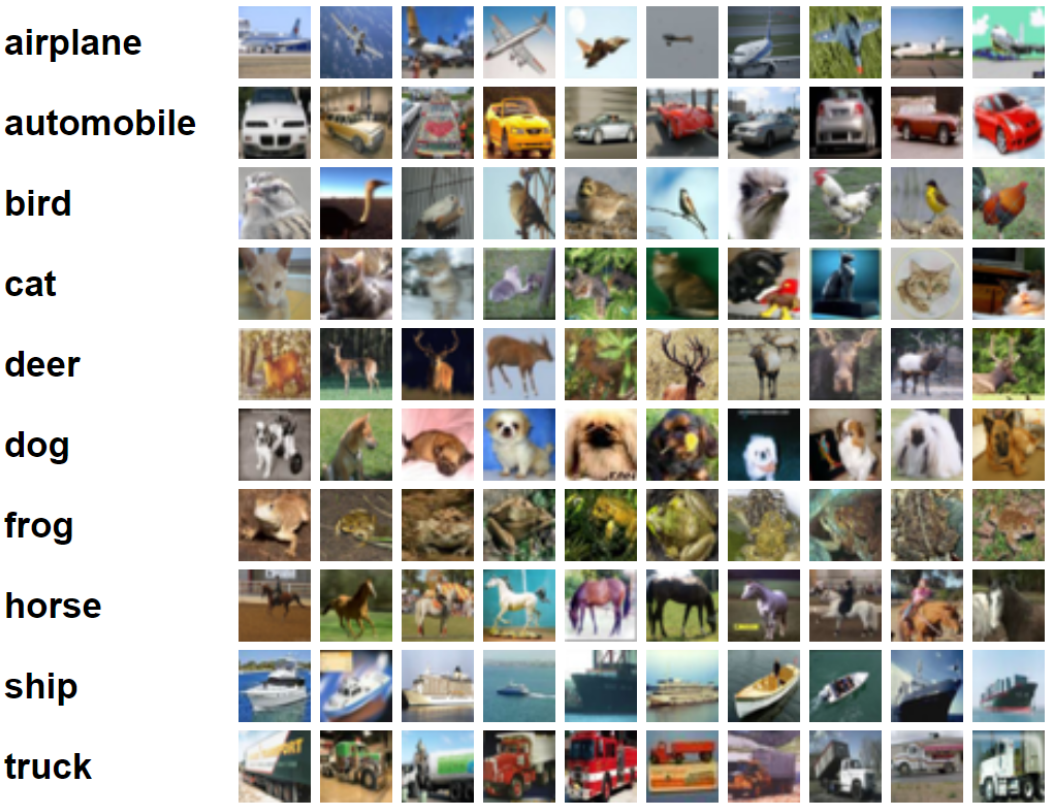


Figure 4.7 – CIFAR-10 classes

(a) Train airplane 0° (b) Train dog 0° (c) Test bird 225° (d) Test car 30°

Figure 4.8 – Up-right training (a,b) and randomly rotated (c, d) examples. All examples are processed to avoid black corners and borders. Test examples are rotated randomly between 0° and 360° clockwise.

comes in the form of black corners to fill the missing information product of the rotation. In some cases, the network could use the orientation of these edges as a feature to aid the angular prediction. Follmann and Bottger [2018] propose a pre-processing step on the dataset images to avoid the accuracy influence of these artifacts on rotated examples.

Following a previous approach (Follmann and Bottger [2018]), we first apply a circular mask to each example. The diameter of the circular mask is equal to the image width. Also, we smooth the boundary of the circular mask by applying a Gaussian blur mask with a kernel size of 5 pixels. It is important to mention that we apply this pre-processing step to the complete dataset (training upright examples and rotated validation examples) (Figure 4.8).

After applying the circular mask and the Gaussian blur to every example of the dataset, we randomly rotate the validation set examples. This pre-processing

step allows us to train the network with the original orientation of the examples (processed) and validate the rotation-invariant properties of the network with randomly rotated samples.

Compared to the MNIST dataset, the CIFAR-10 dataset contains color images. While MNIST contains a single grayscale channel, CIFAR-10 has three channels (red, green, and blue). Consequently, the input shape of the MNIST dataset is [28, 28, 1], and for CIFAR-10, the shape is [32, 32, 3]. As there exist different channel values (1 versus 3), it becomes challenging to use the same architecture used on the MNIST dataset.

One possible approach is to merge the three channels into one grayscale to preserve the network architecture. While this can be a helpful idea, it is commonly known in the deep learning field that color information is advantageous. Usually, more information is beneficial to the learning process.

Another possible approach is to apply the Gabor layer (section 4.1) to each of the color channels separately and then merge them. The merge step can be a linear sum between the colors, a weighted average, or a transformation. Nonetheless, we believe that merging the channels could be equivalent to an RGB to grayscale transformation hence losing information in this step. After some experiments, we validated that using the color increased the classification accuracy on the CIFAR-10 dataset.

Thus, we opted to preserve the information of the color channels through the network. To solve this, we applied the Gabor layer to each color and then concatenated them as a feature space. We apply the Gabor layer to the red channel of the CIFAR-10 example (input: [32, 32, 3]), and it generates a roto-translational feature space with real and imaginary responses (output: [32, 32, N, 2]) with as many orientations as N (section 4.2). We do the same to the green and blue channels. Consequently, we have three roto-translational feature outputs [32, 32, N, 2] with the color information. Finally, we concatenate these outputs to generate ensembled feature space with shape [32, 32, N, 6].

The feature space now contains for each orientation (third dimension) a set of real and imaginary RGB features (fourth dimension). This concatenation preserves the roto-translational feature space properties that the network needs.

After the roto-translational feature space, the following stages of the network are identical to the previously presented one. Recall that we pad the roto-translational feature space by itself. Then we apply 2D convolutions as feature extractors without affecting the translation axis. Finally, we apply the translating predictor to each translation (sharing weights) and outputs a probability distribution (P). Finally, a globalmaxpool operation outputs the predicted class and angle information (section 4.3).

4.3.1 Feature Extraction Stage Alternatives

Until this point, we have used a convolutional (CNN) feature extractor to the roto-translational space. When using MNIST applying the convolutional feature extractor was trivial. However, CIFAR-10 has three channels (RGB). In this sense, the way of connecting the CNN feature extractor to the roto-translational feature space changes.

To preserve the roto-translational properties of the feature space, we need to apply the predictor to each position of the translation and output a prediction of this position. Then move the predictor to the next translation up to all the positions are scanned. As we have a color input and the output of the Gabor layer is [28, 28, i, 6], we apply the convolution to the height and width (first and second dimension) and allow the six channels (last axis) to be treated as input features of the convolution. This method is analog to applying a convolution to an RGB image [x, y, 3], having three channels for the colors. Then we apply the convolution to the orientations N, and the output of each position is preserved in the table P.

It is important to notice that while the convolution has attention over one translation by position, the convolution shares weights and is the same convolution applied to each translation. It means that the network parameters are lower while preserving the predicting capability. The shared weight predictor learns to be angularly selective and output a higher probability when scanning the orientation that corresponds to the training orientation (usually the upright position) (chapter 2).

Finally, the result of each position is saved in a table with shape [K, N] where N are the number of orientations and K the number of classes. Then a maxpooling operation over the rows and columns outputs the class and angle.

4.3.2 ResNet Feature Stage

Up to this point, we have used a set of 2D convolutions applied to the roto-translational feature space as feature extractors. While this has been useful for simple features datasets like MNIST, they could not be enough for complex datasets such as CIFAR-10.

In this sense, we explore the possibility of replacing the feature extraction convolutions (Fig. 4.9c) with state-of-the-art feature extractors. Instead of applying a series of convolutions and maxpooling operations to the roto-translational feature space, we propose using a feature extraction stage from ResNet (He et al. [2016]) network.

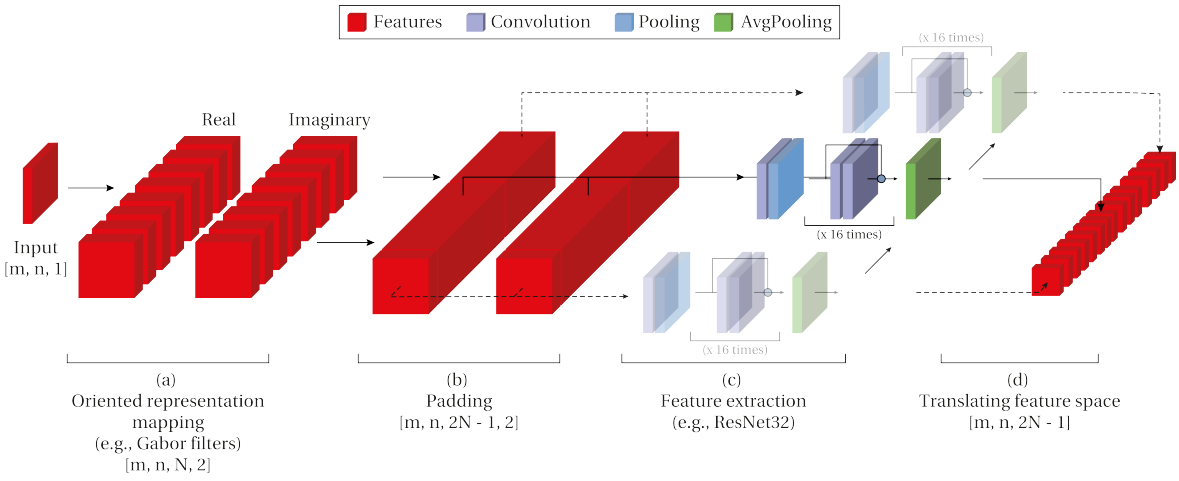


Figure 4.9 – The architecture feature extraction can be replaced from convolutions to a state-of-the-art feature extractor alternative.

Using the feature stages from ResNet would benefit the rotation-invariant network that we propose in several ways. In this sense, we could search for different state-of-the-art feature stages that could benefit our network. Furthermore, we could benefit from the usage of transfer learning and the use of pre-trained networks. Transfer learning refers to the technique of using a previously trained network on a similar task by changing the last layers of the neural network. Hence, we can use a previously trained feature extractor part of the network on classification tasks. This technique would allow us to reduce training time while preserving accuracy.

Nevertheless, the feature stage substitution comes with some challenges to adapt the shapes and connections between the Gabor layer and the possible feature extraction backbone. Usually, the state-of-the-art networks have an input of shape [width, height, channels] where the channels are usually three when used on RGB images. It means that the expected input of these networks is not compatible with the extra dimension that contains the translations.

One way to solve these incompatible shapes is to apply the feature extraction backbone to one orientation instead of to the complete translation of size N. This method would preserve the translation information as it does not affect the translation axis. After this, we concatenate the outputs and apply a small 3D convolution layer as translating predictor. Finally, we apply the translating predictor in the same way as the previous approaches. Recall that we connect several instances of the predictor (sharing weights) to each translation of the roto-translation feature space.

One challenge of using a feature extraction backbone is the roto-translational feature space properties preservation. After several experiments, we observed that when we apply the feature extraction backbone to the roto-translational feature space with an attention window of N, the translation property was lost. This phenomenon is mostly due to the layers acting as a transformation between the input and output. There does not exist any directive for the network to preserve the initial ordering of the filters.

To solve this challenge, we apply the feature extraction backbone to each filter of the roto-translational feature space individually. In this sense, the filter position is preserved until the output of the backbone. Consequently, the roto-translational properties of the feature space are preserved, and the backbone is used as a feature extractor to refine the features.

It is important to notice that while the convolution has attention over one translation by position, the convolution shares weights and is the same convolution applied to each translation. It means that the network parameters are lower while preserving the predicting capability. The shared weight predictor learns to be angularly selective and output a higher probability when scanning the orientation that corresponds to the training orientation (usually the upright position) (2).

Another workaround to preserve the roto-translational properties was to use the channel dimensions of the feature extraction backbone as input for the oriented filters. While this idea was intuitive, after several tests, we find that the network lost the oriented order of the filters, hence losing the roto-translation property. Furthermore, when using this approach, we could not use transfer learning

as the backbone network contains weights for three input channels and not as many as N.

As our main interest in this work is network size reduction, another challenging point was the number of parameters. As we used a feature extraction backbone, the number of trainable parameters increases when evaluating N orientations. For example, using a ResNet20 with 438k parameters multiplicated on $N = 16$ orientations would mean to scan 16 filters yielding about 7M of trainable parameters. To solve this, we implemented sharing weights for the backbone. Instead of N instances, we implement a single instance of the backbone network that scans each feature orientation. Consequently, the computing effort increases N times. As we focus on reducing the network size, we leave the reduction of computing effort to future works.

In the next section, we present the results of different implementations following the method outlined previously. Then, we present results on the CIFAR-10 dataset with steerable filters and the Gabor layer to compare. Also, we present results with the original convolution feature extraction and with different sizes of the ResNet network.

4.3.3 Results

To validate the introduced architectures, we follow the same methodology used on the MNIST dataset. We trained the network with upright-oriented samples and validated random orientations with the previously presented circular crop. We trained the network for 200 epochs using the Adam optimizer on a Titan Xp GPU with 16 GB VRAM, and unless noted, the angular sampling is $N = 16$.

In this section, we test several architecture combinations. We test RIN (steerable + CNN), and RIN (Gabor + CNN) as alternatives to generate the roto-translational feature space. Then, as alternatives to the feature extraction stage, we test a convolutional one (RIN (steerable + CNN)) and several depths of the ResNet feature extraction stage RIN (Gabor + ResNet).

The convolution feature extraction stage has three 2D convolution layers, with the number of filters increasing with the depth. A BatchNormalization layer follows up each one of the convolutional stages. Then we apply the translation predictor to the output of the feature stage, and the 2D table is generated with class and angle information. To make a fair comparison (using approximately the same number of parameters), we tested a different number of filters on each 2D convolution layer to approach the number of parameters of the following approach.

Then, we replace the feature extraction stage with the ResNet feature extraction part. As several depths of ResNet exist in the literature, we think it is also interesting to evaluate the impact of the network's depth on the accuracy. We use the ResNet backbone as found in the literature. The only change is the input filters going from 3 to 6 to allow the feature stage backbone to be compatible with the shape of the real and imaginary information of the Gabor filters.

Table (4.2) condenses the results of these experiments compared to the current state-of-the-art approaches. At first sight, we can observe that RIN (Gabor

Table 4.2 – Obtained error rate on the CIFAR-10 dataset (training: upright/validation: rotated)

Method	Error rate	# parameters
RotInv Conv. (RP_RF_1) Follmann and Bottger [2018]	55.88%	130k
ORN Zhou et al. [2017]	59.31%	382k
RP_1234 Cohen and Welling [2016]	62.55%	130k
RIN (steerable + CNN) Rodriguez Salas et al. [2021b]	36.41%	73k
RIN (Gabor + CNN) Rodriguez Salas et al. [2021b]	37.60%	93k
RIN (Gabor + CNN) Rodriguez Salas et al. [2021b]	28.69%	238k
RIN (Gabor + CNN) Rodriguez Salas et al. [2021b]	33.25%	586k
RIN (Gabor + ResNet8) Rodriguez Salas et al. [2021b]	27.68%	243k
RIN (Gabor + ResNet14) Rodriguez Salas et al. [2021b]	27.34%	341k
RIN (Gabor + ResNet20) Rodriguez Salas et al. [2021b]	21.10%	438k
RIN (Gabor + ResNet26) Rodriguez Salas et al. [2021b]	21.50%	537k

+ ResNet26) performs better in error rate terms than the other approaches. The smallest convolutional network (RIN (steerable + CNN)) with 93k parameters is also the smallest of all the approaches while achieving competitive accuracy. Increasing the number of filters in the convolutions achieved a lower error rate up to 238k. After this value, the error rate difference was not significant. Nonetheless, all the convolution backbone approaches outperformed the state-of-the-art works in error rate.

For the ResNet feature stage, we test 4 different approaches with depth variations. The smallest one (RIN (Gabor + ResNet8)) achieved state-of-the-art values outperforming even the convolutional approaches. While increasing the depth reduced the error rate almost by 7%, the number of parameters also increased quickly. Increasing the number of layers beyond 26 did not result in better error rates.

The presented networks in this section obtained rotation invariant properties while trained with samples in one orientation (upright oriented). These results prove the network's capability to generalize and correctly predict the class despite the orientation of the sample on the validation. Also, the network preserved the equivariance of the roto-translation feature space and predicted the angle of the input. We obtained these results without using data augmentation on the dataset. The state-of-the-art approaches presented in Table (4.2) achieve up to some degree rotation invariance, and they do not predict the input's angle.

4.4 Conclusions

In this chapter, we presented RIN (Gabor + CNN) as an alternative to the steerable filters and scattering transform based networks. Also, we introduced and presented the possibility of replacing the convolutional feature stage of the network with the ResNet feature stage (RIN (Gabor + ResNet))

Gabor filters represented an enhancement to the previous approaches. They present higher flexibility and descriptive potential than the steerable filters and scattering transform. We can observe this flexibility on the angular and frequency

selectivity of the filters on the trained data. Furthermore, these filter shapes are trainable and learn the frequency, angular selectivity, and frequency selectivity from the training data. The steerable filters approach has a unique trainable parameter that changed its size. The enhanced features of the Gabor filters open the possibility to work with datasets containing images with complex features. In this chapter, we presented one set of Gabor basis filters (with real and imaginary responses), but it is possible to add as many filters as necessary for the problem.

The usage and validation of Gabor filters also demonstrated that it is viable to use different types of oriented filters to obtain a roto-translational feature space. Up to this point, we have validated on different architectures and datasets that decomposing the input image in several oriented components and aligning them to a reference generates a feature space that linearly translates with the input rotation. We have observed the same behavior on the scattering transform (for 1D and 3D cases), on the steerable filters (RIN (steerable + CNN)), and finally on the Gabor filters (RIN (Gabor + CNN))described in this chapter. Furthermore, using a translating predictor over this roto-translational feature space to obtain angle prediction has been validated experimentally.

The second main contribution comes in the possibility of using state-of-the-art networks (such as ResNet) as the backbone network of the model. This possibility means that the network can accept different existing network architectures feature stages and allows the developer to select a backbone to suit the task.

While we obtained state-of-the-art results on the MNIST dataset, we moved to a more complex dataset as CIFAR-10. We managed to outperform the state-of-the-art accuracy with RIN (Gabor + CNN) and different backbones RIN (Gabor + ResNet) on this challenging dataset. Also, while RIN (Gabor + CNN) surpassed the state-of-the-art error rate, the lowest error rate reached was about 29%. Increasing the size and number of convolution filters was not productive for the task and quickly overfitted the network. On the other hand, RIN (Gabor + ResNet) reduced the error rate by more than 10% up to 21% of the error rate.

Another interesting result of these experiments is the validation obtained when using the roto-translational feature space and the translating predictor operation. This chapter demonstrated that adding these two functions (roto-translational feature space mapping and translating predictor) to an existing network endows it with rotation-invariant properties.

It is important to notice that the obtained experimental results open the possibility to generalize these results and propose a methodology to obtain rotation invariant networks when following some guidelines. In the next chapter, we study this generalization concept and provide a mathematical proof of the trainability of the networks.

Chapter 5

Mathematical Background and Formal Methodology

In the previous chapters, we introduced the roto-translational feature space concept. Also, we outlined the two operations involved in its creation: oriented feature space and translating predictor. Then, we proposed several alternatives to generate the roto-translational space, such as scattering network, steerable filters, and Gabor filters. Also, we demonstrated results on datasets with simple features (MNIST) and with complex features (CIFAR-10). After obtaining experimental results with these alternatives and their combinations (scattering, steerable, Gabor and CNN, ResNet), we converged on a general methodology to obtain rotation-invariant properties on CNNs.

In this chapter, we formalize the definitions and properties of this methodology. First, we outline the oriented feature space operations and re-orientations. Then, we present proof of the translating features over the roto-translational feature space. In the second part, we present mathematical proof of the trainability of the translating predictor. Finally, we discuss the Vapnik-Chervonenkis Dimension as mathematical proof of the reduced size of the network.

5.1 Oriented Feature Space

This section introduces the theoretical background of the proposed method and proves that the network is trainable. First, we start by introducing the oriented feature space representation.

Let $\mathbf{x}$ be an example (an image) $\mathbf{x} \in \mathbb{R}^{m \times n}$, and let g be a filter acting as an oriented edge detector with the periodicity 2π. Let ρ_φ be a transformation rotating the support by the angle φ. Then $\rho_\varphi g = g^\varphi$, with $g{=}g^{0^o}$, be such a filter oriented along the angle φ.

The product $\mathbf{x} * g^\varphi$ extracts the oriented components of $\mathbf{x}$ that are oriented in φ. We have a set of orientations $\varphi_i = 2\pi i/\mathrm{N}$ for $i = 0, \ldots, \mathrm{N}$, with $\mathrm{N} \in \mathbb{Z}^+$, and $d\varphi = 2\pi/\mathrm{N}$. Where N is the number of orientation components. The ordered set $[\mathbf{x} * g^{\varphi_i}]$ contains all oriented components of $\mathbf{x}$.

Definition 5.1.1 (*feature representation)*. Let Φ be a mapping $\Phi : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{m \times n \times \mathrm{N}}$, and $\Phi(\mathbf{x})$ a feature representation of $\mathbf{x}$ containing oriented components of $\mathbf{x}$, and re-oriented to align with the referential orientation φ_0.

$$\Phi(\mathbf{x}) = [\rho_{-\varphi_i}(\mathbf{x} * g^{\varphi_i})] \tag{5.1}$$

Consider now a special type of permutation τ where the elements move rightwards and the last one replaces the first one. This permutation is cyclic with the period equal to the length of the vector it is applied to. We refer to τ by *translation*, and τ_i denotes τ applied i-times. In this context, the translation is applied along the third dimension. That is, applied to $\mathbf{x}$, Φ and $[g^{\varphi_i}]$, in this context, the following holds:

1. A translation τ applied to some $\mathbf{x}$ is the identity operator $\tau\mathbf{x} = \mathbf{x}$ because $\mathbf{x}$ lacks the third dimension.
2. $\tau[g^{\varphi_i}] = [\rho_{d\varphi} g^{\varphi_i}]$ because of $\rho_{d\varphi} g^{\varphi_i} = g^{\varphi_{i+1}}$
3. $\tau_k[g^{\varphi_i}] = [\rho_\vartheta g^{\varphi_i}]$ for some $\vartheta = kd\varphi,\ \forall k \in \mathrm{Z}$

We have the following property:

Property 1. *There is a covariance of the rotation of* $\mathbf{x}$ *with the translation of* $\Phi(\mathbf{x})$ *along its third axis*

$$\tau_k \Phi(\mathbf{x}) = \Phi(\rho_\vartheta \mathbf{x}) \quad \textit{with} \quad \vartheta = kd\varphi,\ \forall k \in \mathbb{Z} \tag{5.2}$$

Proof 1. We have

$$\begin{aligned}
\Phi(\rho_\vartheta \mathbf{x}) &= [\rho_{-\varphi_i}(\rho_\vartheta \mathbf{x} * g^{\varphi_i})] \\
&= [\rho_{-\varphi_i}\rho_\vartheta(\mathbf{x} * \rho_{-\vartheta} g^{\varphi_i})] \\
&= [\rho_{\vartheta-\varphi_i}(\mathbf{x} * g^{\varphi_i - \vartheta})] \\
&= \tau_k[\rho_{-\varphi_i}(\mathbf{x} * g^{\varphi_i})] = \tau_k \Phi(\mathbf{x})
\end{aligned}$$

Considering all $k = 0, \ldots, \mathrm{N}$ the left-hand side of eq. 5.2 gives access to features coresponding to all rotations $\rho_\vartheta \mathbf{x}$. Among all these rotations, there is one that corresponds to an almost unrotated version of $\mathbf{x}$.

5.2 Prediction Model

Once we obtained a roto-translational feature space as explained in the previous section, the next step is to prove the trainability of the translating predictor. In this section, we first outline the properties and definitions involved in the network training and then mathematically prove the trainability of the prediction model.

For some given, joint probablity $p(y, x)$ let us search for a model assigning to x a class y. However, not x but only $\rho_\vartheta x$ is observable. A classical way of learning a model to approximate p is training the model using a data augmentation $y = f^{\mathrm{DA}}(\rho_\vartheta x)$. Using a feedforward network we learn a mapping

$$y = f^{\mathrm{DA}}(\rho_\vartheta x; \boldsymbol{\theta}) \tag{5.3}$$

where $\boldsymbol{\theta}$ are the model parameters and the superscript DA indicates a data augmented model. We typically proceed by maximizing the probability of the prediction by minimizing a cost function $J(\boldsymbol{\theta}) = -\log p(y \mid \rho_\vartheta x)$.

Instead of that classical way, consider a model f in the form of concatenation of two mappings $f(\cdot) = f^*(\Phi(\cdot))$, where Φ is a representation mapping and f^* a class-probability model. Using the mapping Φ from eq. 5.1 and considering all its permutations $\tau_i\Phi$ for $i = 1,\ldots,N$ we obtain a collection of models

$$[f_i(\cdot)] = [f^*(\tau_i\Phi(\cdot))] \tag{5.4}$$

with $f^*(\cdot) = f^*(\cdot,\boldsymbol{\theta}_{f^*})$ and $\Phi(\cdot) = \Phi(\cdot,\boldsymbol{\theta}_\Phi)$. The parameters $\boldsymbol{\theta}_\Phi$ and $\boldsymbol{\theta}_{f^*}$ are independent of the rotation of x. We analyze this collection of models below.

In the following the term $p(A)$ denotes the probability of some stochastic event A to occur. More particularly, when $p(A)=1$, it indicates the certainty even though A is stochastic.

Definition 5.2.1 (*Indiscernibility*)**.** Let f be a classification model, $\mathbf{x}_1$ and $\mathbf{x}_2$, with $\mathbf{x}_1 \neq \mathbf{x}_2$, two examples and $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$ the predictions by f on these examples. When f predicts the same class for any $\mathbf{x}_1$ and $\mathbf{x}_2$, that is $p(f(\mathbf{x}_1)=f(\mathbf{x}_2)) = 1$ then we say $\mathbf{x}_1$ and $\mathbf{x}_2$ are indiscernible by f, written $\mathbf{x}_1 \simeq_f \mathbf{x}_2$.

Property 2 (*Equivalence of representation*)**.** *Let f be some classification model in the form of $f(\cdot) = f^*(\Phi(\cdot))$, and $\rho_\psi\mathbf{x}_1$ and $\rho_\varphi\mathbf{x}_2$ be two differently oriented examples. If the two different, and differently oriented, examples are indiscernible by the model, that is $p(f(\rho_\psi\mathbf{x}_1)=f(\rho_\varphi\mathbf{x}_2)) = 1$ then we write*

$$p(f^*(\tau_i\Phi(\rho_\psi\mathbf{x}_1)) = f^*(\tau_j\Phi(\rho_\varphi\mathbf{x}_2))) = 1$$

after rotation of the support by $\rho_{-\psi}$ we obtain

$$p(f^*(\tau_i\Phi(\mathbf{x}_1)) = f^*(\tau_j\Phi(\rho_{\varphi-\psi}\mathbf{x}_2))) = 1$$

substitution $\vartheta=\varphi-\psi$

$$p(f^*(\tau_i\Phi(\mathbf{x}_1)) = f^*(\tau_j\Phi(\rho_\vartheta\mathbf{x}_2))) = 1$$

using the Definition 5.2.1 to drop the probability

$$\tau_i\Phi(\mathbf{x}_1) \simeq_{f^*} \tau_j\Phi(\rho_\vartheta\mathbf{x}_2)$$

by translating both sides by τ_{-j} we finally obtain the indiscernibility of representation of some rotated example $\mathbf{x}_2$ and a translated representation of some upright $\mathbf{x}_1$.

$$\exists k,\ \tau_k\Phi(\mathbf{x}_1) \simeq_{f^*} \Phi(\rho_\vartheta\mathbf{x}_2), \qquad \text{with } k = i-j$$

There is a way of finding a convenient k for some given (and unknown) ϑ by using the maximum likelihood.

Lemma 1. For any $\vartheta \in \mathbb{R}$, and $0 < d\varphi$ sufficiently small

$$\exists k \text{ such that } \tau_{-k}\Phi(\rho_\vartheta \mathbf{x}) \simeq \Phi(\mathbf{x}) \tag{5.5}$$

and k depends on $\mathbf{x}$.

Proof 2. From Prop. 1 for $\vartheta = d\varphi$ and $k=1$ we have the equality $\tau_{-k}\Phi(\mathbf{x}) = \Phi(\rho_\vartheta \mathbf{x})$. However, because $\vartheta \in \mathbb{R}$ but $k \in \mathbb{Z}^+$ and when $kd\varphi \neq \vartheta$, $\forall k$, the exact equality is not possible and only $\tau_k\Phi(\rho_\vartheta \mathbf{x}) \simeq \Phi(\mathbf{x})$ since a misalignment occurs between $\Phi^{-1}(\tau_{-k}\Phi(\rho_\vartheta \mathbf{x}))$ and $\Phi(\mathbf{x})$ but at most up to $\pm\frac{d\varphi}{2}$.

Because of the Lemma 1, one of the models in eq. 5.4 will maximize the class probability predicted from each representation mapping $\tau_i\Phi$

$$\max_i p_{f_i}(y = \mathbf{y} \mid \rho\mathbf{x}) \tag{5.6}$$

We want to minimize the negative log-likelihood cost of the model providing the correct class prediction

$$\begin{aligned} J(\boldsymbol{\theta}) &= -\log p(y = f^*(\tau_k\Phi(\rho_\vartheta x) \mid x, \vartheta)) \\ \text{where } k &= \arg\max_i p(y = f^*(\tau_i\Phi(\rho_\vartheta x) \mid x, \vartheta)) \end{aligned} \tag{5.7}$$

in a usual way by minimizing the per-example loss

$$\mathscr{L}(x, y, \boldsymbol{\theta}) = -\log p(y \mid x, \vartheta; \boldsymbol{\theta})$$

When the k-th model maximizes the probability in eq. 5.7 we adapt the weights of the k-th model by taking $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon g$ where $g = \nabla_{\boldsymbol{\theta}} \mathscr{L}(\rho_\vartheta x, y; \boldsymbol{\theta})$. The weights of layers are updated by using the error back propagation algorithm as usual $f_k(.; \boldsymbol{\theta}) = f^*(\tau_i\Phi(.; \boldsymbol{\theta}_\Phi); \boldsymbol{\theta}_{f^*})$. Notice that ϑ in $\rho_\vartheta x$ is not needed to be known since the weights $\boldsymbol{\theta}_\Phi$ and $\boldsymbol{\theta}_{f^*}$ are independent of ϑ. Updating the weights of the mapping $\tau_k\Phi\mathbf{x}$ is done using $\tau_k[\rho_{-\varphi_i}(\mathbf{x} * g^{\varphi_i})]$ which consists of three functions: translation τ_k, a set of fixed-angle rotations $\rho_{-\varphi_i}$ and convolution of x by a set of filters g^{φ_i}, where only the parameters of the edge detector g are trainable. This training is a classical closed-loop training iterated until convergence.

5.2.1 The Vapnik-Chervonenkis Dimension of the Model

Recall the Vapnik-Chervonenkis (VC) dimension of a classifier (Vapnik and Chervonenkis [1971]), defined as the maximal number of different points $\mathbf{x}$ that the model can label arbitrarily. Intuitively, a more complex problem requires a model with a higher VC dimension, and the model size will be bigger.

Consider now a model f assigning a class y to x for some joint probability $p(y, x)$.

When only $\rho_\vartheta x$ are observable, the classification $\hat{y} = f(\rho_\vartheta x)$ can be done by using the data augmentation to train $y = f(\rho_\vartheta x)$, with all ϑ, see eq. 5.3. This model data will have a high VC dimension.

Consider now a f fitted to $p(y, x)$, i.e. with no data augmentation. When only $\rho\mathbf{x}$ is observable, the classification $\hat{y} = f(\rho_\vartheta \mathbf{x})$ and the prediction of the angle ϑ could

also be done by using the maximum likelihood polling, eq. 5.6. This approach however would be computationally costly since the prediction needs to be done over all the rotations.

If the model f can be split into two parts, as in eq. 5.4, the predictions $f(\rho_{\vartheta_i}\mathbf{x})$, for all possible ϑ_i, become very cheap. We compute a unique feature representation $\Phi(\mathbf{x})$. The predictions are then computed by the second part of the model f^* on the mere translations $\tau_i\Phi(\mathbf{x})$.

The VC dimension of f, eq. 5.4, is much smaller than that of f^{DA}, eq. 5.3, since f learns the representation of $p(y, \rho_\vartheta x)$ with only $|\vartheta| < \frac{d\varphi}{2}$ instead of $\vartheta \in (0, 2\pi)$ when data augmentation is used. Consequently, the model size also is smaller.

Moreover, for the model f, eq. 5.4, we have $|\boldsymbol{\theta}_f| = |\boldsymbol{\theta}_{f^*}| + |\boldsymbol{\theta}_\Phi|$, where $|.|$ denotes the number of parameters. The mapping $\Phi(\cdot)$ is computed only once, and the predictions $[f_i^*(\cdot)]$ are cheap since using only a subset of parameters $|\boldsymbol{\theta}_{f^*}|$.

In this sense, a model trained with data augmentation (multiple rotations of the same example) presents an increase in size to overcome the complex problem. Also, it means that the translating predictor naturally has smaller size as it does not need to become rotation invariant but angular selective to one orientation. We can demonstrate this concept experimentally by comparing a CNN baseline model trained on randomly oriented samples with our approach.

The selected baseline model consists of a series of convolution layers followed by maxpooling operations. We can observe in Fig. 5.1 the baseline architecture used in this experiment. Also, we increase the number of filters on the convolutional layers, thus increasing the size of the network up to 3.6 M parameters. Finally, we compare the baseline model with the previously presented RIN (Gabor + CNN) with the same training strategy (RRT).

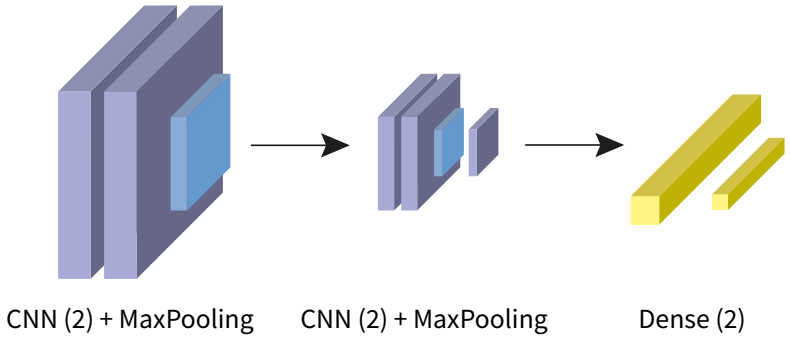


Figure 5.1 – Baseline CNN based on LeNet-5.

As we can observe in Fig. 5.2 the accuracy of the baseline tends to increase when increasing the size of the model (number of convolutional filters). However, the accuracy of the baseline does not significantly increase from 2.5 M of parameters and beyond. On the other hand, RIN (Gabor + CNN) reaches 98% of accuracy with less than 100k parameters. This reduced size is due mainly to the small predictor size present in the model. The predictor does not become invariant to several rotations ; hence the VC dimension of the model is lower than the baseline model.

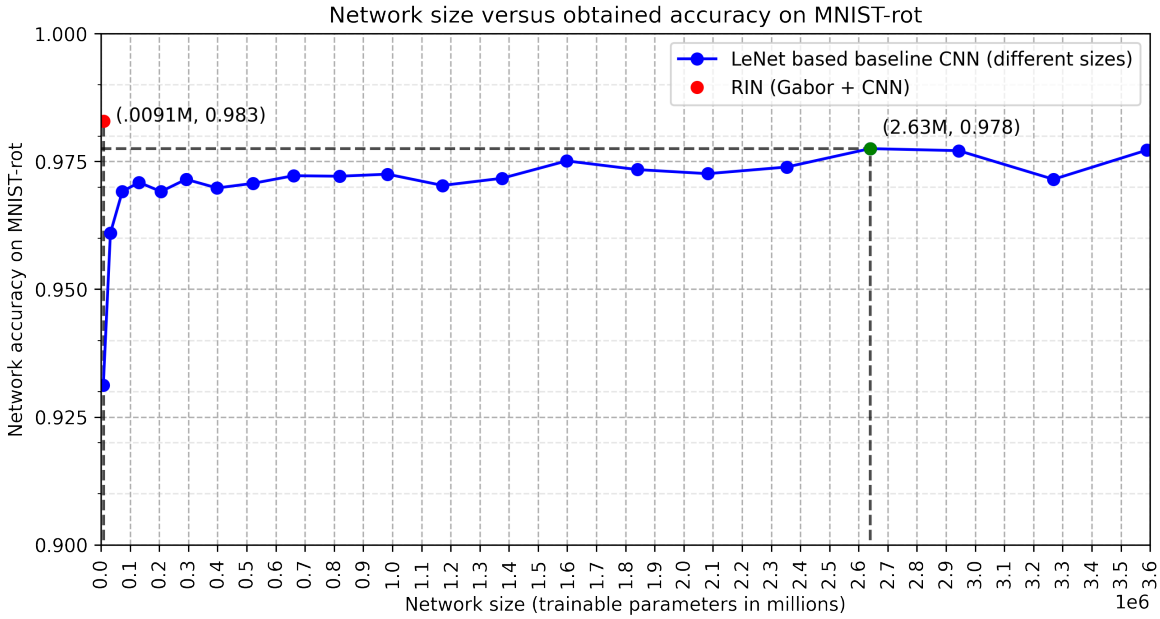


Figure 5.2 – RIN (Gabor + CNN) compared with baseline CNN when trained on randomly oriented samples (RRT).

5.3 Conclusions

In this chapter, we presented a general methodology to obtain rotation-invariant convolutional neural networks. For a more straightforward explanation, we divided this approach into two main parts. First, the oriented feature space generates a roto-translational feature space containing the angular and edge orientation of the image. Second, the prediction model was applied to the roto-translational feature space to obtain a probabilistic distribution with the class and angle inference.

Theoretically, the feature space representation can be built around any filter with the faculty of being oriented in a particular direction. We demonstrated this theory by proposing several alternatives to generate the oriented filters (scattering, steerable, and Gabor). Then, the oriented features are re-aligned to obtain a roto-translational feature space. While this operation is computationally intensive, it is compulsory, as proven in this chapter.

In the second part, we outlined the mathematical proof about obtaining rotation invariant networks while keeping a small predictor size. Also, we highlighted the mathematical notion behind the predictor and proved its trainability when trained as translating predictor.

Finally, we recalled the Vapnik-Chervonenkis dimension of the model, which outlines the properties and proof of a smaller predictor obtained by the methodology.

In this sense, one of the main products of this manuscript work is the generation of this methodology. Following the definitions and lemmas outlined in this chapter, we can endow rotation invariant properties on neural networks. It means that several other possibilities can be explored as an alternative for the oriented features or the translating predictor. Some of these alternatives are discussed in the next chapter.

Chapter 6

Conclusions and Future Works

Deep learning has become the standard solution for visual understanding tasks. In this sense, we can find neural networks running in several hardware targets. With many of these targets having limited resources (low CPU frequency to preserve energy, low memory, lack of GPU), it becomes important to develop smaller and less computational intensive networks.

We study how to endow a classification model with robustness against input image deformations. Specifically, we selected the rotation invariance as we studied that endowing the feature stage with intrinsic rotations would result in a smaller classifier (chapter 5). To conclude this manuscript, we review our contributions to this topic and discuss future work perspectives.

6.1 Contributions

The first step was to obtain a feature stage with intrinsic rotations. To obtain this, we based our work on the previous concept of oriented feature space (Chen et al. [2000]). One of our contributions is re-orienting these features to a horizontal reference (chapter 2). The re-orientation of the oriented feature space outputs a roto-translational feature space. The roto-translational space presents a covariance between the angle of rotation of the input image and the translation of the oriented filters.

One of the most important properties of this roto-translational feature space is capturing the inner angular relationship between the edges of the image and mapping them as a linear distance between the filters. Consequently, the roto-translational feature space stores all the possible rotations (up to N rotations) of the input as translation positions. We can also see this as a domain transformation between the angular and linear domains between the input image and the filter space.

The oriented feature space is a sparse representation of the image. This comes from the fact that a 2D image is projected into a 3D space. We validate the possibility of using sparse, oriented features of the roto-translational feature space as filters of a convolutional network (in the first layer) on a classification problem with a more challenging classification task, such as face recognition. For this problem, we evaluated if the network converged adequately using an oriented feature space re-orientated to obtain a roto-translational feature space.

The translating predictor was not included in this experiment as the main objective was to prove the possibility of convergence with a roto-translational feature space. The network could perform a classification task by effectively recognizing faces, and it served as proof of the concept of the viability of using these filters on classification problems.

After this, we presented the translating predictor. The translating predictor is implemented as a collection of small identical predictors (sharing weights) that read each translation of the feature space. Because the predictors are typically small (no data augmentation by rotation), their execution can be efficiently and easily done in parallel on a GPU. The prediction of each predictor is stored in a table that becomes a probability distribution. In this distribution, the maximum likelihood corresponds to the almost upright orientation of the input image as seen by the predictor at that particular position (stored in one of the translations). As explained previously (chapter 2), during training, the prediction of the predictor in the branch that corresponds to the upright oriented features is reinforced. This training setup does not reinforce the other positions but the upright-oriented features. During the training, the same example is eventually presented to the network again (albeit with another rotation). Consequently, the predictor then becomes angularly selective in the sense that it learns to output lower (near zero) probability when the features do not correspond to the upright oriented example. Consequently, by squeezing out the invariance to the rotation from the predictor, it will become smaller in size compared to other predictors that need to become rotation invariant, as demonstrated by the Vapnik-Chervonenkis conclusion (chapter 5).

We validate this proposal with a dataset that contains simple features (MNIST handwritten numbers). On this dataset and its rotated variations, we reached state-of-the-art performance using the scattering transform as oriented features. Furthermore, we outperformed the previous state-of-the-art approaches with the steerable filters as oriented features (chapter 3).

After this, we presented results on datasets with complex features such as CIFAR-10. We proposed using Gabor filters to generate the oriented feature space to overcome the limitations of the steerable filters (chapter 4). With Gabor filters, we outperformed previous state-of-the-art approaches while keeping the network size smaller than our competitors.

Furthermore, we explored the possibility of using alternative neural network backbones. This possibility means changing the convolutional backbone used with simple feature datasets to a state-of-the-art network. In this sense, the backbone can be any convolutional network oriented towards classification tasks. To prove this, we replaced the convolutional backbone with a ResNet network with different depths. As a result, we obtained an increase of almost 10% in accuracy compared to the convolutional and 34% improvement compared to the current state-of-the-art techniques. Recall that RIN (Gabor + CNN) obtained a 1.71% error rate on the MNIST dataset with 9k parameters. The smallest network approach in the state-of-the-art was the Spherical CNN from Cohen et al. [2018] with 68k parameters and a 6% error rate. On the CIFAR-10 dataset, RIN (Gabor + ResNet20) with 438k parameters obtained a 27.34% error rate compared to the 55.88% error rate obtained by Follmann and Bottger [2018] with 130k.

To complete this work, we presented a mathematical proof of the trainability of the network when using oriented filters and a translating predictor. We also completed this proof by discussing the previous experiments and highlighting the behavior and properties of these operations (chapter 5).

6.2 Future Works

For future works, we present several lines of research that could benefit this work. For an easier understanding, we divide these into methodological and technical sections.

For this work, we focus on in-plane rotations of the input image. The obtained results on the simple features datasets demonstrated that the network gets high accuracy when the pattern rotates in the plane. When there exist out-of-plane rotations (i.e., CIFAR-10 dataset), the network does not obtain high accuracy (80% accuracy with ResNet backbone). These results open the possibility of studying the addition of an extra dimension to the feature space. One possible study axis is 3D inference with a single image (Kanazawa et al. [2018]). A joint effort between the roto-translational feature space and angular prediction of our network with the mesh transformation could benefit the 3D mapping of the texture.

Also, a possible axis of study comes in the form of invariance to scales. The scale invariance problem is usually solved by iterative processing the input image at different scales. There exist some studies (Ghosh and Gupta [2019]) where steerable filters are used to obtain scale-invariant CNNs. In this sense, an additional axis where the scale information is stored could be possible by finding a representation space representing the scale and the angle. It means that the network would store the scale information in a similar way that stores the angle in this work.

Another possible improvement comes from the re-orientation of the feature space. As explained previously (chapter 2), we rotate the oriented features obtained by oriented decomposition (scattering, steerable, and Gabor) to the horizontal reference. When working with small images (28 x 28 pixels), the computational effort of this process is negligible. However, when the input images increase in size, the rotation with bilinear interpolation of each filter becomes computationally costly. Even though it can be executed in parallel, it can present a high latency. Unfortunately, several experiments demonstrated that the re-orientation of the filters is crucial to obtain the roto-translational feature space. It means that it is a process that we can not avoid in the methodology and experimental setup of the network. Finding a workaround to rotate the features efficiently or generate a roto-translational feature space without re-orientation would significantly decrease the inference time and considerably reduce the training time. This workaround could consist of obtaining the angle of a feature and projecting it at the corresponding slice of the feature space using circular harmonics (Worrall et al. [2017]) or kernel-mapping techniques (Zhou et al. [2019]). Worrall et al. [2017] obtains a feature representation which is a complex number (feature and orientation) based on circular harmonics. In this sense, we could replace the oriented representation with a complex number representation of the features.

The roto-translational properties of properties would be preserved, and we could predict the position and angle.

One of the technical improvements comes from making an efficient angular decomposition of the input image. At the moment of this writing, we generate the oriented filter ensemble from a set of trainable variables. Recall that we create a kernel window using these trainable variables and generate a basis filter used in the first layer of the network. To generate this kernel window, we use a cycle (for loop) to set the value of each pixel iteratively. Then, we rotate the basis filter N times. This rotation implies a second cycle that rotates the basis filter (using the angle parameter) as many times as N. The network executes these two loops during the training and inference. Improving the generation of the collection of filters would result in lower forward pass time and reducing the training time epoch significantly. There exist several possible approaches to do this. One is using parallelization techniques where different processor threads manipulate each oriented filter separately. It is possible because there is no data dependency between the orientated filters. Other possibilities are using vectorization techniques, code optimization, or graph theory to simplify the tensor graph. Finally, automatic differentiation to compute the gradient could be another interesting axis of study instead of using the automatic differentiation from TensorFlow.

An interesting future work axis comes in the form of a custom loss. For example, when classifying rotated objects, it might be profitable to use a custom loss such as the Pixels-Intersection-over-Union (PIoU) (Chen et al. [2020b]). The idea is that an oriented bounded box – one aligned with a rotated object – presents less overlap with the background in complex environments. In this sense, the angular prediction would benefit from orienting the bounding box to overlapping correctly with the detected object. Furthermore, this intuition would benefit the detection of rotating objects usually done with horizontal bounding boxes.

Yet another technical improvement is the cyclic convolution. Up to our knowledge at the moment of this writing, there is no cyclic convolution operator in the used framework. Our workaround consists in periodic padding of the roto-translational feature space by itself to emulate this behavior. So, there exist the possibility to develop and find an efficient way to implement a cyclic 3D convolution. Having this cyclic 3D convolution would benefit the network because no padding would be necessary to obtain all the possible translations prediction. Consequently, we could reduce the memory occupation and the computational effort of the network.

The translating predictor also presents another opportunity to improve. In this work, we used 3D convolutions scanning each orientation of the roto-translational feature space and outputting a prediction for each translation. We can avoid using 3D convolutions if we can find an equivalent representation of the roto-translational feature space while preserving the translation properties. At this moment, the roto-translational feature space shape is [N, width, height, filters], where filters correspond to the real and imaginary parts of the Gabor filter. One possibility is to merge filters and angular axis to a shape of [width, height, filters * N]. With this transformation, the convolution part of the predictor could

be implemented as a 2D convolution applied to each translation. Unfortunately, the preliminary tests demonstrated that transforming the 4-dimensional space to a 3D space results in lower accuracy.

Nevertheless, we found a significant training time reduction when using 2D convolutions compared to the 3D convolution approach used in this work. These results mean that there exists a possibility of finding a representation of the roto-translational feature space in conjunction with 2D convolutions without accuracy loss. This solution might consist of using atrous convolutions or dilated convolutions, and further experiments are needed to find a solution.

Finally, the translating predictor implementation opens different possible routes for future works. In this work, we presented and used the translation predictor as multiple instances of identical predictors with attention to each translation. As these instances share weights, the network keeps its size lower compared with other approaches. The network could benefit from this approach using parallelization techniques to assign a processing thread to each translation. Another possibility exists in generating a single instance of this predictor and scanning each translation sequentially. In this sense, the network could be smaller as there not exist multiple instances of the predictor but would be slower due to the sequential operations.

List of research communications

Journals

[1] Rodriguez Salas, R.; Dokladal, P.; Dokladalova, E.
Rotation Invariant Networks for Image Classification for HPC and Embedded Systems.
Electronics 2021, 10, 139. https://doi.org/10.3390/electronics10020139

[2] Rodriguez Salas, R.; Dokladal, P.; Dokladalova, E.
A Minimal Model for Classification of Rotated Objects with Prediction of the Angle of Rotation,
Journal of Visual Communication and Image Representation 2021, Vol. 75,
https://doi.org/10.1016/j.jvcir.2021.103054

International Conferences

[3] R. Rodriguez, E. Dokladalova and P. Dokladal,
Rotation Invariant CNN Using Scattering Transform for Image Classification,
2019 IEEE International Conference on Image Processing (ICIP),
Taipei, Taiwan, 2019, pp. 654-658, doi: 10.1109/ICIP.2019.8804467.

[4] R. Rodriguez, E. Dokladalova and P. Dokladal,
A Minimal Model for Rotation Invariant Convolutional Neural Networks with Prediction of the Angle,
VISAPP 2021 16th Conference on Computer Vision Theory and Applications

Local communications

[5] R. Rodriguez, E. Dokladalova and P. Dokladal,
Rotation Equivariant CNNs for Image Classification
43ème journée ISS France, February 6, 2020, Paris, France

[6] R. Rodriguez, E. Dokladalova and P. Dokladal,
Rotation invariant NN for learning naturally un-oriented data
42ème journée ISS France, February 7, 2019, Paris, France

[7] R. Rodriguez, E. Dokladalova and P. Dokladal,
Rotation invariant NN for learning naturally un-oriented data

Collège Doctoral Franco-Allemand Workshop, 2018, Fontainebleau, France

[8] R. Rodriguez, Deep Learning: Trends and Challenges
Universidad Autonoma de Yucatan (UADY), 2018, Merida, Mexico

[9] R. Rodriguez, Artificial Intelligence Applications
Instituto Tecnologico de Merida, 2018, Merida, Mexico

References

A. Allan. Benchmarking machine learning on the new raspberry pi 4, model b, 2019a. URL https://www.hackster.io/news/benchmarking-machine-learning-on-the-new-raspberry-pi-4-model-b-88db9304ce4. 4, 5

A. Allan. Benchmarking the intel neural compute stick on the new raspberry pi 4, model b, 2019b. URL https://www.hackster.io/news/benchmarking-the-intel-neural-compute-stick-on-the-new-raspberry-pi-4-model-b-e419393f2f97. 5

L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021. 2

S. Arabi, A. Haghighat, and A. Sharma. A deep-learning-based computer vision solution for construction vehicle detection. *Computer-Aided Civil and Infrastructure Engineering*, 35(7):753–767, 2020. 7, 8

I. Ardiyanto, H. A. Nugroho, and R. L. B. Buana. Deep learning-based diabetic retinopathy assessment on embedded system. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1760–1763. IEEE, 2017. 9

S. Basu, A. Magesh, H. Yadav, and L. R. Varshney. Group equivariant neural architecture search via group decomposition and reinforcement learning. *arXiv preprint arXiv:2104.04848*, 2021. 35

V. Bonato, E. Marques, and G. A. Constantinides. A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE transactions on circuits and systems for video technology*, 18(12):1703–1712, 2008. 5

J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, pages 737–737, 1994. 45, 46, 47

J. Bruna and S. Mallat. Classification with scattering operators. In *CVPR 2011*, pages 1561–1566. IEEE, 2011. 63

J. Bruna and S. Mallat. Invariant Scattering Convolution Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1872–1886, 2013a. ISSN 01628828. doi: 10.1109/TPAMI.2012.230. 45, 61, 63

J. Bruna and S. Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013b. 64

H. Bura, N. Lin, N. Kumar, S. Malekar, S. Nagaraj, and K. Liu. An edge based smart parking solution using camera networks and deep learning. In *2018 IEEE International Conference on Cognitive Computing (ICCC)*, pages 17–24. IEEE, 2018. 10

A. Byun. Stanford cs class – cs231n: Convolutional neural networks for visual recognition, 2020. URL https://cs231n.github.io/convolutional-networks. 16

L. Capital. Number of total cameras predicted for 2022, 2017. URL www.ldv.co/insights/2017. 1

J. Chen, Y. Sato, and S. Tamura. Orientation space filtering for multiple orientation line segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(5):417–429, 2000. 23, 27, 101

S. Chen, Y. Liu, X. Gao, and Z. Han. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. In *Chinese Conference on Biometric Recognition*, pages 428–438. Springer, 2018. 12

W. Chen, H. Qiu, J. Zhuang, C. Zhang, Y. Hu, Q. Lu, T. Wang, M. Huang, X. Xu, et al. Quantization of deep neural networks for accurate edgecomputing. *arXiv preprint arXiv:2104.12046*, 2021. 8

X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2156, 2016. 2

Z. Chen, K. Chen, W. Lin, J. See, H. Yu, Y. Ke, and C. Yang. Piou loss: Towards accurate oriented object detection in complex environments. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *Computer Vision – ECCV 2020*, pages 195–211, Cham, 2020a. Springer International Publishing. ISBN 978-3-030-58558-7. 36

Z. Chen, K. Chen, W. Lin, J. See, H. Yu, Y. Ke, and C. Yang. Piou loss: Towards accurate oriented object detection in complex environments. In *European Conference on Computer Vision*, pages 195–211. Springer, 2020b. 104

D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649, 2012. ISBN 9781467312264. doi: 10.1109/CVPR.2012.6248110. 34

T. Cohen and M. Welling. Group equivariant convolutional networks. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/cohenc16.html. 92

T. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling. Gauge equivariant convolutional networks and the icosahedral CNN. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1321–1330, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/cohen19d.html. 31, 37, 73, 85

T. S. Cohen, M. Geiger, J. Koehler, and M. Welling. Spherical CNNs. In *ICLR*, pages 1 – 15, 2018. 31, 37, 73, 85, 102

C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition. *Training*, 100(101):102, 2017. 45

I. Dahi, M. Chikr El Mezouar, N. Taleb, and M. Elbahari. An edge-based method for effective abandoned luggage detection in complex surveillance videos. *Computer Vision and Image Understanding*, 158:141–151, 2017. doi: 10.1016/j.cviu.2017.01.008. 1

J. G. Daugman. Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. *JOSA A*, 2 (7):1160–1169, 1985. 77

J. Deng, J. Guo, N. Xue, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019. 45

J. Deng, U. Roh, J. Bao, Y. Suh, J. Choi, Y. Chen, V. Lin, J. Cheng, Z. Song, M. Cai, et al. 5g and ai integrated high performance mobile soc process-design co-development and production with 7nm euv finfet technology. In *2020 IEEE Symposium on VLSI Technology*, pages 1–2. IEEE, 2020. 6

L. Deng and D. Yu. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014. ISSN 1932-8346. doi: 10.1561/2000000039. URL http://dx.doi.org/10.1561/2000000039. 3

M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2148–2156, Red Hook, NY, USA, 2013. Curran Associates Inc. 8

S. Dieleman, J. D. Fauw, and K. Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *CoRR*, abs/1602.02660, 2016. URL http://arxiv.org/abs/1602.02660. 34

T. M. T. Do and D. Gatica-Perez. Where and what: Using smartphones to predict next locations and applications in daily life. *Pervasive and Mobile Computing*, 12:79–91, 2014. 6

D. A. Dyk and X. L. Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, mar 2001. ISSN 15372715. doi: 10.1198/10618600152418584. 20, 28, 29

C. Esteves, C. Allen-Blanchette, X. Zhou, and K. Daniilidis. Polar transformer networks. In *International Conference on Learning Representations*, pages 1 – 15, 2018. URL https://openreview.net/forum?id=HktRlUlAZ. 35

P. Follmann and T. Bottger. A rotationally-invariant convolution module by feature map back-rotation. In *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, volume 2018-Janua, pages 784–792. IEEE, mar 2018. ISBN 9781538648865. doi: 10.1109/WACV.2018.00091. 35, 37, 73, 74, 85, 87, 92, 102

W. T. Freeman and E. H. Adelson. The Design and Use of Steerable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991. ISSN 01628828. doi: 10.1109/34.93808. 21, 31, 32, 33, 66, 67

D. Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers-Part III: Radio and Communication Engineering*, 93(26):429–441, 1946. 77, 78

C. Gao, A. Gutierrez, M. Rajan, R. G. Dreslinski, T. Mudge, and C.-J. Wu. A study of mobile device utilization. In *2015 ieee international symposium on performance analysis of systems and software (ispass)*, pages 225–234. IEEE, 2015. 12

L. Gao, H. Li, Z. Lu, and G. Lin. Rotation-equivariant convolutional neural network ensembles in image processing. In *Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, UbiComp/ISWC '19 Adjunct, pages 551–557, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6869-8. doi: 10.1145/3341162.3349330. URL http://doi.acm.org/10.1145/3341162.3349330. 34, 73, 85

Y. Ge, R. Zhang, X. Wang, X. Tang, and P. Luo. Deepfashion2: A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5337–5345, 2019. 4

R. Gens and P. M. Domingos. Deep symmetry networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2537–2545. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5424-deep-symmetry-networks.pdf. 34

R. Ghosh and A. K. Gupta. Scale steerable filters for locally scale-invariant convolutional neural networks. *arXiv preprint arXiv:1906.03861*, 2019. 103

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 14

Google. Solutions for on-device intelligence, 2020. URL https://coral.ai. 4

Y. Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018. 9

M. Haldar, M. Abdool, P. Ramanathan, T. Xu, S. Yang, H. Duan, Q. Zhang, N. Barrow-Williams, B. C. Turnbull, B. M. Collins, et al. Applying deep learning to airbnb search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1927–1935, 2019. 7

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015. 34

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 18, 19, 89

G. E. Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009. 3

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. doi: 10.1016/0893-6080(89)90020-8. 16

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 18

G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 45, 47

A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3617–3635. IEEE, 2019. 6

Intel. Intel® movidius™ neural compute sdk documentation, 2019. URL https://movidius.github.io/ncsdk/. 4

C. Intel. Openvino™ toolkit benchmark, 2021a. URL https://docs.openvinotoolkit.org/latest/openvino_docs_performance_benchmarks_openvino.html. 5

C. Intel. Openvino™ toolkit overview, 2021b. URL https://docs.openvinotoolkit.org/latest/index.html. 5

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 18

M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf. 34

A. K. Jain, N. K. Ratha, and S. Lakshmanan. Object detection using gabor filters. *Pattern recognition*, 30(2):295–309, 1997. 78

Kaggle. Dogs vs cats, 2014. URL https://www.kaggle.com/c/dogs-vs-cats. 22

A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018. 103

O. S. Kayhan and J. C. v. Gemert. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14274–14285, 2020. 27

I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller, and E. Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4873–4882, 2016. 12

G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015. 47

T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990. 54

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1907 – 1105. Curran Associates, Inc., 2012. 17, 21, 28

N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and Simile Classifiers for Face Verification. In *IEEE International Conference on Computer Vision (ICCV)*, Oct 2009. 45

N. D. Lane and P. Georgiev. Can deep learning revolutionize mobile sensing ? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 117–122, 2015. 11

D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 289–297, 2016. 33, 37, 74

Y. LeCun and C. Cortes. MNIST handwritten digit database. *http://yann.lecun.com/exdb/mnist/*, 2010. 59, 60

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 2

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 16, 17

C.-J. Lee and S.-D. Wang. Fingerprint feature extraction using gabor filters. *Electronics Letters*, 35(4):288–290, 1999. 78

J. Li, Z. Yang, H. Liu, and D. Cai. Deep Rotation Equivariant Network. *Neurocomputing*, 290:26–33, 2018a. ISSN 18728286. doi: 10.1016/j.neucom.2018.02.029. 34

J. Li, X. Long, S. Hu, Y. Hu, Q. Gu, and D. Xu. A novel hardware-oriented ultra-high-speed object detection algorithm based on convolutional neural network. *Journal of Real-Time Image Processing*, pages 1–12, 2019. 5

S.-A. Li, W.-Y. Wang, W.-Z. Pan, C.-C. J. Hsu, and C.-K. Lu. Fpga-based hardware design for scale-invariant feature transform. *IEEE Access*, 6:43850–43864, 2018b. 5

T. Liang, J. Glossner, L. Wang, and S. Shi. Pruning and quantization for deep neural network acceleration: A survey. *arXiv preprint arXiv:2101.09671*, 2021. 8

L. Lin and N. Purnell. A world with a billion cameras watching you is just around the corner, 2019. URL https://www.wsj.com/articles/a-billion-surveillance-cameras-forecast-to-be-watching-within-two-years-11575565402. 1

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 5

J. Liu. In your face: China's all-seeing state, Dec 2017. URL https://www.bbc.com/news/av/world-asia-china-42248056. 1

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 45

S. Luan, c. chen, B. Zhang, j. han, and J. Liu. Gabor convolutional networks. *IEEE Trans. Image processing.*, 2018. 33, 37, 74

R. Madaan, D. Maturana, and S. Scherer. Wire detection using synthetic data and dilated convolutional networks for unmanned aerial vehicles. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3487–3494. IEEE, 2017. 10

H. Maeda, Y. Sekimoto, T. Seto, T. Kashiyama, and H. Omata. Road damage detection and classification using deep neural networks with smartphone images. *Computer-Aided Civil and Infrastructure Engineering*, 33 (12):1127–1141, 2018. doi: https://doi.org/10.1111/mice.12387. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12387. 6

T. Manderson, J. C. G. Higuera, R. Cheng, and G. Dudek. Vision-based autonomous underwater swimming in dense coral for combined collision avoidance and target selection. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1885–1891. IEEE, 2018. 10

D. Marcos, M. Volpi, N. Komodakis, and D. Tuia. Rotation Equivariant Vector Field Networks. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-Octob, pages 5058–5067, 2017. ISBN 9781538610329. doi: 10.1109/ICCV.2017.540. 22, 34, 37, 74

V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge. Real-time apple detection system using embedded systems with hardware accelerators: an edge ai application. *IEEE Access*, 8:9102–9114, 2020. 4

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943. doi: 10.1007/bf02478259. 2, 14, 15

D. D. McManus, J. Lee, O. Maitas, N. Esa, R. Pidikiti, A. Carlucci, J. Harrington, E. Mick, and K. H. Chon. A novel application for the detection of an irregular pulse using an iphone 4s in patients with atrial fibrillation. *Heart Rhythm*, 10 (3):315–319, 2013. 6

J. Morlet, G. Arens, E. Fourgeau, and D. Glard. Wave propagation and sampling theory—part i: Complex signal and scattering in multilayered media. *Geophysics*, 47(2):203–221, 1982. 61

NVIDIA. Jetson nano developer kit, 2021a. URL https://developer.nvidia.com/embedded/jetson-nano-developer-kit. 4

C. NVIDIA. Jetson benchmarks, 2021b. URL https://developer.nvidia.com/embedded/jetson-benchmarks. 4

S. NXP. Rd-imx6q-sabre: Sabre board for smart devices based on the i.mx 6quad applications processors, 2021. URL https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/sabre-board-for-smart-devices-based-on-the-i-mx-6quad-applications-processors:RD-IMX6Q-SABRE. 9

S. O'Dea. Number of smartphone users worldwide from 2016 to 2021, 2020. URL https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. 1

S. O'Gara and K. McGuinness. Comparing data augmentation strategies for deep image classification. In *IMVIP*. Technological University Dublin, 2019. 7, 29

E. C. Orenstein, O. Beijbom, E. E. Peacock, and H. M. Sosik. Whoi-plankton-a large scale fine grained visual recognition benchmark dataset for plankton classification. *arXiv preprint arXiv:1510.00745*, 2015. 71

Z. W. Pan Zhong. Characterization and design of generalized convolutional neural network. In *Anual Conference in Information Sciences and Systems (CISS)*, pages 1–6. IEEE, 2019. 34

R. Pi. Raspberry pi computers and microcontrollers, 2021. URL https://www.raspberrypi.org/products/. 4

Qualcomm. Qualcomm® snapdragon™ 888 5g mobile platform, 2020. URL https://www.qualcomm.com/media/documents/files/qualcomm-snapdragon-888-mobile-platform-product-brief.pdf. 11

A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*, 2021a. 2

A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-shot text-to-image generation, 2021b. 13

J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 13

Y. Ren, C. Zhu, and S. Xiao. Small object detection in optical remote sensing images via modified faster r-cnn. *Applied Sciences*, 8(5):813, 2018. 2

A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE high performance extreme computing conference (HPEC)*, pages 1–9. IEEE, 2019. 5

S. Rivas-Gomez, A. J. Pena, D. Moloney, E. Laure, and S. Markidis. Exploring the vision processing unit as co-processor for inference. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 589–598, 2018. doi: 10.1109/IPDPSW.2018.00098. 5

R. Rodriguez Salas, E. Dokladalova, and P. Dokládal. Rotation invariant cnn using scattering transform for image classification. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 654–658. IEEE, 2019. 37, 73, 74, 85

R. Rodriguez Salas, P. Dokládal, and E. Dokladalova. A minimal model for classification of rotated objects with prediction of the angle of rotation. *Journal of Visual Communication and Image Representation*, 75:103054, 2021a. 37, 73, 74, 85

R. Rodriguez Salas, P. Dokladal, and E. Dokladalova. Rotation invariant networks for image classification for hpc and embedded systems. *Electronics*, 10(2),

2021b. ISSN 2079-9292. doi: 10.3390/electronics10020139. URL https://www.mdpi.com/2079-9292/10/2/139. 85, 92

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519. 15

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 14

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. 20

I. Sa, Z. Chen, M. Popović, R. Khanna, F. Liebisch, J. Nieto, and R. Siegwart. weednet: Dense semantic weed classification using multispectral images and mav for smart farming. *IEEE Robotics and Automation Letters*, 3(1):588–595, 2017. 9

A. K. Saydjari and D. P. Finkbeiner. Equivariant wavelets: Fast rotation and translation invariant wavelet scattering transforms. *arXiv preprint arXiv:2104.11244*, 2021. 33, 37

M. R. Schädler, B. T. Meyer, and B. Kollmeier. Spectro-temporal modulation subspace-spanning filter bank features for robust automatic speech recognition. *The Journal of the Acoustical Society of America*, 131(5):4134–4151, 2012. 78

F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 45

P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *The 2011 International Joint Conference on Neural Networks*, pages 2809–2813. IEEE, 2011. 18

R. Shanmugamani. *Deep Learning for Computer Vision*. Packt Publishing, O'Reilly Media, Inc, 1 2018. ISBN 9781788295628. 2

W. Shi and S. Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016. 3

W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016. 3

C. Shin and J. Yun. Deep rotating kernel convolution neural network. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 441–442. IEEE, 2019. 73, 74, 85

L. Sifre and S. Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1233–1240, 2013. doi: 10.1109/CVPR.2013.163. 33

P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3. Citeseer, 2003. 28

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 17

S. Singh, A. Aggarwal, and R. Dhir. Use of gabor filters for recognition of handwritten gurmukhi character. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(5), 2012. 78

S. Sivanantham, E. Dokladalova, P. Dokladal, and V. Biri. Reconnaissance faciale pour les bases de donnees reduites. *Internship report (Unpublished)*, 2017. 45

A. A. Süzen, B. Duman, and B. Şen. Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5. IEEE, 2020. 4

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 18

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014. 45

M. Tan and Q. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/tan19a.html. 19, 20

V. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. 98

J. Velasco, C. Pascion, J. W. Alberio, J. Apuang, J. S. Cruz, M. A. Gomez, B. Molina Jr, L. Tuala, A. Thio-ac, and R. Jorda Jr. A smartphone-based skin disease classification using mobilenet cnn. *arXiv preprint arXiv:1911.07929*, 2019. 12

M. Weiler, F. A. Hamprecht, and M. Storath. Learning Steerable Filters for Rotation Equivariant CNNs. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 849–858, 2018. ISBN 9781538664209. doi: 10.1109/CVPR.2018.00095. 21, 29, 33, 34, 37, 74

R. Wiersma, E. Eisemann, and K. Hildebrandt. Cnns on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (TOG)*, 39(4):92–1, 2020. 21

D. Worrall. Mnist-rot experiments, 2017. URL https://github.com/deworrall92/harmonicConvolutions/tree/master/MNIST-rot. 28, 70

D. Worrall and G. Brostow. CubeNet: Equivariance to 3D Rotation and Translation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11209 LNCS, pages 585–602, 2018. ISBN 9783030012274. doi: 10.1007/978-3-030-01228-1_35. 31

D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic Networks: Deep translation and rotation equivariance. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7168–7177, 2017. ISBN 9781538604571. doi: 10.1109/CVPR.2017.758. 21, 30, 31, 37, 74, 103

C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, et al. Machine learning at facebook: Understanding inference at the edge. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 331–344. IEEE, 2019. 12

Xilinx. Fpgas & 3d ics, 2021. URL https://www.xilinx.com/products/silicon-devices/fpga.html. 4

M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, pages 2125–2136, 2019. 11, 12

H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 127–134, 2018. 16

S. I. Young, W. Zhe, D. Taubman, and B. Girod. Transform quantization for cnn compression. *arXiv preprint arXiv:2009.01174*, 2020. 8

M. M. T. Zadeh, M. Imani, and B. Majidi. Fast facial emotion recognition using convolutional neural networks and gabor filters. In *2019 5th Conference on Knowledge Based Engineering and Innovation (KBEI)*, pages 577–581. IEEE, 2019. 78

X. Zhang, L. Liu, Y. Xie, J. Chen, L. Wu, and M. Pietikäinen. Rotation Invariant Local Binary Convolution Neural Networks. In *Proceedings - 2017*

IEEE International Conference on Computer Vision Workshops, ICCVW 2017, volume 2018-Janua, pages 1210–1219, 2018. ISBN 9781538610343. doi: 10.1109/ICCVW.2017.146. URL http://www.outex.oulu.fi/. 33, 34, 37

Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao. Oriented response networks. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January, pages 4961–4970, 2017. ISBN 9781538604571. doi: 10.1109/CVPR.2017.527. 33, 34, 37, 73, 74, 85, 92

Y. Zhou, J. Shi, X. Yang, C. Wang, S. Wei, and X. Zhang. Rotational objects recognition and angle estimation via kernel-mapping cnn. *IEEE Access*, 7:116505–116518, 2019. 35, 37, 103

Z. Zhou, Y. Neo, K.-S. Lui, V. W. Tam, E. Y. Lam, and N. Wong. A portable hong kong sign language translation platform with deep learning and jetson nano. In *The 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–4, 2020. 4

Abstract

During the last years, the number of CNN-based applications for computer vision has grown exponentially. This success is related to their performances and efficient implementations. Nevertheless, their computing requirements remain high and limiting.

In this work, we focus on rotation-invariant classification. The main objective is to reduce the size of the model without sacrificing the accuracy. We propose an original CNN architecture where the first layer consists of an ordered set of filters. We evaluate several filter families such as scattering transform, steerable, and Gabor filters. We also provide proof of the trainability of this architecture.

Finally, the proposed rotation invariance framework allows achieving state-of-the-art error rates on simple datasets and outperforming previous approaches on more complex datasets while reducing the network size by more than 50%. These results are obtained without data-augmentation techniques and with an angular prediction capability.

Keywords: Image classification, CNN, rotation invariance, prediction of angle of rotation, scattering transform, steerable filters, Gabor filters.

Résumé

Au cours des dernières années, le nombre d'applications des CNNs pour la vision par ordinateur a augmenté exponentiellement. Cela est lié à leurs performances et à leurs implémentations efficaces. Néanmoins, leurs coûts de calcul restent élevés.

Dans ce travail, nous nous concentrons sur la classification invariante par rotation. L'objectif principal est de réduire ainsi la taille du modèle sans sacrifier sa précision. Nous proposons une architecture CNN originale où la première couche consiste en un ensemble ordonné de filtres. Nous évaluons plusieurs familles de filtres : la scattering transform, les steerable filtres et les filtres de Gabor. Nous fournissons également les preuves qu'une telle architecture peut être entraînée.

Nous démontrons les résultats très compétitifs les datasets simples et complexes avec taille du réseau réduite de plus de 50 %. Ces résultats sont obtenus sans techniques d'augmentation des données et avec une capacité de prédiction de l'angle de rotation.

Keywords: Classification d'image, CNN, classification invariante par rotation, prédiction de l'angle de rotation, scattering transform, steerable filtres, filtres de Gabor.