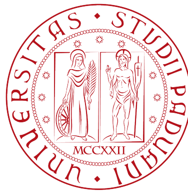


University of Padova

Department of Psychology

Bachelor's Thesis in Psychological Science

Augmenting Convolutional Neural Networks with Kernels Inspired by the Early Visual System

Supervisor
Prof. Alberto Testolin

Bachelor's Candidate
Matteo Bruno Rovoletto

Student ID
1221346

Academic Year
2021-2022

UNIVERSITAS STUDII PADUANI
MCCXXII

Abstract

Early neural networks were inspired by biology: the McCulloch-Pitts neuron, the Perceptron and the Neocognitron were all attempting to imitate the functioning of the brain. [8] To explore ways in which the study of neurology can aid the development of Artificial Intelligence solutions, we describe the research "**On-Off Center-Surround Receptive Fields for Accurate and Robust Image Classification**" (by Babaiee et.al.[13]), in which a Convolutional Neural Network (CNN) is improved with kernels inspired by the On-Off-Center-Surround (OOCS) receptive fields of the vertebrate retina. We then replicate their result and test some variants of their models, proving that the addition of the Off-Center component is redundant.
Finally, we illustrate our experiments, devised to expand on Babaiee's research: with the premise that kernels inspired by the retina were able to improve the performance of the algorithm, we implement new kernels inspired by the early visual system. Specifically, we made kernels that reproduce the function of simple cells of the area V1 of the visual cortex, and kernels that try to replicate some functions of the complex cells of visual area V2. We tested several CNNs that implement these kernels, and also some CNNs that receive as input only the On-Center component plus a downscaled version of the image.
These networks did not achieve a better performance than Babayee's OOCS-CNN, but some did improve upon the CNN used as control.
However, our tests were limited in scope and hence the basic idea may still be implemented successfully. We will therefore describe possible solutions that could be tested in future research.

Keywords: Neural Networks, Visual System, Computer Vision, AI, ImageNet

UNIVERSITAS STUDII PADUANI
MCCCXXII

Contents

Abstract i

Contents iii

1 Introduction to Artificial Neural Networks 1
- 1.1 Early History . . . 1
- 1.2 Convolutional Neural Networks and Bio-Inspired models . . . 3
- 1.3 Modern Convolutional Neural Networks . . . 5

2 Prior Research 11

3 Methods 15
- 3.1 Models' architectures . . . 17

4 Results 25

5 Conclusions and Further Research 31

Bibliography 33

Appendix 35

List of Figures 37

List of Tables 39

Acknowledgements 41

UNIVERSITAS STUDII PADUANI
MCCXXII

1 | Introduction to Artificial Neural Networks

In recent years one of the fastest-growing fields in computer science has been the field of Artificial Intelligence (AI). The ability for computers to recognize objects, find patterns in big datasets and act intelligently without being directly programmed is being applied in many areas, from self-driving vehicles to agriculture. This is happening now because of the increase in the processing speed of computers and because the vast availability of big datasets finally made the training of Artificial Neural Networks (ANNs) easily achievable. ANNs are computing systems, originally inspired by the biological neural networks that constitute animal brains. Therefore, the fields of AI, neuroscience and psychology are highly relevant to each other.

1.1. Early History

The origins of today's Neural Networks can be found in the work of Walter Pitts and Warren McCulloch, who in 1943 published *"A logical calculus of the ideas immanent in nervous activity"* in the Bulletin of Mathematical Biophysics [11]. In this paper, they explain how they tried to understand the functioning of the brain by creating a model of many artificial neurons connected together. They were the first scientists to create an artificial model of a neuron, and hence their work has made an important contribution to the development of modern Artificial Neural Networks (ANNs).
They were inspired by the following principles, mostly derived from neurological theories of the time:

1. The activity of the neuron is an "all-or-none" process.
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron.
3. The only significant delay within the nervous system is a synaptic delay.
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
5. The structure of the net does not change with time.

[11]

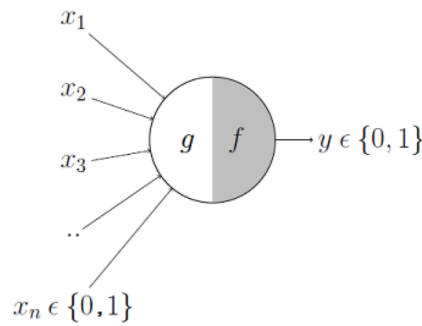


Figure 1.1: McCulloch-Pitts Artificial Neuron

Their model is called **McCulloch-Pitts Neuron** (MCPN) or "Threshold Logic Unit" (see Fig. 1.1). It uses as inputs and outputs only boolean values (i.e 0 or 1), which can be inhibitory or excitatory. First, all the inputs are summed ('g'). Then the neuron outputs 1 only if g reaches a certain threshold value. McCulloch–Pitts units can be used to build networks capable of computing any logical function. [8]
Another relevant development was the publication in 1949 of *"The Organization of Behavior"* by Donald Hebb. In this book, he attempts to connect the psychological and neurological underpinnings of learning. He introduces **Hebb's Learning Rule**, which states that: *"a presynaptic neuron A, if successful in repeatedly activating a postsynaptic neuron B when itself (neuron A) is active, will gradually become more effective in activating neuron B."* This is sometimes stated colloquially as "neurons that fire together, wire together" [3]. This theory eventually found more empirical confirmation when the mechanisms of **Long Term Potentiation** were discovered in the brain.
Inspired by Hebb's rule and by the MCPN, in 1957 Frank Rosenblatt created the **Perceptron Algorithm**, which is known as the first Artificial Neural Network (see Fig. 1.2).

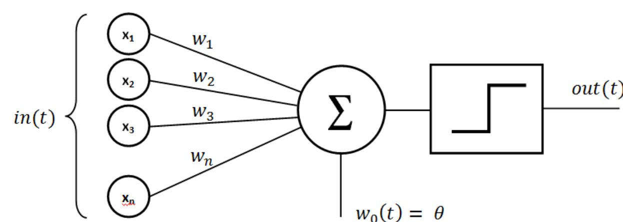


Figure 1.2: Simplest Perceptron

It is similar to the MCPN, but it can take continuous values as input, and every input is multiplied by a learnable weight. Most importantly, Rosenblatt ideated a supervised learning algorithm that enabled the artificial neuron to figure out the correct weights directly from examples given as training data.
The major limitation of the MCPN and the Perceptron is that they are only able to distinguish between linearly separable classes [8]. To solve non-linear problems more neuronal layers would be needed, but at the time it would have been too computationally expensive: just to train the Perceptron Rosenblatt had to build a custom mechanical calculator (the Mark I Perceptron machine) because a

computer of the time would not have had enough processing speed.[9]
Many innovations brought from these models to the NNs of today. The adoption of the backpropagation algorithm and of the sigmoid activation function have been particularly relevant, since they allowed learning to be implemented on deeper NNs.
I am not going to illustrate here all the advances that brought to today's state of the art of AI, but I will talk about the functioning of Convolutional Neural Networks and some bio-inspired NNs, since they constitute a necessary background to understand the scope of the research I am going to describe.

1.2. Convolutional Neural Networks and Bio-Inspired models

The state of the art in AI for image classification are now Convolutional Neural Networks (CNNs). They derive their name from their use of **convolutional layers**.
We can visualize a convolutional layer as many small square templates, called **kernels**, which slide over the image and look for patterns. Where a part of the image matches the kernel's pattern, the kernel returns a large positive value, and when there is no match, the kernel returns zero or a smaller value.

The first CNNs were inspired by biology, and in particular by the Nobel Prize work of **Hubel and Wiesel** (1959, 1962) [2]. Based on experiments on cats' striate cortex (V1), they described a circuit model with **simple cells** and **complex cells**. They found that simple cells in V1 respond to lines with a specific orientation. Complex cells have a similar response to simple cells, but they have larger receptive fields and they are more location invariant.
After that, many biologically inspired computational models for visual recognition were proposed, including the Neocognitron. [12]

The Neocognitron

The **Neocognitron** was proposed by Kunihiko Fukushima in 1980.

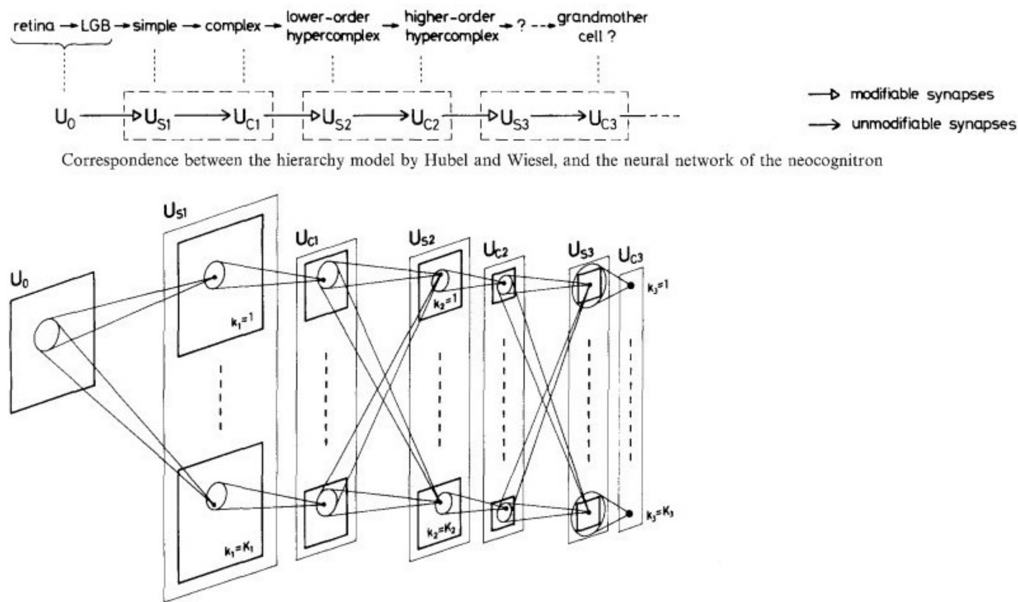


Figure 1.3: Neocognitron architechture

It consists of a cascade connection of several modular structures U_n preceded by an input layer U_0. Each U structure is composed of an S layer and a C layer. Cells in the S layers correspond to the simple edge-detector cells of the primary visual cortex (V1) as described by Hubel and Wiesel; only the weights that end in S cells are learnable. Each S cell is connected to all the cells in one particular area (receptive field) of the previous layers. With training, one set of weights is strengthened, and therefore every S cell becomes sensitized to one particular feature. S cells are connected to cells in the C layers through fixed connections and weights; each C cell is connected to several S cells that are activated by the same feature. This allows the same C cell to be equally activated by the same feature present in different positions of U_0 (see Fig. 1.3), giving the Neocognitron a degree of **location invariance**. [4]

The HMAX Model

Another early CNN model that was inspired by Hubel and Wiesel's research is the **HMAX model**, created in 1999 by Maximilian Riesenhuber and Tomaso Poggio. Its purpose was to provide insights into the functioning of the visual system, and it was also inspired by the experimental data of Logothetis et al. [7] on the invariance properties and shape tuning of neurons in the macaque inferotemporal cortex.
These are the main ideas behind the HMAX model:

- "Immediate" visual processing is feedforward and hierarchical: low levels detect simple features, which are combined hierarchically into **increasingly complex features** to be detected.
- Layers of the hierarchy alternate between "sensitivity" (to particular features) and. "invariance" (to position, scale, orientation)

- The size of receptive fields increases along the hierarchy.
- The degree of invariance increases along the hierarchy.

Therefore, like the Neocognitron, HMAX is constituted by **alternating layers** consisting of simple(S) units and complex(C) units. However, S cells are not confined to particular receptive fields and do not learn their preferred features. Instead, the input is convoluted by a series of kernels inspired by simple cells (see Fig. 1.4).

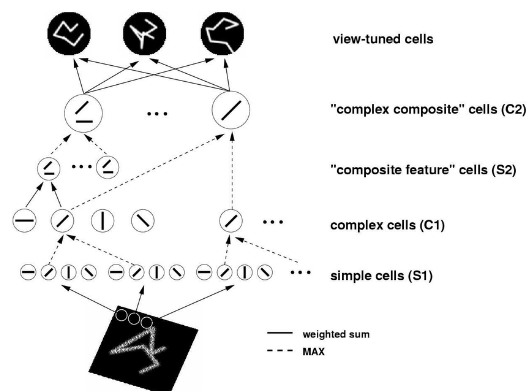


Figure 1.4: HMAX model

The kernels are Gabor filters 5x5 pixels in size. They are available in 4 different angles corresponding to 4 different line orientations. These filters are applied across five different sizes of the input image, and therefore S1's output will be composed of 4 x 5 = 20 images [5].
The role of the C layers is to give the model **position and size invariance**. While the Neocognitron's C cells utilized fixed connections to similar S cells to achieve invariance, HMAX uses an approach that prevails in modern CNNs: it relies on a **max-pooling** operation. A kernel (of 7x7 pixel size for HMAX) slides through the image and gives as output to the following layer the maximum value in its grid. This process removes the variable of spatial position from the data (i.e. it gives position-invariance to the model), especially when iterated multiple times.
Kernels in S2 correspond to a combination of the features selected in C1. The same operation of C1 is repeated in C2. [12]

This model reaches good levels of accuracy in image recognition, and more recent versions are capable of achieving close to human-level performance on several rapid object recognition tasks [10]

1.3. Modern Convolutional Neural Networks

We have seen that early CNNs were inspired by biology. However, modern CNNs managed to reach incredible levels of accuracy by implementing solutions that were not directly inspired by the human visual system.
To effectively explain their functioning we will use feature maps (FM) and lines of code from the experiment we will describe in the 3rd chapter.
There are 4 important elements in modern CNNs: convolutional layers, activation functions, pooling layers, and linear layers.

Convolutional Layers

This is how a Convolutional layer is defined in Python (using the Pytorch extension):

*Conv_1=nn.Conv2d(**in_channels**= 3, **out_channels**= 32, **kernel_size**= 3, **stride**= 1)*

Note the parameters that have to be defined:

- **In_channels** specifies the number of channels of the input given to the convolutional layer. Since this is the first layer, its input will be the original image to be classified.
  An image is encoded as a matrix of size CxWxL. W (width) and L (length) correspond to the 2-D image dimensions, while the C (channel) dimension is used for the color encoding: a colored RGB image has 3 channels, corresponding to the Red Green and Blue intensity values. This parameter is important since the kernel that is going to be convoluted with the image has to have the same number of channels as the input given to its layer.
- **Kernel_size** specifies the width and length of the kernel in pixels.
- **Out_channels** specifies the number of images (i.e. feature maps) that are going to be created by the convolution operation. Each of these outputs is the result of a convolution with a different kernel.
- **Stride** specifies the step the kernel takes after every operation in the convolution.

In modern CNNs, the kernels are initialized with random values. With training, these values are changed by the backpropagation algorithm until they are efficient in highlighting important features of the input image. Every convolution between the image and a kernel results in a feature map, which is a 1-channel image: in this example, for each step of a kernel its 3x3x3 value matrix is multiplied by the values of a 3x3x3 section of the input image (see Fig. 1.5). The sum of the resulting values becomes one pixel of the resulting feature map, which will have only 1 channel.

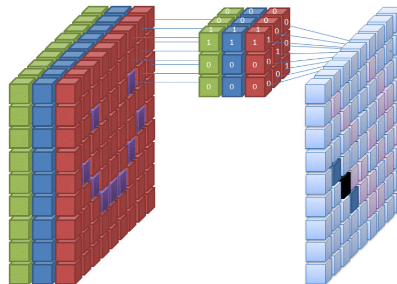


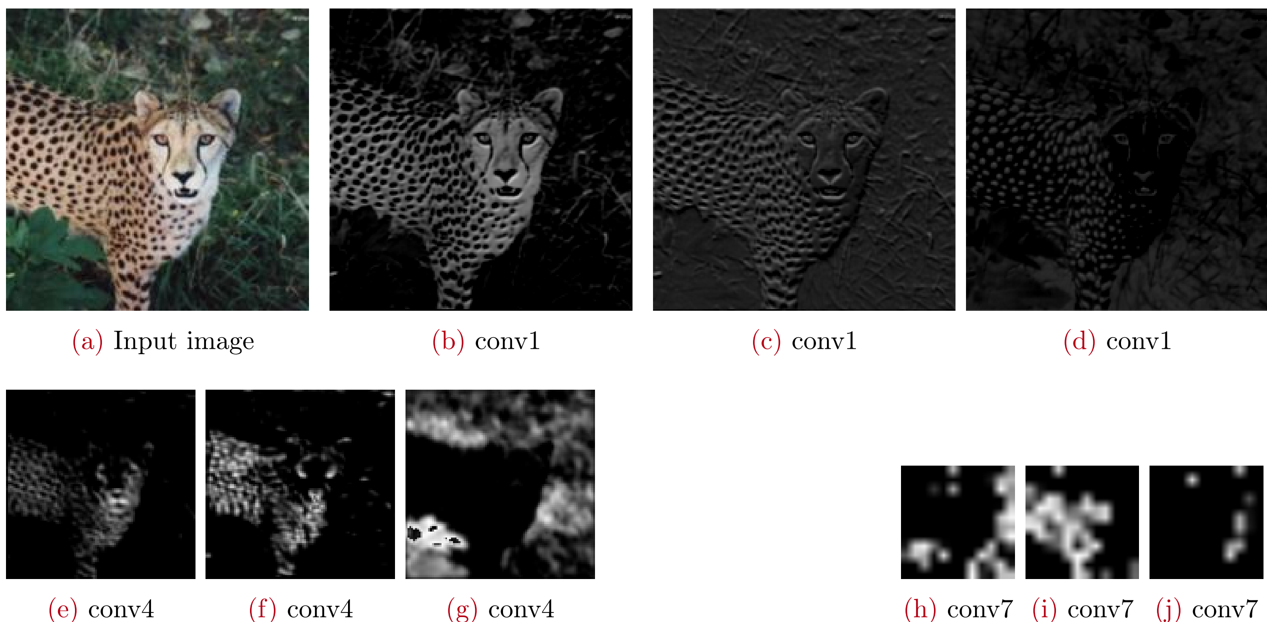
Figure 1.5: Kernel convolution operation

Since the kernel has a stride of 1, after each of these operations it will slide 1 pixel to the right. When it reaches the right edge of the image, it returns to the left border and slides down 1 pixel. The resulting feature map will have a similar dimension to the input image. It can be calculated with:

$$Output_size = \frac{Input_size - Kernel_size}{stride} + 1$$

Each convolutional layer can have many different kernels: one for every feature map it needs to produce as output. In this case, it will have 32 different kernels that will create 32 feature maps.
The output of this layer (Conv_1) will have a shape of 32*image_width*image_length; the 32 feature maps are stacked as image channels. This will become the input to the following layer, which therefore will need the parameter in_channels= 32.

Here are some examples of feature maps, i.e. the outputs of convolutional layers at various depths in the CNN (Fig. 1.6):



(a) Input image (b) conv1 (c) conv1 (d) conv1

(e) conv4 (f) conv4 (g) conv4 (h) conv7 (i) conv7 (j) conv7

Figure 1.6: Examples of feature maps

Activation Function

In Pytorch, a Neural Network is usually defined in 2 steps: first, the layers are defined, then the "forward pass", where we determine the path the input will take through the layers. In the forward pass, we can apply an **activation function** to the layer's output.

An activation function is the mathematical transformation a neuron applies to its output. As we have previously seen, early artificial neurons like the MPCN (being biologically inspired) used a **threshold all-or-none operation**. We can say this threshold function was their activation function.

The threshold function is linear, and it has been found that for a NN algorithm to carry out complex tasks a **non-linear activation function** is needed. If we only allow linear activation functions in a neural network, the output will just be a linear transformation of the input. Such a network can just be represented as a matrix multiplication, and it would not be able to operate as a universal function approximator [6]. Hence, modern NNs use non-linear activation functions such as the **Relu (Rectified Linear Unit)** function. Relu is a very simple operation: it turns to 0 every negative value, and it leaves positive values unchanged (see Fig. 1.7). In most CNNs it is applied to the output of convolutional and linear layers.

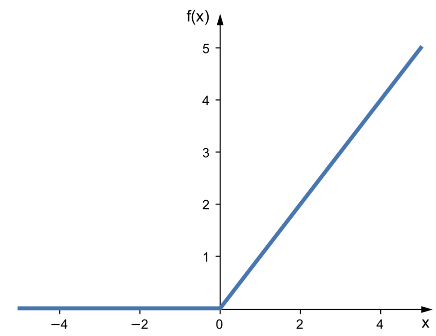
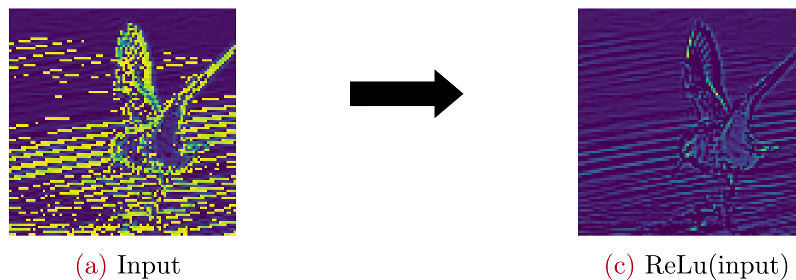


Figure 1.7: The ReLu activation function

Therefore in the forward pass of our CNN in Pytorch we will embed the output of a convolutional layer inside a Relu activation function, using the following line of code:

Output = F.relu(self.Convolution_1(image))

Here is an example (Fig. 1.8):



(a) Input (c) ReLu(input)

Figure 1.8: The ReLu function applied to an image

Pooling Layers

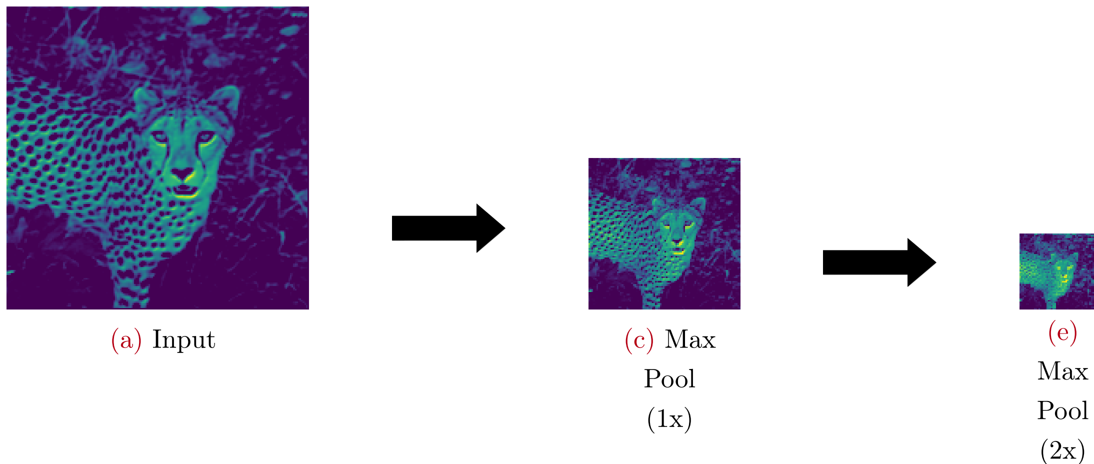
Pooling layers are needed to reduce the size of the image (and hence reduce the computations needed in the deeper layers) and to make the network able to distinguish image features with position and size invariance.

The most commonly used pooling method is the **MaxPool operation**, i.e. the same operation used in the HMAX model previously described. A custom size kernel slides across the input, selecting as output the pixel with the highest value. It usually passes through the input image with a stride of 2 and a kernel of 2*2 pixels. The resulting feature will be half the size of the input image.

This is the code to apply such a MaxPool operation in Pytorch's forward pass:

MaxPool_output = F.max_pool2d(input, kernel_size= 2, stride= 2)

Here is an example (Fig. 1.9):



(a) Input

(c) Max Pool (1x)

(e) Max Pool (2x)

Figure 1.9: MaxPool operation applied to an image

Linear Layers

Linear Layers (or fully connected layers) are usually found at the end of a CNN. Their function is to transform the last series of feature maps into the output (which for an image-classifier is the class of the object in the image).

Before passing the feature maps to the linear layers, it is necessary to "flatten" them: from a three-dimensional matrix they are transformed into a **one-dimensional** series of values. The first linear layers will need to have as many input channels as the number of flattened values. Each linear layer is "**fully connected**" to its adjacent layers, which means that each node has a weight that connects it to every other node in the adjacent layers (see Fig. 1.10).

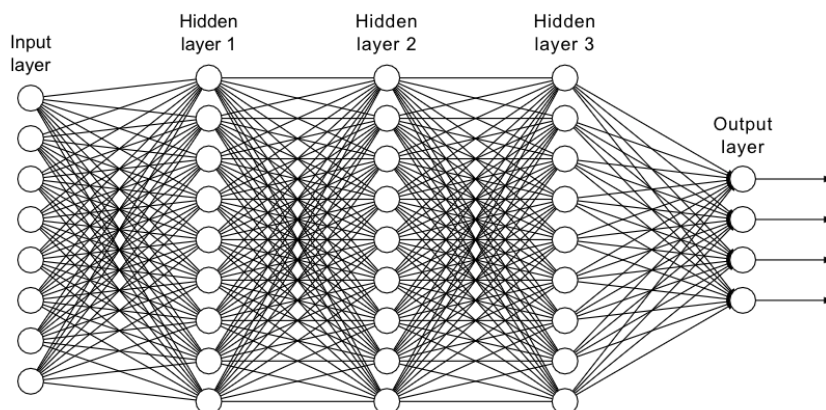


Figure 1.10: 5 linear layers

The last layer in the network is the output layer, which needs to have as many output channels as the possible categories the image can be classified as.

This is how a linear layer is instantiated in PyTorch's forward pass:

Linear_layer_1 = nn.Linear(in_features= 18432, out_features= 512)

A CNN is made up of all these elements, which can vary in position and number. Modern CNNs are many layers deep. Here is **VGG-19**, which was the state of the art for image classification in 2014 (see Fig. 1.11).

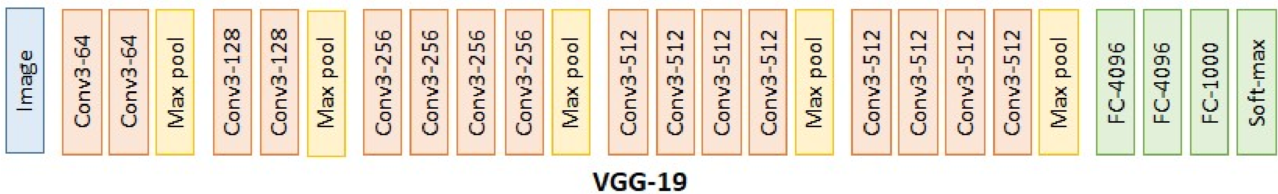


Figure 1.11: VGG-19 architecture

We can see it has 16 convolutional layers and 3 linear layers. Modern CNN can have more than 100 layers.

2 | Prior Research

While modern CNNs are successfully evolving as a separate field, it is safe to affirm that AI and neurology will continue to positively influence each other. For example, in the research "**On-Off Center-Surround Receptive Fields for Accurate and Robust Image Classification**" [13], insights from the functioning of the vertebrate retina are applied to improve the accuracy of modern CNN architectures.

Introduction

The technology of CNNs has already been inspired by receptive fields in the retina. A receptive field defines the region of visual space within which visual stimuli affect the firing of a single ganglial neuron; this pattern is preserved by neurons in the visual cortex. However, the retina utilizes many other artifacts for visual processing. Another important motif is the center-surround (CS) organization: the receptive field of a ganglial neuron is divided into a circular excitatory region (the center), and a concentric inhibitory region (the surround). These center-surround fields can be either on-center (when neurons fire in response to light in the center and are inhibited by light in the surround) or off-center (when neurons fire in response to light in the surround and are inhibited by light in the center). **Babayee et. al.** created a CNN inspired by these concepts.

Materials and Methods

Kernels

To compute the On-Off-center-surround (OOCS) kernels, they used a Difference-of-Gaussians (DoG) function. This allowed them to force positive and negative weights to sum up to 1 and -1, and to avoid normalization issues (Fig. 2.1).

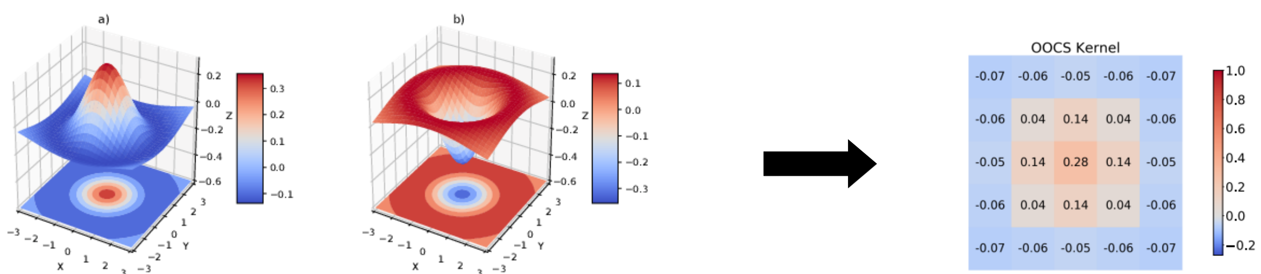
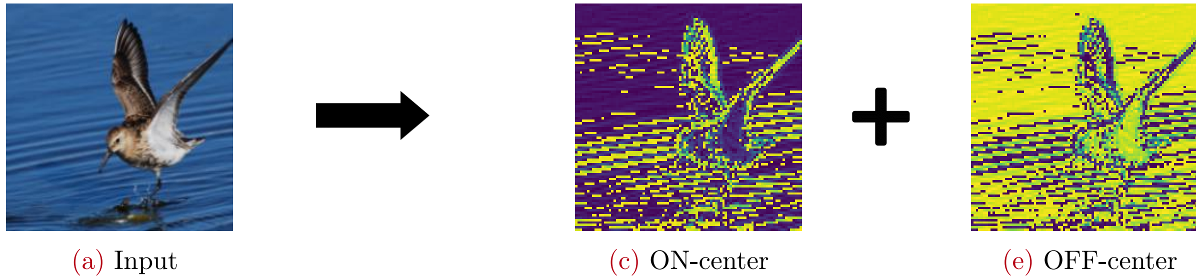


Figure 2.1: On and Off DoG and On-Kernel

These are the feature maps their OOCS kernels create when they are convoluted with an image using a stride= 2 (Fig. 2.2):



(a) Input (c) ON-center (e) OFF-center

Figure 2.2: OOCS kernels' outputs

Model Architecture

The model they used for control is Basenet, a simple CNN with 7 convolutional layers, 5 MaxPool layers and 3 linear layers. Specifically the layers are disposed in this order: conv1, conv2, maxpool1, conv3, conv4, maxpool2, conv5, maxpool3, conv6, maxpool4, conv7, maxpool5, linear1, linear2, linear3(output).

To implement the on-off residual maps, they split Basenet's layers into **two on and off parallel pipelines** between the 3rd and 4th convolutional layers. Each has half the number of channels of the original layers, and thus the number of training parameters remains the same as control. After the 4th convolutional layer, they concatenated the activation maps of the 2 pipelines and fed them to the rest of the network (Fig. 2.3).

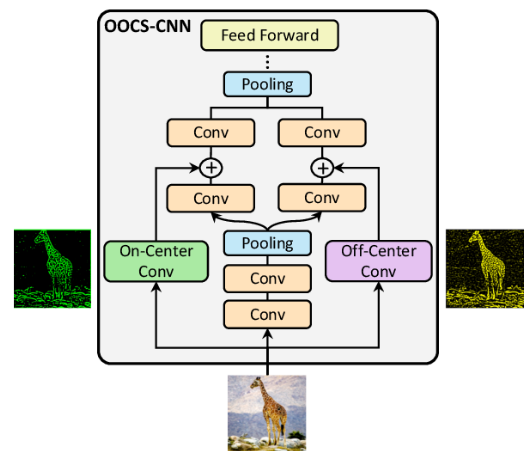


Figure 2.3: OOCS-Network

Dataset

They used a subset of **ImageNet** [A], created by randomly choosing 600 samples from 100 categories. From these samples, they used 500 images of each class for the training set, 50 for the validation set and 50 for the test set [A]. After cropping all the rectangular images to squares around the center, they resized all the images to **192 × 192 pixels**. They did not perform any pre-processing on the images.

Results

They implemented their model (**OOCSN**) and the control (**Basenet**) in TensorFlow 2.3, using Adam as optimization algorithm and setting the learning rate to 10^{-4}. They repeated all of the experiments

six times to report the mean value of the obtained results, together with their standard deviation. Here are their results:

	Accuracy
Basenet	0.408 ± 0.004
OOCSN	0.444 ± 0.006

Table 2.1: Babaiee's OOCS and Basenet results (n_trials= 6)

We can see that the implementation of the OOCS pathways caused a small but relevant improvement in the accuracy of the CNN.

Furthermore, they added their OOCS pathways to a deeper and widely used CNN: ResNet34. Also in this case the performance improved:

	Accuracy
ResNet34	0.617 ± 0.006
ResNet34-OOCS	0.634 ± 0.007

Table 2.2: Babaiee's ResNet results (n_trials= 6)

In the final discussion, they state:

"*There are additional top-down connections between layers in the retina, and the sizes of the receptive fields change depending on their location. It is therefore worth exploring the implementation of a complete model of the retina and to further improving the performance of this model, similar to many works that aim to transform biological mechanisms into better machine learning models.*"

3| Methods

We conducted further experimentation with the intention of verifying if it is possible to expand on the results of Babayee's research by implementing in a CNN new kernels inspired by deeper areas of the visual system. These are the materials and methods we used:

Environment

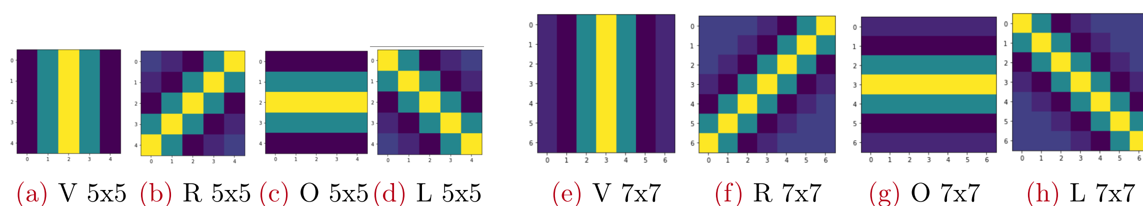
We utilized python's **PyTorch 1.10.2** extension to program the algorithms, using VSCode as programming environment. The calculations ran on GPU, on an NVIDIA GeForce GTX 1650.

Dataset

We replicated the dataset used by Babaiee et. al.[13] by randomly selecting 100 categories from ImageNet [A]. Then we randomly extracted 600 images from each category, assigning 500 to the train set, 50 to the validation set and 50 to the test set. [A]

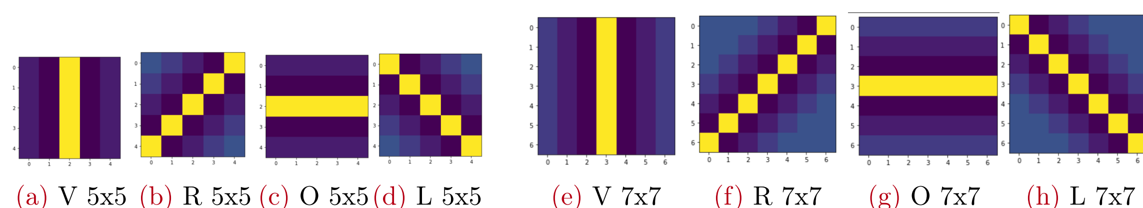
Kernels

Babaiee et. al. calculated the OOCS kernels using a Difference of Gaussians, in which the sum of all positive values amounted to 1 and the sum of all negative values amounted to -1. For simplicity and repeatability, we did not obtain the OOCS kernels from computations but we simply used the kernel they calculated. To mimic simple cells, we used two types of **oriented-line kernels**, each in a 5*5 pixel version and in a 7*7 pixel version (we mostly used the 5*5 variant):



(a) V 5x5 (b) R 5x5 (c) O 5x5 (d) L 5x5 (e) V 7x7 (f) R 7x7 (g) O 7x7 (h) L 7x7

Figure 3.1: SimpleCell Kernels



(a) V 5x5 (b) R 5x5 (c) O 5x5 (d) L 5x5 (e) V 7x7 (f) R 7x7 (g) O 7x7 (h) L 7x7

Figure 3.2: Precise_SimpleCell Kernels

The difference between SimpleCell (Fig. 3.1) and Precise_SimpleCell (Fig. 3.2) kernels is that Precise_SimpleCells are less sensitive to wider lines and in general more specific in their activation, because their activation section is thinner and surrounded by a stronger inhibition section than SimpleCell kernels. The values are not a result of a mathematical calculation: they use similar values to Babaiee's OOCS and they have been hand-tuned through observation of the resulting feature maps..
These are the values used (Fig. 3.3):

```
torch.tensor([
[-0.07, 0.10, 0.30, 0.10, -0.07],
[-0.07, 0.10, 0.30, 0.10, -0.07],
[-0.07, 0.10, 0.30, 0.10, -0.07],
[-0.07, 0.10, 0.30, 0.10, -0.07],
[-0.07, 0.10, 0.30, 0.10, -0.07]])
```

(a) Vertical SimpleCell

```
torch.tensor([
[-0.07, -0.10, 0.30, -0.10, -0.07],
[-0.07, -0.10, 0.30, -0.10, -0.07],
[-0.07, -0.10, 0.30, -0.10, -0.07],
[-0.07, -0.10, 0.30, -0.10, -0.07],
[-0.07, -0.10, 0.30, -0.10, -0.07]])
```

(b) Vertical Precise_SimpleCell

Figure 3.3: SimpleCell kernels' pixel values

Training

The training of the models was done using the sub-set of ImageNet previously described. Some data was saved for every epoch of training, specifically:

- The train accuracy, which is the accuracy of the model in predicting the category of the images in the train set.
- The validation accuracy, which is the accuracy of the model in predicting the category of the images in the validation set.
- The train loss, which is the result of the loss equation. [A] on the train set.
- The validation loss, which is the result of the loss equation on the validation set.

The training phase could end in 2 ways:

- Learning Early-Stop : when the validation accuracy of the CNN did not increase for 6 epochs, the training phase would end. [A]
- Loss Early-Stop: when the validation loss of the CNN did not decrease for 6 epochs, the training phase would end.

After the training phase, the network's accuracy was tested on the test set. This resulted in the final value from which we deduced the network's obtained performance.

Batch size

When not differently specified, the models here presented were trained using a **batch size of 32**. We had to use a lower batch size than Babaiee et. al. because we were limited by the capabilities of our GPU. As proved by our first experiment, the batch size used slightly modifies the models' performance but it does not confer a relevant change to the difference between the models.

3.1. Models' architectures

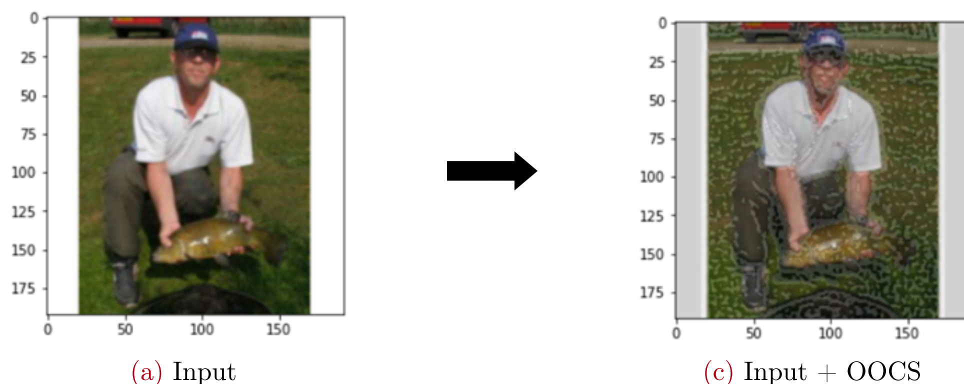
Repeating the results

First of all, we recreated the models used in Babaiee's research and tested their performance in order to see if we would be able to replicate their results. Their models were implemented in TensorFlow, but we implemented them in PyTorch.
Furthermore, we tested how important for the model' s performance is the addition of both the On-Center and Off-Center components. We implemented **OOCSN_ON**, which is a version of OOCSN with only one pipeline (the Off_center pipeline has been removed), and **OOCSN_ON_V2**, which preserves OOCSN's pipeline split but uses the On-Center component in both pipelines.

OOCS at the input level

Since Babaiee et. al implemented the On-Off-Center-Surround (OOCS) filters between the 3rd and the 4th convolutional layers, it was relevant to test whether there would be a difference in the accuracy of the model by adding the OOCS component directly to the input image. Since there are several ways to add the OOCS to the input, we tested 2:

- **Basenet_3-channels**, in which the the ON_center feature map is added equally to each of the 3 color channels of the input (Fig. 3.4)

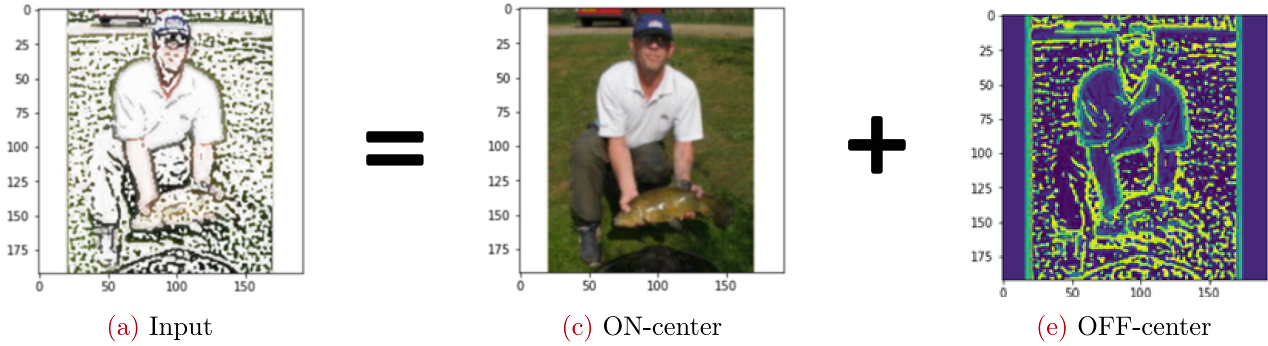


(a) Input (c) Input + OOCS

Figure 3.4: Basenet_3-channel's input

- **Basenet_4-channels**, in which the original input channels are not modified, but the

ON_Center feature is added as a 4th channel, i.e. the model receives a 4-channel image as input, with the channels being R/G/B/OOCS (Fig. 3.5)



(a) Input (c) ON-center (e) OFF-center

Figure 3.5: OOCS kernels' outputs

SimpleCell Network

Since Babaiee et. al. obtained positive results by implementing OOCS filters to a CNN model, we set out to test if implementing kernels inspired by simple cells in the visual area V1 of the vertebrate brain would also result in an improvement of the performance of CNNs.

To test whether SimpleCell kernels intrinsically improve the performance the first network we tested, **SimpleCellNetwork (SCN)**, implements SimpleCell kernels alone, without OOCS kernels.
Similarly to Babayee's OOCSN, the network splits into multiple pipelines at the the 3rd convolutional layer and rejoins them after the 4th convolutional layer.
Differently than OOCSN, our SCN splits into 4 pipelines rather than 2: one pipeline for each orientation of SimpleCell kernels (Fig. 3.6).

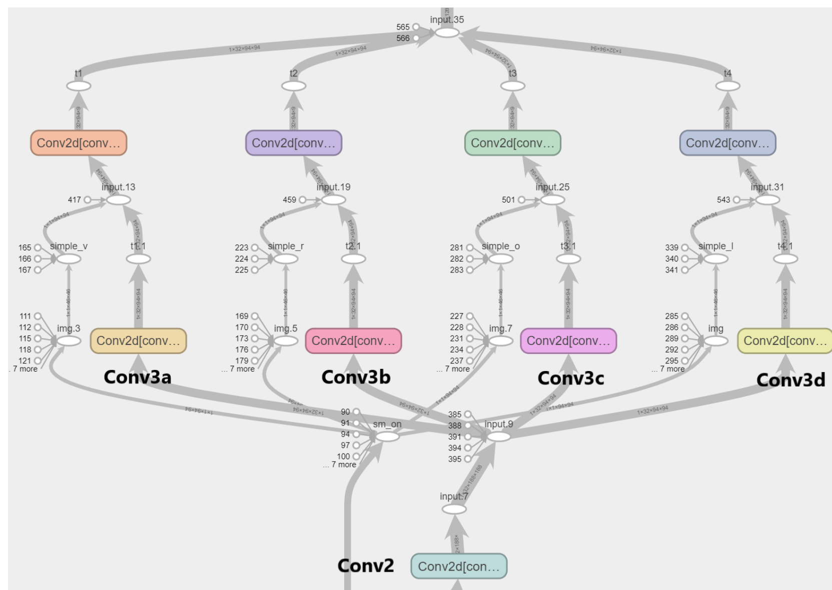
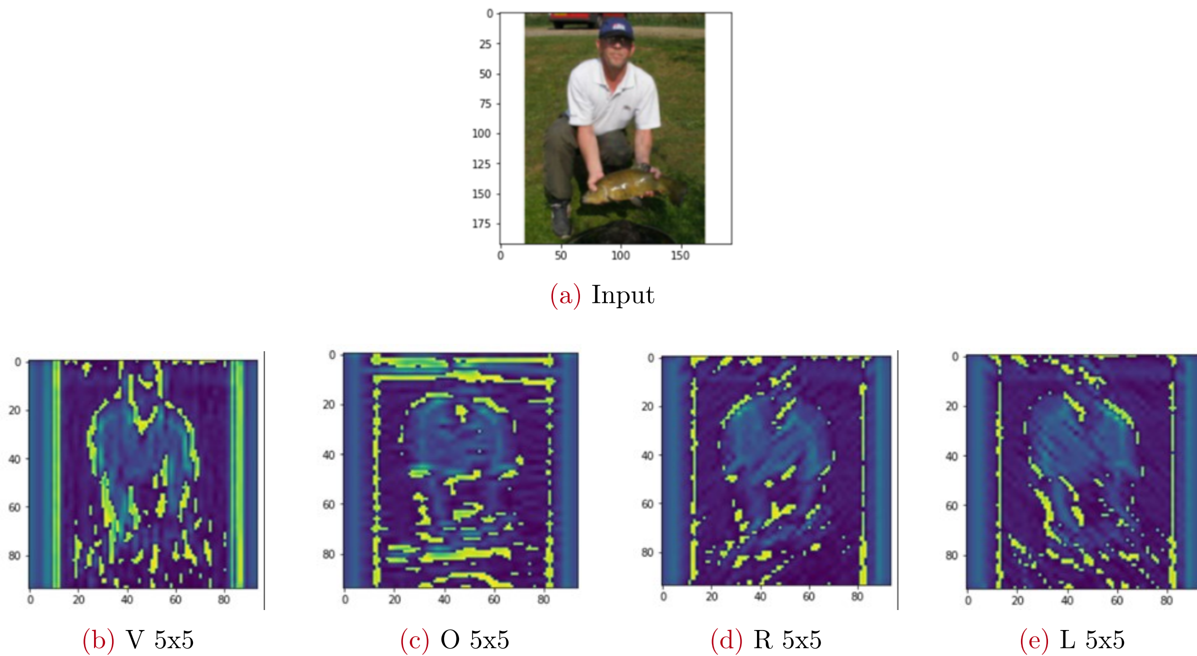


Figure 3.6: SCN's pipeline split

When Babayee et. al. designed OOCSN, they kept it directly comparable with Basenet by keeping the same number of channels for each layer. This means that since Basenet has 64 channels in the 3rd and 4th layers, their OOCSN has 32 channels in the corresponding layers, which summed among the 2 pipelines amount to 64 channels like in Basenet.
Since our SCN splits into 4 pipelines, it would be necessary to assign to its 3rd and 4th layers only 16 channels for each pipeline. However, we kept 32 channels instead, because a lower number may inhibit the model performance. Therefore we needed to train a new version of Basenet, directly comparable to our SCN: we'll call this model Basenet2, which is similar to the original Basenet but has twice the amount of channels on the 3rd and 4th convolutional layers.

This first version of SCN utilizes the first version of our kernels, the SimpleCell kernels we have shown above. These are some feature maps resulting from a convolution with SimpleCell kernels (Fig. 3.7):



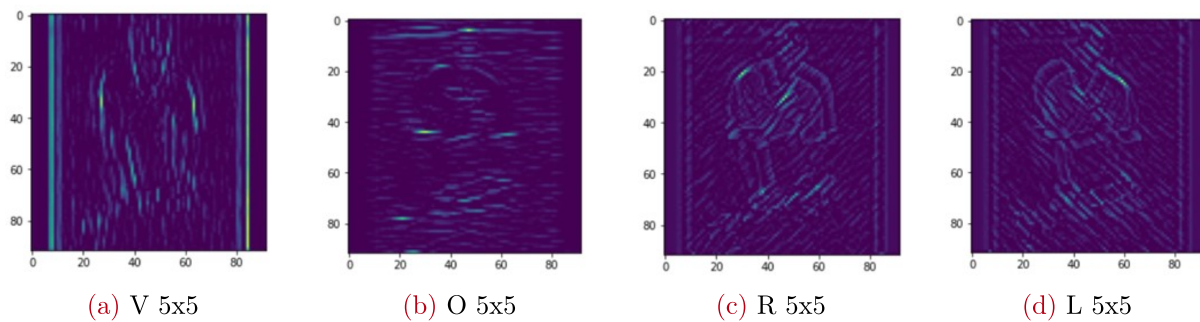
(a) Input

(b) V 5x5 (c) O 5x5 (d) R 5x5 (e) L 5x5

Figure 3.7: SimpleCell Kernels' Feature Maps (without ReLu)

From this image it is possible to infer that our SimpleCell kernels are not optimal in detecting oriented lines: the vertical line formed by the edge between the image and the white background is not only detected by vertically oriented SimpleCell kernels but also by the other orientations.
This is the reason why we designed the second version of SimpleCell kernels: the Precise_SimpleCell kernels we previously described.

These are the feature maps resulting from the output of these new kernels filtered by a ReLu activation function (Fig.3.8):



(a) V 5x5 (b) O 5x5 (c) R 5x5 (d) L 5x5

Figure 3.8: Precise_SimpleCell Kernels' Feature Maps

We can see they are more effective than SimpleCells, since the vertical line of the edge is strongly detected only by vertically oriented Precise_SimpleCells.
Therefore we created **SCN_V2**, which is identical to SCN but utilizes this new version of the kernels.

OOCSSimpleCell Networks

We continued with the second phase of the experiment: testing whether a CNN with both OOCS and SimpleCell filters would improve upon the performance of OOCSN.
We tested several different models, all with OOCS and SimpleCell kernels.

V2Network

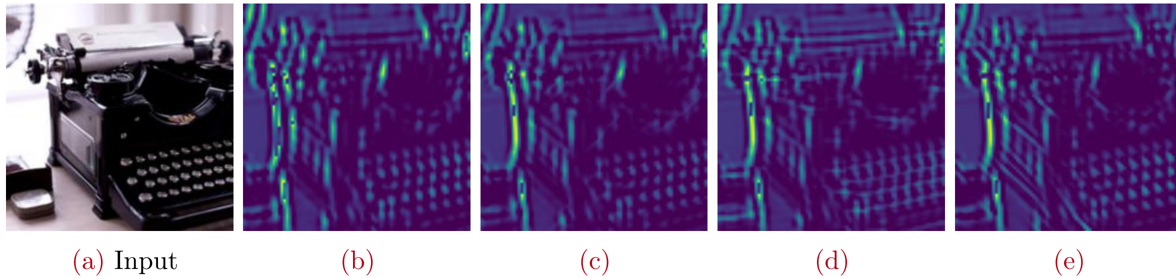
After testing the implementation of SimpleCell kernels inspired by the visual area V1, we set out to test new kernels inspired by visual area V2. Our knowledge of the processing that occurs in this area of the brain is quite fragmentary, but there seems to be consensus on the fact that part of the computations that take place therein involve the association between different line orientations, i.e. the **detection of angles and frequencies** from the oriented lines detected in V1.

Boynton and Hegdé state: "*Perhaps a V2 neuron with two preferred orientations simply receives direct inputs from two orientation-selective V1 neurons. Thus, just as Hubel and Wiesel proposed how a V1 simple cell might be constructed from a series of center-surround neurons in the lateral geniculate nucleus (LGN), perhaps V2 is constructed in a similar manner from V1 inputs*" [1]

AngleKernels_V1

To highlight angles we used the feature maps from SimpleCells. For each image we took the 4 feature maps resulting from the 4 different SimpleCell orientations and concatenated them into a 4-channel image. Then we created 256 filters of shape 4x2x2, such that each of them had all pixel values set to -0,07 apart from 4 **active pixels** in each filter, which were set to 0,30. We created one of these filters for each possible combination of positions for the 4 active pixels in the 4-channel 2x2 grid of each of these kernels. This amounts to 4x4x4 = 256 different filters. When convoluted with

the 4-channel stack of the SimpleCell's feature maps they are supposed to detect different angles and line frequencies (See Fig. 3.9).



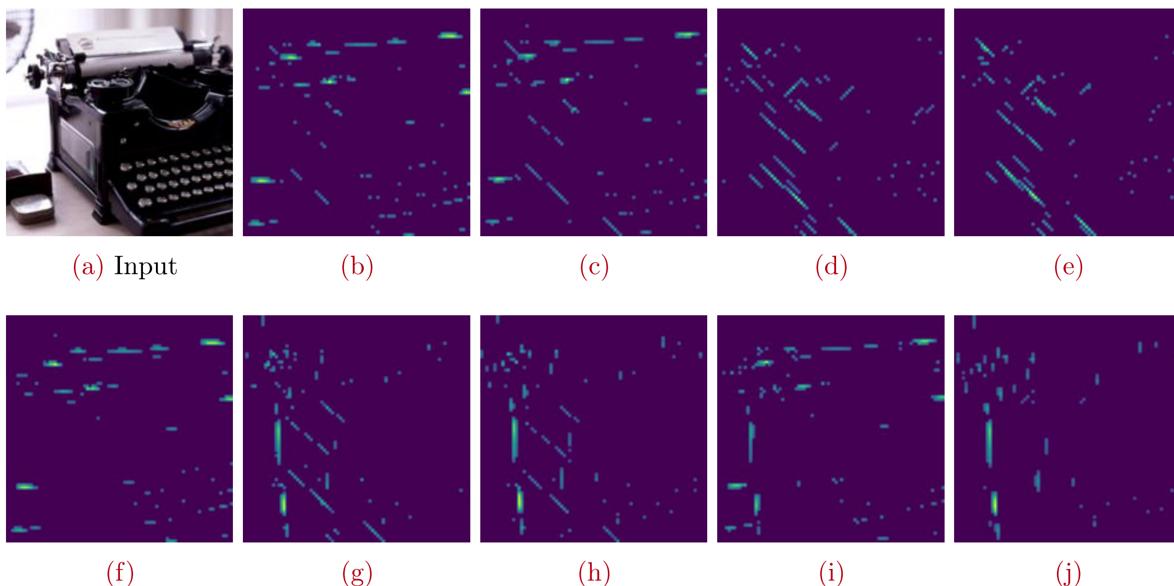
(a) Input (b) (c) (d) (e)

Figure 3.9: Examples of AngleFilters_V1's feature maps (defective)

AngleKernels_V2

AngleKernels_V1 seem to detect some interesting features in images, but from analyzing the resulting feature maps it's clear that they are redundant and that they are not detecting angles and frequencies as planned. Therefore we made AngleCells_V2 with a similar procedure:

We created 96 filters, still of shape 4x2x2. Each filter has 2 **active channels** among its 4 channels, and in each active channel there is an **active pixel**. Therefore, for each combination of 2 active channels there are 4x4= 16 possible combinations of active pixels among the two channels. Considering the number of combinations of possible active channel pairs is 6, the total number of filters resulting from this procedure is 6x4x4 = 96 filters. We also used threshold operations to filter out irrelevant values.
These filters seem to be more effective than AngleFilters_V1 (See Fig. 3.10 for feature maps)



(a) Input (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 3.10: Examples of AngleFilters_V2's feature maps

V2Network_V1

The first network we created with AngleFilters is V2Network_V1. It utilizes AngleFilters_V1, and instead of receiving the usual ImageNet image as input, it receives the 256-channel image made from the stacked feature maps resulting from AngleFilters_V1. While this is passed along the normal pipeline, there is a second pipeline which processes a downscaled 40x40 pixels version of the usual input image: it passes through 2 convolutional layers and 1 MaxPool layer, after which it is concatenated to the first pipeline after its 5th convolution. We also used V2Network_V1_cut for controlling the efficiency of the first pipeline by itself: it is identical to V2Network_V1 but without the second pipeline with the downscaled input image.

V2Network_V2

V2Network_V2 is the same as V1 but it uses AngleNets_V2, and it has fewer channels on the initial convolutional layers so that it is comparable to Basenet. We used V2Network_V2_cut for controlling the efficiency of the first pipeline by itself.

V2Network2_V1

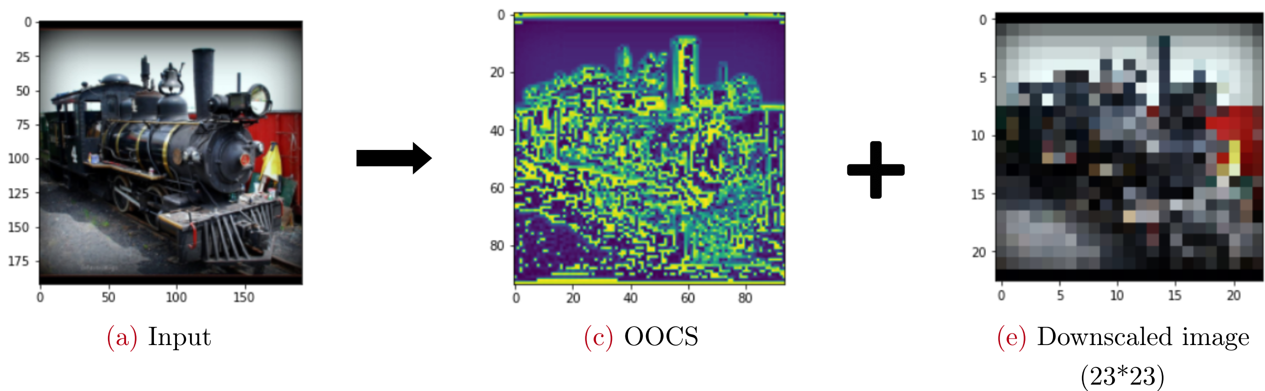
This version of V2Network takes the usual image as input. After passing through the 3rd convolution, the pipeline has 96 channels. To each of these channels, one of the 96 feature maps resulting from AngleFilters_V2 is added.

RawNetwork

Another relevant idea to experiment upon is using filters like the OOCS and SimpleCells kernels to improve the network's efficiency by highlighting the information in an image which is relevant for classification, so that irrelevant information does not have to be processed. This should result in a faster network with similar performance, and it can also be used to save computational time in order to increase the complexity of other parts of the model.

RawNetwork

Raw_Network_V1 takes as input the OOCS feature map instead of the standard input image. The pipeline splits in 4 parallel pipelines at the 3rd convolution. Then, to each pipeline one orientation of SimpleCells is added. After the 4th convolution and the 2nd MaxPool, a downscaled 23*23 pixel of the original input image (Fig. 3.11) is added to every channel after being divided by 5; after the MaxPool layer that follows the 5th convolution, the pipelines join. In order to use the computational time saved in the previous parts of the model, the last two convolutional layers of RawNetwork count twice the amount of channels than in Basenet.



(a) Input (c) OOCS (e) Downscaled image (23*23)

Figure 3.11: RawNetwork inputs

RawNetwork_cut

RawNetwork_cut is similar to RawNetwork but it does not add the downscaled image to the pipeline: it only receives the OOCS feature map as input. It's purpose is to be a control as to how much the addition of the downscaled image contributes to the accuracy reached by RawNetwork models.

Precise_RawNetwork

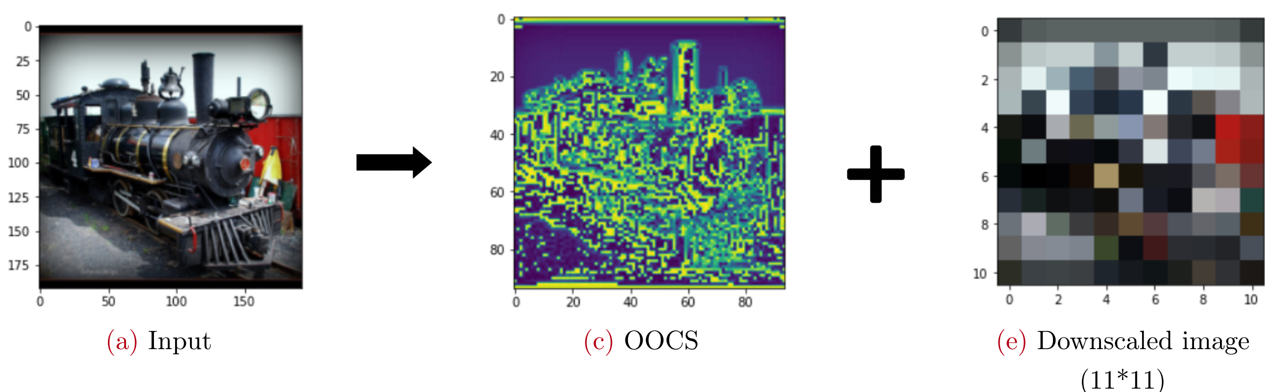
Precise_RawNetwork is identical to RawNetwork but in the pipeline-split section it adds feature maps from Precise_SimpleCells instead than SimpleCells.

RawNetwork_V2

RawNetwork_V2 is similar to RawNetwork but its pipeline does not split since it does not utilize any SimpleCell kernel.

RawNetwork_V3

RawNetwork_V3 is similar to RawNetwork_V2 but it performs a deeper downscale on the image, which reaches a dimension of 11*11 pixels (Fig. 3.12) and is added deeper in the model, after the 3rd MaxPool layer.



(a) Input (c) OOCS (e) Downscaled image (11*11)

Figure 3.12: RawNetwork_V3 inputs

UNIVERSITAS STUDII PADUANI
MCCXXII

4 | Results

In our experiments for some CNNs we will not have data from repeated trials, since for time restrictions it was impossible to train enough models. Therefore when the number of trials is n= 1 we will assume a standard deviation **std=**± 0.01, which is consistent with the std we obtained in some models with higher n values. This is the data we obtained:

Repeating Babaiee's results

These are the results Babaiee et.al. observed:

	Accuracy
Basenet	0.408 ± 0.004
OOCSN	0.444 ± 0.003

Table 4.1: Babaiee's Basenet and OOCSN results with **Batch_size=64** (n_trials= 6)

And these are the results we observed when repeating their experiment:

	Accuracy	Loss
Basenet	0.402 ± 0.013	4.303 ± 0.022
OOCSN	0.445 ± 0.009	4.238 ± 0.01

Table 4.2: Our Basenet and OOCSN results with **Batch_size = 64** (n_trials= 6)

	Accuracy	Loss
Basenet	0.417 ± 0.011	4.272 ± 0.01
OOCSN	0.456 ± 0.006	4.22 ± 0.006

Table 4.3: Our Basenet and OOCSN results with **Batch_size = 32** (n_trials= 10)

From this data, we can conclude that:

- The results of Babaiee's experiment are replicable, since the difference between their data and ours is minimal.
- The models perform slightly better with a batch size of 32. The difference is statistically significant but negligible.

OOCSN_ON

These are the results for OOCSN_ON and OOCSN_ON_V2:

	Accuracy	n_trials
OOCSN_ON	0.42 ± 0.02	6
OOCSN_ON_V2	0.455 ± 0.007	4
OOCSN	0.444 ± 0.003	10

Table 4.4: Our results with OOCSN_ON and OOCSN_ON_V2

These results are interesting since they seem to indicate that the addition of the Off_Center component might not produce an improvement over the addition of the On_Center component alone. OOCSN_ON shows a decrease in performance, while OOCSN_ON_V2 does not: **therefore it is the pipeline division itself that causes a small improvement in performance, and not the addition of the Off-Center component, which is redundant.**

OOCS at the input level

	Accuracy	Loss
3-channel-Basenet	0.397 ± 0.01	4.29 ± 0.01
4-channel-Basenet	0.414 ± 0.01	4.272 ± 0.01
Standard Basenet	0.417 ± 0.011	4.272 ± 0.01

Table 4.5: Our results with 3/4-channel-Basenet

We can see that 3-channel-Basenet performed worse than the original Basenet, while 4-channel-Basenet version had a similar performance.

For the 3-channel variant, it may be the case that adding the OOCS directly to the image channels decreases the clarity of the image data for the model, probably because it has a confounding rather than clarifying effect on the contrast edges of the image.

It is not as clear why the 4-channel variant does not cause any improvement over the standard Basenet model, since it processes the same 3 RGB channels plus the 4th added OOCS channel. It may be that the way we implemented it was defective, or that the addition of OOCS feature maps only confounds if implemented at the input level.

SimpleCell Network

This is the performance reached by our SCNs:

	Accuracy	Loss	Added Kernels	n_trials
SCN_V1	0.446 ± 0.006	4.23 ± 0.014	SimpleCells	3
SCN_V2	0.438 ± 0.002	4.24 ± 0.016	Precise_SimpleCells	3
Basenet2	0.435 ± 0.014	4.251 ± 0.024	None	6
OOCSN	0.456 ± 0.006	4.22 ± 0.006	OOCS	10

Table 4.6: Our results with SCNs

From this data we can observe than SCN_V1 has a statistically significant positive difference in performance compared to Basenet2, while SCN_V2 performance is similar to Basenet2's. This is surprising since by analyzing feature maps it seems that SCN_V2's Precise_Simple cells are better at detecting oriented lines than SCN_V1's SimpleCells. Since (by being less efficient in distinguishing orientations) the output of SimpleCells tends to be more similar to the output of the OOCS filter, this could imply that distinguishing the different orientations and utilizing separate pipelines may not increase performance, and that **the increase may only be related to the OOCS component.** It is therefore relevant to test whether adding both OOCS and SimpleCell filters in the same model would result in an improvement over OOCSN: this would prove that the improvements caused by OOCS and SimpleCells are distinguished from each other and can therefore sum up.

OOCSSimpleCell Networks

We tested a number of different models, each with both the OOCS and SimpleCell kernels. Here is a table with their characteristics and the final performance achieved, with the following parameters:

- The **Accuracy** reached by the model.
- **Pip_Div**= the layers in which the network is divided into multiple parallel pipelines
- **OOCS_add**= where the OOCS filter is added
- **S_add**= where the SimpleCell filters are added
- **n_trials**= the number training trials from which the data for the model has been calculated

- **S_dim=** The size of SimpleCell kernels
- **S_type=** the type of SimpleCell kernels (PSC= Precise_SimpleCells)

	Accuracy	Pip_Div	OOCS_add	S_add	n_trials
Basenet2	0.435 ± 0.014	//	//	//	10
OSCN_V1	0.447	$3^{rd} - 4^{th}$	after $1st$	after 3rd	3
OSCN_V2	0.437	before 5th	$3^{rd} - 4^{th}$	before 5rd	1
POSCN_V1	0.438	$3^{rd} - 4^{th}$	after $1st$	after 3rd	3
POSCN_V2	0.41	6th	$3^{rd} - 4^{th}$	before 6th	1
POSCN_V3	0.459	$3^{rd} - 4^{th}$	after 3rd	after OOCS	1
OOCSN	0.456 ± 0.006	//	//	//	10

Table 4.7: Our results with (P)OSCNs part.1

	S_dim	S_type	Other Info
OSCN_V1	5x5	SC	SC with ReLu
OSCN_V2	7x7	SC	//
POSCN_V1	5x5	PSC	//
POSCN_V2	5x5	PSC	//
POSCN_V3	5x5	PSC	$+$ ON-c to pip 1-2, $+$ OFF-c to pip 3-4

Table 4.8: Our results with (P)OSCNs part.2

We can see that, while some of the models improve upon Basenet2, we never see a statistically significant improvement over OOCSN. This means that our implementation of Simple-Cell-inspired kernels does not give the model any capability that is not already provided by the OOCS filter (i.e. **the detection of oriented lines has proven to be ineffective in this particular implementation**).

V2Network

These are the results we obtained for V2Networks:

	Accuracy	Epoch_duration	n_trials
Basenet	0.417 ± 0.011	748 sec	10
V2Network_V1	0.419	1071 sec	1
V2Network_V1_cut	0.282	693 sec	1
V2Network_V2	0.379	821 sec	1
V2Network_V2_cut	0.19	785 sec	1
V2Network2_V1	0.413	881 sec	1
OOCSN	0.456 ± 0.006	782 sec	10

Table 4.9: Our results with V2Networks

From this data, we can observe that:

- The models do not surpass basenets' performance, and they are slower in training
- With AngleCells_V1 alone, the model reaches an accuracy of 0.282
- With AngleCells_V2 alone, the model reaches an accuracy of 0.19

We will discuss some ideas for future implementations in the final chapter.

RawNetwork

Here is the data we obtained for RawNetwork:

	Accuracy	Epoch_duration	n_trials
Basenet	0.417 ± 0.011	748 sec	10
RawNet_cut	0.392 ± 0.0158	667 sec	2
RawNet_V1	0.404 ± 0.01	695 sec	2
Precise_RawNet_V1	0.425 ± 0.009	884 sec	3
RawNet_V2	0.431 ± 0.012	710 sec	3
RawNet_V3	0.404 ± 0.01	693 sec	3

Table 4.10: Our results with RawNetworks

We can see that RawNetwork_V2 has a better performance (and training time) than Basenet even if it only processes the OOCS feature map and a 23*23 pixel version of the original image. This means

that the OOCS feature map retains the information that was lost while downgrading the image to 23*23 pixels and by adding it only in deeper layers. It is also relevant that RawNet_cut, which only receives the OOCS feature map as input, obtained an accuracy just slightly inferior to Basenet's: this means that **the OOCS feature map contains most of the information that is useful for the model in its classification task.** Therefore color information, homogeneous areas and gradual gradient changes are not very useful for the CNN.

5 | Conclusions and Further Research

Repeatability

Our experiment proves that Babaiee's results are repeatable and robust, since we obtained almost identical accuracy values by implementing their Basenet and OOCSN in a different programming environment and with a different iteration of their stochastic data extraction from Imagenet.

OOCSN_ON

Our results suggest that the addition of both the On_Center and Off_Center components to OOCSN is redundant. It is the operation of splitting the pipeline into two different components that confers a slight increase in performance, not the addition of the Off-Center component.

SimpleCell kernels

Our SimpleCell kernels do not constitute an improvement over Babaiee's OOCS. In their most successful implementation they produce a relevant improvement over Basenet, but they are a weaker and more computationally inefficient alternative to the OOCS filter. Furthermore, when our SimpleCells are coupled with the OOCS filter they do not produce an improvement over the OOCS alone, which means that our implementation of kernels inspired by simple cells is redundant, since it does not give the model any capability that is not already provided by the OOCS filter alone.

However, this does not prove that an effective implementation of kernels inspired by Hubel and Wiesels' Simple Cells is not possible, since our experiment was quite limited in scope: we used only 4 orientations and very few variants of the models' architectures were tested. For further research it would be interesting to use more orientations, to test more variations in the pixel's values, and to use a wider set of kernels produced from **Gabor Patches** with many variations over a continuous range of spatial frequencies, orientations and contrast values.

V2Networks

The first angle-detection kernels we created (**AngleNets_V1**) are not effective in highlighting angles, but the second version (**AngleNets_V2**) seems to be an improvement. However, our implementations of AngleNets (i.e. **V2Networks**) are not more effective than Basenet in classifying images and

require a longer time to train. It is interesting to note that with only AngleNets ' feature maps as input, V2_Network_V1_cut performs only 32% worse than a model with the original image as input. For further experimentation, it would be interesting to remove redundancies from AngleNet_V2's feature maps by subtracting to the 16 feature maps of each of the 6 filter pairs their average values, in order to keep for each feature map only the values of the activated pixels which are mostly specific for that particular AngleFilter combination. Also, it might be possible to design better kernels for these feature maps: since the high-value pixels are quite far from each other, bigger kernels may be more efficient for extracting data, or a MaxPool operation may be implemented to reduce the distance between the high-value pixels.

RawNetwork

Our experimentation with RawNetworks has been quite limited, but the results are promising. The accuracy reached by RawNetwork_cut is only 6% lower than Basenet's; this indicates that the OOCS feature map contains most of the information that is relevant for a CNN in a classification task. Furthermore RawNetwork_V2 constitutes an improvement over Basenet, since it reaches a higher accuracy and it has a faster training time.

Final Remarks

Carrying out these experiments has been a proficuous learning experience.
There are some limitations in the methodology we used since I had to learn from zero almost everything that was needed; looking back at the procedures used, I see our experimentation would have been more effective by changing only one parameter of the models at a time, in order to better understand the effects on the final performance of the algorithms.

However, our experiments were effective in revealing several things:

- The results of Babaiee's experiment are repeatable.
- The addition of the Off-Center component is redundant: the apparent improvement is given by the pipeline split.
- The RawNetworks we created have been successful in demonstrating that there is potential for utilizing the OOCS filter for creating CNNs with better performance and higher efficiency
- The results obtained with RawNetwork_cut prove that the OOCS feature map contains most of the information thet is useful for the model in the classification task. This could lead to interesting considerations about which information in an image is actually relevant for CNNs.

Furthermore, our experimentation with SimpleCell and AngleFilters can be the basis for future research.

Bibliography

[1] G. M. Boynton and J. Hegdé. Visual cortex: The continuing puzzle of area v2. *Current Biology, Vol. 14*, July 13, 2004.

[2] T. N. W. D. H. Hubel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 1959.

[3] R. K. et. al. Computational neuroscience: Mathematical and statistical perspectives. *Annual Review of Statistics and Its Application*, page 6, March 2018.

[4] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. 1980.

[5] S. Z. Hamed Heidari-Gorji, Reza Ebrahimpour. A temporal hierarchical feedforward model explains both the time and the accuracy of object recognition. *Nature*, 11 March 2021.

[6] B. Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *Bulletin of Mathematical Biophysic, volume 5*, 2019.

[7] H. H. B. T. P. N. K. Logothetis, J. Pauls. View-dependent object recognition by monkeys. May 1994.

[8] R. Rojas. *Neural Networks - A Systematic Introduction*. Springer, 1996.

[9] S. Schuchmann. Analyzing the prospect of an approaching ai winter. *Thesis for: Bachelor*, page 9, May 2019.

[10] T. A. P. Sharat Chikkerur. Approximations in the hmax model. *MIT-CSAIL*, April 2011.

[11] W. P. Warren S. McCulloch. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysic, volume 5*, page 118, 1943.

[12] B. Z. e. L. Yinlin Li1, Wei Wu. Enhanced hmax model with feedforward feature learning for multiclass categorization. October 2015.

[13] M. L. D. R. R. G. Zahra Babaiee, Ramin Hasani. On-off center-surround receptive fields for accurate and robust image classification. 13 Jun 2021.

UNIVERSITAS STUDII PADUANI
MCCXXII

Appendix

- **ImageNet:** a large image database, created for use in the field of machine vision and object recognition. The dataset consists of more than 14 million images that have been manually annotated with the objects represented in them. The objects identified have been classified into more than 20,000 categories.
- **Train set:** the set of inputs used by the model to determine, or learn, the optimal combination of variables that will generate a good predictive model.
- **Validation set:** Another set of inputs, used to tune hyperparameters and to provide an unbiased evaluation of a model, i.e. to avoid "overfitting".
- **Overfitting:** when a model becomes able to classify or predict on data that is included in the training set but is not as good at classifying data that it wasn't trained on, i.e. it learns specific features of the train set but it is not able to generalize.
- **Test set:** A set of inputs used to provide a final unbiased evaluation of the model's performance.
- **Early stop:** the procedure of automatically interrupting the model's training when a certain number of training epochs is reached or when performance on the validation set stops increasing (to avoid overfitting). It can be based either on the epoch accuracy or on epoch loss.
- **Loss function:** it is the difference between the expected outcome and the outcome produced by the machine learning model. There are various types of loss functions, and in our particular implementation we used the cross-entropy-loss function (or softmax loss) which is commonly used for multi-class classification. This is its formula:

$$Loss = -\log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

With C being the total number of possible classes, S_p being the probability assigned by the model for the true image class, and S_j standing for the probability assigned by the model for each of every possible class.

UNIVERSITAS STUDII PADUANI
MCCXXII

List of Figures

1.1 McCulloch-Pitts Artificial Neuron 2
1.2 Simplest Perceptron 2
1.3 Neocognitron architechture 4
1.4 HMAX model 5
1.5 Kernel convolution operation 6
1.6 Examples of feature maps 7
1.7 The ReLu activation function 8
1.8 The ReLu function applied to an image 8
1.9 MaxPool operation applied to an image 9
1.10 5 linear layers 9
1.11 VGG-19 architecture 10

2.1 On and Off DoG and On-Kernel 11
2.2 OOCS kernels' outputs 12
2.3 OOCS-Network 12

3.1 SimpleCell Kernels 15
3.2 Precise_SimpleCell Kernels 15
3.3 SimpleCell kernels' pixel values 16
3.4 Basenet_3-channel's input 17
3.5 OOCS kernels' outputs 18
3.6 SCN's pipeline split 18
3.7 SimpleCell Kernels' Feature Maps (without ReLu) 19
3.8 Precise_SimpleCell Kernels' Feature Maps 20
3.9 Examples of AngleFilters_V1's feature maps (defective) 21
3.10 Examples of AngleFilters_V2's feature maps 21
3.11 RawNetwork inputs 23
3.12 RawNetwork_V3 inputs 23

List of Tables

2.1 Babaiee's OOCS and Basenet results (n_trials= 6) 13
2.2 Babaiee's ResNet results (n_trials= 6) 13

4.1 Babaiee's Basenet and OOCSN results with **Batch_size=64** (n_trials= 6) . . 25
4.2 Our Basenet and OOCSN results with **Batch_size = 64** (n_trials= 6) 25
4.3 Our Basenet and OOCSN results with **Batch_size = 32** (n_trials= 10) 25
4.4 Our results with OOCSN_ON and OOCSN_ON_V2 26
4.5 Our results with 3/4-channel-Basenet 26
4.6 Our results with SCNs 27
4.7 Our results with (P)OSCNs part.1 28
4.8 Our results with (P)OSCNs part.2 28
4.9 Our results with V2Networks 29
4.10 Our results with RawNetworks 29

UNIVERSITAS STUDII PADUANI
MCCCXXII

Acknowledgements

I would like to thank my family for their constant affection and support.