KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science

First cycle 15 credits

# Designing Convolutional Neural Networks for Classification of Metastatic Tissue in the Lymph Nodes

## Evaluation of the Effect of Feature Map Distributions Across Convolutional Layers on Network Performance

**PERE GINEBRA SOLANELLAS**

Stockholm, Sweden 2022

Degree Project in Computer Science
Author: Pere Ginebra Solanellas
Supervisor: Kevin Smith
Examiner: Pawel Herman
Date: June 2022
KTH - School of Electrical Engineering and Computer Science

# Abstract

Cancer is one of the leading causes of mortality in the world. It is estimated that about 20% of males and 17% of females will suffer it at some point in their lives, while 13% of males and 9% of females will die from it [1]. Cancer is caused by abnormal cell growth which can spread through the body in a process called metastasis. One of the ways to diagnose and evaluate the severity of cancer is to analyse the lymph nodes for presence of metastatic tissue. Recently, Artificial Neural Networks (ANNs) have been used to aid the detection of such tissue in images, but there are many design decisions to consider in making a network for this task.

One of the most important design decisions when developing an ANN is the choice of the network, as this will define how the model will evaluate patterns from the input. Convolutional Neural Networks are the main ANN architecture used for image analysis, as these use filters to evaluate how prevalent different shapes or patterns are in an image. My research has studied how different ways of using these filters affects lymph node image classification results.

From the worst to best performing models tested, there is a 0.0739 accuracy gap (from 0.8494 to 0.9234), representing an 8.709% increase in performance. Within models of the same size in terms of number of patterns analysed, there are gaps of up to 0.035 accuracy. My research has found network size to be closely related to training time, although different designs have different training complexities, so a case can be made for models of a same size that perform worse but also train faster.

The main conclusion of this project is that the way features are extracted from lymph node images in a Neural Network clearly affect its performance, but also its training complexity. With this knowledge, one can test different approaches and designs according to their requirements and resources.

# Sammanfattning

Cancer är en av de främsta dödsorsakerna i världen. Man uppskattar att cirka 20% av männen och 17% av kvinnorna kommer att drabbas någon gång under sitt liv, medan 13% av männen och 9% av kvinnorna kommer att dö av cancer [1]. Cancer orsakas av onormal celltillväxt som kan spridas i kroppen i en process som kallas metastasering. Ett av sätten att diagnostisera och bedöma cancerens svårighetsgrad är att analysera lymfkörtlarna för att se om det finns metastaserande vävnad. På senare tid har artificiella neurala nätverk (ANN) använts för att underlätta upptäckten av sådan vävnad i bilder, men det finns många konstruktionsbeslut att ta hänsyn till när man gör ett nätverk för denna uppgift.

Ett av de viktigaste besluten när man utvecklar ett ANN är valet av nätverk, eftersom detta kommer att definiera hur modellen kommer att utvärdera mönster från inmatningen. Konvolutionella neurala nätverk är den vanligaste ANN-arkitekturen som används för bildanalys, eftersom dessa använder filter för att utvärdera hur vanligt förekommande olika former eller mönster är i en bild. I min forskning har jag studerat hur olika sätt att använda dessa filter påverkar resultaten av klassificeringen av lymfkörtelbilder.

Från de sämsta till de bäst presterande modellerna som testats finns det en skillnad i noggrannhet på 0,0739 (från 0,8494 till 0,9234), vilket motsvarar en ökning av prestanda med 8,709 %. Inom modeller av samma storlek när det gäller antalet analyserade mönster finns det skillnader på upp till 0,035 noggrannhet. Min forskning har visat att nätverksstorlek är nära relaterad till träningstid, även om olika konstruktioner har olika träningskomplexitet, så det går att argumentera för att modeller av samma storlek presterar sämre men också tränas snabbare.

Den viktigaste slutsatsen av det här projektet är att det sätt på vilket funktionerna extraheras från bilder av lymfkörtlar i ett neuralt nätverk tydligt påverkar dess prestanda, men också dess träningskomplexitet. Med denna kunskap kan man testa olika tillvägagångssätt och konstruktioner utifrån sina krav och resurser.

# Contents

# Chapter 1
# Introduction

Artificial Neural Networks (ANNs) are machine learning algorithms based on the biological neural networks in our brain. The amount of research that has gone into these has increased during the last decade due to their capacity to recognize patterns in many types of data, but the field still has a lot of developments to be made [2].

One of the multiple problems that ANNs have been used for is computer vision, with various models being developed to detect and classify elements in an image. For this, data scientists design networks of neurons that will each learn a small pattern of the data to analyze, but there are no set rules that define the best design for a task. Therefore, scientists must test different designs in order to find the one that gives the best results for the specific problem [3][4].

Convolutional Neural Networks (CNNs) are a subset of ANNs used for multi-dimensional data types with spatially related data points [5][6]. The most common example of this is images [6][7], as they are represented by matrices of pixels, which store position in 2 or 3 dimensions that would be destroyed by flattening. CNNs recognize images by detecting patterns within them known as features.

This project analyzes the classification performance of CNNs in relation to the number of patterns they evaluate and their complexity. For this purpose, the *Patch Camelyon* (PCam) dataset [8][9] is used, which consists of segments of images from histopathological scans of lymph node sections. Each image is labeled according to the presence of metastatic tissue in the center of the image. Detecting this tissue in the lymph node is key to diagnosing and determining the severity of cancer.

## 1.1. Research Question

This thesis studies the effect of some architecture design decisions on the results of ANNs. More specifically, the number of feature maps and their distribution across convolutional layers in a Convolutional Neural Network (CNN) is analyzed. With this I aim to answer the question: *How does the distribution of feature maps in a convolutional neural network's architecture affect its image classification results?*

My main hypothesis is that deeper and larger networks generally perform better, as has been found in the past; this observation has given way to the term Deep Learning which is commonly used to refer to ANNs with multiple layers [5]. Apart from this, the little existing research on the effect of using different distributions of feature maps did not allow me to make more claims on the outcome of the study.

## 1.2. Approach

This project analyzes the results of architectures with different shapes and sizes in respect to the number of features extracted in each layer. This has been done by first designing a base architecture and framework using PyTorch and Jupyter notebooks allowing for easy iteration, modification and training of each specific architecture design.

The results of all these architectures are compared by recording the validation accuracy, precision, recall and f1 scores of each epoch during training, and the same metrics as well as the ROC AUC score over a test dataset with the fully trained model after a set number of epochs.

## 1.3. Scope

The scope of this project is defined by the number of different architectures tested and their maximum sizes. These are limited by the available computing power and time to complete the project. With unlimited resources one could test a broader number of designs of bigger potential sizes and even test them with different hyperparameters and multiple datasets to obtain observations that are easier to extrapolate.

Since this is not the case, experimentation has been restrained to 5 different types of architecture with 9 different size variations in hopes of exploring a wide enough range of designs to provide meaningful insight to the topic. Following this, each of these 45 unique architectures has been trained 5 times, resulting in 225 total models and over 150 hours of total training time on a mobile GPU in a laptop (RTX 2060 Max-Q). The same hyperparameters are used for all tests and only the Patch Camelyon image dataset is used, as previously mentioned.

# Chapter 2
# Background

## 2.1. Artificial Neural Networks and architectures

The field of Machine Learning studies algorithms that can improve or "learn" over time. One subset of these algorithms is Artificial Neural Networks (ANNs), which are inspired by neurons in the brain, as they also consist of interconnected groups of nodes. Feedforward Neural Networks (FNNs) are a very popular and simplified version of ANNs. These networks are made up of layers of neurons, the first being the input layer, the last being the output, and the rest the hidden layers. Each layer takes the previous one as input, so every neuron takes the output of a set of neurons in the previous layer and computes a weighted sum with them, and sometimes also applies a function (activation function) before passing the result as output. These weights represent a pattern of the input, and by adjusting them a set of patterns that best describes the data being analyzed can be found. So, each layer of $N$ neurons learns to identify the contents of the previous one using $N$ different characteristics/patterns observed from it. When layers are chained one after the other, the resulting models can identify the original inputs by evaluating if they follow the patterns that they have learned to recognize by optimizing their weights.

The term 'architecture' in ANNs refers to the design of the network. This design takes into account: how many layers it has in total, or the size/depth of the network; how many neurons make up each layer, or the size/width of the layer; and the type of neurons/layers used. Architecture design plays a big role in defining a model that delivers the desired results. It has been commonly observed that deeper architectures, that is, with more layers, tend to find better patterns to recognize their input datasets [5]. However, given too many layers, these patterns adjust too well to the input and therefore do not recognize new occurrences as well (a problem known as overfitting) [2]. Deep architectures also take a longer time to train, especially when working with big datasets and inputs such as images. For this reason, finding an architecture that suits the user's needs and resources is a lengthy and often tedious process that requires a lot of testing and analysis of results. For now, all we have are general guidelines for finding a good network [10], such as starting small and building up while

carefully analyzing and keeping track of results to avoid common problems such as overfitting.

## 2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a subset of ANNs, and more specifically FNNs, used for data types where data points are spatially related, especially when this space they are located in is not one dimensional [5][6]. The reason for this is that they work with groups of neighboring data points to evaluate patterns instead of using them individually. This process takes place in the convolutional layers of the CNN architecture. In the case of standard CNNs for image classification, convolutional layers apply a set of 2 dimensional filters to their input in a sliding window style to generate an output feature map. Each filter represents a pattern, and the convolutional layer calculates how present that pattern is in each window of the input image/feature map it receives. This is done by multiplying the values of the filter with those of the window of pixels being analysed and then adding all the values of the resulting matrix into one, which represents the resemblance between that window and the feature on the filter. In this case, the weights being optimized by training the model are the values in each filter's matrix, which define what pattern they evaluate. An illustration of this kind of architecture can be seen in *Figure 1*.

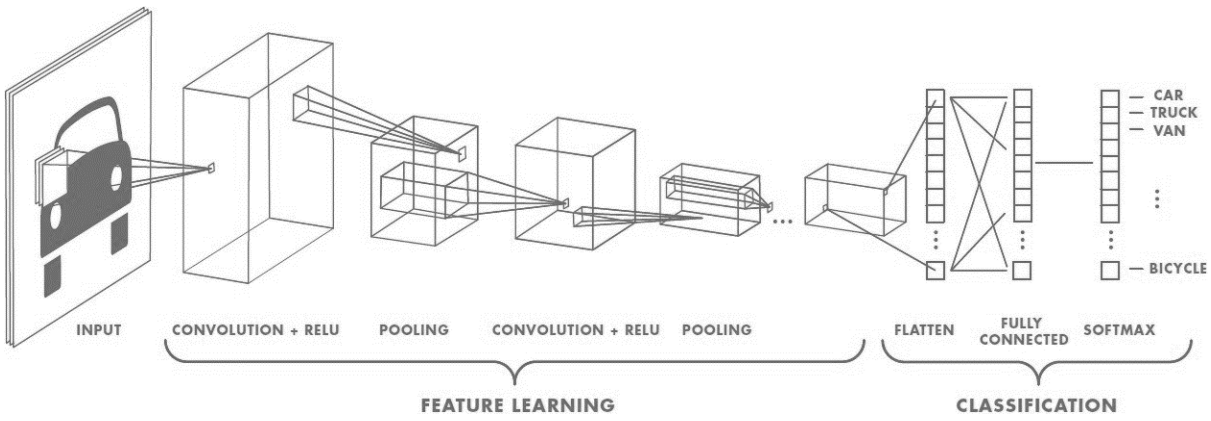


***Figure 1.*** *Basic CNN architecture. Source: MathWorks*

Multiple patterns can be evaluated within one single convolutional layer by simply applying multiple filters individually to the input. Each filter results in a new feature map that stores how present a pattern is in each position of the map, so the dimension of the output of a convolutional layer depends on the number of features it finds (number of filters x size of feature maps). The number of filters applied in a layer can be considered the number of

neurons in that particular layer in the case of a CNN, as each takes the previous feature maps as input and outputs a new one.

These convolutional layers are often combined with other layers and activation functions to introduce non-linearity in the model and increase robustness, like is often done with other ANNs [2]. One of the most common activation functions for convolutional layers is the ReLU function [5], which sets features with negative values to 0. Then, to close out the network, the last layer's output is usually flattened and transformed into the number of outputs needed via a fully connected layer. In the case of classification, the last feature map can be transformed into the same number of neurons as classes, as this will result values representing how close to each class the original input is.

# Chapter 3
# Method

## 3.1. Dataset

The dataset used for this study is the *Patch Camelyon dataset* (PCam)[8][9]. This dataset originally consists of 327,680 color images (96x96 pixels) extracted from whole slide images (WSIs) of histopathologic scans of lymph node sections. For the purposes of this project only 70,000 images are used: 50,000 for training, 10,000 for validation and 10,000 for the testing set. These have been randomly selected but preserve the same slight class imbalance of the whole set, in which 59.49% of images are negative and the rest are positive.

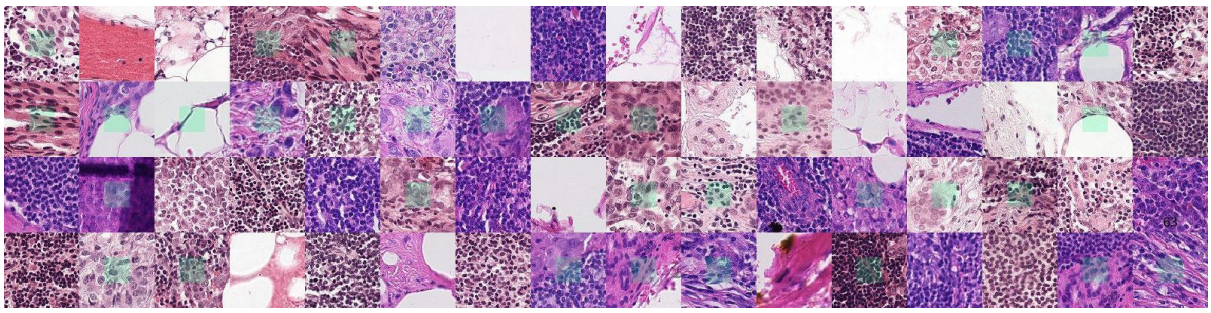

***Figure 2.*** *Example images from PCam. Green boxes indicate tumor tissue in center region. Source: PCam*

The images of this dataset are labeled according to the presence of at least a single pixel of metastatic tissue (indicating possible spread of cancer in the body) in the central 32x32 pixel zone of the image, as represented by a green box in *Figure 2* (not present in the actual dataset patches). One could interpret that this means the rest of the image is not necessary for classification purposes, but using the whole 96x96 image has yielded better results than smaller fragments of 64x64 or 48x48 pixels, so the entire images will be used.

## 3.2. Approach

For this project, a training script has been designed using PyTorch. This script uses a training set divided into batches of images (in this study of size 64) which will be input into the network and evaluated to improve the weights of said network. This is done using a gradient descent algorithm [11], which uses the derivative of the loss function calculated after every batch using cross entropy, to look for a minimal value of such function while updating the weights. In this case PyTorch's Adam gradient descent optimizer function is used.

To run this algorithm, the script takes several parameters: a number of epochs, which defines how many times the gradient descent will be performed on the input (in this case 30); a validation set of images, to evaluate the training done after every epoch; and a learning rate, to set the size of the steps taken by the optimizer function when updating weights (in this case 0.0005). After every epoch, if the classification of the validation dataset obtains better results in the shape of higher validation accuracy, its weights are stored as a possible best end result.

## 3.3. The architecture

Since this project focuses on the effect of the variation of number of features per convolutional layer on classification results, the base architecture is kept simple: a ReLU activation function for every convolutional layer, 3 evenly spaced Max-Pool layers and one final fully connected linear layer with a Softmax activation function are used for every architecture. These design decisions are inspired by early CNN architectures like LeNet-5 [12], AlexNet [13] or VGG-16 [14], which rely on stacking convolutional layers combined with pooling layers to increase performance. An example architecture following the previous base design principles can be seen in *Figure 3*.

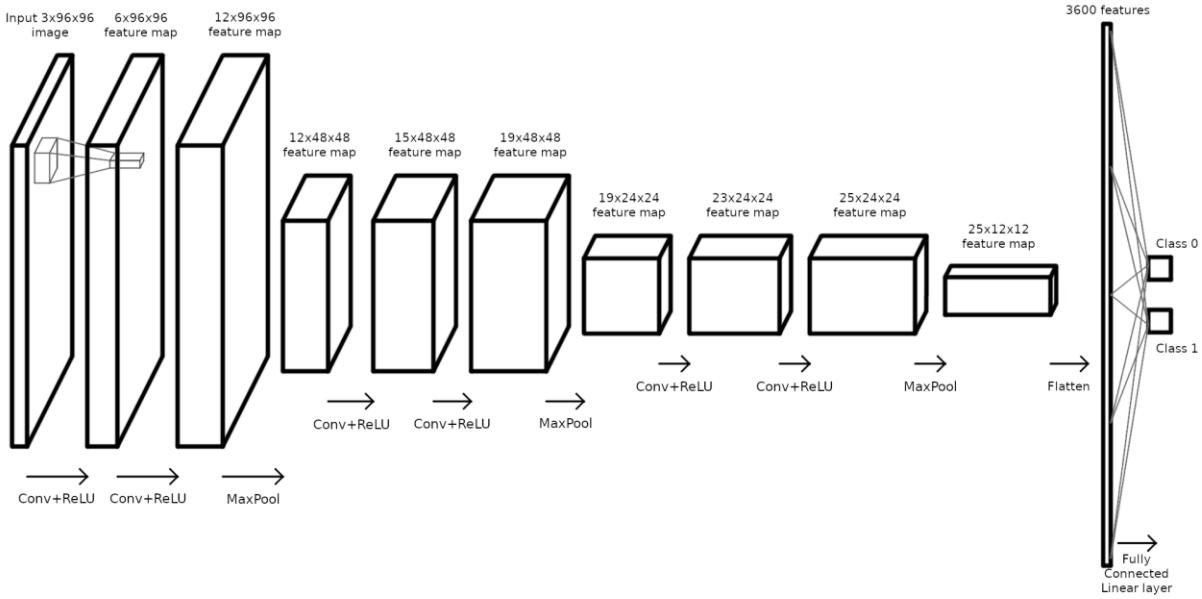


***Figure 3.*** *Example architecture design with 6 convolutional layers, each outputting 3 more feature maps than the previous one. This results in a total of 100 feature maps extracted throughout the network.*

### Activation Functions

Activation functions transform the output of neurons before passing them to the next layer and are commonly used to introduce non-linearity to a model and increase robustness. The two types used in this project are:

- ReLU (Rectified Linear Unit): simple function that is commonly used in CNNs to eliminate negative weights in a feature map after a convolutional layer [5], which would signify a negative presence of a feature in an area. It is defined as:

$$f(x) = \max(0, x)$$

- Softmax: function used to normalize multiple outputs into a probability distribution. This is often used as an activation function for the last layer of a network to define the probability of an input being in each possible class in a classification problem. It is defined as:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_j^K e^{z_j}}$$

  Where *K* is the size of *z*, the number of outputs to be transformed:

$$softmax: \mathbb{R}^K \rightarrow (0,1)^K$$

### Hidden Layers

Hidden layers are the layers of neurons that make up the network between the input and the output. In my model the following ones are used:

- Convolutional Layer: as mentioned in the background section, convolutional layers are used to extract features with more than one data point to preserve the spatial relationship between them that create shapes and patterns. These use matrices as filters which they apply over the input like a sliding window or convolution. This way, the matrix continuously overlaps different sections of the input, performing a weighted sum with the filter's values as weights and placing the result in the center of the overlapped section in the new feature map.

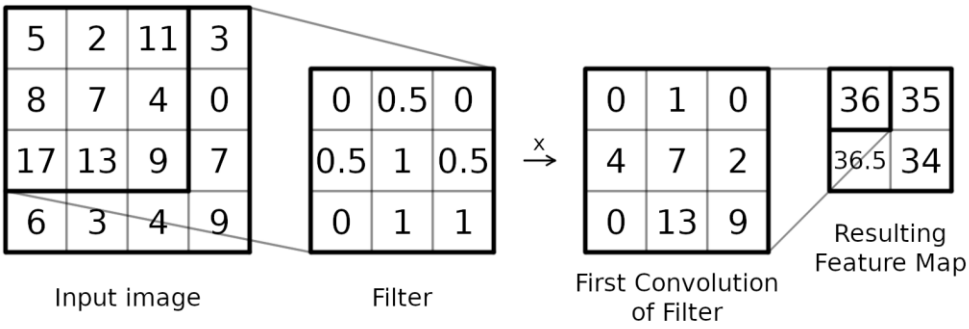


***Figure 4.*** *Example of a 3x3 convolutional filter applied over a 4x4 feature map or image*

In this architecture, the convolutional layers have a kernel size of 3 (3x3 matrix); a stride of 1, so the filter shifts one pixel at a time over the image; and a zero-padding of 1, which adds a 1-pixel margin of value 0 around the image so that the output feature map always has the same resolution as its input, as it would otherwise decrease after each layer.

- Max-Pooling layer: layer used to down-sample feature maps, which helps ease computation as well as make the model more robust by removing fine details that might make it hard to generalize, such as exact position of features in a map [1]. For this, it divides the input into matrices of a given size (in this project 2x2) and replaces each matrix by the single maximum value in it. This effectively divides the input size by the size of the kernel, so a 2x2 max-pool with an input of 96x96 would result in an output of size 48x48.

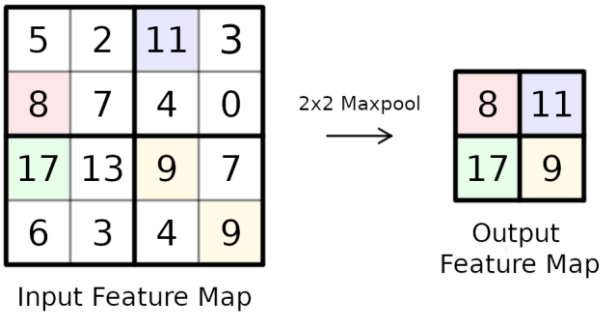


***Figure 5.*** *Example of a 2x2 Maxpool layer applied over a 4x4 feature map*

- Linear fully connected layer: fully connected layers connect each of their neurons with every neuron in the next layer, unlike the previous two types of layers. There are many different types of fully connected layers, in this case PyTorch's Linear layer is used, which applies a linear transformation to its input. With this the features of the last layer are transformed into 2 outputs that represent the similarity of the original input to each of the 2 classes. This is the layer that performs the classification.

## 3.4. Experiment design

The tested architectures are designed around the fact that the focus of this study is on the number of feature maps obtained per convolutional layer. Each distribution has been tested on 9 different network sizes defined by a combination of 3 depths, with 3, 6 or 9 convolutional layers; and 3 widths, with 128, 256 or 512 total number of feature maps. An example architecture with 100 feature maps distributed across 6 convolutional layers is schematically represented in *Figure 3*.

To define the number of feature maps to output on each convolutional layer, 5 simple shapes are used to represent their progression in the architecture:

- *Flat*: architectures where all layers extract a fixed number of feature maps. These are the basic "default" architectures to compare the others to. These will serve to see if performance mostly just comes down to size and not to the complexity of the design.
- *Up*: architectures where each consecutive layer outputs more feature maps than the previous one. The example in *Figure 3* follows this design, with an increase of 3 feature maps per convolutional layer. These will evaluate more complex patterns and less simple ones.
- *Down*: architectures where each consecutive layer outputs less feature maps than the previous one. These will evaluate more simple patterns and less complex ones.
- *Valley*: architectures in which the number of feature maps decrease with each layer until the minimum is reached in the middle layers and then goes back up to the starting size.
- *Hill*: architectures in which the number of feature maps increases with each consecutive layer until the maximum is reached in the middle layers and then goes back down to the starting size.

The combination of sizes and designs results in a total of 45 unique architectures. As can be seen in *Table 1*, the *up*, *down*, *valley* and *hill* models' progression is lineal: features increase/decrease by a flat amount defined by the total number of features and layers.

| Feature maps after each convolutional layer | | Shape | | | | |
|---|---|---|---|---|---|---|
| | | ***Flat*** | ***Up*** | ***Down*** | ***Hill*** | ***Valley*** |
| Layer | **1** | 57 | 11 | 102 | 21 | 88 |
| | **2** | 57 | 23 | 91 | 41 | 71 |
| | **3** | 57 | 34 | 80 | 61 | 53 |
| | **4** | 57 | 46 | 68 | 82 | 35 |
| | **5** | 57 | 57 | 57 | 102 | 18 |
| | **6** | 57 | 68 | 46 | 82 | 35 |
| | **7** | 57 | 80 | 34 | 61 | 53 |
| | **8** | 57 | 91 | 23 | 41 | 71 |
| | **9** | 57 | 102 | 11 | 21 | 88 |
| **Total Features Extracted:** | | 513 | 512 | 512 | 512 | 512 |

***Table 1.*** *Example distribution of 512 features across 9 layers in each architecture design. Each cell represents the number of feature maps generated by each convolutional layer in a specific design.*

## 3.5. Model training

To compare the results of all these architectures, each is trained and tested 5 times to reduce variability, and as mentioned earlier, metrics are stored during the training and after testing. The parameters used for training all the models - batch size of 64, learning rate of 0.0005, 30 epochs and the Adam optimizer [11] – have been selected after manually experimenting with different combinations and architectures to ensure the best possible results for all models while also making sure that most architectures can train in a reasonable amount of time. A lower batch size usually results in better generalization but also makes it computationally harder to train [9]; when using a 32 image batch results are slightly better (up to ~1% increase in accuracy) but 64 is significantly faster (upwards of 10%). A lower learning rate tends to be able to reach better accuracy but also takes more epochs to do so [2]; a rate of 0.001 is a good place to start testing, but in this case it often fails to train on some models, so 0.0005 is the chosen rate to get faster results than 0.0001. Finally, with this combination of parameters, 30 epochs are where most the tested models reach a peak in accuracy.

## 3.6. Metrics

The metrics used to compare models in this project are:

- Accuracy: the percentage of correct predictions. Calculated as:

$$accuracy = \frac{correct\ predictions}{total\ predictions}$$

- AUC ROC: the Receiver Operator Characteristic (ROC) curve is a plot that represents the performance of a binary classifier in relation to its discrimination threshold. It is defined by plotting the true positive rate against the false positive rate of multiple discrimination thresholds.

  The area under said curve (AUC) can be used as a classification score, an AUC of 1 represents a perfect classifier which predicts 100% correctly with every threshold, while an AUC of 0.5 represents a random classifier. It gives a more insightful classification score by evaluating how well the model discriminates between classes, as predicting something to be of class *A* with a certainty of 0.55 is not the same as doing it with 0.9 certainty.

Other basic metrics like average total accuracy, average accuracy per epoch, average accuracy per second of training, total training time, etc., are used.

# Chapter 4
# Results

## 4.1. Classification accuracy

The classification accuracy of each model over the test dataset has been used to evaluate the effect of each design decision on a network's performance. As seen in *Figure 6*, some models have an accuracy of exactly 0.5928. This value is the ratio of class imbalance in the dataset, which means that these models predict the same class (negative) for every input. This happens due to the complexity of some of the architectures, especially the ones with the most feature maps, which sometimes fail to train.

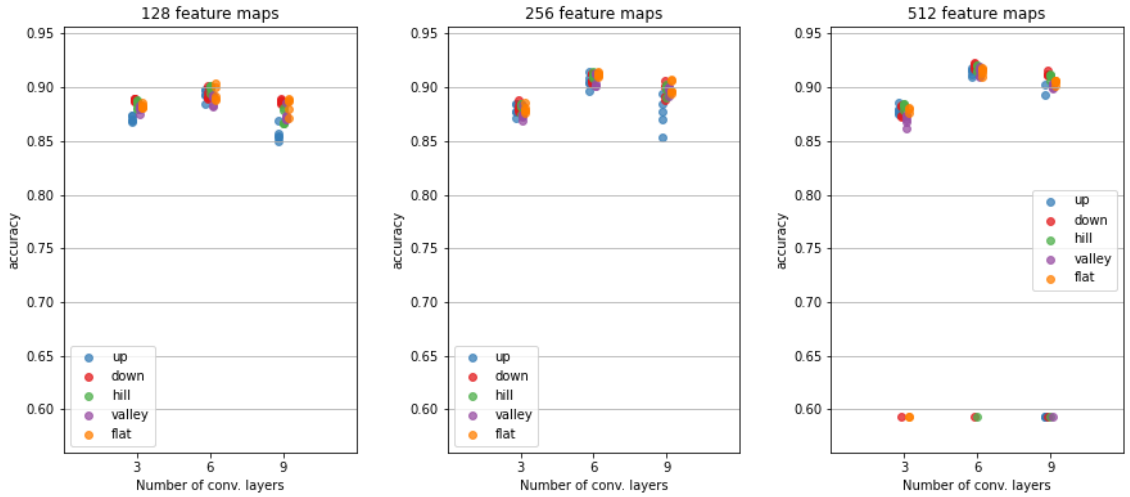


***Figure 6.*** *Test accuracy of each trained model. Each graph contains models with the same architecture width and groups them by their architecture depth.*

Looking at the average accuracy per architecture type and size in *Figure 7*, the models that are affected by this can be seen more clearly, as they have averages around or under 0.85. All distribution designs have problems when training on 512 feature maps, especially with 9 layers, where all of them except *flat* fail to train at least once.

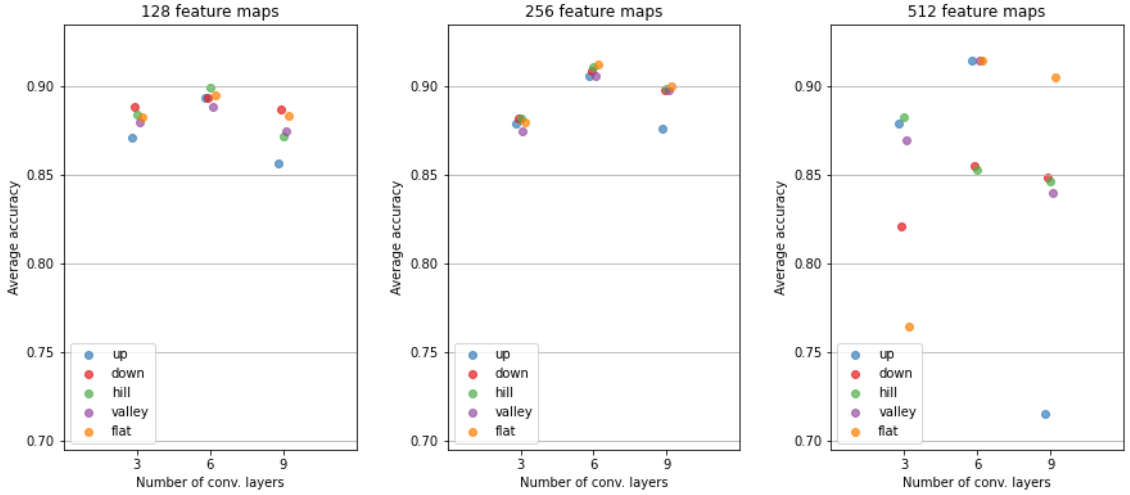


***Figure 7.*** *Average test accuracy of each unique architecture design's 5 tested models. Each graph contains models with the same architecture width and groups them by their architecture depth.*

Removing the untrained models in *Figure 8*, the trained models' results become more clear. The first observation we can make is that in this case having 6 layers is the best option, 3 layer networks perform the worst and lose accuracy when increasing the number of feature maps, and 9 layers underperforms 3 in networks with 128 feature maps but is closer to 6 with 256 and 512. Increasing the number of features generally brings better results in the networks with more than 3 convolutional layers.

Focusing on the performance of each individual distribution shape, a clear best performer cannot be identified. However, some observations can be made. For instance, *up* and *valley* are all around the worst performers, *flat* generally gets slightly higher accuracies and tends to be more consistent in the models with more features, and *down* can get the best results but is more inconsistent and fails to train more often. For more detailed information on the distribution of scores for each unique architecture, refer to *Table 2* in the appendix.

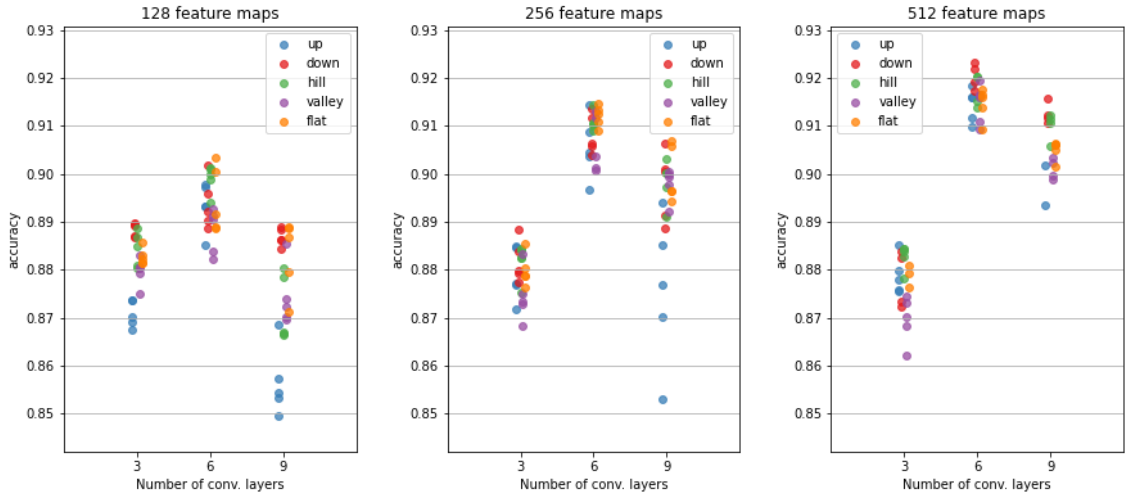


***Figure 8.*** *Test accuracy of each tested model, excluding those with accuracies under 0.6 (untrained). Each graph contains models with the same architecture width and groups them by their architecture depth.*

## 4.2. AUC score

As mentioned earlier, the area under the ROC curve gives more insight into how well a model performs classification by including prediction confidence into the score by using discrimination thresholds. This can be observed in *Figure 9*, as not all the models that did not train correctly have the same score. This is probably because even though they predict the same label for all images, each model's certainty of the chosen label might be different due to weight variations from the training process.

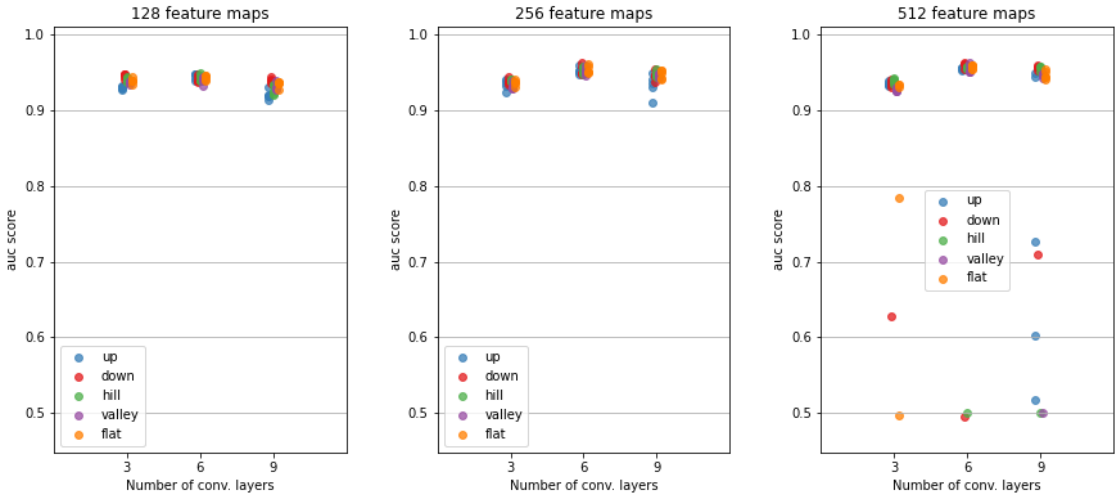


***Figure 9.*** *Test ROC AUC score of all trained models. Each graph contains models with the same architecture width and groups them by their architecture depth.*

By not including the models with scores under 0.8 out of the plot in *Figure 10*, some new observations can be made about the trained models. The *down* architectures show worse results with the parameters they were performing the best in terms of accuracy. Other models like *valley*, which got some of the worst accuracies, seem to do slightly better when comparing AUC scores, which might indicate that they are more confident on their correct classifications and less on their incorrect ones.

Focusing on number of layers, the disparity in this score is smaller between the 3 groups compared to when using accuracy. More specifically, some models with 3 layers and 128 features sometimes match or even outperform their higher layer count versions, when they underperformed in terms of accuracy.

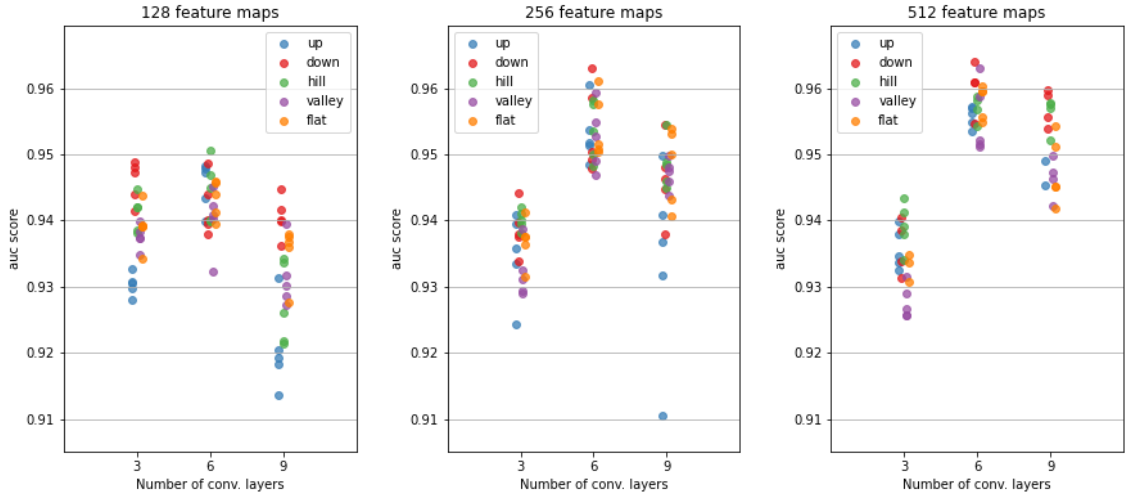


***Figure 10.*** *Test ROC AUC scores, excluding models with a score under 0.8. Each graph contains models with the same architecture width and groups them by their architecture depth.*

For more detailed information on the distribution of scores for each unique architecture, refer to *Table 2* in the appendix.

## 4.3. Training progression

*Figure 11* represents the average validation accuracy of each architecture after every epoch of training. This gives us an indication of how fast each model converges to its maximum accuracy: models with 3 layers approach their peak at around 15 epochs, models with 6 layers start slowing down at around 20-25 epochs and the 9-layer models continue improving in accuracy even past 25.

There is not much difference in epoch training progression between different architecture designs within a set of size parameters. Only slight offsets and models that reach their plateau earlier than others can be observed. For example, the *up* model, which seems to be the worst performing in terms of accuracy is slightly behind with every epoch of training and even reaches its peak a bit earlier on some models but does not show many other differences in terms of progression.

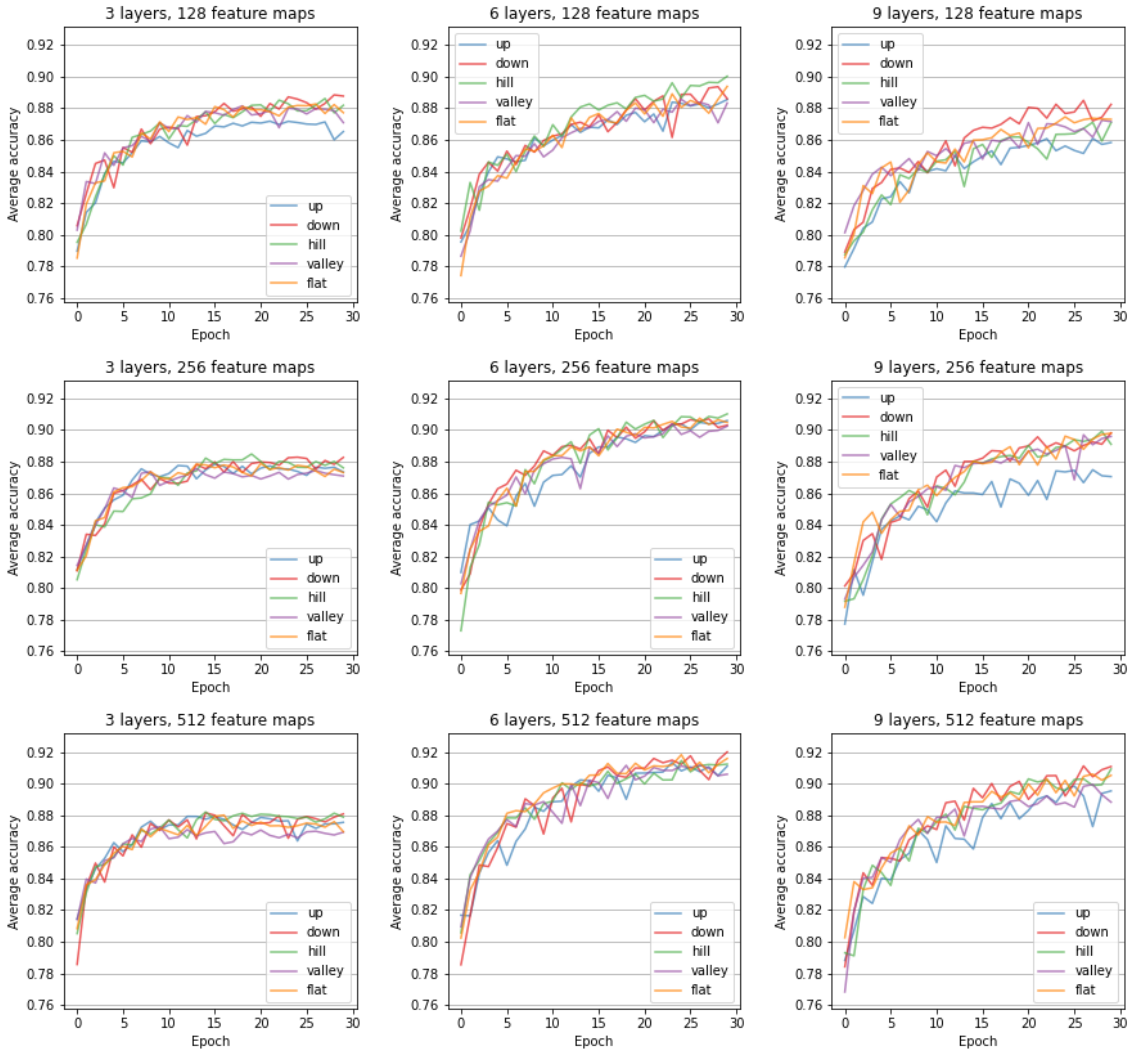


***Figure 11.*** *Average validation accuracy per epoch of each unique architecture design. Each graph represents architectures of the same size.*

## 4.4. Training time

The total training time of models is more constant for each unique architecture than other metrics and offers easier to interpret results. There is a clear relationship between the total feature maps extracted in an architecture and the time it takes to train. Moreover, the number of layers on an architecture affects each design differently, some increase in training time when more layers are added to the network while others train the longest at 6 layers.

Focusing on individual designs, *up* is the design that trains the fastest in most cases, especially on bigger models. Furthermore, *flat* is the next to train faster all around, except in some models with 512 feature maps. Finally, *down* tends to be the hardest to train, made clear in larger models, where it can take more than twice as long to train compared to other designs with the same number of features.

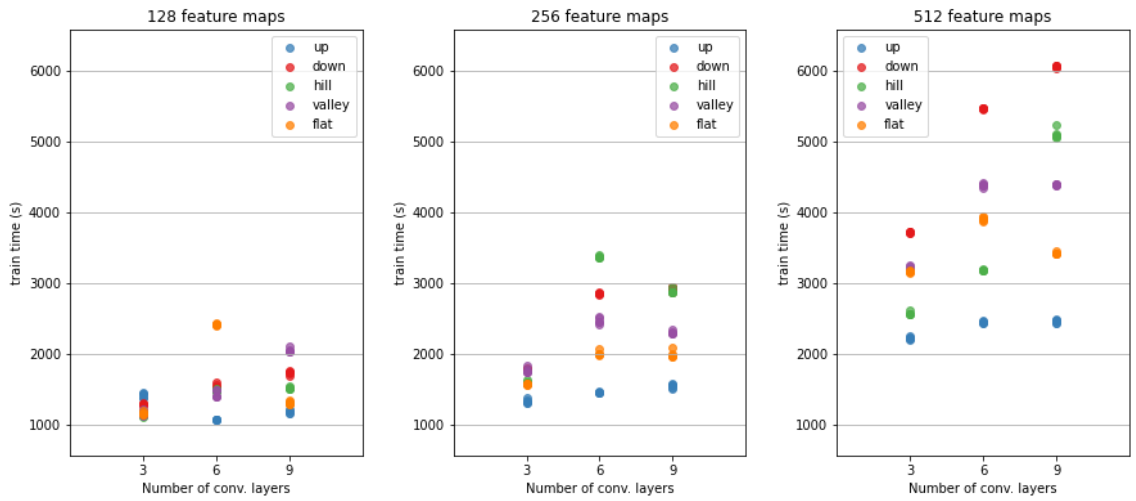


***Figure 12.*** *Training time of every model. Each graph contains models with the same architecture width and groups them by their architecture depth.*

This can be put into context by comparing the average validation accuracy per second of training of every model in *Figure 13*. Firstly, given the same amount of time for training, some models outperform others that they do not with a set number of epochs. Specifically, the *up* models outperform a lot of other models by the time they complete 30 epochs, as they do so in a shorter amount of time; the *flat* models follow a similar trend but sometimes also perform better than other fully trained models that train for a longer time.

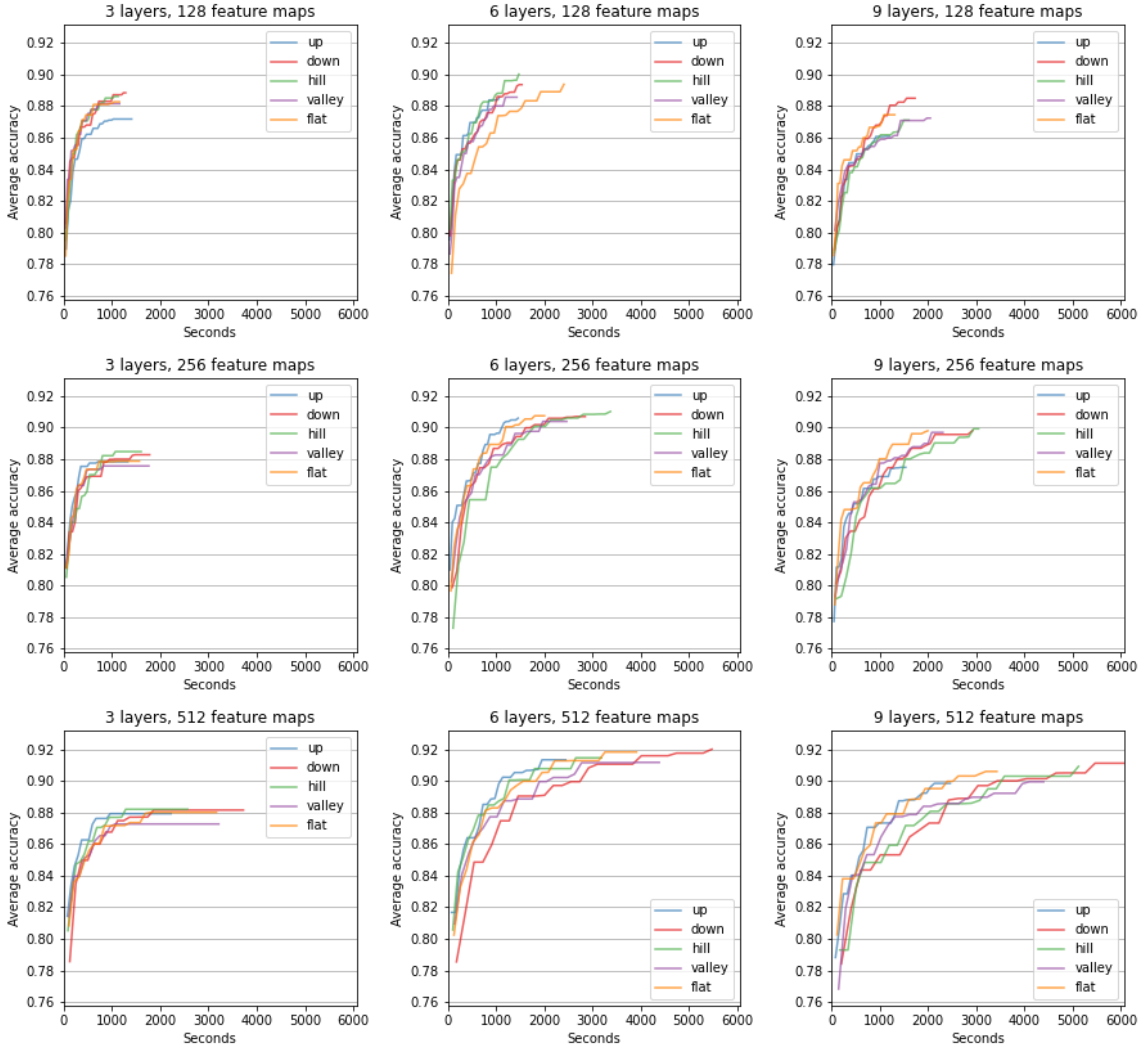


***Figure 13.*** *Average validation accuracy per second of each unique architecture design, excluding models with an accuracy score under 0.6. Each graph represents architectures of the same size.*

These differences could be due to the fact that the resolution of feature maps gets lowered with each pooling layer; so, feature maps in the last layers are easier to extract than feature maps in the earlier ones.

# Chapter 5
# Discussion

This project does not seek to find the best architecture to solve the lymph node image classification problem, as well-known architectures like AlexNet [13], VGG [14], GoogleNet [15] or ResNet [16] have already been applied to this task [17][18][19]. Instead, the goal was to see how specific design decisions affect image classification in this dataset. Nevertheless, for the sake of comparison, the best architectures designed for this project approach or even match the 92-98% accuracy range of the well-known solutions, as seen in *Table 2* in the Appendix. From this we can say that even though they were developed to be simple to easily reflect changes in their design, these architectures still perform well relative to architectures considered state-of-the-art until recently (all developed between 2012 and 2015) and can therefore offer meaningful insight on design decisions.

## 5.1. Number of features

From the results obtained in this study, it appears that extracting more features translates into higher accuracy, as long as the architecture has enough layers to support them. The only models that do not perform better when increasing total number of feature maps are the ones with 3 layers, and even then, the difference in performance is not very significant (up to -2% variation between the same model with 128 and 512 features). However, increasing the number of features also increases the training complexity of a model significantly, so it takes longer to achieve said results.

## 5.2. Number of layers

There is a limit to how many layers you can use with a certain number of features before you start losing accuracy, contrary to my original hypothesis that more layers would result in higher accuracy. It seems that extracting a small number of features per convolutional layer may hinder performance, or there might be a limit to the usefulness of stacking convolutions, an idea that has been explored in the development of newer architectures like ResNet [16]. The cause of this could be explored by testing architectures with more than 512 total feature

maps, as the difference in performance between 6 and 9 layers seems to get smaller when using more feature maps.

## 5.3. Distribution of features

Focusing on the performance of the different architecture design shapes individually:

- *Up* architectures tend to give worse results than the rest, but at the same time they are the fastest to train. Some models with this architecture fail to train, specifically the ones with the most features and layers, so if choosing to work with this design, this should be taken into account. This design is probably best for networks that have to be trained fast or with limited computational power.
- *Down* is one of the best performing architectures if we ignore the times that it fails to train, which is 1/5 for every model with 512 features. It is also the hardest to train in terms of time, which makes the cases were training does not work even worse. These are considerable drawbacks when working with big models and not many resources, which is one of the reasons why I could not work with more feature maps in this project. This being said, given enough resources this is probably the design choice to reach maximum accuracy.
- *Hill* is an all-around good performer but is very unpredictable both in terms of accuracy and training times. Results vary depending on network and layer size, so the time that is saved by training faster might be lost in working on design and testing. *Down* is probably a better option if looking for the highest accuracy.
- *Valley* is, together with *up*, amongst the worst performers. It is also harder to train than *up*, and even other models that tend to outperform it like *flat* or *hill*. Therefore, I would not recommend it, as there are better options for most use cases.
- *Flat* is another average performer, but more constant than *hill*. Furthermore, it is amongst the easiest to train, generally being only slightly slower than *up*. It seems like a good choice to start with for a basic model because of its simple design, and good performance per training time.

## 5.4. Limitations

All the previous results and observations have been made with only a single dataset, and the number of feature maps and their distribution may be related with the type of images being analyzed and the complexity of the components that define them. This means that while the conclusions of this project are good for datasets similar to *PCam*, it is hard to extrapolate to other image classification tasks without more research. Datasets with more variety in components and more classes might require more layers or different feature distributions than the ones observed here. Furthermore, this set contains relatively small images of 96x96 pixels, which might also affect the design of a good architecture.

Other limitations that may have affected the relevance of results are the diversity of models trained and the number of times that each one is trained. In this case each model has only been trained 5 times due to the scope of this project and its time limitations, which makes it hard for accurate statistical analysis to be performed. With more computational power, a bigger sample size would have allowed for more fair comparisons.

# Chapter 6
# Conclusion

## 6.1. Conclusion

The first conclusion of this project is that there is a significant difference in performance between different feature distributions in convolutional neural networks, but also an important difference in training complexity. With this in mind, one can choose to look for a balance between accuracy and complexity by deliberately working with a sub-optimal model accuracy-wise, especially if not many resources are available.

We have also observed that even if not all designs react exactly the same with an increase in total features, they do tend to follow a similar pattern of improvement, and there are some that generally perform better than others. If this stays true for bigger models, then finding a good model with a small number of features and then increasing them for optimal results might save a lot of testing time.

When it comes to the best overall design, *down* looks like a good first recommendation from looking at the bigger models' accuracies, but it has training problems, lacks consistency and does not get as good AUC scores, so it might be a good option with some modifications. However, I find that the *flat* and *hill* designs are generally better choices, as they have good overall results and are easier to train, and in the case of flat, also easier to design. Finally, a case could be made for using the *up* design when working with limited resources if the best accuracy is not needed, as it is the easiest to train and can still perform well. The only model I probably would not recommend is *valley*, as there are better options for most use cases.

## 6.2. Future work

As discussed in the previous chapter, one of the biggest limitations of this project is having only used one dataset. Consequently, a first step to continue this study would be to run the same experiment on other datasets; this could be other medical datasets or more casual images, to see the relationship between design and different image characteristics. The main reason why this has not fit into the current project's scope is that every dataset comes in different sizes and shapes, so these would have to either be resized to 96x96, which is not a

very high resolution, or every architecture would have to be manually modified to accept higher resolutions, resulting in an increased development time as well as greatly increased training times.

Other possible expansions for this study include training each model more times, exploring bigger networks, a wider range of number of layers, more feature distribution designs, or focusing on less designs like the better performing ones found here.

# Appendix

| | Accuracy | | | AUC ROC | | |
|---|---|---|---|---|---|---|
| **Architecture** | **Average** | **Std Deviation** | **90th Percentile** | **Average** | **Std Deviation** | **90th Percentile** |
| Up-128-3 | 0.870767 | 0.002402 | 0.873517 | 0.93033 | 0.001543 | 0.93192 |
| Up-128-6 | 0.893275 | 0.004499 | 0.897449 | 0.945344 | 0.003238 | 0.948119 |
| Up-128-9 | 0.856527 | 0.006567 | 0.864082 | 0.920603 | 0.005801 | 0.926925 |
| Up-256-3 | 0.879035 | 0.005028 | 0.884711 | 0.934822 | 0.00584 | 0.940358 |
| Up-256-6 | 0.905637 | 0.005875 | 0.912164 | 0.953161 | 0.004061 | 0.957813 |
| Up-256-9 | 0.875791 | 0.013919 | 0.890427 | 0.933917 | 0.013152 | 0.946232 |
| Up-512-3 | 0.878817 | 0.00355 | 0.88301 | 0.935703 | 0.002783 | 0.939105 |
| Up-512-6 | 0.914419 | 0.003182 | 0.917504 | 0.955768 | 0.001387 | 0.95716 |
| Up-512-9 | 0.714775 | 0.149338 | 0.898497 | 0.748145 | 0.175802 | 0.947595 |
| Down-128-3 | 0.888449 | 0.001224 | 0.889557 | 0.945927 | 0.002768 | 0.948511 |
| Down-128-6 | 0.89371 | 0.004676 | 0.899387 | 0.942007 | 0.003853 | 0.946736 |
| Down-128-9 | 0.886887 | 0.001652 | 0.888766 | 0.940515 | 0.002752 | 0.943482 |
| Down-256-3 | 0.881685 | 0.003863 | 0.886432 | 0.938696 | 0.003355 | 0.942426 |
| Down-256-6 | 0.908327 | 0.00373 | 0.912915 | 0.953812 | 0.005946 | 0.961255 |
| Down-256-9 | 0.897547 | 0.006598 | 0.904252 | 0.946322 | 0.005279 | 0.951851 |
| Down-512-3 | 0.820985 | 0.11416 | 0.883386 | 0.874283 | 0.123549 | 0.939666 |
| Down-512-6 | 0.854905 | 0.131039 | 0.922725 | 0.867044 | 0.186286 | 0.962841 |
| Down-512-9 | 0.848655 | 0.127908 | 0.914339 | 0.907439 | 0.099292 | 0.959427 |
| Hill-128-3 | 0.884256 | 0.003267 | 0.887876 | 0.941146 | 0.002449 | 0.943729 |
| Hill-128-6 | 0.898991 | 0.002637 | 0.901147 | 0.944521 | 0.004063 | 0.94911 |
| Hill-128-9 | 0.871796 | 0.006247 | 0.879608 | 0.927488 | 0.005562 | 0.934062 |
| Hill-256-3 | 0.881744 | 0.003322 | 0.884276 | 0.940176 | 0.001323 | 0.941627 |
| Hill-256-6 | 0.910937 | 0.001854 | 0.912955 | 0.953456 | 0.003964 | 0.957868 |
| Hill-256-9 | 0.898398 | 0.004111 | 0.901978 | 0.948538 | 0.003304 | 0.95224 |
| Hill-512-3 | 0.882714 | 0.002356 | 0.884395 | 0.939206 | 0.003132 | 0.942549 |
| Hill-512-6 | 0.852512 | 0.129852 | 0.920313 | 0.865601 | 0.182849 | 0.958531 |
| Hill-512-9 | 0.846598 | 0.126889 | 0.911926 | 0.864905 | 0.182465 | 0.957757 |
| Valley-128-3 | 0.879589 | 0.002641 | 0.882002 | 0.937591 | 0.001645 | 0.939291 |
| Valley-128-6 | 0.888113 | 0.004181 | 0.892108 | 0.940139 | 0.004292 | 0.943957 |
| Valley-128-9 | 0.874347 | 0.005778 | 0.880894 | 0.93148 | 0.004328 | 0.936473 |
| Valley-256-3 | 0.874604 | 0.004844 | 0.879964 | 0.932132 | 0.003549 | 0.936256 |
| Valley-256-6 | 0.906032 | 0.005255 | 0.91248 | 0.952591 | 0.004354 | 0.957549 |
| Valley-256-9 | 0.897903 | 0.003099 | 0.900277 | 0.947063 | 0.002056 | 0.949175 |
| Valley-512-3 | 0.86962 | 0.00432 | 0.873873 | 0.92771 | 0.002244 | 0.930505 |
| Valley-512-6 | 0.914458 | 0.003843 | 0.918354 | 0.955314 | 0.004783 | 0.961342 |
| Valley-512-9 | 0.839379 | 0.123271 | 0.902888 | 0.857147 | 0.178591 | 0.948834 |
| Flat-128-3 | 0.882753 | 0.001556 | 0.884593 | 0.939079 | 0.003049 | 0.942011 |
| Flat-128-6 | 0.89462 | 0.006139 | 0.902235 | 0.943283 | 0.002505 | 0.945788 |
| Flat-128-9 | 0.88303 | 0.006728 | 0.888726 | 0.935229 | 0.003803 | 0.937833 |
| Flat-256-3 | 0.879925 | 0.003094 | 0.883465 | 0.936864 | 0.003169 | 0.939831 |
| Flat-256-6 | 0.912124 | 0.001973 | 0.914201 | 0.954278 | 0.00424 | 0.959619 |
| Flat-256-9 | 0.89998 | 0.00527 | 0.906448 | 0.94816 | 0.005306 | 0.953581 |
| Flat-512-3 | 0.764419 | 0.140085 | 0.880241 | 0.815992 | 0.169735 | 0.934313 |
| Flat-512-6 | 0.914676 | 0.002927 | 0.917247 | 0.957971 | 0.00225 | 0.960018 |
| Flat-512-9 | 0.905024 | 0.001777 | 0.90627 | 0.947537 | 0.004533 | 0.953065 |

***Table 2**. Unique architecture design score distribution metrics. Each row represents a unique architecture with its accuracy and AUC ROC distribution metrics specified. For each metric, good values are highlighted in green, and the worst values are highlighted in red*

# Bibliography


[1] International Agency for Research on Cancer, “Latest global cancer data: Cancer burden rises to 18.1 million new cases and 9.6 million cancer deaths in 2018”, World Health Organisation, 2018.

[2] S. Sengupta et al., "A review of deep learning with special emphasis on architectures, applications and recent trends", *Knowledge-Based Systems*, vol. 194, p. 105596, 2020. Available: 10.1016/j.knosys.2020.105596

[3] A. Carvalho, F. Ramos and A. Chaves, "Metaheuristics for the feedforward artificial neural network (ANN) architecture optimization problem", *Neural Computing and Applications*, vol. 20, no. 8, pp. 1273-1284, 2010. Available: 10.1007/s00521-010-0504-3

[4] T. Gupta and K. Raza, "Optimization of ANN Architecture: A Review on Nature-Inspired Techniques", *Machine Learning in Bio-Signal Analysis and Diagnostic Imaging*, pp. 159-182, 2019. Available: 10.1016/b978-0-12-816086-2.00007-2

[5] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436-444, 2015. Available: 10.1038/nature14539

[6] S. Albawi, T. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network", *2017 International Conference on Engineering and Technology (ICET)*, 2017. Available: 10.1109/icengtechnol.2017.8308186

[7] S. S., J. Zong Chen and S. Shakya, "Survey on Neural Network Architectures with Deep Learning", *Journal of Soft Computing Paradigm*, vol. 2, no. 3, pp. 186-194, 2020. Available: 10.36548/jscp.2020.3.007

[8] B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, M. Welling. "Rotation Equivariant CNNs for Digital Pathology". arXiv:1806.03962

[9] B. Ehteshami Bejnordi et al., "Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer", *JAMA*, vol. 318, no. 22, p. 2199, 2017. Available: 10.1001/jama.2017.14585

[10] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie and M. Li, "Bag of Tricks for Image Classification with Convolutional Neural Networks", *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 558-567, 2019. Available: 10.1109/cvpr.2019.00065

[11] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv:1609.04747

[12] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition", Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998. Available: 10.1109/5.726791.

[13] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", Communications of the ACM, vol. 60, no. 6, pp. 84-90, 2017. Available: 10.1145/3065386.

[14] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv:1409.1556*

[15] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015. arXiv:1409.4842

[16] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016. arXiv:1512.03385

[17] Y. Sun, F. A. Binti Hamzah and B. Mochizuki, "Optimized Light-Weight Convolutional Neural Networks for Histopathologic Cancer Detection," *2020 IEEE 2nd Global Conference on Life Sciences and Technologies (LifeTech)*, 2020, pp. 11-14, doi: 10.1109/LifeTech48969.2020.1570619224.

[18] C. Vo-Le, N. H. Son, P. Van Muoi and N. H. Phuong, "Breast Cancer Detection from Histopathological Biopsy Images Using Transfer Learning," 2020 IEEE Eighth International Conference on Communications and Electronics (ICCE), 2021, pp. 408-412, doi: 10.1109/ICCE48956.2021.9352069.

[19] Wang, Dayong, Aditya Khosla, Rishab Gargeya, Humayun Irshad, and Andrew H. Beck. "Deep learning for identifying metastatic breast cancer." arXiv:1606.05718