Image Classification Using K-Fold Cross Validation

E. Erdogdu[1]

July, 2022

Abstract

This study measures different machine learning networks and their performance in cats and dogs dataset. K-Fold Cross Validation with 5 folds were used for the study. Our experiments conclude that CNN with 8 starting layers is the better choice for the given dataset.

List of Symbols

The next list describes several symbols that will be later used within the body of the document

Other symbols

CNN	Convolutional Neural Network
$RMSProp$	Root Mean Squared Propagation
SGD	Stochastic Gradient Descent
VGG	Visual Geometry Group

1. Introduction

Image classification is where a computer can analyse an image and identify the 'class' the image falls under. (Or a probability of the image being part of a 'class'.) A class is essentially a label, for instance, 'car', 'animal', 'building' and so on.

This project aims to create multiple machine learning mdoels to predict if a photo given to the model is either a cat, or a dog. Three different network models were used to find the most efficient and accurate model for image classification.

2. Dataset

For our goal, we are going to use the Cats and Dogs dataset that is given for the project. This dataset has a total of 25000 images, 12500 of them are cats, and the other 12500 are dogs.

These images are all informal pet images like the ones we usually have on our phones, and this is important because it means that our model will be more versatile. Otherwise, it would only be able to understand a pet type if the picture had a certain characteristic or was taken in a specific manner.

A sample image from the dataset can be seen at Figure 1.

3. Models Used

3.1. VGG

VGG stands for Visual Geometry Group; it is a standard deep Convolutional Neural Network (CNN) architecture with multiple layers. The "deep" refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers.

The VGG architecture is the basis of ground-breaking object recognition models. Developed as a deep neural network, the VGGNet also surpasses baselines on many tasks and datasets beyond ImageNet. Moreover, it is now still one of the most popular image recognition architectures.

This study uses VGG-3 and VGG-13 as the two different machine learning networks.

To make the study simple, VGG-13 was chosen as it is the easier to implement out of multi layered machine learning networks. Following the same logic VGG-16 and VGG-19 can also be easily implemented for the dataset. fig. 4 describes VGG-13 and others in a table.

VGG13 is model B in fig. 4. A diagram can be found in fig. 3 to simplify how VGG-13 works in back.

3.2. LeNet

LeNet was introduced in the research paper "Gradient-Based Learning Applied To Document Recognition" in the year 1998 by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Many of the listed authors of the paper have gone on to provide several significant academic contributions to the field of deep learning.

LeNet-5 CNN architecture is made up of 7 layers. The layer composition consists of 3 convolutional layers, 2 subsampling layers and 2 fully connected layers.

fig. 2 shows a depiction of the LeNet-5 architecture, as illustrated in the original paper.

The LeNet architecture is an excellent "first architecture" for Convolutional Neural Networks. LeNet is small and easy to understand — yet large enough to provide interesting results. Furthermore, the combination of LeNet + MNIST is able to run on the CPU,

Figure 1: Sample image from dataset.

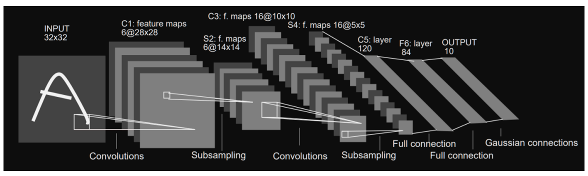


Figure 2: LeNet-5 Architecture

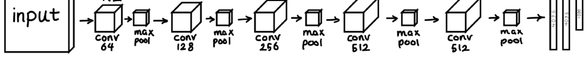
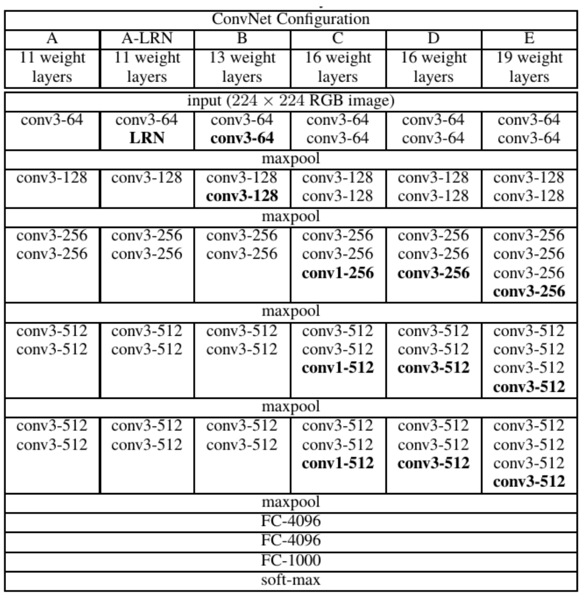


Figure 3: Flow Diagram for VGG-13 Architecture



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 $\times$ 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 4: List Of VGG Architectures.

making it easy for beginners to take their first step in Deep Learning and Convolutional Neural Networks.

3.3. CNN

A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data.

CNNs are powerful image processing, artificial intelligence (AI) that use deep learning to perform both generative and descriptive tasks, often using machine vison that includes image and video recognition, along with recommender systems and natural language processing (NLP).

A neural network is a system of hardware and/or software patterned after the operation of neurons in the human brain. Traditional neural networks are not ideal for image processing and must be fed images in reduced-resolution pieces. CNN have their "neurons" arranged more like those of the frontal lobe, the area responsible for processing visual stimuli in humans and other animals. The layers of neurons are arranged in such a way as to cover the entire visual field avoiding the piecemeal image processing problem of traditional neural networks.

3.4. k-Fold Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

For cross validation kfold method from sklearn library is used. KFold method automatizises the cross validation that we do in our models' training. Figure fig. 5 represents how our 5 fold cross validation works in action.

How the K-Fold Cross validation is implemented is discussed in Appendix section of this paper.

4. Parameters

In the study, different parameters are used in each machine learning network. These are in order:

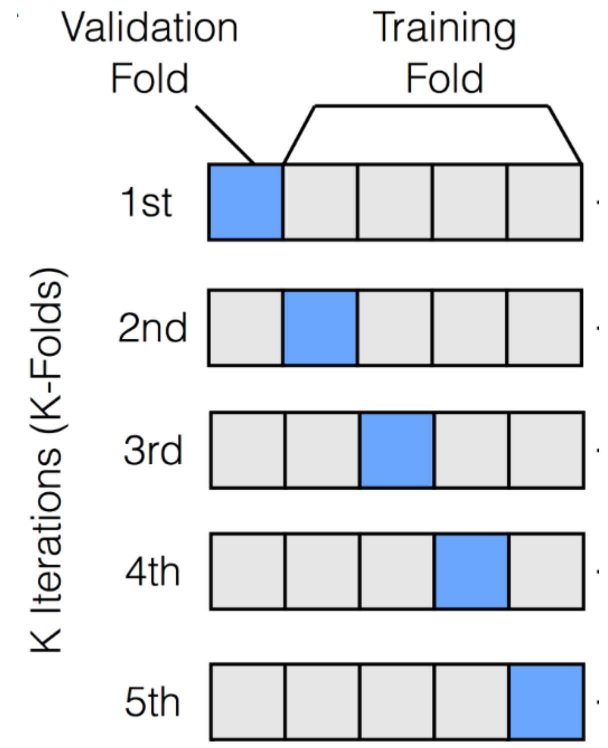


Figure 5: 5-Fold Cross Validation Visualization

4.1. Batch Size

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

4.2. SGD

Stochastic Gradient Descent, or SGD for short, is an optimization algorithm used to train machine learning algorithms, most notably artificial neural networks used in deep learning.

The job of the algorithm is to find a set of internal model parameters that perform well against some performance measure such as logarithmic loss or mean squared error.

Optimization is a type of searching process and you can think of this search as learning. The optimization algorithm is called "gradient descent", where "gradient" refers to the calculation of an error gradient or slope of error and "descent" refers to the moving down along that slope towards some minimum level of error.

4.3. Epoch

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm.

5. Data Processing

The dataset of cats and dogs are compromised of 25000 images that are 400 by 500 photos.

The first step of generating the training data is grayscaling it. Grayscaling is done by OpenCV's IMREAD_GRAYSCALE option.

After grayscaling the images, the next step is resizing it. For resizing, a 128 by 128 resolution were chosen, because it was the better resolution for quality and size. The resizing task was also done by OpenCV.

Processed dataset is split by 90% train and 10% test for checking the accuracies of each network and their one zero losses.

After these steps, the images are saved to use in our training and foldings.

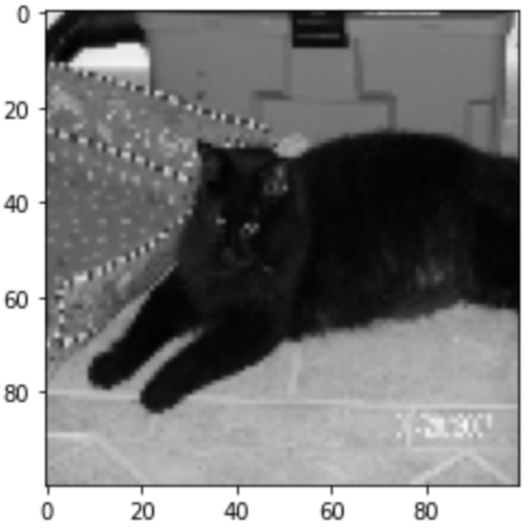


Figure 6: A gray and white, subsampled photo of a cat

6. Results

Here are the results that's collected for each of our machine learning networks

6.1. CNN

Our CNN model performed the best out of the networks that we used. After 5 folds, the validation accuracies were around 70% and our one zero losses are 0.15 mark. For the training of CNN, RMSprop were used for the optimiser.For the learning rate of RMSProp, 0.001 was chosen because with lower learning rates yielded lower validation accuracies. And for our each fold, 10 epoches with 50 steps were done.

Epoch is left at 10, because the more epoches were executed, the overfitting started to occur on the model. Used CNN network has 3 layers with pooling inbetween. For each convolusional layer, relu activation was used differentiate between the inputs and outputs.

6.2. VGG13

Our VGG13 is the second best performant model out of our models. It performed worse than the CNN model, because the VGG13 networks is not the best suited for black and white images. After 5 folds, the validation accuracies were around 60% and our one zero losses are 0.35 mark. For the training of VGG13, RMSProp with a lower learning rate were used for the optimiser. And for our each fold, 10 epoches with no steps were done.

6.3. LeNet

Our LeNet is the third best performant model out of our models. It performed rather poorly because of the fact that LeNet needs images to be high quality as much as possible. After 5 folds, the validation accuracies were around 30% and our one zero losses are 0.70 mark. For the training of LeNet, adam with a lower learning rate were used for the optimiser. 20 epoches were done with no incremental steps were done.

6.4. VGG3

VGG3 is the last network implementation that we used. VGG3 was not suited for our testings, and the results that we collected shows it. We got at max 20% validation accuracy rate, and the one zero loss is nearly 1. VGG3 is not a good choice for informal photos such as what we have in our dataset, because of the fact that the layers are too small to handle informal photos that are big in resolution.

After finishing all the tests, we can conclude that CNN architecture with 8 starting layers with the RMSProp optimiser was the best choice for the kind of dataset that we used. The results may have varied if the images were not black and white, or if the images were higher resolution from what we chose.

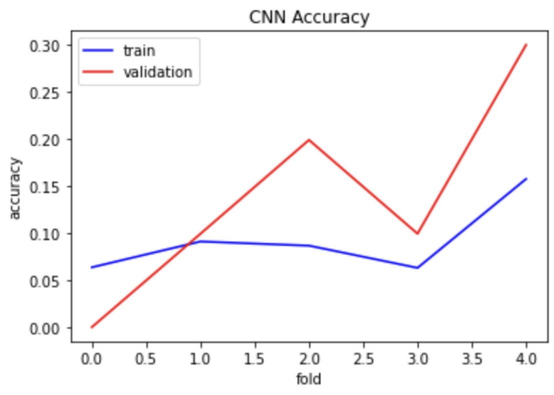


Figure 7: Validation Accuracies of CNN

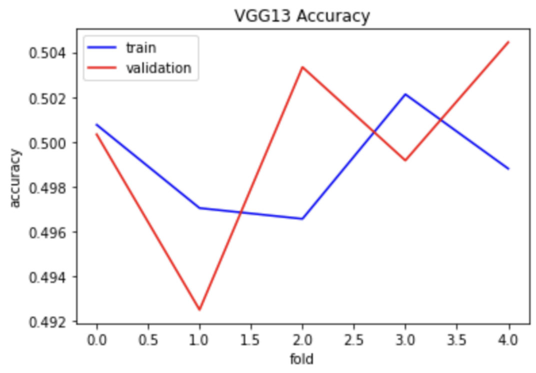


Figure 8: Validation Accuracies of VGG13

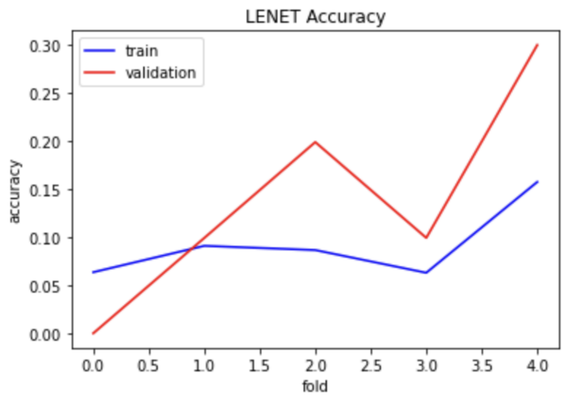


Figure 9: Validation Accuracies of LeNet

Appendix A. Additional Information

Appendix A.1. Python codes

All of the data processing and network trainings are done on Cats_And_Dogs.ipynb file. In listing 1 a basic implementation of cross validation can be seen.

```python
from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
        print('train: %s, test: %s' % (data[train], data[test]))

#outputs:
#train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
#train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
#train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
```

Listing 1: Basic K-Fold implementation

The code from listing 1 was displayed using a `.py` file. Since the lines are not numbered in this code example, you can copy and paste the code from the PDF into python without many issues (does however need to correct indents).

The way cross folding that is implemented in the notebook is vastly different from what its written on here. This code is just an example for it.

Appendix B. Model Summaries

You can see the model summaries from figure fig. B.11 to fig. B.14

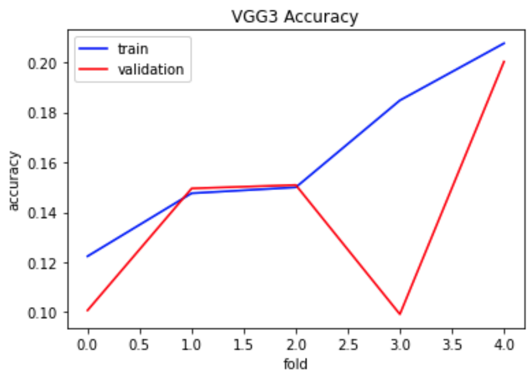
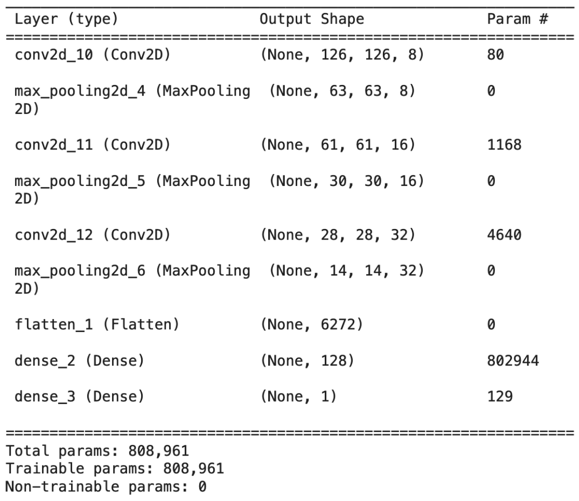
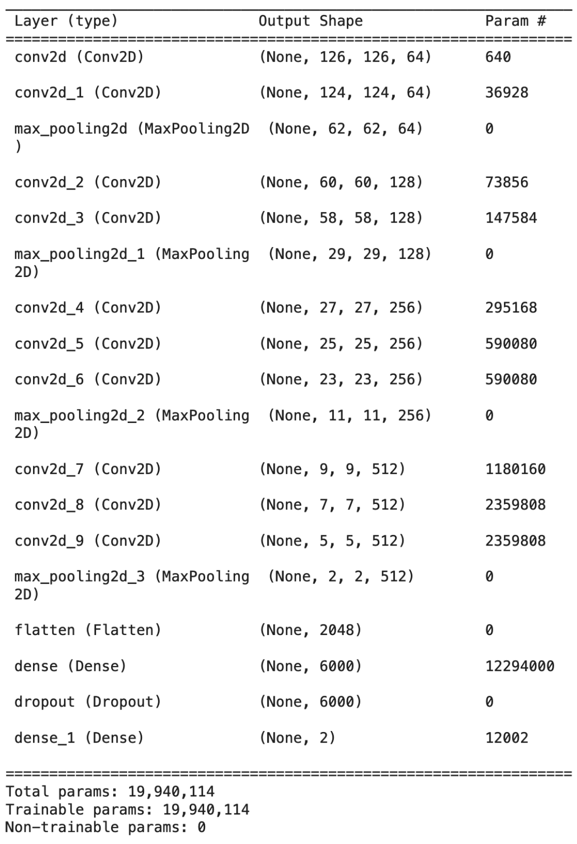


Figure 10: Validation Accuracies of VGG3



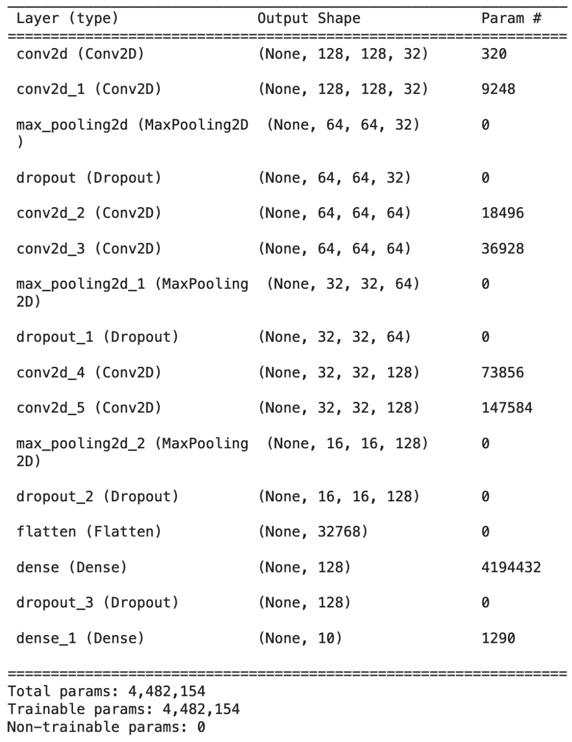
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 126, 126, 8)       80

max_pooling2d_4 (MaxPooling  (None, 63, 63, 8)         0
2D)

conv2d_11 (Conv2D)           (None, 61, 61, 16)        1168

max_pooling2d_5 (MaxPooling  (None, 30, 30, 16)        0
2D)

conv2d_12 (Conv2D)           (None, 28, 28, 32)        4640

max_pooling2d_6 (MaxPooling  (None, 14, 14, 32)        0
2D)

flatten_1 (Flatten)          (None, 6272)              0

dense_2 (Dense)              (None, 128)               802944

dense_3 (Dense)              (None, 1)                 129

=================================================================
Total params: 808,961
Trainable params: 808,961
Non-trainable params: 0
```

Figure B.11: CNN Summary



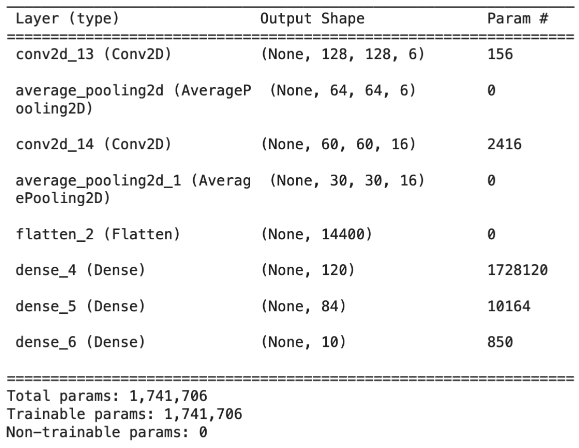
```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 64)      640

 conv2d_1 (Conv2D)           (None, 124, 124, 64)      36928

 max_pooling2d (MaxPooling2D  (None, 62, 62, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 conv2d_3 (Conv2D)           (None, 58, 58, 128)       147584

 max_pooling2d_1 (MaxPooling  (None, 29, 29, 128)      0
 2D)

 conv2d_4 (Conv2D)           (None, 27, 27, 256)       295168

 conv2d_5 (Conv2D)           (None, 25, 25, 256)       590080

 conv2d_6 (Conv2D)           (None, 23, 23, 256)       590080

 max_pooling2d_2 (MaxPooling  (None, 11, 11, 256)      0
 2D)

 conv2d_7 (Conv2D)           (None, 9, 9, 512)         1180160

 conv2d_8 (Conv2D)           (None, 7, 7, 512)         2359808

 conv2d_9 (Conv2D)           (None, 5, 5, 512)         2359808

 max_pooling2d_3 (MaxPooling  (None, 2, 2, 512)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 6000)              12294000

 dropout (Dropout)           (None, 6000)              0

 dense_1 (Dense)             (None, 2)                 12002

=================================================================
Total params: 19,940,114
Trainable params: 19,940,114
Non-trainable params: 0
```

Figure B.12: VGG13 Summary



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      320

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      9248

 max_pooling2d (MaxPooling2D  (None, 64, 64, 32)       0
 )

 dropout (Dropout)           (None, 64, 64, 32)        0

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496

 conv2d_3 (Conv2D)           (None, 64, 64, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 32, 32, 64)        0

 conv2d_4 (Conv2D)           (None, 32, 32, 128)       73856

 conv2d_5 (Conv2D)           (None, 32, 32, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 dropout_2 (Dropout)         (None, 16, 16, 128)       0

 flatten (Flatten)           (None, 32768)             0

 dense (Dense)               (None, 128)               4194432

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 4,482,154
Trainable params: 4,482,154
Non-trainable params: 0
```

Figure B.13: VGG3 Summary



```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 128, 128, 6)       156

 average_pooling2d (AverageP  (None, 64, 64, 6)        0
 ooling2D)

 conv2d_14 (Conv2D)          (None, 60, 60, 16)        2416

 average_pooling2d_1 (Averag  (None, 30, 30, 16)       0
 ePooling2D)

 flatten_2 (Flatten)         (None, 14400)             0

 dense_4 (Dense)             (None, 120)               1728120

 dense_5 (Dense)             (None, 84)                10164

 dense_6 (Dense)             (None, 10)                850

=================================================================
Total params: 1,741,706
Trainable params: 1,741,706
Non-trainable params: 0
```

Figure B.14: LeNet Summary