Cats and Dogs Images Classification with Cross Validation Risk Estimates

Federico Bassi, ID:993443

Machine Learning Project, September 2022

Abstract

In this project, I develop five Convolutional Neural Networks for images classification of cats and dogs. In particular, I study the effects on model performance of increasing the network's depth and the number of filters computed, the impact of regularization methods (such as Data Augmentation and Dropout) on overfitting, and the results of the hyperparameter tuning procedure. Finally, the risk of the best-performing model is estimated through a 5-fold cross validation procedure.

1 Introduction

Convolutional Neural Networks are today extremely powerful models to perform Computer Vision tasks. In this project, I develop five Convolutional Neural Networks for image classification of cats and dogs on a well known dataset, studying the impact on model performances of different architectures of the network and of different techniques. In particular, with Models 1-3 I study how the performances vary when the network becomes deeper and, therefore, more filters are computed. Secondly, I study the impact of regularization techniques such as Data Augmentation and Dropout on overfitting (Model 4). Finally, with Model 5 I tried to improve performances using an hyperparameter tuning procedure.

Model 4 results in having the best performances. A more accurate estimate of the risk of this model has therefore been computed through 5-fold Cross Validation.

This report is organized as follows: Section 2 describes the well-know Cats and Dogs Dataset, and the images preprocessing process. Section 3 gives some insights on the theory of Convolutional Neural Networks and the methodology followed throughout the project. In Section 4, I then describe the architecture of the models built and the experimental results obtained. Finally, Section 5 reports the conclusions.

2 Dataset and Data Preprocessing

The Dataset upon which this project has been developed is a very famous and widely used Dataset for Computer Vision tasks. The Dataset, which originally became famous as part of a Kaggle Competition in 2013, contains images of cats and dogs. The original Dataset contained 25000 images, 12500 cats' images and 12500 dogs' images.

2.1 Filtering of corrupted images

In order to improve the performances of the models in this Dataset, I removed from it all the corrupted images. This operation resulted in the removal of 1590 images [1]. The final version of the Dataset therefore contained 23 410 images, 11741 belonging to the Cat Class, 11669 belonging to the Dog Class.

2.2 Images transformation

All the images in the dataset have been converted from JPEG to RGB and scaled to the standard size (180,180). Therefore, the images in our dataset will be tensors of shape (180, 180, 3). To improve model perfomances, the RGB values have been rescaled from the [0, 255] range to the [0, 1] range.

3 Theoretical Framework and Methodology

This section describes the main features of Convolutionary Neural Networks (or *convets*), the kind of Neural Networks typically used to perform Computer Vision tasks, along with a short explanation of the methodology followed to build them in this project. First of all, I will describe the charachterstics of the layers of a convet, and the main differences with respect to a standard Neural Network.

3.1 Network's Layers

A typical architecture of a convet is composed of sequences of Convolutional and MaxPooling Layers (or blocks of layers of this type), followed by a Flattened Layer and a Dense one (see Figure 1). The Network receives an image as input and usually outputs the probability that an image belongs to each class.

[1] In particular, 2 images have been removed because their size was 0, while 1590 images were removed because the string "JFIF" was not present in their header

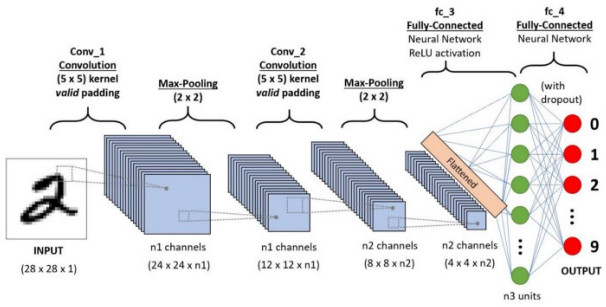

Figure 1: Architecture of a CNN

3.1.1 Convolution Layer

Convolution Layers perform the Convolution Operation, which allows the Network to learn local patterns of the image. Convolution Layer operate over rank-3 tensors called "feature map" (for example, an image received as input), with axis: (height, width, channel). In our case, since each image has been resized and transformed to RGB, the input tensors will have shape (180, 180, 3).

The Convolution Operation is basically a dot product between the input and a series of different filters, each of which is "slid" (see Figure 2[2]) on the input image, outputting a so-called "output feature map". This operation allows to detect the presence of patterns (from more simple ones, like "horizontal lines", to more complex ones, like "presence of a cat's ear") in the image.
The convolution operation is mainly defined by five parameters:

1. **Size of the filter** (or "convolutional kernel"): the dimension of each filters we are using to perform convolution. In our exercises, it has been fixed to 3*3;

2. **Number of filters**: How many different filters we are going to use. This parameter determines the depth of the output feature map. In this exercises, I used layers with 32, 64, 128 or 256 filters.

3. **Padding**: Given an arbitrary feature map, it won't be possible to center a 3*3 window in each of its tile: this will result in an output feature map that is slightly shrinked. For example, for a 5*5 input feature map, a convolutional kernel of 3*3 will yield an output feature map of 3*3. To overcome this problem, it is possible to use "padding", which consists in adding an appropriate number of rows and columns to the input feature map to make it possible to center convolution kernels around every tile of the input. In this exercise, i set padding to "same"; I therefore required the output map to have the same width and height of the input map.

4. **Stride**: Another parameter that influences the size of the output map is the "stride", i.e. the distance between two tiles in which the convolution is computed. In this exercise, I used the default value of 1, therefore, the tiles around which the convolution is performed will be contiguous.

5. **Activation Function**: The activation function defines if a neuron of the Network is activated or not. In this exercise, I used one of the most used function, ReLu function:

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

   ReLu returns 0 if the input it receives is negative, while it returns the input itself if this is positive. ReLu function extends the hypothesis space to non-linear functions.

3.1.2 Max Pooling Layer

Pooling layers are used to down-sample the feature map. Similarly to what happens in a Convolution Layer, in a Max Pooling Layer, a 2*2 window is slid over the input feature map with stride 2, but in this case, a max

[2]Convolutional Neural Networks — A Beginner's Guide - Medium

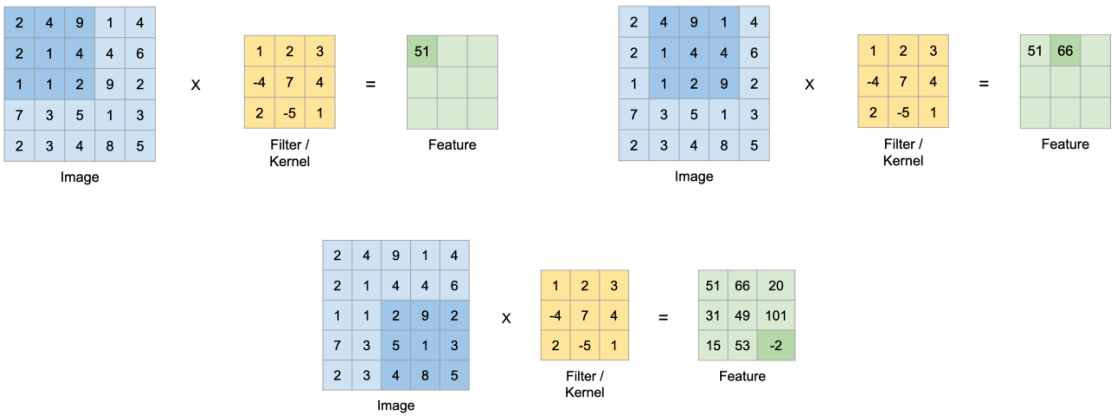

Figure 2: The Convolution operation over 1 filter

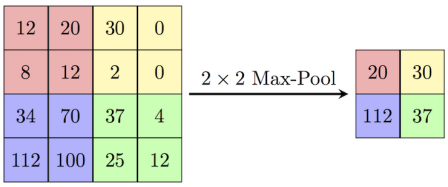

Figure 3: 2*2 Max Pooling Operation (stride=2)

operation[3] is perfomed (see Figure 2).
Why do we need downsampling? For two main reasons:

1. To reduce the number of coefficients to compute;
2. To allow deeper Convolution layers to learn high-level patterns. If we do not down-sample the image, deeper layers won't receive information about the totality of the input, but to a relatively small portion of it.

3.1.3 Flatten Layer and Dense Layer

Blocks of Convolution and MaxPooling Layers are usually followed by a Flatten Layer, which transforms the 3D input into a one-dimensional vector.

The one-dimensional vector outputted by the Flatten Layer can then pass through a series Dense Layers. These layers can learn complex relationships between the high-level features learnt by the preceding blocks of Convolution and MaxPooling Layers. This part of the convet is the one which performs the classification task, outputting the probability for each image of belonging to each class.

Dense Layers are defined by these hyperparameters:

1. **Number of node**: How many neurons each layer is composed of;
2. **Activation Function**: As for Convolution Layers, each layer can have an activation function. In this exercise, I used again ReLu for the hidden layer, while the Sigmoid Function has been used for the output layer composed of one neuron: this is the layer that outputs the probability that the image is a dog's or a cat's one. The Sigmoid function is a monotonic non linear function, with values in [0,1], typically used for binary classification: its output can be interpreted as probability that an image belongs to one class or the other.

[3]Average pooling can also be perfomed, but usually looking at the "maximal presence" of some features is more informative.

Figure 4: Output of the Data Augmentation Layer on a single image

3.1.4 Data Augmentation Layer

As we will see in Section 4, convets usually suffer from overfitting. This is why, in this exercise, I added to my models layers that "add randomness" to the inputs received[4].

The first strategy that I followed is to add a Data Augmentation Layer: this layer performs random operations on the image, allowing the model to see different aspects of it and, therefore, generalize better.

The kind of random operation that can be perfomed on an image could be, for example[5]:

1. Randomly flip horizontally half of the images;
2. Randomly rotate the images by a factor in the range [-10%, 10%];
3. Randomly zoom the image by a factor in the range [-20%,20%].

The results of performing these operation on an image present in our dataset can be seen in Figure 4.

3.1.5 Dropout Layer

The Data Augmentation Layer will output images that are not exactly the same, however, they will be highly correlated. To decorrelate the inputs, a regulariation technique called "dropout" can be used. Dropout consists in randomly setting to zero a number of output features (depending on the dropout rate) of the layer during the training. This will have the effect of adding noise to the output of a Layer, and thus reducing the variance error of the network.

3.2 Model Compilation

3.2.1 Optimizer

The optimization algorithm used in this exercise was Root Mean Squared Propagation ("RMSprop"). RMSpropr is an adaptive version of gradient descent, developed as an extension of the AdaGrad algorithm. Intuitively, AdaGrad adapts the gradient by dividing it by the sum of past gradients squared, with the idea of "prioritizing"

[4]For details of the models specification see Section 4

[5]Example taken from: Chollet, F. (2017). Deep learning with python. Manning Publications, p.222

the update of features not much updated before. RMSprop speeds up this process by "forgetting" gradients computed long ago, and keeping only recent gradients: this is done, intuitively, by dividing the gradient by a *decayed* sum of gradient squared.

3.2.2 Loss Function

Since the task of this exercise was binary classification, the loss function chosen was binary cross-entropy:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i))$$

where y_i is the label associated to each point, while $p(y_i)$ is its predicted probability. Intuitively, binary cross-entropy computes a (negative) average of the log of predicted probabilities.

3.2.3 Metrics

The metric chosen in this exercise is accuracy. Despite its simplicity, accuracy is an appropriate metric in this exercise, given that the classification classes are just two and the dataset is (almost) perfectly balanced. Accuracy simply computes the number of correctly predicted images as a fraction of the total number of predicted images.

4 Models description and Experimental Results

4.1 Models 1-3: Baseline models

4.1.1 Models Architecture

The first models I built were relatively simple convets, composed of blocks of Convolution (with ReLu activation function) and MaxPooling Layer, followed by two Dense Layers with 128 and 1 nodes, respectively with ReLu and Sigmoid activation function.

The differences among Model 1, 2 and 3 lies in the number of blocks of Convolution and MaxPooling (see Figure 5).
In particular:

1. **Model 1**: Two blocks of Convolution and Max Pooling layers, with 32 and 64 filters computed.
2. **Model 2**: Three blocks of Convolution and Max Pooling layers, with 32, 64 and 128 filters computed.
3. **Model 3**: Four blocks of Convolution and Max Pooling layers, with 32, 64, 128 and 256 filters computed.

In this case my aim was, first of all, to build a baseline model and to observe the behaviour of training and validation accuracy and loss over the epochs. In the second place, comparing the results of these three models will help us understand the effect of building deeper networks which computes higher number of filters: the expected result is that deeper model should learn more and more complex features of the image yielding, therefore, an higher test accuracy.

4.1.2 Models compilation and Fitting

Models 1-3 have been compiled and fitted according to the following specification:

1. **Optimizer**: RMSprop
2. **Loss Function**: Binary Crossentropy
3. **Metrics**: Accuracy
4. **Number of epochs**: 50

All the models have been trained on the same training dataset and evaluated on the same test dataset.

```python
# Model 1 Specification
def model_1_specification():
    inputs = keras.Input(shape = (180,180,3))

    # Rescale the values to the [0,1] range
    x = layers.Rescaling(1./255)(inputs)

    # First block w/ 32 filters
    x = layers.Conv2D(filters = 32, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Second block w/ 64 filters
    x = layers.Conv2D(filters = 64, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Flatten Layer
    x = layers.Flatten()(x)

    # Fully Connected Layers
    x = layers.Dense(128, activation = "relu")(x)
    outputs = layers.Dense(1, activation = "sigmoid")(x)

    model = keras.Model(inputs = inputs, outputs = outputs)
    return model
```

```python
# Model 2 specification
def model_2_specification():
    inputs = keras.Input(shape = (180,180,3))
    x = layers.Rescaling(1./255)(inputs)

    # First block w/ 32 filters
    x = layers.Conv2D(filters = 32, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Second block w/ 64 filters
    x = layers.Conv2D(filters = 64, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Thir block w/ 128 filters
    x = layers.Conv2D(filters = 128, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Flatten Layers
    x = layers.Flatten()(x)

    # Fully Connected Layers
    x = layers.Dense(128, activation = "relu")(x)
    outputs = layers.Dense(1, activation = "sigmoid")(x)

    model = keras.Model(inputs = inputs, outputs = outputs)
    return model
```

```python
# Model 3 specification
def model_3_specification():
    inputs = keras.Input(shape = (180,180,3))
    x = layers.Rescaling(1./255)(inputs)

    # First block w/ 32 filters
    x = layers.Conv2D(filters = 32, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Second block w/ 64 filters
    x = layers.Conv2D(filters = 64, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Third block w/ 128 filters
    x = layers.Conv2D(filters = 128, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Fourth block w/ 256 filters
    x = layers.Conv2D(filters = 256, kernel_size = 3, activation = "relu", padding="same")(x)
    x = layers.MaxPooling2D(pool_size = 2)(x)

    # Flatten Layer
    x = layers.Flatten()(x)

    # Fully Connected Layers
    x = layers.Dense(128, activation = "relu")(x)
    outputs = layers.Dense(1, activation = "sigmoid")(x)

    model = keras.Model(inputs = inputs, outputs = outputs)
    return model
```

Figure 5: Models 1-3 Architecture

4.1.3 Model Evaluation

The plots in Figure 6 describe the progress of training and test accuracy and loss over the epochs. These plots exhibit an overfitting problem: while training accuracy goes up in the training set over the epochs, test accuracy remains stable over the epochs (model 1), or improves slightly (models 2,3).
We would like to reduce the variance error of these models and therefore to reduce overfitting for two main reasons:

1. We would like our estimate of the accuracy and the loss to be a good estimate of the error the model makes on unseen data;
2. We would like the model to generalise well, i.e. to have good performances on unseen data.

Reducing the variance error was therefore the main aim of Model 4.

In the following table, we can see that adding blocks of Convolution and Max-Pooling Layers clearly improves the model performances: the model is able to learn better higher-order features of the images.

Model	Test Accuracy	Test Loss
Model 1	0.759	4.098
Model 2	0.824	3.731
Model 3	0.883	1.509

4.2 Model 4: Reducing overfitting

4.2.1 Model Architecture

The architecture of Model 4 reproduced the one of Model 3; however, a Data Augmentation Layer was placed as the first layer, while Dropout Layers were placed after each MaxPooling Layer, after the Flatten Layer and after the first fully connected layer, with a dropout rate of respectively 0.05, 0.2 and 0.2.

The aim was, in this case, studying the effects of such regularization techniques on overfitting: by adding noise through Data Augmentation and Dropout, we expect the variance error of the model to decrease and therefore, overfitting to decrease.

4.2.2 Model compilation and Fitting

The model has been compiled with the same parameters of Models 1-3.

4.2.3 Model Evaluation

As can be seen in Figure 7, in this model overfitting was reduced with respect to the previous experiment: training and validation accuracy and loss are closer, expecially during the first epochs. Training accuracy, in this case, is a more reliable approximation of test accuracy. Moreover, the model seems to be better at generalizing.

Model	Test Accuracy	Test Loss
Model 4	0.931	0.241

4.3 Model 5: Hyperparameter tuning

With Model 5, I performed hyperparameter tuining, a powerful technique used to choose optimal hyperparameters values, with the aim of improving model performance.

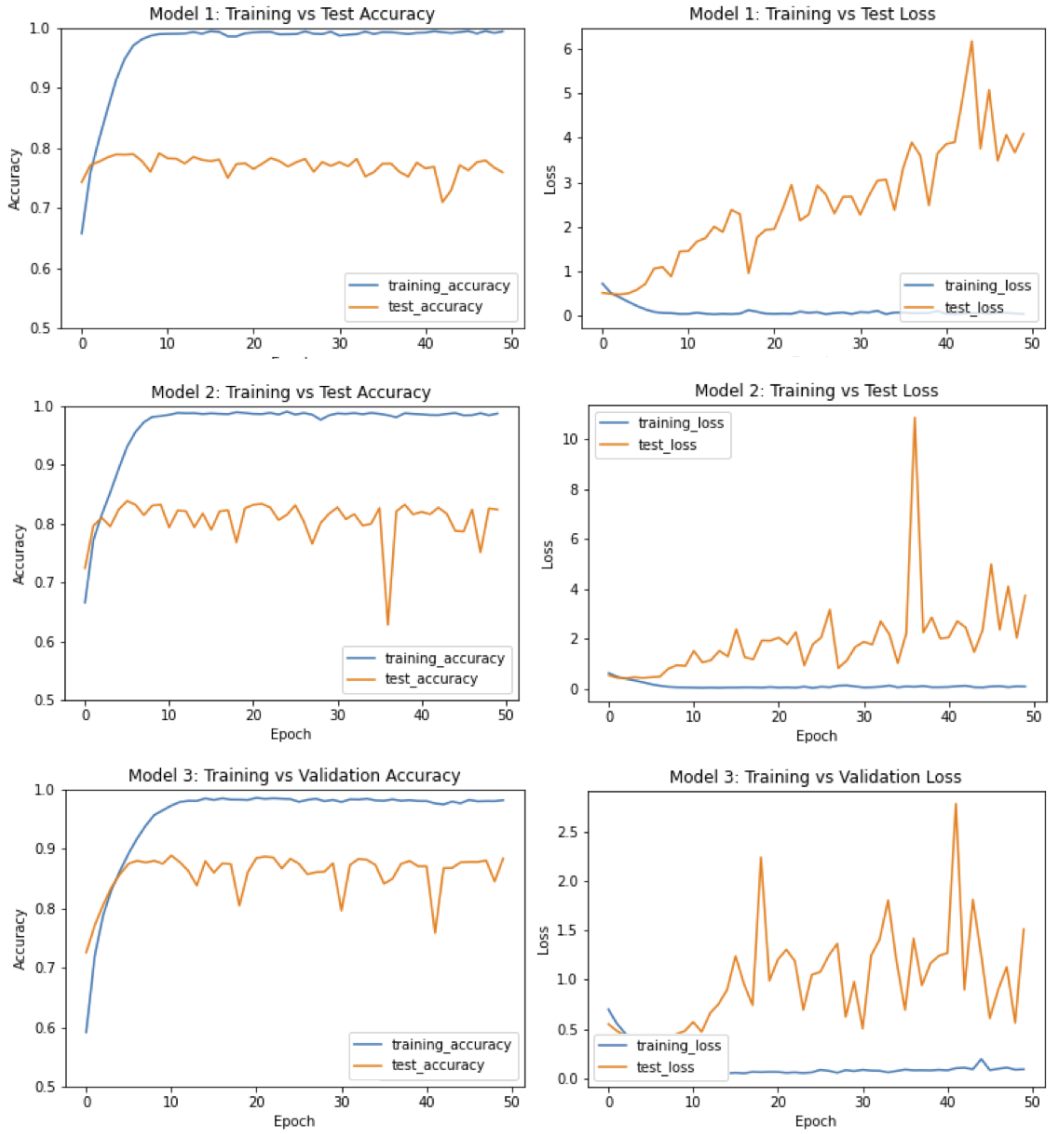

Figure 6: Models 1-3 Accuracy and Loss over the Epochs

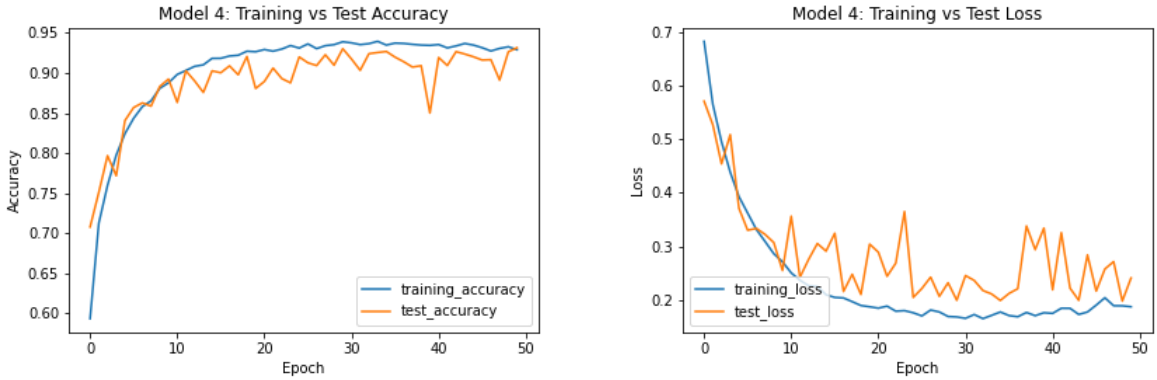

Figure 7: Model 4 Accuracy and Loss over the Epochs

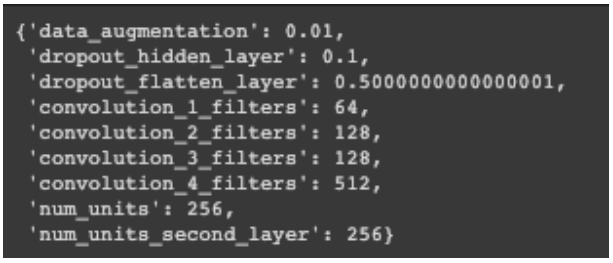

Figure 8: Best hyperparameters

4.3.1 Model Architecture

Due to computational limitations, not all the hyperparameters of a convet can be tuned and not all their possible values can be explored. Therefore, I tried to choose hyperparameters and ranges of values which made sense and which were somehow interesting to explore. In particular, I was interested in tuning the amount of regularization (data augmentation and dropout rates) and the number of filters computed by each Convolution Layer. The choice resulted in the following grid:

1. **Rate of Data Augmentation**. Values: [0.0, 0.1] with 0.01 step;
2. **Dropout rate for the hidden layers**. Values: [0.05, 0.1] with 0.01 step;
3. **Dropout rate for the fully connected layers**. Values: [0.2, 0.5] with 0.1 step;
4. **Number of filter of the first Convolution layer**. Values: 16, 32, 48, 64;
5. **Number of filter of the second Convolution layer**. Values: 64, 96, 128;
6. **Number of filter of the third Convolution layer**. Values: 128, 192, 256;
7. **Number of filter of the fourth Convolution layer**. Values: 256, 384, 512;
8. **Number of units of the first Fully Connected layer**. Values: [64, 256] with step 64;
9. **Number of units of the second Fully Connected layer**[6]. Values: [64, 256] with step 64.

The Cartesian product of these values results in the search space of the tuner. The results of the tuning procedure are displayed in Figure 8.

The model with the best hyperparameters has then been fitted for 50 epochs, to decide which epoch resulted in highest validation accuracy[7]. Epoch 21 resulted to be the best.

The model was therefore fit once again on the whole Training + Validation Set for 25 epochs, to account for the fact that the training occurs now on slightly more data.

4.3.2 Model compilation and Fitting

The model has been compiled and fit with the same parameters of Models 1-4.

4.3.3 Model Evaluation

The results of Model 5 evaluation on test set are displayed in the following table and Figure 9:

Model	Test Accuracy	Test Loss
Model 5	0.914	0.219

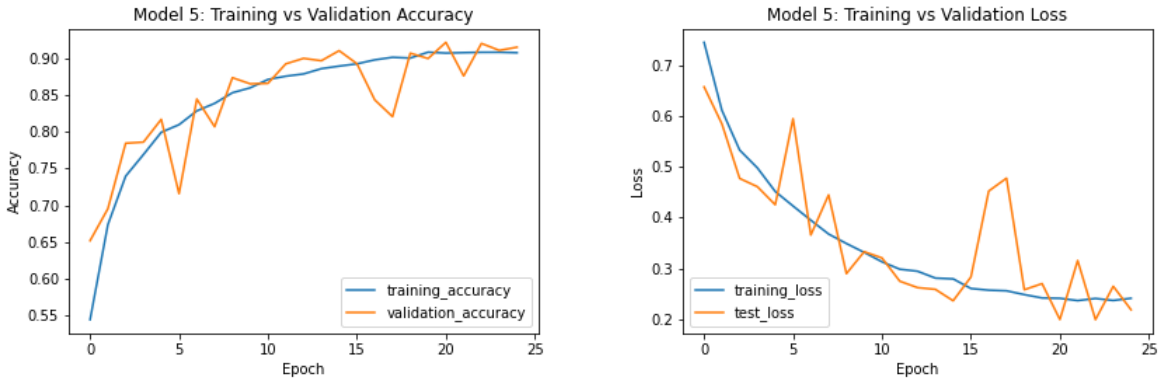

Figure 9: Model 5 Accuracy and Loss over the Epochs

As we can see from the Figure 9, this model does quite a good job at reducing overfitting. However, quite surprisingly, this model does not outperform the previous model. While loss is slightly lower with respect to Model 4, accuracy is is not as high as in Model 4. This might be due to the fact that the computational time required by the Hyperparameter Tuning procedure has been necessarily limited.

4.4 5-Fold Cross Validation Risk Estimates

Since Model 4 perfomed the best -with respect to accuracy- on this task, I computed 5-fold cross validation risk estimates of its performances. Since in cross validation we repeat the procedure of fitting and evaluating the network for each iteration, estimates will in this case be more accurate.

Model	Test Accuracy (CV estimate)	Test Loss (CV estimate)
Model 4	0.917	0.228

Finally, Figure 10 displays the trend of training and validation accuracy and loss over the epochs.

5 Conclusion

The experimental results of this exercise reveal three main conclusions, which could be however backed up by further data and experiments:

1. Deepening the depth of the convet results in networks that learn better, i.e. have higher performances and generalise better. However, if not regularized, they are easily exposed to high overfitting;
2. Regularization techniques such as Data Augmentation and Dropout are effective in reducing overfitting;
3. Techniques such as hyperparameters tuning can help raising the testing performances of the model, however - in this exercise - they did not impact significantly on the performance with respect to model for which hyperparameters values were set in advance. To further clarify this issue, new experiments with longer search processes and wider search spaces should be perfomed.

[6]This additional Fully Connected Layer was introduced only in this last mode.

[7]While in the previous model a validation set was not needed (no hyperparameters tuning was performed and the number of epochs was chosen in advance), in this case data have been split in Training, Validation and Test Set. Validation Set constituted 20% of the whole training set.

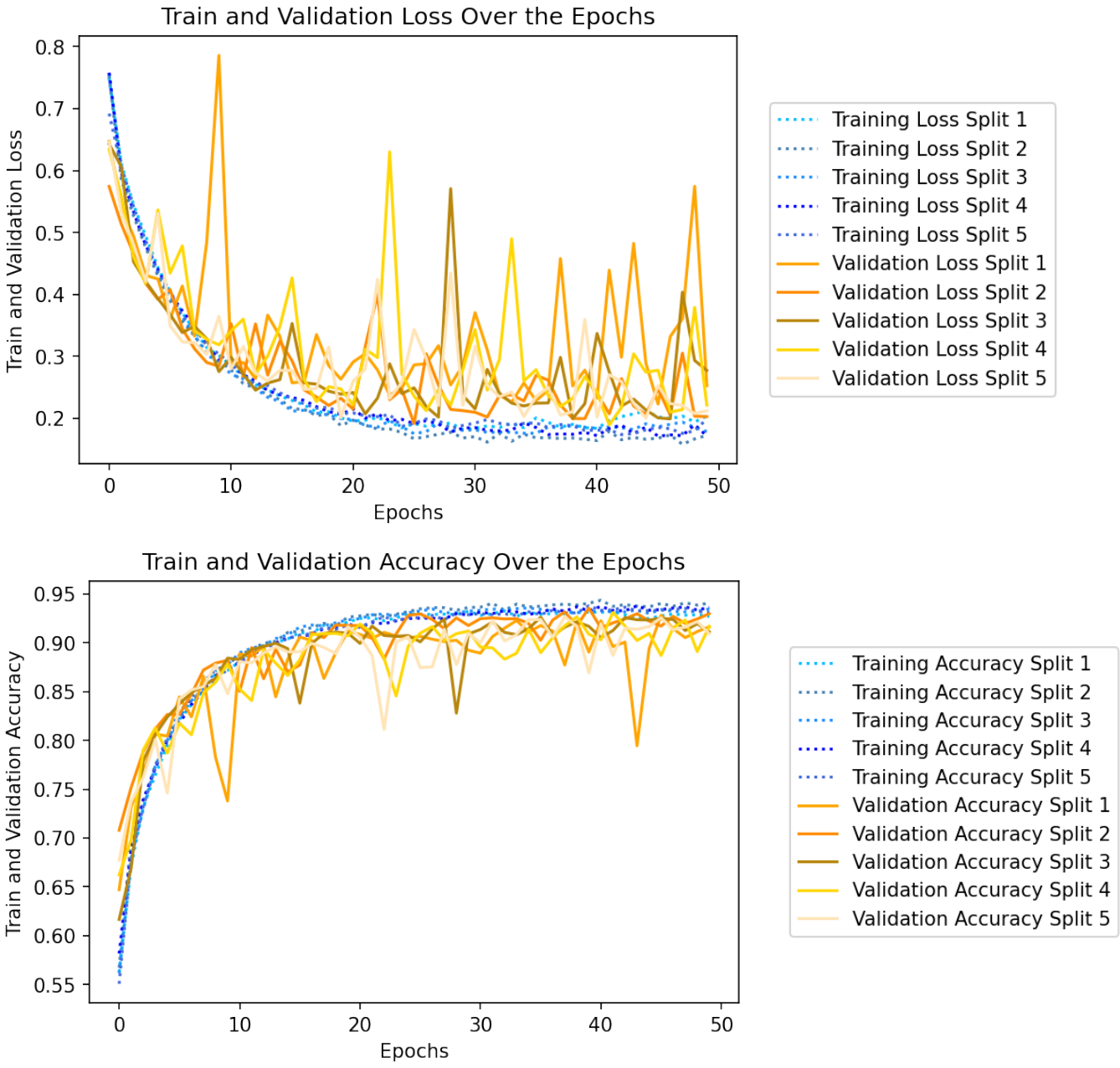

Figure 10: Model 4 Cross Validation Accuracy and Loss Trend Over the Epochs