Neural Networks Project Report

Course of Statistical Methods for Machine Learning

Giulia Scarpa 26780A

July 2023

Abstract

Neural Networks are a very spread method in the field of the Artificial Intelligence. In this research we want to deal about Neural Networks applied to a binary classification problem. The purpose is to do a correct classification of the images of muffins and chihuahuas, so I will test different kinds of Neural Networks models to find the best way to solve this research problem. The dataset is taken from Kaggle[1], a website that provides lots of datasets for data science researches, and the project is developed in the Google colab environment, a free platform made available from Google. The main library used to compute the project is Keras taken from Tensorflow's packages, a common library used from data science to test different neural network models.

1 Data Pre-processing

First of all I download the dataset from the kaggle website to obtain the data to do the analysis described below. The dataset is divided in training and test set. The training set is composed by 4733 images, where 3787 I will use for the training set and 946 for the validation set, while the test set is composed by 1184 images.

1. The first point is to pre-process images from the training and test datasets. I resize the image color mode, the size of the image and of the batch. In this step I decided to use a batch size of 32, a image size of 128x128, the color mode 'rgb' and I categorize the label of images with a binary mode label. I divide the training set in 2 different subset: training (80 percent) and validation set (20 percent). At the end of this pre-process I obtain: 119 training batches, 30 validation batches and 37 test batches.

2. In the second step I'm going to use two important methods (cache and prefetch) for loading the data that allow the dataset to be executed correctly during the training of the models and not to overload the memory.

[1] https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification

3. The last step is to normalize the dataset from range of the rgb channel [0, 255] to the range [0, 1] to obtain better performance of the models.

2 Network Architectures

As I'll show below, I test different kinds of network architectures to show how the results might change preferring a network than another one.
Every model was compiled using:

- **Optimizer:** *Adam Optimizer*, a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. Some of Adam's advantages are that the magnitudes of parameter updates are invariant to rescaling of the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require a stationary objective, it works with sparse gradients, and it naturally performs a form of step size annealing [1].
- **Loss:** *Binary Cross Entropy Loss*, the loss for binary classification with binary label (0 or 1).
- **Metrics:** *Accuracy*, the metric to compute the performance of the model.

was fitted using:

- **Training set:** 80 percent of the training set to fit the model.
- **Epochs:** 10, the number of epochs to run the model to learn from the training set.
- **Validation set:** 20 percent of the training set to validate the model during the training phase.

and evaluated using:

- **Testing set:** used to evaluate the model after the fitting phase.

2.1 Multi-Layer Perceptron (MLP)

A Multi-Layer Perceptron (MLP) is a type of artificial neural network that is used in machine learning for classification and regression tasks. The multilayer perceptron is known as a feed-forward neural network. The architecture of a multilayer perceptron consists of an input layer, one or more hidden layers, and an output layer. The input layer receives input data and passes them to the next layers through the network. At the end the output layer produces the final output that corresponds to a a vector of probabilities for classification tasks. Multilayer perceptrons are described as being fully connected, with each node connected to every node in the next and previous layer. Multilayer perceptrons have the ability to learn through training, they learn in a supervised manner.

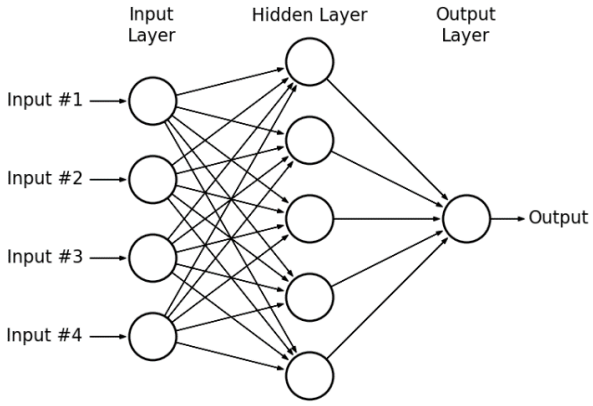

Figure 1: Multi-Layer Perceptron architecture

Training requires a set of training data, which consists of a series of input and associated output vectors. During training the multilayer perceptron is repeatedly presented with the training data and the weights in the network are adjusted to minimize a given loss function until the desired input - output mapping occurs. The loss function measures the difference between the predicted output of the network and the true output for a given input. This difference is used to compute the gradient of the loss with respect to the weights to determine to what degree the weights in the network should be adjusted so that the overall error of the multilayer perceptron is reduced. This process is called backpropagation. Once trained with representative training data the multilayer perceptron can generalise to new input data [2] [3].

2.1.1 Experiment MLP models

First of all I define the model architecture. The input data pass to a flatten layer and they are flattened to a single dimension. I use a image dimension of 128 x 128 x 3. Thus, when flattened, they corresponds to 49152 entries representing all the pixels of every image. Then I use for the first model 2 dense layers, for the next ones I increment of one dense layer every time for every model and I use the rectified linear unit (ReLU) activation function. In the last layer I use a dense layer with a sigmoid function as the activation function, that it is usually used for the binary classification tasks. The results obtained from the evaluation of the models are the following:

MLP model	Loss	Accuracy
First mlp:	0.674851	0.683277
Second mlp:	0.578183	0.721284
Third mlp:	0.520004	0.769426
Fourth mlp:	0.505556	0.766892

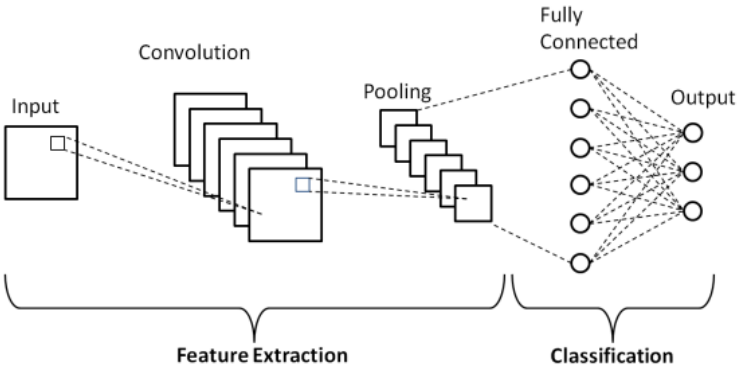

Figure 2: Convolutional Neural Network architecture

As we have seen in this case, the more layers we have, the better the performance will be.

2.2 Convolutional Neural Network (CNN)

Convolutional neural network layer types mainly include three types, namely Convolutional layer, pooling layer and fully-connected layer. Convolutional layer is the core part of the Convolutional neural network, the aim is to learn feature representations of the inputs. Convolutional layer is consists of several feature maps. Each neuron of the same feature map is used to extract local characteristics of different positions in the former layer, but for single neurons, its extraction is local characteristics of same positions in former different feature map. In order to obtain a new feature, the input feature maps are first convolved with a learned kernel and then the results are passed into a nonlinear activation function. We will get different feature maps by applying different kernels. The typical activation function are sigmoid, tanh and Relu. Then we have the pooling layer, where pooling means down sampling of an image. It takes small region of the convolutional output as input and sub-samples it to produce a single output. It is usually placed between two Convolutional layers. The size of feature maps in pooling layer is determined according to the moving step of kernels. The typical pooling operations are average pooling and max pooling. Max pooling takes largest of the pixel values of a region. Pooling reduces the number of parameter to be computed but makes the network invariant to translations in shape, size and scale. We can extract the high level characteristics of inputs by stacking several Convolutional layer and pooling layer. The last part of a CNN is the classifier, which corresponds to one or more fully-connected layers. They take all neurons in the previous layer and connect them to every single neuron of current layer. There is no spatial information preserved in fully-connected layers. The last fully-connected layer is followed by an output layer. For classification tasks, softmax or sigmoid regression are commonly used

because of it generating a well-performed probability distribution of the outputs [4][5].

2.2.1 Experiment CNN models

First of all I define the model architecture. The input data pass to a convolutional layer, which act as filters for images, and I use an image dimension of 128 x 128 x 3. Then I use the MaxPooling2D layer, that takes large images and shrink them down while preserving the most important information in them. I add a Dropout layer. This layer is usually used to improve the generalization of the network. In the last step I use a fully connected layer that corresponds to a simple perceptron with a flatten layer as an input, a dense layer with a ReLU activation function and the output layer with a sigmoid activation function. This described above is the first CNN architecture, for the next ones I add new convolutional and pooling layers to observe if the computation can have better accuracy adding other layers. The results obtained from the evaluation of the models are the following:

CNN model	Loss	Accuracy
First cnn:	0.535288	0.855574
Second cnn:	0.503913	0.887669
Third cnn:	0.455143	0.889358

As we have seen in this case, the more layers we have, the better the performance will be. We can also observe the difference of accuracy of the CNN models respect to MLP models. We can say that CNN has a better accuracy in the computation.

2.3 Residual Neural Network (ResNet)

ResNet50 is another convolutional neural network-based model, consisting of 48 convolution layers. The other two layers are 1 Max Pooling layer and 1 Average Pooling layer. ResNet architecture allowed CNN to operate with multiple layers. This set of layers is called residual building block (RBB). RBB consists of several convolutional layers (Conv), batch normalizations (BN), Relu activation function and one shortcut. RBB is based on the idea of skipping blocks of convolutional layers by using shortcut connections. Deep neural networks with multiple stacked layers tended to produce greater training error percentages than models with lesser layers. This residual network framework allowed the addition of shortcut connections and usage of residual functions, thus allowing stacked layers of deep neural networks to reduce their training errors. These shortcuts are useful for optimizing trainable parameters in error backpropagation to avoid the vanishing/exploding gradients problem, which can help to construct deeper CNN structure to improve final performance [6][7].

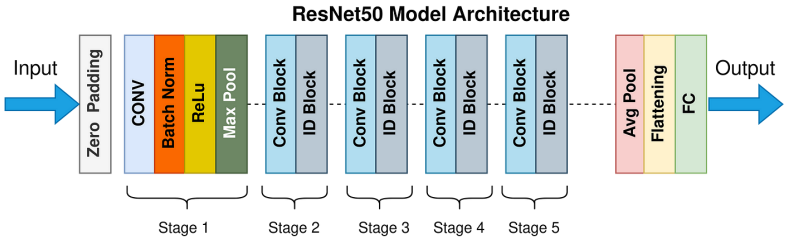

Figure 3: ResNet50 architecture

2.3.1 Experiment ResNet50 model

Initialize the structure of ResNet-50 and restore the pre-trained weights into ResNet-50, then I add new layers for the computation: the GlobalAveragePooling2D layer, 128 the number of hidden layers and the sigmoid classifier layer. Fitting the ResNet50 model I obtained the following results:

ResNet50	
Loss:	0.466725
Accuracy:	0.771115

We can observe that we obtain a worst accuracy than the CNN model computing previous.

3 Hyperparameters Tuning

Hyperparameter tuning is the process of selecting the right set of hyperparameters for your learning algorithms. These are special parameters (ex. the learning rate in neural networks) whose value must be determined before the training phase can start [8]. Hyperparameters are the variables that govern the training process and topology of a machine learning model. These variables remain constant throughout the training process and directly affect the performance of the machine learning program. We can find two types of hyperparameters:

- **Model hyper-parameters:** that affect model selection such as the number and width of hidden layers.
- **Algorithm hyperparameters:** affecting the speed and quality of the learning algorithm such as learning rate for stochastic gradient descent [9].

I decided to compute and apply hyperparameter tuning to the best MLP model and the best CNN model obtained. The computation of hyperparameter tuning for the MLP gives us as the following results:

Hyperparameter	Values
Units:	512
Learning rate:	0.001
Epochs:	10

While the computation of hyperparameter for the CNN gives the same results except for the number of epochs that results to be 7. The performance, applying the hyperparameters to the models, are summary below:

MLP model	Loss	Accuracy
Results:	0.505556	0.766892
with hyperparameters:	0.484637	0.782939
CNN model	Loss	Accuracy
Results:	0.455143	0.889358
with hyperparameters:	0.364465	0.875000

4 K-fold Cross Validation

K-fold cross validation is the most popular technique of cross validation techniques. It divides the whole data into K subsamples (known as folds) with approximately equal cardinality m/K samples. Each subsample successively plays the role of validating dataset (test set), while the rest K-1 subsamples are used as the training set for training the classifier [7]. We call S_i the testing part of the i-th fold while S_{-i} is the training part. The K-fold CV estimate of $E[l_D(A)]$ on S, denoted by $l_S^{CV}(A)$, is then computed as follows: we run A on each training part of the folds $i = 1, ..., K$ and obtain the predictor $h_1 = A(S_{-i}), ..., h_K = A(S_{-K})$. We then compute the (rescaled) errors on the testing part of each fold,

$$l_{S_i}(h_i) = \frac{K}{m} \sum_{x,y \in S_i} l(y, h_i(x))$$

Finally, we compute the CV estimate by averaging these errors [8],

$$l_S^{CV}(A) = \frac{1}{K} \sum_{i=1}^{K} l_{S_i}(h_i)$$

4.1 Experiment 5-fold Cross Validation

To experiment the Cross Validation I use the entire dataset as the training set and the model chosen for the computation is the best CNN model obtained, because as we saw above is the model that produce better performance. The model for Cross Validation was compiled using:

- **Optimizer:** *Adam Optimizer.*

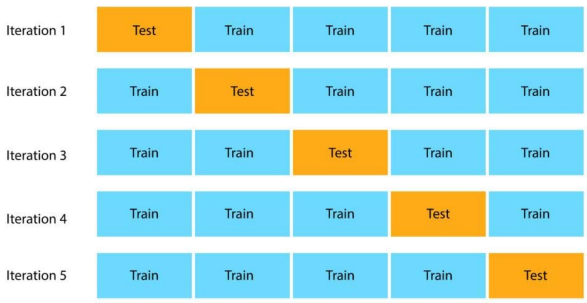

Figure 4: 5-fold Cross Validation

- **Loss:** *Binary Cross Entropy*, the loss for binary classification with binary label (0 or 1).
- **Metrics:** *Accuracy*, the metric to compute the performance of the model.

and fitted using:

- **Training set:** is the entire dataset that during the computation of the algorithm is splitted into 80 percent training set and 20 percent validation set.
- **Epochs:** 5, the number of epochs to run the model to learn from the training set. I decided to use less step of epochs because for the long time of the computation of this algorithm.

As we can see in the figure 5 the steps of the cross validation have the same trend, it means that we have a good approximation of the model, infact doing the mean of the evaluation of every step of the cross validation we obtain an accuracy of 89% with a low error loss (28.6%).

Training Hyperparameters Tuning

Training the model with hyperparameter tuning didn't produce considerable changing in the value of the accuracy or the loss. We can see in the table below the results of cross validation with and without the use of the hyperparameter. For the cross validation I decided to use the best CNN model, so I reuse the hyperparameters computing before for this CNN model.

Cross Validation	Loss	Accuracy
Results:	0.286075	0.891164
with hyperparameters:	0.298993	0.890302

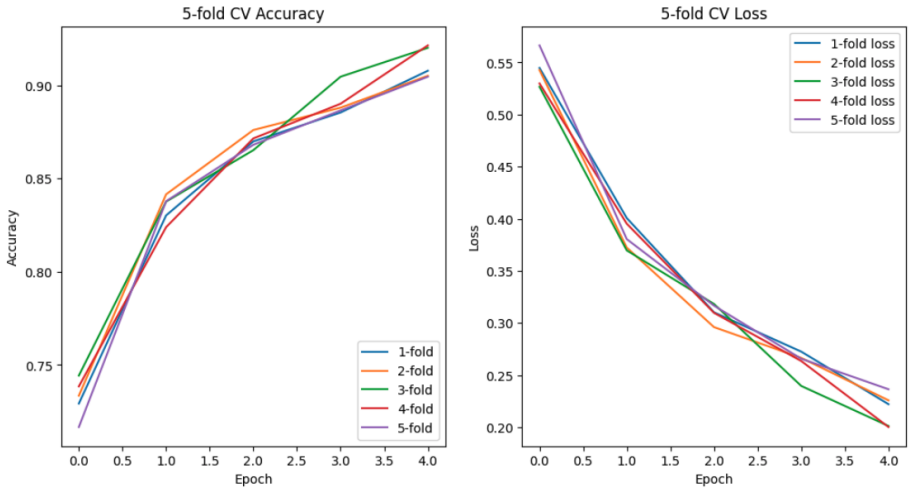

Figure 5: 5-fold Cross Validation Experiments

5 Discussion

To summary what we have said until now, we can conclude saying that:

- We can obtain better results with a low loss and an high accuracy when we use the convolutional neural network than a simple multi-layer perceptron, because CNN is the best architecture to use for an image classification task.
- We expected to obtain better performance with the ResNet architecture, but on the contrary as we saw above, we obtain better result with convolutional neural network that have less levels. Therefore we can say that It's not strictly necessary to use a deep convolutional neural network with many levels to obtain better approximations, this depends on the dataset and the aim of the task that we have.
- Computing hyperparameter tuning applied to the network architectures can produce better performance of the network.
- Cross Validation is a good algorithm to produce a right estimate of the performance of the model, in this case we use the best model obtained in the previous computations. On the contrary that we say above, in this case applied hyperparameter tuning to cross validation doesn't produce an increment of the performance, but the values obtained are very similar to the values obtained with the cross-validation computation without hyperparameter tuning.

References

[1] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[2] Walter H Delashmit, Michael T Manry, et al. "Recent developments in multilayer perceptron neural networks". In: *Proceedings of the seventh annual memphis area engineering and science conference, MAESC*. 2005, pp. 1–15.

[3] Matt W Gardner and SR Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[4] Tianmei Guo et al. "Simple convolutional neural network on image classification". In: *2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA)*. IEEE. 2017, pp. 721–724.

[5] Farhana Sultana, Abu Sufian, and Paramartha Dutta. "Advancements in image classification using convolutional neural network". In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*. IEEE. 2018, pp. 122–129.

[6] Sheldon Mascarenhas and Mukul Agarwal. "A comparison between VGG16, VGG19 and ResNet50 architecture frameworks for Image Classification". In: *2021 International conference on disruptive technologies for multi-disciplinary research and applications (CENTCON)*. Vol. 1. IEEE. 2021, pp. 96–99.

[7] Long Wen, Xinyu Li, and Liang Gao. "A transfer convolutional neural network for fault diagnosis based on ResNet-50". In: *Neural Computing and Applications* 32 (2020), pp. 6111–6124.

[8] Nicolò Cesa-Bianchi. "Hyperparameter tuning and risk estimates". In: *Machine Learning - Statistical Methods for Machine Learning*. 2023.

[9] Tensorflow. "Hyperparameter optimization with Keras Tuner". In: *https : //www.tensorflow.org/tutorials/keras/keras_tuner?hl = it*.

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

A Multi-Layer Perceptron Appendix

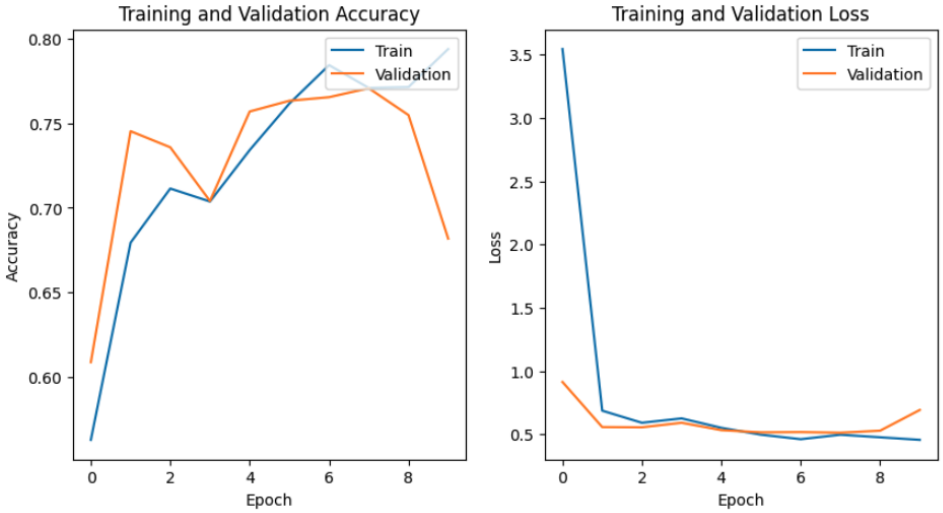

Figure 6: First MLP model

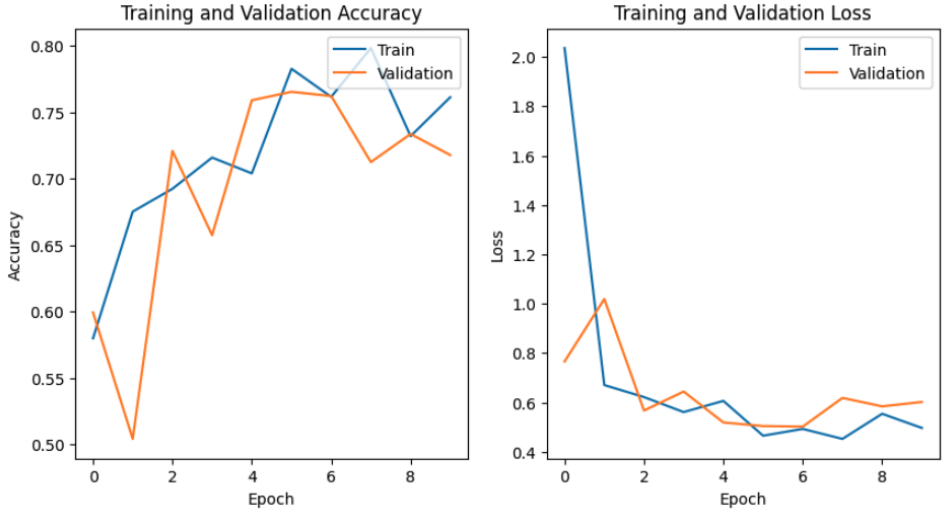

Figure 7: Second MLP model

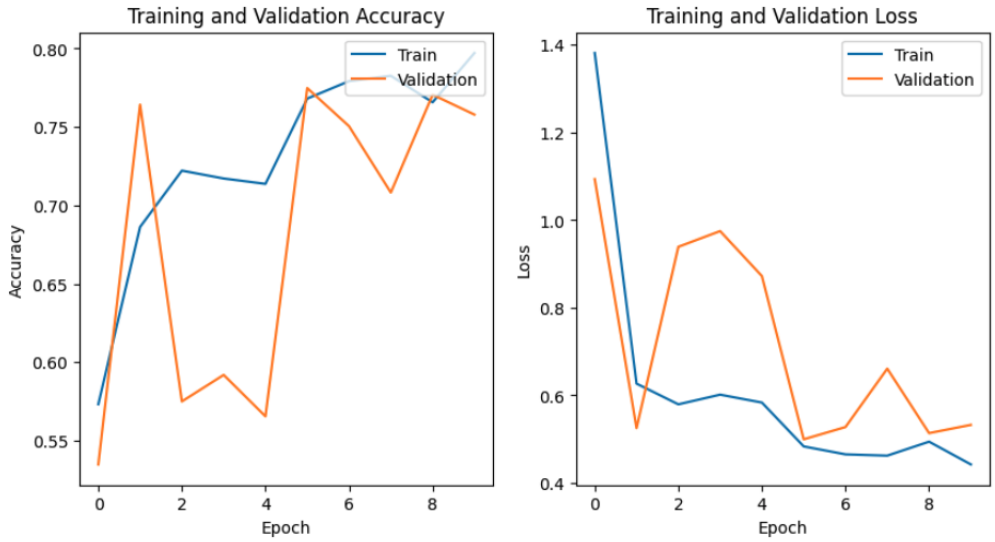

Figure 8: Third MLP model

Figure 9: Fourth MLP model

B Convolutional Neural Network Appendix

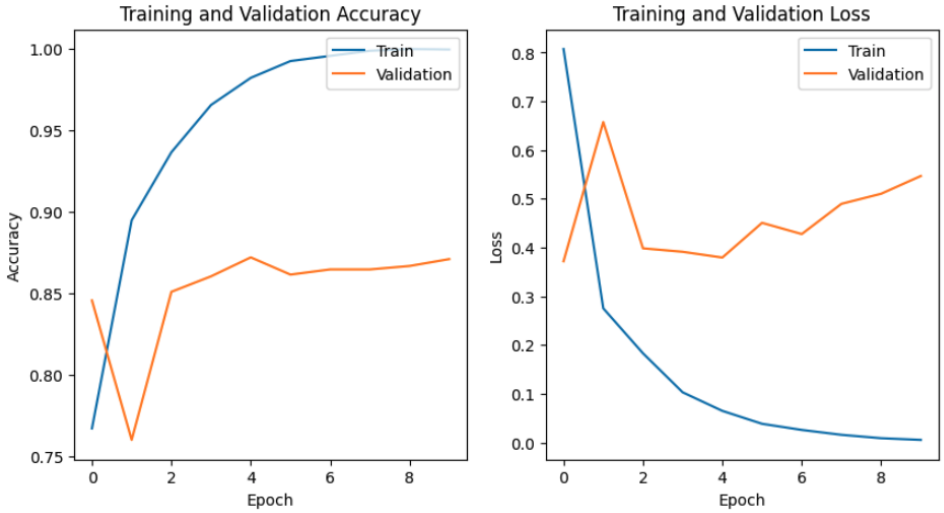

Figure 10: First CNN model

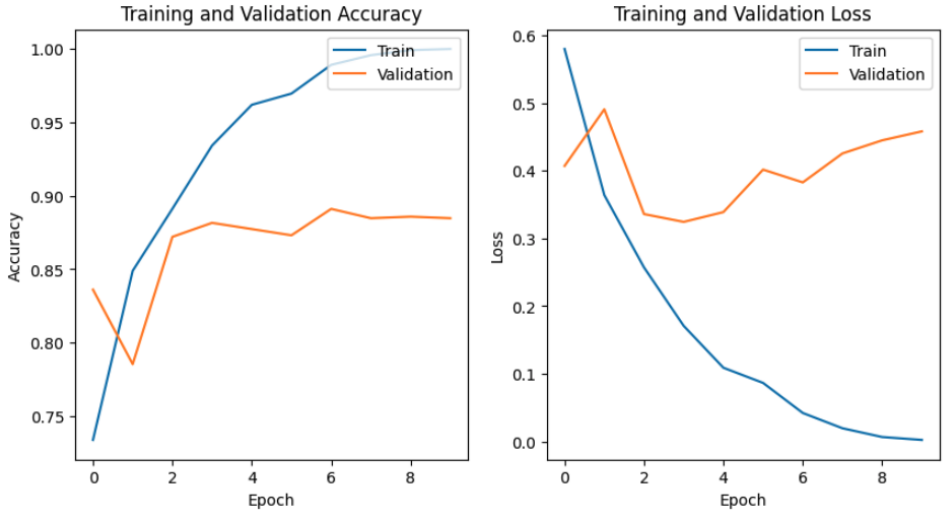

Figure 11: Second CNN model

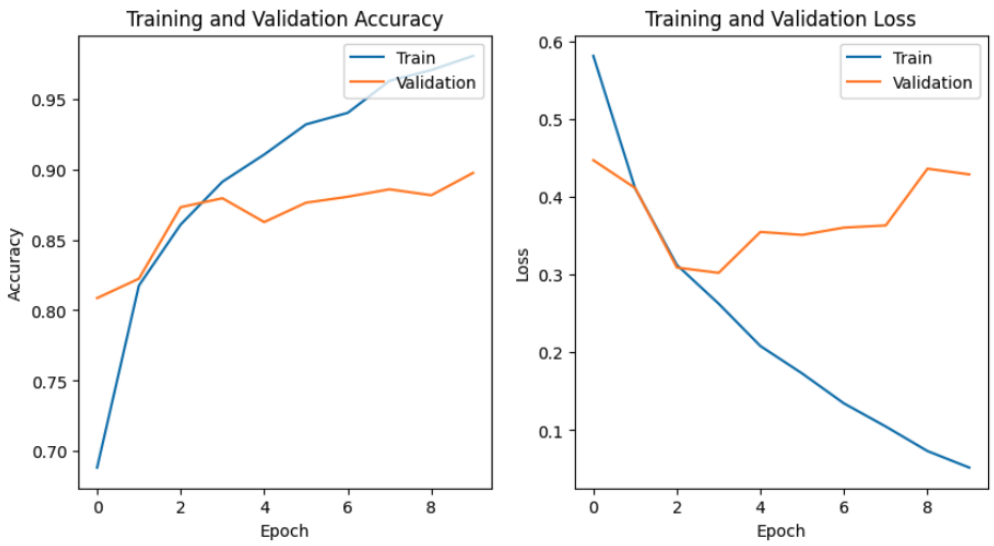

Figure 12: Third CNN model

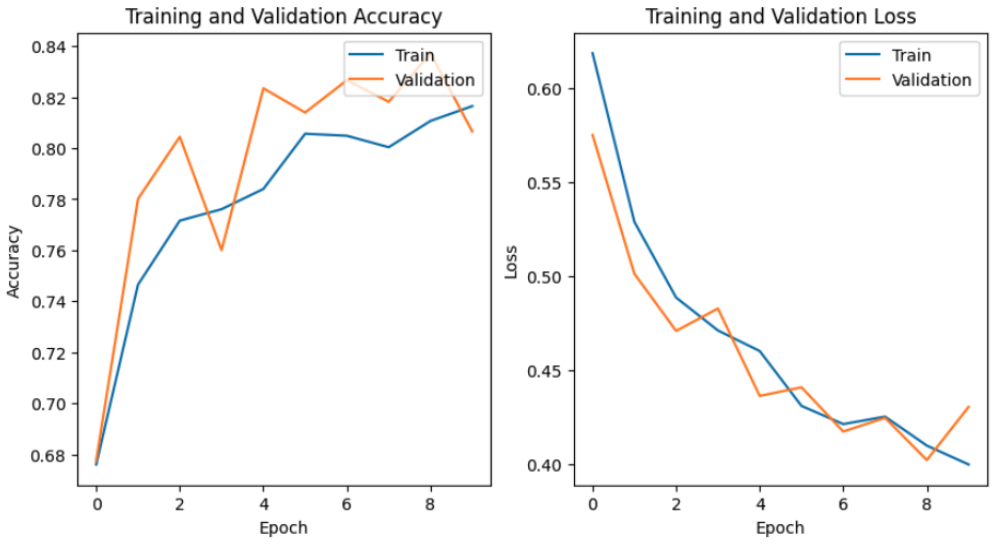

Figure 13: ResNet50 model