ROSKILDE UNIVERSITY

ARTIFICIAL INTELLIGENCE, DEEP LEARNING – E2022

Applying Convolutional Neural Network for classification of chest x-ray images to identify pneumonia in patients

Author: Jakub Schovanec

Student Number: 74357

Hand in: 21.11.2022

Number of characters: 12 832

Examiners: Henning Christiansen, Masoumeh Vahedi

Table of Contents

1 Introduction

The purpose of this report is to document a mini project where the goal is to experiment with training a Convolutional Neural Network that can be given an x-ray image of lungs and will determine whether the lungs are either infected with pneumonia or not. The project was built in Jupyter Notebooks and used TensorFlow version 2.3.1.

Convolutional Neural Networks or Convnets is a neural-like network consisting of input, hidden and output layers. There is only one input and output layer but there can be many hidden layers in between them. Each layer contains nodes and arranges them in three dimensions. The nodes are all fully connected to the nodes in previous layer, where each connection is assigned a random weight at the start of the training process and gets adjusted by feed forward and back propagation algorithms. (Stanford, 2022)

There are three types of layers that are used when building Convnets:

1. Convolutional

   Convolution layer is responsible for computing the output of the nodes by computing a dot product between weights and inputs. (Stanford, 2022) Convolutional layers apply the Rectified Linear Activation function (ReLU) that gives the output of the nodes in a way that it returns the result if positive or returns 0 if the result is negative. (Kalita, 2022)

2. Pooling

   Pooling layer will downsample the output of the convolutional layer nodes. (Stanford, 2022) In my project I have made use of MaxPooling. MaxPooling takes the most significant element from the feature map. Another pooling method is AveragePooling that computes the average of elements from the feature map. (Kalita, 2022)
3. Fully-Connected

   Fully-Connected layer is responsible for computing the final scores.

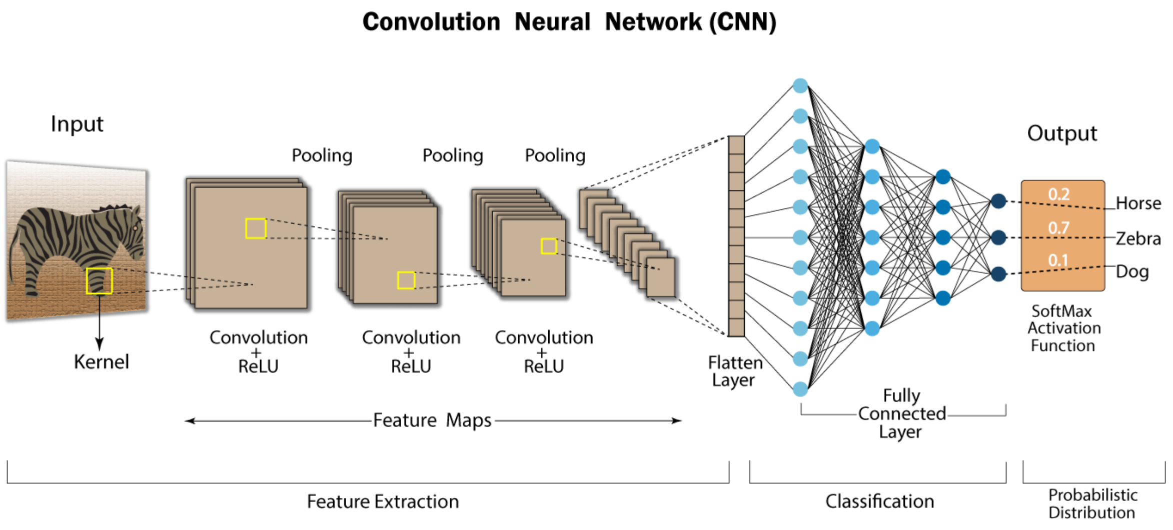
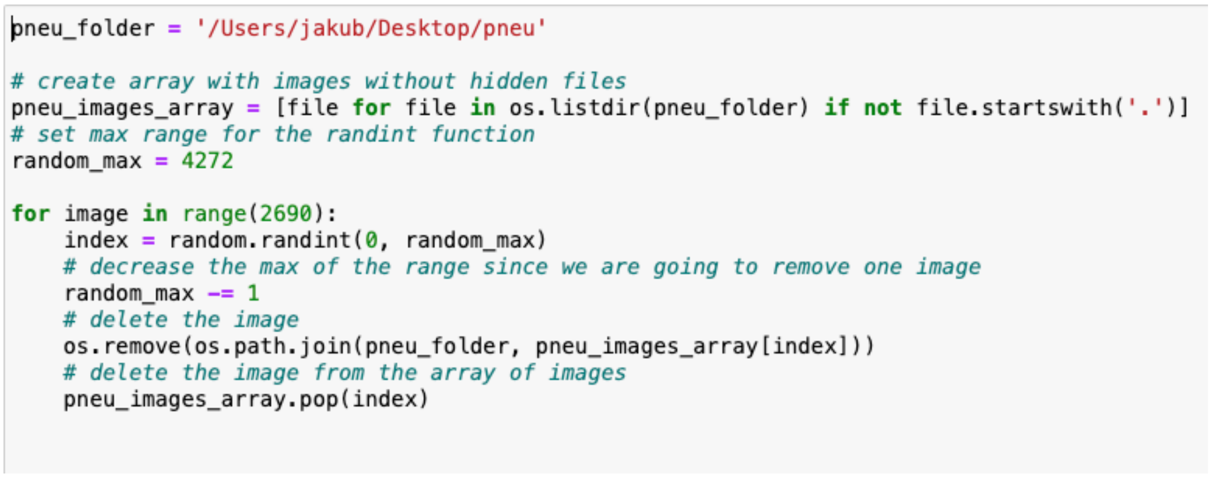


Figure 1-1: Visual representation of different layers in CNN. (Kalita, 2022)

As shown on the figure above, the Convolutional layers with ReLU and Pooling are part of the feature extractions. The feature map is afterwards flattened to a vector and from there Fully-Connected layer is applied in order to classify the image.

2 Data Set

The data set for the project was created by Kermany, Zhang and Goldbaum (2018). The data set consists of 5 856 x-ray images of lungs. More specifically, it contains 1 583 images of healthy/normal lungs and 4 273 images of lungs infected with pneumonia. These images were split into two folders – Train and Test. Each folder contained additional two folders named NORMAL and PNEUMONIA. From this I could observe that the data needs to be restructured because of two reasons. First was that the data set was missing a validation folder and the second reason was that the data in Train folder was heavily imbalanced. Therefore, I had to decide to either over- or under-sample the dataset. Oversampling the dataset of x-ray images would not make sense because the data provided to the application will be in a consistent form and oversampling x-ray images would just cause overfitting by having too many similar images. That is the reason why I decided to go with undersampling to balance the binary classification even though I took a risk of getting rid of essential data that could be used to classify the differences between the images. The under-sampling process was random, so I put all the pneumonia images into one folder and deleted 2 690 random images from the folder to keep only 1583 images which is the same amount as NORMAL images. The process was done by following code that generated random numbers and continuously deleted a file from the folder with all PNEUMONIA images.



```
pneu_folder = '/Users/jakub/Desktop/pneu'

# create array with images without hidden files
pneu_images_array = [file for file in os.listdir(pneu_folder) if not file.startswith('.')]
# set max range for the randint function
random_max = 4272

for image in range(2690):
    index = random.randint(0, random_max)
    # decrease the max of the range since we are going to remove one image
    random_max -= 1
    # delete the image
    os.remove(os.path.join(pneu_folder, pneu_images_array[index]))
    # delete the image from the array of images
    pneu_images_array.pop(index)
```

Figure 2-1: Function to under-sample the PNEUMONIA images by randomly deleting images to balance the binary classification.

After that I redistributed the images into three folders – Train, Validation, Test in ratio approximately 63/19/18. This amounts to 1 000 images in Train folder, 300 images in validation folder and 283 images in Test folder.

3 Modelling

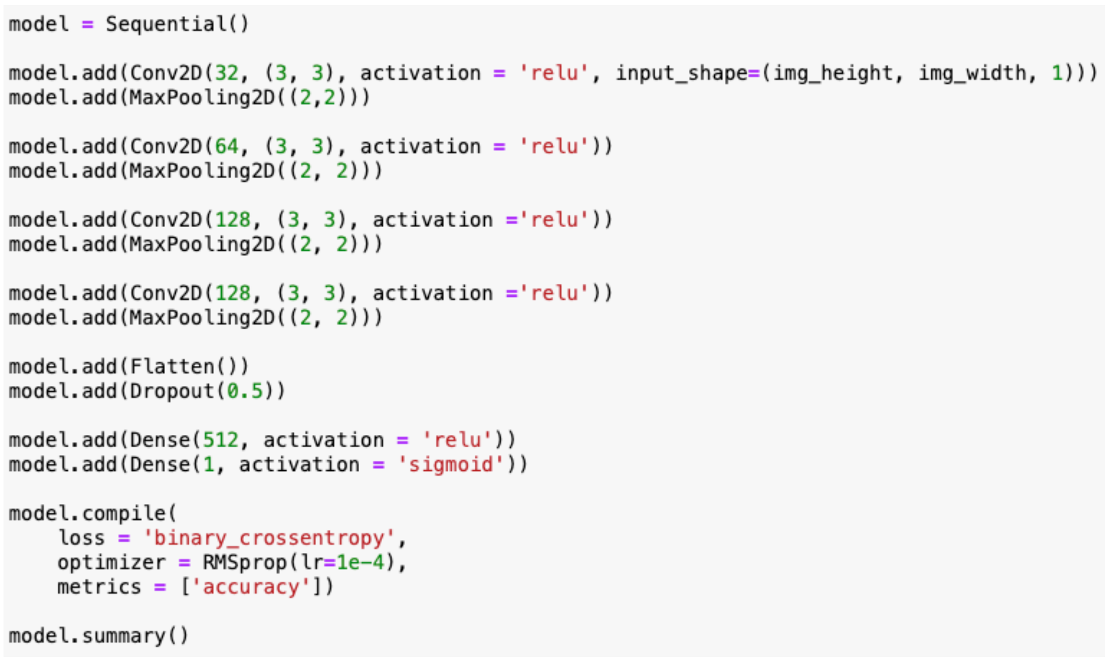
3.1 Image Pre-processing

In the Data Set chapter, I have explained the process of dividing the images into separate folders to create a balanced binary classification. After this, the images still needed to be processed because most importantly they were in too large format that was not necessary to classify the images properly. For the first run of the model, I have resized the images to 512x512 pixels, however, after applying the layers there was 59 223 681 trainable parameters found and each batch was expected to run for approx. 25 minutes. Because of this I have decided to drop the size of the input images to 128x128 pixels. While applying the layers to images of this size, only 2 600 577 trainable parameters were found. For this size of images, each batch took around 50 seconds to compute.

Apart from resizing the images, they had to be converted to the grayscale and rescale the pixel values from range between 0 – 255 to 0 – 1 since Convolutional Neural Networks are working better with tiny input values.

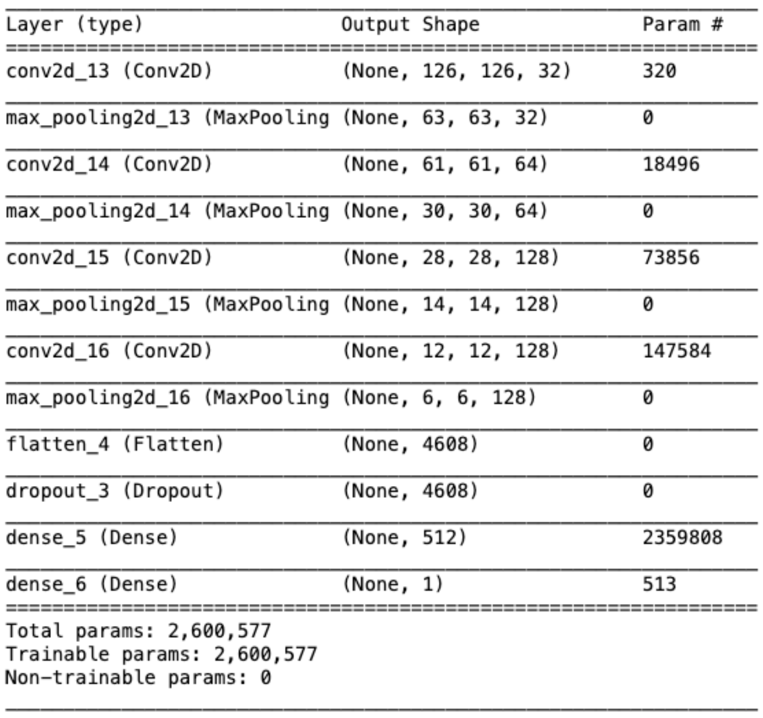
3.2 Building and training the model

The model was built after the data was pre-processed. The model consists of 4 layers and each of those layers consists of one Conv2D and one MaxPooling2D layer. In the first Conv2D layer specifies the size of the input shape which consists of the pixel size of the image and channels which equal to 1 since we use grayscale. After the layers are applied, the result is flattened, and Dropout layer is applied to randomly set unit to 0 in a rate of 0.5 which amounts to 50% to help prevent overfitting. After that, the vector goes through two Dense layers. Afterwards, the model is compiled with loss function set to "binary_crossentropy" since we are dealing with binary classification and as an optimizer RMSprop was chosen with a learning rate of 1e-4.



```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation = 'relu', input_shape=(img_height, img_width, 1)))
model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation ='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(128, (3, 3), activation ='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(512, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(
    loss = 'binary_crossentropy',
    optimizer = RMSprop(lr=1e-4),
    metrics = ['accuracy'])

model.summary()
```

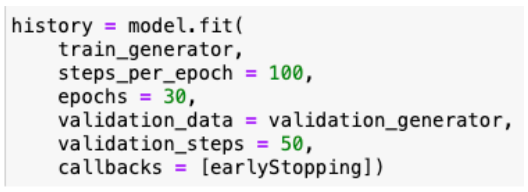
Figure 3-1: First model.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 126, 126, 32)      320
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 12, 12, 128)       147584
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 4608)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4608)              0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               2359808
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 513
=================================================================
Total params: 2,600,577
Trainable params: 2,600,577
Non-trainable params: 0
_________________________________________________________________
```

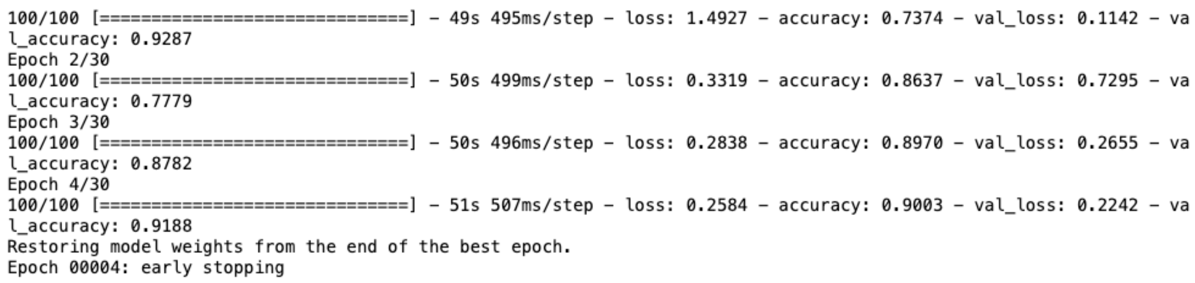
Figure 3-2: Summary of the first model.

After the model was built, the training on train and validation data can start. The steps of the epoch are set to 100, the number of epochs is set to 30. The EarlyStopping function was used to prevent the model from training further if the accuracy starts to decrease and restore the weights to the best run.



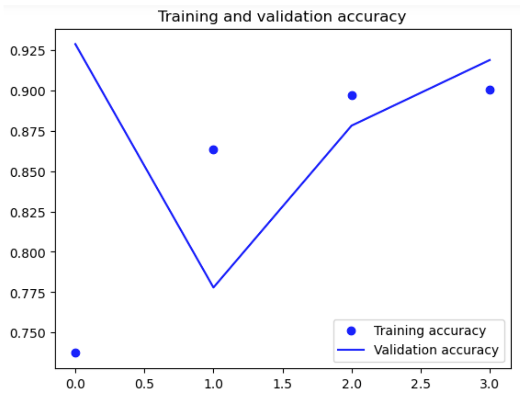
```
history = model.fit(
    train_generator,
    steps_per_epoch = 100,
    epochs = 30,
    validation_data = validation_generator,
    validation_steps = 50,
    callbacks = [earlyStopping])
```

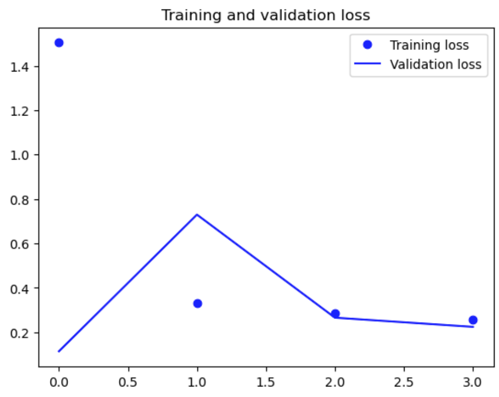
Figure 3-3: Function that trains the model with train and validation data.



```
100/100 [==============================] - 49s 495ms/step - loss: 1.4927 - accuracy: 0.7374 - val_loss: 0.1142 - va
l_accuracy: 0.9287
Epoch 2/30
100/100 [==============================] - 50s 499ms/step - loss: 0.3319 - accuracy: 0.8637 - val_loss: 0.7295 - va
l_accuracy: 0.7779
Epoch 3/30
100/100 [==============================] - 50s 496ms/step - loss: 0.2838 - accuracy: 0.8970 - val_loss: 0.2655 - va
l_accuracy: 0.8782
Epoch 4/30
100/100 [==============================] - 51s 507ms/step - loss: 0.2584 - accuracy: 0.9003 - val_loss: 0.2242 - va
l_accuracy: 0.9188
Restoring model weights from the end of the best epoch.
Epoch 00004: early stopping
```

Figure 3-4: First training run.





The first run of the model ran just for 4 epochs before early stoppage and gave us a test accuracy of 91,20%.

3.3 Model Adjustments

NR.	EXPLANATION	EPOCHS	TEST ACC.	MODEL
0	Original state	4	91,20%	pneu1.h5
1	Decreased the size of epochs to 20, to adjust to the training dataset size and decreased the number of validation steps to 30 to adjust to the size of the validation dataset.	9	93,48%	pneu2.h5
2	Resized images to 64x64 pixels.	12	94,05%	pneu3.h5
3	Swapped RMSprop optimizer for Adam with the same learning rate.	14	90,83%	pneu4.h5
4	Reverted to RMSprop, removed the Dropout layer after flattening and increased the filters applied in the last Convolutional layer to 256.	4	90,10%	pneu5.h5
5	Returned the Dropout layer with 50% frequency.	8	89,37%	pneu6.h5
6	Resized images back to 128x128 pixels.	5	92,06%	pneu7.h5
7	Added dropout layer with 25% frequency after each MaxPooling layer.	16	91,00%	pneu8.h5

All the results from the runs after adjustments in can be found in Appendix 2 in graphical form.

During the first run, the model started to overfit already in the second epoch. We can observe overfitting happening when either the validation accuracy is lower than training accuracy or when validation loss gets higher than training loss. However, the model bounced back at produced 91,20% test accuracy before early stoppage.

The first adjustment was made to make sure that I use the whole dataset. Running 30 epochs while steps per epoch are 100, meaning that for each epoch the model will train on 100 images, would require 3 000 images in the dataset. Since I had only 2 000 images in the training folder, I decided to reduce the number of epochs to 20. It is important to note however, that neither of my training runs reached the last epoch before early stoppage. Similarly, I had to consider the amount of steps for validation. I have reduced it from 50 to 30 to adjust for the number of images in the validation folder which is 600. This run improved the test accuracy by more than 2% to 93,48%, while it overfitted in sixth and seventh epoch.

For the second adjustment, I have decided to resize the images to even smaller scale 64x64 pixels. This reduced the number of trainable parameters significantly and reduced computation time of each epoch. This training run ran for 12 epochs and produced a 94,05% in accuracy which was a slight increase from the previous run. The run started overfitting in the last epoch.

For the third adjustment I have swapped the optimizer from RMSprop to Adam. The training run started to overfit in the fifth epoch and the test accuracy dropped to 90,83% which was lower than the first run. That is why I decided to switch back to RMSprop optimizer which uses adaptive learning rate that changes over the time. (Sanghvi, 2021)

As previously mentioned, in the fourth adjustment I reverted to RMSprop optimizer and removed the Dropout layer with 50% frequency that was applied after flattening the vector. Apart from those two changes, I have increased the filters applied in the last Convolutional layer to 256. This training run started overfitting in the fourth epoch where it also halted because of the value loss. The accuracy dropped even further to 90,10%.

In the fifth adjustment, I have returned the Dropout layer with 50% frequency. This run ran for 8 epochs and started overfitting in sixth epoch where the validation loss was higher than training loss. This run produced the lowest test accuracy, only 89,37%.

In the sixth adjustment, I have tried resizing the images back to 128x128 pixels and this gave a little bit of success by improving the test accuracy to 92,06%.

Finally, to try and reduce the overfitting, I have tried to add a Dropout layer with 25% frequency after each of the MaxPooling layers. The run ran for 16 epochs, the highest number of epochs before early stoppage, but started to overfit already in the third epoch where the validation loss got higher than training loss and it did not get lower again until the run halted. The test accuracy was at 91,00%.

There are two additional experiments that I would like to work on with the model but did not try those options for a lack of time.

The first experiment is to take all the images with pneumonia again and randomly delete them to under sample the dataset and see what results the best model (pneu3.h5) would produce. It is possible that by under fitting, I have removed some images that would help the classification and therefore improve the test accuracy, but it is also possible that by doing this, there would be no increase in accuracy.

The second experiment would be to split the images with pneumonia into two categories and that is viral and bacterial pneumonia. The PNEUMONIA images are labelled as either bacterial or viral therefore this would be possible.

4 Conclusion

A simple Convnet was built to classify x-ray images of normal lungs and lungs infected with pneumonia. It can be concluded that that the Convent was successful at the binary classification and the best result was achieved by scaling the image to only 64x64 pixels and applying 4 layers of combination of convolutional and maxpooling layers.

5 Bibliography

Chollet, François (2021), Deep Learning with Python, Second Edition. Manning

Kalita, D. (2022), Basics of CNN in Deep Learning. Analytics Vidhya. Accessed at: 20th November 2022. Available at: https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/

Kermany, D.; Zhang, K.; Goldbaum, M. (2018), Labeled Optical Coherence Tomography (OCT) and Chest X-Ray Images for Classification. Mendeley Data, V2, doi: 10.17632/rscbjbr9sj.2, at: https://data.mendeley.com/datasets/rscbjbr9sj/2

Sanghvi, R. (2021), A Complete Guide to Adam and RMSprop Optimizer. Medium. Accessed at: 20th November 2022. Available at: https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be

Stanford (2022), Convolutional Neural Networks. Stanford. Accessed at: 20th November 2022 Available at: https://cs231n.github.io/convolutional-networks/

6 Appendices

6.1 Appendix 1 – Google Drive folder with Notebook, saved models and dataset

https://drive.google.com/drive/folders/1--6qNbLwY5QW-MnnFIuAOMl6YO7qp36t?usp=sharing

6.2 Appendix 2 – Training and validation results

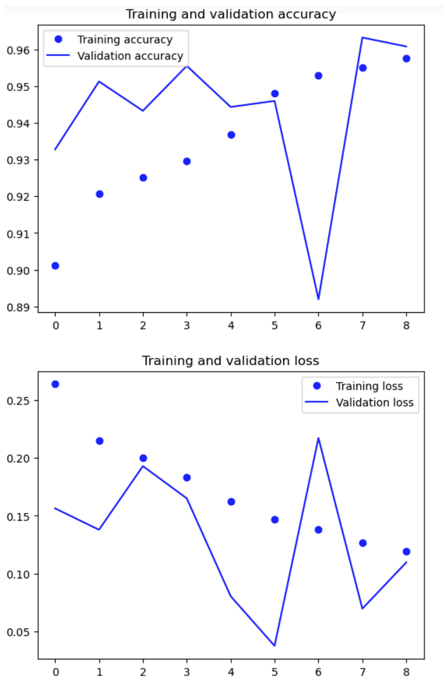


Figure 6-1: Results after the first adjustment

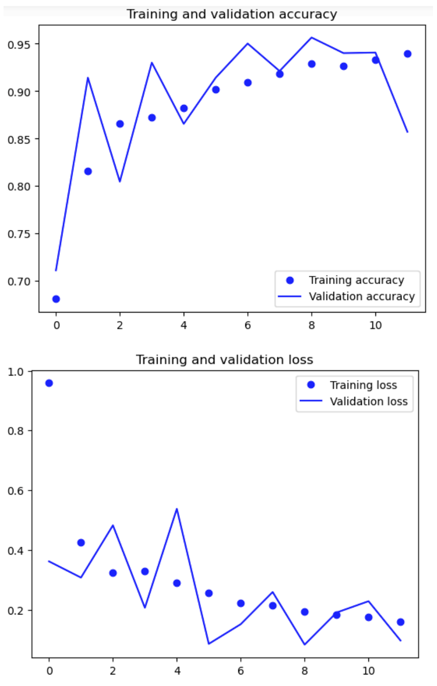


Figure 6-2: Results after the second adjustment

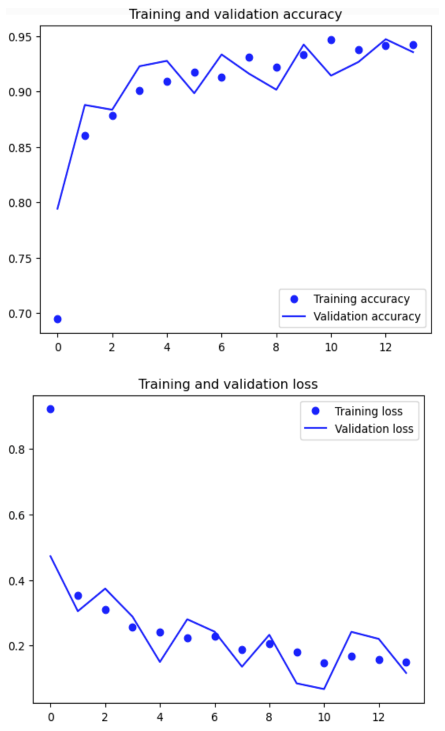


Figure 6-3: Results after the third adjustment

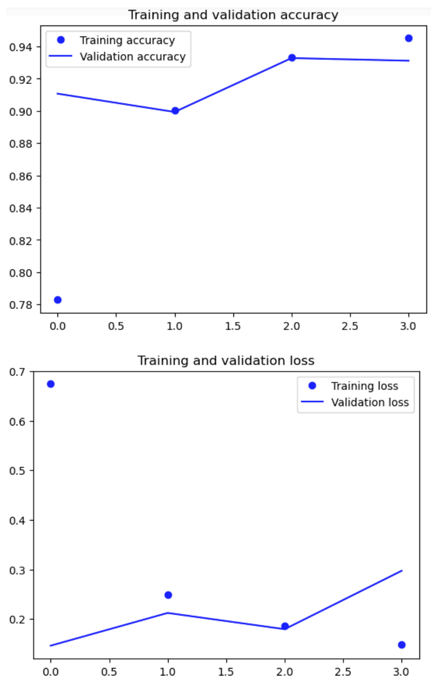


Figure 6-4: Results after the fourth adjustment

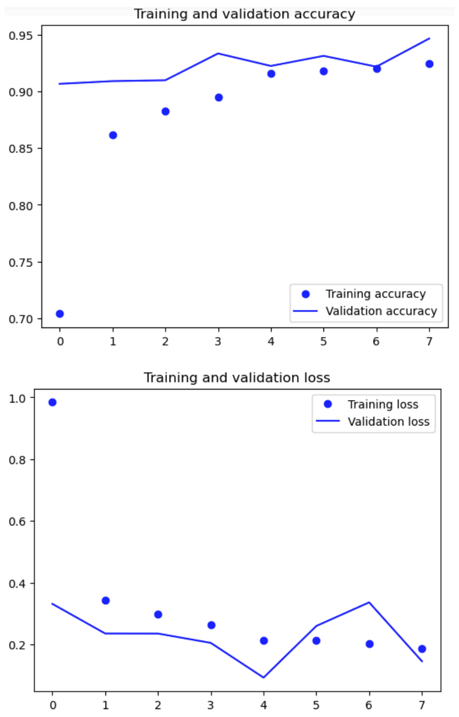


Figure 6-5: Results after the fifth adjustment

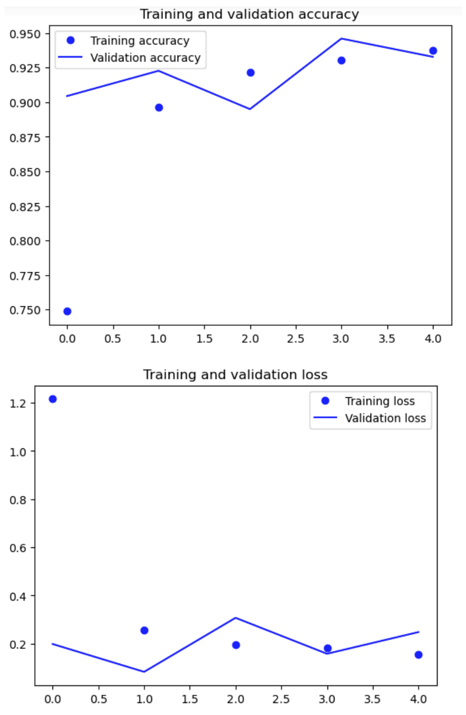


Figure 6-6: Results after the sixth adjustment

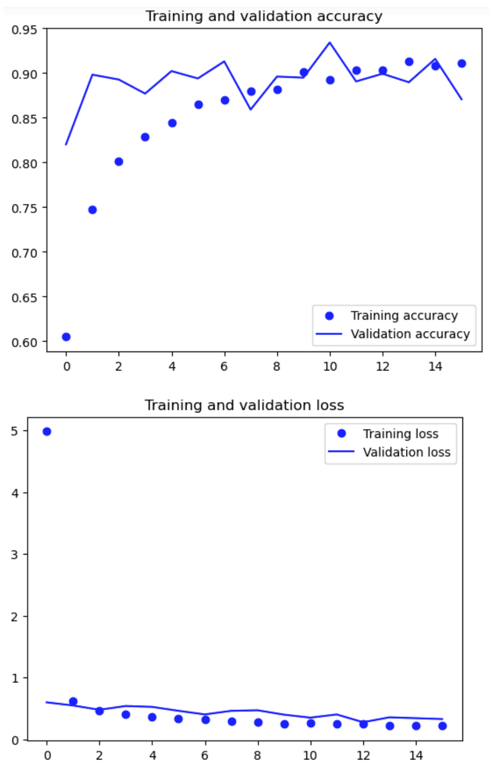


Figure 6-7: Results after the seventh adjustment