



UNIVERSITÀ
DEGLI STUDI
DI MILANO

Machine Learning

Cats and Dogs Image Recognition Algorithm

Luca Graziano, ID:987996

February 4, 2023

Abstract

The following analysis acts as a first introduction to Convolutional Neural Network (CNN) for image recognition problems and is part of the Machine Learning course for the degree course "Data Science for Economics" by Università degli studi di Milano. The dataset is composed of images representing cats and dogs and the main aim of the analysis is to build an efficient and well-performing binary classification CNN algorithm using Tensorflow. Multiple models with different complexity and architectures have been created to achieve this purpose.

Introduction

The analysis follows the preparation of the dataset as well as the development of several CNNs with different architectures in an attempt to maximise accuracy. First a baseline model has been created to act as benchmark. Immediately after, a second model with the same architectures has been run with a different optimiser to determine which is the optimal one which will be used for all other models. Secondly, the analysis focuses on finding an ideal depth and complexity for the CNN: additional convolution, pooling and dense layers have been added to the baseline model. Finally, different regularisation techniques are applied to the best performing model so far to see whether results could be improved.

Data and Preprocessing

The Dataset is a famously used dataset in the computer vision field. It became renowned following a Kaggle Competition in 2013 and is composed in total of 25000 images of cats and dogs.

The preprocessing of the images follows four steps:

- Removal of corrupted images
- Re-formatting original .Jpg images into RGB
- Scaling to a standard size of 180 height, 180 width
- Normalization of the RGB values from [0,255] to [0,1] for improved model performance

Setup and Evaluation

The evaluation metric and the setup are due to the fact that this is a classification problem. The loss function considered for this problem is binary cross entropy, that is the negative average of the log of corrected predicted probabilities computed as follows:

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

where y_i is the label associated with each data point and p_i is its predicted probability. Intuitively compare the predicted probability

The default optimiser for the CNN is the widely popular Adam optimiser that has proven to return the best results on a number of different problems. The RMSprop has been considered as an alternative and a specific test for determining whether it would be better has been made in the second model.

The evaluation metric chosen for analysing the results and comparing the models is accuracy computed as the fraction of the number of correctly predicted images over the total number of images. In this case accuracy suffices as an evaluation metric since the classes of the dataset are extremely balanced.

All models have been trained through 8 epochs with an early stopping function that prevents the model on training further if the accuracy tested on the validation set does not improve for three epochs consecutively. The number of epochs is relatively low but it has been chosen as it strikes a perfect balance between computation times and results achieved. No doubts, even better results would be achieved with more training epochs.

Convolutional Neural Network

Before starting with the analysis and the model development, a quick recap of what a convolutional neural network is and its properties is necessary. A convolutional neural network is a particular neural network typically used to perform Computer vision tasks. It receives images as inputs and outputs probabilities of how much the image belongs to each class.

A traditional architecture of a CNN is represented by Convolutional and Pooling Layers followed by a Flattened Layer and a Dense Layer.

A Convolution layer performs convolution, that is the process of applying filters to an input that results in activation in order to identify and learn specific features, that can either be a simple as lines or more complex ones, across the whole image. By adding multiple convolutional layers on top of another, it allows the CNN to identify higher level features.

A convolution layer is defined by five parameters:

- **Number of Filters** that determines the depth of the output feature map
- **Size of Filter** or also referred to as "kernel"
- **Padding** that solves the shrinkage of the kernel output feature map by adding a given amount of pixels
- **Stride** that influences the size of the output map by determining the distance between two tiles where the convolution is performed
- **Activation Function** that defines whether a neuron is activated or not

A pooling layer solves an issue raised with convolution, that is the sensitivity to where a specific identified feature is found in an image making the feature maps more robust to changes in position of the various features. There are two types of pooling layers: max-pooling layers and average pooling layer. The former allows the CNN to focus on the most important features while the latter balance the attention across all features.

A flatten layer turns the convolution output into a one-dimensional vector to be passed into a dense layer whose purpose is to match the extracted features to the image and identify the class to which it belongs. Adding multiple dense layers can improve the classification of the features.

The Baseline Model

The baseline model is a very simple one composed of:

- A 32 filters with size (3 x 3) convolution layer with a Relu activation function
- A max-pooling layer
- A dense of 256 nodes

The baseline model has Adam as optimiser and a total number of 19,670,401 trainable parameters. The accuracy on the test set is 71%.

```
# First model with one basic convolutional layer
model_1 = Sequential()

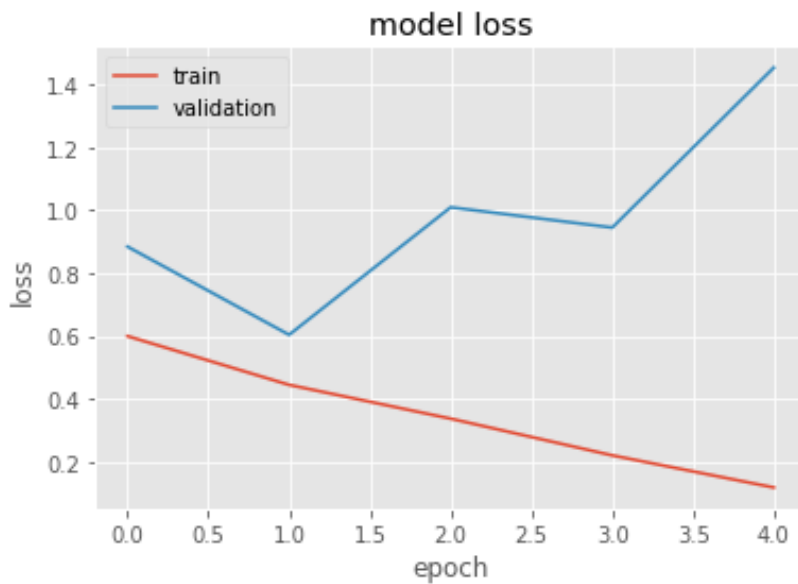
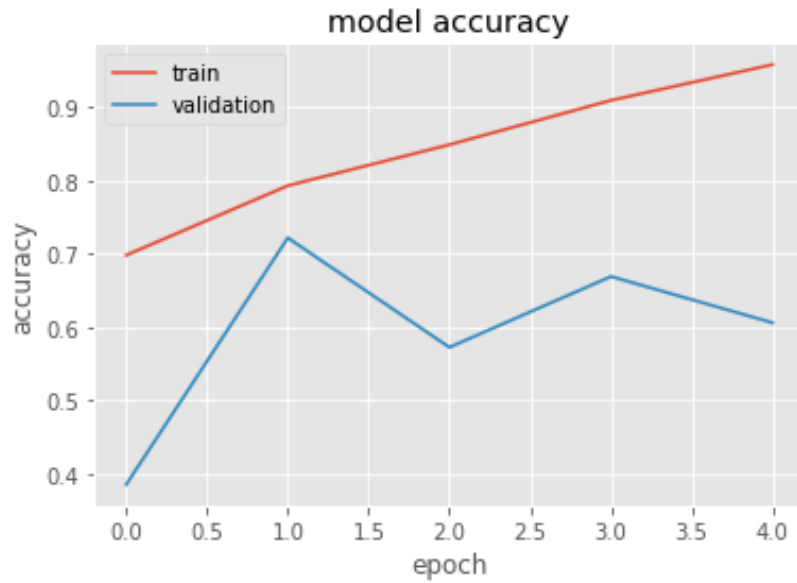
model_1.add(Conv2D(32, (3, 3), input_shape = image_shape)) #Basic convolutional for low level feature detection
model_1.add(Activation('relu'))
model_1.add(MaxPooling2D(pool_size=(2, 2))) # Focus on important features

model_1.add(Flatten())
model_1.add(Dense(256))
model_1.add(Activation('relu'))

model_1.add(Dense(1))
model_1.add(Activation('sigmoid'))

model_1.summary()

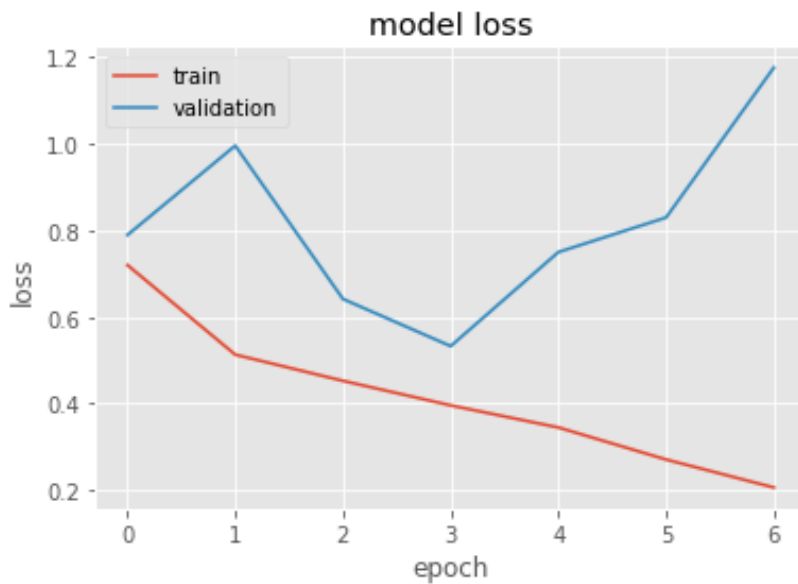
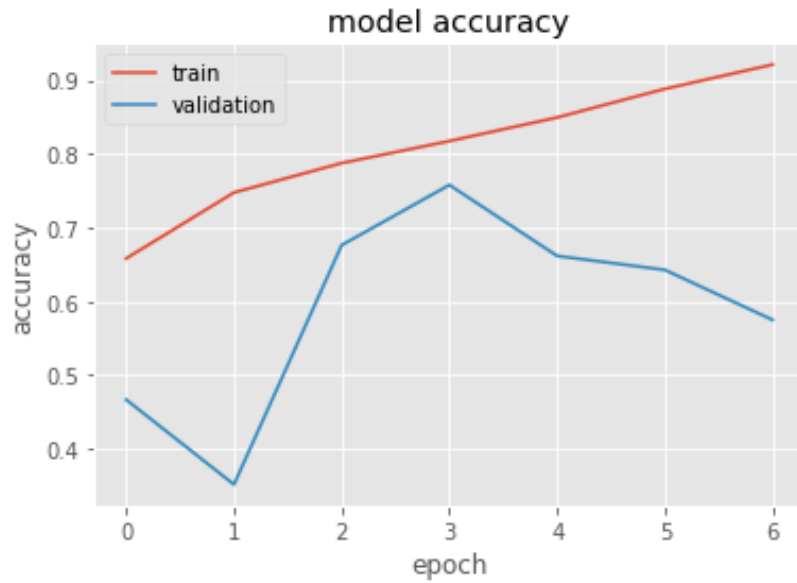
# Compile model
model_1.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])
```



Second model: determining the best optimiser

The second model has been specifically built with the purpose of determining the best optimiser to act as default for all subsequent models. The results achieved in the baseline model with the Adam optimiser are now compared to the results achieved using the baseline model with the RMSprop optimiser.

The accuracy of the second model on the test set is again 71%



The experiment turned out to be pretty inconclusive: the second model proved to be just as good with the RMSprop as it was with the Adam optimiser. All other models will now present Adam as optimiser being the default one for no particular reason.

Following, this conclusion the analysis continues by analysing results stemming from increased model complexity.

Third Model: increasing complexity

The third model is the first model in which the attention is shifted directly to the complexity/depth of the model. This model presents three convolution and max-pooling blocks and two dense layers. The structure breaks down as follows:

- a first 32 filters with size (3 x 3) convolution layer with a Relu activation function

- a first max-pooling layer
- a second 64 filters with size (3 x 3) convolution layer with Relu
- a second max-pooling layer
- a third 128 filters with size (3 x 3) convolution layer with Relu
- a third max-pooling layer
- a first 256 nodes dense layer
- a second 512 nodes dense layer

The third model has a 3,502,401 trainable parameters. The test accuracy is a striking: 80%. The results finally turn out to be a 9% improvement compared to the baseline model, highlighting the importance of having a deep enough CNN for ideal results.

```
# Adding complexity to the model

model_2 = Sequential()

model_2.add(Conv2D(32, (3, 3), input_shape = image_pixel))
model_2.add(Activation('relu'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

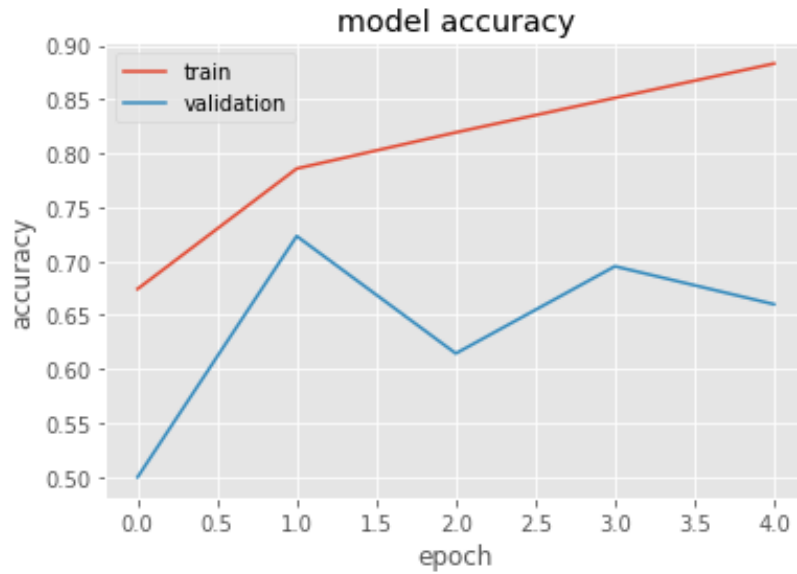
model_2.add(Conv2D(64, (3, 3)))
model_2.add(Activation('relu'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

model_2.add(Conv2D(128, (3, 3)))
model_2.add(Activation('relu'))
model_2.add(MaxPooling2D(pool_size=(2, 2)))

model_2.add(Flatten())
model_2.add(Dense(256, activation='relu'))

model_2.add(Flatten())
model_2.add(Dense(512, activation='relu'))

model_2.add(Dense(1, activation="sigmoid"))
```



Fourth Model: further increasing complexity

The fourth model tries to push the complexity of the CNN even further by adding another convolution and dense layer to the third model. The structure is the following:

- a first 32 filters with size (3 x 3) convolution layer with a Relu activation function
- a first max-pooling layer
- a second 64 filters with size (3 x 3) convolution layer with Relu
- a second max-pooling layer
- a third 128 filters with size (3 x 3) convolution layer with Relu

- a third max-pooling layer
- a fourth 256 filters with size (3 x 3) convolution layer with Relu
- a fourth max-pooling layer
- a first 256 nodes dense layer
- a second 512 nodes dense layer
- a third 512 nodes dense layer

The fourth model has 3,011,905 trainable parameters. Once again, adding complexity proved to be worth it: the test accuracy has now reached 85%. This a 5% improvement to the third model and a 14% improvement to the baseline model. Though it proved to be useful so far, adding complexity can be a double-edge sword and actually harmful to the model performance resulting in overfitting, that is the loss of generalisation by the trained model. Thus the analysis moves into understanding how different regularisation techniques can alter the behaviour of the model.

```
# Adding complexity

model_3 = Sequential()

model_3.add(Conv2D(32, (3, 3), input_shape = image_shape))
model_3.add(Activation('relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Conv2D(64, (3, 3)))
model_3.add(Activation('relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Conv2D(128, (3, 3)))
model_3.add(Activation('relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))

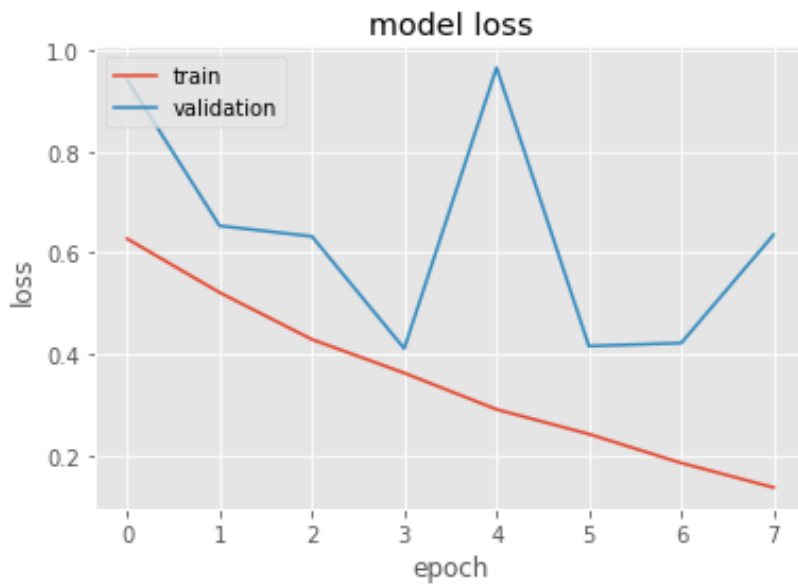
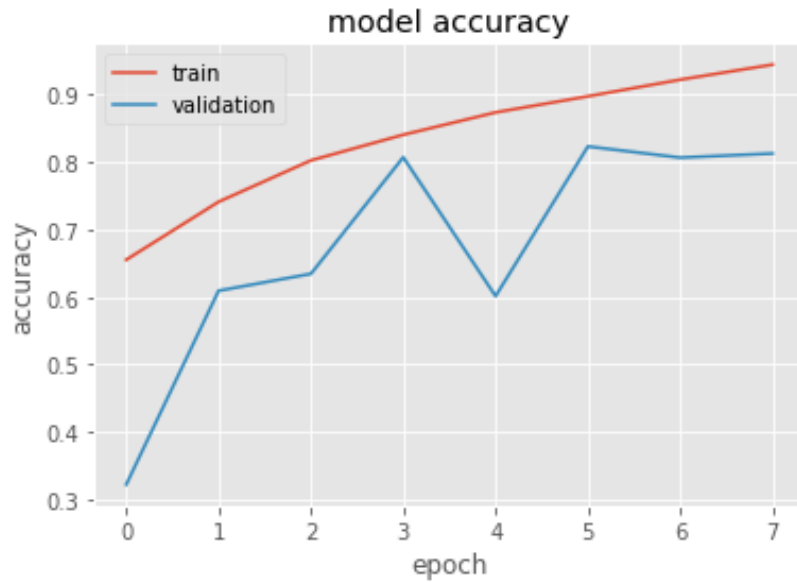
model_3.add(Conv2D(256, (3, 3)))
model_3.add(Activation('relu'))
model_3.add(MaxPooling2D(pool_size=(2, 2)))

model_3.add(Flatten())
model_3.add(Dense(512, activation='relu'))

model_3.add(Flatten())
model_3.add(Dense(512, activation='relu'))

model_3.add(Flatten())
model_3.add(Dense(512, activation='relu'))

model_3.add(Dense(1, activation="sigmoid"))
```



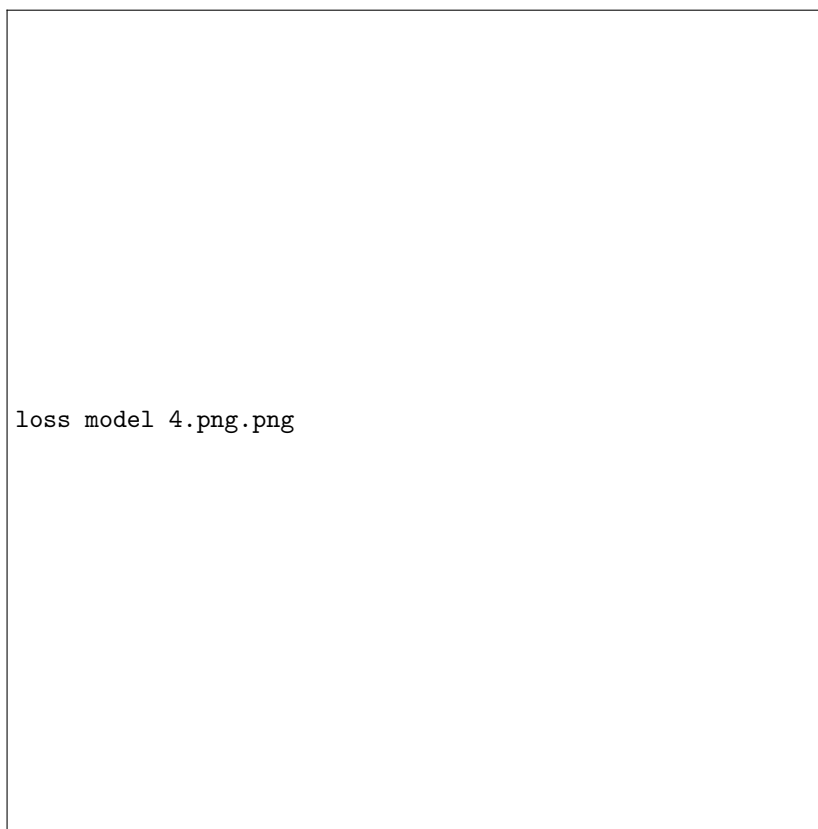
Fifth model: data augmentation

This fifth model is the first attempt at introducing regularisation as a form of overfitting prevention, a problem that NNs tend to suffer from a lot.

Data augmentation is a process to add some degree of randomness and improve the model's ability to generalise. It does so by generating new images out of existing ones, thus increasing the data points in the dataset, by altering some of its properties through three operations:

- Randomly flip horizontally half of the images
- Randomly rotate the images by a factor in the range $[-10\%, 10\%]$;
- Randomly zoom the image by a factor in the range $[-10\%, 10\%]$.

The test accuracy from the model trained on this supplemented dataset turned out to be 49%. This shows the degree of randomness added to the model resulted in the inability for the model to discern between the two classes and thus provide predictions based on some feature that stand out. Overall the data augmentation experiment turned out to be disappointing and no additional use of augmented data will be done in the next model.



Sixth model: Dropout and Batch Normalisation layers

On the sixth model different regularisation techniques are introduced to try and manage any degree of overfitting reached by the model.

This is done by introducing a Batch Normalisation layer, that applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1, and three dropout layers that randomly set according to some defined probability input units to 0.

The test accuracy of the model is 83%. Once again it goes down from the previous best model though the model is still able to generalise to a great degree. This poorer performance is probably due to the fact that the complex fourth model is already able not to overfit on the training data, thus making regularisation techniques unnecessary and apparently harmful to the model performance.

```
# Adding regularization with batch normalization and dropout

model_5 = Sequential()

model_5.add(Conv2D(32, (3, 3), input_shape = image_shape))
model_5.add(Activation('relu'))
model_5.add(MaxPooling2D(pool_size=(2, 2)))

model_5.add(Conv2D(64, (3, 3)))
model_5.add(Activation('relu'))
model_5.add(MaxPooling2D(pool_size=(2, 2)))

model_5.add(Conv2D(128, (3, 3)))
model_5.add(BatchNormalization())
model_5.add(Activation('relu'))
model_5.add(MaxPooling2D(pool_size=(2, 2)))
model_5.add(Dropout(0.1))

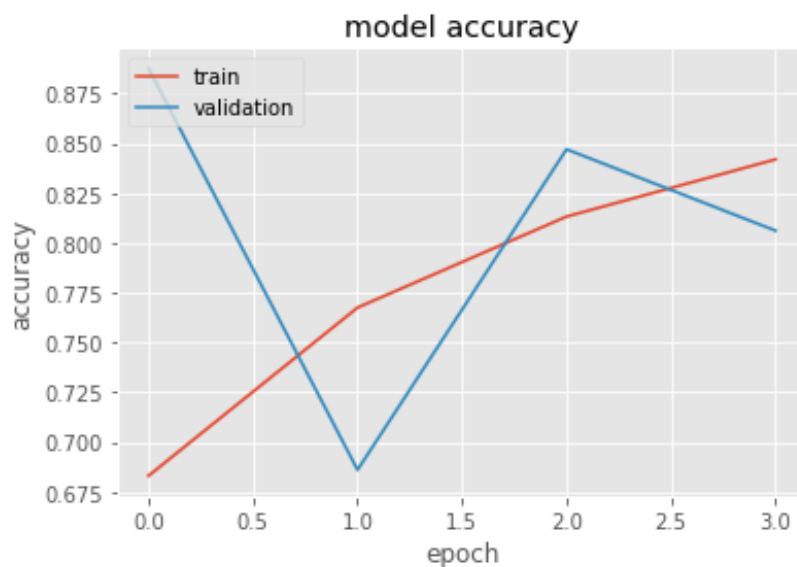
model_5.add(Conv2D(256, (3, 3)))
model_5.add(Activation('relu'))
model_5.add(MaxPooling2D(pool_size=(2, 2)))

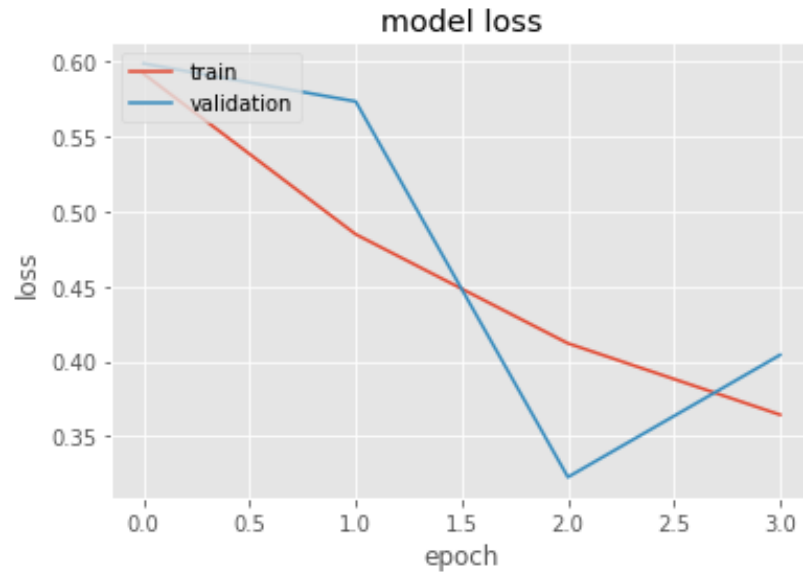
model_5.add(Flatten())
model_5.add(Dense(512, activation='relu'))

model_5.add(Flatten())
model_5.add(Dense(512, activation='relu'))
model_5.add(Dropout(0.1))

model_5.add(Flatten())
model_5.add(Dense(512, activation='relu'))
model_5.add(Dropout(0.2))

model_5.add(Dense(1, activation="sigmoid"))
```

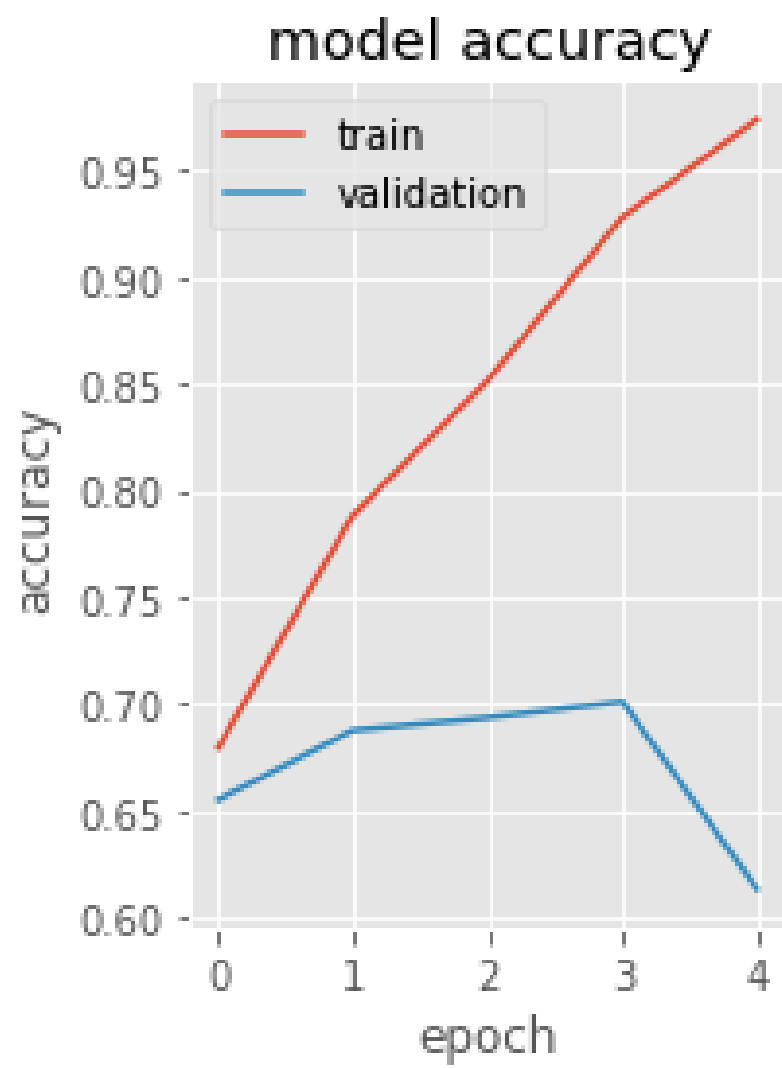


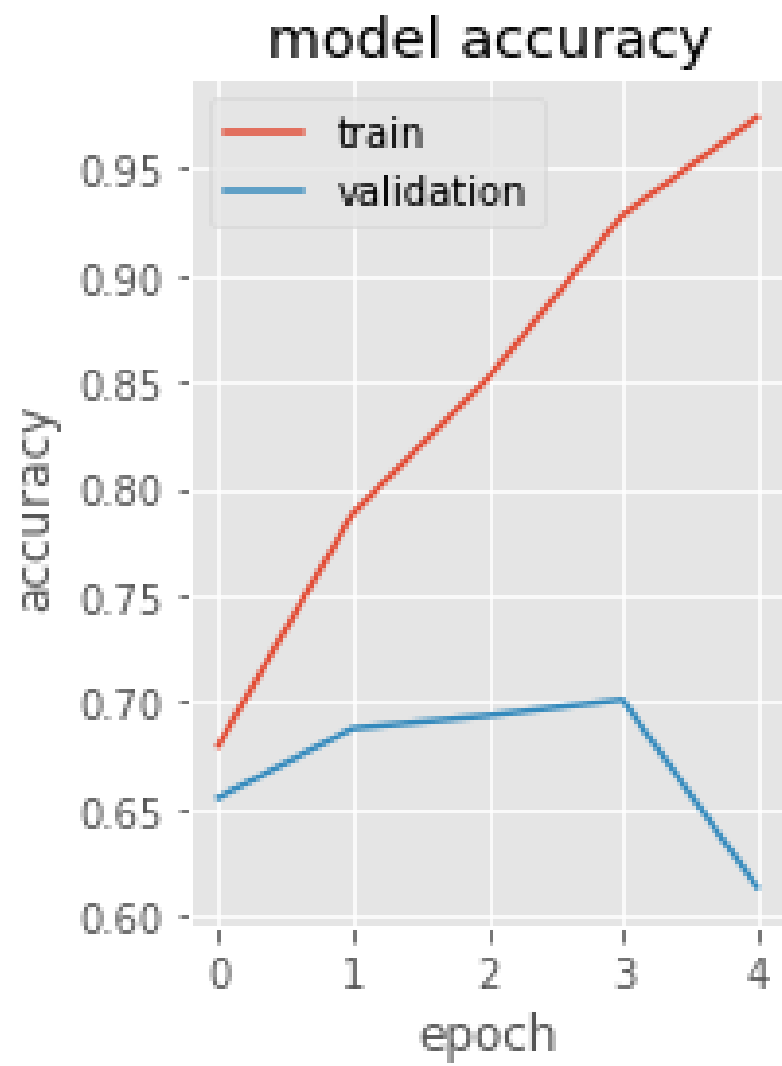


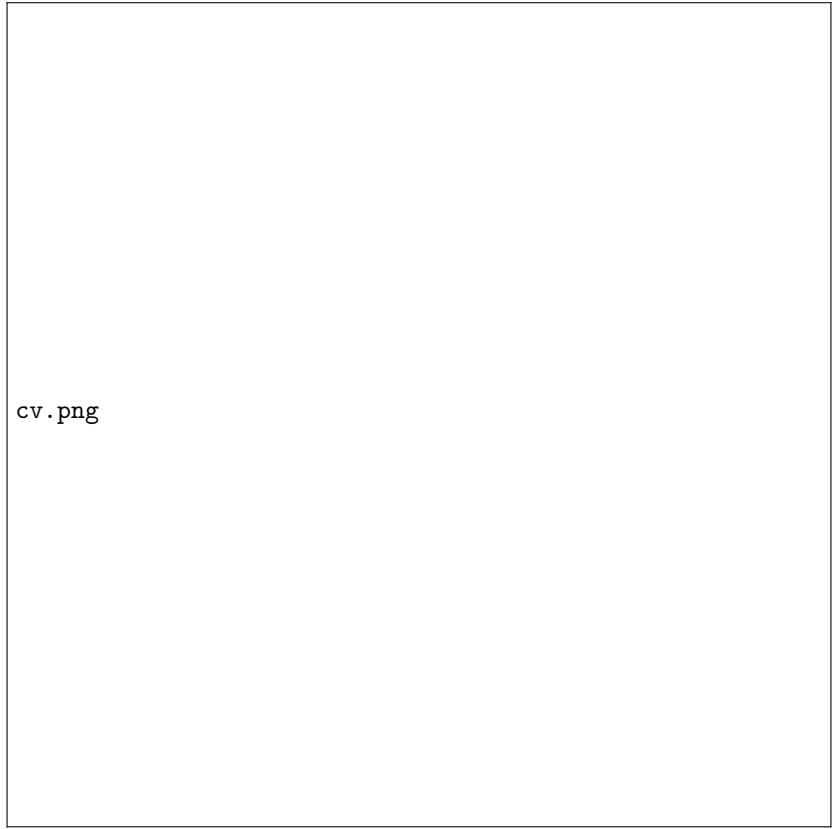
Seventh Model: 5-fold cross validation

The seventh and final model considers the best performing model so far and runs 5-fold cross validation on top of it as the repetition of the fitting of the model will improve estimates.

The test accuracy of this final model is 73%. Cross validation did not prove to be useful in improving the model performance possibly suggesting a possibility for model overfit in the previous experiments.







cv.png