Medical Image Classification

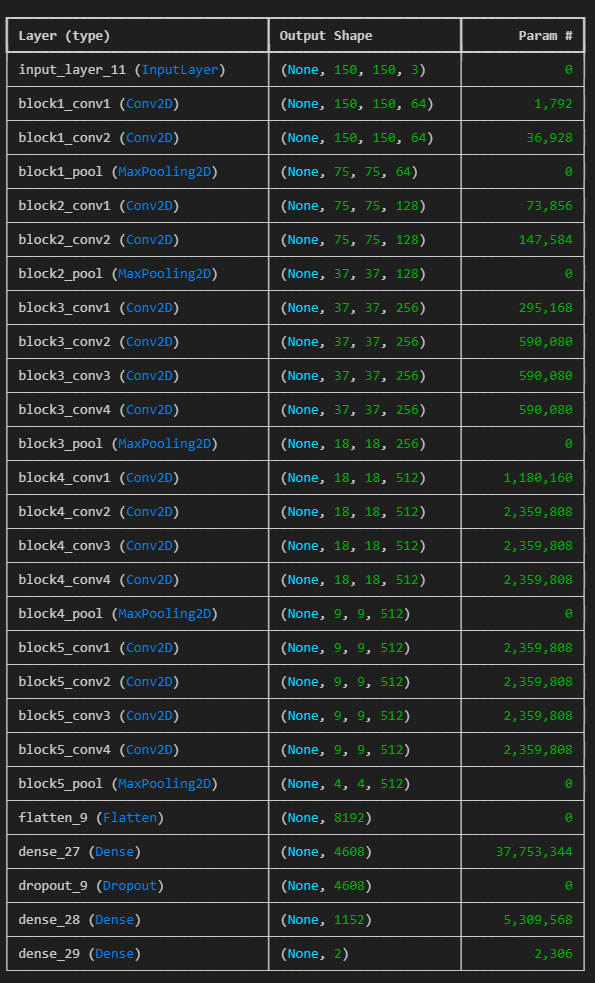
The 2 neural networks for the task will be described one after the other in order to improve the readability of the report instead of constantly comparing them in the same text.

Both networks use the same data set for pneumonia found on Kraggle **(ADD CITATION HERE)**

The naming of the dataset subfolders has been altered slightly but they retain their original structure of 3 folders – training, testing and validation. There are a total of 5863 files in the dataset stored in the form of JPEGs which eliminates the need to sanitize the data.

# Network №1

For the first network I decided to use the Visual Geometry Group (VGG-19) architecture. It consists of 19 layers – 16 convolutional and 3 fully connected. One of the reasons for my choice was the simplicity and modularity of the baseline model.

My approach consisted of creating a first-iteration model using the pretrained weights available from the base model and training it on the training data in my dataset. After that I would save the evaluate the model against the testing and validation data and save the weights. Furthermore, this process is repeated 2 more times by utilizing the saved weights from the previous iteration. For initial runs I utilized parameters with the following values for all 3 iterations:

* 1 epoch with 50 steps
* Plateau learning rate reduction with a factor of “0.5” and patience of “3”
* Stochatic Gadient Descent with a leaning rate of “0.0001” and momentum of “0.01”
* Image size of 150x150

The images were loaded without any changes for the validation and testing data. The training data was loading with the following parameters:

* Horizontal\_flip = 0.4
* Vertical\_flip = 0.4
* rotation\_range=40,
* shear\_range=0.1
* width\_shift\_range=0.4
* height\_shift\_range=0.3

Image 1. Model summary during initial testing

During the first testing period the network achieved an accuracy of “0.5000” against the testing data and “0.6308” against the validation data. Which only slightly improved to “0.6420” against the validation data due to using the same parameters everywhere.

For my next iteration I decided it increase the image size to 224x224 to match the designed size for VGG-19. This change did not change the evaluation accuracy which was expected. On the other hand, it increased the average accuracy when fitting the training data to the model from “0.6201” to “0.8021”. However, this increased the execution time by a factor of 3, which highlights one of the drawbacks of the VGG-19 architecture – very taxing on hardware in case of bigger datasets.

For the next iteration I decided to increase the batch size of the training data generator to 64 (from 32) and the epochs to the following:

* Model #1 is increased to 15 epochs with 250 steps per epoch
* Model #2 is increased to 10 epochs with 200 steps
* Model #3 is increased to 5 epochs with 200 steps

All models have had their patience increased to 8 for early stopping and 5 for rate reduction on plateau. These changes resulted in the following results:

* Model #1 – 0.8012 accuracy against testing data and 0.75 accuracy against validation data
* Model #2 – 0.7804 against testing data and 0.625 against validation data
* Model #3 – 0.7916 against testing data and 0.625 against validation data

Such results most likely indicate overfitting. In order to combat this I made changes to the training image generation in the following ways:

* Reduce the rotation range from 40 degrees to 20 degrees
* Reduce the width and height shift changes to 0.1
* Disable vertical flipping
* Add brightness range between 80% and 120% of the original

The changes resulted in 0.75 validation and 0.8493 testing accuracy.  
The next iteration consisted of changing the momentum of the final model to be “0.8”. This change led to no increase in validation accuracy but increased testing accuracy to 0.889 which means that the model is still overfitting. In order to combat this I decided to lower the steps per epoch in the final model down to 100 which brought the validation accuracy to “0.8750” and the testing accuracy to “0.9208”

# Network №2

I decided to make the second network use the Sequential API. I started with a bare-bones model at the start by adding an Input Layer, a single convolutional layer, pooling layer and a fully connected output layer. The performance was expected to be bad since it’s used as a proof of concept for the pipeline. The images are loaded in batches of 64 and size of 224x224.

The train generator had the following parameters:

* horizontal\_flip = True,
* vertical\_flip =False,
* rotation\_range = 20,
* shear\_range = 0.1
* width\_shift\_range = 0.2
* height\_shift\_range = 0.2
* brightness\_range = [0.8, 1.2]
* fill\_mode = "nearest"

The initial fitting and evaluation with 2 epochs and 100 steps per epoch returned 0.5000 accuracy to both testing and validation data. Below you can see the model summary as Image 2.

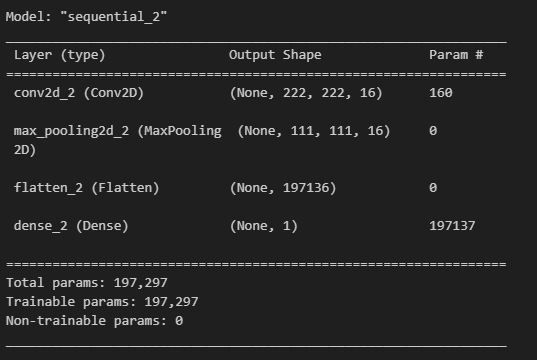


Image 2. Initial model summary

For the next iteration I decided to add Dropout to avoid overfitting and randomly disable neurons, the initial value I chose was 0.2. Additionally, I added another convolutional layer with the doubled filters (32) and increased the epochs to 5 and decreased the steps per epoch to 50. These changes lead to a slightly lower loss but no change in accuracy.

For the next iteration I added 2 more convolutional layers with 64 and 128 filters respectively. Furthermore, I change the class mode for the image generators to “binary” from “categorical” which was an oversight on my part due to the network only needed to differentiate between pneumonia and healthy. These changes skyrocketed the accuracy to 0.7836 against testing data and 0.875 against validation data.

For the next iteration I decided to play around with the hyperparameters. I started by adding and early-stop with patience of 4 and increasing the epochs to 10. Which increased the loss by 3 on the validation data and decreased the accuracy to 0.5625 against validation data and 0.7115 against the testing data.

In order to combat the issues from the previous iteration I added a rate reduction to avoid plateauing. This led to 0.6875 accuracy against validation data and 0.8782 against testing data. Such values suggested overfitting.

I attempted to combat this by increasing the epochs to 15, adding 2 more Dense layers with 0.5 dropout between them with 256 and 512 respectfully. This lowered the test loss but skyrocketed the validation loss. After noticing that the validation accuracy never changed after the 2nd epoch, I realized that I am over-regularizing the model with the dropouts.

In order to combat the issue, I lowered the dropout to 0.3 and 0.1 from 0.5 and increased the epochs to 20. Additionally, I set the starting learning rate to 0.0005 and added a new convolutional layer with 256 filters in order to increase the model complexity. These changes lead to no changes in the accuracy.

The next iteration involved adding batch normalization and increasing the early stopping patience to 7 due to the model training stopping after 11 epochs. This led to a validation accurate of 0.8750 and testing accuracy of 0.8301.

I noticed that after the rate reduction activates the validation accuracy dropped drastically. In order to avoid this, I increased the factor of reduction from 0.5 to 0.8 in the next iteration as well as increase the minimum learning rate to 0. 0005. Additionally, I increased the epochs to 30 and the steps per epoch to 82 in order to process all the images and give the model plenty of time to train itself. This led to a validation accuracy of 0.75 and testing accuracy of 0.729 but the early stopping triggered at epoch 19/30.

For the next iteration I decided to up the patience on it to 15. This change allowed the model to fully train and thus increased the validation accuracy to 0.8125 and the testing accuracy to 0.8782.

Attached below as Image 3 is the final summary of the model.

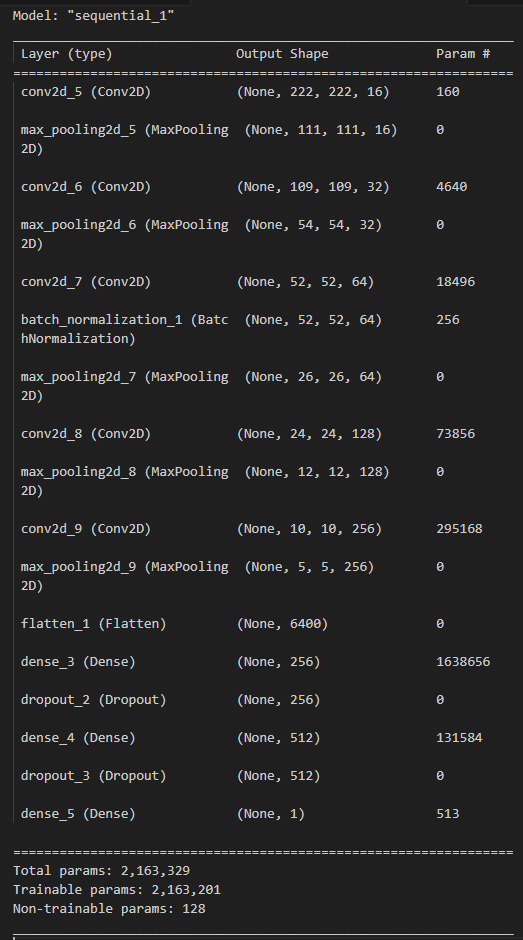


Image 3 Final model summary

Sentiment Analysis

Both networks use a popular dataset from Kaggle (ADD CITATION HERE). The dataset itself consists of a CSV with 50000 values in 2 columns – the review and the sentiment. The sentiments are evenly split.

# Network №1

After starting to explore the data I decided to change the format of the data. Instead of sentiment being “positive” or “negative” I switched it to “1” and “0”.

For splitting the dataset into training and testing I decided to use the sklearn package and train\_test\_split. Initially I set the test\_size to be 0.2 (20% of the data goes for testing and the remaining is used for training). I chose this method since the data is balanced but I was prepared to fine-tune the test size parameter.

I decided to use the Tokenizer with max word count of 3000 for the initial model creation.

For cleanup I looked up a regular expression for removing emoticons and shapes. Additionally, I removed all non-standard characters and any HTML tags.

For the model I decided to start with a basic Sequential model with an Embedding layer (input\_dim = 5000, output\_dim = 128, input\_length = 200), an LSTM layer with 0.2 dropout and a final Dense layer. The model summary can be seen in Image 4.

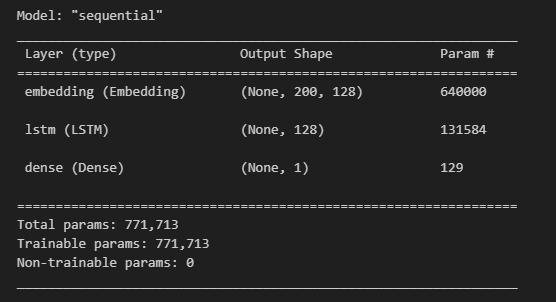


Image 4 Sentimental Analysis model 1

When fitting the model with the training data I decided to test with 5 epochs and batch size of 64 and an initial validation split of 20% and 150 steps per epoch. After the first run the results were the following:

* 0.8743 accuracy against training data
* 0.3097 loss against training data
* 0.8592 accuracy against testing data
* 0.3332 loss against testing data

This indicated the possibility of slight overfitting. In order to combat it I increased the dropout to 0.4.

* 0.8708 accuracy against training data
* 0.3142 loss against training data
* 0.8630 accuracy against testing data
* 0.3152 loss against testing data

For the next iteration I decided to play around with the parameters for the words by increasing the tokenizer words to 10000 and the input\_dim on the embedding layer to 10000. Additionally, I increased the max length to 400 in case some of the reviews are longer. These changes increased the time to fit the model by 2 times but resulted in the following:

* 0.8899 accuracy against training data
* 0.2716 loss against training data
* 0.8786 accuracy against testing data
* 0.3141 loss against testing data

For the next iteration I decided to increase the epochs from 5 to 10 in order to give the model sufficient time to train. This resulted in the following:

* 0.9153 accuracy against training data
* 0.2252 loss against training data
* 0.8921 accuracy against testing data
* 0.2672 loss against testing data

The final state of model can be seen in Image 5 below.

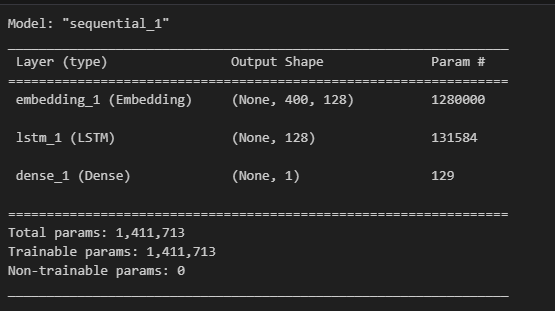


Image 5 Sentimental Analysis model 1 Final state

# Network №2

The data cleanup and tokenizer are identical to the previous network. The parameters used were:

* Tokenizer with num\_words = 10000
* Maxlen = 400 for padded sequences

The major differences are in the model itself.  
The initial version had 4 layers:

* Embedding with input\_dim = 10000; output\_dim = 128, input\_length = 400
* Conv1D layer with a filter width (kernel size) of “5”
* GlobalMaxPooling1D layer
* Dense layer to output binary

A summary of the model can be found below as Image 6

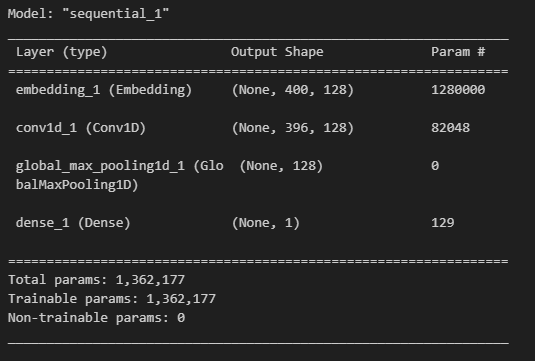


Image 5 Sentimental Analysis model 2 model

When fitting the model, I used 10 epochs with a batch\_size of 64, 150 steps per epoch and a 0.3 validation split. This resulted in the following:

* 0.9911 accuracy against training data
* 0.0458 loss against training data
* 0.8907 accuracy against testing data
* 0.2906 loss against testing data

While the time it took was considerably faster than the first network, these results showed a massive overfitting when using the training data. In order to counteract it I added a Dropout layer with 0.5:

* 0.9808 accuracy against training data
* 0.0682 loss against training data
* 0.8812 accuracy against testing data
* 0.2771 loss against testing data

The overfitting was lowered in the first 5 epochs, however after that it memorized the data again. My next attempt was to add a BatchNormalization layer and lower the filters in the Convolution layer to 64 as well as adding L2 regularization in the Convolutional layer with a value of 0.03.

* 0.9276 accuracy against training data
* 0.2935 loss against training data
* 0.8705 accuracy against testing data
* 0.4119 loss against testing data