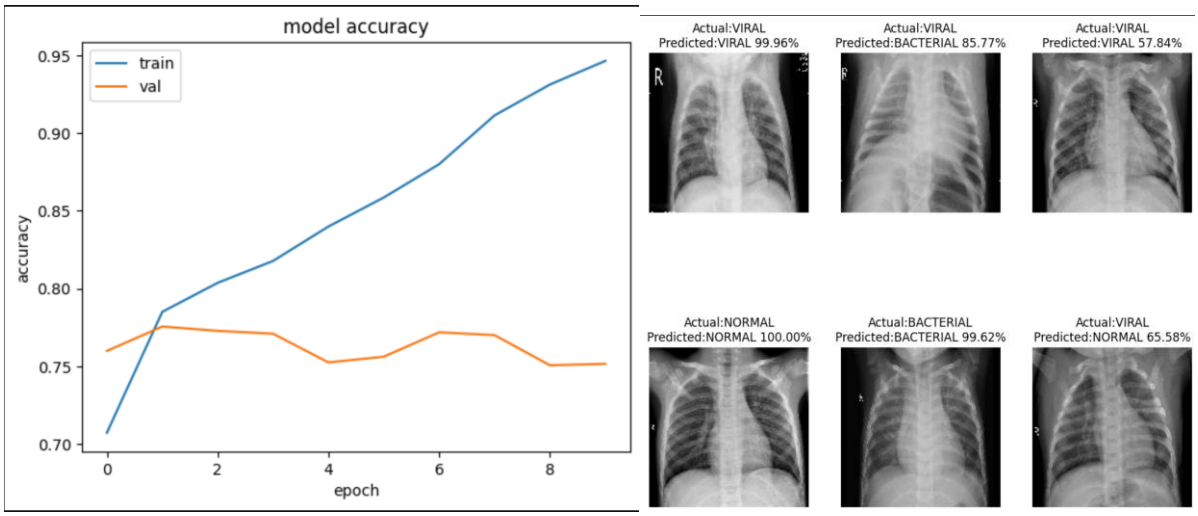
Honours Degree in Computing

Computer Vision
Assignment 2

Submitted by:
Konstantinas Avlasevicius B00148740

Baseline

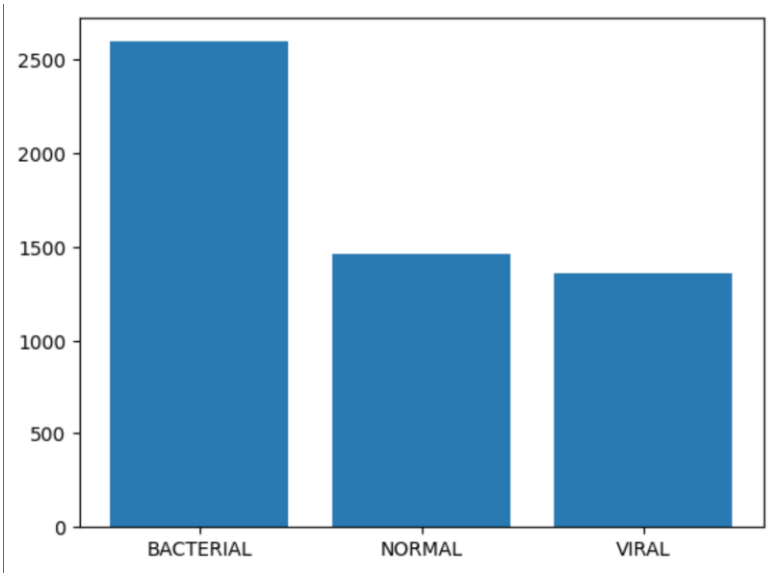


While training time can vary from the device that was used, the current device (specs in Appendix B) takes 10 seconds per epoch with a total time of 110 seconds per 10 epochs (including plotting charts and displaying images, etc.).

Test accuracy: 0.744
Time: 10s PE (Per Epoch)

As per the model accuracy chart, the model seems to be overfitting (accuracy while training is increasing per epoch count while validation data remains the same and slightly decreasing). Further inspection shows a high-class imbalance as per the image below with class distribution:

- Class 0 Bacterial – 2596
- Class 1 Normal – 1461
- Class 2 Viral – 1362



Address class imbalance

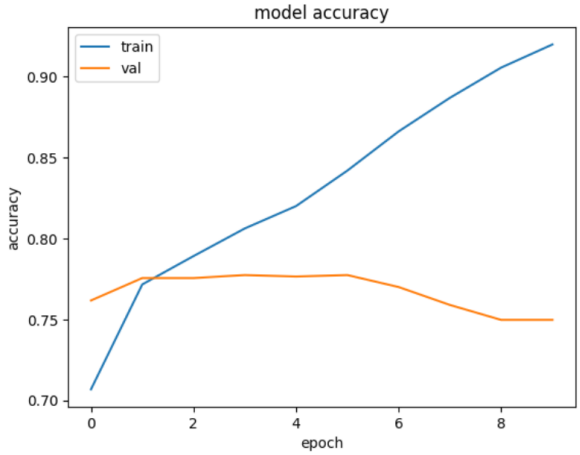
To address the class imbalance, the weight will be calculated for each class:

- Class 0 Bacterial – Images: 2596, Weight: 1.0
- Class 1 Normal – Images: 1461, Weight: 1.7914163090128756
- Class 2 Viral - Images: 1362, Weight: 1.9252767527675276

```
def getClassWeights(ds):
    labels = np.asarray(list( ds.unbatch().map(lambda x, y: y)  ))
    count_0 = np.sum(labels == 0)
    count_1 = np.sum(labels == 1)
    count_2 = np.sum(labels == 2)

    print(count_0, count_1, count_2)

    max_class = max(count_0,count_1,count_2)

    weight_0 = max_class/count_0
    weight_1 = max_class/count_1
    weight_2 = max_class/count_2
    class_wght = { 0 : weight_0 , 1 : weight_1, 2 :  weight_2 }
    return class_wght
```

Applying weights to a model slightly increased the performance of the model while it was still overfitting.

Test accuracy: 0.789
Time: 10s PE (Per Epoch)



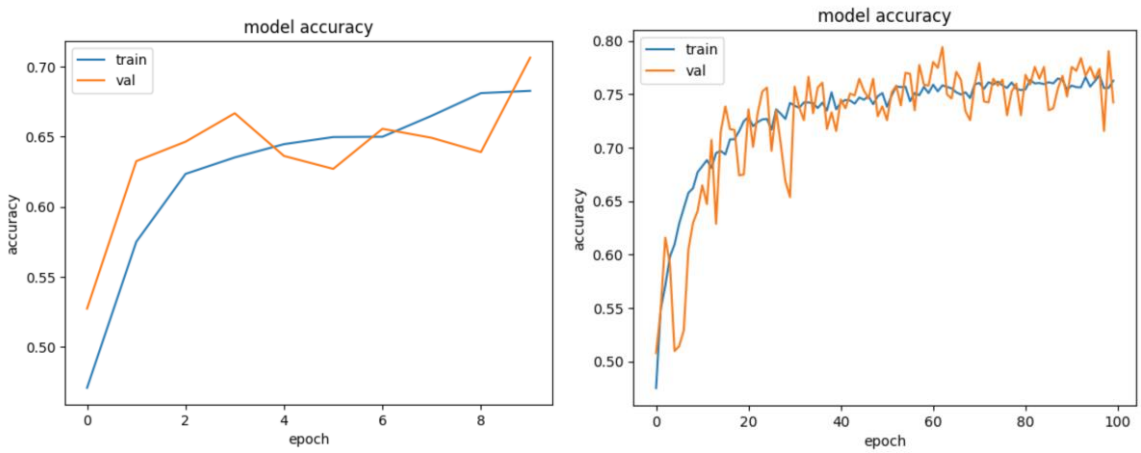
Overfitting

Data augmentation layers can address this issue since overfitting could be introduced due to a lack of data. Four hidden layers were added to a base model.

- RandomFlip – flips images vertically and horizontally.
- RandomRotation – randomly rotate the image.
- RandomTranslation – slightly translate/move the image to asides
- RandomZoom – zoom an image

```
model.add(RandomFlip("horizontal and vertical"))
model.add(RandomRotation(0.2))
model.add(RandomTranslation(height_factor=0.2, width_factor=0.2))
model.add(RandomZoom(0.3))
```

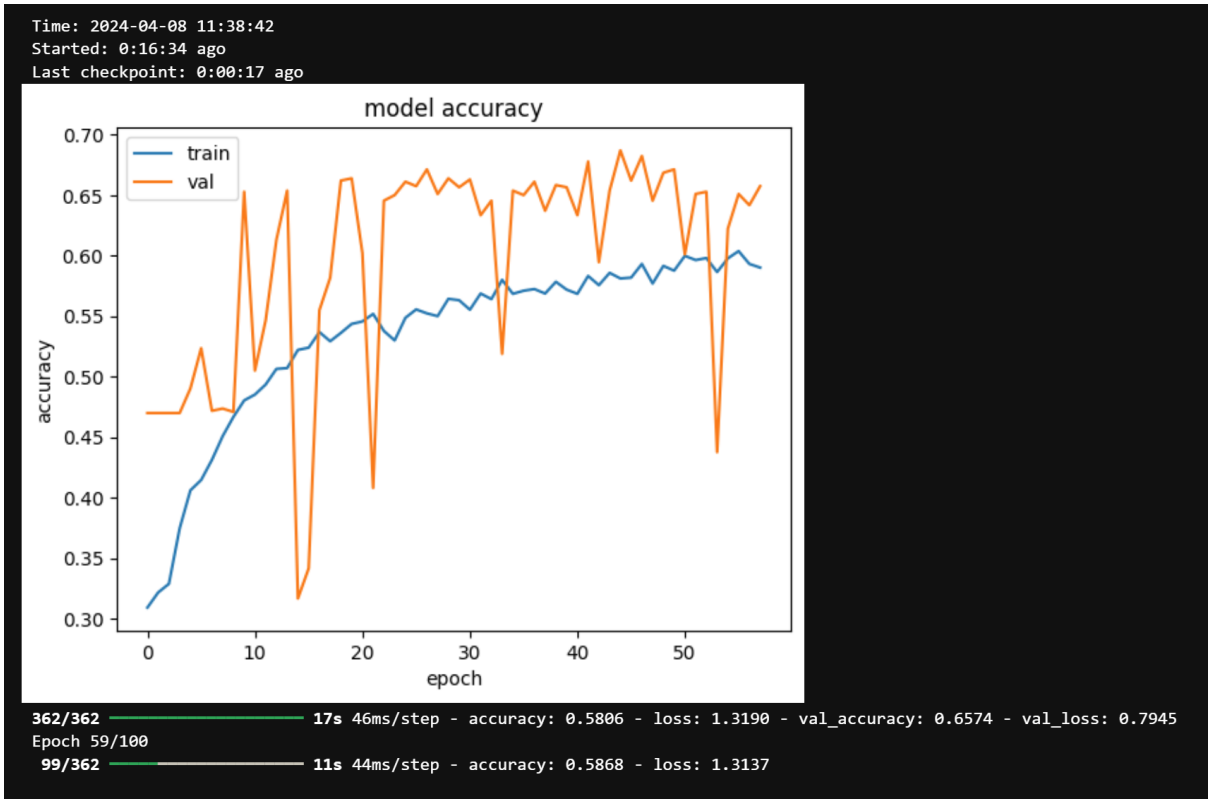
The first attempt with augmentation layers shows that it manages to deal with overfitting while performance drops to 0.707. The second attempt was made by increasing the epoch count, which further confirms that augmentation worked well in dealing with an overfitting issue. While model accuracy is positively correlated to epochs, it becomes inefficient after around 30-40 epochs. After 100 epochs, accuracy was still below baseline.



Test accuracy: 0.70
Time: 12s PE

Attempt to increase hidden layers

An attempt was made to increase the Dense layers. Adam's Optimiser does not work well with 10+ hidden layers; therefore, an SGD Optimiser was used instead. Tested with 10, 15 and 20 layers, which didn't give any advantage (Image below test with 15 layers). The training model becomes much slower (from 10s PE to 17s PE), but it still stabilises around 20-30 epochs. For further tests, additional layers were removed from a model while the optimiser remained SGD.



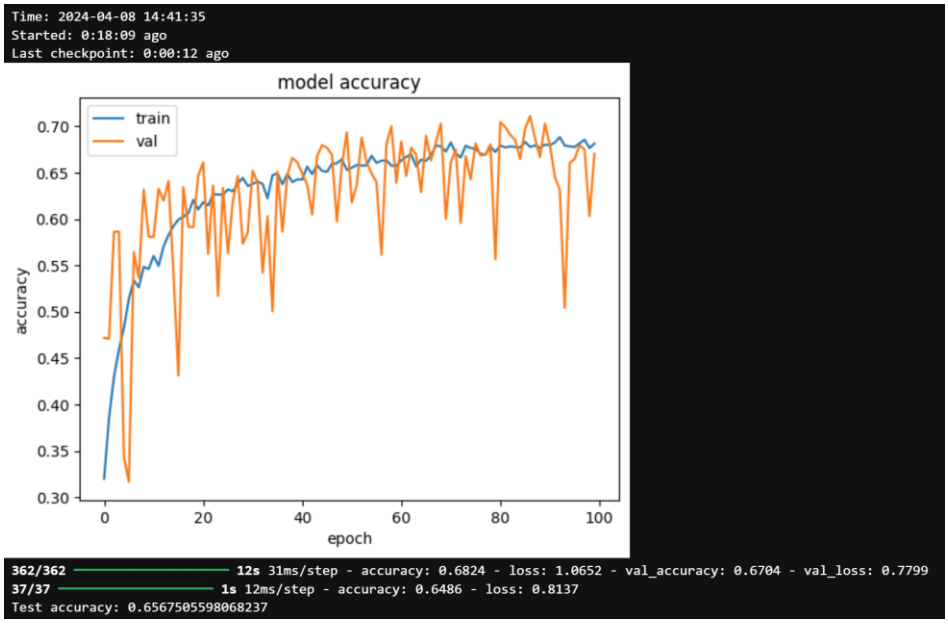
Keras tuner

A Keras tuner is an open-source project that automates parameter tuning and advises the best set to use in your model. It's probably the easiest way to automate parameter tuning and will be applied for this project.

The parameters were selected for tuning as per the table below.

	Parameter	Values to test	Result
Optimiser (SGD)	Learning rate	[0.01, 0.001, 0.0001]	0.01
	Momentum	[0.3, 0.5, 0.7]	0.3
Augmentation Layers	Random Rotation	[0.3, 0.5, 0.7]	0.5
	Translation factors	[0.3, 0.5, 0.7]	0.3
	Random Zoom	[0.3, 0.5, 0.7]	0.5
Dropout Layer	dropout	[0.1, 0.2, 0.4, 0.6]	0.6
Units for Dense Layers	units	min_value=32, max_value=512, step=32	256

After receiving the results, they were applied to a model. Unfortunately, the Keras tuner proposed settings did not work well, and the results were below the baseline. That could happen because the model shows unstable performance. There are high spikes in performance, which may mislead the tuner. Results from the tuner were not applied for further testing and may be revised at a later time.



Pretrained models

Many pre-trained models are available and could be applied for this task. A list of pre-trained models available for Keras can be found at https://keras.io/api/applications/. Most of the pre-trained models have a large set of layers, and they are much slower to train than the current baseline. Testing all of them is a time-consuming process; only a few will be tested to take a brief overview of performance gains and the time it takes to run them. Due to shape reduction produced with the MaxPooling2D(2,2) layer, the shape becomes too small to feed into the pre-trained model (some of the pre-trained models accept a minimum shape of 32x32). Therefore, all convolution layers were removed before the test.

Model	Model Accuracy	Time (PE)
VGG19	0.57	1m20s
Xception	0.68	50s
ResNet50V2	0.73	2m15s
EfficientNetB0	0.62	1m40s

Experimentation

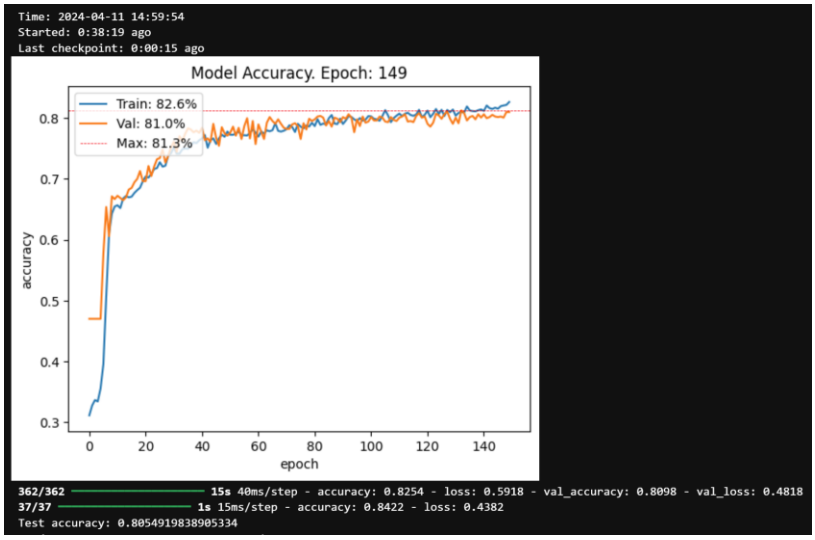
Experimentation to create a similar model with fewer pooling layers so it can be mixed with pre-trained models. Also, some layers' parameters were changed: the Filter for Conv2D increased, and Dense units in the hidden layer increased the number of pixels in an image. I added the ResNet50V2 pre-trained model as it showed the best performance in the previous step. The model trains very slowly (2m10s PE) and does not produce stable results.

```
h.tuner(train_ds,overwrite=False)

model = Sequential()

model.add(Input(shape = (cfg.img_height, cfg.img_width, cfg.img_channels)))
# Augmentation  Layer
model.add(RandomFlip("horizontal and vertical"))
model.add(RandomRotation(random_Rotation))
model.add(RandomTranslation(height_factor=translation_factors, width_factor=translation_factors))
model.add(RandomZoom(randomZoom))
#preprocessing
model.add(Rescaling(1.0/255))
# conv layer
model.add(Conv2D(128, (2,2), activation = 'relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (2,2), activation = 'relu'))
model.add(Conv2D(128, (2,2), activation = 'relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(3, (3,3), activation = 'relu'))
model.add(BatchNormalization())
#model.add(pred_model)
model.add(keras.layers.Flatten())
model.add(Dense(units, activation = 'relu'))
model.add(Dropout(dropout))
model.add(Dense(cfg.img_height*cfg.img_width, activation = 'relu'))
model.add(Dropout(dropout))
# Output layer
model.add(Dense(cfg.num_classes, activation = 'softmax'))
```

Layer (type)	Output Shape	Param #
random_flip (RandomFlip)	(None, 128, 128, 3)	0
random_rotation (RandomRotation)	(None, 128, 128, 3)	0
random_translation (RandomTranslation)	(None, 128, 128, 3)	0
random_zoom (RandomZoom)	(None, 128, 128, 3)	0
rescaling_2 (Rescaling)	(None, 128, 128, 3)	0
conv2d_4 (Conv2D)	(None, 127, 127, 128)	1,664
conv2d_5 (Conv2D)	(None, 126, 126, 128)	65,664
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 128)	0
conv2d_6 (Conv2D)	(None, 61, 61, 3)	3,459
resnet50v2 (Functional)	(None, 2048)	23,564,800
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 512)	1,049,088
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 16384)	8,404,992
dropout_1 (Dropout)	(None, 16384)	0
dense_2 (Dense)	(None, 3)	49,155

Total params: 33,138,822 (126.41 MB)
Trainable params: 33,093,382 (126.24 MB)
Non-trainable params: 45,440 (177.50 KB)

Further experimentation was conducted to exclude the pre-trained model. As most impacts had an augmentation layer, an attempt was made to find the balance between overfitting and its effect on image transformation/zoom … etc. Many experiments were conducted to decrease augmentation and increase hidden layers to find the best prediction result. The model ran for 150 epochs. While the model lost performance in about epoch 80 and there is some noticeable overfitting starting at epoch around 130, it did produce output slightly higher than baseline. **Test accuracy: 0.805, 15s PE**



Model Performance Measurements

Model performance can be measured in multiple ways; most commonly, for classification problems, performance is measured using *Accuracy*, *Precision*, *Recall* and *F-1* (harmonic mean).

Accuracy is probably the most straightforward measurement, where the total of correctly predicted samples is divided by the total number of samples.

$$\frac{Total\ Predicted\ Correctly}{Total\ Samples}$$

Precision evaluates the performance of the positively predicted class actually belonging to this class. It is calculated by taking the sum of the correctly predicted label and dividing it by the sum of both the correctly predicted label and the incorrectly predicted label.

$$\frac{True\ Positive}{True\ Positive + False\ Positive}$$

Recall is a very similar approach. The main difference that recalls aim to address is falsely predicted negative (does not belong to this class), and it does this by adding false negatives to the equation.

$$\frac{True\ Positive}{True\ Positive + False\ Negative}$$

F1 attempts to combine both Precision and Recall measurements into a single equation. This helps to have better observation of model performance.

$$\frac{2 * True\ Positive}{2 * True\ Positive + False\ Positive + False\ Negative}$$

Model evaluation

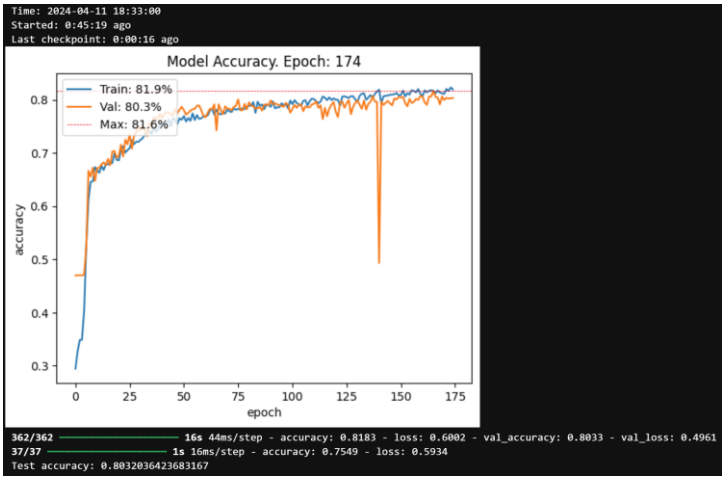
A classification report is generated from the sklearn module to measure this model's performance. The project is based on predicting medical conditions. Therefore, recall is probably the most important measurement as it addresses false negative prediction, which is preferred to avoid medical conditions (telling a patient that he has no infection in his chest when he actually has one could lead to serious complications or even death).

The classification report shows that overall accuracy is good, with a score of 86%. If we look at recall (as a more critical metric for medical diagnosis), the result is disappointing in predicting the Viral class, while other classes, such as Bacterial, have pretty good results. The best results are for the Normal class, with 96% for both precision and recall, which is also reflected in the F1 score.

Classifier with train data:

Label mapping: 0 – Bacterial, 1 – Normal, 2 – Viral.

	precision	recall	f1-score	support
0	0.83	0.91	0.87	2087
1	0.96	0.96	0.96	1165
2	0.78	0.63	0.70	1084
accuracy			0.86	4336
macro avg	0.86	0.84	0.84	4336
weighted avg	0.85	0.86	0.85	4336



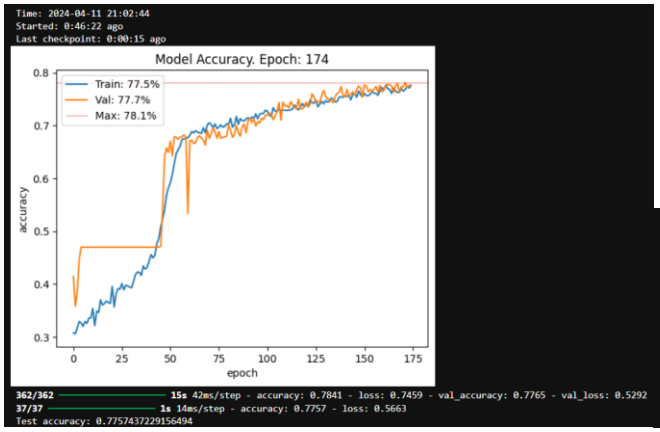
Classifier with test data:

	precision	recall	f1-score	support
0	0.82	0.86	0.84	184
1	0.94	0.95	0.95	122
2	0.74	0.69	0.71	131
accuracy			0.83	437
macro avg	0.84	0.83	0.83	437
weighted avg	0.83	0.83	0.83	437

Further Experimentation

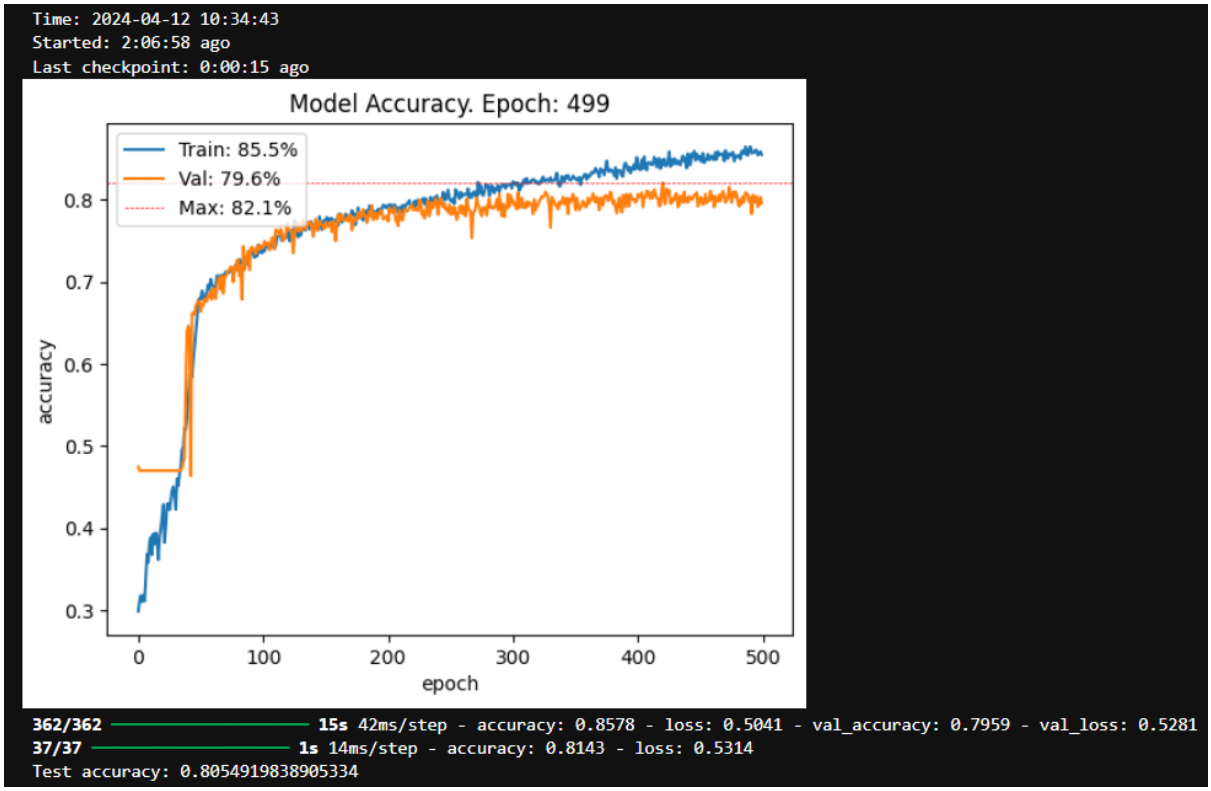
Tuning - learning rate

Tuning the learning rate is a time-consuming process. A low learning rate may produce good results, but training becomes slow. Attempting to set the learning rate to 0.001 for 175 epochs produces a steady increase in performance, while it needs at least 50 epochs to obtain a valid result. 175 epochs seem too low, as progress is steady, and there are no signs of overfitting, while results are slightly lower than in the previous step.
NOTE: for classification, use test data (in previous steps, used train data)



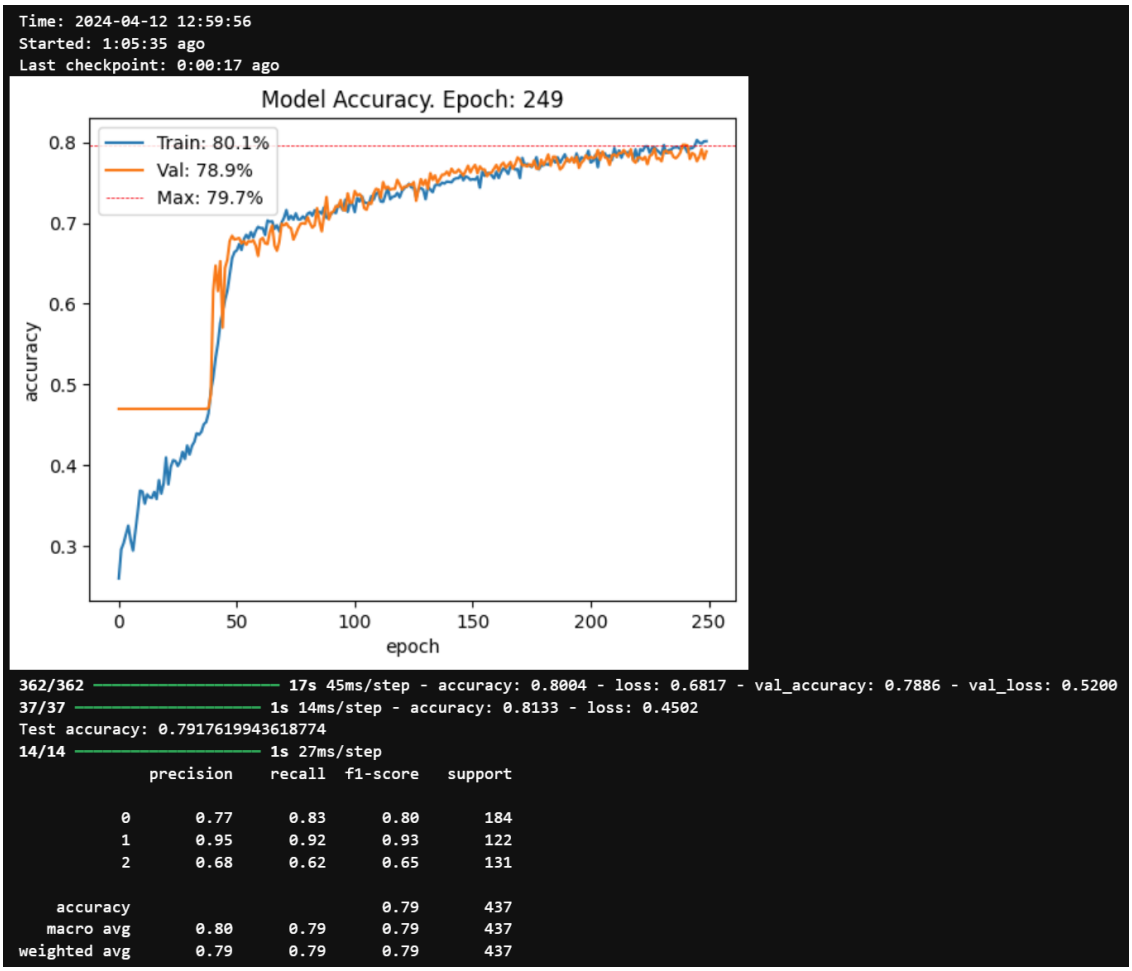
	precision	recall	f1-score	support
0	0.76	0.78	0.77	184
1	0.90	0.96	0.93	122
2	0.66	0.60	0.63	131
accuracy			0.78	437
macro avg	0.77	0.78	0.78	437
weighted avg	0.77	0.78	0.77	437

The second attempt had a 0.001 learning rate with an epoch count of 500. The model starts to overfit in around 200-250 epochs. To train the model takes around 2 hours, while the result is very similar to the step with a higher learning rate (0.01)

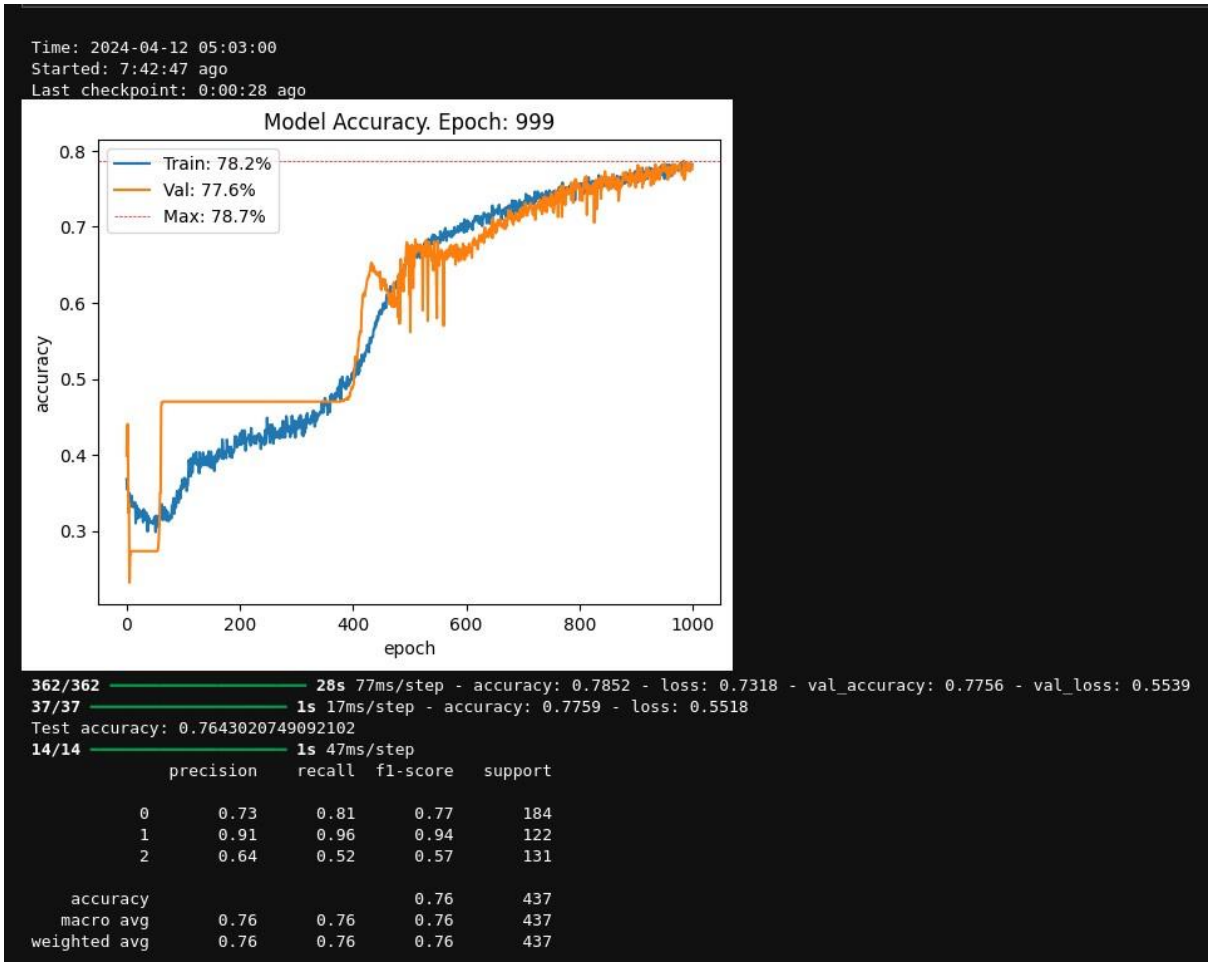


	precision	recall	f1-score	support
0	0.76	0.87	0.81	184
1	0.97	0.94	0.95	122
2	0.72	0.59	0.65	131
accuracy			0.81	437
macro avg	0.81	0.80	0.80	437
weighted avg	0.80	0.81	0.80	437

Results for running 250 epochs only seem slightly lower.



Additionally, I attempted to run the model with a very low learning rate (0.0001) for 1000 epochs. The time it takes is around 7h40min. Different devices were used (specs in Appendix B device B), which is slightly slower; therefore, time cannot be compared to results in previous steps. The model showed some signs of performance after around 450 epochs, while more stable results were produced at around 600 epochs. For this model with a 0.0001 learning rate, 1000 epochs were too little as there was no sign of overfitting, and the performance score was lower than expected, but it constantly rises.



Possible Improvements

Augmentation worked well to address the class imbalance, but it badly affected the accuracy of the model. Even more so, epochs were added, but it still didn't reach baseline performance. Adding actual data (images) might help to increase model performance.

Images that feed to a network have shapes of (128,128,3), where the number 3 indicates the colour channel. They could probably be dropped as images are grayscale. Reducing dimensionality may increase training speed.

The model is very good at distinguishing between Normal and other classes, and I wonder if creating two models instead of one that first, probably the simpler model eliminates the Normal class and a second model focusing on predicting which infection the patient has.

Appendix A (model)

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 128, 128, 3)	0
random_rotation (RandomRotation)	(None, 128, 128, 3)	0
random_translation (RandomTranslation)	(None, 128, 128, 3)	0
random_zoom (RandomZoom)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 256)	3,211,520
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 256)	65,792
dense_2 (Dense)	(None, 256)	65,792
dropout_1 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65,792
dense_4 (Dense)	(None, 256)	65,792
dropout_2 (Dropout)	(None, 256)	0
dense_5 (Dense)	(None, 256)	65,792
dense_6 (Dense)	(None, 256)	65,792
dropout_3 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 256)	65,792
dense_8 (Dense)	(None, 256)	65,792
dropout_4 (Dropout)	(None, 256)	0
dense_9 (Dense)	(None, 256)	65,792
dense_10 (Dense)	(None, 256)	65,792
dropout_5 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 256)	65,792
dense_12 (Dense)	(None, 3)	771

Appendix B (Device Spec)

Device A (main):

CPU: Intel(R) Core(TM) i7-7800X @ 3.50GHz (6C/12T)
RAM: 64GB 3600MHz
GPU: 1080Ti (not enabled for ML)
Storage: 4x NVMe RAID-0

Device B (for long runs):

CPU: AMD Ryzen 5 3600 @ 3.6 GHz (6C/12T)
RAM: 16GB 3200MHz
GPU: AMD RX 580 (not enabled for ML)
Storage: NVMe