

## **Journal Title:**

# Pneumonia Classification Problem Utilising Convolutional Neural Network and Transfer Learning

## **Abstract:**

This journal documents the development process of two convolutional neural network (CNN) models for pneumonia classification using chest X-ray images. This journal includes problem statement, data collection, preprocessing, model implementation, training, evaluation and reflections on this project.

The University of Technology Sydney  
Dohun Won  
24682318

Colab Link:

[https://colab.research.google.com/drive/1ugMBwk5YA\\_H-UN1Tu0-zLMTZyMCtH4lr?usp=sharing](https://colab.research.google.com/drive/1ugMBwk5YA_H-UN1Tu0-zLMTZyMCtH4lr?usp=sharing)

# Problem statement

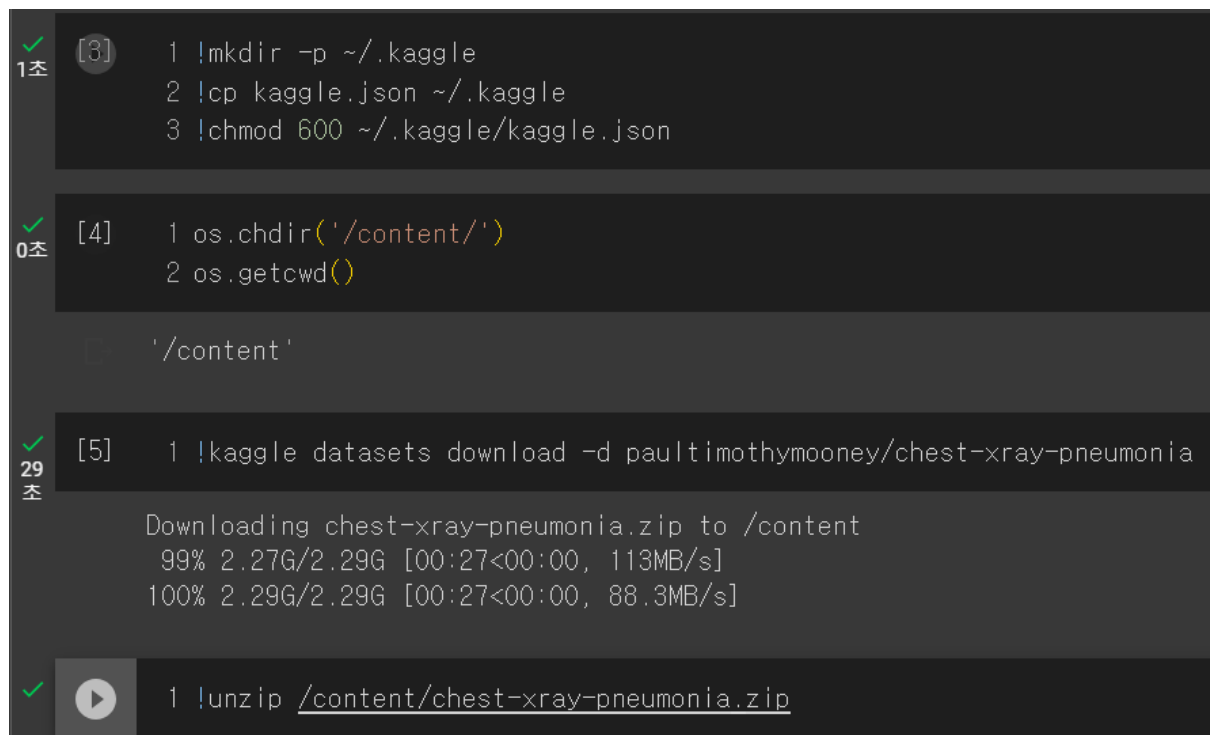
## Problem:

Pneumonia, a deadly respiratory infection claiming the lives of over 2,000 individuals annually, is a severe disease that ranks among Australia's top 20 causes of death (Australian Bureau of Statistics, 2022). However, the huge number of doctors is expected to decrease due to retirement in Australia (Health Workforce Australia, 2014).

## Objective:

My goal is to implement a machine-learning model that can diagnose pneumonia in the early stages using chest images of existing pneumonia patients and normal patients. This model could lead to an increase in the probability that medical professionals will be able to diagnose pneumonia cases in patients early and the accuracy of pneumonia diagnosis.

## Data Collection



```
[3] 1 !mkdir -p ~/.kaggle
    2 !cp kaggle.json ~/.kaggle
    3 !chmod 600 ~/.kaggle/kaggle.json

[4] 1 os.chdir('/content/')
    2 os.getcwd()

'/content'

[5] 1 !kaggle datasets download -d paultimothymooney/chest-xray-pneumonia

Downloading chest-xray-pneumonia.zip to /content
99% 2.27G/2.29G [00:27<00:00, 113MB/s]
100% 2.29G/2.29G [00:27<00:00, 88.3MB/s]

1 !unzip /content/chest-xray-pneumonia.zip
```

Figure 1, Download chest-xray-pneumonia data from Kaggle

## Data source:

The chest x-ray image data is sourced from Kaggle. Kaggle is a platform that organises many competitions to solve data science problems and provides a

wide range of data. The dataset includes images from both pneumonia-positive cases and normal cases.

## Data Preprocessing

**The structure of the folder where the data is stored:**

chest\_xray

- train
  - NORMAL
  - PNEUMONIA
- test
  - NORMAL
  - PNEUMONIA
- val
  - NORMAL
  - PNEUMONIA

```
[34] 1 train_data = ImageDataGenerator(rescale=1./255, validation_split=0.2)
      2
      3 train_generator = train_data.flow_from_directory('/content/chest_xray/train',
      4                                                    target_size=(75, 75),
      5                                                    batch_size=32,
      6                                                    class_mode='binary',
      7                                                    subset='training')
```

Found 4173 images belonging to 2 classes.

*Figure 2, Normalising, Splitting and Loading for training data*

Before loading the data, i use *ImageDetaGenerator* to set the parameter *rescale* for normalisation and *validation\_split* to split the data into training and validation data. The reason why image data is not used in the val folder is that the number of image data is 16, which is too small to validate my model. After that, i use *flow\_from\_directory* to set the target and batch size. I will mention later why i set *target\_size* to 75, 75. Moreover, the *class\_mode* is set to binary because of solving a binary classification task while the subset is training which specifies this is training data.

```
[ ] 1 class_indices = train_generator.class_indices
    2
    3 # Get the class labels and their respective counts
    4 class_labels = list(class_indices.keys())
    5 class_counts = train_generator.classes.sum(axis=0)
    6
    7 class_labels[0], 4173 - class_counts, class_labels[1], class_counts

('NORMAL', 1073, 'PNEUMONIA', 3100)
```

*Figure 3, The number of each data*

The training data has 1073 images of a normal chest and 3100 images of a pneumonia-positive chest.

```
[ ] 1 val_generator = train_data.flow_from_directory('/content/chest_xray/train',
    2                                             target_size=(75, 75),
    3                                             batch_size=32,
    4                                             class_mode='binary',
    5                                             subset='validation')

Found 1043 images belonging to 2 classes.

[ ] 1 class_indices = val_generator.class_indices
    2
    3 # Get the class labels and their respective counts
    4 class_labels = list(class_indices.keys())
    5 class_counts = val_generator.classes.sum(axis=0)
    6
    7 class_labels[0], 1043 - class_counts, class_labels[1], class_counts

('NORMAL', 268, 'PNEUMONIA', 775)

[ ] 1 test_data = ImageDataGenerator(rescale=1./255)
    2 test_generator = test_data.flow_from_directory('/content/chest_xray/test',
    3                                             target_size=(75, 75),
    4                                             batch_size=32,
    5                                             class_mode='binary',
    6                                             shuffle=False)

Found 624 images belonging to 2 classes.

[ ] 1 class_indices = test_generator.class_indices
    2
    3 # Get the class labels and their respective counts
    4 class_labels = list(class_indices.keys())
    5 class_counts = test_generator.classes.sum(axis=0)
    6
    7 class_labels[0], 624 - class_counts, class_labels[1], class_counts

('NORMAL', 234, 'PNEUMONIA', 390)
```

*Figure 4, Performing the same logic as before*

I perform the same logic for normalising and loading data as before. The validation data has 268 normal chest images and 775 pneumonia-positive images whereas the test data includes 234 and 390 of normal and pneumonia-positive, respectively.

## Model Implementation

### Model Architecture:

```
[ ] 1 model = Sequential()  
    2  
    3 model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(75, 75, 3)))  
    4 model.add(MaxPooling2D(2, 2))  
    5  
    6 model.add(Conv2D(64, (3, 3), activation='relu'))  
    7 model.add(MaxPooling2D(2, 2))  
    8  
    9 model.add(Conv2D(128, (3, 3), activation='relu'))  
   10 model.add(MaxPooling2D(2, 2))  
   11  
   12 model.add(Flatten())  
   13 model.add(Dense(256, activation='relu'))  
   14 model.add(Dense(128, activation='relu'))  
   15 model.add(Dense(32, activation='relu'))  
   16 model.add(Dense(1, activation='sigmoid'))  
   17  
   18 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  
   19
```

*Figure 5, machine-learning model architecture*

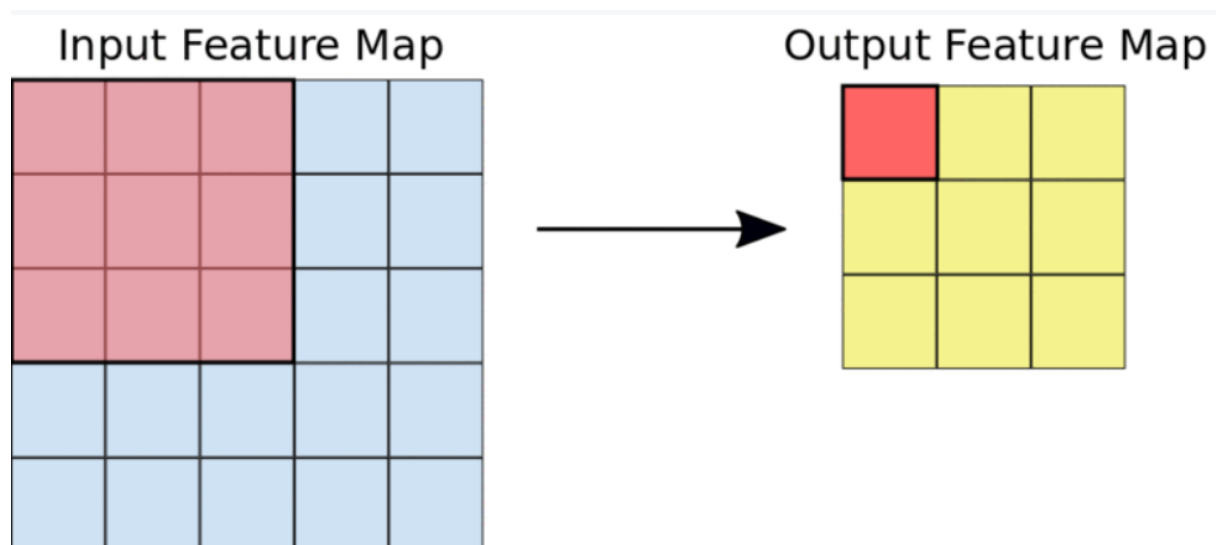


Figure 6, A 3x3 convolution of depth 1 performed over a 5x5 input feature map, also of depth 1. There are nine possible 3x3 locations to extract tiles from the 5x5 feature map, so this convolution produces a 3x3 output feature map.

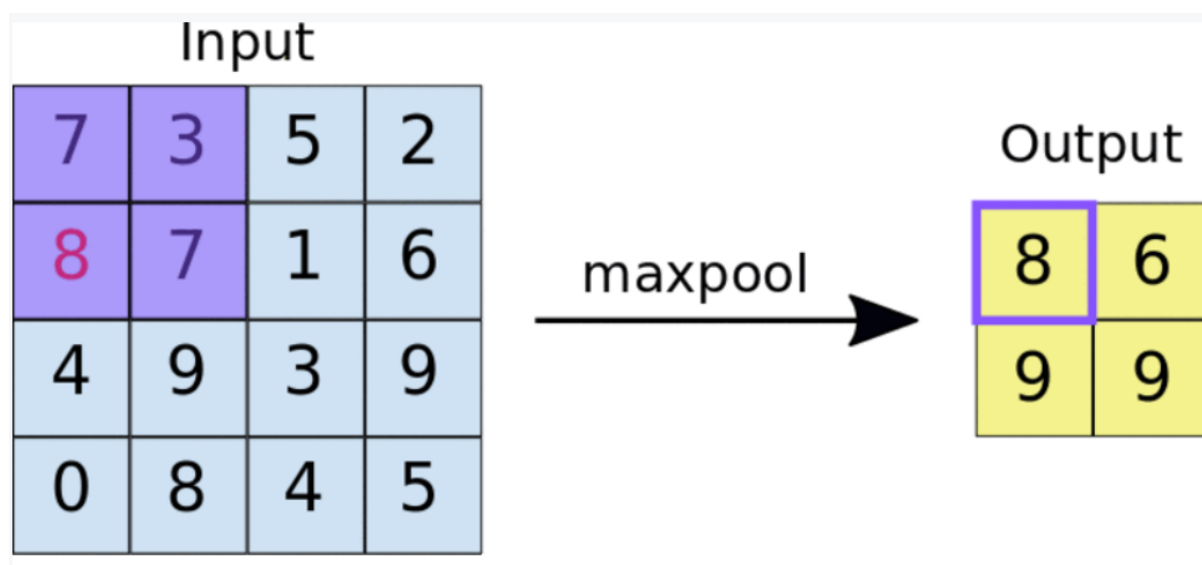


Figure 7, Left: Max pooling performed over a 4x4 feature map with a 2x2 filter and stride of 2. Right: the output of the max pooling operation. Note the resulting feature map is now 2x2, preserving only the maximum values from each tile.

## Conv2D

First, *Conv2D* functions to extract the features of the input image (See Figure 6). The first parameter to enter *Conv2D* is a number that determines how many features to extract and the second tuple parameter represents the size of the filter.

## Relu

The activation function receives the values from the previous layer and outputs the results from the activation function to the next layer. *Relu* is a slope of 0 if the input value is less than 0, and the slope is x if the input value is greater than 0. The reason why *relu* is used is that compared to the *sigmoid* function or *tanh* functions, the learning speed is much faster and the computational time is not large. However, when the input value is less than zero in the *relu* function, the slope becomes zero and the corresponding neuron can die called *dying relu*.

## Input Shape

The *input shape* indicates that the first two numbers represent the width and height of the image, and the last number represents the number of channels in the image colour.

### ***maxPooling2D***

*maxPooling2D* is a code that subsamples feature maps, which are the calculation results of the convolution layer. *maxPooling2D* receives a tuple with the first parameter and extracts only the largest value among existing feature maps as the size of the tuple received (See Figure 7).

### ***Flatten***

This helps to convert 2D arrays from pooled feature maps to a single linear vector.

### ***Dense***

The Dense layer takes a flattened characteristic vector as input and each neuron learns complex data patterns by leveraging the characteristics of the previous layer. However, the first parameter of the last dense layer is one because this is a binary classification problem, so the answer is yes or no. In addition, the sigmoid function makes the output range from 0 to 1, which is the probability of the pneumonia-positive.

### ***Adam***

The Adam optimizer is a widely used optimization algorithm in deep learning. It helps to improve the speed and accuracy of neural network training by adaptively adjusting the learning rates for each parameter.

### ***Binary\_crossentropy***

*Binary\_crossentropy* is a standard loss function used for binary classification tasks. It quantifies the difference between two probability distributions such as the true distribution and the estimated distribution from the model.

## **Model training**

```

1 from tensorflow.keras.callbacks import TensorBoard, EarlyStopping
2 import time
3
4 tensorboard = TensorBoard(log_dir = 'logs/{}'.format('first model' + str(int(time.time()))))
5 es = EarlyStopping(monitor='val_accuracy', patience=5, mode='max')
6
7 history = model.fit(train_generator, validation_data=(val_generator), epochs=20, callbacks=[tensorboard, es])

```

Figure 8, Training the model

In this training code, i use TensorBoard to track the changes in the accuracy and loss of the model while the EarlyStopping is used to stop the training provided that the validation accuracy is not enhanced during the set epoch. Even though i set the epochs to 20 in the *fit* parameter, the total number of epochs for training is 16 and the training and validation accuracy are 0.9990 and 0.9578, respectively.

## Model Evaluation

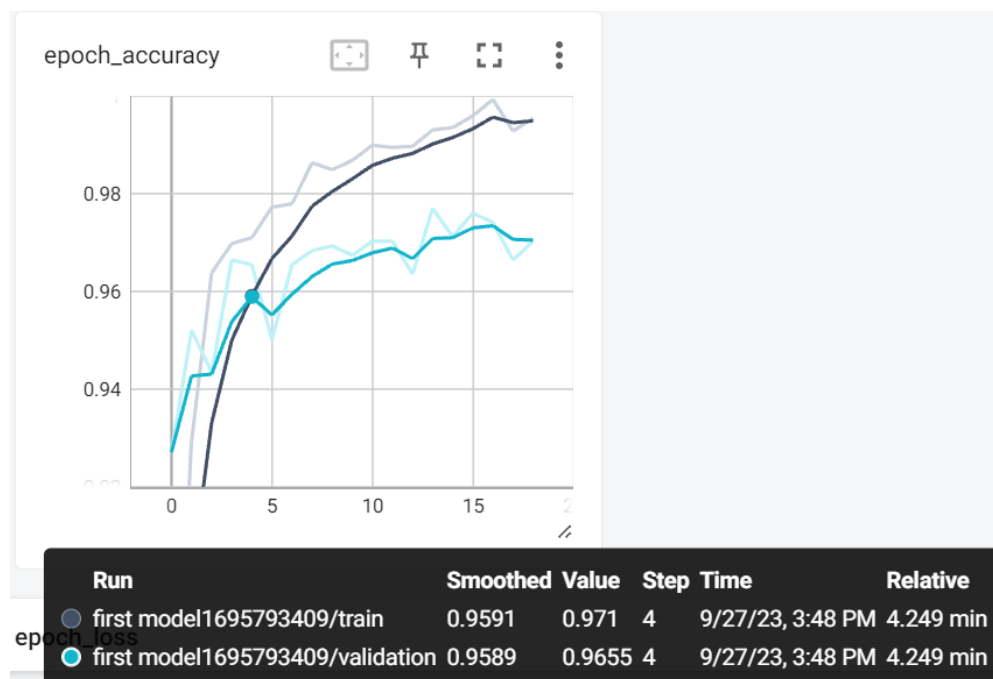


Figure 9, The changes in the accuracy of the model

This graph represents the changes in the accuracy of the model. When the number of epochs increases, the training accuracy tends to rise rapidly. In contrast, the validation accuracy increases remarkably until the fourth epoch and there has been no significant change since then. This could be suspected of overfitting after the third epoch.



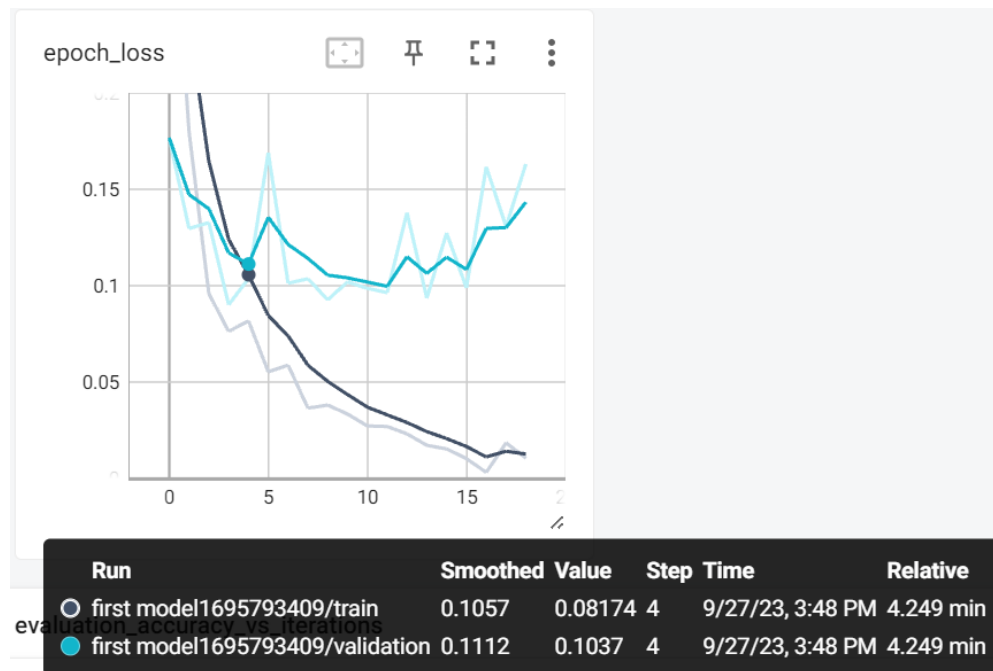


Figure 10, The changes in the loss of the model

This graph illustrates the changes in the loss of the model. Similar to accuracy, training loss and validation loss show a similar drop until the fourth epoch, but after that, only training loss decreases significantly whereas validation loss fluctuates slightly.

```
[24] 1 predictions = model.predict(test_generator)
```

20/20 [=====] - 5s 235ms/step

```
[25] 1 from sklearn.metrics import classification_report
2
3 predictions_labels = (predictions > 0.5).astype(int)
4 true_labels = test_generator.classes
5
6 report = classification_report(true_labels, predictions_labels, target_names=['NORMAL', 'PNEUMONIA'])
7 print(report)
```

	precision	recall	f1-score	support
NORMAL	0.96	0.36	0.53	234
PNEUMONIA	0.72	0.99	0.83	390
accuracy			0.75	624
macro avg	0.84	0.68	0.68	624
weighted avg	0.81	0.75	0.72	624

Figure 11, The classification report of the model

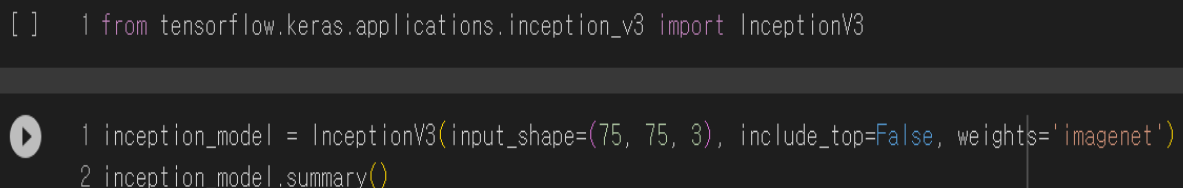
The output of the *predict* method is the probabilities but the *classification\_report* requires the answer label and predicted label. Therefore, i change the

probabilities to label following the condition. If the probability is greater than 0.5, it is one but if the probability is smaller than 0.51, it is zero.

## Analysis

Even though the accuracy of the training and validation data was high, this model tends to over-diagnose pneumonia when classifying the image for the testing data. The evidence to support that is the recall of pneumonia is quite high while the precision for normal is high. In addition, the recall of the normal is low and the precision for pneumonia is low.

## Transfer Learning Model(InceptionV3)



```
[ ] 1 from tensorflow.keras.applications.inception_v3 import InceptionV3

1 inception_model = InceptionV3(input_shape=(75, 75, 3), include_top=False, weights='imagenet')
2 inception_model.summary()
```

*Figure 12, Load pre-trained model from the TensorFlow library*

Transfer learning refers to using a pre-trained model designed to solve a complex problem for solving the other problem. In this part, i will use the *InceptionV3* model from the *Tensorflow* library. The reason that i set the image size to (75, 75) is that the official document for *InceptionV3* mentions that the height and weight should not be smaller than 75 (TensorFlow, 2023). Furthermore, `include_top` should be false because *InceptionV3* is not designed for binary classification. This helps me to perform fine-tuning. Fine-tuning means people can customise the output layer fitting to their data problem. Lastly, i will use the weights generated when InceptionV3 was trained.

## Fine-tuning

```

1 for i in inception_model.layers:
2     i.trainable = False

1 additional_layer_1 = tf.keras.layers.Flatten()(inception_model.get_layer('mixed10').output)
2 additional_layer_2 = tf.keras.layers.Dense(512, activation='relu')(additional_layer_1)
3 last_layer = tf.keras.layers.Dense(1, activation='sigmoid')(additional_layer_2)
4
5 pre_model = tf.keras.Model(inception_model.input, last_layer)
6
7 pre_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
8
9 tensorboard = TensorBoard(log_dir = 'pre_model_logs/{}'.format('first pre_model' + str(int(time.time()))))
10 es = EarlyStopping(monitor='val_accuracy', patience=5, mode='max')
11
12 history_pre = pre_model.fit(train_generator, validation_data=(val_generator), epochs=20, callbacks=[tensorboard, es])
13
14

```

*Figure 13, Fine-tuning the InceptionV3 model*

First of all, i will not use new weights for the customised InceptionV3 model so i set the whole previous layers not to be trainable. Furthermore, I add some layers but the crucial point is the last layer. My data problem is to predict whether pneumonia-positive or not so the output should be the probability of pneumonia. The other settings are the same as before.

## Model evaluation

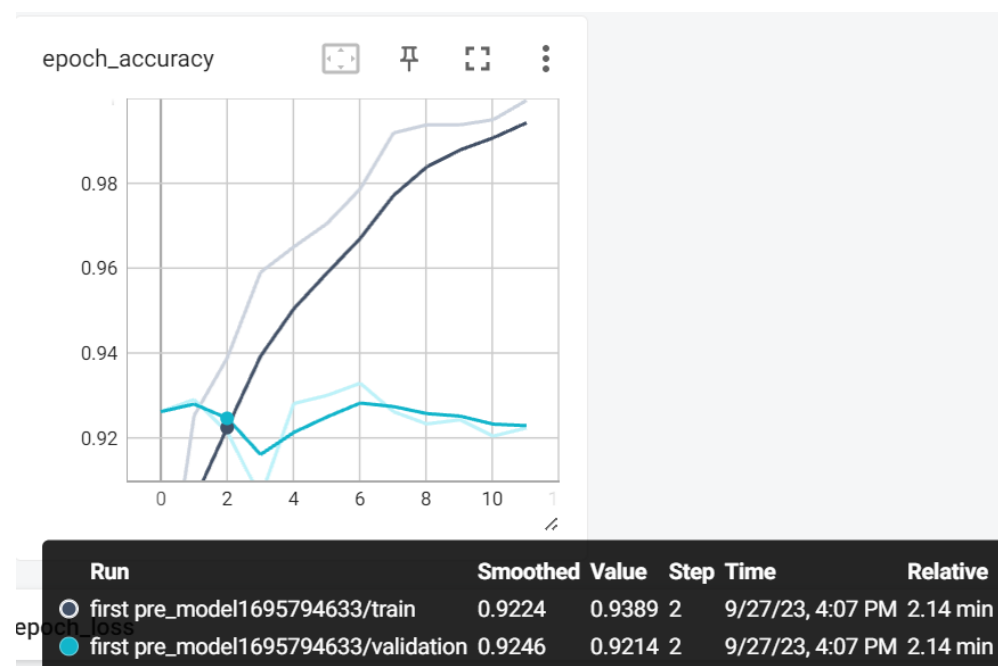


Figure 14, The changes in the accuracy of the customised InceptionV3 model

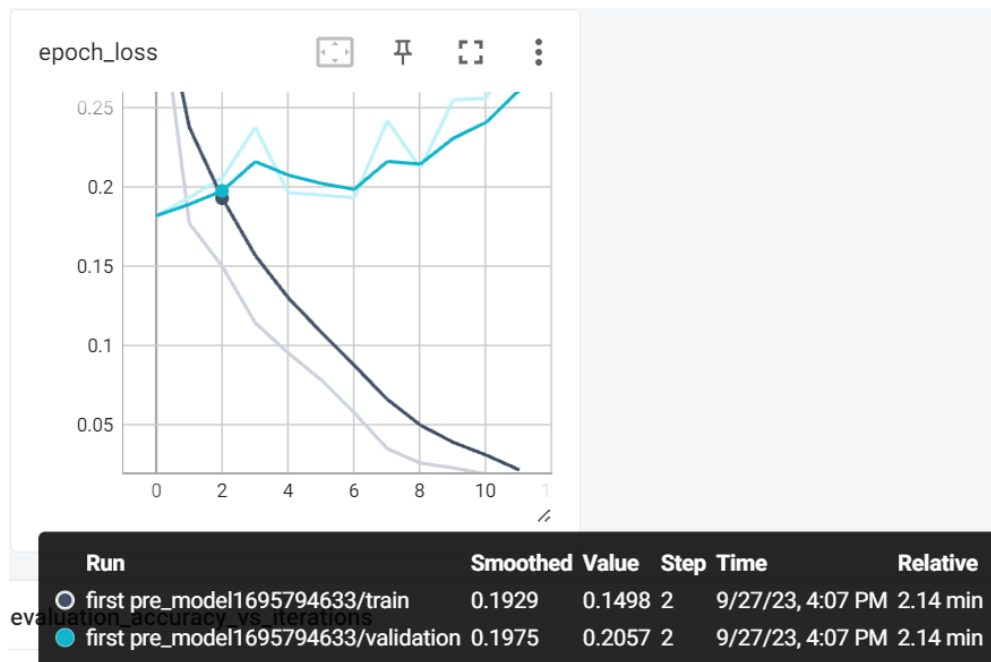


Figure 15, The changes in the loss of the customised InceptionV3 model

The accuracy of the validation data decreases after the second epoch whereas the accuracy for training data increases gradually every epoch (See Figure 14). It can be hypothesised that the pre-trained model does not require a large number of epochs. Many epochs could lead to overfitting.

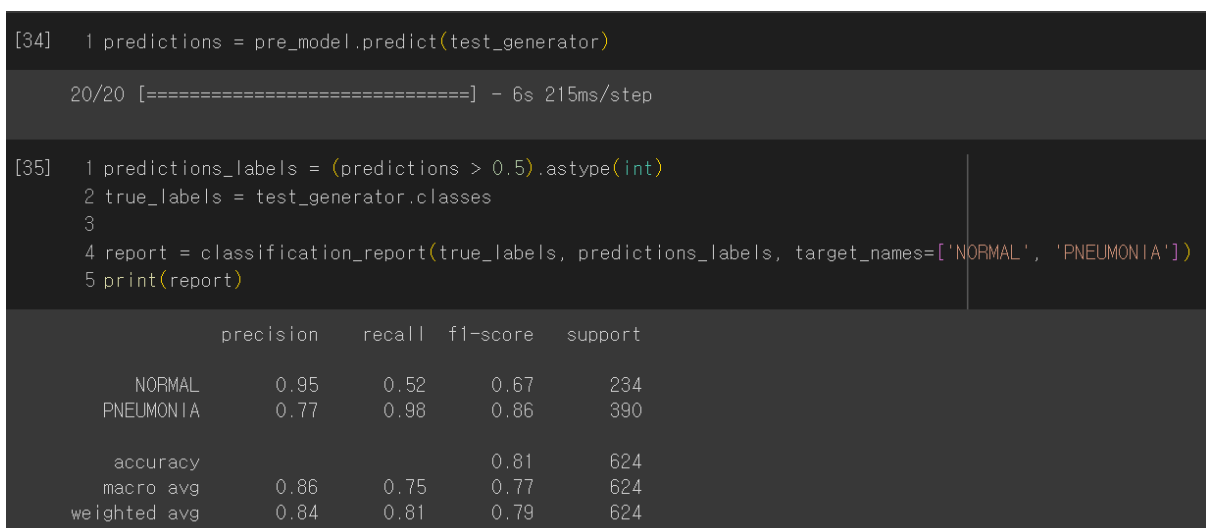


Figure 16, The classification report of the customised InceptionV3 model

The customised InceptionV2 model has a similar tendency to the previous model because the *classification\_report* shows that the precision for normal

and the recall of pneumonia are high. However, it has better performance in terms of the performance indicator.

## Model save

```
[38] 1 model.save('/content/drive/MyDrive/Machine Learning Lecture/Pneumonia Classification Model.h5')  
     2 pre_model.save('/content/drive/MyDrive/Machine Learning Lecture/Pneumonia Classification Model(Transfer learning).h5')
```

My models have been saved to Google Drive.

## Conclusion

The model that i designed showed a tendency to overfit from the fourth epoch but the pre-trained model showed a tendency to overfit from the second epoch. The pre-trained model has the advantage of shortening the model training time because the pre-trained model does not require much training compared to the model that i designed. Furthermore, the customized model showed higher performance.

## References

Australian Bureau of Statistics. (2022, October 19). *Causes of Death, Australia, 2021*

| *Australian Bureau of Statistics*. [Www.abs.gov.au](http://www.abs.gov.au).

<https://www.abs.gov.au/statistics/health/causes-death/causes-death-australia/2021>

Google. (2022a). A 3x3 convolution of depth 1 performed over a 5x5 input feature map, also of depth 1. There are nine possible 3x3 locations to extract tiles from the 5x5 feature map, so this convolution produces a 3x3 output feature map. In *Google for Developers*.

<https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?ref=machinelearningnuggets.com>

Google. (2022b). Left: Max pooling performed over a 4x4 feature map with a 2x2 filter and stride of 2. Right: the output of the max pooling operation. Note the resulting feature map is now 2x2, preserving only the maximum values from each tile. In *Google for Developers*.

<https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks?ref=machinelearningnuggets.com>

Health Workforce Australia. (2014). Australia's Future Health Workforce – Doctors. In *Australian Government Department of Health and Aged Care*.

<https://www.health.gov.au/sites/default/files/documents/2021/03/doctors-australia-s-future-health-workforce-report.pdf>

TensorFlow. (2023, August 1). *tf.keras.applications.inception\_v3.InceptionV3 | TensorFlow v2.13.0*. TensorFlow.

[https://www.tensorflow.org/api\\_docs/python/tf/keras/applications/inception\\_v3/InceptionV3](https://www.tensorflow.org/api_docs/python/tf/keras/applications/inception_v3/InceptionV3)