PREDICTING COVID-19 STATUS FROM CHEST X-RAY IMAGES

IFN 646 – BIOMEDICAL DATA SCIENCES

PREPARED BY

Nhung Nguyen – 10655549

An Ngo – 10324399

Long Ngan Nguyen – 9786198

Date: 10/11/2020

Lecturer: Dr. Dimitri Perrin

INTRODUCTION

In December 2019, a new virus known as "2019 novel corona-virus" or "Severe Acute Respiratory Syndrome Coronavirus 2 (SARS-CoV-2)" was firstly identified from Wuhan, China and its infectious disease is called COVID-19. Until 14th October 2020, more than 37.7 billion confirmed cases have been results in 235 countries, areas and territories according to announcement of WHO (2020). Currently, early symptom diagnosing is highly importance to self-isolate the suspected people and decrease the risk of spreading to public community's health due to lack of specific treatment or vaccine for this virus.

In many countries, to detect suspected COVID-19 individuals, governments have applied reverse transcriptase–polymerase chain reaction (RT-PCR) or collecting pharyngeal swabs or blood specimens to detect people who is suspected as positive with COVID-19 (Wang et al., 2020). According to Department of Heath of Australia Government (2020), it may take 1 or 2 days to get the results of PCR and while people is waiting for their testing result, they need to perform self-isolation at home. Compared to chest radiography, X-Ray imaging is a method which is easy to apply and fast diagnosis for pneumonia. In the research of Kanne et al. (2020), the author reported that radiography images are able to visualize the correlation with COVID-19. The symptoms of COVID-19 are reported to be as following, ground-glass (57%) and mixed attenuation (29%) (Kong and Agarwal as cited in Minaee et al., 2020), the pulmonary vessels are edged by ground glass pattern make it becomes more difficult to appreciate visually reported in the research of Feng et al. (2016). In addition, Asymmetric patchy or diffuse airspace opacities are also reported for COVID-19 (Rodrigues et al., 2020). These abnormalities are only be observed by expert radiologists. However, the ratio between trained radiologists and suspected cases is imbalance and with the number of suspected cases continues to increase, an automatic method for identification of such subtle abnormalities to assist with diagnosis is crucial. Therefore, Artificial Intelligence (AI) solutions are promising methods which are significant for solving such problems.

The pandemic that caused by COVID-19 has raised an alarm to the way people react to diseases and viruses. Although machine learning was applied to support medical image classification, this traditional approach was not powerful enough to win against the race of detecting COVID-19 amid the escalation of the epidemic. Thus, instead of following a two-step procedure including feature extraction and recognition, we use an end-to-end deep learning framework. The approach will directly predict the COVID-19 based on raw X-Ray images of lungs without requiring feature extraction. Deep learning model, or more specifically Convolutional Neural Networks (CNN) have been proved that outstrip traditional AI approaches in the field of computer vision in recent years, and have been applied widely to solve various problems, such as classification, segmentation, face recognition and image enhancement (Bhosle et al., 2018; Li et al., 2018).

Additionally, we adopt the approach of training traditional Convolutional Neural Network on COVID-19 dataset of Wang et al. (2020), and evaluate the performance of models on predicting COVID-19 detection. However, since the medical images of COVID-19 are not widely published, there is a limited number of available publicity images. We collected and prepared a dataset of around more than 2,000 images (using images from two datasets). In addition, we performed a detail experimental analysis evaluating the performance of the model in different approaches. Thus, in order to improve and measure the performance of COVID-19 detection experiment, we apply 2 following strategies*:*

- We apply different techniques to transform the images in the effort of data augmentation, such as flipping and small rotation, to increase the number of samples.
- We also calculate the confidence interval of the performance metrics on the respective models. In addition, we provide Area Under the Curve (AUC) to summarize the performance of models in our report.

The data availability including codes and dataset are now published on Git (https://github.com/ifn-646/x_ray_project)

APPROACH

The COVID-19 X-ray dataset consists of chest X-rays from two datasets:

1. **Covid-19 Radiography Dataset (dataset A)** which contains a total of 219 Covid-19 Positive images and 1341 Covid-19 Negative test images are separated in two labelled folders.

2. Another dataset is the **Covid-Chest Xray-Dataset (dataset B)**, which was recently published and is comprised of a set of images from published sources regarding COVID-19 topics collected by https://github.com/ieee8023/covid-chestxray-dataset , Cohen et al. (2020). This dataset is compiled of a blend of CT images along with chest X-rays (Figure 1.1).

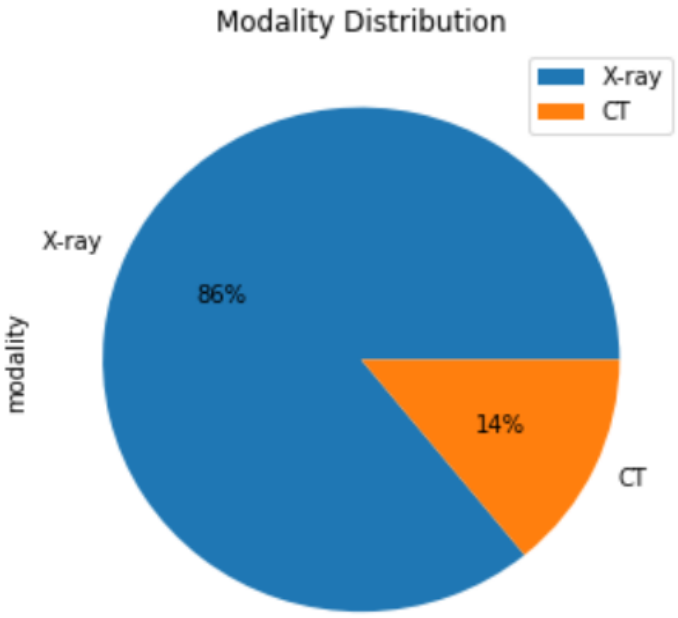


Figure 1.1. X-ray and CT Scan Proportion

This pool of data is unceasingly updated, and meta-data such as age, findings, sex, survival and hospitalization status are also recorded. Part of the COVID-19 images used for this project contains the images that derived from this dataset. We have only kept the PA view images for COVID-19 prediction (Figure 1.2).

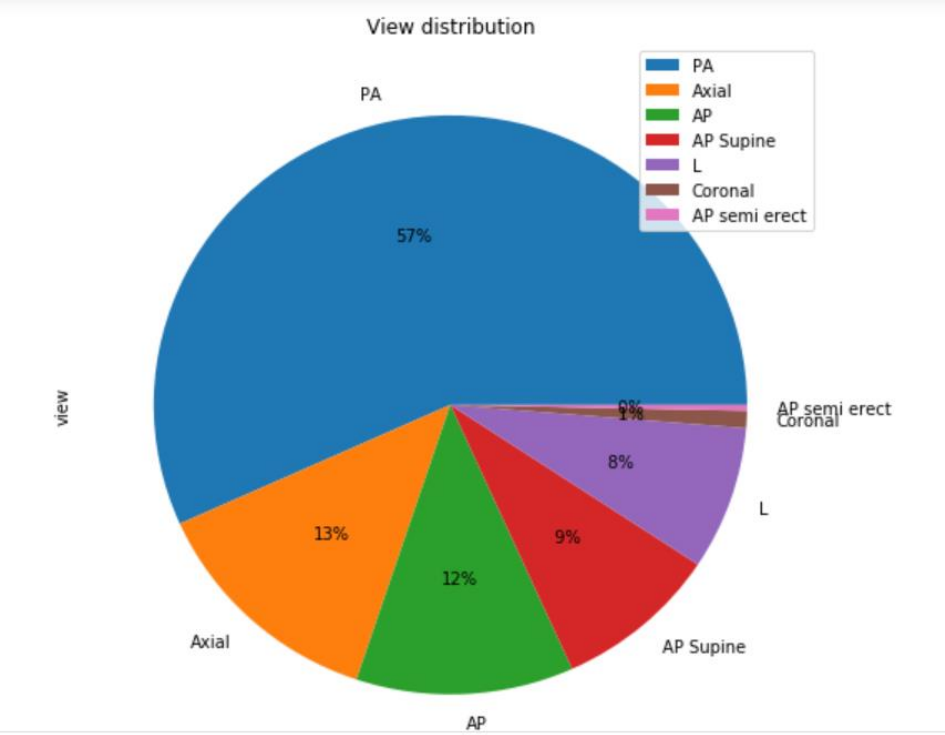


Figure 1.2. Views of Recorded Images

The resolution should be mentioned regarding this dataset, as it varies widely. The lower-resolution COVID-19 images are below 400 x 400, while some of the higher ones are exceeding 1900 x 1400. The models produce accurate results regardless of this disparity in image quality. Although exceptional image quality should be strived for, it is a focus in the machine learning realm to create systems that are able to operate well regardless of differing image qualities. The dynamic ranges also differ as they are images collected from various providers. These images are indeed normalized to decrease model confusion.

PROPOSED FRAMEWORK

This project is to classify whether the given image belongs to the positive or negative class, Convolutional Neural Network with Tensorflow Keras is the main method that we applied.

1. Model:

According to Lecun et al. (1998), Convolutional neural network is a class of deep learning, this learning algorithm builds a model with multi-layer neural networks which can learn major and relevant feature for images to deal diversity piece of works such as, detection, segmentation and classification.

In general, after receiving data from the post-processing step, these data will be passed through 4 layers of the CNN machine learning model with the first layer of data compression and the next 3 layers of convolution and this model used activation fuctions as “relu”. Finally the result is a possibility that could be close to the positive or negative class.

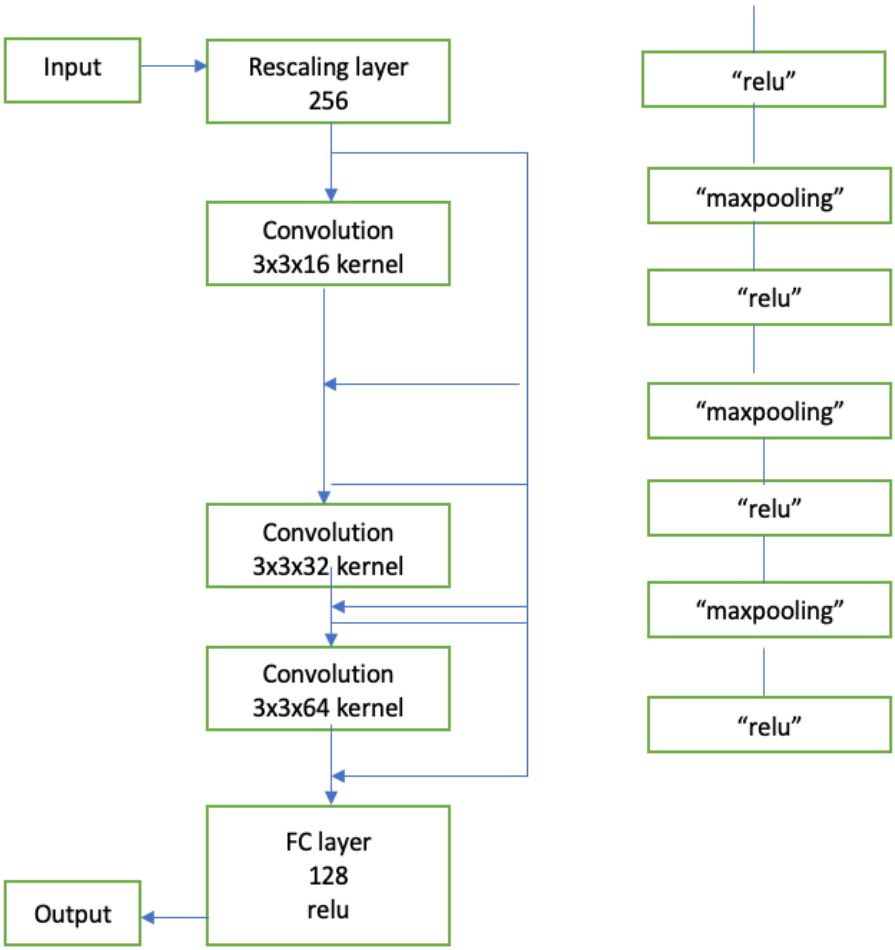


Figure 2.1 Convolutional neural network model for Covid-19 prediction

Figures 2.2 below shows the process in detail. All of image crossing through Rescalling layer will be converted to image with 256 pixels.

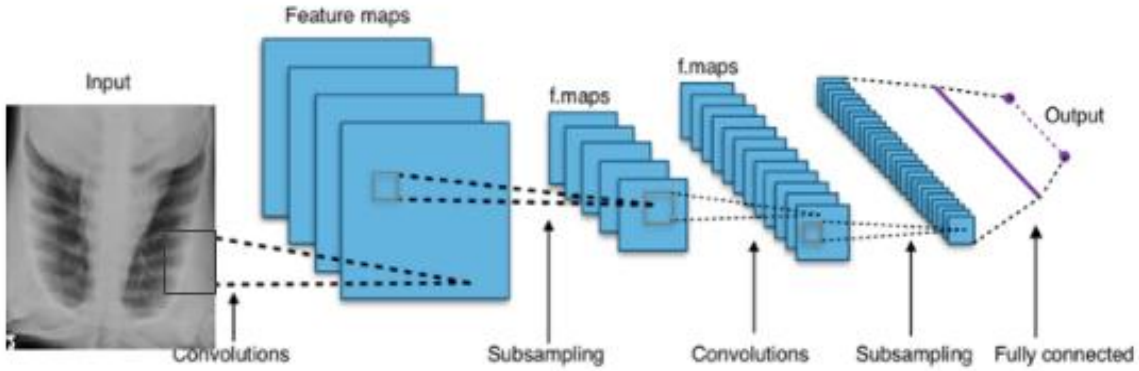


Figure 2.2 Model convolutional layer of chest X-ray images.

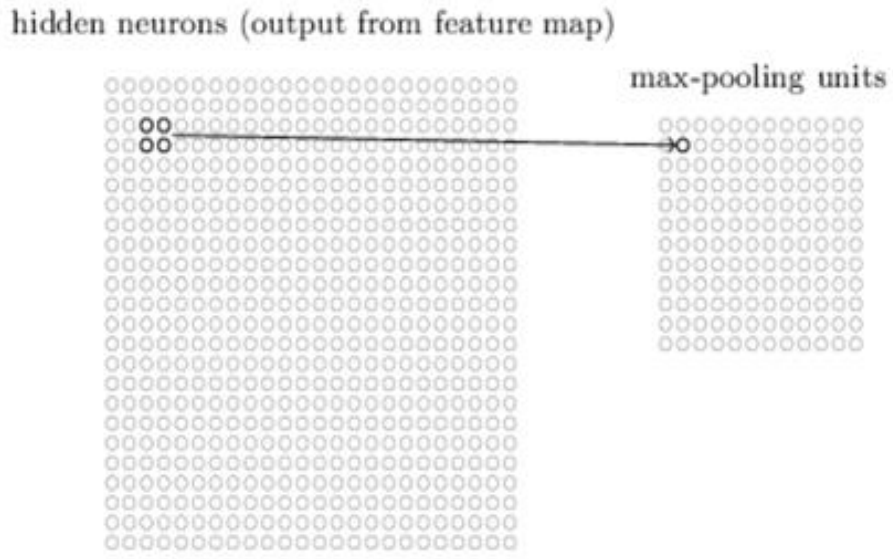


Figure 2.3. Max-pooling layer

Maxpooling layer bases on the feature map to calculate the largest, maximum or value in each patch of each feature map.

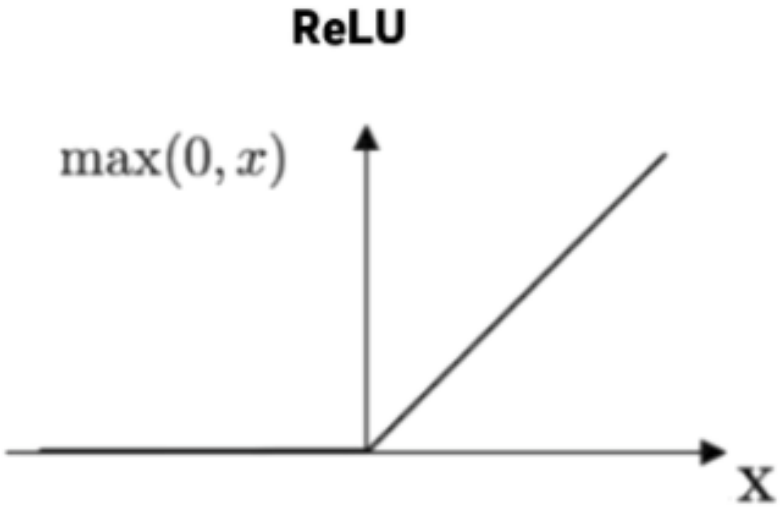


Figure 2.4. Rectified Linear Unit (ReLU)

Finally, “relu” the activation function is a non-linear function will calculate on the Maxpooling layer’s results whether it is positive or negative.

EXPERIMENTS & RESULTS

We have approached in 2 ways and we have trained each approach for 10 epochs. We apply all images into neural network with their size down sampled to 180x180.

a. Data Split on dataset A

In this approach, we have chosen only dataset A for training and testing. Figure 3.1 shows the first approach which splits 80% dataset for training data and the 20% for test with validation_split by 0.2. This figure also indicates the results of model following the data split.

```
In [16]: train_ds = tf.keras.preprocessing.image_dataset_from_directory(
           data_dir,
           validation_split=0.2,
           subset="training",
           seed=123,
           image_size=(IMG_HEIGHT, IMG_WIDTH),
           batch_size=BATCH_SIZE)

         Found 1560 files belonging to 2 classes.
         Using 1248 files for training.

In [17]: val_ds = tf.keras.preprocessing.image_dataset_from_directory(
           data_dir,
           validation_split=0.2,
           subset="validation",
           seed=123,
           image_size=(IMG_HEIGHT, IMG_WIDTH),
           batch_size=BATCH_SIZE)

         Found 1560 files belonging to 2 classes.
         Using 312 files for validation.

In [26]: epochs=10
         history = model.fit(
           train_ds,
           validation_data=val_ds,
           epochs=epochs
         )

         Epoch 1/10
         39/39 [==============================] - 25s 638ms/step - loss: 0.1879 - accuracy: 0.9046 - val_loss: 0.0957 - val
         _accuracy: 0.9679
         Epoch 2/10
         39/39 [==============================] - 20s 506ms/step - loss: 0.0655 - accuracy: 0.9736 - val_loss: 0.0527 - val
         _accuracy: 0.9776
         Epoch 3/10
         39/39 [==============================] - 21s 528ms/step - loss: 0.0289 - accuracy: 0.9872 - val_loss: 0.0488 - val
         _accuracy: 0.9776
         Epoch 4/10
         39/39 [==============================] - 20s 518ms/step - loss: 0.0152 - accuracy: 0.9944 - val_loss: 0.0698 - val
         _accuracy: 0.9808
         Epoch 5/10
         39/39 [==============================] - 20s 505ms/step - loss: 0.0195 - accuracy: 0.9904 - val_loss: 0.0725 - val
         _accuracy: 0.9808
         Epoch 6/10
         39/39 [==============================] - 21s 530ms/step - loss: 0.0161 - accuracy: 0.9928 - val_loss: 0.0935 - val
         _accuracy: 0.9840
         Epoch 7/10
         39/39 [==============================] - 20s 521ms/step - loss: 0.0063 - accuracy: 0.9976 - val_loss: 0.0426 - val
         _accuracy: 0.9872
         Epoch 8/10
         39/39 [==============================] - 21s 531ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.0768 - val
         _accuracy: 0.9840
         Epoch 9/10
         39/39 [==============================] - 21s 534ms/step - loss: 4.6038e-04 - accuracy: 1.0000 - val_loss: 0.0512 -
         val_accuracy: 0.9904
         Epoch 10/10
         39/39 [==============================] - 21s 546ms/step - loss: 2.4353e-04 - accuracy: 1.0000 - val_loss: 0.0551 -
         val_accuracy: 0.9872
```

Figure 3.1. 80-20 Split and results of 10 epochs

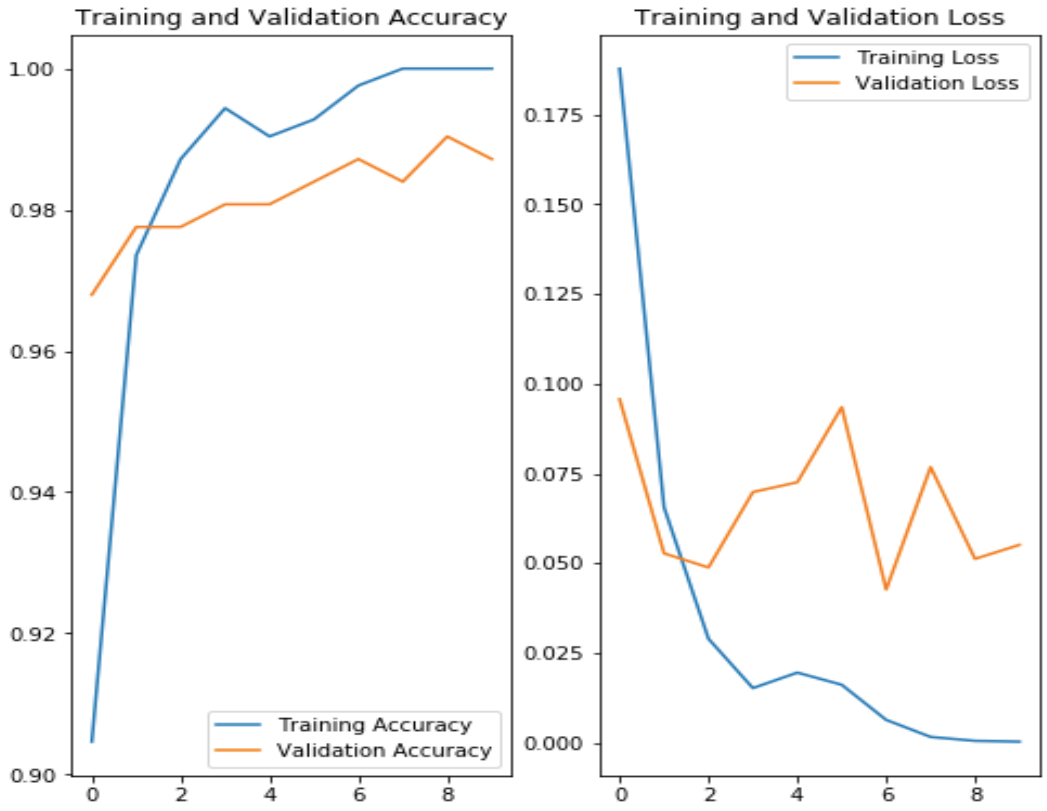


Figure 3.2. Training and Validation Accuracy/Loss

The left graph in figure 3.2 illustrates that the model is not overfitting. However, it also shows Train and Validation accuracy are doubted high, targeting to 100%, which is a thing we are suspected. We have recognised that because we stored all dataset in one folder, then split it through the code which leads to the risk that model has gone through and memorized all of the pictures in the dataset after it ran about 10 epochs.

b. Applying both A and B datasets into the model

Another approach was chosen to avoid the issue addressed above. We set Train dataset on dataset A and Test dataset on dataset B.

```
In [38]: train_ds = tf.keras.preprocessing.image_dataset_from_directory(
           data_dir,
           seed=123,
           image_size=(IMG_HEIGHT, IMG_WIDTH),
           batch_size=BATCH_SIZE)

         Found 1560 files belonging to 2 classes.

In [50]: val_ds = tf.keras.preprocessing.image_dataset_from_directory(
           test_dir,
           seed=123,
           image_size=(IMG_HEIGHT, IMG_WIDTH),
           batch_size=BATCH_SIZE)

         Found 338 files belonging to 2 classes.

In [58]: epochs=10
         history = model.fit(
           train_ds,
           validation_data=val_ds,
           epochs=epochs
         )

         Epoch 1/10
         49/49 [==============================] - 29s 586ms/step - loss: 0.2573 - accuracy: 0.9295 - val_loss: 1.6717 - val
         _accuracy: 0.5533
         Epoch 2/10
         49/49 [==============================] - 25s 503ms/step - loss: 0.0510 - accuracy: 0.9814 - val_loss: 1.4199 - val
         _accuracy: 0.5592
         Epoch 3/10
         49/49 [==============================] - 24s 497ms/step - loss: 0.0326 - accuracy: 0.9885 - val_loss: 1.2636 - val
         _accuracy: 0.5828
         Epoch 4/10
         49/49 [==============================] - 24s 498ms/step - loss: 0.0242 - accuracy: 0.9929 - val_loss: 1.8061 - val
         _accuracy: 0.5828
         Epoch 5/10
         49/49 [==============================] - 24s 498ms/step - loss: 0.0215 - accuracy: 0.9917 - val_loss: 1.3066 - val
         _accuracy: 0.5680
         Epoch 6/10
         49/49 [==============================] - 24s 495ms/step - loss: 0.0255 - accuracy: 0.9897 - val_loss: 2.4238 - val
         _accuracy: 0.5503
         Epoch 7/10
         49/49 [==============================] - 24s 500ms/step - loss: 0.0091 - accuracy: 0.9955 - val_loss: 2.4393 - val
         _accuracy: 0.5473
         Epoch 8/10
         49/49 [==============================] - 24s 499ms/step - loss: 0.0048 - accuracy: 0.9987 - val_loss: 2.7689 - val
         _accuracy: 0.5503
         Epoch 9/10
         49/49 [==============================] - 24s 497ms/step - loss: 6.7694e-04 - accuracy: 1.0000 - val_loss: 3.1252 -
         val_accuracy: 0.5562
         Epoch 10/10
         49/49 [==============================] - 25s 501ms/step - loss: 3.9267e-04 - accuracy: 1.0000 - val_loss: 3.2104 -
         val_accuracy: 0.5562
```

Figure 3.2. New model with 2 separated datasets

Figure 3.2 describes the detail of a new approach in our model without splitting data 80-20, instead of that, we feed the model with the train and test data separately from two folders. Eventually, we plot the results (Figure 3.3) below:

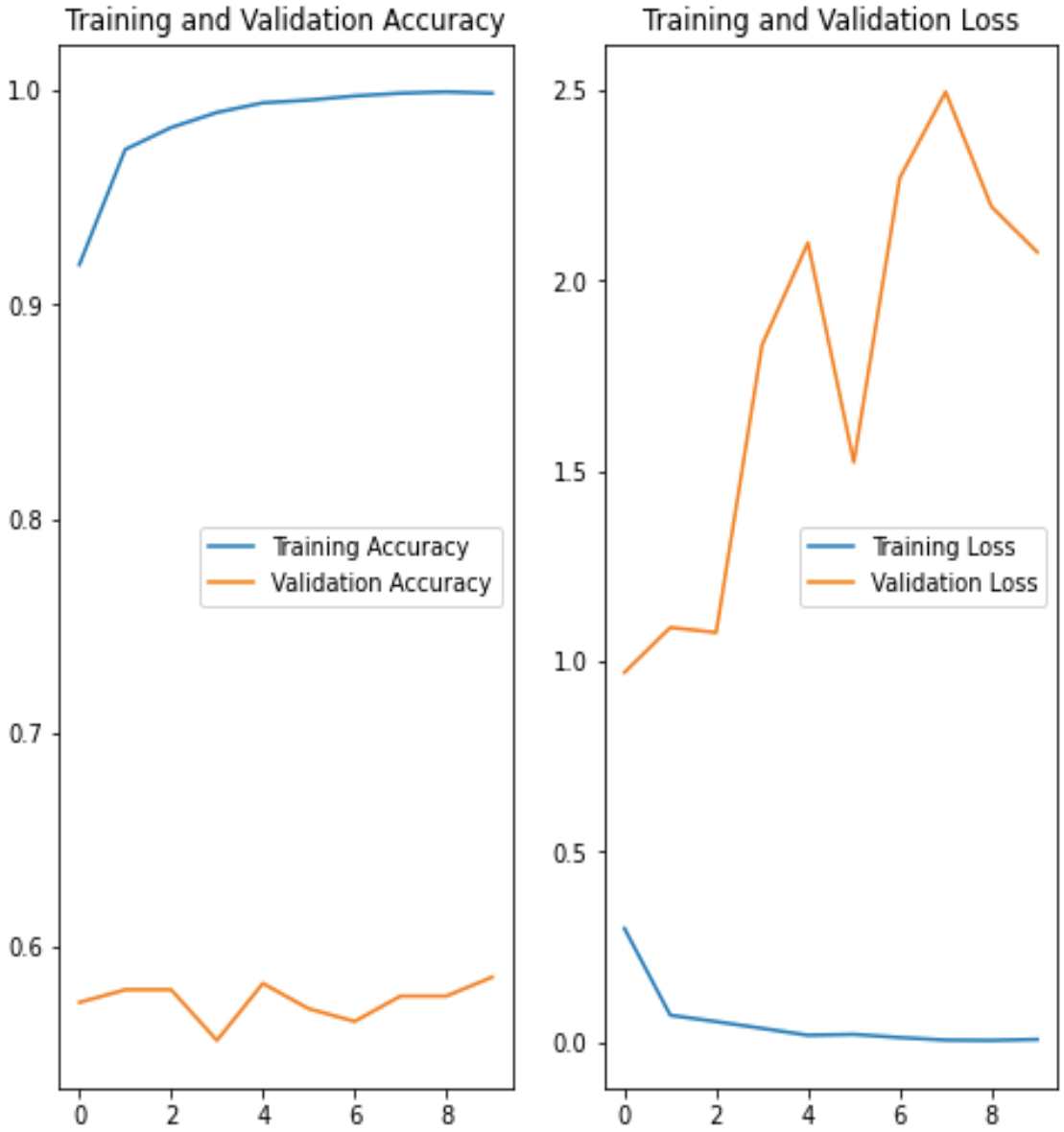


Figure 3.3. Training and Validation Accuracy/Loss

Figure 3.3 indicates that model is overfitting as there is a significant gap between Training Accuracy and Validation Accuracy.

c. Data augmentation

Recognizing the overfitting showed above, data augmentation was applied which rotated the image with different directions but the same class with the original. The purpose of this step is making a larger of the dataset and avoid the possibility that the model could learn too well in the training dataset but performed poorly in the validation dataset.

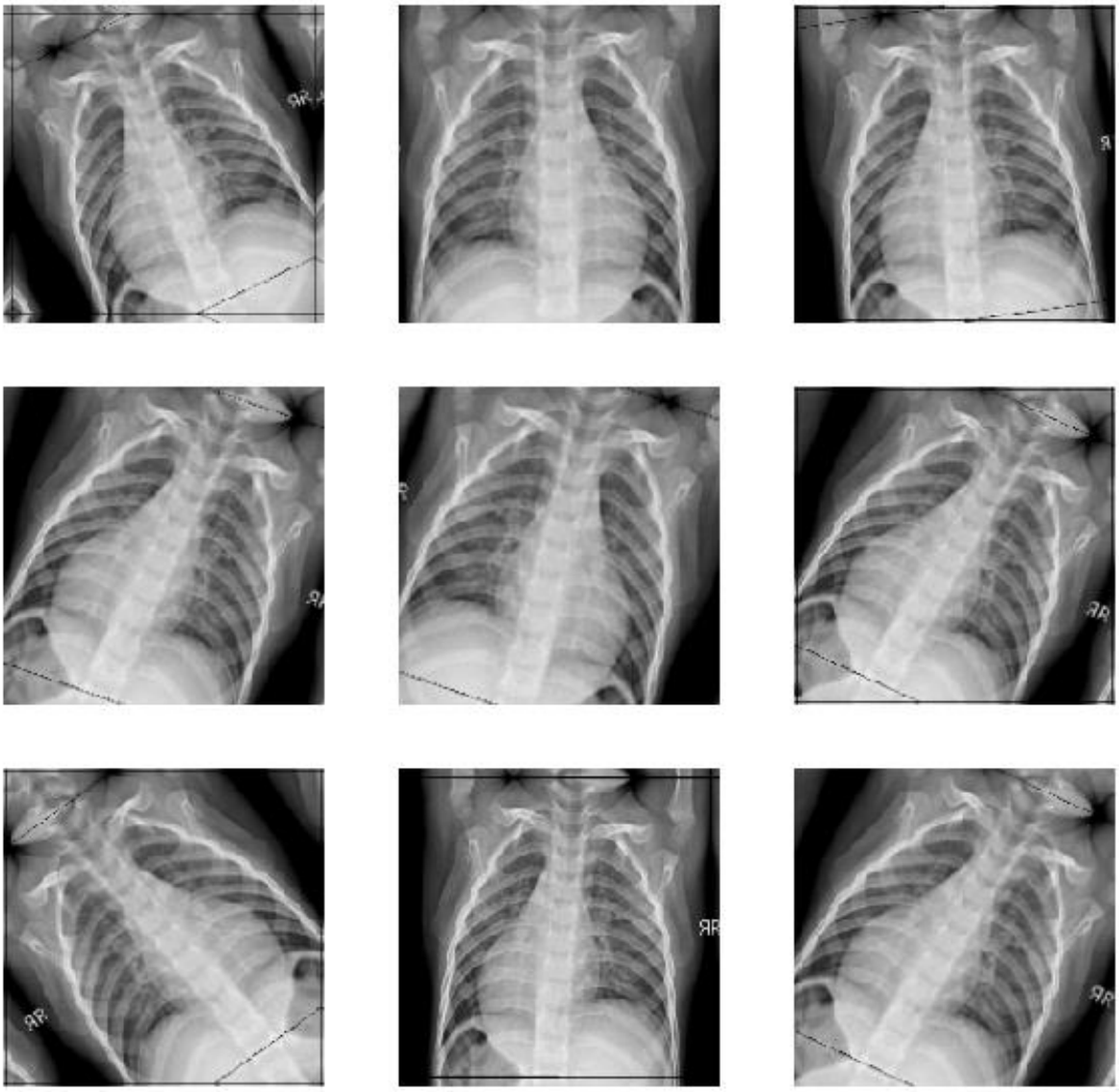


Figure 3.4. Sample of data augmentation.

Figure 3.5. Training and Validation Accuracy/Loss

Figure 3.5 unexpectedly shows that the model is still strongly overfitting and there is no improvement on validation accuracy.

DISCUSSION

After training through 3 different approaches on the same model, we found that using only a single Radiography dataset, the model was able to accurately classify people who are positive or negative with COVID as absolute with the accuracy of train and validation dataset equal to 100%. Moreover, this approach has trained our model too well and without over-fitting, which makes the team members concerned about it because using only one dataset makes the machine be able to learn and remember all images so it loses objectivity. Comparing to the second approach which is using both Radiography and Covid-Chest Xray-Dataset, the results were not our expective with the accuracy around 50 percents.

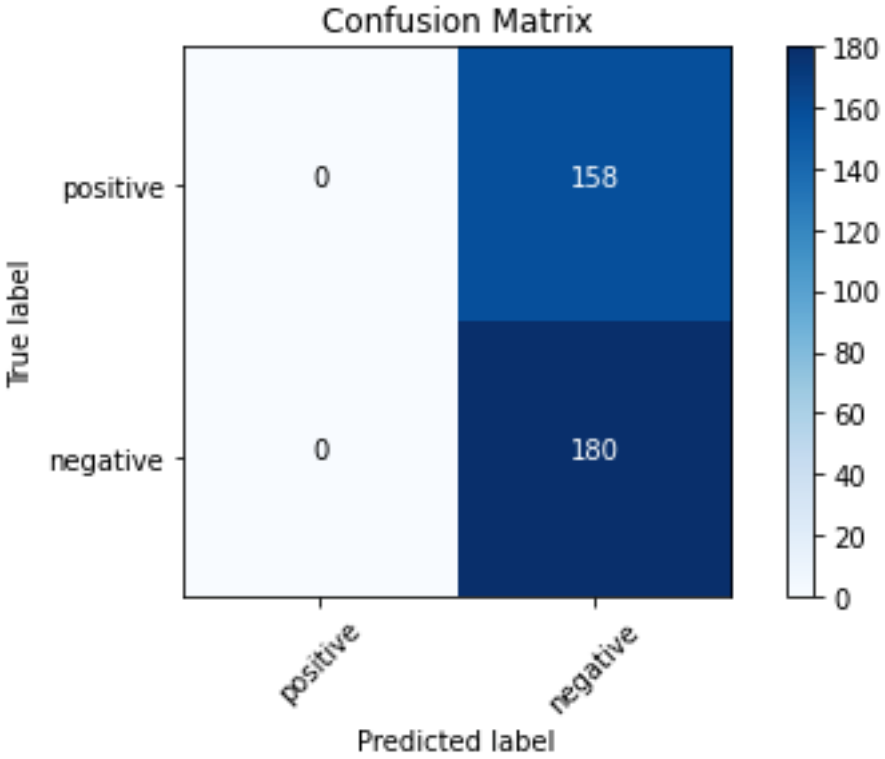


Figure 3.6. Validation Confusion Matrix for Using both dataset approach.

Looking at the above Confusion Matrix, we can calculate the accuracy = (0 + 180) / (0 + 0 + 180 + 158) = 53.25% which can be considered as the coin flipping's rate. The True Positive by zero which strongly indicates that the model cannot predict a person is positive with COVID-19 and it is significant dangerous. Whereas, 158 Positive input cases were all predicted as Negative with True Negative is 100%. We can conclude that the model has failed because it returned Negative regardless input.

The use of data augmentation did not solve the overfitting problem of trained model. This issue may be caused by various aspects. We are considering below methods for further approaching:

- Resampling, collecting more authorized and publicity available X-ray images, so that the model will be able to train more data.
- Applying early stopping iteration. Until certain number of epochs, new epochs may improve the model. However, after this point, the ability of model for generalizing may be weaken since it starts overfitting on training dataset.
- Adapting some frameworks of Convolutional Networks such as ResNet18, ResNet50 or SqueezeNet which are popular achieved promising outcomes in many tasks recent years (Minaee et al., 2020).

CONCLUSION

We demonstrated a machine learning model for COVID-19 detection from Chest X-ray images, by Convolutional Neural Network (CNN) on the training set. Over 2,000 images were collected from 2 datasets and prepared as the main dataset in this project. Then, we experimented a detailed analysis to evaluate the performance of the learning model by different approaches. Based on the results, the models achieved a specificity rate of around 55% on average. It can be seen that the model has a low capability of COVID-19 detection from radiography.

Although the results did not meet the team's expectation, it is reasonable firstly due to the lack of publicity available images that could be considered as a limitation. Secondly, the unbalanced input dataset which consisted of 400 COVID-19 images and 2,000 images of negative patients, would be another considerable limitation that could affect the quality of the model. Importantly, even though there are significant abnormalities been recognized by the radiologist in Covid-19 Positive X-ray images such as ground-glass, mixed attenuation, asymmetric patchy or diffuse airspace opacities, however, it's apparently still at the early stage of saying it happens in all positive cases from the medical point of view. In saying this, we are not sure that all actual 400 positive Xray images that we used to feed the model contain these abnormalities to expect the good learning performance of the model.

In conclusion, due to the limited number of COVID-19 images publicly available so far, and due to the simple applied framework, further experiments are needed on a larger set of cleanly labelled COVID-19 images and more advanced machine learning model need to be applied for a more reliable estimation of the accuracy.

REFERENCES

Bhosle, V., Supriya, S., Sowmya, M., Subramani, V., & Shruthi, G. (2018). Face Recognition with 2D-Convolutional Neural Network. *International Journal of Advanced Research in Computer Science, 9*(Special Issue 3), 373-377. https://doi.org/10.26483/ijarcs.v9i0.6271

Feng, Z.-M., Zhuang, Z.-J., He, W.-B., Ding, J.-P., Yang, W.-J., & Chen, X.-Y. (2016). Lung Cancer with Diffuse Ground-glass Shadow in Two Lungs and Respiratory Failure. *Chinese Medical Journal, 129*(15). https://doi.org/10.4103/0366-6999.186632

Department of Heath of Australia Government. (2020). *What you need to know about coronavirus (COVID-19)*. https://www.health.gov.au/news/health-alerts/novel-coronavirus-2019-ncov-health-alert/what-you-need-to-know-about-coronavirus-covid-19

Kanne, J. P., Little, B. P., Chung, J. H., Elicker, B. M., & Ketai, L. H. (2020). Essentials for Radiologists on COVID-19: An Update- Scientific Expert Panel. *Radiology, 296*(2), E113. https://doi.org/10.1148/radiol.2020200527

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324. https://doi.org/10.1109/5.726791

Li, C., Guo, J., Porikli, F., & Pang, Y. (2018). LightenNet: A Convolutional Neural Network for weakly illuminated image enhancement. *Pattern recognition letters, 104*, 15-22. https://doi.org/10.1016/j.patrec.2018.01.010

Li, H., Jiang, G., Zhang, J., Wang, R., Wang, Z., Zheng, W.-S., & Menze, B. (2018). Fully convolutional network ensembles for white matter hyperintensities segmentation in MR images. *NeuroImage (Orlando, Fla.), 183*, 650-665. https://doi.org/10.1016/j.neuroimage.2018.07.005

Minaee, S., Kafieh, R., Sonka, M., Yazdani, S., & Jamalipour Soufi, G. (2020). Deep-COVID: Predicting COVID-19 from chest X-ray images using deep transfer learning. *Medical image analysis, 65*. https://doi.org/10.1016/j.media.2020.101794

Rodrigues, J. C. L., Hare, S. S., Edey, A., Devaraj, A., Jacob, J., Johnstone, A., McStay, R., Nair, A., & Robinson, G. (2020). An update on COVID-19 for the radiologist - A British society of Thoracic Imaging statement. *Clinical radiology, 75*(5), 323-325. https://doi.org/10.1016/j.crad.2020.03.003

Wang, L., & Wong, A. (2020). COVID-Net: A Tailored Deep Convolutional Neural Network Design for Detection of COVID-19 Cases from Chest X-Ray Images. *arXiv.org*.

Wang, W., Xu, Y., Gao, R., Lu, R., Han, K., Wu, G., & Tan, W. (2020). Detection of SARS-CoV-2 in Different Types of Clinical Specimens. *JAMA : the journal of the American Medical Association, 323*(18), 1843-1844. https://doi.org/10.1001/jama.2020.3786

WHO. (2020). *Coronavirus disease (COVID-19) pandemic*. https://www.who.int/emergencies/diseases/novel-coronavirus-2019

APPENDIX

CODE CAPTURES:

```
#import numpy as np
import shutil
import os
# import data
#metadata_dir = './metadata.csv'
metadata = pd.read_csv('./metadata.csv')


# Selecting all combination of 'COVID-19' patients with 'PA' X-Ray view
virus = "COVID-19" # Virus to look for
x_ray_view = "PA" # View of X-Ray

metadata = "./metadata.csv" # Meta info
imageDir = "./images" # Directory of images
outputDir = './output' # Output directory to store selected images

posDir = './output/positive'
negDir = './output/negative'
metadata_csv = pd.read_csv(metadata)

# loop over the rows of the COVID-19 data frame
for (i, row) in metadata_csv.iterrows():
    if row['view'] == x_ray_view:
        if row['finding'] == virus:
            # Copy to positive folder
            filename = row["filename"].split(os.path.sep)[-1]
            filePath = os.path.sep.join([imageDir, filename])
            shutil.copy2(filePath, posDir)
        else:
            filename = row["filename"].split(os.path.sep)[-1]
            filePath = os.path.sep.join([imageDir, filename])
            shutil.copy2(filePath, negDir)
```

=> This code was used to separate Positive and Negative images from raw dataset.

```
In [80]: # Importing the required libraries
         import matplotlib.pyplot as plt
         import numpy as np
         import os
         import PIL
         import tensorflow as tf
         from sklearn.metrics import classification_report, confusion_matrix,roc_auc_score, roc_curve
         import itertools

         from tensorflow import keras
         from tensorflow.keras import layers
         from tensorflow.keras.models import Sequential

         import pathlib
```

In [81]:
```
'''Setting up the env'''

TRAIN_DIR = './TrainData'
TEST_DIR = './TestData'

IMG_HEIGHT = 180
IMG_WIDTH = 180
BATCH_SIZE = 32
LR = 1e-3


'''Setting up the model which will help with tensorflow models'''
MODEL_NAME = 'XrayImagesToCovidPrediction-{}-{}.model'.format(LR, '6conv-basic')

data_dir = pathlib.Path(TRAIN_DIR)
test_dir = pathlib.Path(TEST_DIR)
print(data_dir)
train_image_count = len(list(data_dir.glob('*/*.png')))

test_image_count = len(list(test_dir.glob('*/*.*')))

print(train_image_count)
print(test_dir)
print(test_image_count)
```

```
/content/x_ray_project/TrainData
1560
/content/x_ray_project/TestData
338
```

In [82]:
```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  seed=123,
  image_size=(IMG_HEIGHT, IMG_WIDTH),
  batch_size=BATCH_SIZE)
```

```
Found 1560 files belonging to 2 classes.
```

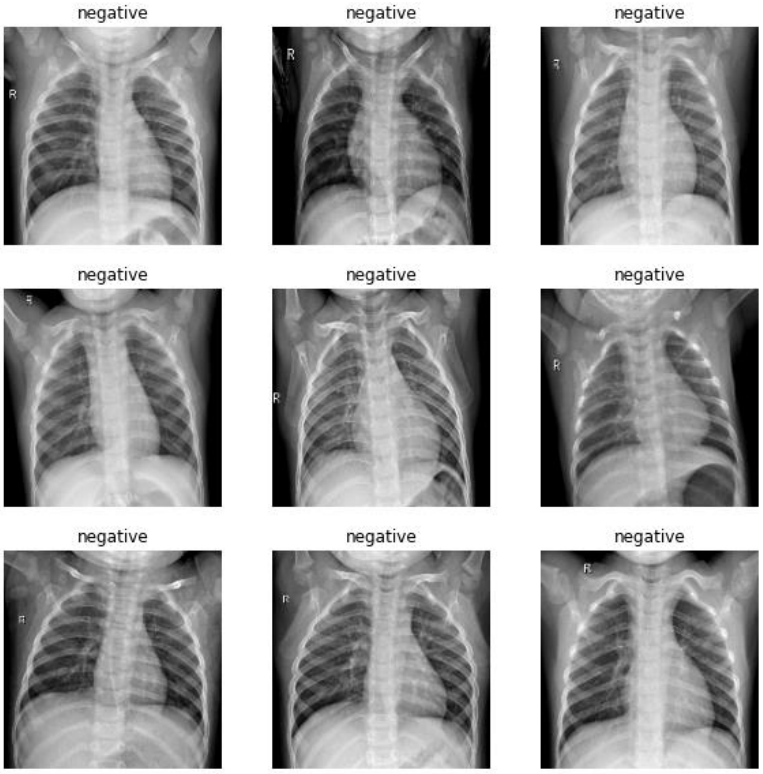
In [83]:
```
print(train_ds.class_names)
class_names = train_ds.class_names
```

```
['negative', 'positive']
```

In [84]:
```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  test_dir,
  seed=123,
  image_size=(IMG_HEIGHT, IMG_WIDTH),
  batch_size=BATCH_SIZE)
```

```
Found 338 files belonging to 2 classes.
```

In [86]:
```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



click to expand output; double click to hide output

```
IMG_WIDTH = 180
BATCH_SIZE = 32
train_datagen =keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255)

test_datagen = keras.preprocessing.image.ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(data_dir,target_size=(IMG_HEIGHT, IMG_WI

val_generator = test_datagen.flow_from_directory(test_dir,target_size=(IMG_HEIGHT, IMG_WIDTH
```

```
Found 1560 images belonging to 2 classes.
Found 338 images belonging to 2 classes.
```

In [87]:
```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 180, 180, 3)
(32,)
```

In [88]:
```
AUTOTUNE = tf.data.experimental.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [89]:
```
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)
```

In [90]:
```
num_classes = 2

model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

In [91]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [92]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_3 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               3965056
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 3,988,898
Trainable params: 3,988,898
Non-trainable params: 0
_________________________________________________________________
```

In [93]:
```
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
49/49 [==============================] - 6s 118ms/step - loss: 0.2976 - accuracy: 0.9186 -
val_loss: 0.9703 - val_accuracy: 0.5740
Epoch 2/10
49/49 [==============================] - 1s 20ms/step - loss: 0.0693 - accuracy: 0.9724 - v
al_loss: 1.0887 - val_accuracy: 0.5799
Epoch 3/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0526 - accuracy: 0.9827 - v
al_loss: 1.0755 - val_accuracy: 0.5799
Epoch 4/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0348 - accuracy: 0.9897 - v
al_loss: 1.8315 - val_accuracy: 0.5562
Epoch 5/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0167 - accuracy: 0.9942 - v
al_loss: 2.1000 - val_accuracy: 0.5828
Epoch 6/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0190 - accuracy: 0.9955 - v
al_loss: 1.5232 - val_accuracy: 0.5710
Epoch 7/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0106 - accuracy: 0.9974 - v
al_loss: 2.2705 - val_accuracy: 0.5651
Epoch 8/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0040 - accuracy: 0.9987 - v
al_loss: 2.4968 - val_accuracy: 0.5769
Epoch 9/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0031 - accuracy: 0.9994 - v
al_loss: 2.1947 - val_accuracy: 0.5769
Epoch 10/10
49/49 [==============================] - 1s 19ms/step - loss: 0.0058 - accuracy: 0.9987 - v
al_loss: 2.0754 - val_accuracy: 0.5858
```

In [95]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='center right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='center right')
plt.title('Training and Validation Loss')
plt.show()
```

In [96]:
```
Y_pred = model.predict_generator(val_generator)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
cm = confusion_matrix(val_generator.classes, y_pred)
print(cm)
```

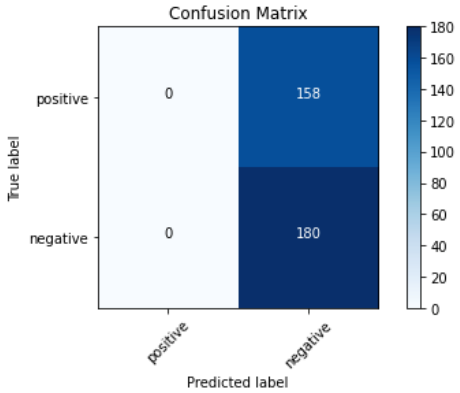
```
Confusion Matrix
[[  0 158]
 [  0 180]]
```

In [97]:
```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [98]:
```
cm_plot_labels = ['positive','negative']
```

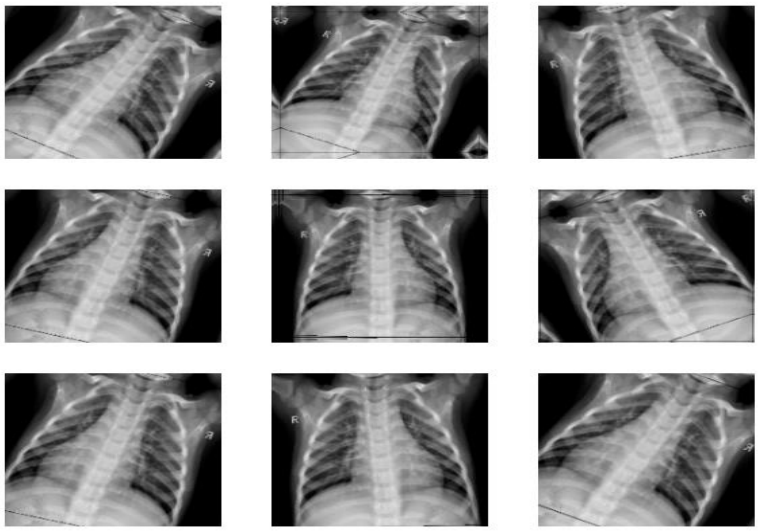
In [99]:
```
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[  0 158]
 [  0 180]]
```



In [100]:
```
data_augmentation = keras.Sequential(
  [
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(IMG_HEIGHT,
                                                              IMG_WIDTH,
                                                              3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

In [101]:
```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```



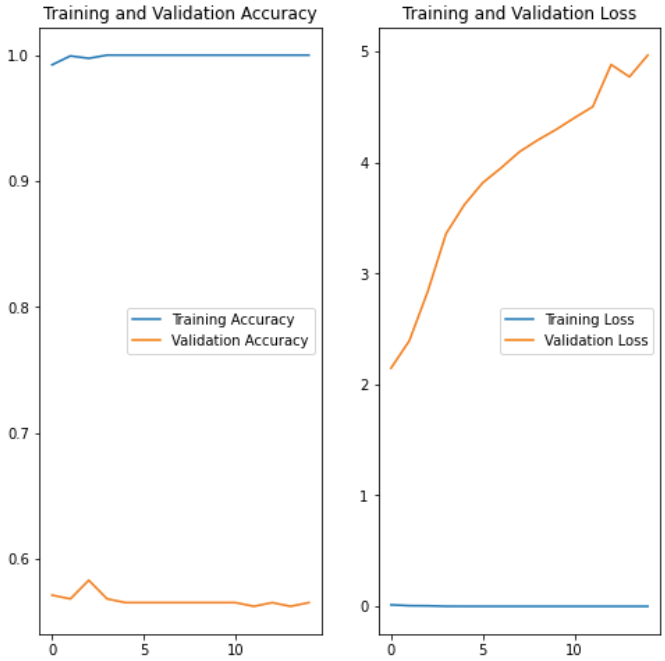
In [102]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [103]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_3 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 180, 180, 16)      448
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 90, 90, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 90, 90, 32)        4640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 45, 45, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 30976)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               3965056
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 3,988,898
Trainable params: 3,988,898
Non-trainable params: 0
_________________________________________________________________
```

In [104]:
```
epochs = 15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

In [106]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='center right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='center right')
plt.title('Training and Validation Loss')
plt.show()
```



In [108]:
```
#test_dir = pathlib.Path(TEST_DIR)
test_neg = '.TestData/CR.1.2.840.113564.192168196.2020031913094890017.1203801020003.png'
img = keras.preprocessing.image.load_img(
    test_neg
    , target_size=(IMG_HEIGHT, IMG_WIDTH)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
This image most likely belongs to negative with a 98.81 percent confidence.
```

In [109]:
```
pos = list(data_dir.glob('positive/*'))
PIL.Image.open(str(pos[0]))
```

In [110]:
```
test_neg = '.TestData/CR.1.2.840.113564.1722810170.202003211940047348.1003000225002.png'
img = keras.preprocessing.image.load_img(
    test_neg
    , target_size=(IMG_HEIGHT, IMG_WIDTH)
)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
This image most likely belongs to positive with a 100.00 percent confidence.
```