In [ ]:
```
## Authenticate and mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
%cd /content/drive/My Drive/X_ray_photos/
```

```
/content/drive/My Drive/X_ray_photos
```

In [ ]:
```
!ls
```

```
covid19  foo.png  model.png  normal  test  train  validation
```

Importing libraries

In [ ]:
```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os
import glob
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
base_dir = !pwd
base_dir = base_dir[0]
print(base_dir)
print(type(base_dir))
```

```
/content/drive/My Drive/X_ray_photos
<class 'str'>
```

In [ ]:
```
# classes = ['covid19', 'normal']
```

In [ ]:
```
# # The code below creates a train and a val folder each containing 2 folders (one f
# # It then copies the images from the original folders to these new folders such th
# # to the training set and 25% of the images go into the validation set.

# for i in classes:

#   os.makedirs(base_dir +'/train/' + i)
#   os.makedirs(base_dir +'/validation/' + i)
#   source = base_dir + '/' + i
#   allFileNames = os.listdir(source)
#   np.random.shuffle(allFileNames)
#   test_ratio = 0.25

#   train_FileNames, test_FileNames = np.split(np.array(allFileNames),
#                                                          [int(len(allFileNames)* (1 -

#   train_FileNames = [source+'/'+ name for name in train_FileNames.tolist()]
#   test_FileNames = [source+'/' + name for name in test_FileNames.tolist()]

#   for name in train_FileNames:
```

```
#     shutil.copy(name, base_dir +'/train/' + i)

#   for name in test_FileNames:
#     shutil.copy(name, base_dir +'/validation/' + i)
```

In [ ]:
```
!ls
```

```
covid19  foo.png  model.png  normal  test  train  validation
```

In [ ]:
```
# Setting paths for training an validation sets
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

In [ ]:
```
print(train_dir)
print(val_dir)
print(test_dir)
print(type(train_dir))
```

```
/content/drive/My Drive/X_ray_photos/train
/content/drive/My Drive/X_ray_photos/validation
/content/drive/My Drive/X_ray_photos/test
<class 'str'>
```

In [ ]:
```
BATCH_SIZE = 20  # Number of training examples to process before updating our models
IMG_SHAPE  = 180  # Our training data consists of images with width of 180 pixels an
```

In [ ]:
```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
      img = img.reshape((180,180))
      ax.imshow(img, cmap='gray')
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
# Read images from the disk.
# Decode contents of these images and convert it into proper grid format as per thei
# Convert them into floating point tensors.
# Rescale the tensors from values between 0 and 255 to values between 0 and 1,
# as neural networks prefer to deal with small input values.
image_gen_train = ImageDataGenerator(rescale=1./255,
                    rotation_range=45,
                    width_shift_range=.15,
                    height_shift_range=.15,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )

train_data_gen = image_gen_train.flow_from_directory(
                                                batch_size=BATCH_SIZE,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE, IMG_SHAPE),
                                                color_mode='grayscale',
                                                class_mode='binary'
                                                )
```

```
Found 194 images belonging to 2 classes.
```

In [ ]:
```
type(train_data_gen)
print(len(train_data_gen))
```

```
10
```

In [ ]:
```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
```

In [ ]:
```
# plotImages(augmented_images)
```

In [ ]:
```
image_gen_val = ImageDataGenerator(rescale=1./255)

val_data_gen = image_gen_val.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=val_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 color_mode='grayscale',
                                                 class_mode='binary')

test_data_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=test_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 color_mode='grayscale',
                                                 class_mode='binary')
```

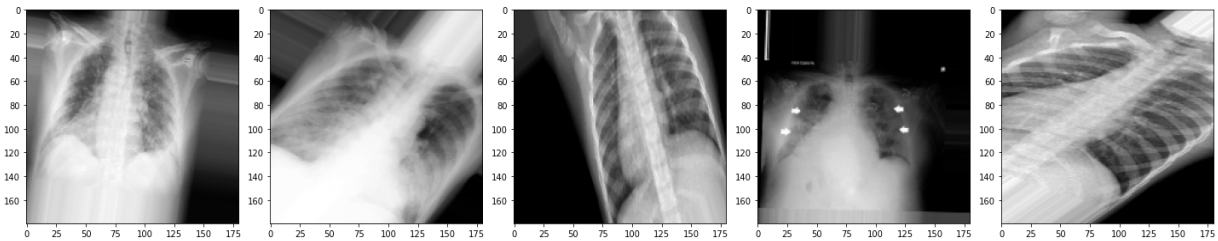
```
Found 66 images belonging to 2 classes.
Found 32 images belonging to 2 classes.
```

In [ ]:
```
len(val_data_gen)
```

Out[ ]:
```
4
```

In [ ]:
```
sample_training_images, _ = next(train_data_gen)

# next function returns a batch from the dataset. One batch is a tuple of (many imag
# For right now, we're discarding the labels because we just want to look at the ima
```

In [ ]:
```
plotImages(sample_training_images[:5])  # Plot images 0-4
```



Creating model

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, Activation
import math
```

In [ ]:

```
cnn = Sequential()

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Flatten())

cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'relu', units = 64))

cnn.add(Dense(activation = 'sigmoid', units = 1))
```

In [ ]:

```
cnn.summary()
```

```
Model: "sequential_24"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_77 (Conv2D)          (None, 178, 178, 32)      320

 max_pooling2d_32 (MaxPoolin  (None, 89, 89, 32)       0
 g2D)

 conv2d_78 (Conv2D)          (None, 87, 87, 32)        9248

 max_pooling2d_33 (MaxPoolin  (None, 43, 43, 32)       0
 g2D)

 conv2d_79 (Conv2D)          (None, 41, 41, 32)        9248

 max_pooling2d_34 (MaxPoolin  (None, 20, 20, 32)       0
 g2D)

 conv2d_80 (Conv2D)          (None, 18, 18, 64)        18496

 max_pooling2d_35 (MaxPoolin  (None, 9, 9, 64)         0
 g2D)

 conv2d_81 (Conv2D)          (None, 7, 7, 64)          36928

 max_pooling2d_36 (MaxPoolin  (None, 3, 3, 64)         0
 g2D)

 flatten_21 (Flatten)        (None, 576)               0

 dense_47 (Dense)            (None, 128)               73856

 dense_48 (Dense)            (None, 64)                8256

 dense_49 (Dense)            (None, 1)                 65

=================================================================
```

```
Total params: 156,417
Trainable params: 156,417
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=3)
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', p
callbacks_list = [ early, learning_rate_reduction]
```

In [ ]:
```
cnn.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [ ]:
```
EPOCHS = 30
history = cnn.fit_generator(
    train_data_gen,
    epochs=EPOCHS,
    validation_data=val_data_gen
    )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Model.
fit_generator` is deprecated and will be removed in a future version. Please use `Mo
del.fit`, which supports generators.
  """
Epoch 1/30
10/10 [==============================] - 5s 439ms/step - loss: 0.7150 - accuracy: 0.
5000 - val_loss: 0.6925 - val_accuracy: 0.5000
Epoch 2/30
10/10 [==============================] - 4s 400ms/step - loss: 0.6922 - accuracy: 0.
5361 - val_loss: 0.6890 - val_accuracy: 0.5000
Epoch 3/30
10/10 [==============================] - 4s 452ms/step - loss: 0.6907 - accuracy: 0.
5876 - val_loss: 0.6783 - val_accuracy: 0.5000
Epoch 4/30
10/10 [==============================] - 4s 410ms/step - loss: 0.6846 - accuracy: 0.
5309 - val_loss: 0.6569 - val_accuracy: 0.7727
Epoch 5/30
10/10 [==============================] - 4s 403ms/step - loss: 0.6446 - accuracy: 0.
6804 - val_loss: 0.9743 - val_accuracy: 0.5000
Epoch 6/30
10/10 [==============================] - 4s 397ms/step - loss: 0.6787 - accuracy: 0.
5979 - val_loss: 0.6294 - val_accuracy: 0.7727
Epoch 7/30
10/10 [==============================] - 4s 410ms/step - loss: 0.5935 - accuracy: 0.
6959 - val_loss: 0.4290 - val_accuracy: 0.8636
Epoch 8/30
10/10 [==============================] - 4s 402ms/step - loss: 0.6434 - accuracy: 0.
6237 - val_loss: 0.4420 - val_accuracy: 0.9242
Epoch 9/30
10/10 [==============================] - 4s 417ms/step - loss: 0.5676 - accuracy: 0.
7216 - val_loss: 0.4924 - val_accuracy: 0.8485
Epoch 10/30
10/10 [==============================] - 4s 420ms/step - loss: 0.5947 - accuracy: 0.
7165 - val_loss: 0.5117 - val_accuracy: 0.7879
Epoch 11/30
10/10 [==============================] - 4s 430ms/step - loss: 0.5623 - accuracy: 0.
7113 - val_loss: 0.3862 - val_accuracy: 0.8939
Epoch 12/30
10/10 [==============================] - 4s 448ms/step - loss: 0.4759 - accuracy: 0.
7835 - val_loss: 0.2058 - val_accuracy: 0.9848
Epoch 13/30
10/10 [==============================] - 4s 434ms/step - loss: 0.4417 - accuracy: 0.
7732 - val_loss: 0.1889 - val_accuracy: 0.9394
Epoch 14/30
```

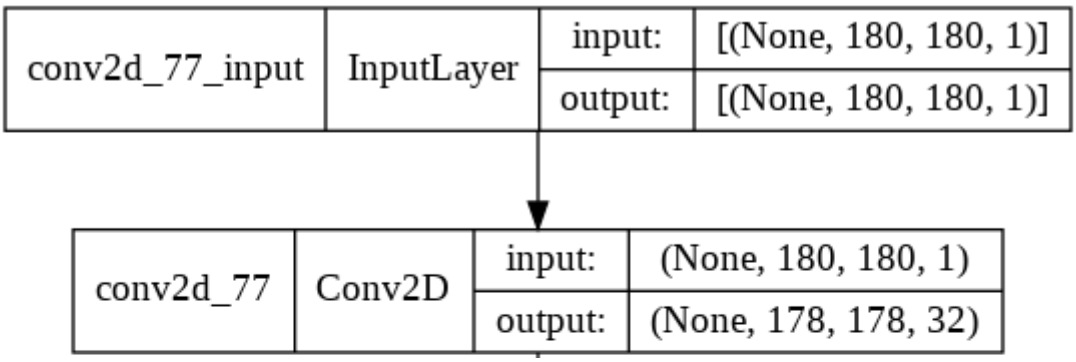
```
10/10 [==============================] - 4s 433ms/step - loss: 0.4562 - accuracy: 0.
8093 - val_loss: 0.2593 - val_accuracy: 0.8939
Epoch 15/30
10/10 [==============================] - 4s 433ms/step - loss: 0.5292 - accuracy: 0.
7113 - val_loss: 0.3136 - val_accuracy: 0.9697
Epoch 16/30
10/10 [==============================] - 4s 421ms/step - loss: 0.4802 - accuracy: 0.
7577 - val_loss: 0.2536 - val_accuracy: 0.9394
Epoch 17/30
10/10 [==============================] - 4s 428ms/step - loss: 0.4812 - accuracy: 0.
7732 - val_loss: 0.1994 - val_accuracy: 0.9545
Epoch 18/30
10/10 [==============================] - 4s 433ms/step - loss: 0.4599 - accuracy: 0.
7938 - val_loss: 0.2074 - val_accuracy: 0.9545
Epoch 19/30
10/10 [==============================] - 4s 438ms/step - loss: 0.4084 - accuracy: 0.
8196 - val_loss: 0.1063 - val_accuracy: 0.9545
Epoch 20/30
10/10 [==============================] - 4s 420ms/step - loss: 0.3832 - accuracy: 0.
8608 - val_loss: 0.0822 - val_accuracy: 0.9848
Epoch 21/30
10/10 [==============================] - 4s 422ms/step - loss: 0.3247 - accuracy: 0.
8557 - val_loss: 0.0907 - val_accuracy: 0.9697
Epoch 22/30
10/10 [==============================] - 4s 433ms/step - loss: 0.3193 - accuracy: 0.
8505 - val_loss: 0.0793 - val_accuracy: 1.0000
Epoch 23/30
10/10 [==============================] - 4s 436ms/step - loss: 0.2898 - accuracy: 0.
8918 - val_loss: 0.1014 - val_accuracy: 0.9848
Epoch 24/30
10/10 [==============================] - 4s 434ms/step - loss: 0.2741 - accuracy: 0.
8918 - val_loss: 0.0743 - val_accuracy: 0.9848
Epoch 25/30
10/10 [==============================] - 4s 413ms/step - loss: 0.3478 - accuracy: 0.
8660 - val_loss: 0.1119 - val_accuracy: 0.9848
Epoch 26/30
10/10 [==============================] - 4s 421ms/step - loss: 0.2981 - accuracy: 0.
8918 - val_loss: 0.0743 - val_accuracy: 1.0000
Epoch 27/30
10/10 [==============================] - 4s 415ms/step - loss: 0.2824 - accuracy: 0.
8918 - val_loss: 0.0555 - val_accuracy: 0.9848
Epoch 28/30
10/10 [==============================] - 4s 409ms/step - loss: 0.2274 - accuracy: 0.
9124 - val_loss: 0.0795 - val_accuracy: 0.9697
Epoch 29/30
10/10 [==============================] - 4s 412ms/step - loss: 0.2891 - accuracy: 0.
8557 - val_loss: 0.0889 - val_accuracy: 0.9848
Epoch 30/30
10/10 [==============================] - 4s 403ms/step - loss: 0.2696 - accuracy: 0.
9175 - val_loss: 0.1163 - val_accuracy: 0.9848
```

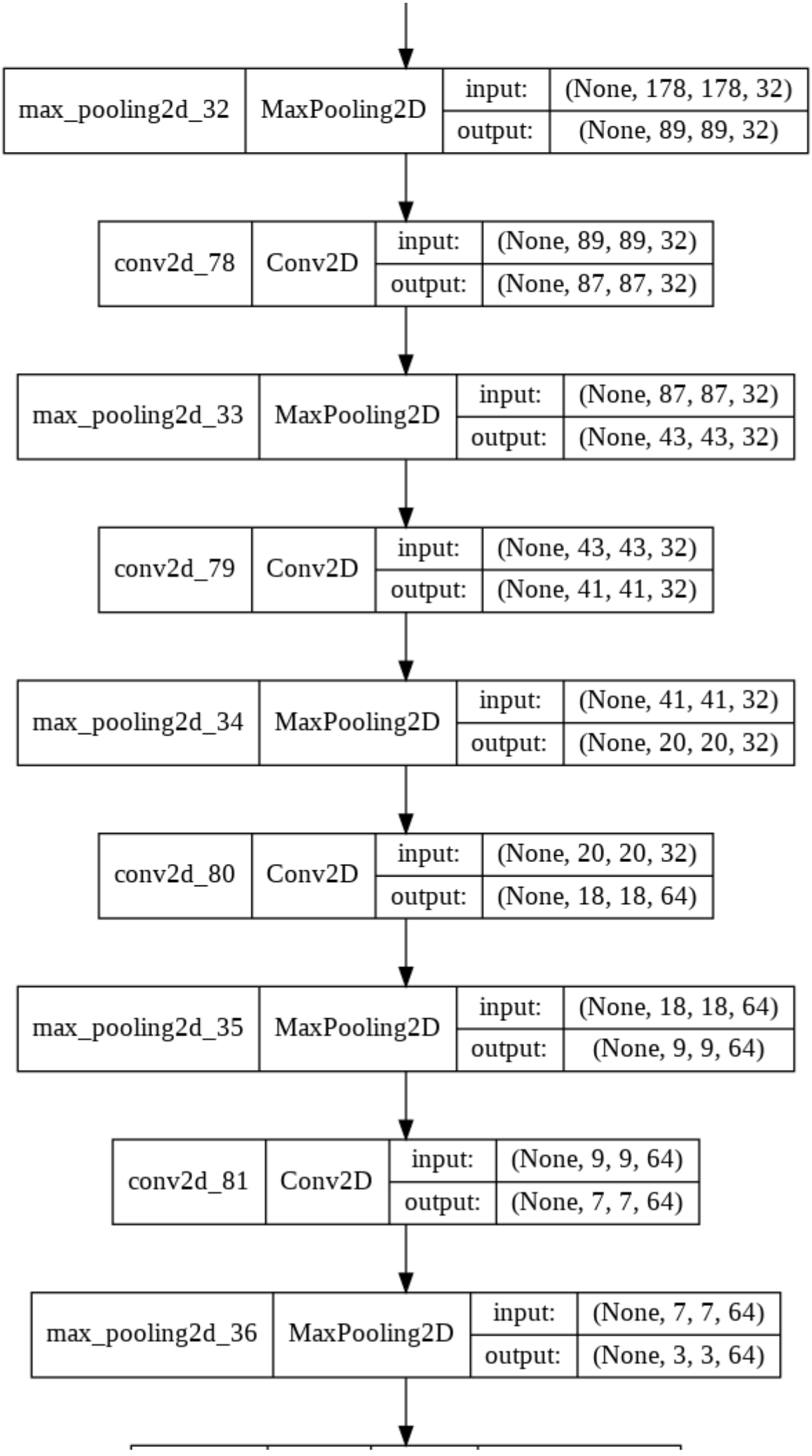
Visualizing CNN model

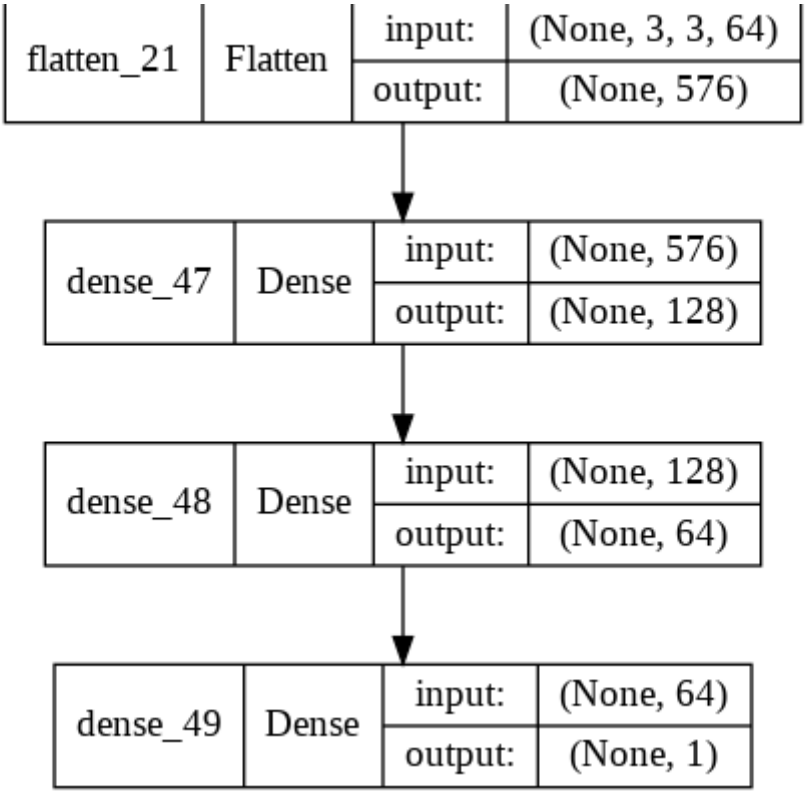
In [ ]:

```
from tensorflow.keras.utils import plot_model
plot_model(cnn,show_shapes=True, show_layer_names=True, rankdir='TB', expand_nested=
```

Out[ ]:




max_pooling2d_32
MaxPooling2D
input: (None, 178, 178, 32)
output: (None, 89, 89, 32)
conv2d_78
Conv2D
input: (None, 89, 89, 32)
output: (None, 87, 87, 32)
max_pooling2d_33
MaxPooling2D
input: (None, 87, 87, 32)
output: (None, 43, 43, 32)
conv2d_79
Conv2D
input: (None, 43, 43, 32)
output: (None, 41, 41, 32)
max_pooling2d_34
MaxPooling2D
input: (None, 41, 41, 32)
output: (None, 20, 20, 32)
conv2d_80
Conv2D
input: (None, 20, 20, 32)
output: (None, 18, 18, 64)
max_pooling2d_35
MaxPooling2D
input: (None, 18, 18, 64)
output: (None, 9, 9, 64)
conv2d_81
Conv2D
input: (None, 9, 9, 64)
output: (None, 7, 7, 64)
max_pooling2d_36
MaxPooling2D
input: (None, 7, 7, 64)
output: (None, 3, 3, 64)



Prediction and Results

In [ ]:
```
test_accu = cnn.evaluate(test_data_gen)
print('The testing accuracy is :',test_accu[1]*100, '%')
```

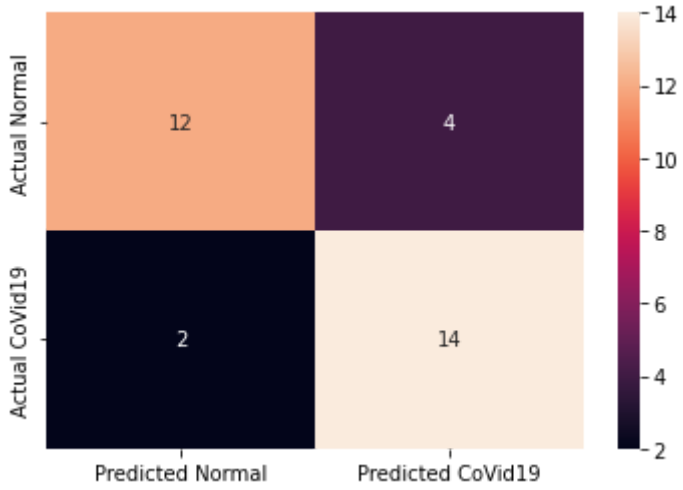
```
2/2 [==============================] - 1s 351ms/step - loss: 0.4656 - accuracy: 0.81
25
The testing accuracy is : 81.25 %
```

In [ ]:
```
preds = cnn.predict(test_data_gen,verbose=1)
predictions = preds.copy()
predictions[predictions <= 0.5] = 0
predictions[predictions > 0.5] = 1
```

```
2/2 [==============================] - 1s 350ms/step
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix

cm = pd.DataFrame(data=confusion_matrix(test_data_gen.classes, predictions, labels=[
columns=["Predicted Normal", "Predicted CoVid19"])

import seaborn as sns
sns.heatmap(cm,annot=True,fmt="d")
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7e801fff50>
```



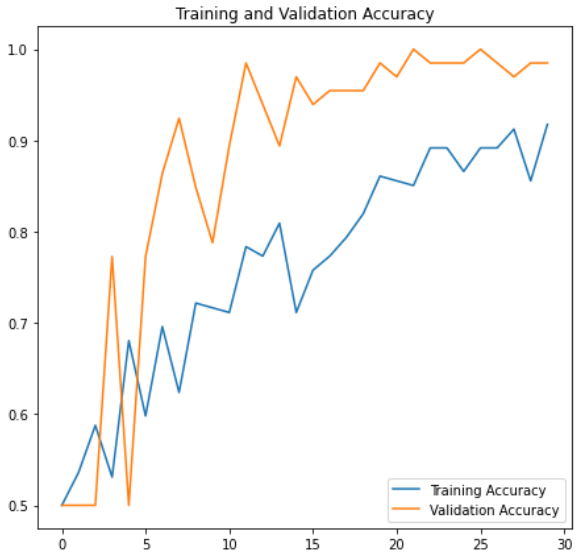
In [ ]:

```
print(classification_report(y_true=test_data_gen.classes,y_pred=predictions,target_n
```

```
              precision    recall  f1-score   support

      NORMAL       0.86      0.75      0.80        16
    CoVid19        0.78      0.88      0.82        16

    accuracy                           0.81        32
   macro avg       0.82      0.81      0.81        32
weighted avg       0.82      0.81      0.81        32
```

In [ ]:

```
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=(16, 7))
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='lower 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='upper right')
plt.title('Training and Validation Loss')
plt.savefig('./foo.png')
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy


Training and Validation Loss
Training Loss
Validation Loss

In [ ]:
```
## Authenticate and mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
%cd /content/drive/My Drive/X_ray_photos/
```

```
/content/drive/My Drive/X_ray_photos
```

In [ ]:
```
!ls
```

```
covid19  foo2.png  foo.png  model.png  normal  test  train  validation
```

Importing libraries

In [ ]:
```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os
import glob
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
base_dir = !pwd
base_dir = base_dir[0]
print(base_dir)
print(type(base_dir))
```

```
/content/drive/My Drive/X_ray_photos
<class 'str'>
```

In [ ]:
```
# classes = ['covid19', 'normal']
```

In [ ]:
```
# # The code below creates a train and a val folder each containing 2 folders (one f
# # It then copies the images from the original folders to these new folders such th
# # to the training set and 25% of the images go into the validation set.

# for i in classes:

#   os.makedirs(base_dir +'/train/' + i)
#   os.makedirs(base_dir +'/validation/' + i)
#   source = base_dir + '/' + i
#   allFileNames = os.listdir(source)
#   np.random.shuffle(allFileNames)
#   test_ratio = 0.25

#   train_FileNames, test_FileNames = np.split(np.array(allFileNames),
#                                                          [int(len(allFileNames)* (1 -

#   train_FileNames = [source+'/'+ name for name in train_FileNames.tolist()]
#   test_FileNames = [source+'/' + name for name in test_FileNames.tolist()]

#   for name in train_FileNames:
```

```
#     shutil.copy(name, base_dir +'/train/' + i)

#   for name in test_FileNames:
#     shutil.copy(name, base_dir +'/validation/' + i)
```

In [ ]:
```
!ls
```

```
covid19  foo.png  model.png  normal  test  train  validation
```

In [ ]:
```
# Setting paths for training an validation sets
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

In [ ]:
```
print(train_dir)
print(val_dir)
print(test_dir)
print(type(train_dir))
```

```
/content/drive/My Drive/X_ray_photos/train
/content/drive/My Drive/X_ray_photos/validation
/content/drive/My Drive/X_ray_photos/test
<class 'str'>
```

In [ ]:
```
BATCH_SIZE = 15  # Number of training examples to process before updating our models
IMG_SHAPE  = 224  # Our training data consists of images with width of 180 pixels an
```

In [ ]:
```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
      img = img.reshape((224,224))
      ax.imshow(img, cmap='gray')
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
# Read images from the disk.
# Decode contents of these images and convert it into proper grid format as per thei
# Convert them into floating point tensors.
# Rescale the tensors from values between 0 and 255 to values between 0 and 1,
# as neural networks prefer to deal with small input values.
image_gen_train = ImageDataGenerator(rescale=1./255,
                    rotation_range=45,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )

train_data_gen = image_gen_train.flow_from_directory(
                                                batch_size=BATCH_SIZE,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE, IMG_SHAPE),
                                                color_mode='grayscale',
                                                class_mode='binary'
                                                )
```

```
Found 194 images belonging to 2 classes.
```

In [ ]:
```
type(train_data_gen)
print(len(train_data_gen))
```

```
10
```

In [ ]:
```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
```

In [ ]:
```
# plotImages(augmented_images)
```

In [ ]:
```
image_gen_val = ImageDataGenerator(rescale=1./255)

val_data_gen = image_gen_val.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=val_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 color_mode='grayscale',
                                                 class_mode='binary')

test_data_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=test_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 color_mode='grayscale',
                                                 class_mode='binary')
```

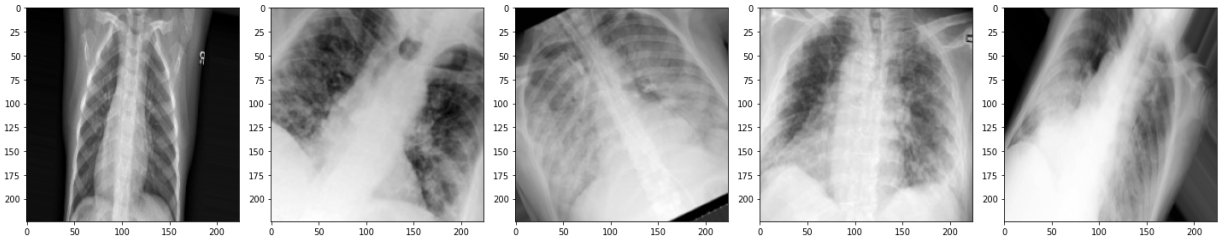
```
Found 66 images belonging to 2 classes.
Found 32 images belonging to 2 classes.
```

In [ ]:
```
len(val_data_gen)
```

Out[ ]:
```
4
```

In [ ]:
```
sample_training_images, _ = next(train_data_gen)

# next function returns a batch from the dataset. One batch is a tuple of (many imag
# For right now, we're discarding the labels because we just want to look at the ima
```

In [ ]:
```
plotImages(sample_training_images[:5])  # Plot images 0-4
```



Creating model

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, Activation
import math
```

In [ ]:
```
```

```
cnn = Sequential()

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Flatten())

cnn.add(Dense(activation = 'relu', units = 128))
cnn.add(Dense(activation = 'relu', units = 64))
cnn.add(Dropout(0.5))

cnn.add(Dense(activation = 'sigmoid', units = 1))
```

In [ ]:

```
cnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 222, 222, 32)      320

 max_pooling2d_5 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_6 (Conv2D)           (None, 109, 109, 32)      9248

 max_pooling2d_6 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_7 (Conv2D)           (None, 52, 52, 32)        9248

 max_pooling2d_7 (MaxPooling  (None, 26, 26, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 24, 24, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 conv2d_9 (Conv2D)           (None, 10, 10, 64)        36928

 max_pooling2d_9 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dense_2 (Dense)             (None, 128)               204928

 dense_3 (Dense)             (None, 64)                8256

 dropout (Dropout)           (None, 64)                0

 dense_4 (Dense)             (None, 1)                 65
```

```
=================================================================
Total params: 287,489
Trainable params: 287,489
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=3)
learning_rate_reduction = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', p
callbacks_list = [ early, learning_rate_reduction]
```

In [ ]:

```
cnn.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [ ]:

```
EPOCHS = 30
history = cnn.fit_generator(
    train_data_gen,
    epochs=EPOCHS,
    validation_data=val_data_gen
    )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: `Model.
fit_generator` is deprecated and will be removed in a future version. Please use `Mo
del.fit`, which supports generators.
  """
Epoch 1/30
13/13 [==============================] - 119s 9s/step - loss: 0.6962 - accuracy: 0.5
309 - val_loss: 0.6865 - val_accuracy: 0.5000
Epoch 2/30
13/13 [==============================] - 4s 344ms/step - loss: 0.6919 - accuracy: 0.
5309 - val_loss: 0.6837 - val_accuracy: 0.5000
Epoch 3/30
13/13 [==============================] - 4s 342ms/step - loss: 0.6644 - accuracy: 0.
6237 - val_loss: 0.5501 - val_accuracy: 0.8485
Epoch 4/30
13/13 [==============================] - 4s 342ms/step - loss: 0.5359 - accuracy: 0.
7423 - val_loss: 0.1207 - val_accuracy: 1.0000
Epoch 5/30
13/13 [==============================] - 4s 333ms/step - loss: 0.4270 - accuracy: 0.
7990 - val_loss: 0.1123 - val_accuracy: 0.9848
Epoch 6/30
13/13 [==============================] - 4s 339ms/step - loss: 0.4877 - accuracy: 0.
7784 - val_loss: 0.3661 - val_accuracy: 0.8636
Epoch 7/30
13/13 [==============================] - 4s 338ms/step - loss: 0.4362 - accuracy: 0.
8093 - val_loss: 0.1509 - val_accuracy: 0.9848
Epoch 8/30
13/13 [==============================] - 4s 340ms/step - loss: 0.3201 - accuracy: 0.
8660 - val_loss: 0.0433 - val_accuracy: 1.0000
Epoch 9/30
13/13 [==============================] - 4s 338ms/step - loss: 0.3081 - accuracy: 0.
8814 - val_loss: 0.0507 - val_accuracy: 0.9848
Epoch 10/30
13/13 [==============================] - 4s 337ms/step - loss: 0.2967 - accuracy: 0.
8814 - val_loss: 0.1411 - val_accuracy: 0.9848
Epoch 11/30
13/13 [==============================] - 4s 338ms/step - loss: 0.3325 - accuracy: 0.
8557 - val_loss: 0.0714 - val_accuracy: 1.0000
Epoch 12/30
13/13 [==============================] - 4s 339ms/step - loss: 0.4676 - accuracy: 0.
7784 - val_loss: 0.1879 - val_accuracy: 0.9848
Epoch 13/30
13/13 [==============================] - 4s 344ms/step - loss: 0.3270 - accuracy: 0.
```

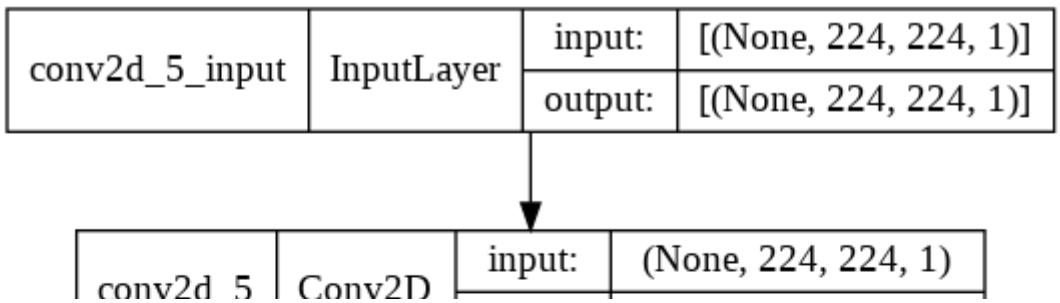
```
8711 - val_loss: 0.0816 - val_accuracy: 0.9848
Epoch 14/30
13/13 [==============================] - 4s 342ms/step - loss: 0.3299 - accuracy: 0.
8557 - val_loss: 0.0331 - val_accuracy: 0.9848
Epoch 15/30
13/13 [==============================] - 4s 339ms/step - loss: 0.2366 - accuracy: 0.
9227 - val_loss: 0.0438 - val_accuracy: 0.9848
Epoch 16/30
13/13 [==============================] - 4s 344ms/step - loss: 0.2568 - accuracy: 0.
8969 - val_loss: 0.0800 - val_accuracy: 1.0000
Epoch 17/30
13/13 [==============================] - 4s 340ms/step - loss: 0.2387 - accuracy: 0.
9227 - val_loss: 0.0295 - val_accuracy: 1.0000
Epoch 18/30
13/13 [==============================] - 4s 343ms/step - loss: 0.2283 - accuracy: 0.
9124 - val_loss: 0.0604 - val_accuracy: 0.9848
Epoch 19/30
13/13 [==============================] - 4s 336ms/step - loss: 0.2062 - accuracy: 0.
9175 - val_loss: 0.0311 - val_accuracy: 1.0000
Epoch 20/30
13/13 [==============================] - 4s 340ms/step - loss: 0.1566 - accuracy: 0.
9433 - val_loss: 0.1889 - val_accuracy: 0.8939
Epoch 21/30
13/13 [==============================] - 4s 334ms/step - loss: 0.2309 - accuracy: 0.
9175 - val_loss: 0.0163 - val_accuracy: 1.0000
Epoch 22/30
13/13 [==============================] - 4s 336ms/step - loss: 0.2841 - accuracy: 0.
8814 - val_loss: 0.0724 - val_accuracy: 1.0000
Epoch 23/30
13/13 [==============================] - 4s 341ms/step - loss: 0.2670 - accuracy: 0.
8969 - val_loss: 0.0373 - val_accuracy: 0.9848
Epoch 24/30
13/13 [==============================] - 4s 333ms/step - loss: 0.1540 - accuracy: 0.
9330 - val_loss: 0.0266 - val_accuracy: 1.0000
Epoch 25/30
13/13 [==============================] - 4s 339ms/step - loss: 0.1782 - accuracy: 0.
9227 - val_loss: 0.0114 - val_accuracy: 1.0000
Epoch 26/30
13/13 [==============================] - 4s 345ms/step - loss: 0.1457 - accuracy: 0.
9433 - val_loss: 0.0306 - val_accuracy: 1.0000
Epoch 27/30
13/13 [==============================] - 4s 347ms/step - loss: 0.1486 - accuracy: 0.
9485 - val_loss: 0.0280 - val_accuracy: 0.9848
Epoch 28/30
13/13 [==============================] - 4s 331ms/step - loss: 0.1935 - accuracy: 0.
9175 - val_loss: 0.0205 - val_accuracy: 1.0000
Epoch 29/30
13/13 [==============================] - 4s 345ms/step - loss: 0.1131 - accuracy: 0.
9639 - val_loss: 0.0283 - val_accuracy: 0.9848
Epoch 30/30
13/13 [==============================] - 4s 335ms/step - loss: 0.0745 - accuracy: 0.
9845 - val_loss: 0.0086 - val_accuracy: 1.0000
```

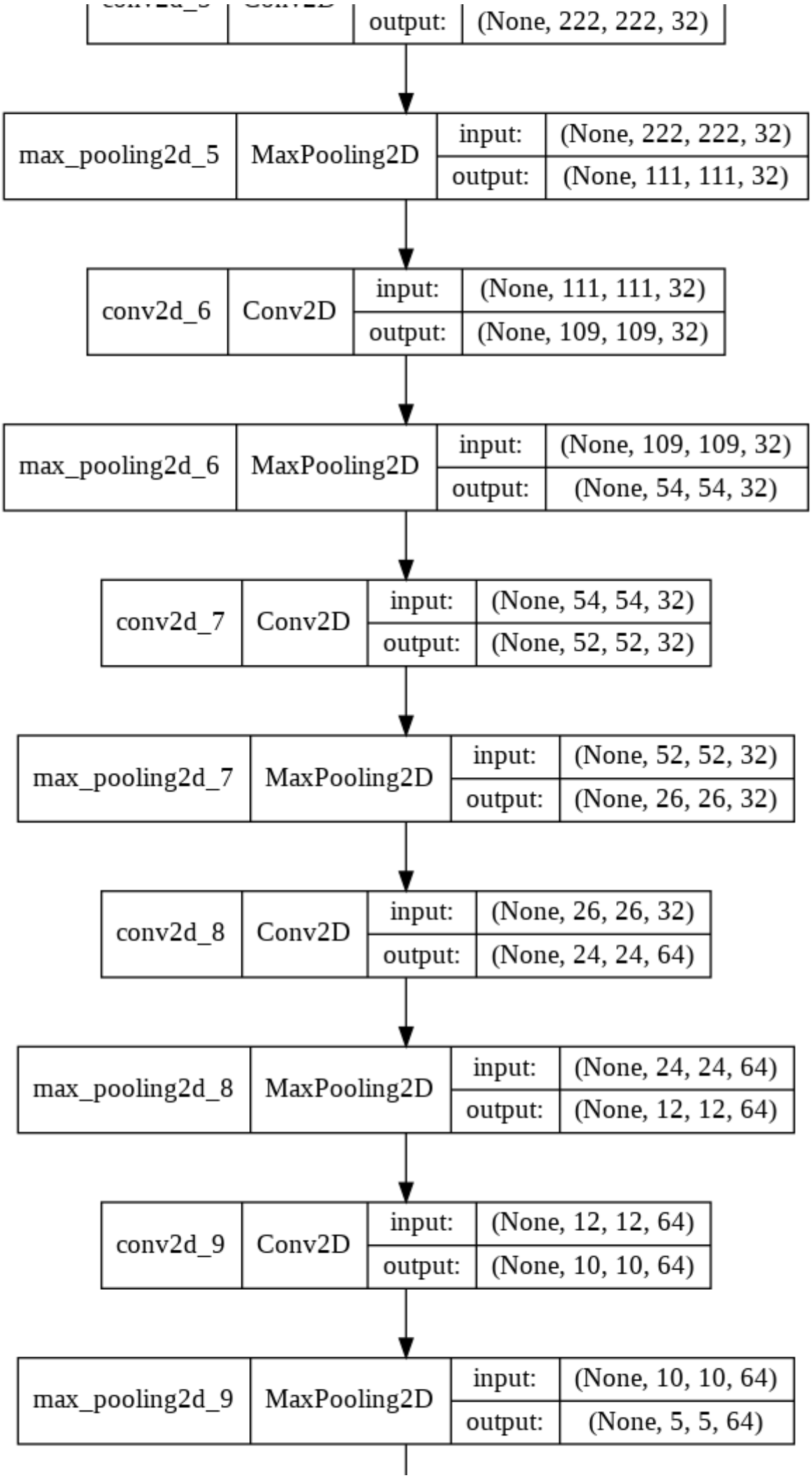
Visualizing CNN model

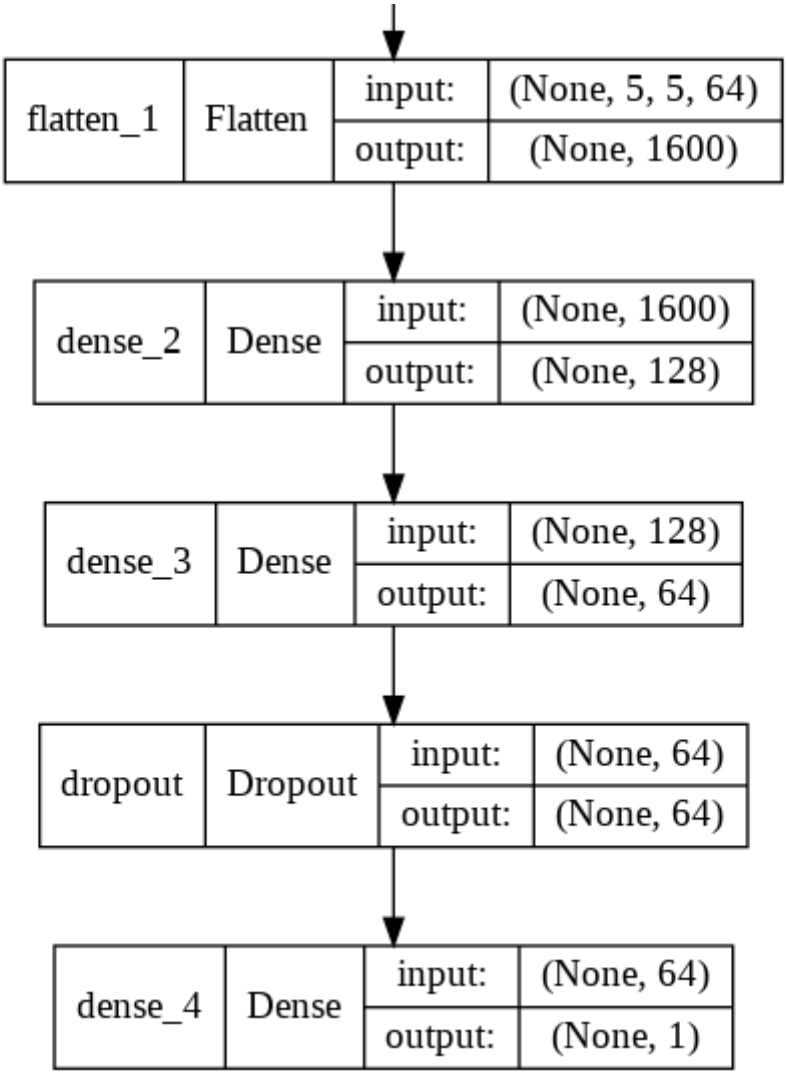
In [ ]:

```
from tensorflow.keras.utils import plot_model
plot_model(cnn,show_shapes=True, show_layer_names=True, rankdir='TB', expand_nested=
```

Out[ ]:




output: (None, 222, 222, 32)
max_pooling2d_5 MaxPooling2D input: (None, 222, 222, 32) output: (None, 111, 111, 32)
conv2d_6 Conv2D input: (None, 111, 111, 32) output: (None, 109, 109, 32)
max_pooling2d_6 MaxPooling2D input: (None, 109, 109, 32) output: (None, 54, 54, 32)
conv2d_7 Conv2D input: (None, 54, 54, 32) output: (None, 52, 52, 32)
max_pooling2d_7 MaxPooling2D input: (None, 52, 52, 32) output: (None, 26, 26, 32)
conv2d_8 Conv2D input: (None, 26, 26, 32) output: (None, 24, 24, 64)
max_pooling2d_8 MaxPooling2D input: (None, 24, 24, 64) output: (None, 12, 12, 64)
conv2d_9 Conv2D input: (None, 12, 12, 64) output: (None, 10, 10, 64)
max_pooling2d_9 MaxPooling2D input: (None, 10, 10, 64) output: (None, 5, 5, 64)



Prediction and Results

In [ ]:
```
test_accu = cnn.evaluate(test_data_gen)
print('The testing accuracy is :',test_accu[1]*100, '%')
```

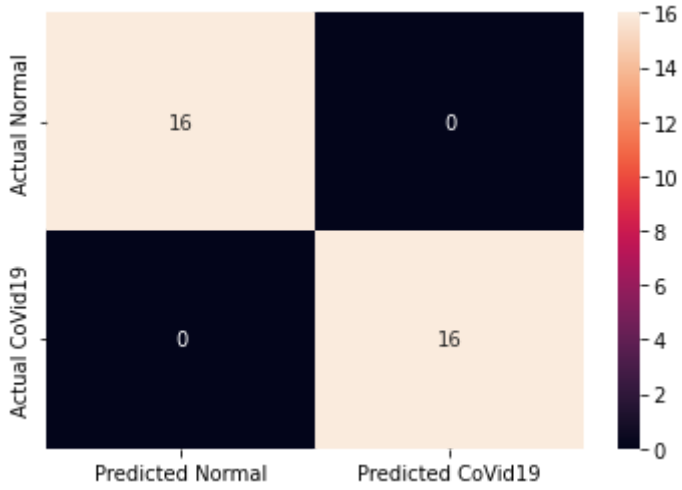
```
3/3 [==============================] - 12s 6s/step - loss: 0.1756 - accuracy: 0.9688
The testing accuracy is : 96.875 %
```

In [ ]:
```
preds = cnn.predict(test_data_gen,verbose=1)
predictions = preds.copy()
predictions[predictions <= 0.5] = 0
predictions[predictions > 0.5] = 1
```

```
3/3 [==============================] - 1s 233ms/step
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix

cm = pd.DataFrame(data=confusion_matrix(test_data_gen.classes, predictions, labels=[
columns=["Predicted Normal", "Predicted CoVid19"])

import seaborn as sns
sns.heatmap(cm,annot=True,fmt="d")
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f48df843810>
```



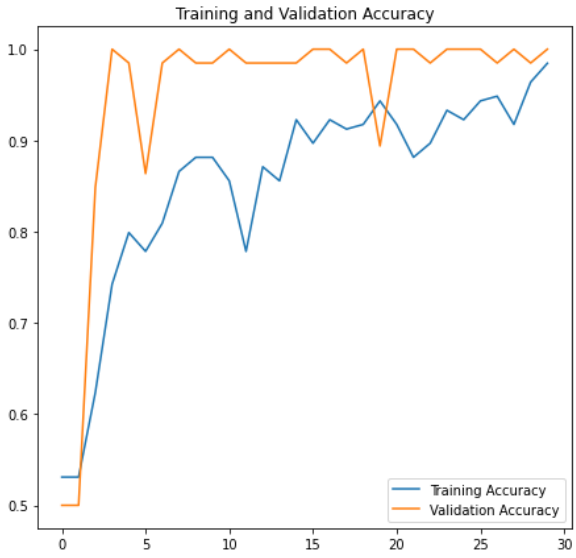
In [ ]:

```
print(classification_report(y_true=test_data_gen.classes,y_pred=predictions,target_n
```

```
              precision    recall  f1-score   support

      NORMAL       1.00      1.00      1.00        16
    CoVid19        1.00      1.00      1.00        16

    accuracy                           1.00        32
   macro avg       1.00      1.00      1.00        32
weighted avg       1.00      1.00      1.00        32
```

In [ ]:

```
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=(16, 7))
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='lower 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='upper right')
plt.title('Training and Validation Loss')
plt.savefig('./foo3.png')
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30


Training and Validation Loss
Training Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30

In [ ]:
```
## Authenticate and mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
%cd /content/drive/My Drive/X_ray_photos/
```

```
/content/drive/My Drive/X_ray_photos
```

In [ ]:
```
!ls
```

```
covid19  foo2.png  foo.png  model.png  normal  test  train  validation
```

Importing libraries

In [ ]:
```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os
import glob
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
base_dir = !pwd
base_dir = base_dir[0]
print(base_dir)
print(type(base_dir))
```

```
/content/drive/My Drive/X_ray_photos
<class 'str'>
```

In [ ]:
```
# classes = ['covid19', 'normal']
```

In [ ]:
```
# # The code below creates a train and a val folder each containing 2 folders (one f
# # It then copies the images from the original folders to these new folders such th
# # to the training set and 25% of the images go into the validation set.

# for i in classes:

#   os.makedirs(base_dir +'/train/' + i)
#   os.makedirs(base_dir +'/validation/' + i)
#   source = base_dir + '/' + i
#   allFileNames = os.listdir(source)
#   np.random.shuffle(allFileNames)
#   test_ratio = 0.25

#   train_FileNames, test_FileNames = np.split(np.array(allFileNames),
#                                                          [int(len(allFileNames)* (1 -

#   train_FileNames = [source+'/'+ name for name in train_FileNames.tolist()]
#   test_FileNames = [source+'/' + name for name in test_FileNames.tolist()]

#   for name in train_FileNames:
```

```
#     shutil.copy(name, base_dir +'/train/' + i)

#   for name in test_FileNames:
#     shutil.copy(name, base_dir +'/validation/' + i)
```

In [ ]:
```
!ls
```

```
covid19  foo.png  model.png  normal  test  train  validation
```

In [ ]:
```
# Setting paths for training an validation sets
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

In [ ]:
```
print(train_dir)
print(val_dir)
print(test_dir)
print(type(train_dir))
```

```
/content/drive/My Drive/X_ray_photos/train
/content/drive/My Drive/X_ray_photos/validation
/content/drive/My Drive/X_ray_photos/test
<class 'str'>
```

In [ ]:
```
BATCH_SIZE = 15  # Number of training examples to process before updating our models
IMG_SHAPE  = 224  # Our training data consists of images with width of 180 pixels an
```

In [ ]:
```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
      img = img.reshape((224,224))
      ax.imshow(img, cmap='gray')
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
# Read images from the disk.
# Decode contents of these images and convert it into proper grid format as per thei
# Convert them into floating point tensors.
# Rescale the tensors from values between 0 and 255 to values between 0 and 1,
# as neural networks prefer to deal with small input values.
image_gen_train = ImageDataGenerator(rescale=1./255,
                    rotation_range=45,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )

train_data_gen = image_gen_train.flow_from_directory(
                                                batch_size=BATCH_SIZE,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE, IMG_SHAPE),
                                                color_mode='grayscale',
                                                class_mode='binary'
                                                )
```

```
Found 194 images belonging to 2 classes.
```

In [ ]:
```
type(train_data_gen)
print(len(train_data_gen))
```

```
10
```

In [ ]:
```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
```

In [ ]:
```
# plotImages(augmented_images)
```

In [ ]:
```
image_gen_val = ImageDataGenerator(rescale=1./255)

val_data_gen = image_gen_val.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=val_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 color_mode='grayscale',
                                                 class_mode='binary')

test_data_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=test_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 color_mode='grayscale',
                                                 class_mode='binary')
```

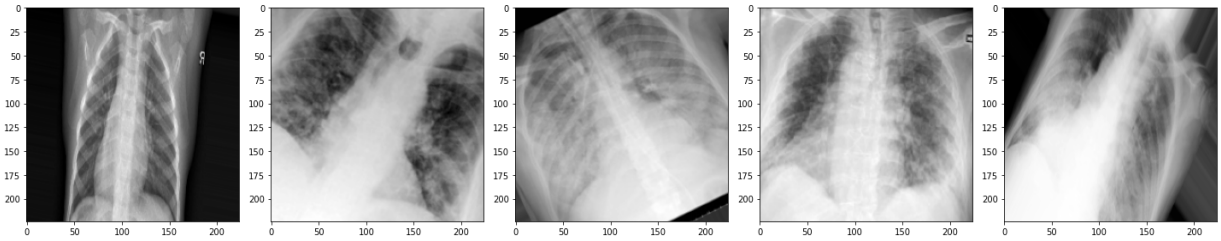
```
Found 66 images belonging to 2 classes.
Found 32 images belonging to 2 classes.
```

In [ ]:
```
len(val_data_gen)
```

Out[ ]:
```
4
```

In [ ]:
```
sample_training_images, _ = next(train_data_gen)

# next function returns a batch from the dataset. One batch is a tuple of (many imag
# For right now, we're discarding the labels because we just want to look at the ima
```

In [ ]:
```
plotImages(sample_training_images[:5])  # Plot images 0-4
```



Creating model

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, Activation
import math
```

In [ ]:
```
```

```
cnn = Sequential()

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SHAPE, IMG_SHAPE, 1))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(32, (3, 3), input_shape=(IMG_SHAPE, IMG_SHAPE, 1), use_bias=False))
cnn.add(BatchNormalization(scale=False, center=True))
cnn.add(Activation('relu'))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), input_shape=(IMG_SHAPE, IMG_SHAPE, 1), use_bias=False))
cnn.add(BatchNormalization(scale=False, center=True))
cnn.add(Activation('relu'))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Conv2D(64, (3, 3), input_shape=(IMG_SHAPE, IMG_SHAPE, 1), use_bias=False))
cnn.add(BatchNormalization(scale=False, center=True))
cnn.add(Activation('relu'))
cnn.add(MaxPooling2D(pool_size = (2, 2)))

cnn.add(Flatten())

cnn.add(Dense(units = 128, use_bias=False))
cnn.add(BatchNormalization(scale=False, center=True))
cnn.add(Activation('relu'))

cnn.add(Dense(activation = 'relu', units = 64))
cnn.add(Dropout(0.5))

cnn.add(Dense(activation = 'sigmoid', units = 1))
```

In [ ]:

```
cnn.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 222, 222, 32)      320

 max_pooling2d_20 (MaxPoolin  (None, 111, 111, 32)     0
 g2D)

 conv2d_21 (Conv2D)          (None, 109, 109, 32)      9248

 max_pooling2d_21 (MaxPoolin  (None, 54, 54, 32)       0
 g2D)

 conv2d_22 (Conv2D)          (None, 52, 52, 32)        9216

 batch_normalization_5 (Batc  (None, 52, 52, 32)       96
 hNormalization)

 activation_5 (Activation)   (None, 52, 52, 32)        0

 max_pooling2d_22 (MaxPoolin  (None, 26, 26, 32)       0
 g2D)

 conv2d_23 (Conv2D)          (None, 24, 24, 64)        18432

 batch_normalization_6 (Batc  (None, 24, 24, 64)       192
 hNormalization)
```

```
activation_6 (Activation)   (None, 24, 24, 64)        0

max_pooling2d_23 (MaxPoolin  (None, 12, 12, 64)       0
g2D)

conv2d_24 (Conv2D)          (None, 10, 10, 64)        36864

batch_normalization_7 (Batc  (None, 10, 10, 64)       192
hNormalization)

activation_7 (Activation)   (None, 10, 10, 64)        0

max_pooling2d_24 (MaxPoolin  (None, 5, 5, 64)         0
g2D)

flatten_4 (Flatten)         (None, 1600)              0

dense_9 (Dense)             (None, 128)               204800

batch_normalization_8 (Batc  (None, 128)              384
hNormalization)

activation_8 (Activation)   (None, 128)               0

dense_10 (Dense)            (None, 64)                8256

dropout_2 (Dropout)         (None, 64)                0

dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 288,065
Trainable params: 287,489
Non-trainable params: 576
_________________________________________________________________
```

In [ ]:

```
# lr decay function
def lr_decay(epoch):
  return 0.01 * math.pow(0.6, epoch)

# lr schedule callback
lr_decay_callback = tf.keras.callbacks.LearningRateScheduler(lr_decay, verbose=True)

# important to see what you are doing
# plot_learning_rate(lr_decay, EPOCHS)
```

In [ ]:

```
cnn.compile(optimizer=tf.keras.optimizers.Adam(lr=0.666),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning:
The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [ ]:

```
EPOCHS = 25
history = cnn.fit_generator(
    train_data_gen,
    epochs=EPOCHS,
    validation_data=val_data_gen,
    callbacks=[lr_decay_callback]
    )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: UserWarning: `Model.
fit_generator` is deprecated and will be removed in a future version. Please use `Mo
```

```
del.fit`, which supports generators.

Epoch 00001: LearningRateScheduler setting learning rate to 0.01.
Epoch 1/25
13/13 [==============================] - 6s 364ms/step - loss: 0.4915 - accuracy: 0.
7784 - val_loss: 43.8554 - val_accuracy: 0.5000 - lr: 0.0100

Epoch 00002: LearningRateScheduler setting learning rate to 0.006.
Epoch 2/25
13/13 [==============================] - 4s 338ms/step - loss: 0.2880 - accuracy: 0.
8711 - val_loss: 28.0572 - val_accuracy: 0.5000 - lr: 0.0060

Epoch 00003: LearningRateScheduler setting learning rate to 0.0036.
Epoch 3/25
13/13 [==============================] - 4s 344ms/step - loss: 0.1898 - accuracy: 0.
9485 - val_loss: 24.7326 - val_accuracy: 0.5000 - lr: 0.0036

Epoch 00004: LearningRateScheduler setting learning rate to 0.0021599999999999996.
Epoch 4/25
13/13 [==============================] - 4s 337ms/step - loss: 0.1693 - accuracy: 0.
9536 - val_loss: 20.4538 - val_accuracy: 0.5000 - lr: 0.0022

Epoch 00005: LearningRateScheduler setting learning rate to 0.001296.
Epoch 5/25
13/13 [==============================] - 4s 342ms/step - loss: 0.2842 - accuracy: 0.
9021 - val_loss: 16.0017 - val_accuracy: 0.5000 - lr: 0.0013

Epoch 00006: LearningRateScheduler setting learning rate to 0.0007775999999999998.
Epoch 6/25
13/13 [==============================] - 4s 346ms/step - loss: 0.1769 - accuracy: 0.
9536 - val_loss: 12.8258 - val_accuracy: 0.5000 - lr: 7.7760e-04

Epoch 00007: LearningRateScheduler setting learning rate to 0.0004665599999999999.
Epoch 7/25
13/13 [==============================] - 4s 344ms/step - loss: 0.0864 - accuracy: 0.
9691 - val_loss: 8.6854 - val_accuracy: 0.5000 - lr: 4.6656e-04

Epoch 00008: LearningRateScheduler setting learning rate to 0.00027993599999999994.
Epoch 8/25
13/13 [==============================] - 4s 349ms/step - loss: 0.0720 - accuracy: 0.
9794 - val_loss: 6.4101 - val_accuracy: 0.5000 - lr: 2.7994e-04

Epoch 00009: LearningRateScheduler setting learning rate to 0.00016796159999999993.
Epoch 9/25
13/13 [==============================] - 4s 335ms/step - loss: 0.1141 - accuracy: 0.
9691 - val_loss: 4.8966 - val_accuracy: 0.5000 - lr: 1.6796e-04

Epoch 00010: LearningRateScheduler setting learning rate to 0.00010077695999999997.
Epoch 10/25
13/13 [==============================] - 4s 337ms/step - loss: 0.1371 - accuracy: 0.
9588 - val_loss: 3.7813 - val_accuracy: 0.5000 - lr: 1.0078e-04

Epoch 00011: LearningRateScheduler setting learning rate to 6.046617599999974e-05.
Epoch 11/25
13/13 [==============================] - 4s 338ms/step - loss: 0.1113 - accuracy: 0.
9588 - val_loss: 2.7936 - val_accuracy: 0.5000 - lr: 6.0466e-05

Epoch 00012: LearningRateScheduler setting learning rate to 3.627970559999999e-05.
Epoch 12/25
13/13 [==============================] - 4s 341ms/step - loss: 0.1559 - accuracy: 0.
9433 - val_loss: 2.0348 - val_accuracy: 0.5758 - lr: 3.6280e-05

Epoch 00013: LearningRateScheduler setting learning rate to 2.1767823359999992e-05.
Epoch 13/25
13/13 [==============================] - 4s 335ms/step - loss: 0.1038 - accuracy: 0.
9639 - val_loss: 1.4486 - val_accuracy: 0.6364 - lr: 2.1768e-05

Epoch 00014: LearningRateScheduler setting learning rate to 1.3060694015999994e-05.
Epoch 14/25
```

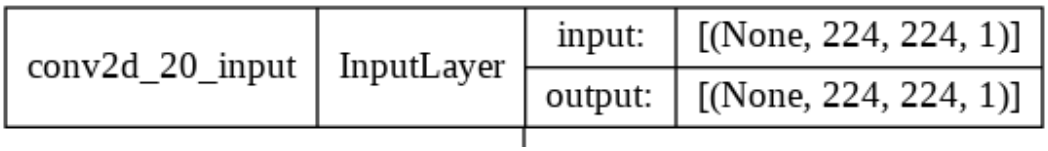
```
13/13 [==============================] - 4s 333ms/step - loss: 0.0699 - accuracy: 0.
9845 - val_loss: 1.0718 - val_accuracy: 0.6818 - lr: 1.3061e-05

Epoch 00015: LearningRateScheduler setting learning rate to 7.836416409599996e-06.
Epoch 15/25
13/13 [==============================] - 4s 338ms/step - loss: 0.1016 - accuracy: 0.
9742 - val_loss: 0.8041 - val_accuracy: 0.7424 - lr: 7.8364e-06

Epoch 00016: LearningRateScheduler setting learning rate to 4.701849845759998e-06.
Epoch 16/25
13/13 [==============================] - 4s 331ms/step - loss: 0.1426 - accuracy: 0.
9433 - val_loss: 0.6065 - val_accuracy: 0.7879 - lr: 4.7018e-06

Epoch 00017: LearningRateScheduler setting learning rate to 2.8211099074559985e-06.
Epoch 17/25
13/13 [==============================] - 4s 340ms/step - loss: 0.1143 - accuracy: 0.
9433 - val_loss: 0.4693 - val_accuracy: 0.8636 - lr: 2.8211e-06

Epoch 00018: LearningRateScheduler setting learning rate to 1.6926659444735988e-06.
Epoch 18/25
13/13 [==============================] - 4s 339ms/step - loss: 0.1175 - accuracy: 0.
9639 - val_loss: 0.3750 - val_accuracy: 0.8939 - lr: 1.6927e-06

Epoch 00019: LearningRateScheduler setting learning rate to 1.0155995666841593e-06.
Epoch 19/25
13/13 [==============================] - 4s 340ms/step - loss: 0.0742 - accuracy: 0.
9845 - val_loss: 0.3090 - val_accuracy: 0.9091 - lr: 1.0156e-06

Epoch 00020: LearningRateScheduler setting learning rate to 6.093597400104956e-07.
Epoch 20/25
13/13 [==============================] - 4s 334ms/step - loss: 0.1138 - accuracy: 0.
9691 - val_loss: 0.2584 - val_accuracy: 0.9091 - lr: 6.0936e-07

Epoch 00021: LearningRateScheduler setting learning rate to 3.6561584400629735e-07.
Epoch 21/25
13/13 [==============================] - 4s 336ms/step - loss: 0.1219 - accuracy: 0.
9588 - val_loss: 0.2185 - val_accuracy: 0.9091 - lr: 3.6562e-07

Epoch 00022: LearningRateScheduler setting learning rate to 2.193695064037784e-07.
Epoch 22/25
13/13 [==============================] - 4s 331ms/step - loss: 0.0775 - accuracy: 0.
9588 - val_loss: 0.1848 - val_accuracy: 0.9091 - lr: 2.1937e-07

Epoch 00023: LearningRateScheduler setting learning rate to 1.3162170384226703e-07.
Epoch 23/25
13/13 [==============================] - 4s 339ms/step - loss: 0.1146 - accuracy: 0.
9485 - val_loss: 0.1619 - val_accuracy: 0.9242 - lr: 1.3162e-07

Epoch 00024: LearningRateScheduler setting learning rate to 7.897302230536021e-08.
Epoch 24/25
13/13 [==============================] - 4s 336ms/step - loss: 0.1385 - accuracy: 0.
9588 - val_loss: 0.1413 - val_accuracy: 0.9242 - lr: 7.8973e-08

Epoch 00025: LearningRateScheduler setting learning rate to 4.738381338321613e-08.
Epoch 25/25
13/13 [==============================] - 4s 339ms/step - loss: 0.1306 - accuracy: 0.
9381 - val_loss: 0.1259 - val_accuracy: 0.9394 - lr: 4.7384e-08
```

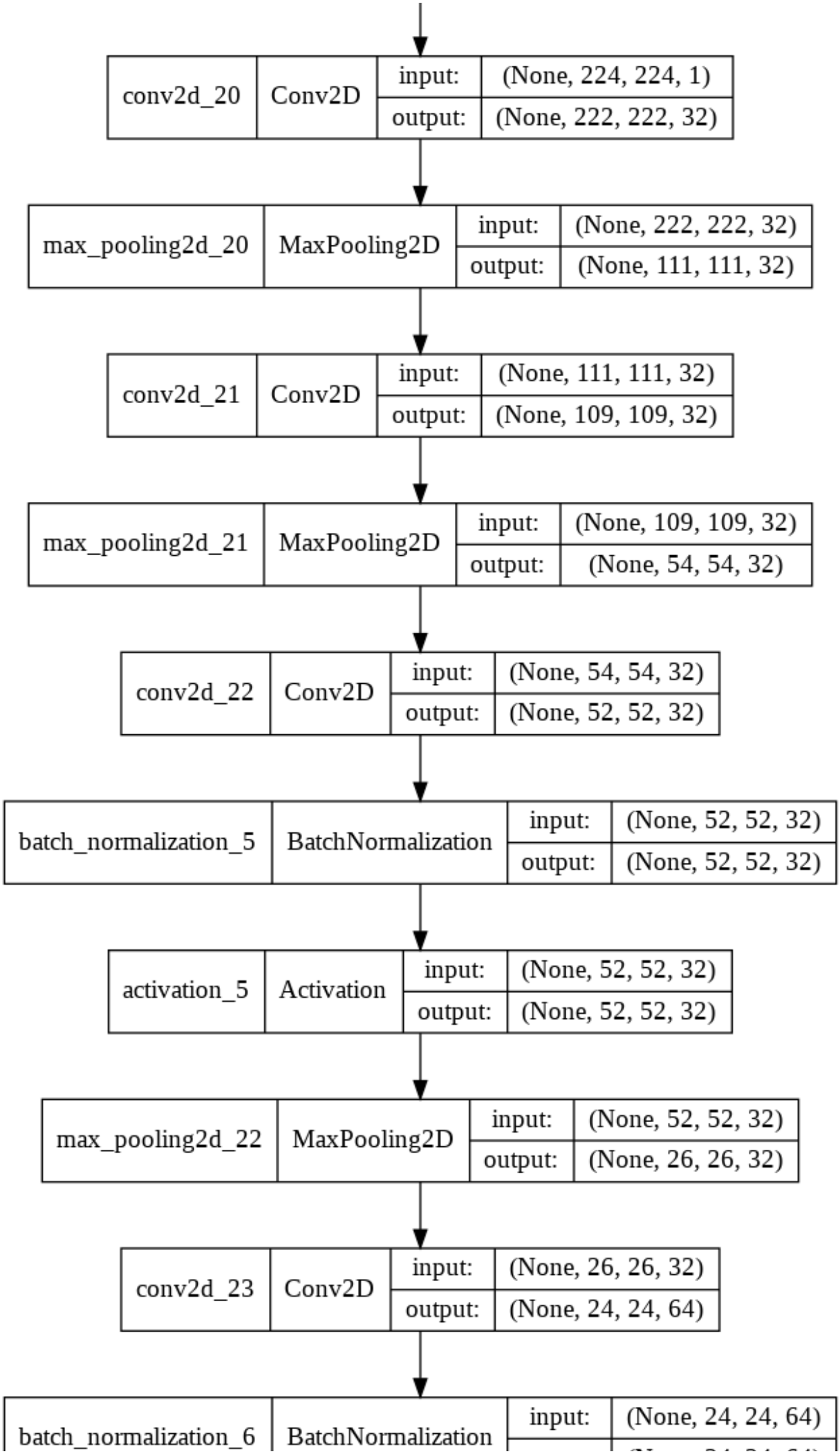
Visualizing CNN model

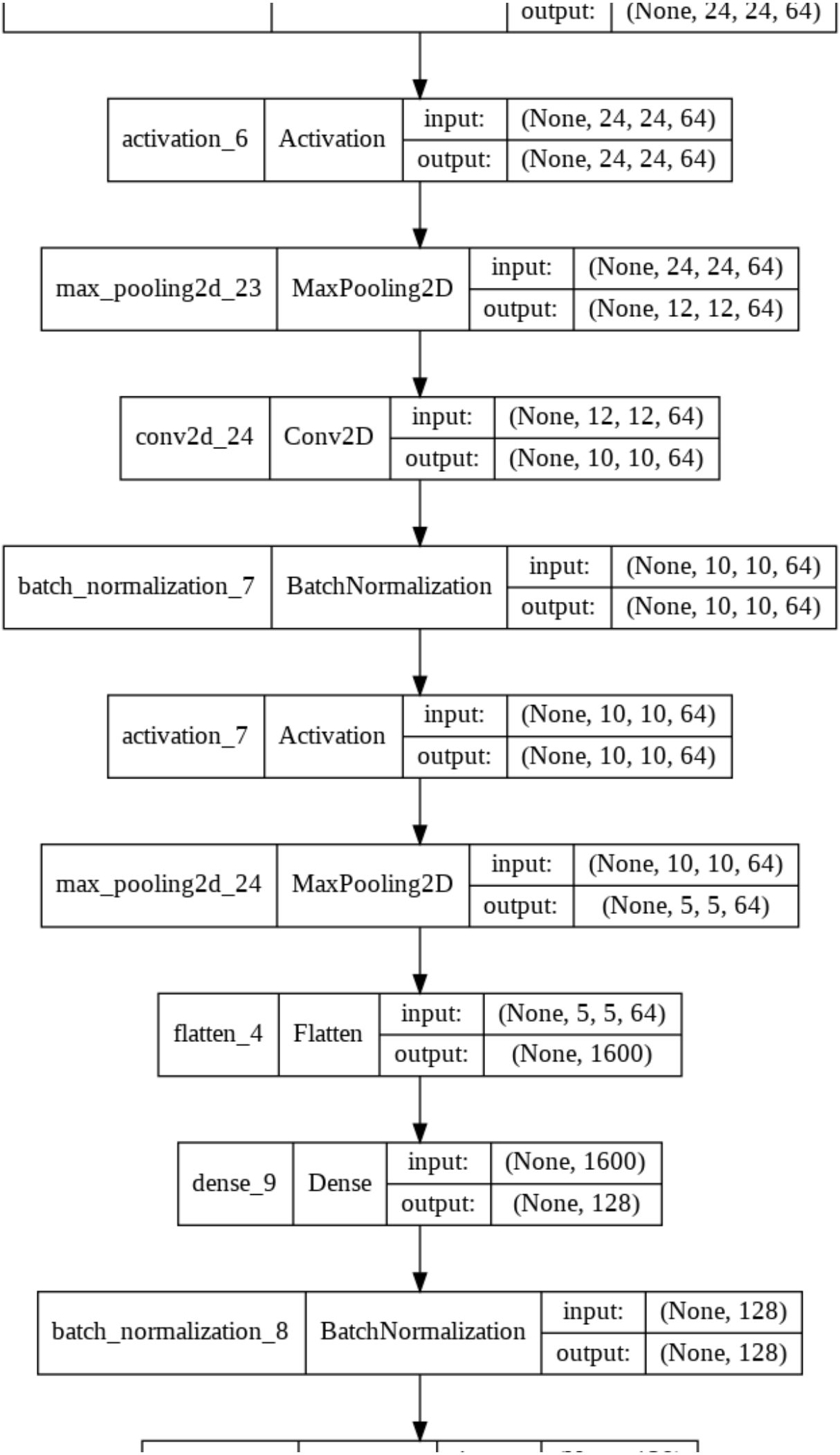
In [ ]:

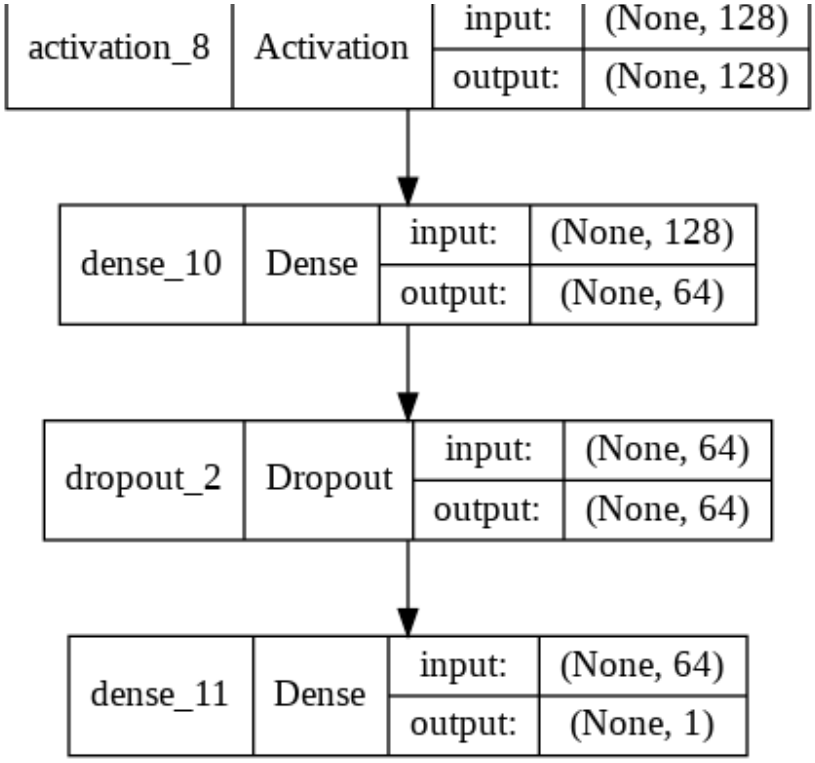
```
from tensorflow.keras.utils import plot_model
plot_model(cnn,show_shapes=True, show_layer_names=True, rankdir='TB', expand_nested=
```

Out[ ]:




conv2d_20
Conv2D
input: (None, 224, 224, 1)
output: (None, 222, 222, 32)
max_pooling2d_20
MaxPooling2D
input: (None, 222, 222, 32)
output: (None, 111, 111, 32)
conv2d_21
Conv2D
input: (None, 111, 111, 32)
output: (None, 109, 109, 32)
max_pooling2d_21
MaxPooling2D
input: (None, 109, 109, 32)
output: (None, 54, 54, 32)
conv2d_22
Conv2D
input: (None, 54, 54, 32)
output: (None, 52, 52, 32)
batch_normalization_5
BatchNormalization
input: (None, 52, 52, 32)
output: (None, 52, 52, 32)
activation_5
Activation
input: (None, 52, 52, 32)
output: (None, 52, 52, 32)
max_pooling2d_22
MaxPooling2D
input: (None, 52, 52, 32)
output: (None, 26, 26, 32)
conv2d_23
Conv2D
input: (None, 26, 26, 32)
output: (None, 24, 24, 64)
batch_normalization_6
BatchNormalization
input: (None, 24, 24, 64)


output: (None, 24, 24, 64)
activation_6 Activation input: (None, 24, 24, 64) output: (None, 24, 24, 64)
max_pooling2d_23 MaxPooling2D input: (None, 24, 24, 64) output: (None, 12, 12, 64)
conv2d_24 Conv2D input: (None, 12, 12, 64) output: (None, 10, 10, 64)
batch_normalization_7 BatchNormalization input: (None, 10, 10, 64) output: (None, 10, 10, 64)
activation_7 Activation input: (None, 10, 10, 64) output: (None, 10, 10, 64)
max_pooling2d_24 MaxPooling2D input: (None, 10, 10, 64) output: (None, 5, 5, 64)
flatten_4 Flatten input: (None, 5, 5, 64) output: (None, 1600)
dense_9 Dense input: (None, 1600) output: (None, 128)
batch_normalization_8 BatchNormalization input: (None, 128) output: (None, 128)



Prediction and Results

In [ ]:
```
test_accu = cnn.evaluate(test_data_gen)
print('The testing accuracy is :',test_accu[1]*100, '%')
```

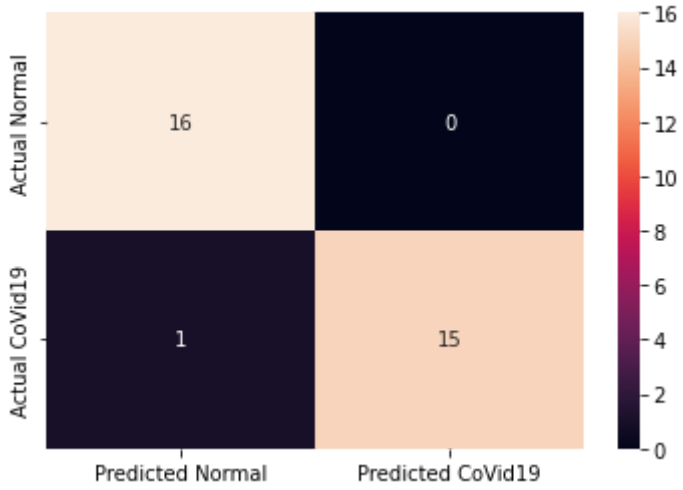
```
3/3 [==============================] - 1s 235ms/step - loss: 0.0979 - accuracy: 0.96
88
The testing accuracy is : 96.875 %
```

In [ ]:
```
preds = cnn.predict(test_data_gen,verbose=1)
predictions = preds.copy()
predictions[predictions <= 0.5] = 0
predictions[predictions > 0.5] = 1
```

```
3/3 [==============================] - 1s 225ms/step
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix

cm = pd.DataFrame(data=confusion_matrix(test_data_gen.classes, predictions, labels=[
columns=["Predicted Normal", "Predicted CoVid19"])

import seaborn as sns
sns.heatmap(cm,annot=True,fmt="d")
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49c0279e10>
```



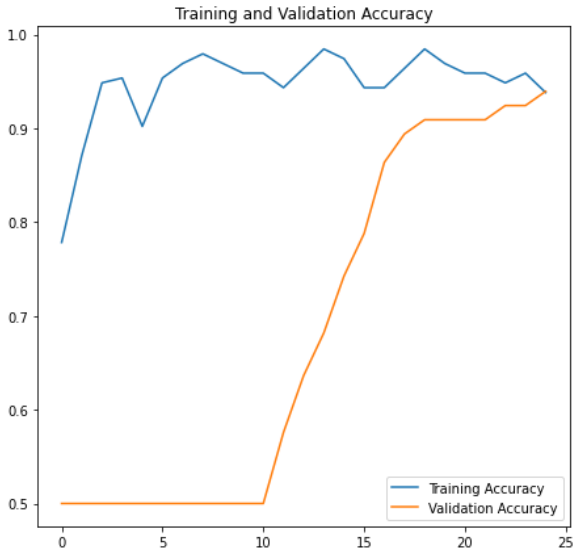
In [ ]:

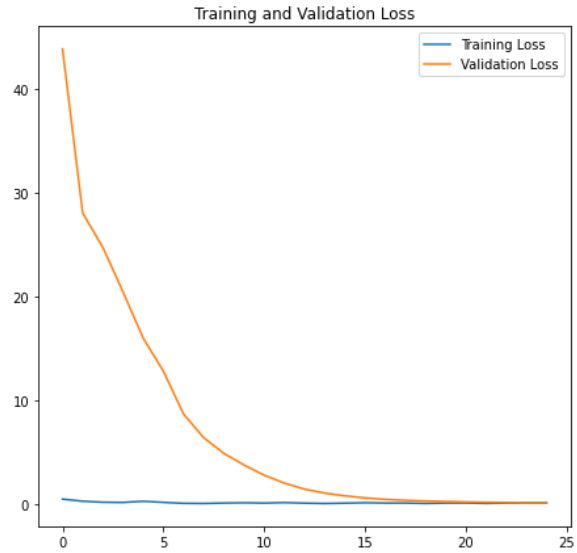
```
print(classification_report(y_true=test_data_gen.classes,y_pred=predictions,target_n
```

```
              precision    recall  f1-score   support

      NORMAL       0.94      1.00      0.97        16
    CoVid19        1.00      0.94      0.97        16

    accuracy                           0.97        32
   macro avg       0.97      0.97      0.97        32
weighted avg       0.97      0.97      0.97        32
```

In [ ]:

```
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=(16, 7))
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='lower 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='upper right')
plt.title('Training and Validation Loss')
plt.savefig('./foo4.png')
plt.show()
```


Training and Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
Training Accuracy
Validation Accuracy


Training and Validation Loss
40
30
20
10
0
0
5
10
15
20
25
Training Loss
Validation Loss

In [ ]:
```
## Authenticate and mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
%cd /content/drive/My Drive/X_ray_photos/
```

```
/content/drive/My Drive/X_ray_photos
```

In [ ]:
```
!ls
```

```
covid19  foo.png  model.png  normal  test  train  validation
```

Importing libraries

In [ ]:
```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16


import os
import glob
import shutil
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:
```
base_dir = !pwd
base_dir = base_dir[0]
print(base_dir)
print(type(base_dir))
```

```
/content/drive/My Drive/X_ray_photos
<class 'str'>
```

In [ ]:
```
# classes = ['covid19', 'normal']
```

In [ ]:
```
# # The code below creates a train and a val folder each containing 2 folders (one f
# # It then copies the images from the original folders to these new folders such th
# # to the training set and 25% of the images go into the validation set.

# for i in classes:

#   os.makedirs(base_dir +'/train/' + i)
#   os.makedirs(base_dir +'/validation/' + i)
#   source = base_dir + '/' + i
#   allFileNames = os.listdir(source)
#   np.random.shuffle(allFileNames)
#   test_ratio = 0.25

#   train_FileNames, test_FileNames = np.split(np.array(allFileNames),
#                                                         [int(len(allFileNames)* (1 -

#   train_FileNames = [source+'/'+ name for name in train_FileNames.tolist()]
#   test_FileNames = [source+'/' + name for name in test_FileNames.tolist()]
```

```
#   for name in train_FileNames:
#     shutil.copy(name, base_dir +'/train/' + i)

#   for name in test_FileNames:
#     shutil.copy(name, base_dir +'/validation/' + i)
```

In [ ]:
```
!ls
```

```
covid19  foo.png  model.png  normal  test  train  validation
```

In [ ]:
```
# Setting paths for training an validation sets
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')
```

In [ ]:
```
print(train_dir)
print(val_dir)
print(test_dir)
print(type(train_dir))
```

```
/content/drive/My Drive/X_ray_photos/train
/content/drive/My Drive/X_ray_photos/validation
/content/drive/My Drive/X_ray_photos/test
<class 'str'>
```

In [ ]:
```
BATCH_SIZE = 15  # Number of training examples to process before updating our models
IMG_SHAPE  = 224  # Our training data consists of images with width of 180 pixels an
```

In [ ]:
```
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
      img = img.reshape((224,224))
      ax.imshow(img, cmap='gray')
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
# Read images from the disk.
# Decode contents of these images and convert it into proper grid format as per thei
# Convert them into floating point tensors.
# Rescale the tensors from values between 0 and 255 to values between 0 and 1,
# as neural networks prefer to deal with small input values.
image_gen_train = ImageDataGenerator(rescale=1./255,
                    rotation_range=45,
                    horizontal_flip=True,
                    zoom_range=0.5
                    )

train_data_gen = image_gen_train.flow_from_directory(
                                                batch_size=BATCH_SIZE,
                                                directory=train_dir,
                                                shuffle=True,
                                                target_size=(IMG_SHAPE, IMG_SHAPE),
                                                class_mode='binary'
                                                )
```

```
Found 194 images belonging to 2 classes.
```

In [ ]:
```
print(train_data_gen[0][0].shape)
```

```
(15, 224, 224, 3)
```

In [ ]:
```
augmented_images = [train_data_gen[0][0][0] for i in range(5)]
```

In [ ]:
```
# plotImages(augmented_images)
```

In [ ]:
```
image_gen_val = ImageDataGenerator(rescale=1./255)

val_data_gen = image_gen_val.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=val_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 class_mode='binary')

test_data_gen = image_gen_train.flow_from_directory(batch_size=BATCH_SIZE,
                                                 directory=test_dir,
                                                 shuffle=False,
                                                 target_size=(IMG_SHAPE, IMG_SHAPE),
                                                 class_mode='binary')
```

```
Found 66 images belonging to 2 classes.
Found 32 images belonging to 2 classes.
```

In [ ]:
```
val_data_gen[2][1].shape
```

Out[ ]:
```
(15,)
```

In [ ]:
```
sample_training_images, _ = next(train_data_gen)

# next function returns a batch from the dataset. One batch is a tuple of (many imag
# For right now, we're discarding the labels because we just want to look at the ima
```

In [ ]:
```
# plotImages(sample_training_images[:5])  # Plot images 0-4
```

Creating model

In [ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization, Activation, AveragePooling2D
import math
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

In [ ]:
```
baseModel = VGG16(weights="imagenet", include_top=False, input_tensor=Input(shape=(2

# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
```

```
headModel = Dense(64, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(1, activation="sigmoid")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
        layer.trainable = False
```

In [ ]:

```
# compile our model
EPOCHS = 25
INIT_LR = 1e-3
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)

print("[INFO] compiling model...")

model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])
# train the head of the network

print("[INFO] training head...")
```

```
[INFO] compiling model...
[INFO] training head...
/usr/local/lib/python3.7/dist-packages/keras/optimizer_v2/adam.py:105: UserWarning:
The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [ ]:

```
history = model.fit_generator(
    train_data_gen,
    epochs=EPOCHS,
    validation_data=val_data_gen
    )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: UserWarning: `Model.
fit_generator` is deprecated and will be removed in a future version. Please use `Mo
del.fit`, which supports generators.
  after removing the cwd from sys.path.
Epoch 1/25
13/13 [==============================] - 101s 7s/step - loss: 0.6805 - accuracy: 0.5
876 - val_loss: 0.5809 - val_accuracy: 0.7727
Epoch 2/25
13/13 [==============================] - 7s 508ms/step - loss: 0.6291 - accuracy: 0.
6340 - val_loss: 0.5108 - val_accuracy: 0.9394
Epoch 3/25
13/13 [==============================] - 7s 507ms/step - loss: 0.5804 - accuracy: 0.
7113 - val_loss: 0.4702 - val_accuracy: 0.9091
Epoch 4/25
13/13 [==============================] - 7s 509ms/step - loss: 0.5013 - accuracy: 0.
8093 - val_loss: 0.4075 - val_accuracy: 0.9545
Epoch 5/25
13/13 [==============================] - 7s 517ms/step - loss: 0.4817 - accuracy: 0.
8299 - val_loss: 0.3754 - val_accuracy: 0.9545
Epoch 6/25
13/13 [==============================] - 7s 519ms/step - loss: 0.4457 - accuracy: 0.
8196 - val_loss: 0.3306 - val_accuracy: 0.9545
Epoch 7/25
13/13 [==============================] - 7s 520ms/step - loss: 0.4254 - accuracy: 0.
8402 - val_loss: 0.3189 - val_accuracy: 0.9545
Epoch 8/25
13/13 [==============================] - 7s 498ms/step - loss: 0.3769 - accuracy: 0.
8660 - val_loss: 0.2731 - val_accuracy: 0.9697
```

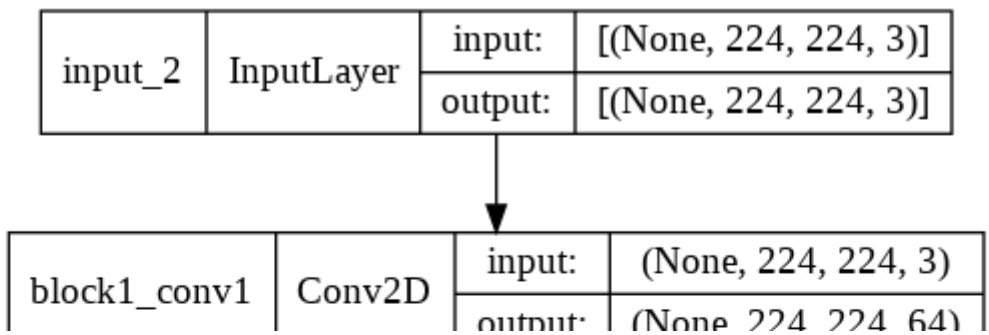
```
Epoch 9/25
13/13 [==============================] - 7s 513ms/step - loss: 0.3447 - accuracy: 0.
8763 - val_loss: 0.2585 - val_accuracy: 0.9697
Epoch 10/25
13/13 [==============================] - 7s 512ms/step - loss: 0.3506 - accuracy: 0.
8505 - val_loss: 0.2825 - val_accuracy: 0.9394
Epoch 11/25
13/13 [==============================] - 7s 511ms/step - loss: 0.3528 - accuracy: 0.
8711 - val_loss: 0.2266 - val_accuracy: 0.9697
Epoch 12/25
13/13 [==============================] - 7s 510ms/step - loss: 0.3326 - accuracy: 0.
8814 - val_loss: 0.2289 - val_accuracy: 0.9697
Epoch 13/25
13/13 [==============================] - 7s 504ms/step - loss: 0.3335 - accuracy: 0.
8557 - val_loss: 0.1957 - val_accuracy: 0.9697
Epoch 14/25
13/13 [==============================] - 7s 524ms/step - loss: 0.3018 - accuracy: 0.
8660 - val_loss: 0.2027 - val_accuracy: 0.9697
Epoch 15/25
13/13 [==============================] - 7s 519ms/step - loss: 0.2862 - accuracy: 0.
8918 - val_loss: 0.1873 - val_accuracy: 0.9697
Epoch 16/25
13/13 [==============================] - 7s 522ms/step - loss: 0.2792 - accuracy: 0.
9072 - val_loss: 0.1758 - val_accuracy: 0.9697
Epoch 17/25
13/13 [==============================] - 7s 516ms/step - loss: 0.2209 - accuracy: 0.
9433 - val_loss: 0.1637 - val_accuracy: 0.9697
Epoch 18/25
13/13 [==============================] - 7s 516ms/step - loss: 0.3000 - accuracy: 0.
8711 - val_loss: 0.1777 - val_accuracy: 0.9697
Epoch 19/25
13/13 [==============================] - 7s 526ms/step - loss: 0.2389 - accuracy: 0.
9124 - val_loss: 0.1413 - val_accuracy: 0.9697
Epoch 20/25
13/13 [==============================] - 7s 508ms/step - loss: 0.2799 - accuracy: 0.
8866 - val_loss: 0.1449 - val_accuracy: 0.9697
Epoch 21/25
13/13 [==============================] - 7s 513ms/step - loss: 0.2650 - accuracy: 0.
9021 - val_loss: 0.1566 - val_accuracy: 0.9697
Epoch 22/25
13/13 [==============================] - 7s 512ms/step - loss: 0.2404 - accuracy: 0.
9021 - val_loss: 0.1332 - val_accuracy: 0.9697
Epoch 23/25
13/13 [==============================] - 7s 514ms/step - loss: 0.2078 - accuracy: 0.
9227 - val_loss: 0.1207 - val_accuracy: 0.9697
Epoch 24/25
13/13 [==============================] - 7s 508ms/step - loss: 0.2084 - accuracy: 0.
9330 - val_loss: 0.1565 - val_accuracy: 0.9697
Epoch 25/25
13/13 [==============================] - 7s 521ms/step - loss: 0.2451 - accuracy: 0.
9072 - val_loss: 0.1157 - val_accuracy: 0.9697
```

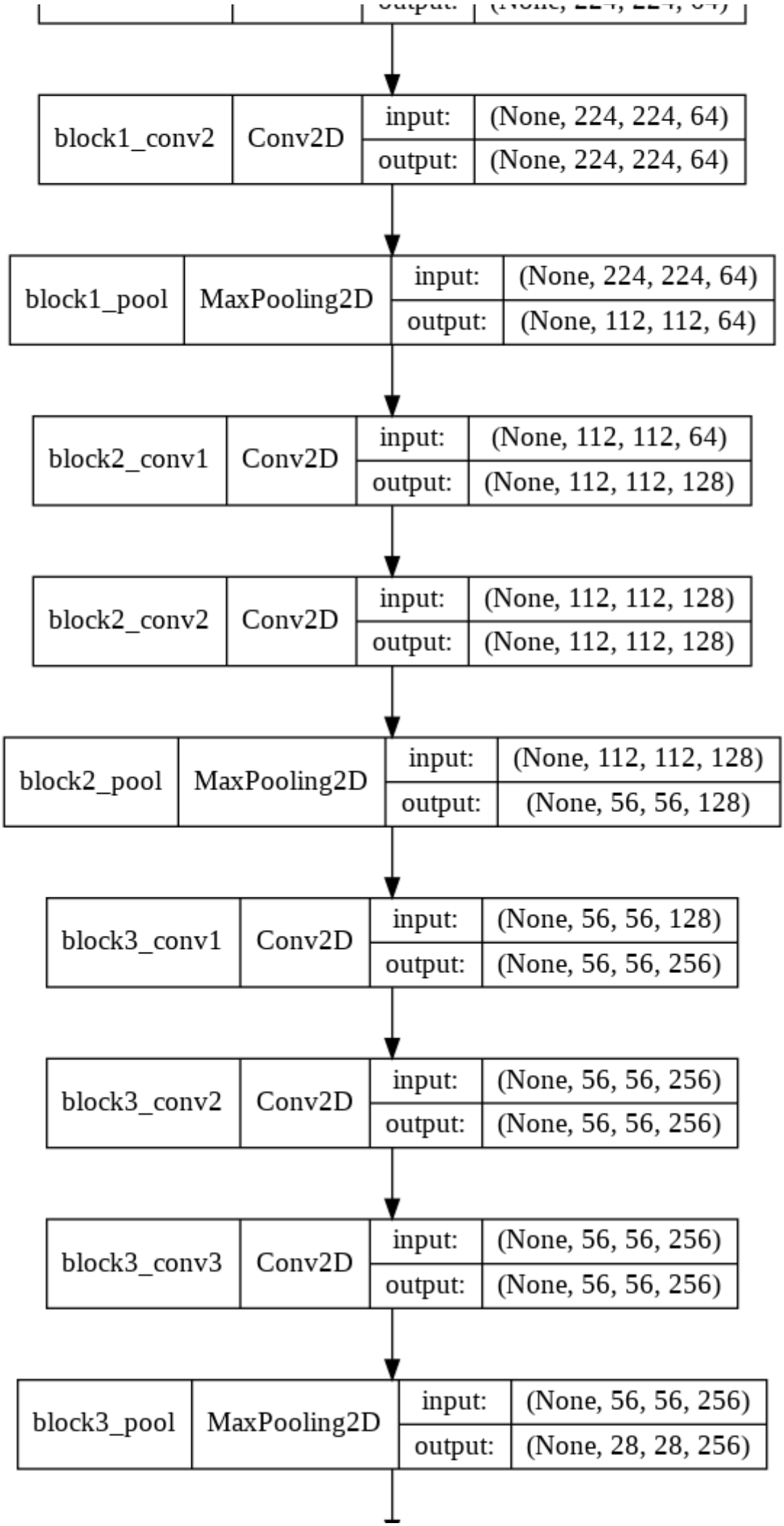
Visualizing CNN model

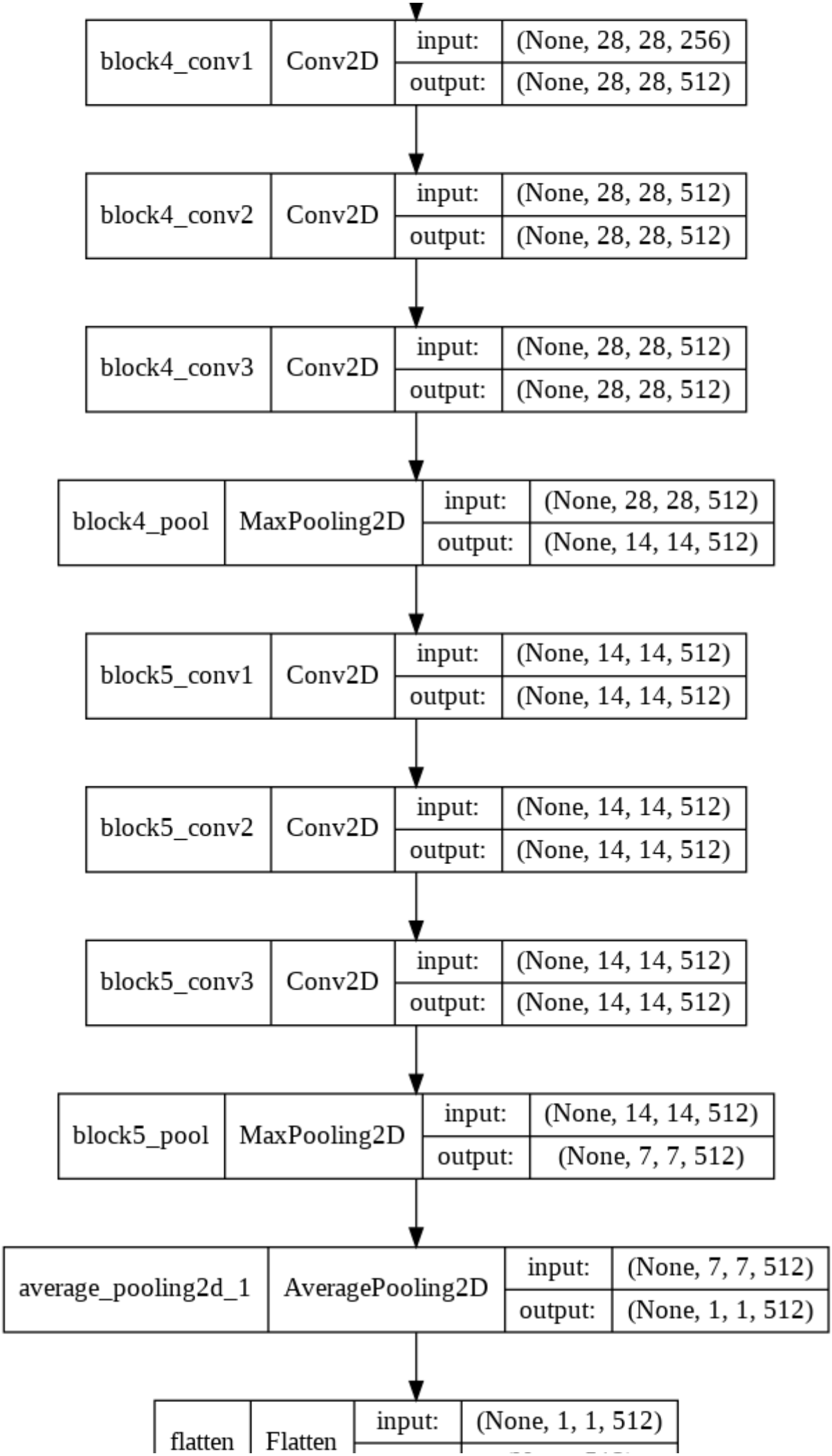
In [ ]:

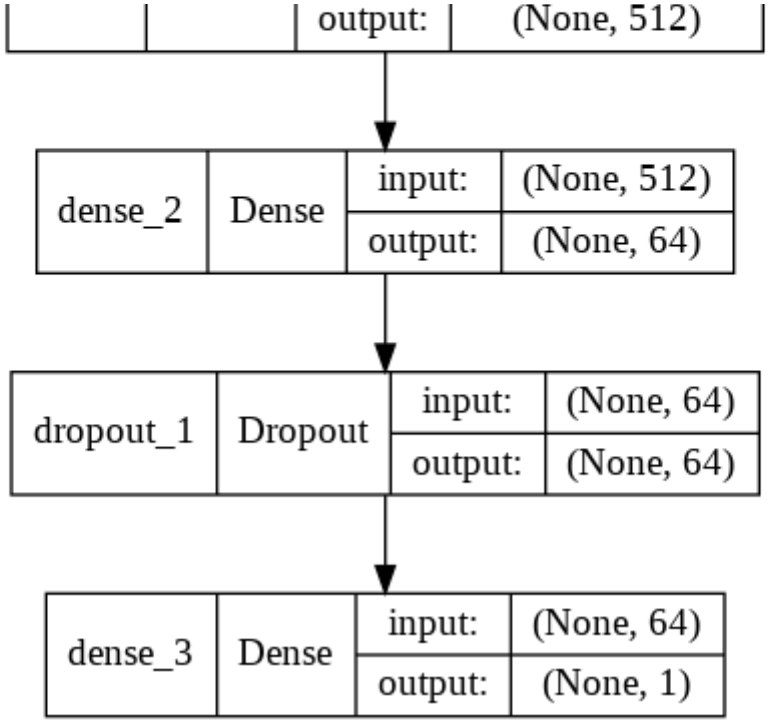
```
from tensorflow.keras.utils import plot_model
plot_model(model,show_shapes=True, show_layer_names=True, rankdir='TB', expand_neste
```

Out[ ]:




block1_conv2 Conv2D input: (None, 224, 224, 64) output: (None, 224, 224, 64)
block1_pool MaxPooling2D input: (None, 224, 224, 64) output: (None, 112, 112, 64)
block2_conv1 Conv2D input: (None, 112, 112, 64) output: (None, 112, 112, 128)
block2_conv2 Conv2D input: (None, 112, 112, 128) output: (None, 112, 112, 128)
block2_pool MaxPooling2D input: (None, 112, 112, 128) output: (None, 56, 56, 128)
block3_conv1 Conv2D input: (None, 56, 56, 128) output: (None, 56, 56, 256)
block3_conv2 Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 256)
block3_conv3 Conv2D input: (None, 56, 56, 256) output: (None, 56, 56, 256)
block3_pool MaxPooling2D input: (None, 56, 56, 256) output: (None, 28, 28, 256)


block4_conv1 Conv2D input: (None, 28, 28, 256) output: (None, 28, 28, 512)
block4_conv2 Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 512)
block4_conv3 Conv2D input: (None, 28, 28, 512) output: (None, 28, 28, 512)
block4_pool MaxPooling2D input: (None, 28, 28, 512) output: (None, 14, 14, 512)
block5_conv1 Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv2 Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_conv3 Conv2D input: (None, 14, 14, 512) output: (None, 14, 14, 512)
block5_pool MaxPooling2D input: (None, 14, 14, 512) output: (None, 7, 7, 512)
average_pooling2d_1 AveragePooling2D input: (None, 7, 7, 512) output: (None, 1, 1, 512)
flatten Flatten input: (None, 1, 1, 512)



Prediction and Results

In [ ]:
```
test_accu = model.evaluate(test_data_gen)
print('The testing accuracy is :',test_accu[1]*100, '%')
```

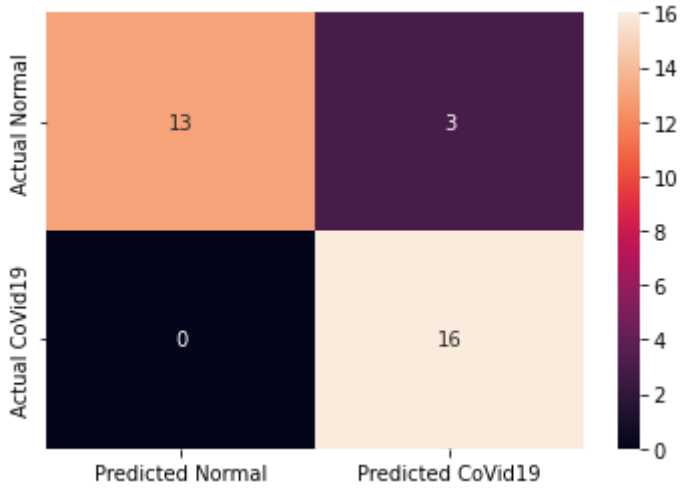
```
3/3 [==============================] - 10s 5s/step - loss: 0.2084 - accuracy: 0.9062
The testing accuracy is : 90.625 %
```

In [ ]:
```
preds = model.predict(test_data_gen,verbose=1)
predictions = preds.copy()
predictions[predictions <= 0.5] = 0
predictions[predictions > 0.5] = 1
```

```
3/3 [==============================] - 1s 345ms/step
```

In [ ]:
```
from sklearn.metrics import classification_report,confusion_matrix

cm = pd.DataFrame(data=confusion_matrix(test_data_gen.classes, predictions, labels=[
columns=["Predicted Normal", "Predicted CoVid19"])

import seaborn as sns
sns.heatmap(cm,annot=True,fmt="d")
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa488d0ad90>
```



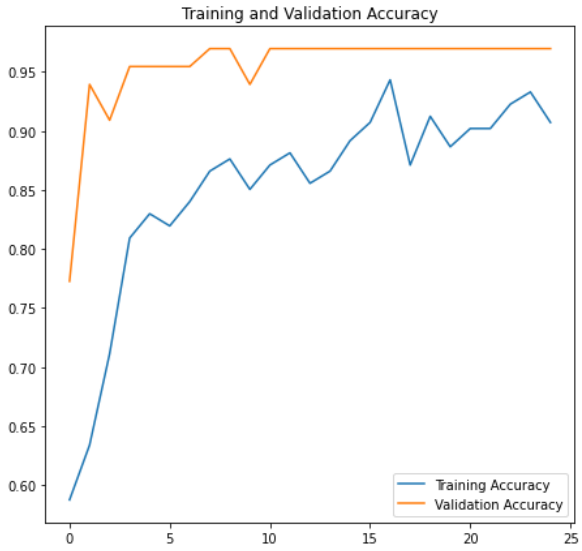
In [ ]:

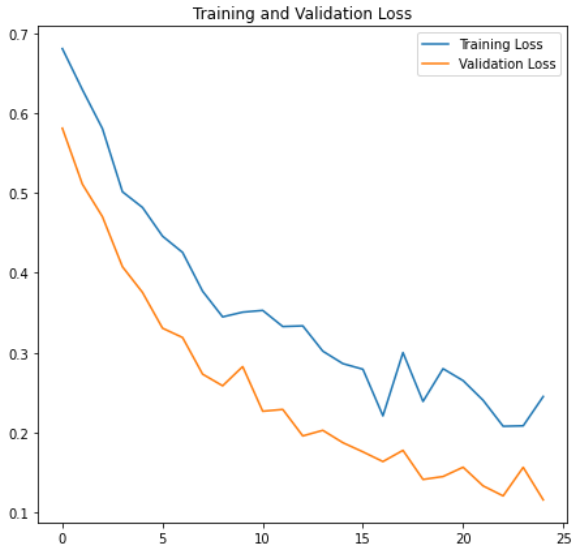
```
print(classification_report(y_true=test_data_gen.classes,y_pred=predictions,target_n
```

```
              precision    recall  f1-score   support

      NORMAL       1.00      0.81      0.90        16
    CoVid19        0.84      1.00      0.91        16

    accuracy                           0.91        32
   macro avg       0.92      0.91      0.91        32
weighted avg       0.92      0.91      0.91        32
```

In [ ]:

```
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=(16, 7))
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='lower 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='upper right')
plt.title('Training and Validation Loss')
plt.savefig('./foo2.png')
plt.show()
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
5
10
15
20
25


Training and Validation Loss
Training Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
5
10
15
20
25