Pneumonia Detection Inception

May 27, 2024

[ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras import layers
import tensorflow_addons as tfa
import pandas as pd
import json
import zipfile
import os
import seaborn as sns
import random
import shutil
import time
from PIL import Image
from matplotlib import pyplot as plt
from keras.models import Sequential, Model
from keras.applications import InceptionV3, Xception, InceptionResNetV2
from keras.applications.resnet import preprocess_input
from keras.layers import Dense, Conv2D, Flatten, MaxPooling2D,␣
↪Dropout,GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, TensorBoard
import wandb
!mkdir output
!mkdir output/tmp-augmented-images/
random.seed(123)
```

```
mkdir: output: File exists
mkdir: output/tmp-augmented-images/: File exists
```

[ ]:
```
def resample_data(move_from, move_to, cl, images_to_move=100):
  path = "./" + 'DATASET/data/pneumonia_data/'

  classes = os.listdir(path + move_from)

  cl += '/'
  curr_path = path + move_from + cl
  for _, _, files in os.walk(curr_path):
```

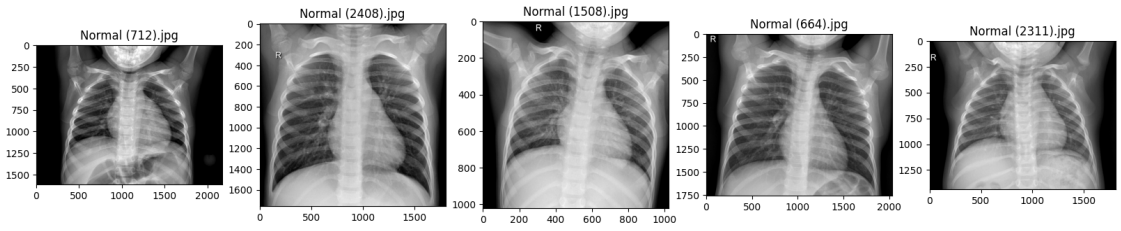
```
    random.shuffle(files)
    files_to_move = files[:images_to_move]
    for fn in files_to_move:
      shutil.move(curr_path + fn, path + move_to + cl + fn)
      #print('Moved ' + curr_path + fn)

  print('Resampled Images')


move_from, move_to = 'train/', 'test/'
#resample_data(move_from, move_to, 'PNEUMONIA', 100)
# Training images
print('Number of COVID training images:')
!ls DATASET/data/pneumonia_data/train/COVID_19/ | wc -l
print('Number of NORMAL training images:')
!ls DATASET/data/pneumonia_data/train/Normal/ | wc -l
print('Number of PNEUMONIA training images:')
!ls DATASET/data/pneumonia_data/train/Pneumonia// | wc -l
print()

# Validation images
print('Number of COVID training images:')
!ls DATASET/data/pneumonia_data/val/COVID_19/ | wc -l
print('Number of NORMAL validation images:')
!ls DATASET/data/pneumonia_data/val/Normal/ | wc -l
print('Number of PNEUMONIA validation images:')
!ls DATASET/data/pneumonia_data/val/Pneumonia/ | wc -l
print()


# Test images
#resample_data('test/', 'val/', 'PNEUMONIA', 2690)
print('Number of COVID test images:')
!ls DATASET/data/pneumonia_data/test/COVID_19/ | wc -l
print('Number of NORMAL test images:')
!ls DATASET/data/pneumonia_data/test/Normal/ | wc -l
print('Number of PNEUMONIA test images:')
!ls DATASET/data/pneumonia_data/test/Pneumonia/ | wc -l
```

```
Number of COVID training images:
    1000
Number of NORMAL training images:
    2925
Number of PNEUMONIA training images:
    3772

Number of COVID training images:
     171
```

```
Number of NORMAL validation images:
     235
Number of PNEUMONIA validation images:
     765

Number of COVID test images:
     110
Number of NORMAL test images:
     110
Number of PNEUMONIA test images:
     120
```

[ ]:
```python
def viewImagesFromDir(path, num=5):
  #Display num random images from dataset. Rerun cell for new random images.
  ↪The images are only single-channel

  img_paths_visualise = sorted(
        os.path.join(path, fname)
        for fname in os.listdir(path)
        if fname.endswith(".jpg")
  )

  random.shuffle(img_paths_visualise)

  fig, ax = plt.subplots(1, num, figsize=(20, 10))
  print(num)
  for i in range(num):
    ax[i].imshow(Image.open(img_paths_visualise[i]))
    index = img_paths_visualise[i].rfind('/') + 1
    ax[i].title.set_text(img_paths_visualise[i][index:])

  fig.canvas.draw()
  time.sleep(1)


viewImagesFromDir('DATASET/data/pneumonia_data/train/Normal/', num=5)
```

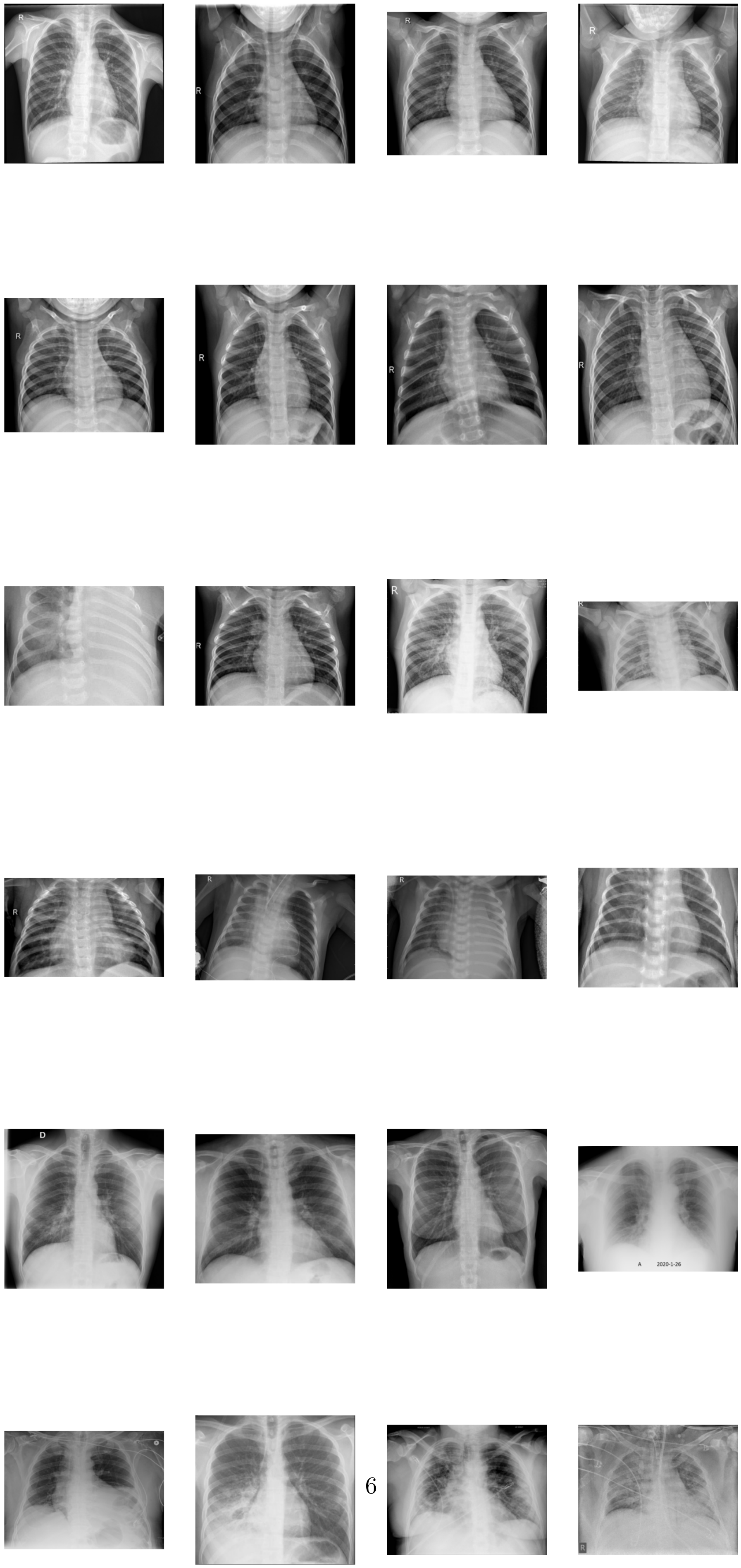
```
5
```



[ ]:
```
base_dir = 'DATASET/data/pneumonia_data'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'val')

# Directory with our training covid 19 pictures
train_covid_dir = os.path.join(train_dir, 'COVID_19')

# Directory with our training normal pictures
train_normal_dir = os.path.join(train_dir, 'NORMAL')

# Directory with our training pneumonia pictures
train_pneumonia_dir = os.path.join(train_dir, 'PNEUMONIA')

# Directory with our validation covid 19 pictures
validation_covid_dir = os.path.join(validation_dir, 'COVID_19')

# Directory with our validation normal pictures
validation_normal_dir = os.path.join(validation_dir, 'NORMAL')

# Directory with our validation pneumonia pictures
validation_pneumonia_dir = os.path.join(validation_dir, 'PNEUMONIA')
```

[ ]:
```
# Set up matplotlib fig, and size it to fit 4x4 pics
import matplotlib.image as mpimg
nrows = 6
ncols = 4

fig = plt.gcf()
fig.set_size_inches(ncols*4, nrows*6)
pic_index = 100
train_covid_fnames = os.listdir( train_covid_dir)
train_normal_fnames = os.listdir( train_normal_dir )
train_pneumonia_fnames = os.listdir( train_pneumonia_dir )

next_covid_pix = [os.path.join(train_covid_dir, fname)
                  for fname in train_covid_fnames[ pic_index-8:pic_index]
                 ]
next_normal_pix = [os.path.join(train_normal_dir, fname)
                  for fname in train_normal_fnames[ pic_index-8:pic_index]
                 ]

next_pneumonia_pix = [os.path.join(train_pneumonia_dir, fname)
                  for fname in train_pneumonia_fnames[ pic_index-8:pic_index]
                 ]

for i, img_path in enumerate(next_normal_pix+next_pneumonia_pix+next_covid_pix):
  # Set up subplot; subplot indices start at 1
```

```
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('Off') # Don't show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

plt.show()
```


A 2020-1-26

[ ]:
```
# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255., rotation_range = 40,␣
 ↪width_shift_range = 0.2, height_shift_range = 0.2,shear_range = 0.2,␣
 ↪zoom_range = 0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator( rescale = 1.0/255. )
```

[ ]:
```
train_generator = train_datagen.flow_from_directory(train_dir, batch_size = 32,␣
 ↪class_mode = 'categorical', target_size = (150, 150))
validation_generator = test_datagen.flow_from_directory(validation_dir,␣
 ↪batch_size = 32, class_mode = 'categorical', target_size = (150, 150))
```

```
Found 7697 images belonging to 3 classes.
Found 1171 images belonging to 3 classes.
```

[ ]:
```
from keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(input_shape = (150, 150, 3), include_top = False,␣
 ↪weights = 'imagenet')
```

[ ]:
```
for layer in base_model.layers:
    layer.trainable = False
```

[ ]:
```
from keras.optimizers import RMSprop
from keras.regularizers import l2

l2_lambda = 0.001  # Adjust this value as needed


x = layers.Flatten()(base_model.output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)

# Add a final sigmoid layer with 1 node for classification output
x = layers.Dense(3, activation='softmax')(x)

model = tf.keras.models.Model(base_model.input, x)

# Use the legacy Keras optimizer
optimizer_legacy = tf.keras.optimizers.legacy.RMSprop(learning_rate=0.0001)

model.compile(optimizer = optimizer_legacy, loss = 'categorical_crossentropy',␣
 ↪metrics = ['acc'])
```

[ ]:
```
# Import necessary libraries
from keras.callbacks import EarlyStopping, CSVLogger
```

```
# Add CSVLogger to save training history
csv_logger = CSVLogger('training_inception.log', append=True, separator=';')

# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True)

# Compile the model
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model with callbacks
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=20,  # Adjust the number of epochs as needed
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.
 ↪batch_size,
    callbacks=[csv_logger, early_stopping]
)

# Save the model in the native Keras format
model.save('best_inception_model.keras')
```

```
Epoch 1/20
240/240 [==============================] - 71s 288ms/step - loss: 8.5627 -
accuracy: 0.7533 - val_loss: 0.6382 - val_accuracy: 0.7847
Epoch 2/20
240/240 [==============================] - 68s 282ms/step - loss: 0.4808 -
accuracy: 0.8343 - val_loss: 0.4470 - val_accuracy: 0.8194
Epoch 3/20
240/240 [==============================] - 71s 294ms/step - loss: 0.4272 -
accuracy: 0.8638 - val_loss: 0.3602 - val_accuracy: 0.8819
Epoch 4/20
240/240 [==============================] - 72s 299ms/step - loss: 0.3659 -
accuracy: 0.8719 - val_loss: 0.3787 - val_accuracy: 0.8681
Epoch 5/20
240/240 [==============================] - 69s 289ms/step - loss: 0.3583 -
accuracy: 0.8781 - val_loss: 0.2502 - val_accuracy: 0.9054
Epoch 6/20
240/240 [==============================] - 69s 286ms/step - loss: 0.3372 -
accuracy: 0.8832 - val_loss: 0.2087 - val_accuracy: 0.9227
Epoch 7/20
240/240 [==============================] - 67s 280ms/step - loss: 0.3278 -
accuracy: 0.8903 - val_loss: 1.1043 - val_accuracy: 0.7378
```

```
Epoch 8/20
240/240 [==============================] - 68s 283ms/step - loss: 0.3332 -
accuracy: 0.8933 - val_loss: 0.4170 - val_accuracy: 0.8715
Epoch 9/20
240/240 [==============================] - 68s 282ms/step - loss: 0.3388 -
accuracy: 0.8891 - val_loss: 0.2624 - val_accuracy: 0.9193
```

[ ]:
```
history = history.history
```

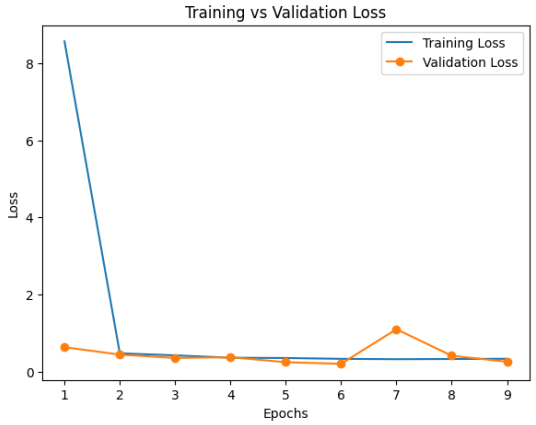
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[170], line 1
----> 1 history = history.history

AttributeError: 'dict' object has no attribute 'history'
```

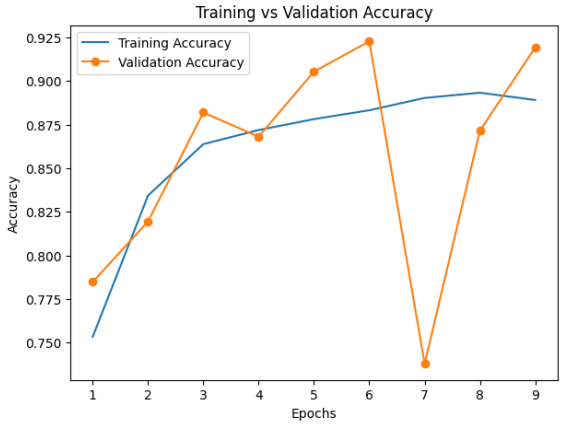
[ ]:
```
history.keys()
```

[ ]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[ ]:
```
train_loss, val_loss = history['loss'], history['val_loss']
train_acc, val_acc = history['accuracy'], history['val_accuracy']
```

[ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))
epoch_runs = [i+1 for i in range(9)]

ax1.plot(epoch_runs, train_loss, label='Training Loss')
ax1.plot(epoch_runs, val_loss, label='Validation Loss', marker='o')
ax1.set(title='Training vs Validation Loss', xlabel='Epochs',ylabel='Loss')
ax1.legend()

ax2.plot(epoch_runs, train_acc, label='Training Accuracy')
ax2.plot(epoch_runs, val_acc, label='Validation Accuracy', marker='o')
ax2.set(title='Training vs Validation Accuracy',␣
 ↪xlabel='Epochs',ylabel='Accuracy')
ax2.legend()

plt.show()
```





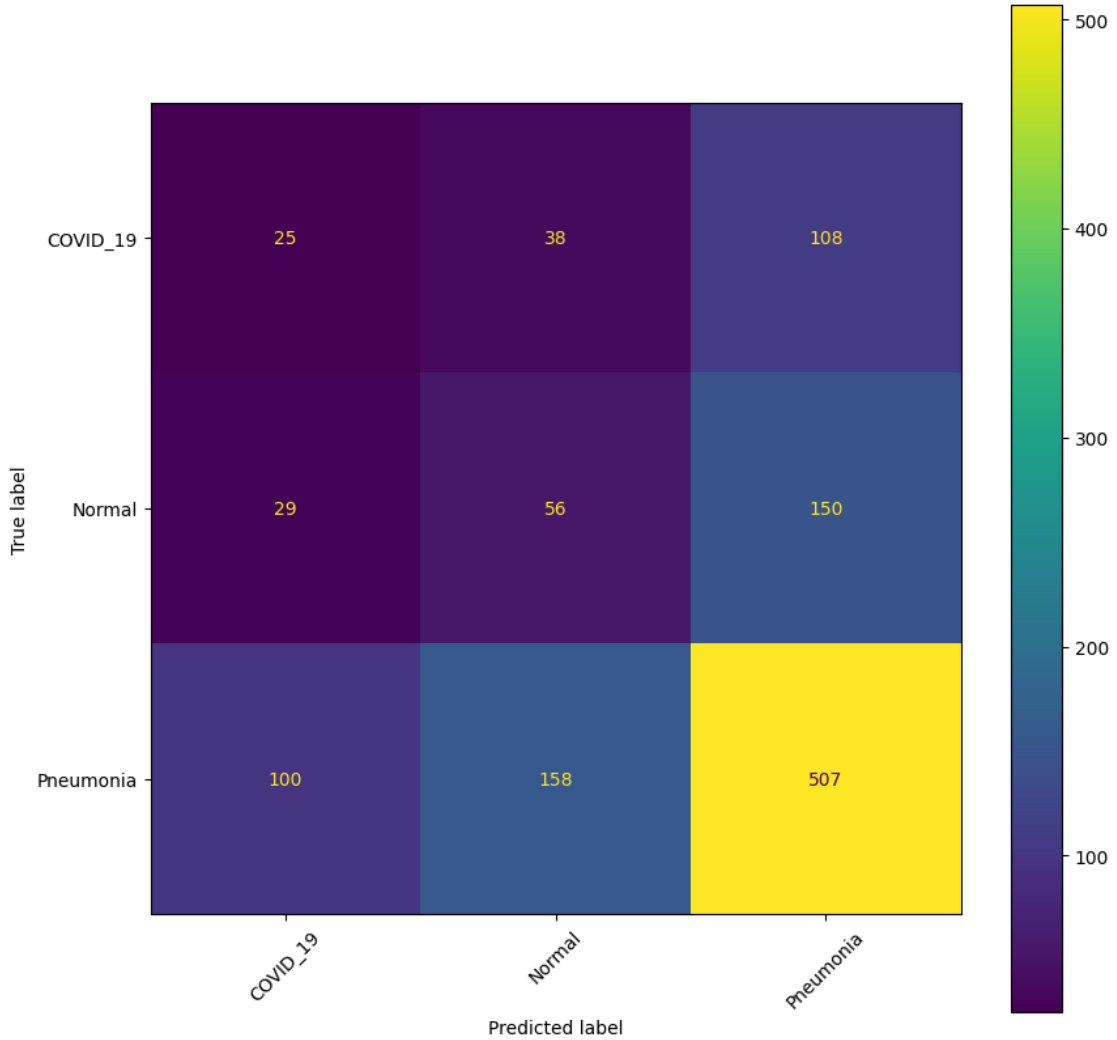
[ ]:
```
test_dir = 'DATASET/data/pneumonia_data/test'
# Assuming you have a separate test dataset stored in the variable test_dir
test_generator = test_datagen.flow_from_directory(test_dir, batch_size=32,␣
 ↪class_mode='categorical', target_size=(150,150))

# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(test_generator)

print("Test Loss:", test_loss)
print("Test Accuracy:", test_acc)
```

```
Found 340 images belonging to 3 classes.
11/11 [==============================] - 2s 182ms/step - loss: 0.2862 -
accuracy: 0.9118
Test Loss: 0.28617045283317566
Test Accuracy: 0.9117646813392639
```

[ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
validation_steps = validation_generator.samples // validation_generator.
 ↪batch_size
predictions = model.predict(validation_generator, steps=validation_steps)
predicted_classes = np.argmax(predictions, axis=1)

# Get the true labels from the validation generator
true_classes = validation_generator.classes
class_labels = list(validation_generator.class_indices.keys())

# Predict the classes for the entire validation dataset
validation_generator.reset()  # Ensure generator starts from the beginning
predictions = model.predict(validation_generator, steps=validation_generator.
 ↪samples // validation_generator.batch_size + 1)
predicted_classes = np.argmax(predictions, axis=1)
```

```
# Ensure the number of predictions matches the number of samples
predicted_classes = predicted_classes[:len(true_classes)]

# Compute the confusion matrix
cm = confusion_matrix(true_classes, predicted_classes)
cmd = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)

# Plot the confusion matrix
fig, ax = plt.subplots(figsize=(10, 10))
cmd.plot(ax=ax)
plt.xticks(rotation=45)
plt.show()
```

```
36/36 [==============================] - 10s 261ms/step
37/37 [==============================] - 10s 257ms/step
```



[ ]:
```
import pandas as pd

# Extract classes
classes = list(test_generator.class_indices.keys())

# Calculate metrics
tn = np.diag(cm)
fp = cm.sum(axis=0) - np.diag(cm)
fn = cm.sum(axis=1) - np.diag(cm)
tp = cm.sum() - (fp + fn + tn)

confusion_matrix_df = pd.DataFrame({
    'Class': classes,
    'TN': tn,
    'FP': fp,
    'FN': fn,
    'TP': tp
})

print('Confusion Matrix Metrics:')
print(confusion_matrix_df)
```

```
Confusion Matrix Metrics:
       Class   TN   FP   FN   TP
0   COVID_19   25  129  146  871
1     Normal   56  196  179  740
2  Pneumonia  507  258  258  148
```

[ ]:
```
from sklearn.metrics import classification_report

# Predict probabilities for the validation set
y_pred_probs = model.predict(validation_generator)

# Convert probabilities to class labels
y_pred = np.argmax(y_pred_probs, axis=1)

# Get true labels
y_true = validation_generator.classes

# Get class names
class_names = list(validation_generator.class_indices.keys())

# Generate classification report
report = classification_report(y_true, y_pred, target_names=class_names,␣
 ↪output_dict=True)
```

```python
# Convert the report to a pandas DataFrame for easier manipulation
import pandas as pd
report_df = pd.DataFrame(report).transpose()

# Print or display the DataFrame
print(report_df)
```

```
37/37 [==============================] - 10s 252ms/step
              precision    recall  f1-score      support
COVID_19       0.162338  0.146199  0.153846   171.000000
Normal         0.242063  0.259574  0.250513   235.000000
Pneumonia      0.667974  0.667974  0.667974   765.000000
accuracy       0.509821  0.509821  0.509821     0.509821
macro avg      0.357458  0.357916  0.357444  1171.000000
weighted avg   0.508663  0.509821  0.509119  1171.000000
```