```
from __future__ import absolute_import, division, print_function, unicode_literals

try:
    import tensorflow.compat.v2 as tf
except Exception:
    import tensorflow as tf


tf.enable_v2_behavior()

print(tf.__version__)
```

```
2.7.0
```

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

```
import os
import glob
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
import pathlib
import shutil
import cv2
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive
```

```
path= '/content/gdrive/My Drive/PREPROCESS'
```

```
data_dir = pathlib.Path(path)
image_count = len(list(data_dir.glob('*/*.jpg')))
image_count
```

```
844
```

```
CLASS_NAMES = np.array([item.name for item in data_dir.glob('*')])
CLASS_NAMES
```

```
array(['N', 'MODERATE', 'MILD', 'NORMAL-PCR+', 'SEVERE'], dtype='<U11')
```

```
BATCH_SIZE = 32
IMG_SIZE = 224
EPOCS = 50
```

```
list_ds = tf.data.Dataset.list_files(str(data_dir/'*/*.jpg'))
list_ds
```

```
<ShuffleDataset shapes: (), types: tf.string>
```

```
temp=list_ds.take(1)
print(next(iter(temp)))
```

```
tf.Tensor(b'/content/gdrive/My Drive/PREPROCESS/N/fce878de9b689a03dbcaf39f8781ed3b_cr
```

```
train_size = int(0.7 * image_count)
val_size = int(0.2 * image_count)
test_size = image_count - train_size - val_size

print("Total Images     : ", image_count)
print("train Images     : ", train_size)
print("validation Images: ", val_size)
print("test Images      : ", test_size)

SUFFLE_BUFFER_SIZE = int(test_size/2)
STEPS_PER_EPOCH = np.ceil(train_size/BATCH_SIZE)
VALIDATION_STEPS = np.ceil(val_size/BATCH_SIZE)

full_list_dataset = list_ds.shuffle(buffer_size=SUFFLE_BUFFER_SIZE)
train_list_dataset = full_list_dataset.take(train_size)
test_list_dataset = full_list_dataset.skip(train_size)
val_list_dataset = test_list_dataset.take(val_size)
test_list_dataset = test_list_dataset.skip(val_size)
```

```
Total Images     :  844
train Images     :  590
validation Images:  168
test Images      :  86
```

```
def get_label(file_path):
  # convert the path to a list of path components
  parts = tf.strings.split(file_path, os.path.sep)
  # # The second to last is the class-directory
  return parts[-2] == CLASS_NAMES
```

```
image_path = next(iter(test_list_dataset))
print(image_path)
```

```
tf.Tensor(b'/content/gdrive/My Drive/PREPROCESS/MILD/5739b66f3e5a086cb1d5273bd880cc05
```

```
def decode_img(img):
  # convert the compressed string to a 3D uint8 tensor
  img = tf.image.decode_jpeg(img, channels=3)
  # Use `convert_image_dtype` to convert to floats in the [0,1] range.
  img = tf.image.convert_image_dtype(img, tf.float32)
```

```
  # resize the image to the desired size.
  return tf.image.resize(img, [IMG_SIZE, IMG_SIZE])
```

```
def process_path(file_path):
    label = get_label(file_path)
    # load the raw data from the file as a string
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label
```

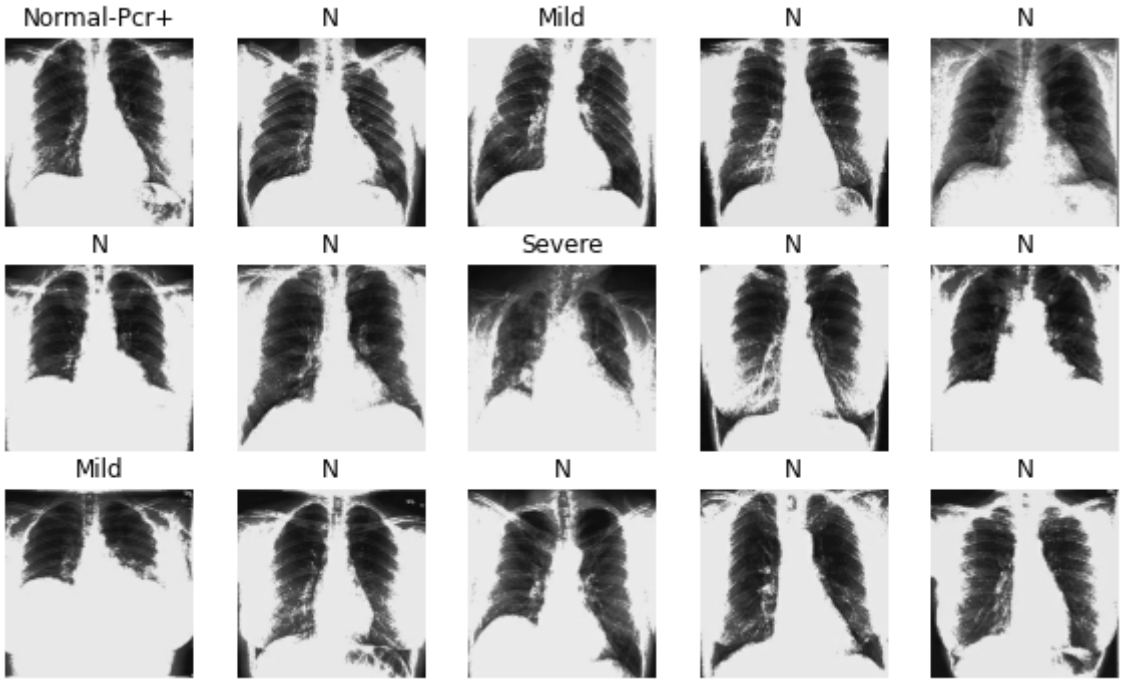
```
# Set `num_parallel_calls` so multiple images are loaded/processed in parallel.
labeled_normal_ds = train_list_dataset.map(process_path, num_parallel_calls=AUTOTUNE)
train_dataset = labeled_normal_ds
labeled_normal_ds = val_list_dataset.map(process_path, num_parallel_calls=AUTOTUNE)
val_dataset = labeled_normal_ds

```

```
def prepare_for_training(ds, cache=False, shuffle_buffer_size=SUFFLE_BUFFER_SIZE):
  # This is a small dataset, only load it once, and keep it in memory.
  # use `.cache(filename)` to cache preprocessing work for datasets that don't
  # fit in memory.
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()

    ds = ds.shuffle(buffer_size=shuffle_buffer_size)

    # Repeat forever
    ds = ds.repeat()

    ds = ds.batch(BATCH_SIZE)

    # `prefetch` lets the dataset fetch batches in the background while the model
    # is training.
    ds = ds.prefetch(buffer_size=AUTOTUNE)

    return ds
```

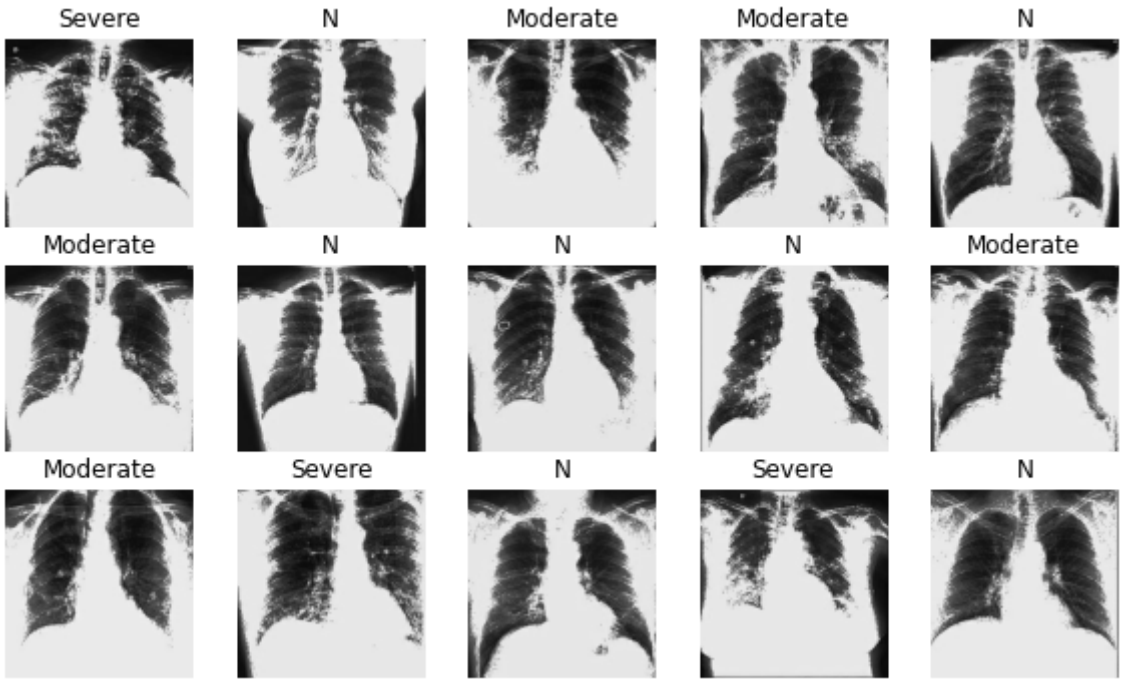
```
def show_batch(image_batch, label_batch):
    plt.figure(figsize=(10,10))
    for n in range(15):
        ax = plt.subplot(5,5,n+1)
        img = image_batch[n]
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        plt.imshow(img, cmap="gray")
        plt.title(CLASS_NAMES[label_batch[n]==1][0].title())
        plt.axis('off')
```

```
train_ds = prepare_for_training(train_dataset)

```

```
train_image_batch, train_label_batch = next(iter(train_ds))

show_batch(train_image_batch.numpy(), train_label_batch.numpy())
```




```
val_ds = prepare_for_training(val_dataset)

val_image_batch, val_label_batch = next(iter(val_ds))

show_batch(val_image_batch.numpy(), val_label_batch.numpy())
```




```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, MaxPool2D, Dropout, Flatten, Activat
from tensorflow.keras.layers import BatchNormalization
```

```
model = Sequential()

model.add(Conv2D(filters=16, kernel_size=(3,3), strides=(1,1), padding='same', input_sh
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Conv2D(filters=128, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation("relu"))

model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='valid'))
model.add(Activation("relu"))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Activation("relu"))
model.add(Dropout(0.2))


model.add(Flatten())

model.add(Dense(128))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Dense(84))
model.add(Activation("relu"))
model.add(Dropout(0.2))

model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.25))

model.add(Dense(len(CLASS_NAMES), activation='softmax'))

model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])

```

```
model.summary()
```

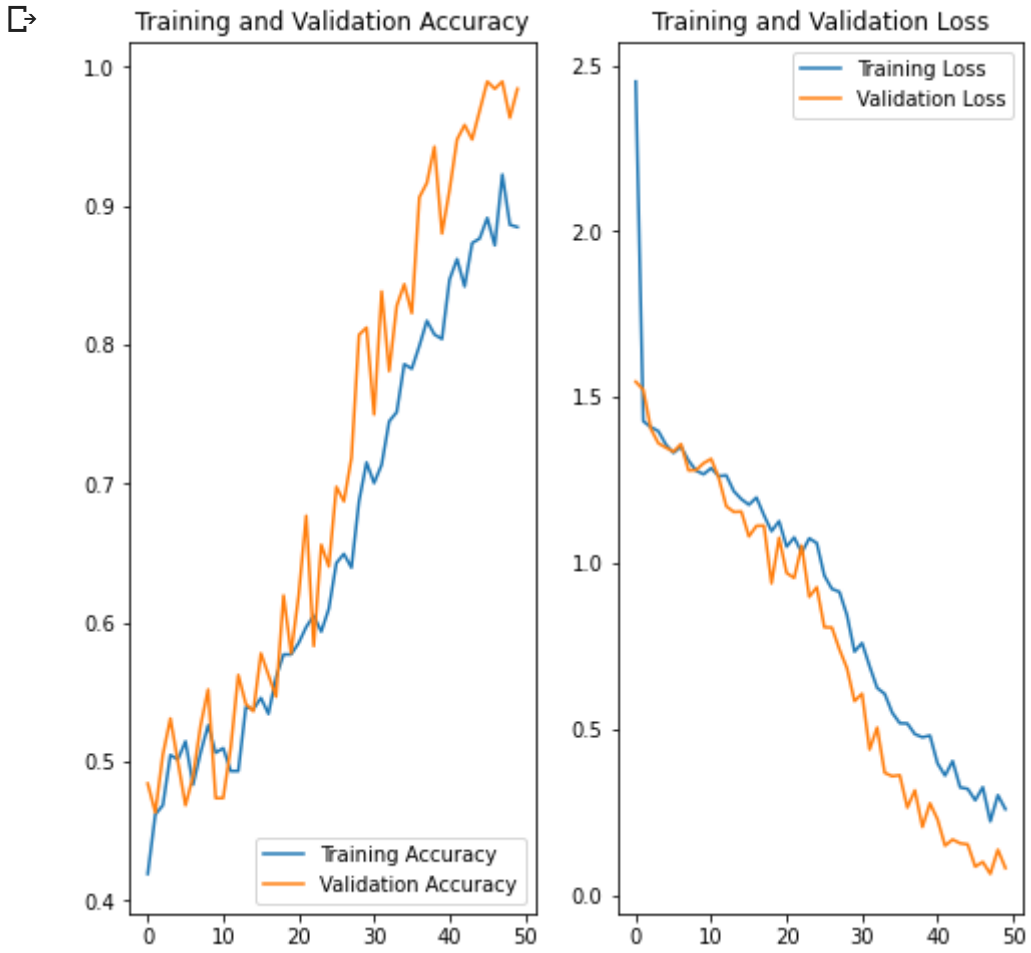
```
history = model.fit(
    train_ds,
    steps_per_epoch=STEPS_PER_EPOCH,
    epochs=50,
    validation_data=val_ds,
    validation_steps=VALIDATION_STEPS
)
```

```
19/19 [==============================] - 12s 611ms/step - loss: 1.0776 - accuracy:
Epoch 23/50
19/19 [==============================] - 12s 612ms/step - loss: 1.0315 - accuracy:
Epoch 24/50
19/19 [==============================] - 12s 613ms/step - loss: 1.0747 - accuracy:
Epoch 25/50
```

```
Epoch 25/50
19/19 [==============================] - 12s 615ms/step - loss: 1.0605 - accuracy:
Epoch 26/50
19/19 [==============================] - 12s 613ms/step - loss: 0.9625 - accuracy:
Epoch 27/50
19/19 [==============================] - 12s 614ms/step - loss: 0.9225 - accuracy:
Epoch 28/50
19/19 [==============================] - 12s 610ms/step - loss: 0.9136 - accuracy:
Epoch 29/50
19/19 [==============================] - 12s 613ms/step - loss: 0.8434 - accuracy:
Epoch 30/50
19/19 [==============================] - 12s 614ms/step - loss: 0.7334 - accuracy:
Epoch 31/50
19/19 [==============================] - 12s 613ms/step - loss: 0.7595 - accuracy:
Epoch 32/50
19/19 [==============================] - 12s 613ms/step - loss: 0.6893 - accuracy:
Epoch 33/50
19/19 [==============================] - 12s 614ms/step - loss: 0.6240 - accuracy:
Epoch 34/50
19/19 [==============================] - 12s 612ms/step - loss: 0.6062 - accuracy:
Epoch 35/50
19/19 [==============================] - 12s 613ms/step - loss: 0.5493 - accuracy:
Epoch 36/50
19/19 [==============================] - 12s 613ms/step - loss: 0.5178 - accuracy:
Epoch 37/50
19/19 [==============================] - 12s 612ms/step - loss: 0.5169 - accuracy:
Epoch 38/50
19/19 [==============================] - 12s 614ms/step - loss: 0.4848 - accuracy:
Epoch 39/50
19/19 [==============================] - 12s 613ms/step - loss: 0.4754 - accuracy:
Epoch 40/50
19/19 [==============================] - 12s 611ms/step - loss: 0.4806 - accuracy:
Epoch 41/50
19/19 [==============================] - 12s 610ms/step - loss: 0.3968 - accuracy:
Epoch 42/50
19/19 [==============================] - 12s 613ms/step - loss: 0.3599 - accuracy:
Epoch 43/50
19/19 [==============================] - 12s 613ms/step - loss: 0.4036 - accuracy:
Epoch 44/50
19/19 [==============================] - 12s 613ms/step - loss: 0.3242 - accuracy:
Epoch 45/50
19/19 [==============================] - 12s 613ms/step - loss: 0.3201 - accuracy:
Epoch 46/50
19/19 [==============================] - 12s 613ms/step - loss: 0.2855 - accuracy:
Epoch 47/50
19/19 [==============================] - 12s 613ms/step - loss: 0.3249 - accuracy:
Epoch 48/50
19/19 [==============================] - 12s 612ms/step - loss: 0.2220 - accuracy:
Epoch 49/50
19/19 [==============================] - 12s 613ms/step - loss: 0.3009 - accuracy:
Epoch 50/50
19/19 [==============================] - 12s 612ms/step - loss: 0.2586 - accuracy:
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

```
6
7 epochs_range = range(50)
8
9 plt.figure(figsize=(8, 8))
10 plt.subplot(1, 2, 1)
11 plt.plot(epochs_range, acc, label='Training Accuracy')
12 plt.plot(epochs_range, val_acc, label='Validation Accuracy')
13 plt.legend(loc=4)
14 plt.title('Training and Validation Accuracy')
15
16 plt.subplot(1, 2, 2)
17 plt.plot(epochs_range, loss, label='Training Loss')
18 plt.plot(epochs_range, val_loss, label='Validation Loss')
19 plt.legend(loc=1)
20 plt.title('Training and Validation Loss')
21 plt.show()
```



+ Code + Text

Double-click (or enter) to edit

FINAL GRAPH

```
1 def load_image(img_path, show = False):
2     img_tensor, label = process_path(img_path) # Image With(height, width, channels)
3
4     if show:
5         img = cv2.cvtColor(img_tensor.numpy(), cv2.COLOR_RGB2GRAY)
```

```
        plt.imshow(img, cmap="gray")
        #plt.imshow(img_tensor.numpy())
        plt.title(CLASS_NAMES[label.numpy()==1][0].title())
        plt.axis('off')
        plt.show()

    img_tensor = np.expand_dims(img_tensor, axis=0) # (1, height, width, channels), add

    return img_tensor, label
```

```
test_images = []
true_labels = []
for p in test_list_dataset.take(test_size):
  i, l = load_image(p, False)
  test_images.append(i)
  true_labels.append(l)
```

```
test_dataset = tf.data.Dataset.from_tensor_slices(test_images)
```

```
predictions = model.predict_generator(test_dataset)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `Model.p
  """Entry point for launching an IPython kernel.
```

```
predict_labels = []
for pred in predictions:
    max_value = max(pred)
    boolArr = (pred == max_value)
    predict_labels.append(boolArr)
```

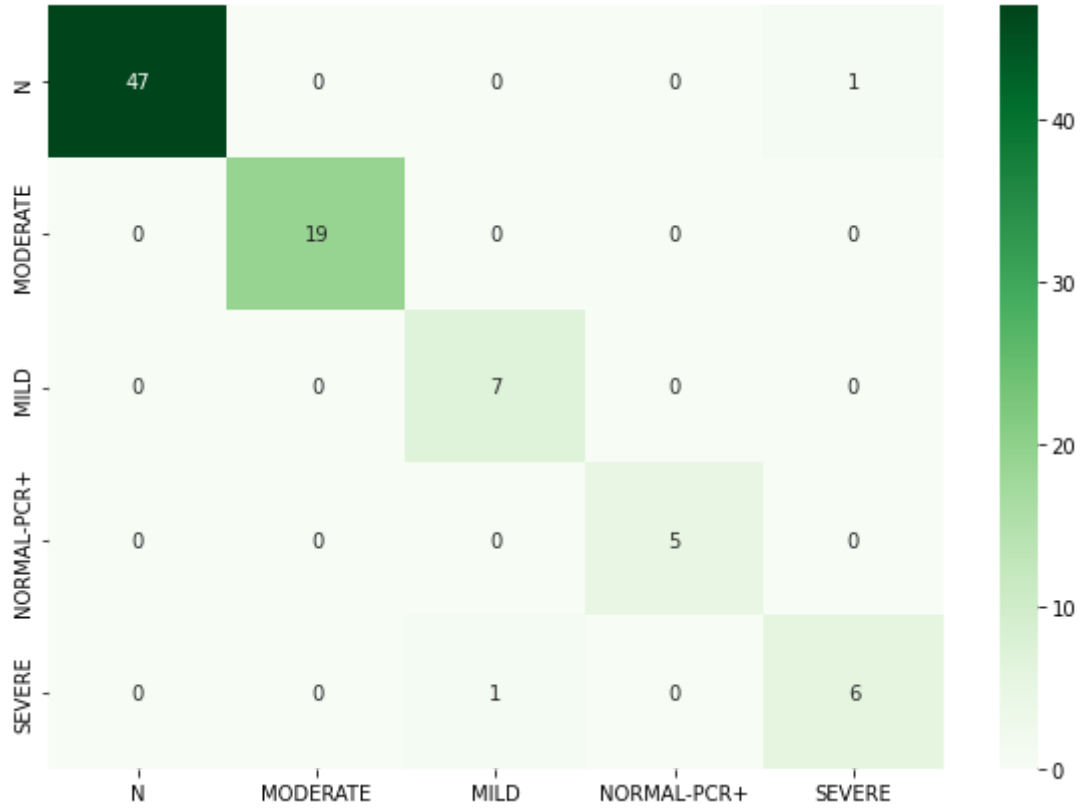
```
true_labels = np.argmax(true_labels, axis=1)
predict_labels = np.argmax(predict_labels, axis=1)
```

```
from sklearn.metrics import classification_report
confusion = tf.math.confusion_matrix(labels=true_labels, predictions=predict_labels, nu
print(classification_report(true_labels, predict_labels, target_names=CLASS_NAMES))
print(confusion)
```

```
                precision    recall  f1-score   support

             N       1.00      0.98      0.99        48
      MODERATE       1.00      1.00      1.00        19
          MILD       0.88      1.00      0.93         7
    NORMAL-PCR+       1.00      1.00      1.00         5
        SEVERE       0.86      0.86      0.86         7

      accuracy                           0.98        86
     macro avg       0.95      0.97      0.96        86
  weighted avg       0.98      0.98      0.98        86

tf.Tensor(
```

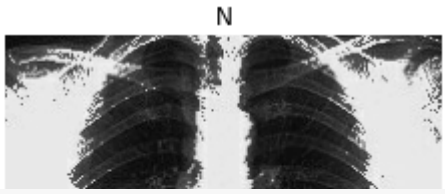
```
[[47  0  0  0  1]
 [ 0 19  0  0  0]
 [ 0  0  7  0  0]
 [ 0  0  0  5  0]
 [ 0  0  1  0  6]], shape=(5, 5), dtype=int32)
```

```
import pandas as pd
import seaborn as sn
df_cm = pd.DataFrame(confusion, index = [i for i in CLASS_NAMES],
                     columns = [i for i in CLASS_NAMES], dtype=float)

plt.figure(figsize = (10,7))
sn.heatmap(df_cm, annot=True, cmap="Greens")
plt.show()
```

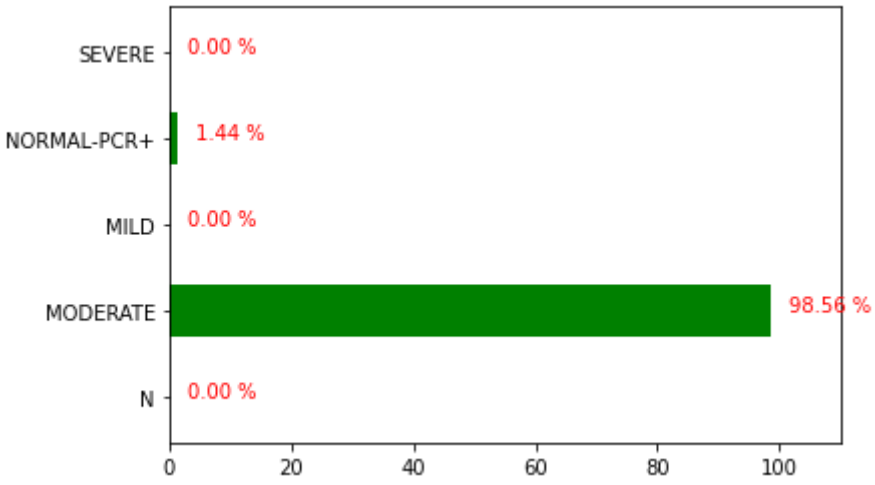



```
#single image prediction
image_path = next(iter(test_list_dataset))
print('Image path: ', image_path.numpy())
img, lbl = load_image(image_path, True)
#pred = model.predict(img)
#pred = pred[0]

```

```
Image path:  b'/content/gdrive/My Drive/PREPROCESS/N/96a6fd2052db2283879a9cf3de7e35e6
```




```
x = CLASS_NAMES
x_pos = [i for i, _ in enumerate(x)]
y_pos = [(per * 100) for per in [0,0.9856,0,0.0144,0]]
plt.barh(x_pos, y_pos, height=0.6, color='green')

for i, v in enumerate(y_pos):
  val = str("{0:.2f}".format(v)) + ' %'
  plt.text(v + 3, i , val, color='red')

plt.yticks(x_pos, x)
plt.xlim([0,110])
plt.show()
```



GRAD CAM

SAVE MODEL

```
if not os.path.exists('models'):
  os.mkdir('models')

if not os.path.exists('models/trained'):
  os.mkdir('models/trained')
```

```
# Saving as .h5 model

model.save('models/trained/model.h5')
```

```
# Saving as .tflite model

```

```
''' Covnvert from saved model
converter = tf.lite.TFLiteConverter.from_saved_model('models/trained/model.h5')
'''

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quantized_model = converter.convert()
open("models/trained/model_lite.tflite", "wb").write(tflite_quantized_model)
```

```
INFO:tensorflow:Assets written to: /tmp/tmpir8jhgvr/assets
24818576
```