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

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

```
!pip install split-folders
```

```
     Collecting split-folders
       Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
     Installing collected packages: split-folders
     Successfully installed split-folders-0.5.1
```

```
import tensorflow as tf
tf.random.set_seed(42)
import keras
import matplotlib.pyplot as plt
%matplotlib inline
import os
import pathlib
import splitfolders
import shutil
import warnings
import numpy as np
np.random.seed(42)
warnings.filterwarnings('ignore')
```

```
normal = "/content/drive/MyDrive/cse498r/normal"
pneumonia = "/content/drive/MyDrive/cse498r/pneumonia"

print(len(os.listdir(normal)))
print(len(os.listdir(pneumonia)))
```

```
     515
     114
```

data **augmentation**

```
main_dir = 'temp'
os.mkdir(main_dir)
no_dir = os.path.join(main_dir, 'no')
os.mkdir(no_dir)
yes_dir = os.path.join(main_dir, 'yes')
os.mkdir(yes_dir)
```

```
aug_dir = 'aug_dir'
os.mkdir(aug_dir)

# create a dir within the base dir to store images of the same class
img_dir = os.path.join(aug_dir, 'img_dir')
os.mkdir(img_dir)

# We are only data augmenting the "NO" class
img_class = 'normal'

# list all images in that directory
img_list = os.listdir('/content/drive/MyDrive/cse498r' + img_class)


for fname in img_list:
    # source path to image
    src = os.path.join('/content/drive/MyDrive/cse498r' + img_class, fname)
    # destination path to image
    dst = os.path.join(img_dir, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)


# point to a dir containing the images and not to the images themselves
path = aug_dir
save_path = '/content/drive/MyDrive/cse498r' + img_class

# Create a data generator
datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    fill_mode='nearest')

batch_size = 1

aug_datagen = datagen.flow_from_directory(path,
                                          save_to_dir=save_path,
                                          save_format='jpg',
                                          target_size=(224,224),
                                          batch_size=batch_size)


# Generate the augmented images and add them to the training folders

# total number of images we want to have in each class
num_aug_images_wanted = 250
# (Note: We may or may not get the intended num of augmented images, so play will the number. I put 250 here to get 150 images)

num_files = len(os.listdir(img_dir))
num_batches = int(np.ceil((num_aug_images_wanted-num_files)/batch_size))

# run the generator and create our augmented images
for i in range(0,num_batches):
    imgs, labels = next(aug_datagen)

# delete temporary directory with the raw image files
shutil.rmtree('aug_dir')
```

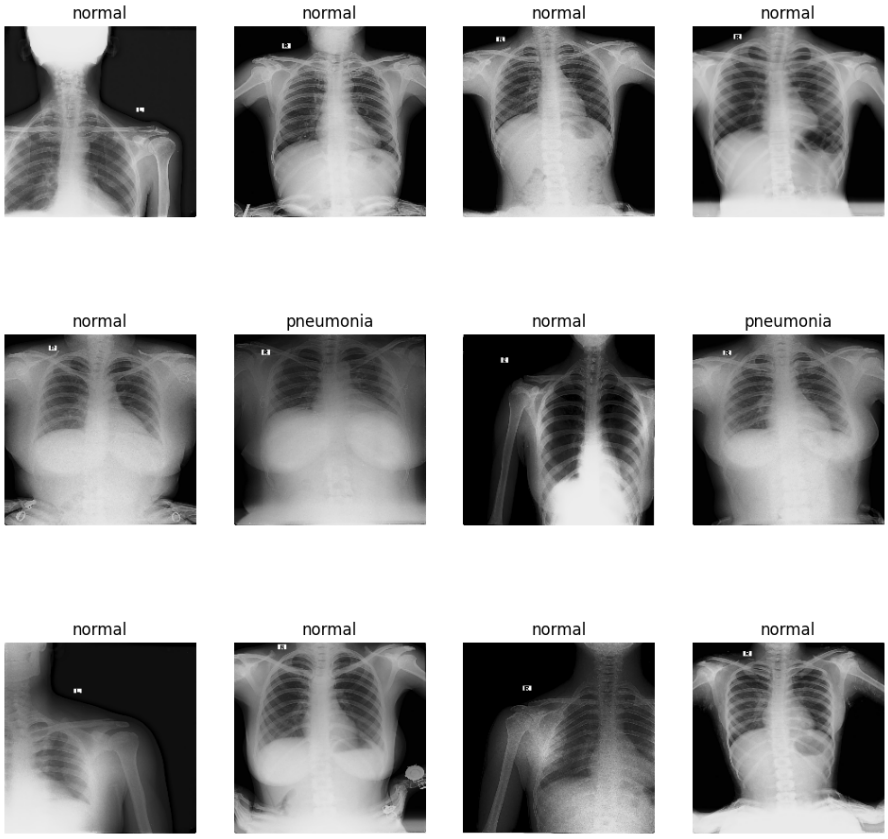
```
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-8-6565bad698b1> in <cell line: 2>()
      1 aug_dir = 'aug_dir'
----> 2 os.mkdir(aug_dir)
      3
      4 # create a dir within the base dir to store images of the same class
      5 img_dir = os.path.join(aug_dir, 'img_dir')

FileExistsError: [Errno 17] File exists: 'aug_dir'
```

SEARCH STACK OVERFLOW

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt


import os
import pathlib


data_path = pathlib.Path("/content/drive/MyDrive/cse498r")


dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_path, seed=123,
    shuffle=True,
    image_size = (224, 224),
    batch_size = 68
)
```

```
 Found 629 files belonging to 2 classes.
```

```
class_names = dataset.class_names


plt.figure(figsize=(12, 12))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```




```
def split_data(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=699):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)
    if shuffle:
        ds.shuffle(shuffle_size, seed=123)

    train_size = int(ds_size * train_split)
    val_size = int(ds_size * val_split)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, test_ds, val_ds


train_ds, test_ds, val_ds = split_data(dataset)


train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])

train_ds = train_ds.map(lambda x,y: (data_augmentation(x, training=True), y)).prefetch(buffer_size=tf.data.AUTOTUNE)


resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(224, 224),
  layers.experimental.preprocessing.Rescaling(1./255),
])


vgg16 = tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet")

vgg16.trainable = False

inputs = tf.keras.Input(shape=(224, 224, 3))
x = tf.keras.applications.vgg16.preprocess_input(
    inputs, data_format=None)

x = vgg16(x, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Dense(4096, activation="relu")(x)
x = tf.keras.layers.Dense(4096, activation="relu")(x)
outputs = tf.keras.layers.Dense(2, activation="sigmoid")(x)
model = tf.keras.Model(inputs, outputs)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 2s 0us/step
```

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 vgg16 (Functional)          (None, None, None, 512)   14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 512)               0

 dense (Dense)               (None, 4096)              2101248

 dense_1 (Dense)             (None, 4096)              16781312

 dense_2 (Dense)             (None, 2)                 8194

=================================================================
Total params: 33605442 (128.19 MB)
Trainable params: 18890754 (72.06 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
from tensorflow.keras.optimizers import Adam
opt = Adam(learning_rate=0.001)


model.compile(optimizer=opt, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), metrics=['accuracy'])
```

```
from keras.callbacks import ModelCheckpoint, EarlyStopping
checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_only=False, mode='auto', period=1
early = EarlyStopping(monitor='val_acc', min_delta=0, patience=6, verbose=1, mode='auto')
```

```
tf.config.experimental.list_physical_devices('GPU')
```

```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
history = model.fit(train_ds, epochs=30, validation_data=val_ds, callbacks = cb)
```

```
8/8 [==============================] - 5s 651ms/step - loss: 0.3233 - accuracy: 0.8382 - val_loss: 0.3196 - val_accuracy: 0.8235
Epoch 17/30
8/8 [==============================] - ETA: 0s - loss: 0.3427 - accuracy: 0.8474WARNING:tensorflow:Can save best model only with val_
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_lo
8/8 [==============================] - 5s 661ms/step - loss: 0.3427 - accuracy: 0.8474 - val_loss: 0.3539 - val_accuracy: 0.7941
Epoch 18/30
8/8 [==============================] - ETA: 0s - loss: 0.3308 - accuracy: 0.8456WARNING:tensorflow:Can save best model only with val_
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_lo
8/8 [==============================] - 6s 719ms/step - loss: 0.3308 - accuracy: 0.8456 - val_loss: 0.3102 - val_accuracy: 0.8382
Epoch 19/30
8/8 [==============================] - ETA: 0s - loss: 0.3307 - accuracy: 0.8382WARNING:tensorflow:Can save best model only with val_
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_lo
8/8 [==============================] - 6s 690ms/step - loss: 0.3307 - accuracy: 0.8382 - val_loss: 0.3117 - val_accuracy: 0.8824
Epoch 20/30
8/8 [==============================] - ETA: 0s - loss: 0.3451 - accuracy: 0.8493WARNING:tensorflow:Can save best model only with val_
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_lo
8/8 [==============================] - 5s 665ms/step - loss: 0.3451 - accuracy: 0.8493 - val_loss: 0.3333 - val_accuracy: 0.8235
Epoch 21/30
8/8 [==============================] - ETA: 0s - loss: 0.3350 - accuracy: 0.8419WARNING:tensorflow:Can save best model only with val_
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_lo
8/8 [==============================] - 6s 756ms/step - loss: 0.3350 - accuracy: 0.8419 - val_loss: 0.3656 - val_accuracy: 0.7794
```