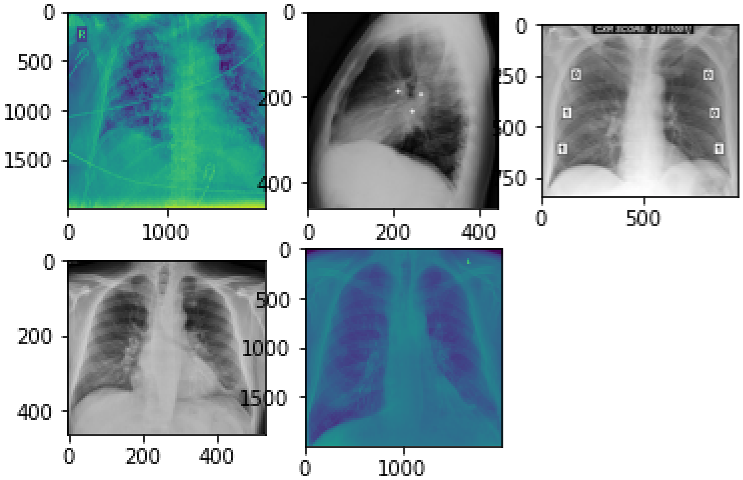
APPENDIX

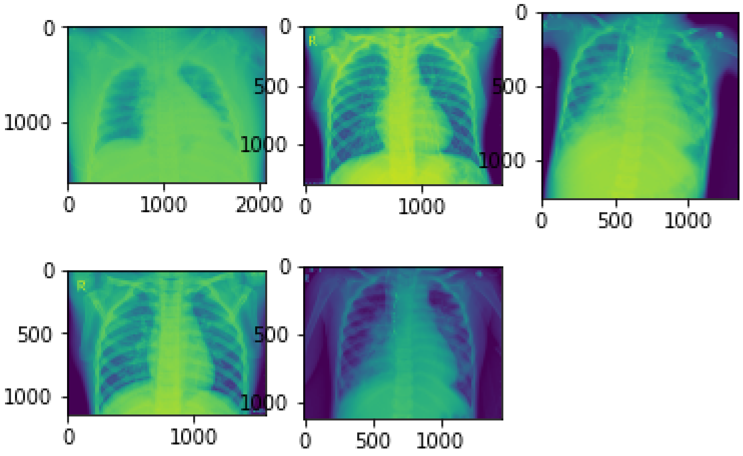
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dro
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam

from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

import tensorflow as tf
from sklearn.decomposition import PCA

import cv2
import os

import numpy as np

from sklearn.model_selection import train_test_split

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import models, layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dropout, Flatten, Input, Dense

# Load the Drive helper and mount
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import matplotlib.image as mpimg
directory=os.listdir('/content/drive/MyDrive/COVID/train/')
for each in directory:
    plt.figure()
    currentFolder = '/content/drive/MyDrive/COVID/train/' + each
    for i, file in enumerate(os.listdir(currentFolder)[0:5]):
        fullpath = currentFolder  + "/" + file
        print(fullpath)
        img=mpimg.imread(fullpath)
        plt.subplot(2, 3, i+1)
        plt.imshow(img)
```

```
/content/drive/MyDrive/COVID/train/Covid Positive/00870a9c.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/000025-1.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/11547_2020_1200_Fig2_HTML-a.png
/content/drive/MyDrive/COVID/train/Covid Positive/000024-1.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/1052b0fe.jpg
/content/drive/MyDrive/COVID/train/Covid Negative/person108_virus_199.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person120_virus_226.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person124_virus_238.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person130_virus_263.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person124_virus_236.jpeg
```

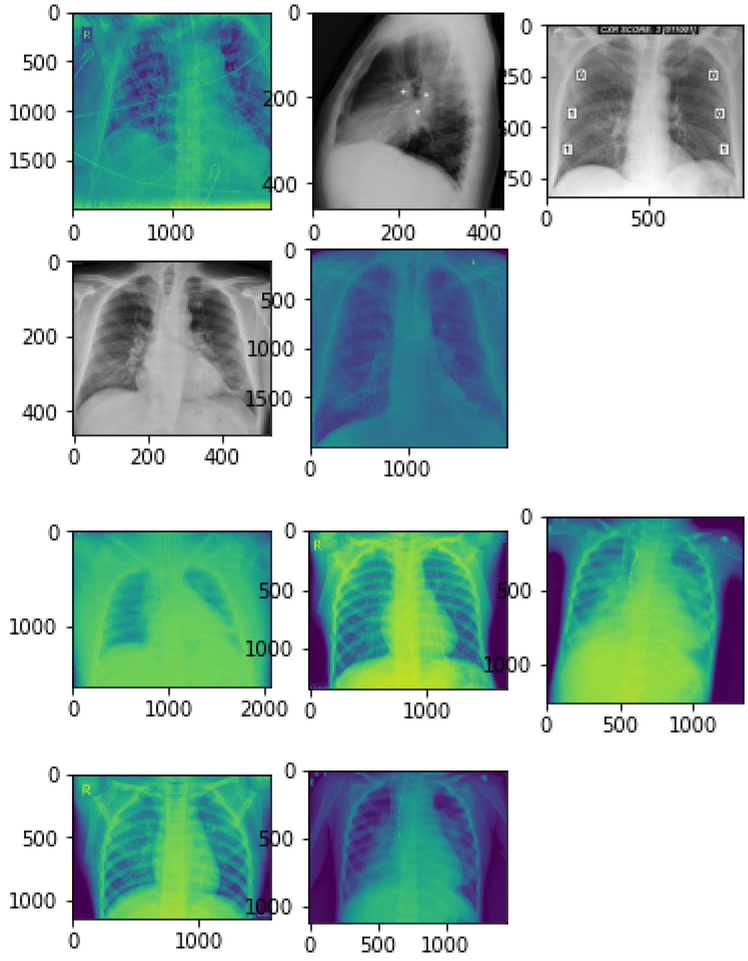




```
import matplotlib.image as mpimg
directory=os.listdir('/content/drive/MyDrive/COVID/train/')
```

```
for each in directory:
    plt.figure()
    currentFolder = '/content/drive/MyDrive/COVID/train/' + each
    for i, file in enumerate(os.listdir(currentFolder)[0:5]):
        fullpath = currentFolder  + "/" + file
        print(fullpath)
        img=mpimg.imread(fullpath)
        plt.subplot(2, 3, i+1)
        plt.imshow(img)
```

```
/content/drive/MyDrive/COVID/train/Covid Positive/00870a9c.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/000025-1.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/11547_2020_1200_Fig2_HTML-a.png
/content/drive/MyDrive/COVID/train/Covid Positive/000024-1.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/1052b0fe.jpg
/content/drive/MyDrive/COVID/train/Covid Negative/person108_virus_199.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person120_virus_226.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person124_virus_238.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person130_virus_263.jpeg
/content/drive/MyDrive/COVID/train/Covid Negative/person124_virus_236.jpeg
```

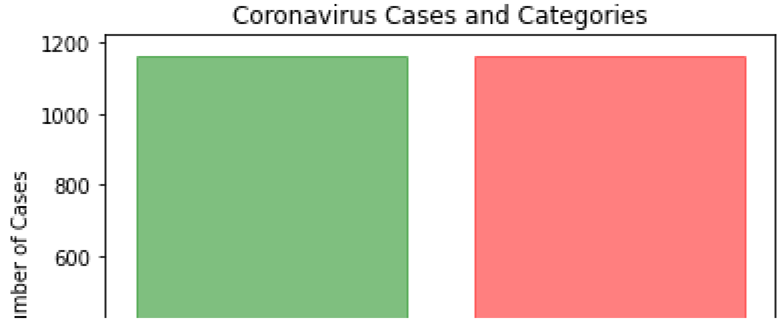


```
labels = ['Covid Negative', 'Covid Positive']
img_size = 64
def get_data(data_dir):
    data = []
    for label in labels:
        path = os.path.join(data_dir, label)
        class_num = labels.index(label)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img))[...,
                resized_arr = cv2.resize(img_arr, (img_size, img_s
                data.append([resized_arr, class_num])
            except Exception as e:
                print(e)
    return np.array(data)

train = get_data('/content/drive/MyDrive/COVID/train/')
test = get_data('/content/drive/MyDrive/COVID/test/')
```

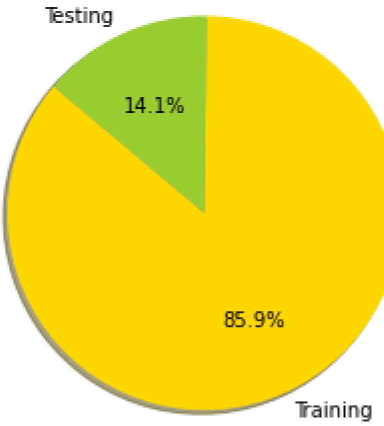
```
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: VisibleDeprecationWarnir
      from ipykernel import kernelapp as app
```

```
path = '/content/drive/MyDrive/COVID/train/Covid Positive'
path1 = '/content/drive/MyDrive/COVID/test/Covid Positive'
path2 = '/content/drive/MyDrive/COVID/train/Covid Negative'
path3 = '/content/drive/MyDrive/COVID/test/Covid Negative'
covidpositives = len([f for f in os.listdir(path)if os.path.isfile
covidnegatives = len([f for f in os.listdir(path2)if os.path.isfil
```

```
Cats = ['Covid Positive', 'Covid Negative']
y_pos = np.arange(len(Cats))
barlist = plt.bar(y_pos,[covidpositives, covidnegatives], align='c
barlist[0].set_color('g')
barlist[1].set_color('r')
plt.xticks(y_pos,['Covid Positive', 'Covid Negative'])
plt.ylabel('Number of Cases')
plt.title('Coronavirus Cases and Categories')

plt.show()
```

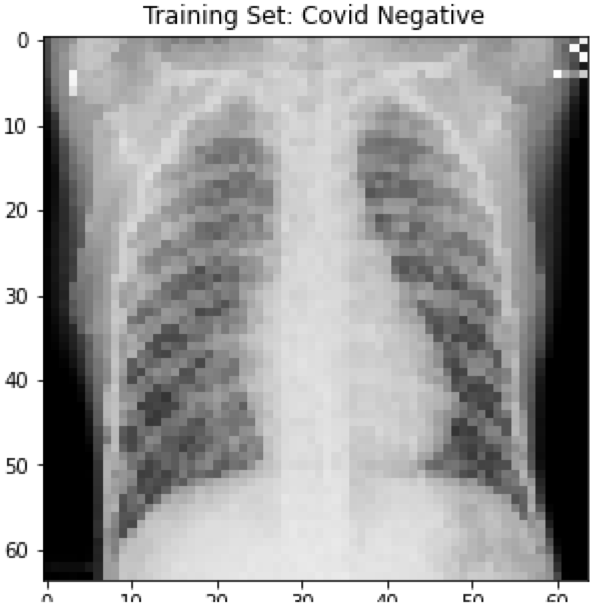


```
path = '/content/drive/MyDrive/COVID/test/Covid Negative'
path1 = '/content/drive/MyDrive/COVID/test/Covid Positive'
path2 = '/content/drive/MyDrive/COVID/train/Covid Negative'
path3 = '/content/drive/MyDrive/COVID/train/Covid Positive'
Test = len([f for f in os.listdir(path)if os.path.isfile(os.path.j
Train = len([f for f in os.listdir(path2)if os.path.isfile(os.path
# Data to plot
labels = 'Training', 'Testing'
sizes = [Train, Test]
colors = ['gold', 'yellowgreen']
explode = (0, 0)  # explode 1st slice

# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')
plt.show()
```



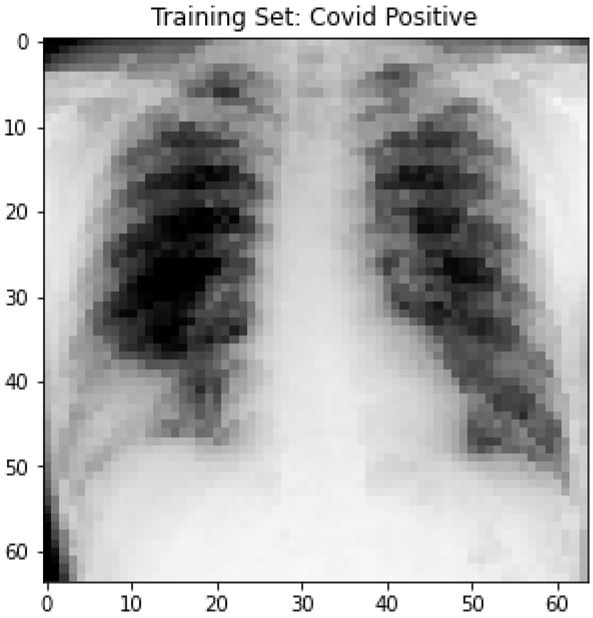
```
plt.figure(figsize = (5,5))
plt.imshow(train[1][0])
plt.title('Training Set: Covid Negative')
```

```
Text(0.5, 1.0, 'Training Set: Covid Negative')
```



```
plt.figure(figsize = (5,5))
plt.imshow(train[-1][0])
plt.title('Training Set: Covid Positive')
```

```
Text(0.5, 1.0, 'Training Set: Covid Positive')
```



```
x_train = []
y_train = []
x_test = []
y_test = []

for feature, label in train:
  x_train.append(feature)
  y_train.append(label)
```

```
for feature, label in test:
  x_test.append(feature)
  y_test.append(label)

# Normalize the data
x_train = np.array(x_train) / 255
x_test = np.array(x_test) / 255

x_train.reshape(-1, img_size, img_size, 1)
y_train = np.array(y_train)

x_test.reshape(-1, img_size, img_size, 1)
y_test = np.array(y_test)


datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by s
        samplewise_std_normalization=False,  # divide each input b
        zca_whitening=False,  # apply ZCA whitening
        rotation_range = 30,  # randomly rotate images in the rang
        zoom_range = 0.2, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontal
        height_shift_range=0.1,  # randomly shift images verticall
        horizontal_flip = True,  # randomly flip images
        vertical_flip=True)  # randomly flip images



datagen.fit(x_train)


X_train, X_val, y_train, y_val = train_test_split(x_train, y_train
```

Experiment 1

```
model = models.Sequential()
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(1
model.add(layers.MaxPooling2D((2, 2),strides=2))
model.add(layers.Flatten())
model.add(layers.Dense(units=32, activation=tf.nn.relu))
model.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
model.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
%%time
history = model.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
import numpy as np
```

```
loss, accuracy = model.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 2

```
model2 = models.Sequential()
model2.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model2.add(layers.MaxPooling2D((2, 2),strides=2))
model2.add(layers.Flatten())
model2.add(layers.Dense(units=64, activation=tf.nn.relu))
model2.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model2.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
%%time
history2 = model2.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model2.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 3

```
model3 = models.Sequential()
model3.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides=
model3.add(layers.MaxPooling2D((2, 2),strides=2))
model3.add(layers.Flatten())
model3.add(layers.Dense(units=128, activation=tf.nn.relu))
model3.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model3.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
%%time
history3 = model3.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model3.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 4

```
model4 = models.Sequential()
model4.add(layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(
model4.add(layers.MaxPooling2D((2, 2),strides=2))
model4.add(layers.Conv2D(filters=32, kernel_size=(3, 3), strides=(
model4.add(layers.MaxPooling2D((2, 2),strides=2))
```

```
model4.add(layers.MaxPooling2D((2, 2),strides=2))
model4.add(layers.Flatten())4
model4.add(layers.Dense(units=256, activation=tf.nn.relu))
model4.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model4.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])

%%time
history4 = model4.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )

loss, accuracy = model4.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 5

```
model5 = models.Sequential()
model5.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model5.add(layers.MaxPooling2D((2, 2),strides=2))
model5.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model5.add(layers.MaxPooling2D((2, 2),strides=2))
model5.add(layers.Flatten())
model5.add(layers.Dense(units=256, activation=tf.nn.relu))
model5.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model5.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])

%%time
history5 = model5.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model5.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 6

```
model6 = models.Sequential()
model6.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides=
model6.add(layers.MaxPooling2D((2, 2),strides=2))
model6.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides=
model6.add(layers.MaxPooling2D((2, 2),strides=2))
model6.add(layers.Flatten())
model6.add(layers.Dense(units=256, activation=tf.nn.relu))
model6.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model6.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
%%time
history6 = model6.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model6.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 7

```
model7 = models.Sequential()
model7.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model7.add(layers.MaxPooling2D((2, 2),strides=2))
model7.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model7.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model7.add(layers.MaxPooling2D((2, 2),strides=2))
model7.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model7.add(layers.MaxPooling2D((2, 2),strides=2))
model7.add(layers.Flatten())
model7.add(layers.Dense(units=256, activation=tf.nn.relu))
model7.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model7.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
%%time
history7 = model7.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model7.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 8

```
from keras.layers import AveragePooling2D
model8 = models.Sequential()
model8.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model8.add(layers.AveragePooling2D((2, 2),strides=2))
model8.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model8.add(layers.AveragePooling2D((2, 2),strides=2))
model8.add(layers.Conv2D(filters=64, kernel_size=(3, 3), strides=(
model8.add(layers.AveragePooling2D((2, 2),strides=2))
model8.add(layers.Flatten())
model8.add(layers.Dense(units=256, activation=tf.nn.relu))
model8.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model8.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history8 = model8.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )

loss, accuracy = model8.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 9

```
model9 = models.Sequential()
model9.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides=
model9.add(layers.MaxPooling2D((2, 2),strides=2))
model9.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides=
model9.add(layers.MaxPooling2D((2, 2),strides=2))
model9.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides=
model9.add(layers.MaxPooling2D((2, 2),strides=2))
model9.add(layers.Flatten())
model9.add(layers.Dense(units=256, activation=tf.nn.relu))
model9.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model9.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history9 = model9.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
Epoch 1/20
4/4 [==============================] - 1s 191ms/step - loss: 0.7121 - accuracy: 0.5463 -
Epoch 2/20
4/4 [==============================] - 1s 146ms/step - loss: 0.6498 - accuracy: 0.7436 -
```

```
Epoch 3/20
4/4 [==============================] - 1s 144ms/step - loss: 0.5600 - accuracy: 0.6937 -
Epoch 4/20
4/4 [==============================] - 1s 148ms/step - loss: 0.3948 - accuracy: 0.8584 -
Epoch 5/20
4/4 [==============================] - 1s 143ms/step - loss: 0.2281 - accuracy: 0.9257 -
Epoch 6/20
4/4 [==============================] - 1s 146ms/step - loss: 0.1593 - accuracy: 0.9396 -
Epoch 7/20
4/4 [==============================] - 1s 148ms/step - loss: 0.1313 - accuracy: 0.9500 -
Epoch 8/20
4/4 [==============================] - 1s 144ms/step - loss: 0.1165 - accuracy: 0.9555 -
Epoch 9/20
4/4 [==============================] - 1s 145ms/step - loss: 0.0998 - accuracy: 0.9671 -
Epoch 10/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0963 - accuracy: 0.9609 -
Epoch 11/20
4/4 [==============================] - 1s 145ms/step - loss: 0.0561 - accuracy: 0.9823 -
Epoch 12/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0782 - accuracy: 0.9717 -
Epoch 13/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0618 - accuracy: 0.9727 -
Epoch 14/20
4/4 [==============================] - 1s 144ms/step - loss: 0.0616 - accuracy: 0.9746 -
Epoch 15/20
4/4 [==============================] - 1s 147ms/step - loss: 0.0717 - accuracy: 0.9764 -
Epoch 16/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0551 - accuracy: 0.9777 -
Epoch 17/20
4/4 [==============================] - 1s 148ms/step - loss: 0.0480 - accuracy: 0.9832 -
Epoch 18/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0365 - accuracy: 0.9877 -
Epoch 19/20
4/4 [==============================] - 1s 146ms/step - loss: 0.0314 - accuracy: 0.9874 -
Epoch 20/20
4/4 [==============================] - 1s 143ms/step - loss: 0.0350 - accuracy: 0.9848 -
CPU times: user 7.58 s, sys: 3.79 s, total: 11.4 s
Wall time: 12.9 s
```

```
loss, accuracy = model9.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

```
11/11 [==============================] - 0s 5ms/step - loss: 0.0435 - accuracy: 0.9848
test set accuracy:  98.47561120986938
```

```
y_pred = (model9.predict(X_train) > 0.5).astype("int32")
confusion_matrix(y_train, y_pred)
```

```
array([[787,  14],
       [  5, 794]])
```

```
f1_score(y_train, y_pred, average='macro')
```

```
0.9881247727007274
```

```
recall_score(y_train, y_pred, average='macro')
recall_score(y_train, y_pred, average='macro')
```

```
    0.9881320127062698
```

```
precision_score(y_train, y_pred, average='macro')
```

```
    0.9881800680068007
```

```
y_pred = (model9.predict(x_test) > 0.5).astype("int32")
confusion_matrix(y_test, y_pred)
```

```
    array([[162,   2],
           [  3, 161]])
```

```
f1_score(y_test, y_pred, average='macro')
```

```
    0.9847559558666332
```

```
recall_score(y_test, y_pred, average='macro')
```

```
    0.9847560975609756
```

```
precision_score(y_test, y_pred, average='macro')
```

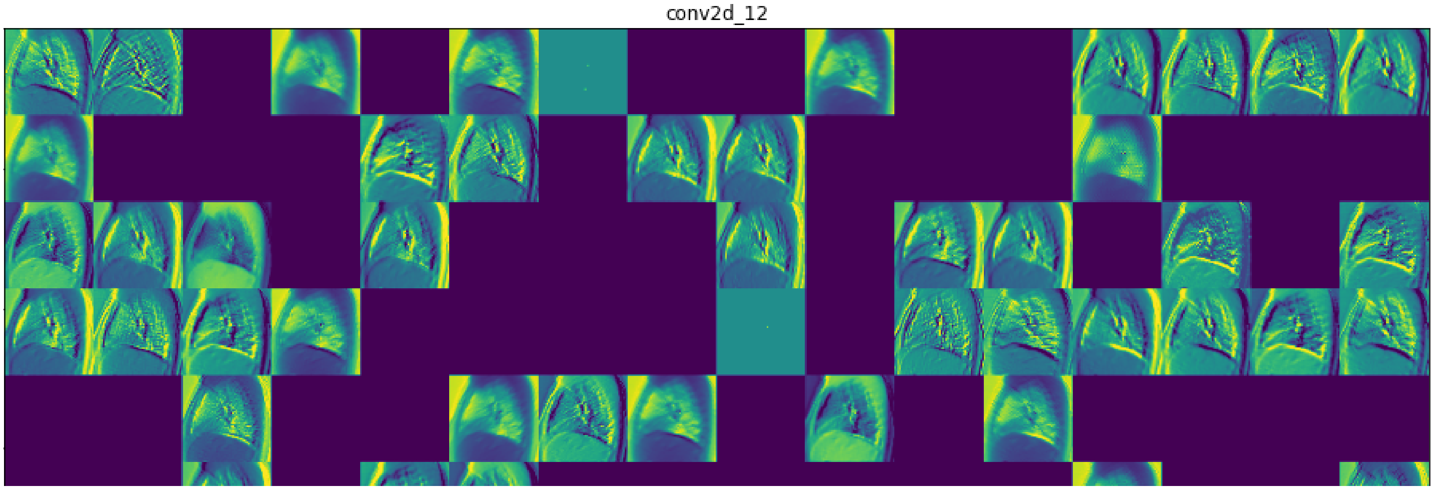
```
    0.9847741215839375
```

```
from keras.preprocessing import image
import numpy as np
```

```
img_tensor = image.img_to_array(x_test[2])
img_tensor = np.expand_dims(img_tensor, axis=0)
# Remember that the model was trained on inputs
# that were preprocessed in the following way:
img_tensor /= 255.
```

```
from keras import models

# Extracts the outputs of the top 8 layers:
layer_outputs = [layer.output for layer in model9.layers[:2]]
# Creates a model that will return these outputs, given the model
activation_model = models.Model(inputs=model9.input, outputs=layer
# This will return a list of 5 Numpy arrays:
# one array per layer activation
activations = activation_model.predict(img_tensor)

first_layer_activation = activations[-1]
print(first_layer_activation.shape)


import keras

# These are the names of the layers, so can have them as part of c
layer_names = []
for layer in model9.layers[:8]:
    layer_names.append(layer.name)

images_per_row = 16

# Now let's display our feature maps
for layer_name, layer_activation in zip(layer_names, activations):
    # This is the number of features in the feature map
    n_features = layer_activation.shape[-1]

    # The feature map has shape (1, size, size, n_features)
    size = layer_activation.shape[1]

    # We will tile the activation channels in this matrix
    n_cols = n_features // images_per_row
    display_grid = np.zeros((size * n_cols, images_per_row * size)

    # We'll tile each filter into this big horizontal grid
    for col in range(n_cols):
        for row in range(images_per_row):
            channel_image = layer_activation[0,
                                             :, :,
                                             col * images_per_row
            # Post-process the feature to make it visually palatab
            channel_image -= channel_image.mean()
            channel_image /= channel_image.std()
            channel_image *= 64
            channel_image += 128
            channel_image = np.clip(channel_image, 0, 255).astype(
            display_grid[col * size : (col + 1) * size,
                         row * size : (row + 1) * size] = channel
                         row * size : (row + 1) * size] = channel_

    # Display the grid
    scale = 1. / size
    plt.figure(figsize=(scale * display_grid.shape[1],
                        scale * display_grid.shape[0]))
    plt.title(layer_name)
    plt.grid(False)
    plt.imshow(display_grid, aspect='auto', cmap='viridis')

plt.show()
```

```
NING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<loca
 31, 31, 128)
r/local/lib/python3.7/dist-packages/ipykernel_launcher.py:57: RuntimeWarning: invalid valu
```



```
pred_classes = (model9.predict(x_test) > 0.5).astype("int32").rave
```

```
pred_classes
```

```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
      dtype=int32)
```

```
layer_outputs = [layer.output for layer in model9.layers]
activation_model = models.Model(inputs=model9.input, outputs=layer
layer_outputs
```

```
[<KerasTensor: shape=(None, 62, 62, 128) dtype=float32 (created by layer 'conv2d_3')>,
 <KerasTensor: shape=(None, 31, 31, 128) dtype=float32 (created by layer 'max_pooling2d_3
 <KerasTensor: shape=(None, 29, 29, 128) dtype=float32 (created by layer 'conv2d_4')>,
 <KerasTensor: shape=(None, 14, 14, 128) dtype=float32 (created by layer 'max_pooling2d_4
 <KerasTensor: shape=(None, 12, 12, 128) dtype=float32 (created by layer 'conv2d_5')>,
 <KerasTensor: shape=(None, 6, 6, 128) dtype=float32 (created by layer 'max_pooling2d_5')
 <KerasTensor: shape=(None, 4608) dtype=float32 (created by layer 'flatten_1')>,
 <KerasTensor: shape=(None, 256) dtype=float32 (created by layer 'dense_2')>,
 <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'dense_3')>]
```

```
# Get the outputs of all the hidden nodes for each of the 60000 tr
activations = activation_model.predict(x_test)
activations = activation_model.predict(x_test)
hidden_layer_activation = activations[7]
output_layer_activations = activations[8]
hidden_layer_activation.shape   #  each of the 128 hidden nodes ha
```

```
(328, 256)
```

```
#Get the dataframe of all the node values
activation_data = {'pred_class':pred_classes[0:328]}
for k in range(0,256):
    activation_data[f"act_val_{k}"] = hidden_layer_activation[:,k]


activation_df = pd.DataFrame(activation_data)
activation_df.head()
```

	pred_class	act_val_0	act_val_1	act_val_2	act_val_3	act_val_4	act_val_5	act_val_6
0	0	0.0	0.261437	0.0	0.0	2.302884	0.0	0.0
1	0	0.0	0.000000	0.0	0.0	2.508843	0.0	0.0
2	0	0.0	0.000000	0.0	0.0	2.587324	0.0	0.0
3	0	0.0	0.000000	0.0	0.0	2.679549	0.0	0.0
4	0	0.0	0.000000	0.0	0.0	3.288160	0.0	0.0

5 rows × 257 columns

```
# Separating out the features
features = [*activation_data][1:] # ['act_val_0', 'act_val_1',...]
x = activation_df.loc[:, features].values

pca = PCA(n_components=3)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['pca-one', 'pca-two', 'pca-three'])
principalDf.head()
```

	pca-one	pca-two	pca-three
0	-5.853893	2.599480	1.266800
1	-9.155988	1.404219	-0.198507
2	-10.286340	2.204174	-0.435203
3	-8.915661	1.304460	0.064735
4	-12.442112	5.414866	-0.132032

```
activation_pca_df = pd.concat([principalDf, activation_df[['pred_c
activation_pca_df.head()
```

	pca-one	pca-two	pca-three	pred_class
0	-5.853893	2.599480	1.266800	0
1	-9.155988	1.404219	-0.198507	0
2	-10.286340	2.204174	-0.435203	0
3	-8.915661	1.304460	0.064735	0
4	-12.442112	5.414866	-0.132032	0

```
N=10000
activation_df_subset = activation_df.iloc[:N].copy()
activation_df_subset.shape
```

```
(328, 257)
```

```
data_subset = activation_df_subset[features].values
data_subset.shape
```

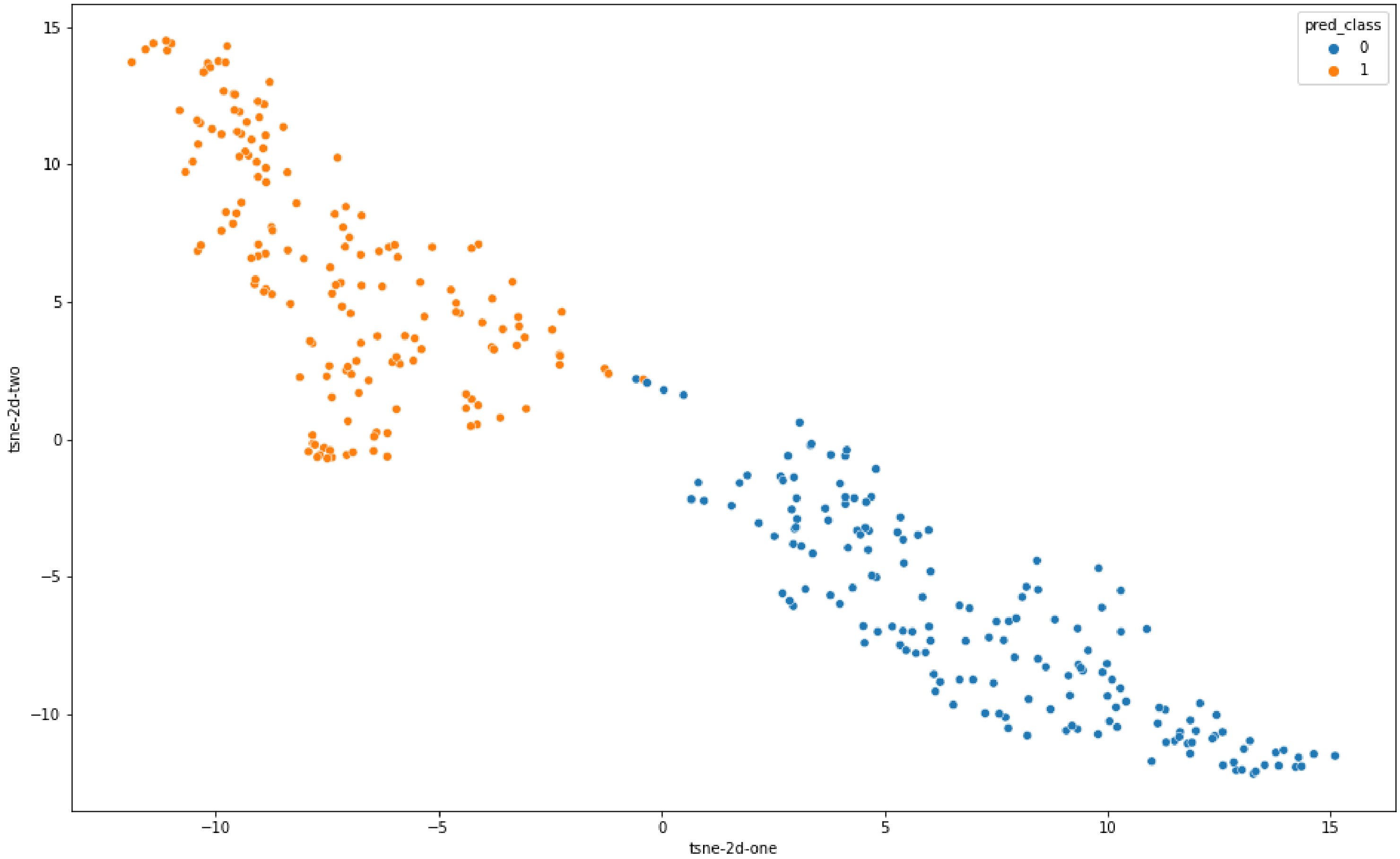
```
(328, 256)
```

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(data_subset)
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 328 samples in 0.004s...
[t-SNE] Computed neighbors for 328 samples in 0.040s...
[t-SNE] Computed conditional probabilities for sample 328 / 328
[t-SNE] Mean sigma: 1.609862
[t-SNE] KL divergence after 250 iterations with early exaggeration: 52.930374
[t-SNE] KL divergence after 300 iterations: 0.244041
```

```
activation_df_subset['tsne-2d-one'] = tsne_results[:,0]
activation_df_subset['tsne-2d-two'] = tsne_results[:,1]

plt.figure(figsize=(16,10))
sns.scatterplot(
    x="tsne-2d-one", y="tsne-2d-two",
    hue="pred_class",
    palette=sns.color_palette(n_colors = 2),
    data=activation_df_subset,
    legend="full",
    alpha = 1
)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb30d64a990>
```



Experiment 10

```
model10 = models.Sequential()
model10.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model10.add(layers.MaxPooling2D((2, 2),strides=2))
model10.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model10.add(layers.MaxPooling2D((2, 2),strides=2))
model10.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model10.add(layers.MaxPooling2D((2, 2),strides=2))
model10.add(layers.Flatten())
model10.add(layers.Dense(units=256, activation=tf.nn.relu))
model10.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model10.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history10 = model10.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model10.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 11

```
model11 = models.Sequential()
model11.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model11.add(layers.MaxPooling2D((2, 2),strides=2))
model11.add(layers.Dropout(.2))
model11.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model11.add(layers.MaxPooling2D((2, 2),strides=2))
model11.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model11.add(layers.MaxPooling2D((2, 2),strides=2))
model11.add(layers.Flatten())
model11.add(layers.Dense(units=256, activation=tf.nn.relu))
model11.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model11.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history11 = model11.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                    batch_size=512
                   )

loss, accuracy = model11.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 12

```
model12 = models.Sequential()
model12.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model12.add(layers.MaxPooling2D((2, 2),strides=2))
model12.add(layers.Dropout(.2))
model12.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model12.add(layers.MaxPooling2D((2, 2),strides=2))
model11.add(layers.Dropout(.2))
model12.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model12.add(layers.MaxPooling2D((2, 2),strides=2))
model12.add(layers.Flatten())
model12.add(layers.Dense(units=256, activation=tf.nn.relu))
model12.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model12.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history12 = model12.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model12.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 13

```
model13 = models.Sequential()
```

```
model13 = models.Sequential()
model13.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model13.add(layers.MaxPooling2D((2, 2),strides=2))
model13.add(layers.Dropout(.2))
model13.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model13.add(layers.MaxPooling2D((2, 2),strides=2))
model13.add(layers.Dropout(.2))
model13.add(layers.Conv2D(filters=128, kernel_size=(3, 3), strides
model13.add(layers.MaxPooling2D((2, 2),strides=2))
model13.add(layers.Dropout(.2))
model13.add(layers.Flatten())
model13.add(layers.Dense(units=256, activation=tf.nn.relu))
model13.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model13.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history13 = model13.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512
                   )
```

```
loss, accuracy = model13.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 14

```
model14 = models.Sequential()
model14.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model14.add(layers.MaxPooling2D((2, 2),strides=2))
model14.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model14.add(layers.MaxPooling2D((2, 2),strides=2))
model14.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model14.add(layers.MaxPooling2D((2, 2),strides=2))
model14.add(layers.Flatten())
model14.add(layers.Dense(units=256, activation=tf.nn.relu))
model14.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model14.add(layers.Dense(units=1, activation=tf.nn.sigmoid))

model14.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history14 = model14.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512,
                   callbacks = [tf.keras.callbacks.EarlyStopping(m
```

```
loss, accuracy = model14.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 15

```
model15 = models.Sequential()
model15.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model15.add(layers.MaxPooling2D((2, 2),strides=2))
model15.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model15.add(layers.MaxPooling2D((2, 2),strides=2))
model15.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model15.add(layers.MaxPooling2D((2, 2),strides=2))
model15.add(layers.Flatten())
model15.add(layers.Dense(units=256, activation=tf.nn.relu))
model15.add(layers.Dense(units=1, activation=tf.nn.sigmoid))
```

```
model15.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history15 = model15.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512,
                    batch_size=512,
                   callbacks = [tf.keras.callbacks.EarlyStopping(m

loss, accuracy = model15.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

Experiment 16

```
model16 = models.Sequential()
model16.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model16.add(layers.MaxPooling2D((2, 2),strides=2))
model16.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model16.add(layers.MaxPooling2D((2, 2),strides=2))
model16.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model16.add(layers.MaxPooling2D((2, 2),strides=2))
model16.add(layers.Flatten())
model16.add(layers.Dense(units=256, activation=tf.nn.relu))
model16.add(layers.Dense(units=1, activation=tf.nn.sigmoid, kernel

model16.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
              metrics=['accuracy'])
```

```
 %%time
history16 = model16.fit(X_train,
                    y_train,
                    validation_data = (X_val, y_val),
                    epochs=20,
                    batch_size=512)
```

```
Epoch 1/20
4/4 [==============================] - 47s 2s/step - loss: 1.0549 - accuracy: 0.4987 - va
Epoch 2/20
4/4 [==============================] - 1s 262ms/step - loss: 0.8844 - accuracy: 0.5331 -
Epoch 3/20
4/4 [==============================] - 1s 262ms/step - loss: 0.8499 - accuracy: 0.7346 -
Epoch 4/20
4/4 [==============================] - 1s 260ms/step - loss: 0.7749 - accuracy: 0.6703 -
Epoch 5/20
4/4 [==============================] - 1s 263ms/step - loss: 0.6602 - accuracy: 0.8174 -
Epoch 6/20
4/4 [==============================] - 1s 256ms/step - loss: 0.4749 - accuracy: 0.8920 -
Epoch 7/20
4/4 [==============================] - 1s 260ms/step - loss: 0.3564 - accuracy: 0.9365 -
```

```
Epoch 8/20
4/4 [==============================] - 1s 260ms/step - loss: 0.3122 - accuracy: 0.9358 -
Epoch 9/20
4/4 [==============================] - 1s 259ms/step - loss: 0.3390 - accuracy: 0.9288 -
Epoch 10/20
4/4 [==============================] - 1s 261ms/step - loss: 0.2863 - accuracy: 0.9482 -
Epoch 11/20
4/4 [==============================] - 1s 260ms/step - loss: 0.2545 - accuracy: 0.9641 -
Epoch 12/20
4/4 [==============================] - 1s 261ms/step - loss: 0.2311 - accuracy: 0.9642 -
Epoch 13/20
4/4 [==============================] - 1s 266ms/step - loss: 0.1969 - accuracy: 0.9788 -
Epoch 14/20
4/4 [==============================] - 1s 263ms/step - loss: 0.2070 - accuracy: 0.9730 -
Epoch 15/20
4/4 [==============================] - 1s 261ms/step - loss: 0.1974 - accuracy: 0.9742 -
Epoch 16/20
4/4 [==============================] - 1s 259ms/step - loss: 0.1684 - accuracy: 0.9816 -
Epoch 17/20
4/4 [==============================] - 1s 263ms/step - loss: 0.1659 - accuracy: 0.9813 -
Epoch 18/20
4/4 [==============================] - 1s 267ms/step - loss: 0.1605 - accuracy: 0.9809 -
Epoch 19/20
4/4 [==============================] - 1s 267ms/step - loss: 0.1694 - accuracy: 0.9806 -
Epoch 20/20
4/4 [==============================] - 1s 263ms/step - loss: 0.1550 - accuracy: 0.9809 -
CPU times: user 19.5 s, sys: 14.3 s, total: 33.8 s
Wall time: 1min 7s
```

```
loss, accuracy = model16.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

```
11/11 [==============================] - 1s 22ms/step - loss: 0.1612 - accuracy: 0.9787
test set accuracy:  97.86585569381714
```

Experiment 17

```
model17 = models.Sequential()
model17.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model17.add(layers.MaxPooling2D((2, 2),strides=2))
model17.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model17.add(layers.MaxPooling2D((2, 2),strides=2))
model17.add(layers.Conv2D(filters=256, kernel_size=(3, 3), strides
model17.add(layers.MaxPooling2D((2, 2),strides=2))
model17.add(layers.Flatten())
model17.add(layers.Dense(units=256, activation=tf.nn.relu))
model17.add(layers.Dense(units=1, activation=tf.nn.sigmoid, kernel

model17.compile(optimizer='adam',
              loss=tf.keras.losses.BinaryCrossentropy(),
                metrics=['accuracy'])
```

```
 %%time
history17 = model17.fit(X_train,
                        y_train,
                        validation_data = (X_val, y_val),
                        epochs=20,
                        batch_size=512)
```

```
Epoch 1/20
4/4 [==============================] - 2s 323ms/step - loss: 0.8549 - accuracy: 0.5537 -
Epoch 2/20
4/4 [==============================] - 1s 270ms/step - loss: 0.7012 - accuracy: 0.4921 -
Epoch 3/20
4/4 [==============================] - 1s 269ms/step - loss: 0.6303 - accuracy: 0.6534 -
Epoch 4/20
4/4 [==============================] - 1s 272ms/step - loss: 0.6281 - accuracy: 0.6660 -
Epoch 5/20
4/4 [==============================] - 1s 273ms/step - loss: 0.5481 - accuracy: 0.7703 -
Epoch 6/20
4/4 [==============================] - 1s 272ms/step - loss: 0.3985 - accuracy: 0.8938 -
Epoch 7/20
4/4 [==============================] - 1s 272ms/step - loss: 0.2443 - accuracy: 0.9416 -
Epoch 8/20
4/4 [==============================] - 1s 276ms/step - loss: 0.2414 - accuracy: 0.9226 -
Epoch 9/20
4/4 [==============================] - 1s 276ms/step - loss: 0.2119 - accuracy: 0.9321 -
Epoch 10/20
4/4 [==============================] - 1s 275ms/step - loss: 0.1519 - accuracy: 0.9507 -
Epoch 11/20
4/4 [==============================] - 1s 269ms/step - loss: 0.1328 - accuracy: 0.9552 -
Epoch 12/20
4/4 [==============================] - 1s 276ms/step - loss: 0.1518 - accuracy: 0.9455 -
Epoch 13/20
4/4 [==============================] - 1s 275ms/step - loss: 0.1502 - accuracy: 0.9485 -
Epoch 14/20
4/4 [==============================] - 1s 273ms/step - loss: 0.1332 - accuracy: 0.9558 -
Epoch 15/20
4/4 [==============================] - 1s 280ms/step - loss: 0.0969 - accuracy: 0.9733 -
Epoch 16/20
4/4 [==============================] - 1s 273ms/step - loss: 0.0864 - accuracy: 0.9765 -
Epoch 17/20
4/4 [==============================] - 1s 272ms/step - loss: 0.0896 - accuracy: 0.9742 -
Epoch 18/20
4/4 [==============================] - 1s 272ms/step - loss: 0.0676 - accuracy: 0.9834 -
Epoch 19/20
4/4 [==============================] - 1s 273ms/step - loss: 0.0825 - accuracy: 0.9776 -
Epoch 20/20
4/4 [==============================] - 1s 275ms/step - loss: 0.0590 - accuracy: 0.9851 -
CPU times: user 11.5 s, sys: 7.28 s, total: 18.8 s
Wall time: 23.7 s
```

```
loss, accuracy = model17.evaluate(x_test, y_test)
print('test set accuracy: ', accuracy * 100)
```

```
11/11 [==============================] - 0s 8ms/step - loss: 0.0659 - accuracy: 0.9817
```

```
test set accuracy:  98.17073345184326
```