Gurjus Singh

MSDS 458 – Artificial Intelligence and Deep Learning

March 14th, 2021

Week 10: A.4 Fourth Research/Programming Assignment

Abstract

In this research, I came up with a way to classify X-Ray Chest Images with COVID-19. The goal of this research was to use supervised learning binary classification with Normal and COVID-19 X-Ray Images. I used a CNN architecture as this was important in image classification. I used accuracy for deciding which model was performing the best, but also looked at other metrics in my best model to see how it was performing. I also performed T-SNE on my best model as well as saw how the Convolution/Max Pooling by making a channel grid.

Introduction

In purpose of this research was to use supervised learning binary classification to classify X-Ray Chest Images of suspected COVID-19 patients. The dataset I used was from Kaggle and contained 2000 train images and 328 test images divided evenly between COVID Positive and COVID Negative images. The Neural Network architecture I used was a Convolution Neural Network architecture as this was suitable for image data. I tried out different CNN architectures before choosing my best model. I also decided to find out what features the layers of the CNN were learning by creating a grid of channels. I also decided to perform T-SNE on my best model. After experimenting with my best model, I then made a management recommendation.

Literature Review

As COVID-19 spreads, and vaccines are distributed, there continues to be growing research on how to diagnose patients. One such paper that I found, talked about how countries have adopted reverse transcription polymerase chain reaction to diagnose COVID-19 [1]. The authors of the paper point out this test takes 4-6 hours or even a whole day to get results [1]. The test can also give false positives and false negatives [1]. The authors then point out that one solution to preventing false positives/negatives is to test COVID-19 infections by using XRAY/CT Scans of the Chest in COVID patients [1].

The paper goes on to share the results of XRAY/CT Scan classification [1]. They use multi-image classification to decrease overfitting [1]. This is a way to generate more images for the CNN architecture to train on [1]. In the results of the experiments with a 70 percent-30 percent training-testing split the authors obtained a 95.38 percent accuracy for CT Scans, while for X-ray they obtained a 98.97 percent accuracy [1].

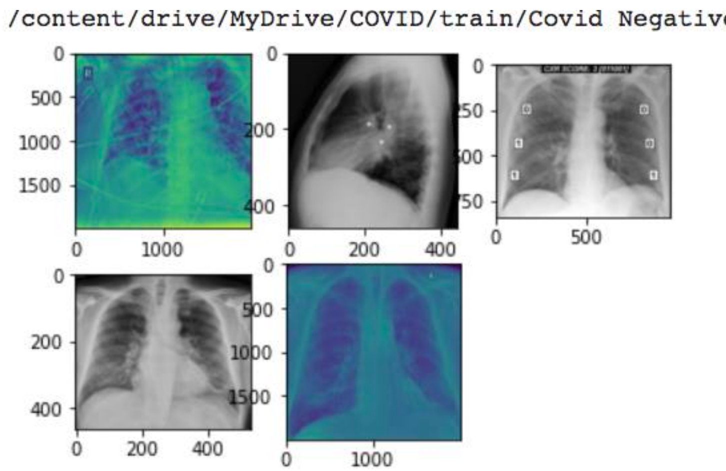
Methods

I first start out this research by importing the necessary packages to complete this project. I found the keras and tensoflow packages important for implementing the CNN architecture. I also found sklearn package important for computing metrics. I also found matplotlib package important for creating the EDA, and the os package was important for extracting the files from the directories. The imported packages are shown below in 1-1.

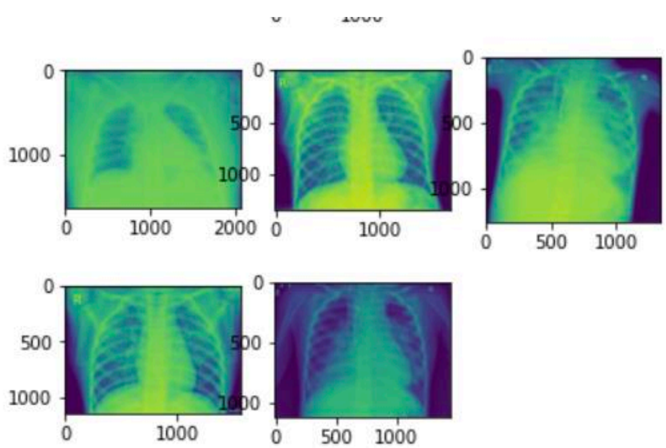
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D , MaxPool2D , Flatten , Dropout
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam

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
```

Import Packages 1-1

After importing the packages my next step was to do some EDA. I plotted the first 5 images of COVID-POSITIVE patients as seen in 1-2 and plotted first 5 photos of COVID NEGATIVE patients as seen in 1-3.

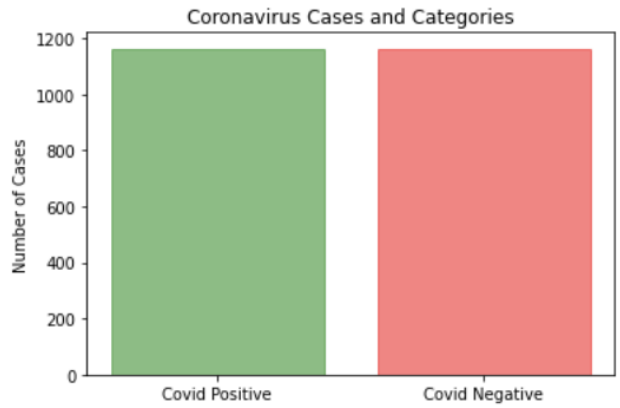


COVID-POSITIVE PATIENT XRAYS 1-2



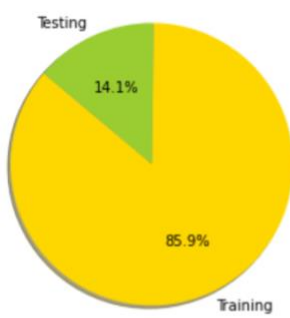
COVID-NEGATIVE PATIENT XRAYS 1-2

After taking an initial look at the data, the next thing I did is loaded the data in the RGB format and also created labels which we will use for training. I also resized each image to 64 x64. After loading of the images and creation of the labels, I looked how many COVID-POSTIVE and NEGATIVE X-RAY Images were in the dataset as seen in 1-3. I noticed there were even number of both sets of images.



Number of COVID CASES 1-3

I also wanted to see what the Train and Test split percentage was as it was already split when taking the dataset from Kaggle. What I noticed was that it was a 85.9 percent to 14.1 percent Train to Test split as seen in 1-4.



Training-Test Split 1-4

After looking at the split I then extracted the labels I created and encoded and stored them in an array, I did the same with the images. I then normalized the images by dividing each image by 255 to get it between 0 and 1. After normalizing the images, I then added more images to the training data to prevent overfitting by using an ImageDataGenerator. This is known as Data Augmentation, and it creates more images by manipulating the existing images to create more images for training. After doing data augmentation, I then split the training data 80 percent to 20 percent to create a validation set.

I then went ahead to create my CNN Architectures which involved Max Pooling, Convolution and Average Pooling layers. Max pooling helps in reducing the number of parameters in the network by “downsizing feature maps” by only extracting the important features of an image[2]. The convolutional layer functions in that tries to extract local patterns from a window [2]. The window is called a filter which goes over each part of the image extracting the features in the image [2]. Average pooling retains less important information of an image by averaging the features [4].

After building each CNN Architecture, I then observed the accuracy and obtained the best model. On the best model, I performed TSNE, also built a grid for each layer in the CNN architecture, and computed other metrics such as F1 Score, Precision Score etc.

Results

In the research I performed a total of **17 experiments, 20 Epochs each**, and also performed regularization techniques such as L1, L2 Regularizers, Dropout and Early Stopping. My Performance Summary is below in 1-5. What I noticed from these experiments was that **Experiment 9 with 3 Max Pooling, 3 Convolution Layers with 128 Filters; and 1 Dense Layer with 256 Neurons** was performing the best. I noticed it’s testing accuracy was 98.47 while Training accuracy was 98.50 which means it was not overfitting. This was the highest testing accuracy that I had among the 17 models which was why I chose this model.

EXPERIMENTS - 20 EPOCHS EACH;	Regularization	TRAIN ACCURACY	VAL ACCURACY	TEST ACCURACY	TRAIN LOSS	VAL LOSS	TEST LOSS	TIME (HR, MIN, SECS)
1 - 1 MAX POOLING AND CONV LAYER - 32 FILTERS; 1 DENSE LAYER 32 NEURONS	None	97.62%	96.25%	96.03%	0.0725	0.0823	0.1094	4.23 SECS
2 - 1 MAX POOLING AND CONV LAYER - 64 FILTERS; 1 DENSE LAYER 64 NEURONS	None	98.50%	97.50%	96.95%	0.0532	0.0634	0.0866	5.04 SECS
3 - 1 MAX POOLING AND CONV LAYER - 128 FILTERS; 1 DENSE LAYER 128 NEURONS	None	98%	96.75%	96.03%	6.52E-02	0.0752	0.0958	7.1 SECS
4 - 2 MAX POOLING AND CONV LAYER - 32 FILTERS; 1 DENSE LAYER 256 NEURONS	None	98%	97.75%	96.95%	6.39E-02	0.0564	0.0762	4.54 SECS
5 - 2 MAX POOLING AND CONV LAYER - 64 FILTERS; 1 DENSE LAYER 256 NEURONS	None	98%	97.75%	97.25%	0.0584	0.658	0.0834	6.1 SECS
6 - 2 MAX POOLING AND CONV LAYER - 128 FILTERS; 1 DENSE LAYER 256 NEURONS	None	98%	96.25%	97.86%	0.0603	0.0714	0.0614	9.6 SECS
7 - 3 MAX POOLING AND CONV LAYER - 64 FILTERS; 1 DENSE LAYER 256 NEURONS	None	98.45%	99.00%	99.08%	0.0442	0.0342	0.0384	6 SECS
8 - 3 AVERAGE POOLING AND CONV LAYER - 64 FILTERS; 1 DENSE LAYER 256 NEURONS	None	97.74%	97.75%	94.82%	0.0674	0.0655	0.1216	6.05 SECS
9 - 3 MAX POOLING AND CONV LAYER - 128 FILTERS; 1 DENSE LAYER 256 NEURONS	None	98.50%	98.25%	98.47%	0.0514	0.053	0.0485	9.32 SECS
10 - 3 MAX POOLING AND CONV LAYER - 256 FILTERS; 1 DENSE LAYER 256 NEURONS	None	97.61%	98.75%	97.56%	0.0643	0.0338	0.0577	21.2 SECS
11 - 3 MAX POOLING AND CONV LAYER - 128 FILTERS; 1 DENSE LAYER 256 NEURONS	1 DROPOUT LAYER 20%	88.50%	97.25%	97.56%	1.5564	0.0753	0.083	9.29 SECS
12 - 3 MAX POOLING AND CONV LAYER - 128 FILTERS; 1 DENSE LAYER 256 NEURONS	2 DROPOUT LAYER 20%	98.33%	99.00%	96.95%	0.0516	0.0365	0.0777	9.7 SECS
13 - 3 MAX POOLING AND CONV LAYER - 128 FILTERS; 1 DENSE LAYER 256 NEURONS	3 DROPOUT LAYER 20%	98.66%	97.75%	96.95%	0.0574	0.0518	0.0872	9.4 SECS
14- 3 MAX POOLING AND CONV LAYER - 256 FILTERS; 1 DENSE LAYER 256 NEURONS	EARLY STOPPIN PATIENCE 1	55.99%	49.75%	50.00%	0.6849	0.7115	0.7338	2.4 SECS
15 - 3 MAX POOLING AND CONV LAYER - 256 FILTERS; 1 DENSE LAYER 256 NEURONS	EARLY STOPPIN PATIENCE 2	94.39%	94.50%	94.21%	0.161	0.1336	0.1306	8.26 SECS
16 - 3 MAX POOLING AND CONV LAYER - 256 FILTERS; 1 DENSE LAYER 256 NEURONS	L1 REGULARIZER	98.09%	97.25%	97.86%	0.155	0.1615	0.1612	33.8 SECS
17 - 3 MAX POOLING AND CONV LAYER - 256 FILTERS; 1 DENSE LAYER 256 NEURON	L2 REGULARIZER	98.51%	98.50%	98.17%	0.059	0.0483	0.0659	18.8 SECS

Performance Summary; See Excel Sheet attached for better look 1-5

Next, I looked into the metrics to see how good of a job the model 9 was doing. From the results in 1-6, 1-7. From the Confusion Matrices, I felt like the model was doing a fantastic job. I also noticed that Precision and Recall scores were around 0.98, and the F1 Scores were high as well being close to 1 which is the best score.

```
[ ] 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')

    0.9881320127062698
```

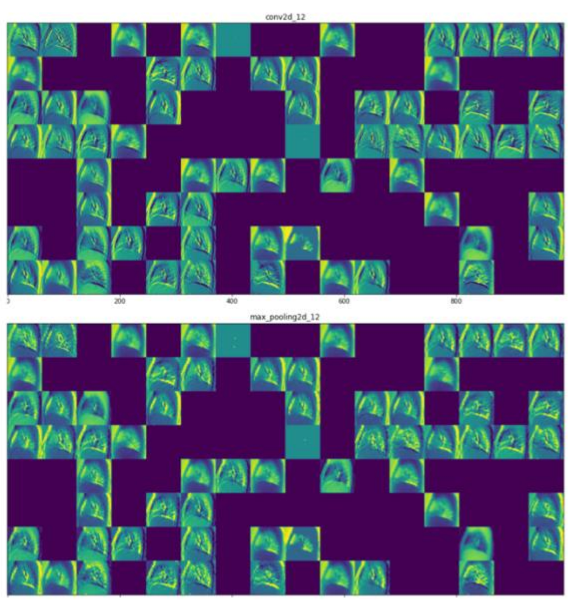
```
[ ] precision_score(y_train, y_pred, average='macro')

    0.9881800680068007
```

Training Set Metrics 1-6

```
[ ] 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
```

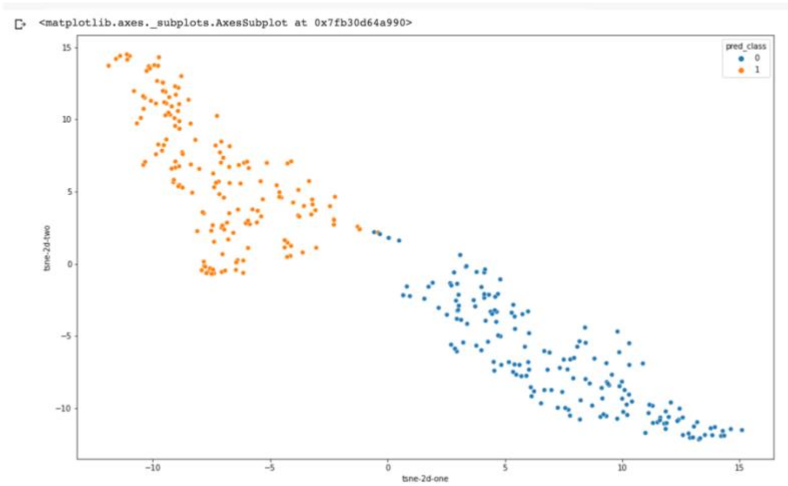
Testing Set Metrics 1-7

After looking at key metrics for my model, I then looked at the grid for how the layers in the architecture learned in 1-8. I noticed the layers was trying to learn the features of the bones in the X-ray.



Grid for Layers 1-8

Lastly, I performed T-SNE, which is dimensionality reduction unsupervised learning technique to visualize higher dimension data as seen in 1-9 [3].



T-SNE on COVID Dataset 1-9

Conclusion

In this experiment, I did binary classification on COVID-19 Xray images using CNN Architecture. From doing 17 experiments I chose Experiment 9 which had an architecture of **3 Max Pooling, 3 Convolution Layers with 128 Filters; and 1 Dense Layer with 256 Neurons** as the best experiment. After choosing the best model, I also computed key metrics in 1-6, 1-7, saw how the layers learned by creating a grid in 1-8, and performed T-SNE in 1-9**. For my recommendation to management,** I choose an architecture of **3 Max Pooling, 3 Convolution Layers with 128 Filters; and 1 Dense Layer with 256 Neurons** to diagnose COVID Patients. In the future, I hope to see if the CNN Architecture can distinguish between other fluid like illnesses like Pneumonia.

References

[1] Purohit, K., Kesarwani, A., Kisku, D. R., & Dalui, M. (2020, January 1). *COVID-19 detection on chest x-ray and ct scan images using multi-image augmented deep learning model*. BioRxiv. https://www.biorxiv.org/content/10.1101/2020.07.15.205567v2.full

[2] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications Company.

[3] Introduction to t-SNE. (n.d.). Data Camp. Retrieved February 6, 2021, from https://www.datacamp.com/community/tutorials/introduction-t-sne

[4] C. (2020, January 30). *What are Max Pooling, Average Pooling, Global Max Pooling and Global Average Pooling? –*. MachineCurve. https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/

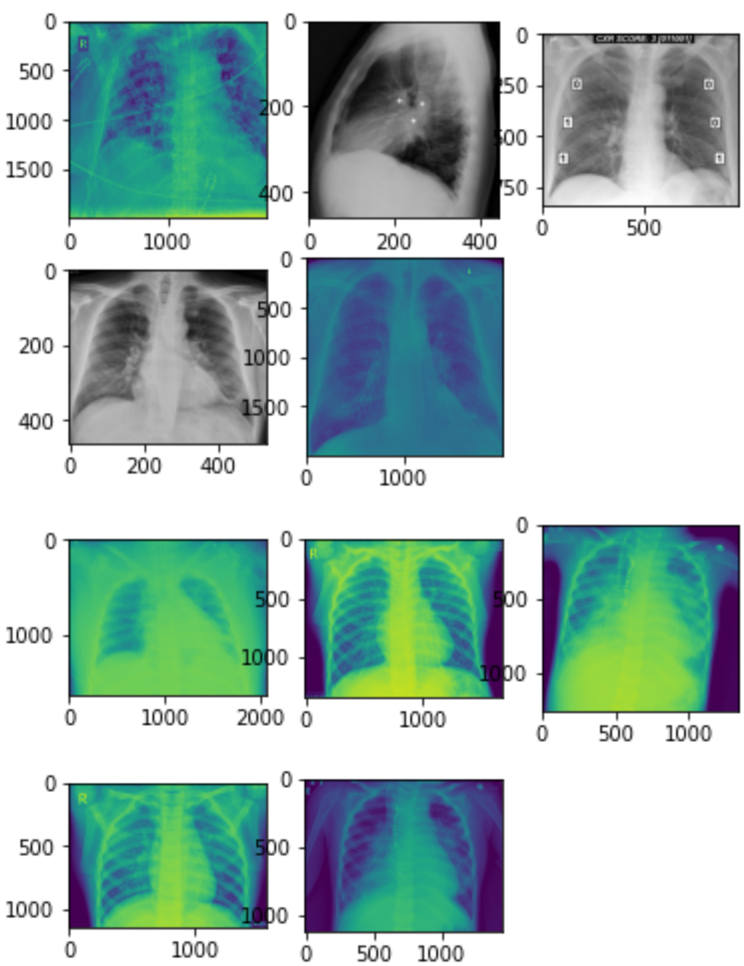
APPENDIX

```
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
```

```python
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)
```

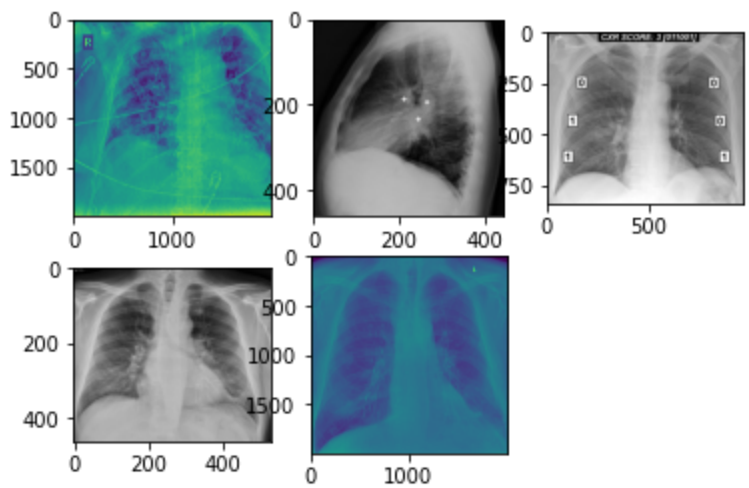
```
/content/drive/MyDrive/COVID/train/Covid Positive/00870a9c.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/000025-1.jpg
/content/drive/MyDrive/COVID/train/Covid Positive/11547_2020_1200_Fig2_HTML-a.png
/content/drive/MyDrive/COVID/train/Covid Positive/000024-1.jpg
/content/drive/MyDrive/COVID/train/Covid 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
```

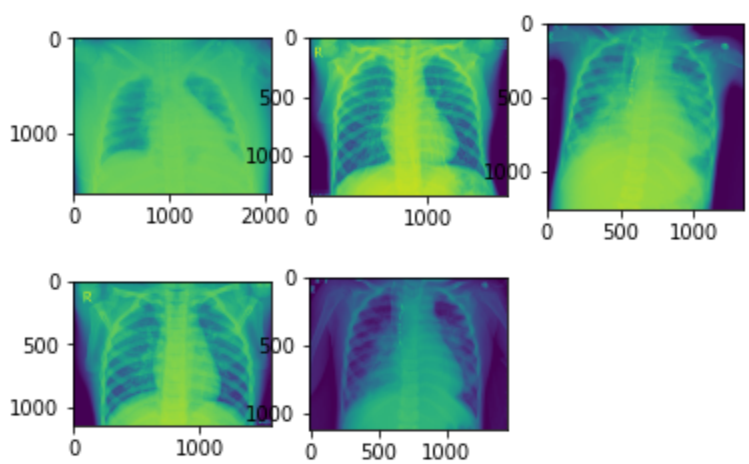


```python
import matplotlib.image as mpimg
directory=os.listdir('/content/drive/MyDrive/COVID/train/')
```

```
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
```





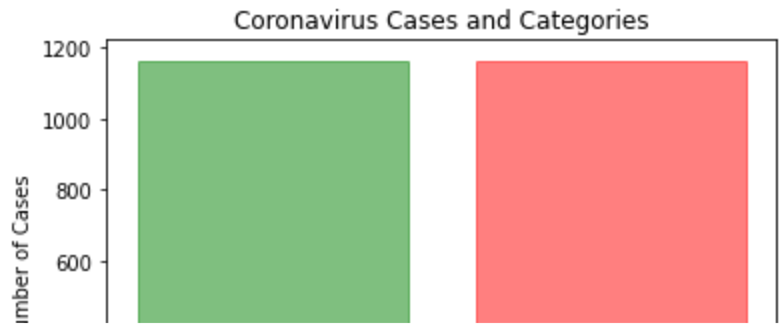
```
labels = ['Covid Negative', 'Covid Positive']
img_size = 64
def get_data(data_dir):
    data = []
    for label in labels:
```

```
        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/')
```

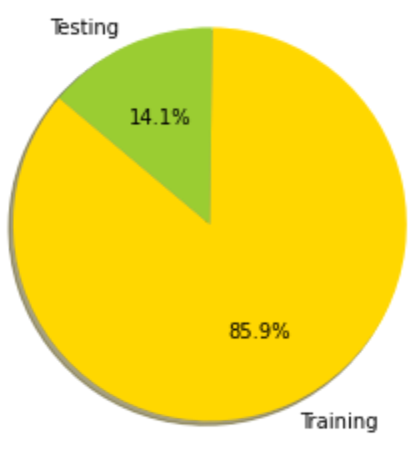
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: VisibleDeprecationWarnir
  from ipykernel import kernelapp as app
```

```
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()
```

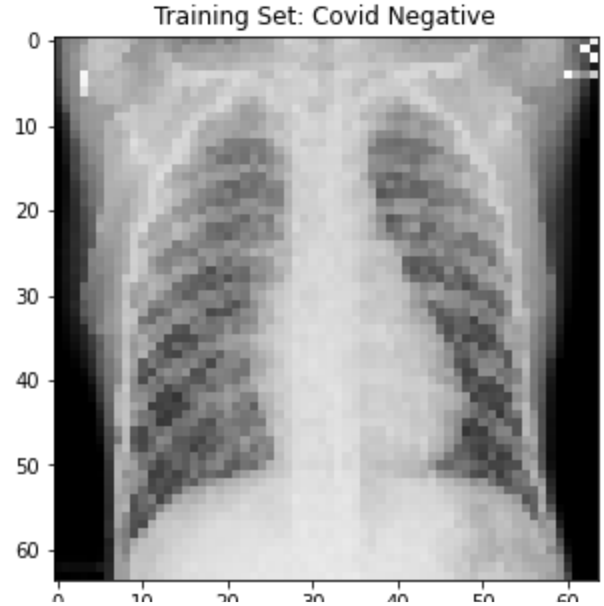



```
path = '/content/drive/MyDrive/COVID/test/Covid Negative'
path1 = '/content/drive/MyDrive/COVID/test/Covid Positive'
path2 = '/content/drive/MyDrive/COVID/train/Covid Negative'
path3 = '/content/drive/MyDrive/COVID/train/Covid Positive'
Test = len([f for f in os.listdir(path)if os.path.isfile(os.path.j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



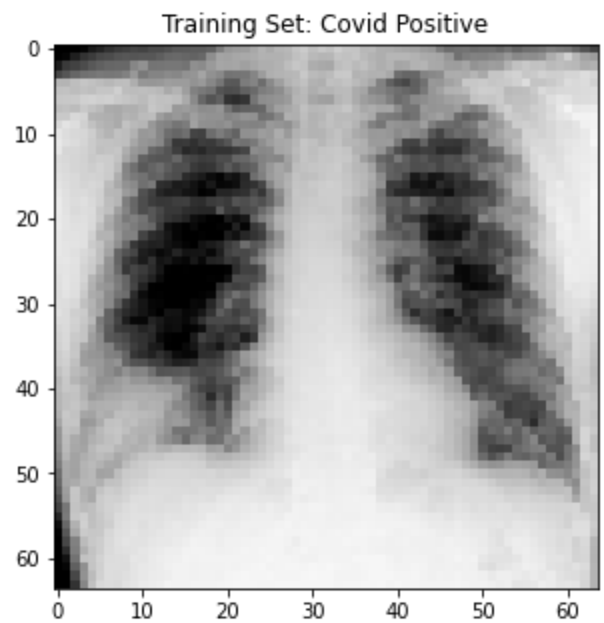

```
plt.figure(figsize = (5,5))
plt.imshow(train[1][0])
plt.title('Training Set: Covid Negative')
```

```
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.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
```

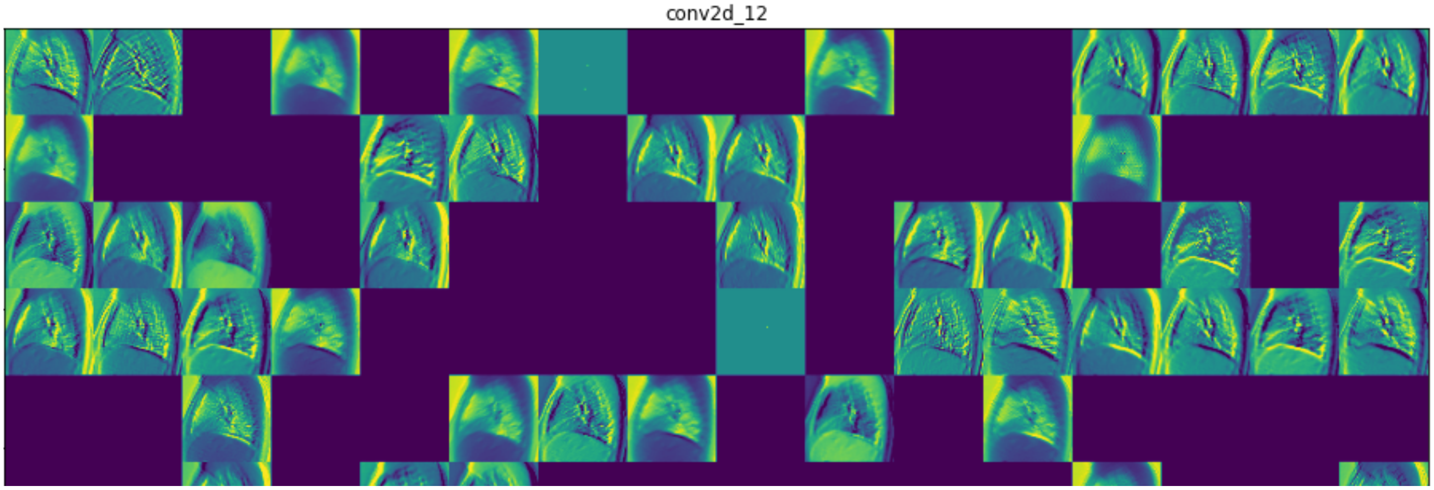
```
img_tensor = image.img_to_array(x_test[2])
img_tensor = np.expand_dims(img_tensor, axis=0)
# Remember that the model was trained on inputs
# that were preprocessed in the following way:
img_tensor /= 255.
```

```
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_

    # 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()
```

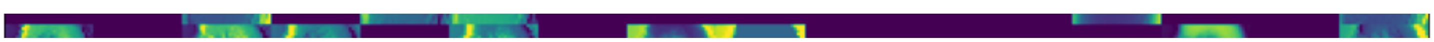
```
NING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<loca
 31, 31, 128)
r/local/lib/python3.7/dist-packages/ipykernel_launcher.py:57: RuntimeWarning: invalid valu
```




```
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)
```

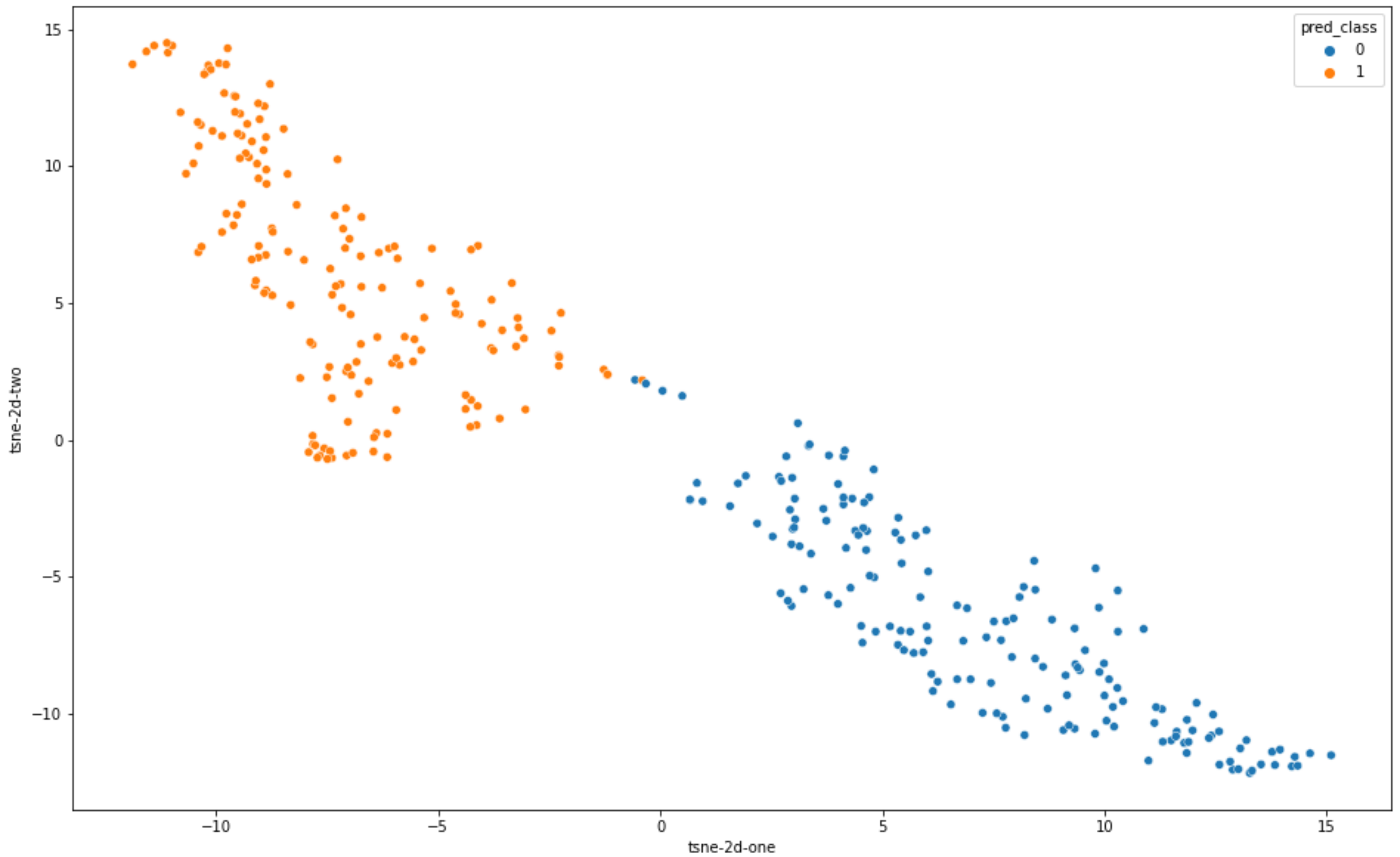
```
from sklearn.manifold import TSNE

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
```

```
                    )

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,
```

```
                    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
```