Classification of Melanoma using Deep Learning and Transfer Learning Neural Networks

MSc Research Project
Data Analytics

Vishal Satishbhai Goyal
Student ID: X18180540

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vishal Satishbhai Goyal		
Student ID:	X18180540		
Programme:	Data Analytics	**Year:**	2019-2020
Module:	MSc Research Project		
Supervisor:	Dr. Catherina Mulwa		
Submission Due Date:	17/08/2020		
Project Title:	Classification of Melanoma Using Deep Learning And Transfer Learning Neural Networks		
Word Count:	485	**Page Count:**	25

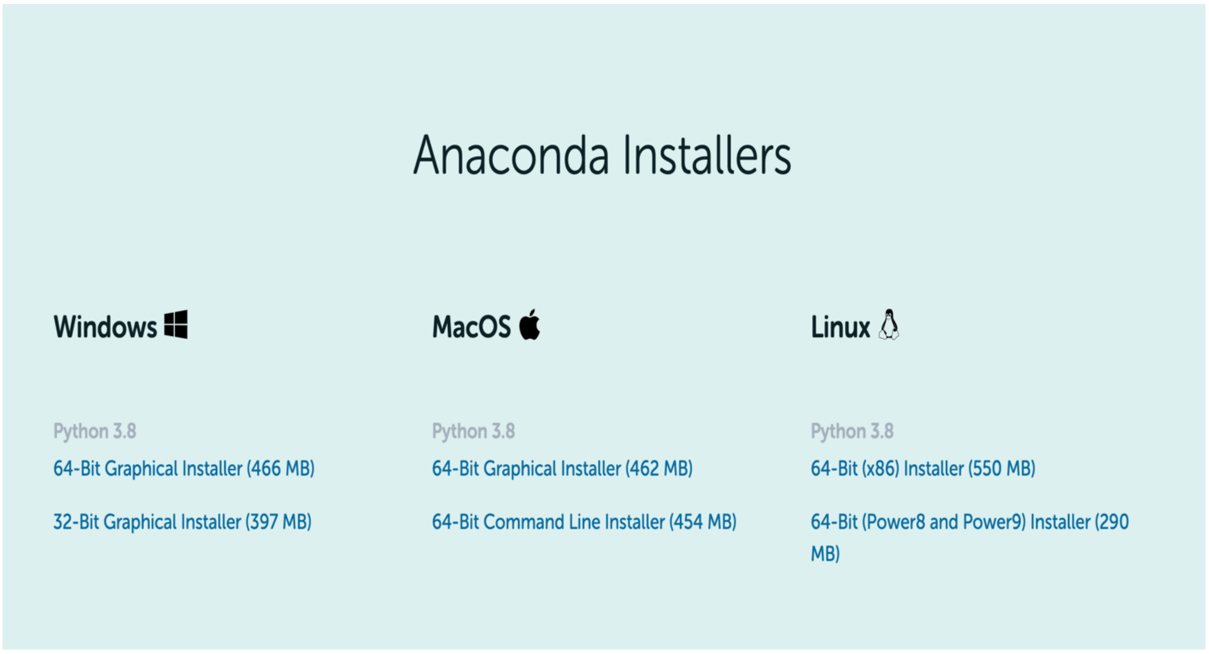
Signature:	Vishal Goyal
Date:	16/08/2020

Configuration Manual

Vishal Satishbhai Goyal
X18180540

1. System and Software requirements

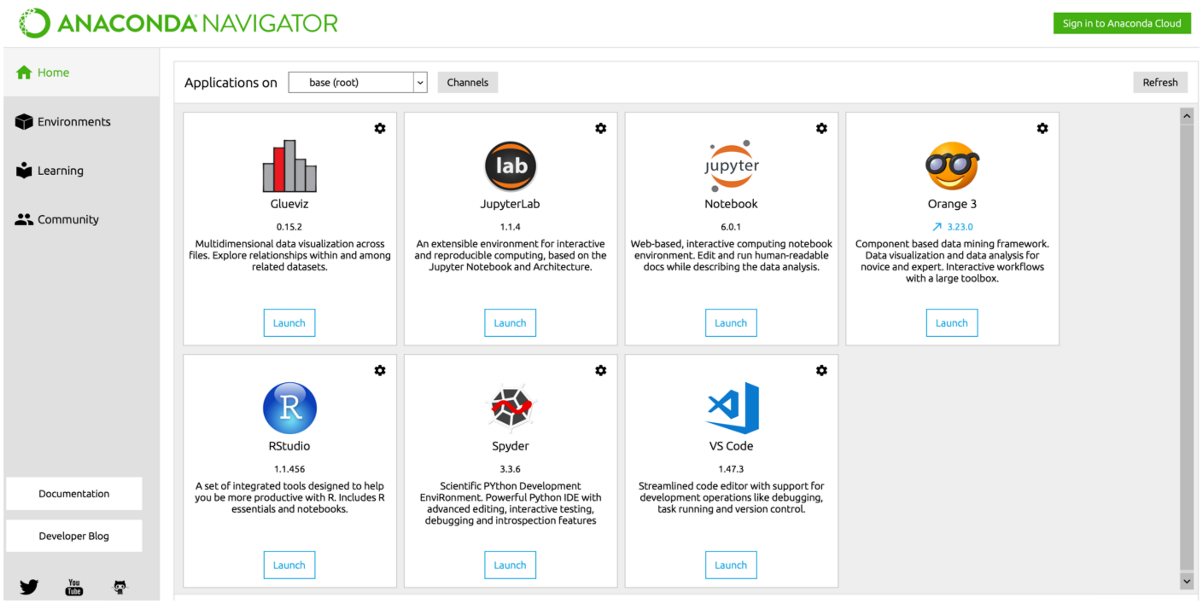
The research project was entirely build using the python 3 environment. The code using python was run using the jupyter notebook on an open source platform called Anaconda Navigator. The system configuration needed for the project to run is 64-bit Mac OS/Windows OS and 8 GB RAM. A 64-bit version of Anaconda Navigator setup is needed which can be downloaded from the website:- https://www.anaconda.com/products/individual#


Anaconda Installers

Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (466 MB)	64-Bit Graphical Installer (462 MB)	64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

The benefits of the Anaconda platform are provided at below link:
https://assets-cdn.anaconda.com/assets/2020-Anaconda-Updated-AE-Datasheet_June2020.pdf?mtime=20200630095556&focal=none

Once the installation is done (For windows) open the start menu and type anaconda navigator. (For Mac OS) search for anaconda navigator and launch it. The next screen that appears is as shown below. There are various navigations and applications which are present on the Anaconda platform which can be integrated like Glueviz for data visualization tool, Rstudio for running R code, Spyder, Jupyter Notebook, JupyterLab for Python code. For this project Jupyter Notebook will be used. The Jupyter notebook will open on the chrome browser on default port 8888.



Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.

JupyterLab
1.1.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook
6.0.1
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Orange 3
3.23.0
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

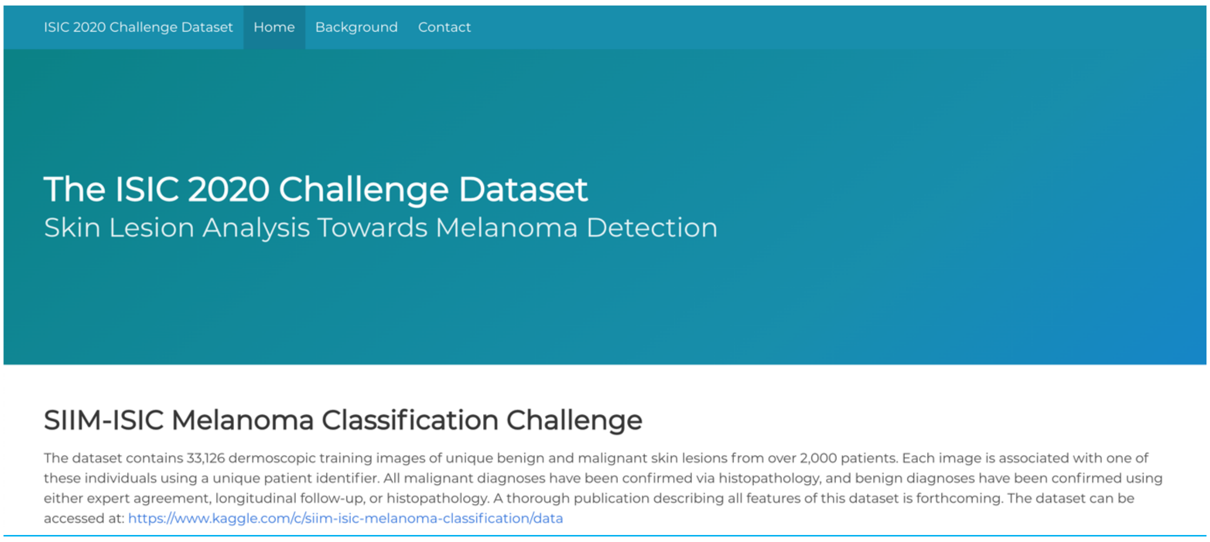
RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Spyder
3.3.6
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code
1.47.3
Streamlined code editor with support for development operations like debugging, task running and version control.

2. Data Collection ([Train] for images and Melanoma.csv)

The Melanoma skin lesion images were taken from the ISIC archive repository. This dataset is shared by ISIC as a challenge for year 2020. The link for the dataset is as below: https://challenge2020.isic-archive.com/

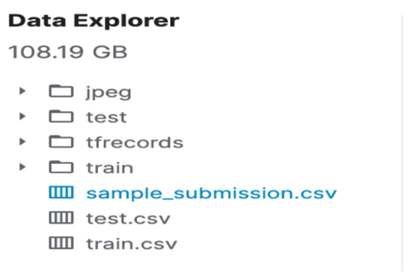


The ISIC 2020 Challenge Dataset

Skin Lesion Analysis Towards Melanoma Detection

SIIM-ISIC Melanoma Classification Challenge

The dataset contains 33,126 dermoscopic training images of unique benign and malignant skin lesions from over 2,000 patients. Each image is associated with one of these individuals using a unique patient identifier. All malignant diagnoses have been confirmed via histopathology, and benign diagnoses have been confirmed using either expert agreement, longitudinal follow-up, or histopathology. A thorough publication describing all features of this dataset is forthcoming. The dataset can be accessed at: https://www.kaggle.com/c/siim-isic-melanoma-classification/data



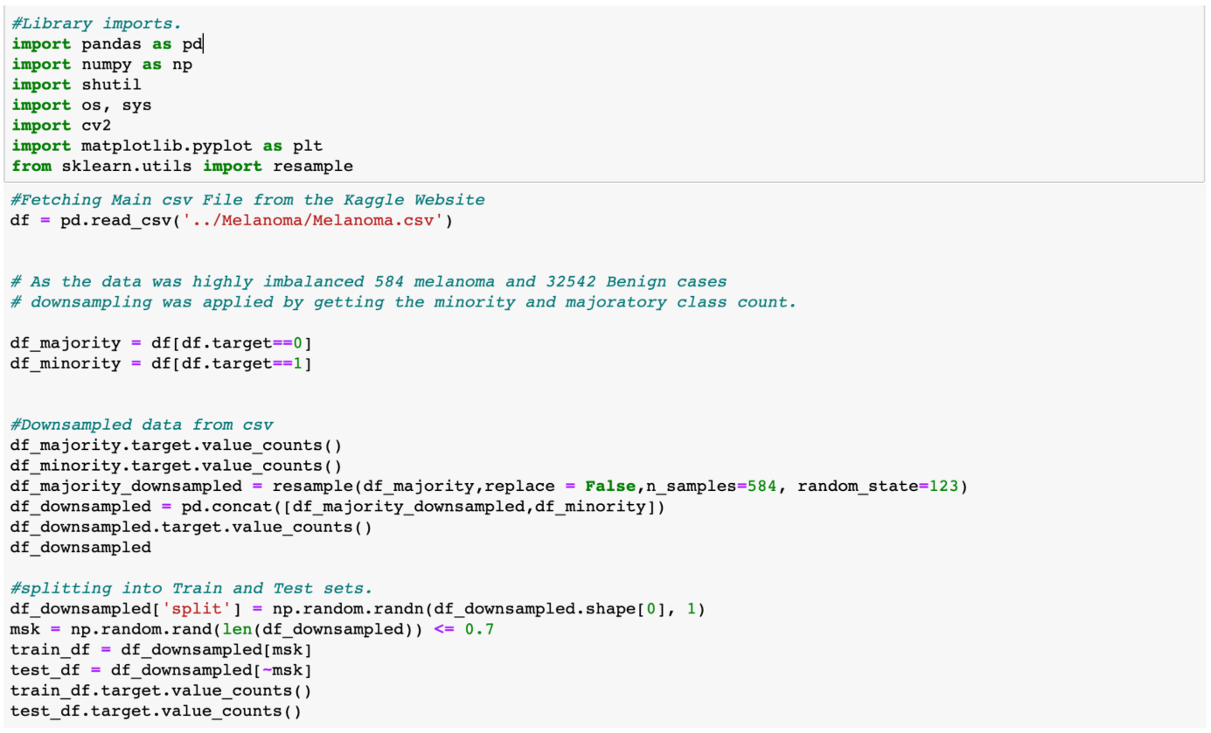
Data Explorer
108.19 GB

- jpeg
- test
- tfrecords
- train
- sample_submission.csv
- test.csv
- train.csv

The images were present in TFRecord, DICOM and JPEG format. The JPEG format images were downloaded. The ground truth for the testing images were not present as it was a competition data, so only the train data is considered for the project. The train.csv file is downloaded and renamed to Melanoma.csv. It contains 33126 images, out of which 584 images are of melanoma and rest are the benign category. The data set was highly imbalanced so downsampling was performed on the dataset.

The JPEG folder contains train and test images, where only the training images were considered and downloaded from the website. The folder name for the images used for the project is contained in train folder. The train.csv files from Kaggle is renamed to Melanoma.csv file in the project folder. First the Melanoma Exploratory Data Analysis.ipynb file should be executed then, the Image Preprocessing and Downsampling.ipynb file should be executed and then rest all model should be executed seperately.

Code for downsampling (Image Preprocessing and Downsampling.ipynb)



```
#Library imports.
import pandas as pd
import numpy as np
import shutil
import os, sys
import cv2
import matplotlib.pyplot as plt
from sklearn.utils import resample
```

```
#Fetching Main csv File from the Kaggle Website
df = pd.read_csv('../Melanoma/Melanoma.csv')


# As the data was highly imbalanced 584 melanoma and 32542 Benign cases
# downsampling was applied by getting the minority and majoratory class count.

df_majority = df[df.target==0]
df_minority = df[df.target==1]


#Downsampled data from csv
df_majority.target.value_counts()
df_minority.target.value_counts()
df_majority_downsampled = resample(df_majority,replace = False,n_samples=584, random_state=123)
df_downsampled = pd.concat([df_majority_downsampled,df_minority])
df_downsampled.target.value_counts()
df_downsampled

#splitting into Train and Test sets.
df_downsampled['split'] = np.random.randn(df_downsampled.shape[0], 1)
msk = np.random.rand(len(df_downsampled)) <= 0.7
train_df = df_downsampled[msk]
test_df = df_downsampled[~msk]
train_df.target.value_counts()
test_df.target.value_counts()
```

```
#Creating Directories for storing the images in folder

os.mkdir("/Users/vishalgoyal/Downloads/TrainImage")
os.mkdir("/Users/vishalgoyal/Downloads/TestImage")
os.mkdir("/Users/vishalgoyal/Downloads/TrainImage/1")
os.mkdir("/Users/vishalgoyal/Downloads/TrainImage/0")
os.mkdir("/Users/vishalgoyal/Downloads/TestImage/1")
os.mkdir("/Users/vishalgoyal/Downloads/TestImage/0")


#Trainset copying images from main image folder to 0 and 1 folder of newly created train folder.
src = "/Users/vishalgoyal/Downloads/Melanoma/train"
traindes1 = "/Users/vishalgoyal/Downloads/TrainImage/1/"
traindes0 = "/Users/vishalgoyal/Downloads/TrainImage/0/"

data_list = train_df.values.tolist()

for i in range(0, len(data_list)):
    if(data_list[i][7] == 0):
        shutil.copy2(src+"/"+data_list[i][0]+".jpg", traindes0)
    else:
        shutil.copy2(src+"/"+data_list[i][0]+".jpg", traindes1)

#Trainset copying images from main image folder to 0 and 1 folder of newly created train folder.
src = "/Users/vishalgoyal/Downloads/Melanoma/train"
testdes0 = "/Users/vishalgoyal/Downloads/TestImage/0/"
testdes1 = "/Users/vishalgoyal/Downloads/TestImage/1/"


data_list = test_df.values.tolist()

for i in range(0, len(data_list)):
    if(data_list[i][7] == 0):
        shutil.copy2(src+"/"+data_list[i][0]+".jpg", testdes0)
    else:
        shutil.copy2(src+"/"+data_list[i][0]+".jpg", testdes1)
```

3. Exploratory Data Analysis (Melanoma Exploratory Data Analysis.ipynb)

The below screenshot shows the YouTube video of melanoma types and symmetry through python code.

What is Melanoma?

Watch this video for medical facts about Melanoma

```
import random
from IPython.display import YouTubeVideo
YouTubeVideo('eHtu4HWFCFM')
```

Skin Cancer Pictures | Pictures O...
Skin Cancer Pictures - Skin Cancer Symptoms

```
#importing libraries

import os
from os import listdir
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

#plotly
!pip install chart_studio
import plotly.express as px
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot
import cufflinks
cufflinks.go_offline()
cufflinks.set_config_file(world_readable=True, theme='pearl')
import seaborn as sns
sns.set(style="whitegrid")


# Settings for pretty nice plots
plt.style.use('fivethirtyeight')
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from skimage.transform import rotate
from skimage.util import random_noise
from skimage.filters import gaussian
from scipy import ndimage
```

2. Reading the Image datasets

```
# Defining data path

random.seed(142)

df = pd.read_csv('Melanoma.csv')
df['split'] = np.random.randn(df.shape[0], 1)

msk = np.random.rand(len(df)) <= 0.7

train_df = df[msk]
test_df = df[~msk]

train_df=train_df.drop(['split'], axis = 1)
test_df=test_df.drop(['split'], axis = 1)

train_df.to_csv(r'Train.csv', index=False)
test_df.to_csv(r'Test.csv', index=False)


#Training data
print('Training data shape: ', train_df.shape)
train_df.head(5)
```

	image_name	patient_id	sex	age_approx	anatom_site_general_challenge	diagnosis	benign_malignant	target
2	ISIC_0052212	IP_2842074	female	50.0	lower extremity	nevus	benign	0
3	ISIC_0068279	IP_6890425	female	45.0	head/neck	unknown	benign	0
4	ISIC_0074268	IP_8723313	female	55.0	upper extremity	unknown	benign	0
5	ISIC_0074311	IP_2950485	female	40.0	lower extremity	unknown	benign	0
6	ISIC_0074542	IP_4698288	male	25.0	lower extremity	unknown	benign	0

```
train_df.groupby(['benign_malignant']).count()['sex'].to_frame()
```

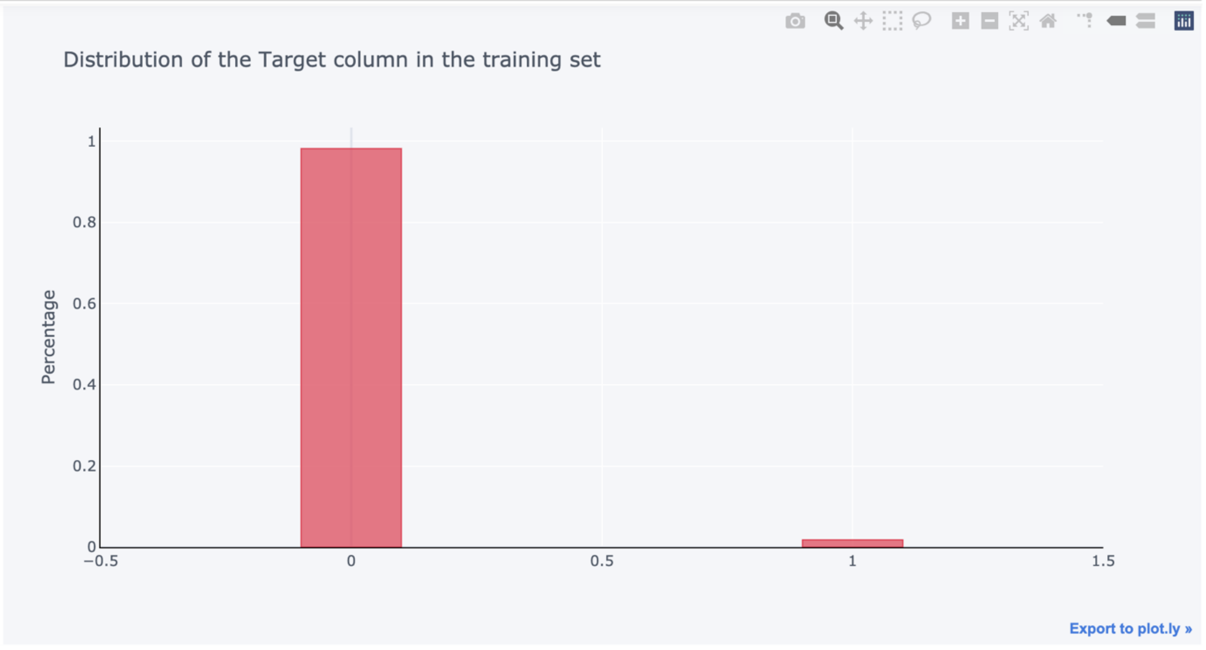
3. Data Exploration

Missing Values

```
# Null values and Data types
print('Train Set')
print(train_df.info())
print('-------------')
print('Test Set')
print(test_df.info())
```

```
Train Set
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23192 entries, 2 to 33125
Data columns (total 8 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   image_name                     23192 non-null  object 
 1   patient_id                     23192 non-null  object 
 2   sex                            23144 non-null  object 
 3   age_approx                     23142 non-null  float64
 4   anatom_site_general_challenge  22817 non-null  object 
 5   diagnosis                      23192 non-null  object 
 6   benign_malignant               23192 non-null  object 
 7   target                         23192 non-null  int64  
dtypes: float64(1), int64(1), object(6)
memory usage: 1.6+ MB
None
-------------
Test Set
```

```
train_df['target'].value_counts(normalize=True).iplot(kind='bar',
                                                       yTitle='Percentage',
                                                       linecolor='black',
                                                       opacity=0.7,
                                                       color='red',
                                                       theme='pearl',
                                                       bargap=0.8,
                                                       gridcolor='white',

                                                       title='Distribution of the Target column in the training set')
```

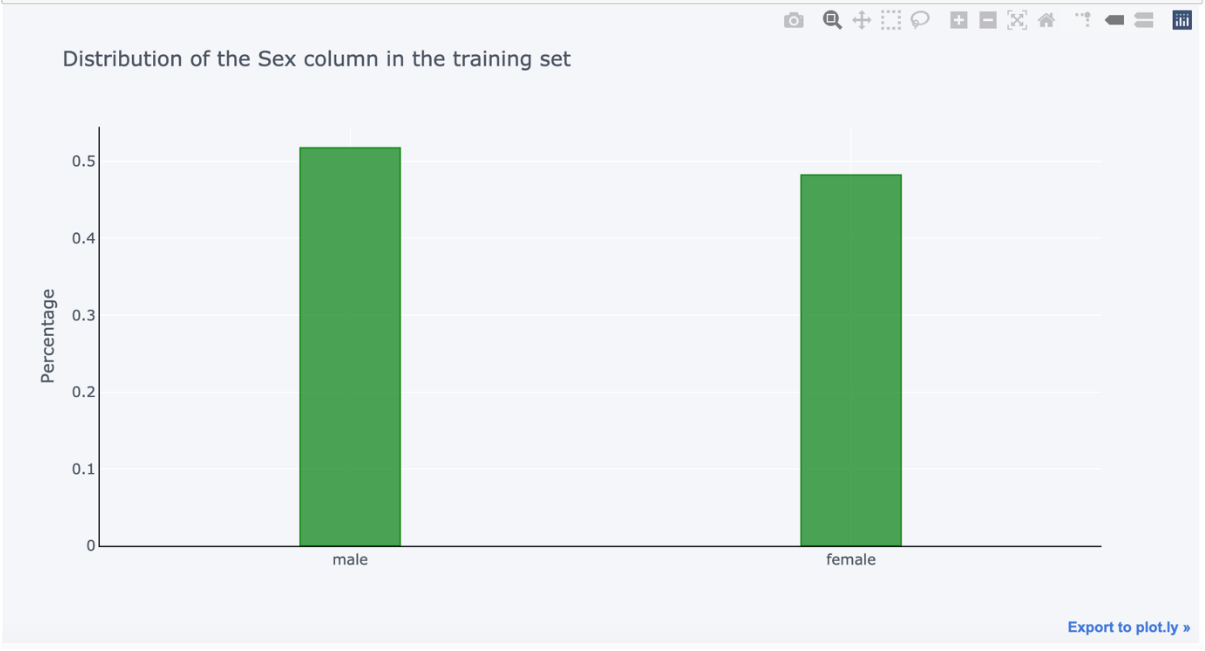


Gender wise distribution

```
train_df['sex'].value_counts(normalize=True)
```

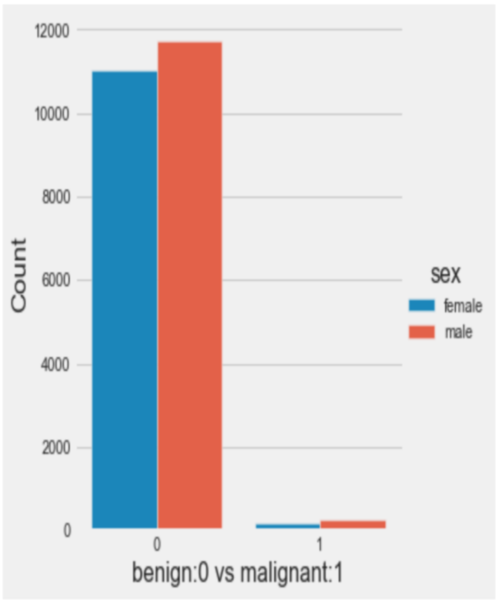
```
male      0.517715
female    0.482285
Name: sex, dtype: float64
```

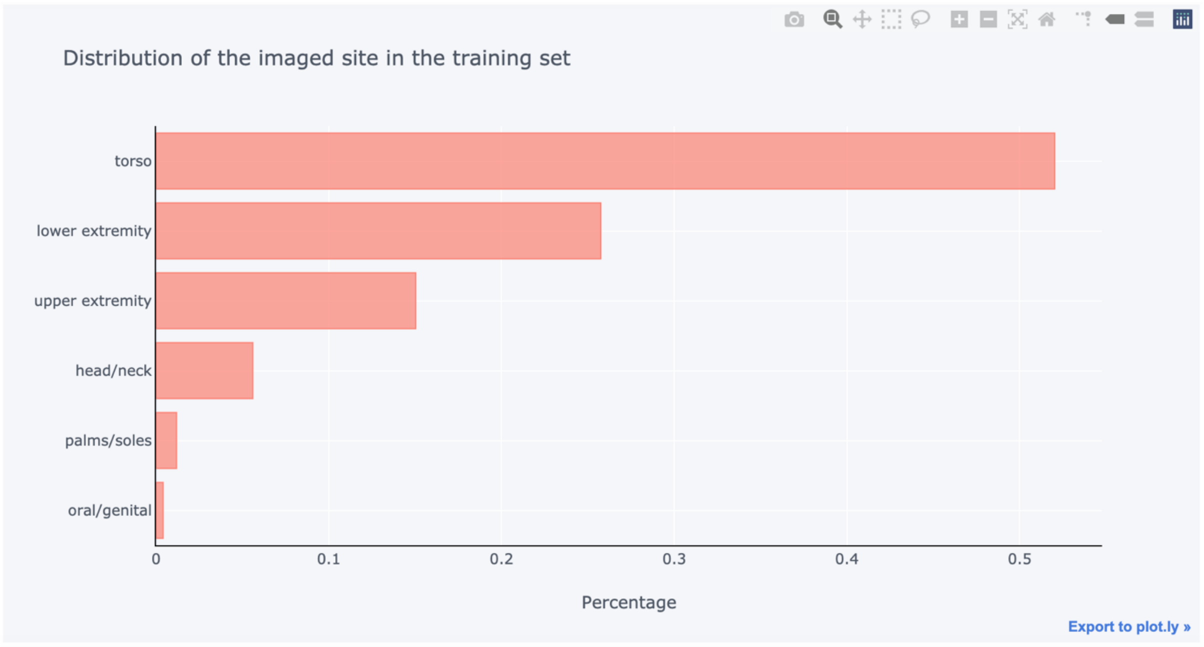
```
train_df['sex'].value_counts(normalize=True).iplot(kind='bar',
                                                      yTitle='Percentage',
                                                      linecolor='black',
                                                      opacity=0.7,
                                                      color='green',
                                                      theme='pearl',
                                                      bargap=0.8,
                                                      gridcolor='white',

                                                      title='Distribution of the Sex column in the training set')
```

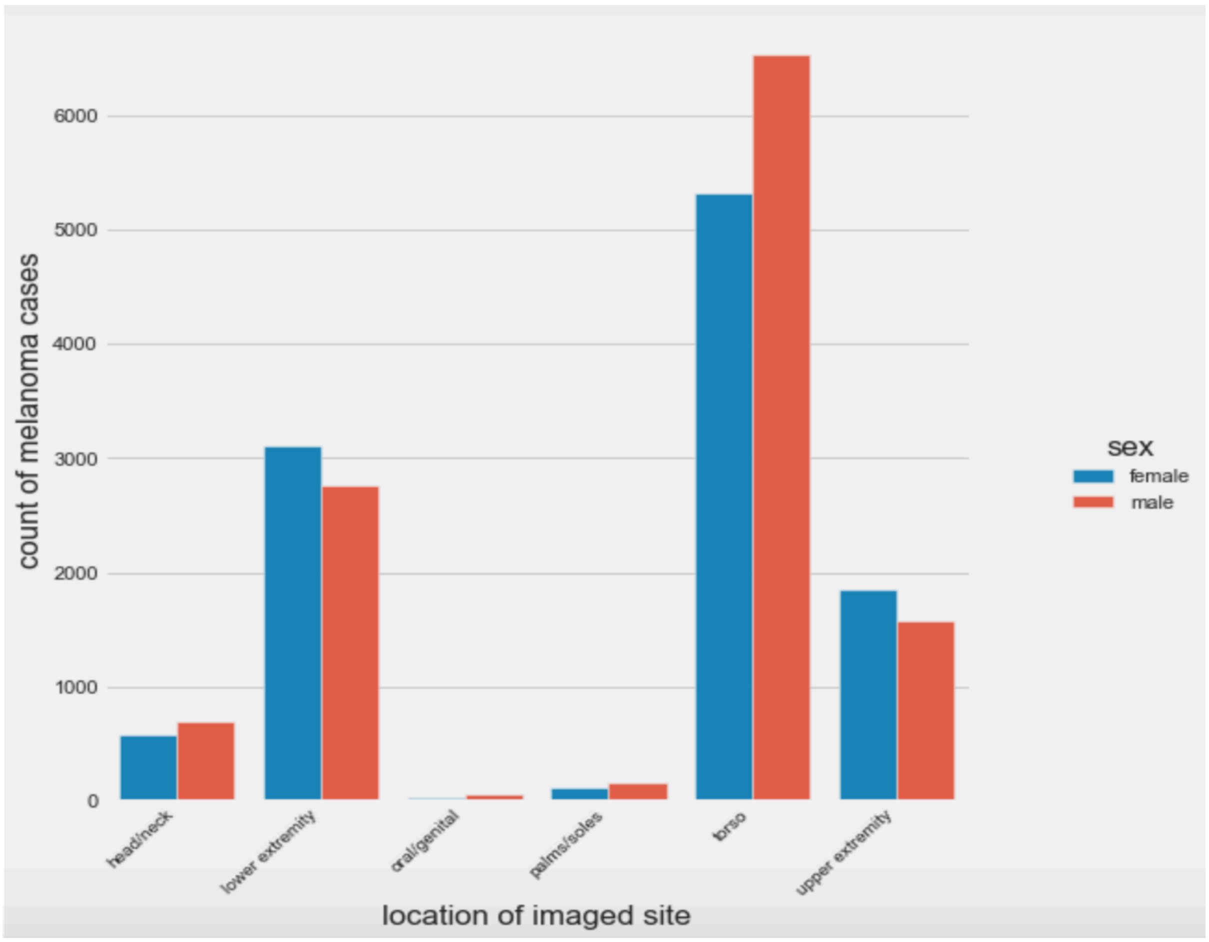


```
sns.catplot(x='target',y='benign_malignant', hue='sex',data=z,kind='bar')
plt.ylabel('Count')
plt.xlabel('benign:0 vs malignant:1')
```

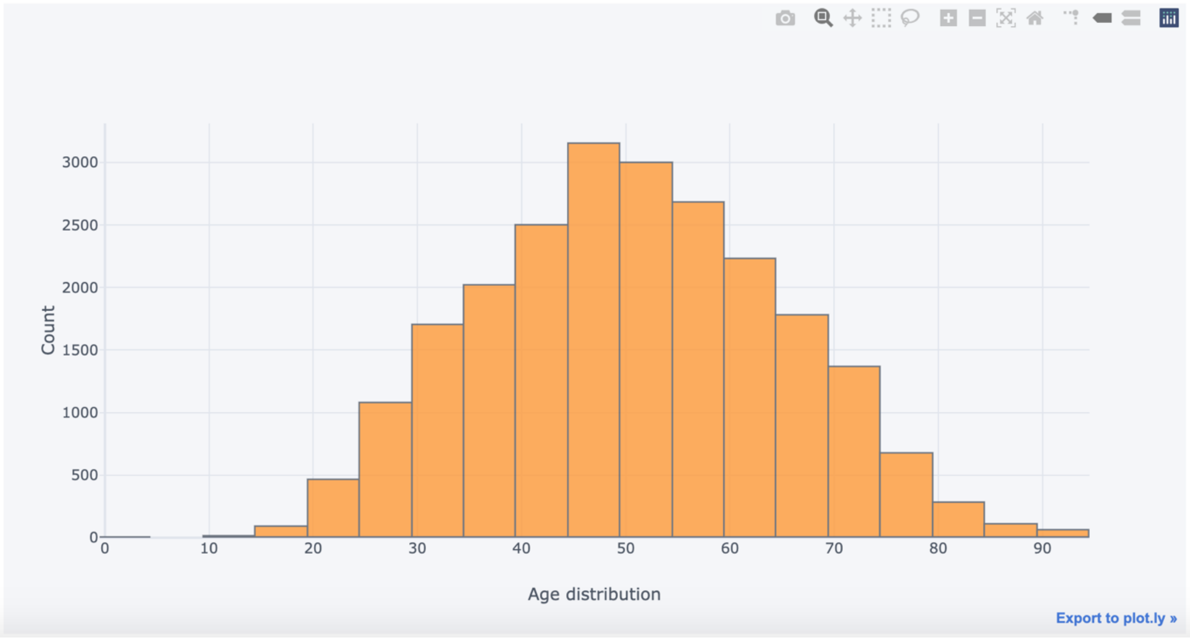
```
Text(0.5, 30.198124999999997, 'benign:0 vs malignant:1')
```







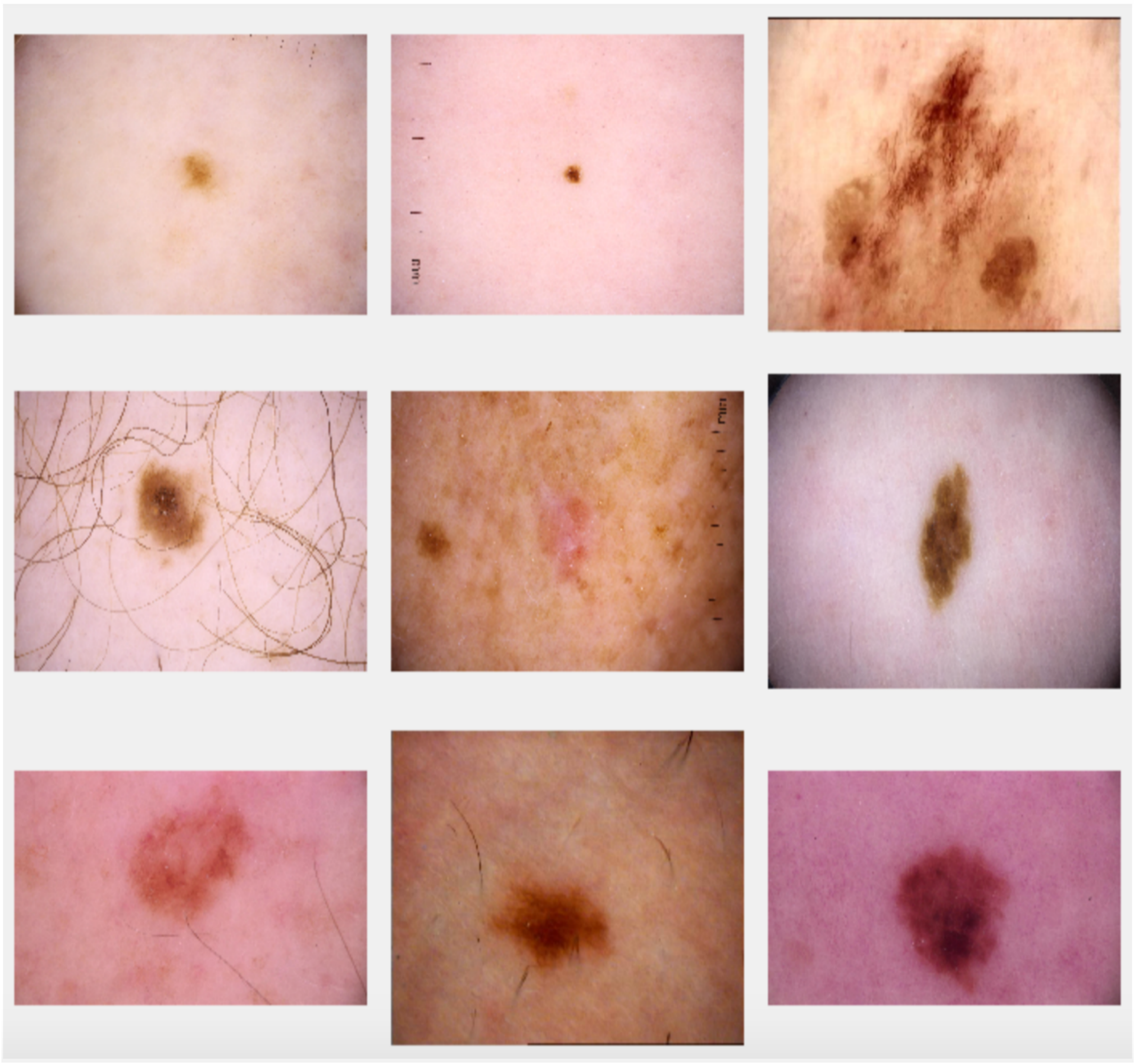
```
train_df['age_approx'].iplot(kind='hist',bins=30,color='orange',xTitle='Age distribution',yTitle='Count')
```



Visualizing Images with benign lesions

```
benign = train_df[train_df['benign_malignant']=='benign']
malignant = train_df[train_df['benign_malignant']=='malignant']
```

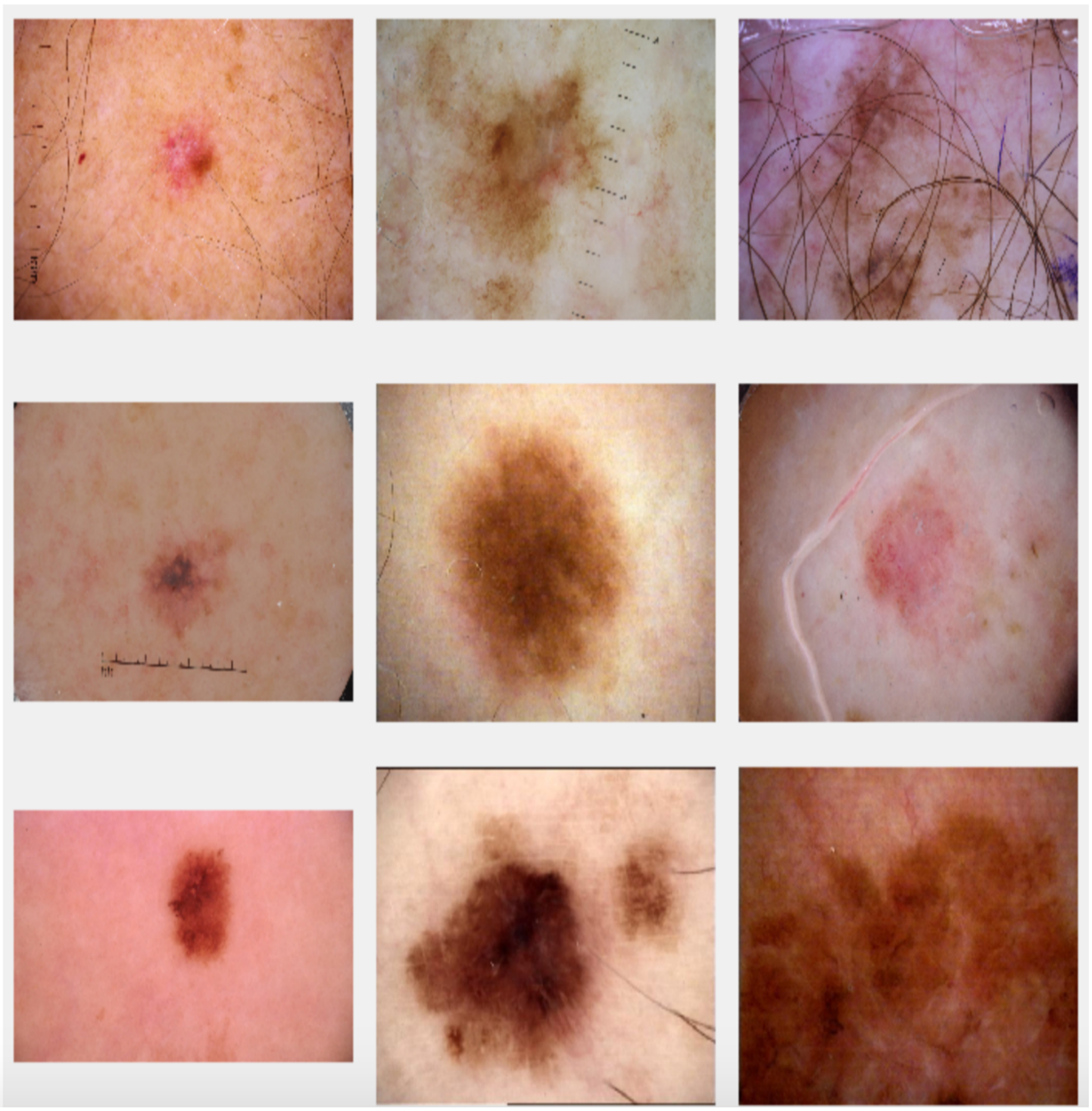
```
images = benign['image_name'].values

# Extract 9 random images from it
random_images = [np.random.choice(images+'.jpg') for i in range(9)]

# Location of the image dir
img_dir = IMAGE_PATH

print('Display benign Images')

# Adjust the size of your images
plt.figure(figsize=(10,8))

# Iterate and plot random images
for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(img_dir, random_images[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

# Adjust subplot parameters to give specified padding
plt.tight_layout()
```



Visualizing Images with Malignant lesions

```python
images = malignant['image_name'].values

# Extract 9 random images from it
random_images = [np.random.choice(images+'.jpg') for i in range(9)]

# Location of the image dir
img_dir = IMAGE_PATH

print('Display malignant Images')

# Adjust the size of your images
plt.figure(figsize=(10,8))

# Iterate and plot random images
for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(img_dir, random_images[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

# Adjust subplot parameters to give specified padding
plt.tight_layout()
```

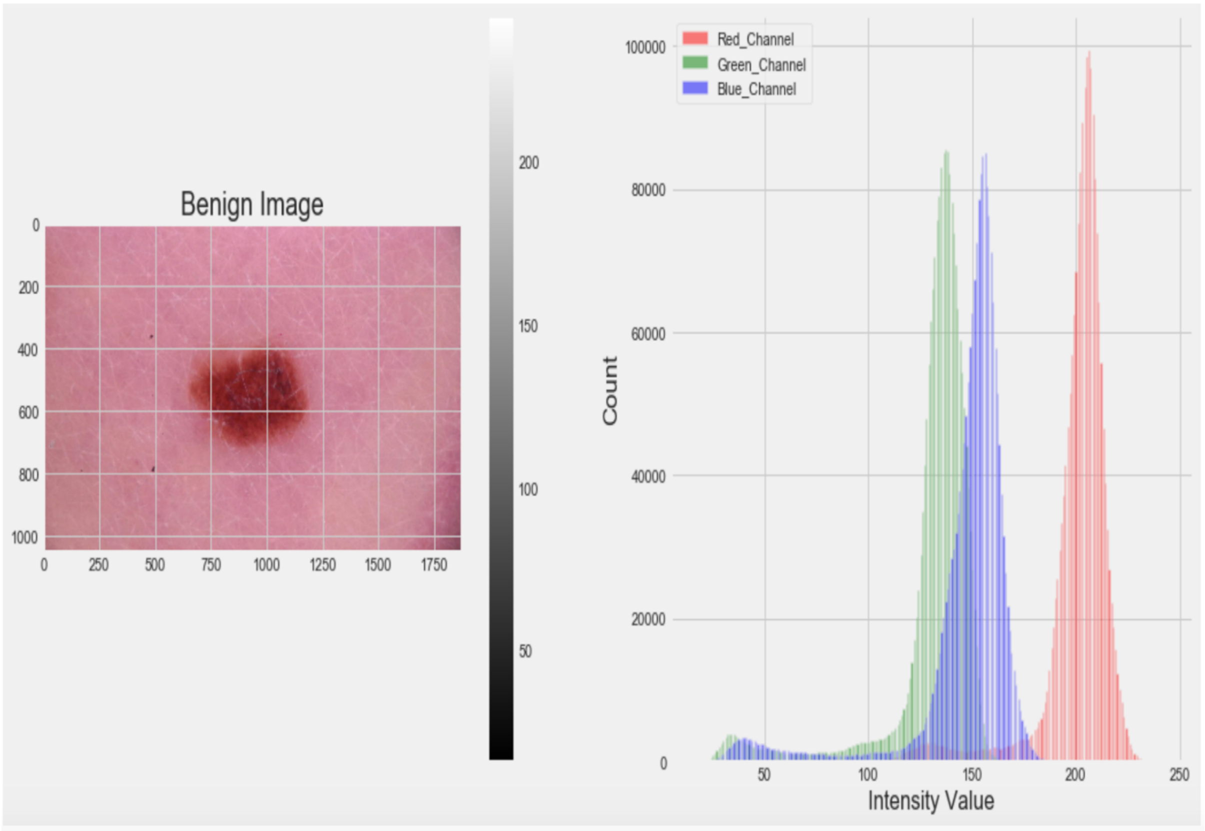
```
Display malignant Images
```



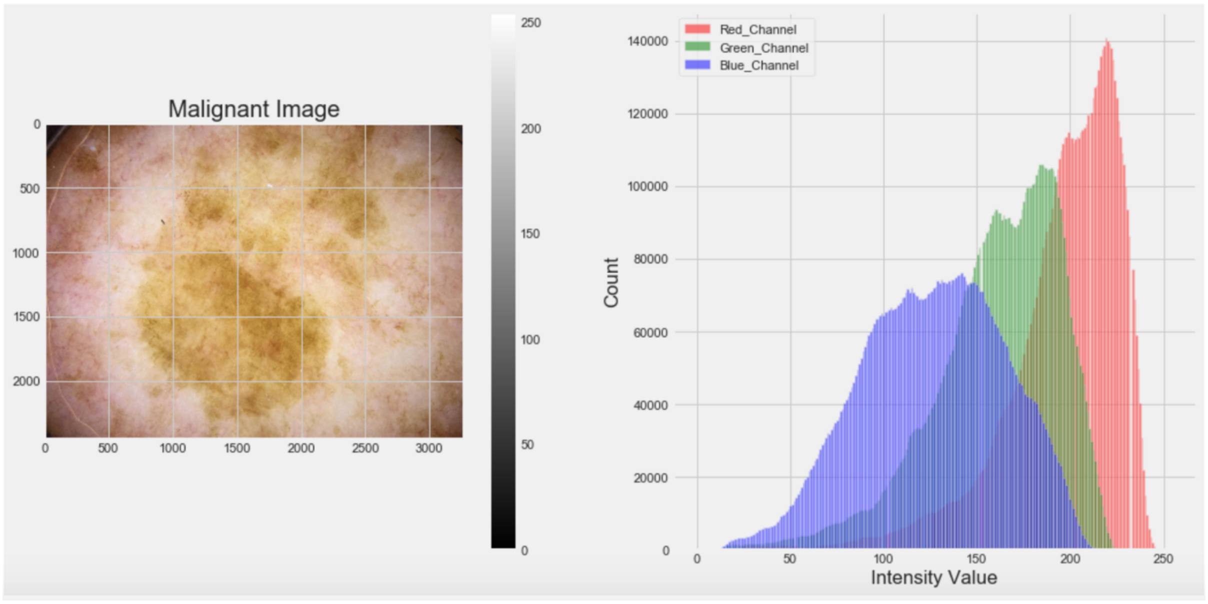
```
## Histograms

# Histograms are a graphical representation showing how frequently various color values occur in the image.
# In a RGB color space, pixel values range from 0 to 255 where 0 stands for black and 255 stands for white.
# Analysis of a histogram can help us understand thee brightness, contrast and intensity distribution of an image.
# Now let''s look at the histogram of a random selected sample from each category.

### Benign category
```

```
f = plt.figure(figsize=(16,8))
f.add_subplot(1,2, 1)

sample_img = benign['image_name'][2]+'.jpg'
raw_image = plt.imread(os.path.join(img_dir, sample_img))
plt.imshow(raw_image, cmap='gray')
plt.colorbar()
plt.title('Benign Image')
print(f"Image dimensions:  {raw_image.shape[0],raw_image.shape[1]}")
print(f"Maximum pixel value : {raw_image.max():.1f} ; Minimum pixel value:{raw_image.min():.1f}")
print(f"Mean value of the pixels : {raw_image.mean():.1f} ; Standard deviation : {raw_image.std():.1f}")

f.add_subplot(1,2, 2)

#_ = plt.hist(raw_image.ravel(),bins = 256, color = 'orange',)
_ = plt.hist(raw_image[:, :, 0].ravel(), bins = 256, color = 'red', alpha = 0.5)
_ = plt.hist(raw_image[:, :, 1].ravel(), bins = 256, color = 'Green', alpha = 0.5)
_ = plt.hist(raw_image[:, :, 2].ravel(), bins = 256, color = 'Blue', alpha = 0.5)
_ = plt.xlabel('Intensity Value')
_ = plt.ylabel('Count')
_ = plt.legend(['Red_Channel', 'Green_Channel', 'Blue_Channel'])
plt.show()
```

```
Image dimensions:  (1053, 1872)
Maximum pixel value : 245.0 ; Minimum pixel value:16.0
Mean value of the pixels : 158.7 ; Standard deviation : 37.4
```



```
Image dimensions:  (2448, 3264)
Maximum pixel value : 255.0 ; Minimum pixel value:0.0
Mean value of the pixels : 161.0 ; Standard deviation : 45.1
```

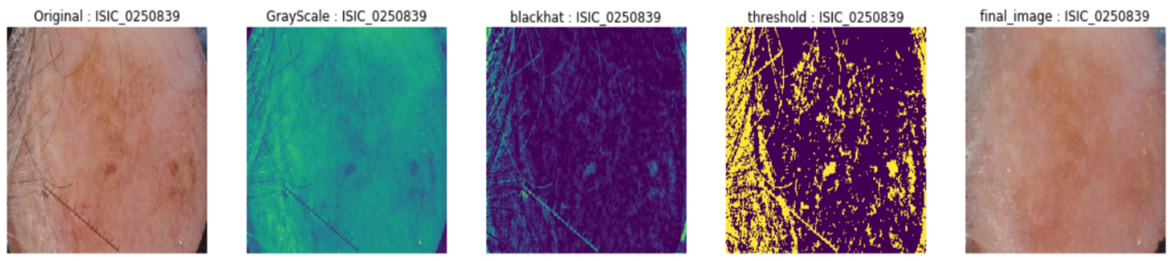


4. Image Pre-processing (Image Preprocessing and Downsampling.ipynb)

Various filters were applied on the downsampled data to remove the noise and unwanted artifacts from the images. Below is the code for the same. Grayscale filter, Blackhat filter, Hair inpainting filter and threshold filter was used for transformation and images were resized to 256x256.

```
#Making directories to store the data after image processing

os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed')
os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed/train')
os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed/test')
os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed/train/0')
os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed/train/1')
os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed/test/1/')
os.mkdir('/Users/vishalgoyal/Downloads/ImageProcessed/test/0/')
```

```
#Function For processing the images
list_of_filename = []
def filters(source_path,destination_path):
    for root, dirs, files in os.walk(source_path):
        for filename in files:
            list_of_filename.append(filename)

    l = len(list_of_filename)
    for i,image_name in enumerate(list_of_filename):
        image = cv2.imread(source_path+image_name)
        image_resize = cv2.resize(image,(256,256))
        grayScale = cv2.cvtColor(image_resize, cv2.COLOR_RGB2GRAY)
        kernel = cv2.getStructuringElement(1,(17,17))
        # Perform the blackHat filtering on the grayscale image to find the hair countours
        blackhat = cv2.morphologyEx(grayScale, cv2.MORPH_BLACKHAT, kernel)
        # intensify the hair countours in preparation for the inpainting
        ret,threshold = cv2.threshold(blackhat,10,255,cv2.THRESH_BINARY)
        # inpaint the original image depending on the mask
        final_image = cv2.inpaint(image_resize,threshold,1,cv2.INPAINT_TELEA)
        cv2.imwrite(destination_path+image_name,final_image)
```



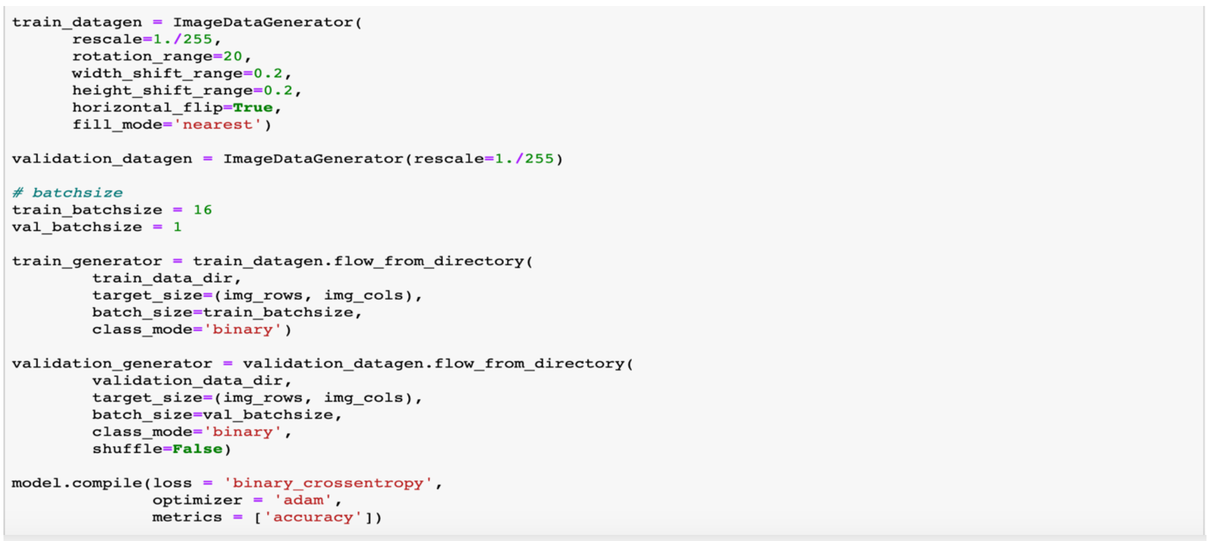
```
#Train Category 0 preprocessing
destination_path = '/Users/vishalgoyal/Downloads/ImageProcessed/train/0/'
source_path = '/Users/vishalgoyal/Downloads/TrainImage/0/'
filters(source_path,destination_path)
list_of_filename.clear()

#Train Category 1 preprocessing
destination_path = '/Users/vishalgoyal/Downloads/ImageProcessed/train/1/'
source_path = '/Users/vishalgoyal/Downloads/TrainImage/1/'
filters(source_path,destination_path)
list_of_filename.clear()
```

```
#Test Category 0 preprocessing
destination_path = '/Users/vishalgoyal/Downloads/ImageProcessed/test/0/'
source_path = '/Users/vishalgoyal/Downloads/TestImage/0/'
filters(source_path,destination_path)
list_of_filename.clear()

#Test Category 1 preprocessing
destination_path = '/Users/vishalgoyal/Downloads/ImageProcessed/test/1/'
source_path = '/Users/vishalgoyal/Downloads/TestImage/1/'
filters(source_path,destination_path)
list_of_filename.clear()
```

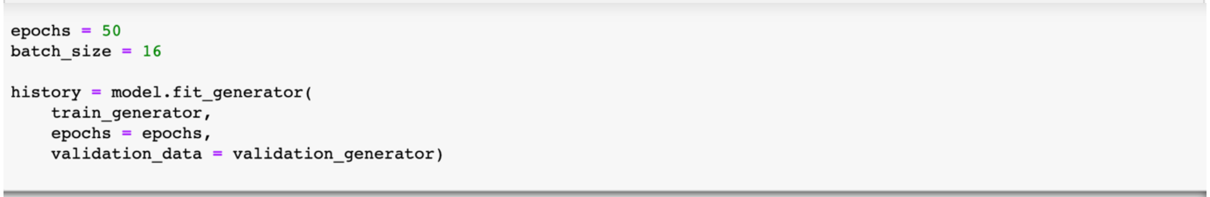
5. Model Implementation

MobileNet Implementation

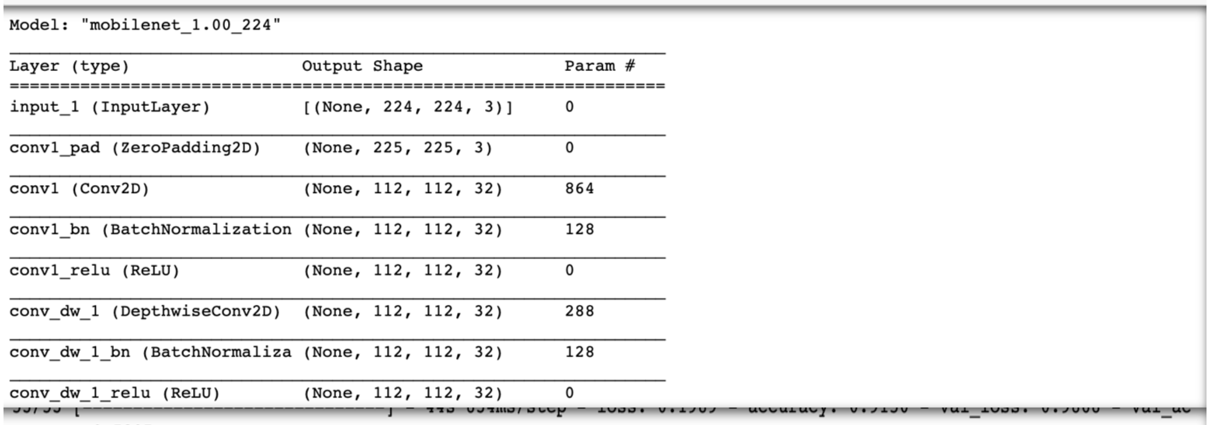
```python
#Importing libraries
import pandas as pd
import numpy as np
import shutil
import os, sys
import cv2
import matplotlib.pyplot as plt
from sklearn.utils import resample
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.models import Model
#MobileNet Model
from keras.applications import MobileNet

# # MobileNet was designed to work on 224 x 224 pixel input images sizes
img_rows, img_cols = 224, 224

# Re-loads the MobileNet model without the top or FC layers
MobileNet = MobileNet(weights = 'imagenet',
                 include_top = False,
                 input_shape = (224, 224, 3))

MobileNet.summary()
# Here we freeze the last 4 layers
# Layers are set to trainable as True by default
for layer in MobileNet.layers:
    layer.trainable = False

# Let's print our layers
for (i,layer) in enumerate(MobileNet.layers):
    print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
# Let's make a function that returns our FC Head
def addTopModelMobileNet(bottom_model, num_classes):
    """creates the top or head of the model that will be
    placed ontop of the bottom layers"""

    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(1,activation='sigmoid')(top_model)
    return top_model

num_classes = 2

FC_Head = addTopModelMobileNet(MobileNet, num_classes)

model = Model(inputs = MobileNet.input, outputs = FC_Head)

print(model.summary())


#Loading our Cancer Dataset

from keras.preprocessing.image import ImageDataGenerator

train_data_dir =  '/Users/vishalgoyal/Downloads/ImageProcessed/train/'
validation_data_dir ='/Users/vishalgoyal/Downloads/ImageProcessed/test/'
```



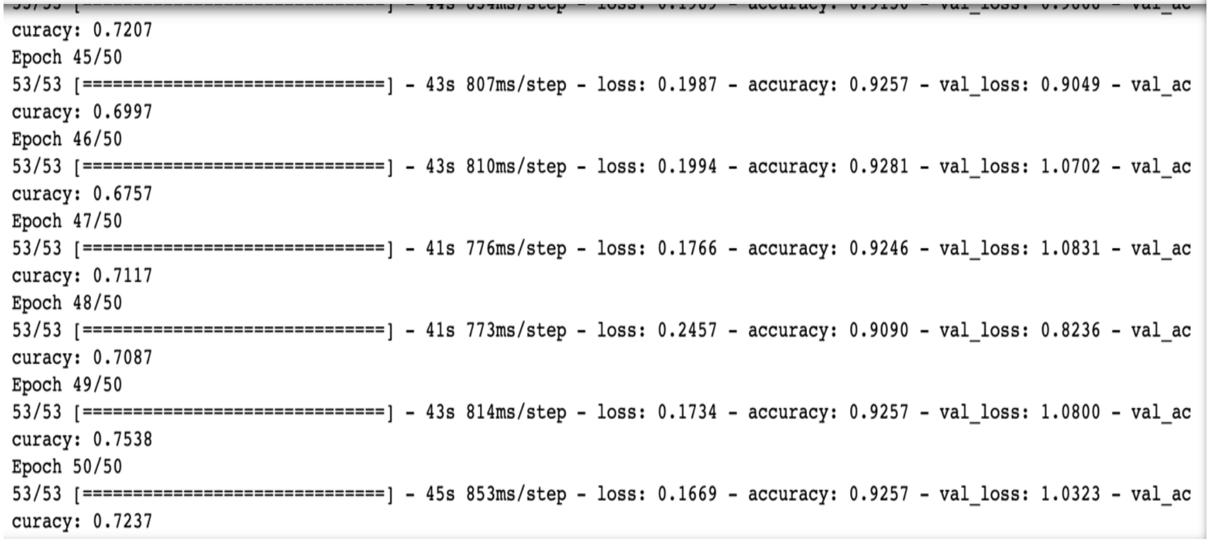
```
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# batchsize
train_batchsize = 16
val_batchsize = 1

train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=train_batchsize,
        class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=val_batchsize,
        class_mode='binary',
        shuffle=False)

model.compile(loss = 'binary_crossentropy',
              optimizer = 'adam',
              metrics = ['accuracy'])
```



```
epochs = 50
batch_size = 16

history = model.fit_generator(
    train_generator,
    epochs = epochs,
    validation_data = validation_generator)
```

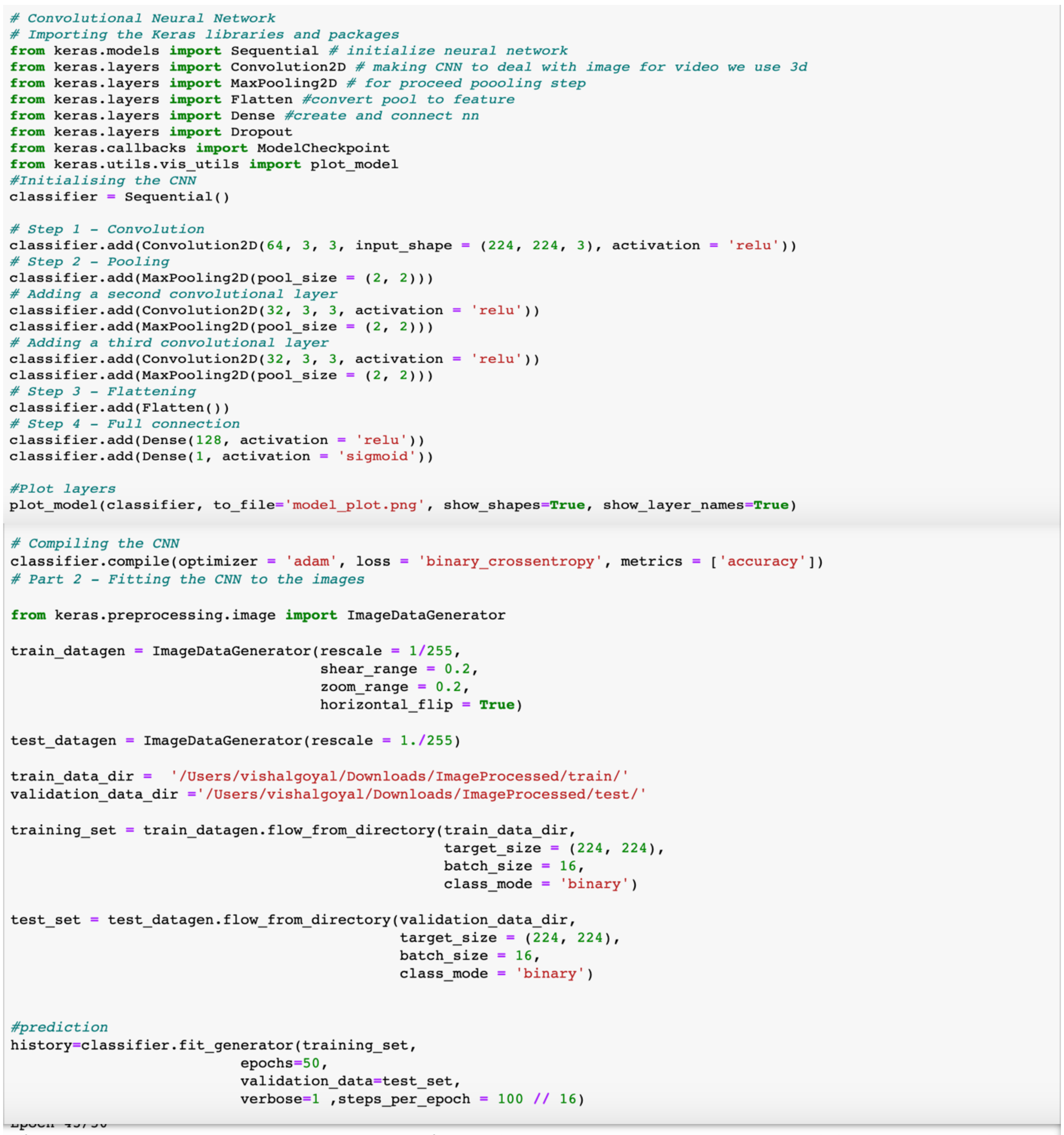


```
Model: "mobilenet_1.00_224"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv1_pad (ZeroPadding2D)    (None, 225, 225, 3)       0
_________________________________________________________________
conv1 (Conv2D)               (None, 112, 112, 32)      864
_________________________________________________________________
conv1_bn (BatchNormalization (None, 112, 112, 32)      128
_________________________________________________________________
conv1_relu (ReLU)            (None, 112, 112, 32)      0
_________________________________________________________________
conv_dw_1 (DepthwiseConv2D)  (None, 112, 112, 32)      288
_________________________________________________________________
conv_dw_1_bn (BatchNormaliza (None, 112, 112, 32)      128
_________________________________________________________________
conv_dw_1_relu (ReLU)        (None, 112, 112, 32)      0
```



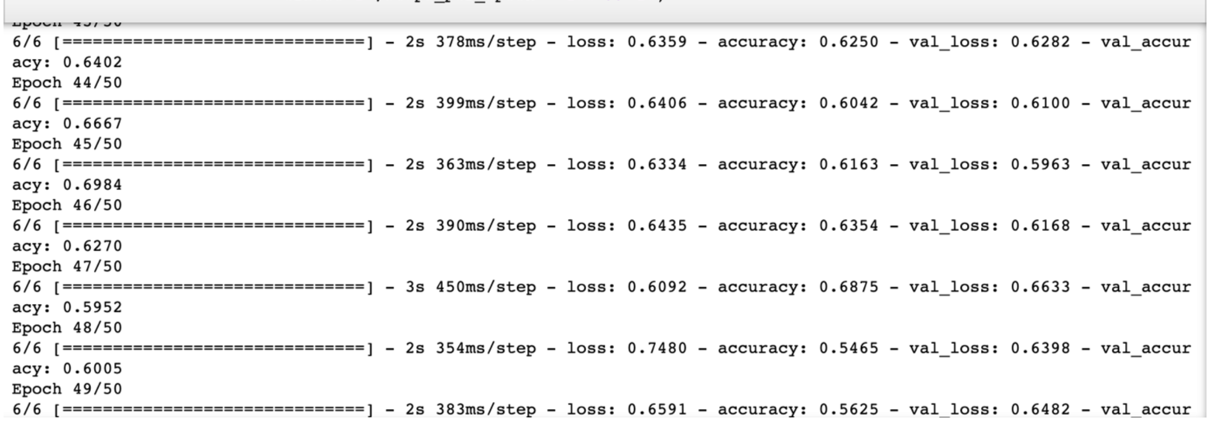
```
curacy: 0.7207
Epoch 45/50
53/53 [==============================] - 43s 807ms/step - loss: 0.1987 - accuracy: 0.9257 - val_loss: 0.9049 - val_ac
curacy: 0.6997
Epoch 46/50
53/53 [==============================] - 43s 810ms/step - loss: 0.1994 - accuracy: 0.9281 - val_loss: 1.0702 - val_ac
curacy: 0.6757
Epoch 47/50
53/53 [==============================] - 41s 776ms/step - loss: 0.1766 - accuracy: 0.9246 - val_loss: 1.0831 - val_ac
curacy: 0.7117
Epoch 48/50
53/53 [==============================] - 41s 773ms/step - loss: 0.2457 - accuracy: 0.9090 - val_loss: 0.8236 - val_ac
curacy: 0.7087
Epoch 49/50
53/53 [==============================] - 43s 814ms/step - loss: 0.1734 - accuracy: 0.9257 - val_loss: 1.0800 - val_ac
curacy: 0.7538
Epoch 50/50
53/53 [==============================] - 45s 853ms/step - loss: 0.1669 - accuracy: 0.9257 - val_loss: 1.0323 - val_ac
curacy: 0.7237
```

CNN implementation



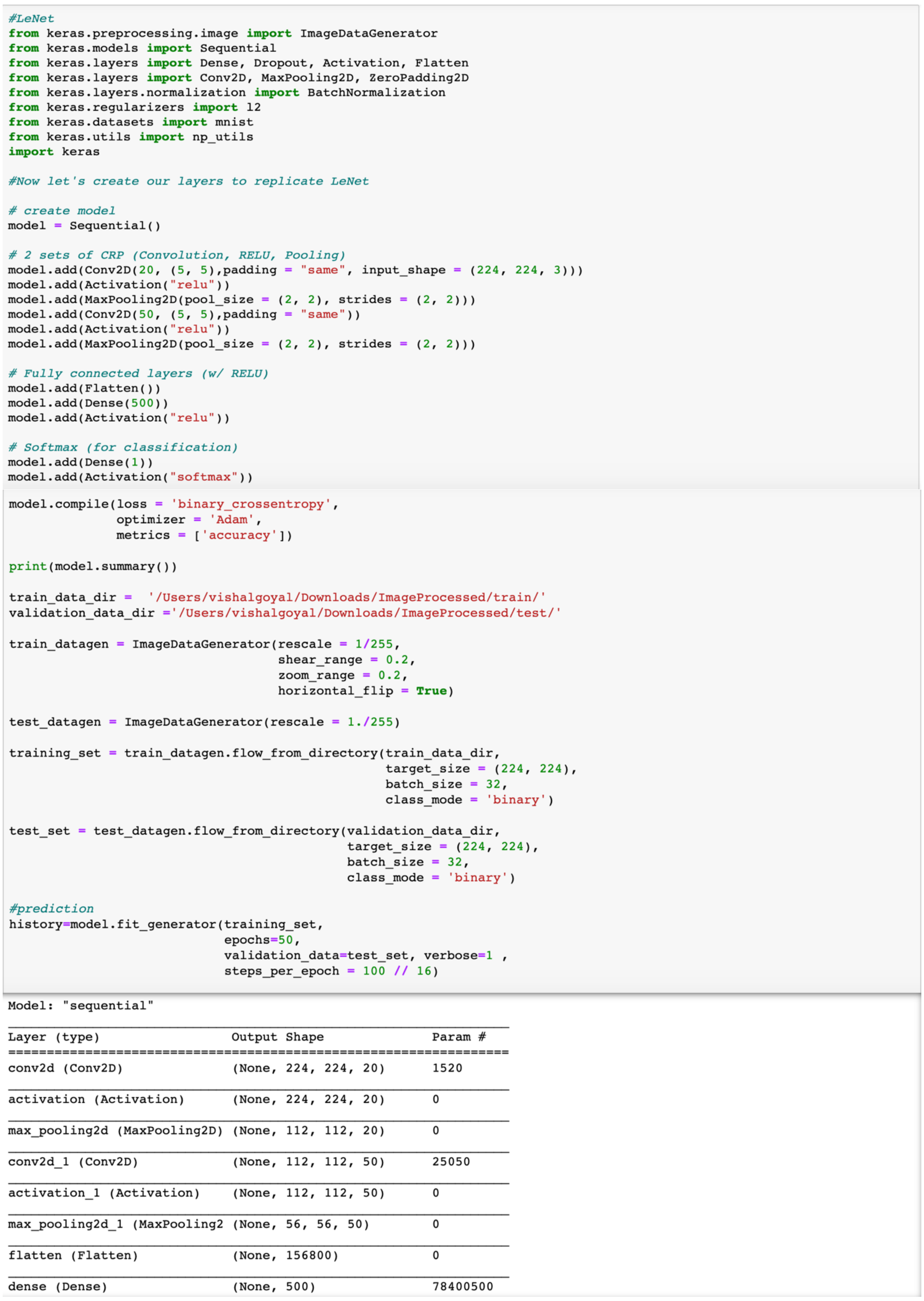
```
# Convolutional Neural Network
# Importing the Keras libraries and packages
from keras.models import Sequential # initialize neural network
from keras.layers import Convolution2D # making CNN to deal with image for video we use 3d
from keras.layers import MaxPooling2D # for proceed poooling step
from keras.layers import Flatten #convert pool to feature
from keras.layers import Dense #create and connect nn
from keras.layers import Dropout
from keras.callbacks import ModelCheckpoint
from keras.utils.vis_utils import plot_model
#Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Convolution2D(64, 3, 3, input_shape = (224, 224, 3), activation = 'relu'))
# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a second convolutional layer
classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a third convolutional layer
classifier.add(Convolution2D(32, 3, 3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Step 3 - Flattening
classifier.add(Flatten())
# Step 4 - Full connection
classifier.add(Dense(128, activation = 'relu'))
classifier.add(Dense(1, activation = 'sigmoid'))

#Plot layers
plot_model(classifier, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

```
# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1/255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

train_data_dir =  '/Users/vishalgoyal/Downloads/ImageProcessed/train/'
validation_data_dir ='/Users/vishalgoyal/Downloads/ImageProcessed/test/'

training_set = train_datagen.flow_from_directory(train_data_dir,
                                                 target_size = (224, 224),
                                                 batch_size = 16,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory(validation_data_dir,
                                            target_size = (224, 224),
                                            batch_size = 16,
                                            class_mode = 'binary')


#prediction
history=classifier.fit_generator(training_set,
                         epochs=50,
                         validation_data=test_set,
                         verbose=1 ,steps_per_epoch = 100 // 16)
```



```
6/6 [==============================] - 2s 378ms/step - loss: 0.6359 - accuracy: 0.6250 - val_loss: 0.6282 - val_accur
acy: 0.6402
Epoch 44/50
6/6 [==============================] - 2s 399ms/step - loss: 0.6406 - accuracy: 0.6042 - val_loss: 0.6100 - val_accur
acy: 0.6667
Epoch 45/50
6/6 [==============================] - 2s 363ms/step - loss: 0.6334 - accuracy: 0.6163 - val_loss: 0.5963 - val_accur
acy: 0.6984
Epoch 46/50
6/6 [==============================] - 2s 390ms/step - loss: 0.6435 - accuracy: 0.6354 - val_loss: 0.6168 - val_accur
acy: 0.6270
Epoch 47/50
6/6 [==============================] - 3s 450ms/step - loss: 0.6092 - accuracy: 0.6875 - val_loss: 0.6633 - val_accur
acy: 0.5952
Epoch 48/50
6/6 [==============================] - 2s 354ms/step - loss: 0.7480 - accuracy: 0.5465 - val_loss: 0.6398 - val_accur
acy: 0.6005
Epoch 49/50
6/6 [==============================] - 2s 383ms/step - loss: 0.6591 - accuracy: 0.5625 - val_loss: 0.6482 - val_accur
```

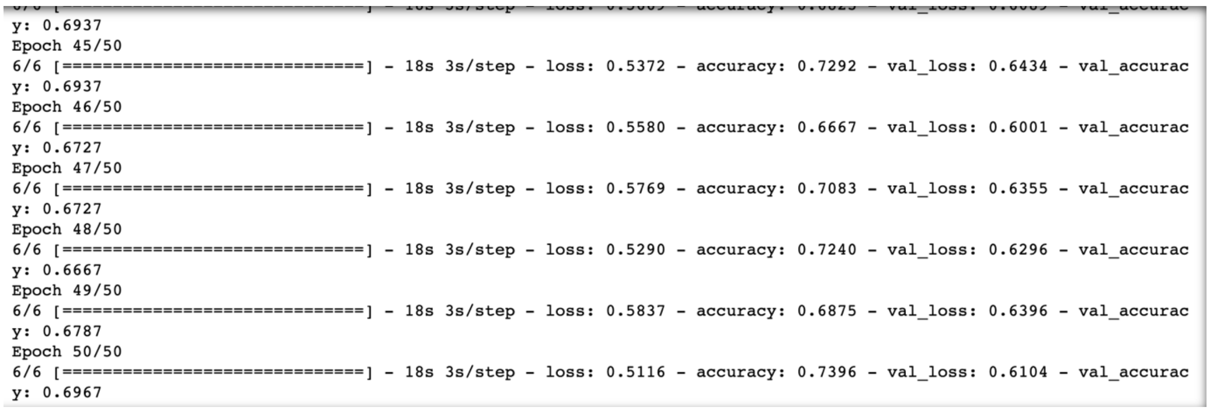
LeNet implementation



```python
#LeNet
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.regularizers import l2
from keras.datasets import mnist
from keras.utils import np_utils
import keras

#Now let's create our layers to replicate LeNet

# create model
model = Sequential()

# 2 sets of CRP (Convolution, RELU, Pooling)
model.add(Conv2D(20, (5, 5),padding = "same", input_shape = (224, 224, 3)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))
model.add(Conv2D(50, (5, 5),padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))

# Fully connected layers (w/ RELU)
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))

# Softmax (for classification)
model.add(Dense(1))
model.add(Activation("softmax"))
```

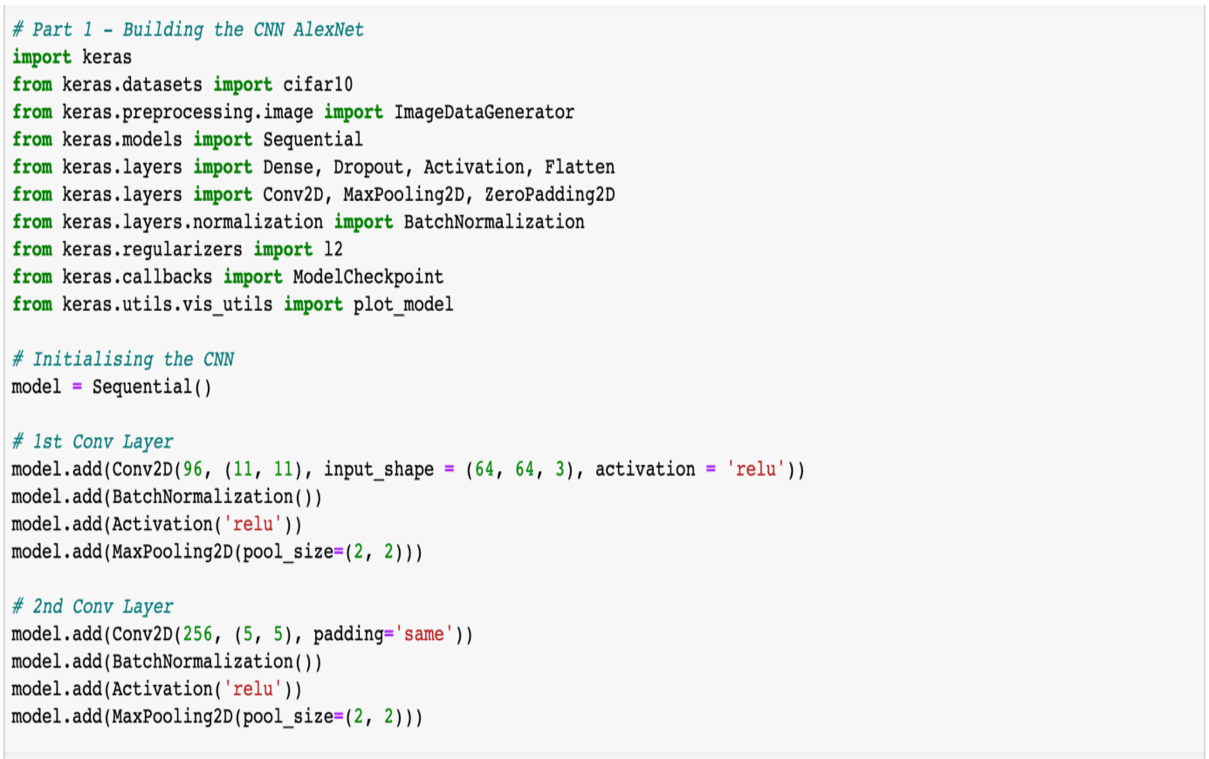
```python
model.compile(loss = 'binary_crossentropy',
              optimizer = 'Adam',
              metrics = ['accuracy'])

print(model.summary())

train_data_dir =  '/Users/vishalgoyal/Downloads/ImageProcessed/train/'
validation_data_dir ='/Users/vishalgoyal/Downloads/ImageProcessed/test/'

train_datagen = ImageDataGenerator(rescale = 1/255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory(train_data_dir,
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory(validation_data_dir,
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'binary')

#prediction
history=model.fit_generator(training_set,
                            epochs=50,
                            validation_data=test_set, verbose=1 ,
                            steps_per_epoch = 100 // 16)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 224, 224, 20)      1520      
_________________________________________________________________
activation (Activation)      (None, 224, 224, 20)      0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 20)      0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 50)      25050     
_________________________________________________________________
activation_1 (Activation)    (None, 112, 112, 50)      0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 50)        0         
_________________________________________________________________
flatten (Flatten)            (None, 156800)            0         
_________________________________________________________________
dense (Dense)                (None, 500)               78400500  
```

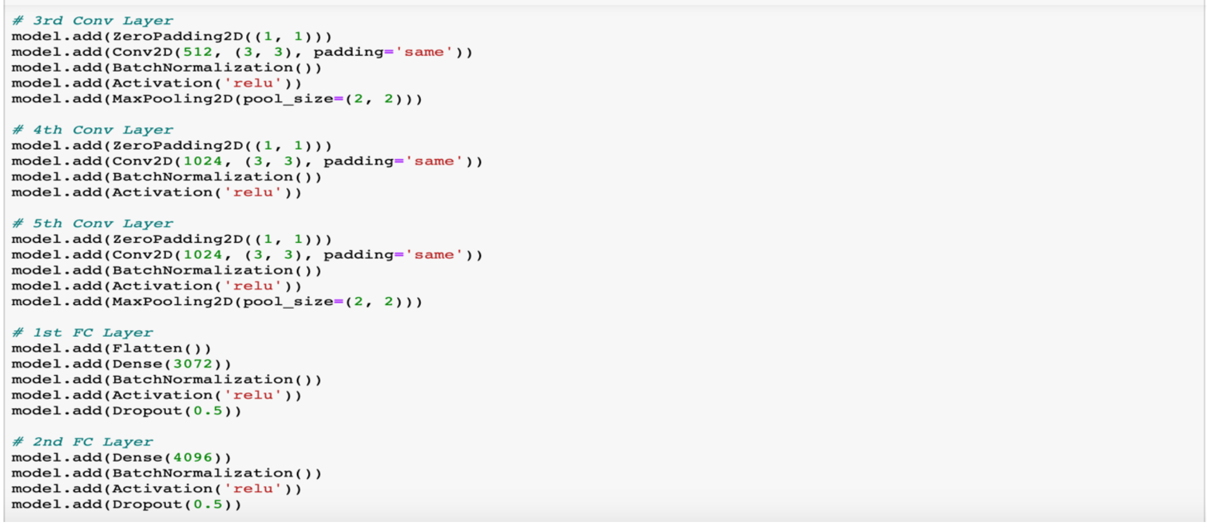


```
y: 0.6937
Epoch 45/50
6/6 [==============================] - 18s 3s/step - loss: 0.5372 - accuracy: 0.7292 - val_loss: 0.6434 - val_accurac
y: 0.6937
Epoch 46/50
6/6 [==============================] - 18s 3s/step - loss: 0.5580 - accuracy: 0.6667 - val_loss: 0.6001 - val_accurac
y: 0.6727
Epoch 47/50
6/6 [==============================] - 18s 3s/step - loss: 0.5769 - accuracy: 0.7083 - val_loss: 0.6355 - val_accurac
y: 0.6727
Epoch 48/50
6/6 [==============================] - 18s 3s/step - loss: 0.5290 - accuracy: 0.7240 - val_loss: 0.6296 - val_accurac
y: 0.6667
Epoch 49/50
6/6 [==============================] - 18s 3s/step - loss: 0.5837 - accuracy: 0.6875 - val_loss: 0.6396 - val_accurac
y: 0.6787
Epoch 50/50
6/6 [==============================] - 18s 3s/step - loss: 0.5116 - accuracy: 0.7396 - val_loss: 0.6104 - val_accurac
y: 0.6967
```

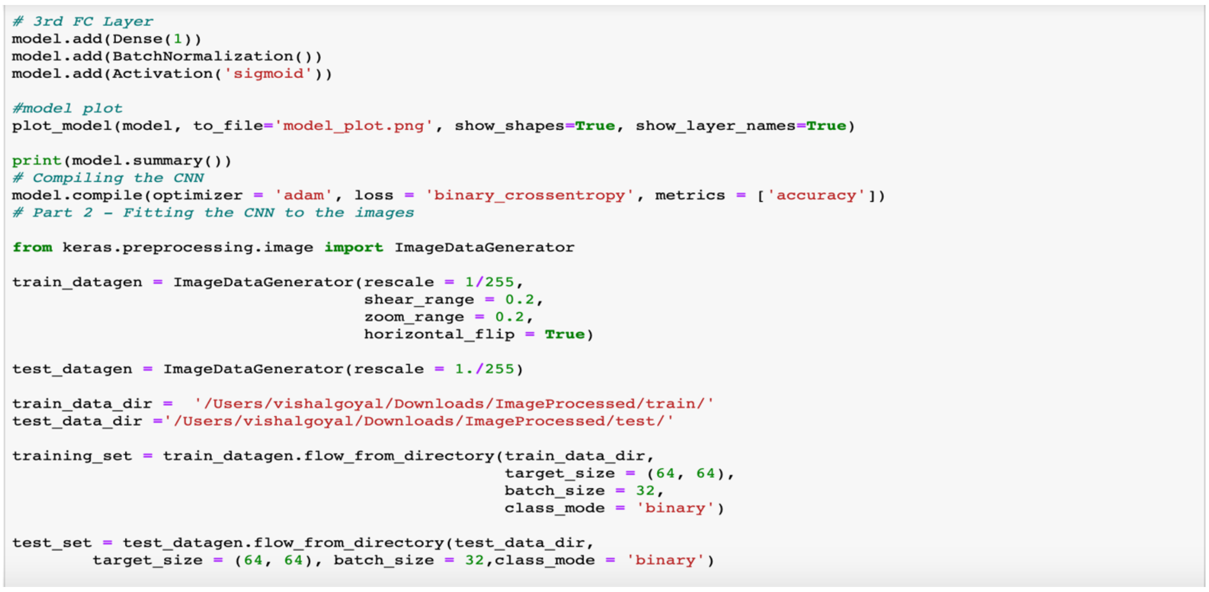
AlexNet Implementation



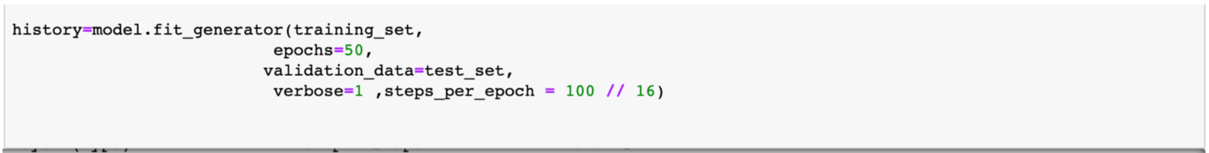
```python
# Part 1 - Building the CNN AlexNet
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.regularizers import l2
from keras.callbacks import ModelCheckpoint
from keras.utils.vis_utils import plot_model

# Initialising the CNN
model = Sequential()

# 1st Conv Layer
model.add(Conv2D(96, (11, 11), input_shape = (64, 64, 3), activation = 'relu'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 2nd Conv Layer
model.add(Conv2D(256, (5, 5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```



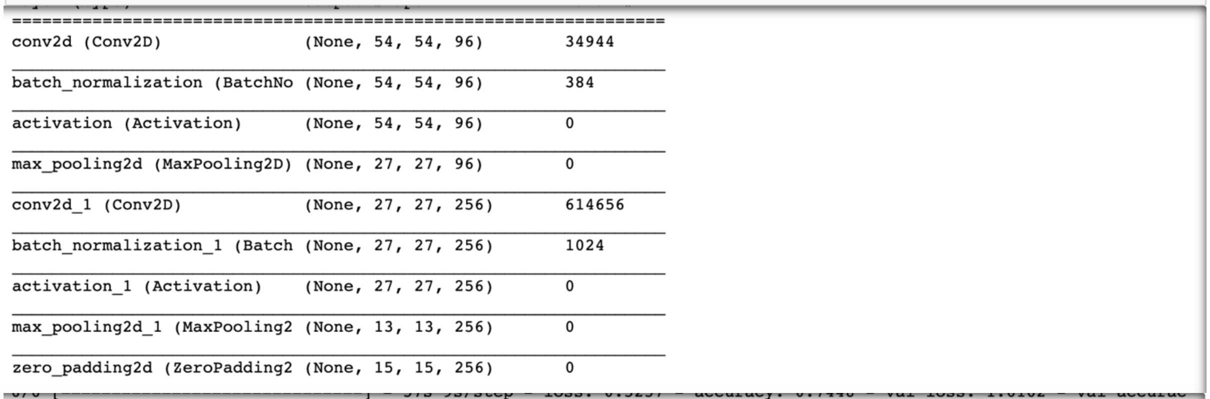
```python
# 3rd Conv Layer
model.add(ZeroPadding2D((1, 1)))
model.add(Conv2D(512, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 4th Conv Layer
model.add(ZeroPadding2D((1, 1)))
model.add(Conv2D(1024, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))

# 5th Conv Layer
model.add(ZeroPadding2D((1, 1)))
model.add(Conv2D(1024, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 1st FC Layer
model.add(Flatten())
model.add(Dense(3072))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))

# 2nd FC Layer
model.add(Dense(4096))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
```



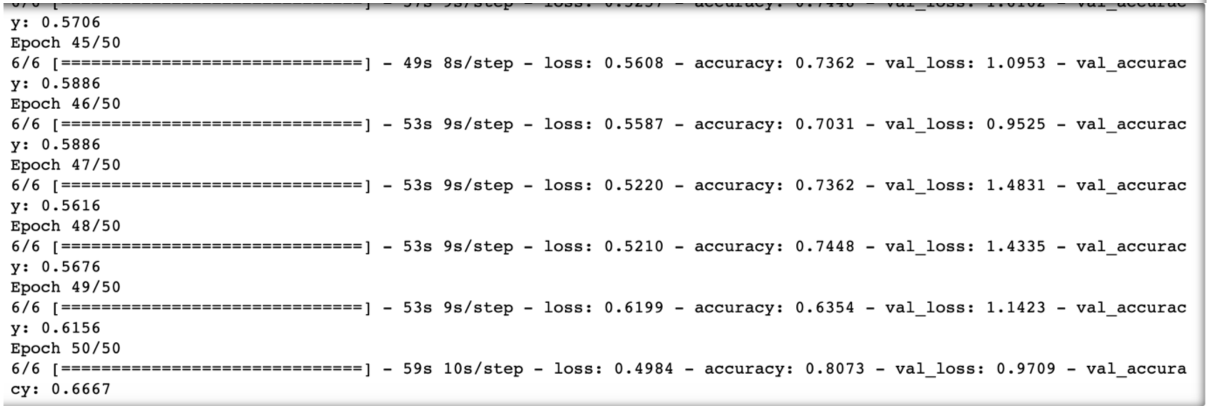
```
# 3rd FC Layer
model.add(Dense(1))
model.add(BatchNormalization())
model.add(Activation('sigmoid'))

#model plot
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

print(model.summary())
# Compiling the CNN
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1/255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

train_data_dir =  '/Users/vishalgoyal/Downloads/ImageProcessed/train/'
test_data_dir ='/Users/vishalgoyal/Downloads/ImageProcessed/test/'

training_set = train_datagen.flow_from_directory(train_data_dir,
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')

test_set = test_datagen.flow_from_directory(test_data_dir,
        target_size = (64, 64), batch_size = 32,class_mode = 'binary')
```



```
history=model.fit_generator(training_set,
                          epochs=50,
                         validation_data=test_set,
                          verbose=1 ,steps_per_epoch = 100 // 16)
```



```
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944     
_________________________________________________________________
batch_normalization (BatchNo (None, 54, 54, 96)        384       
_________________________________________________________________
activation (Activation)      (None, 54, 54, 96)        0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 256)       614656    
_________________________________________________________________
batch_normalization_1 (Batch (None, 27, 27, 256)       1024      
_________________________________________________________________
activation_1 (Activation)    (None, 27, 27, 256)       0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256)       0         
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 15, 15, 256)       0         
```



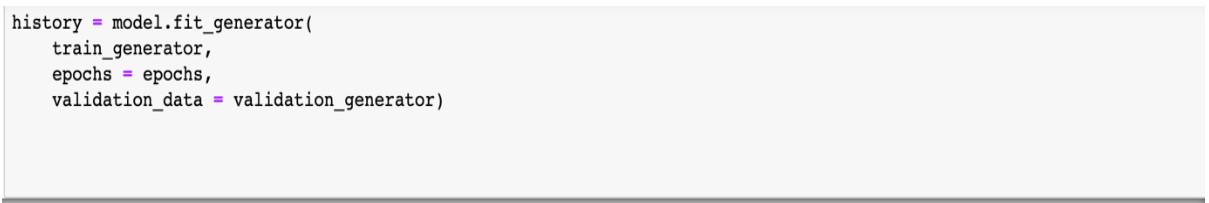
```
y: 0.5706
Epoch 45/50
6/6 [==============================] - 49s 8s/step - loss: 0.5608 - accuracy: 0.7362 - val_loss: 1.0953 - val_accurac
y: 0.5886
Epoch 46/50
6/6 [==============================] - 53s 9s/step - loss: 0.5587 - accuracy: 0.7031 - val_loss: 0.9525 - val_accurac
y: 0.5886
Epoch 47/50
6/6 [==============================] - 53s 9s/step - loss: 0.5220 - accuracy: 0.7362 - val_loss: 1.4831 - val_accurac
y: 0.5616
Epoch 48/50
6/6 [==============================] - 53s 9s/step - loss: 0.5210 - accuracy: 0.7448 - val_loss: 1.4335 - val_accurac
y: 0.5676
Epoch 49/50
6/6 [==============================] - 53s 9s/step - loss: 0.6199 - accuracy: 0.6354 - val_loss: 1.1423 - val_accurac
y: 0.6156
Epoch 50/50
6/6 [==============================] - 59s 10s/step - loss: 0.4984 - accuracy: 0.8073 - val_loss: 0.9709 - val_accura
cy: 0.6667
```

ResNet50 Implementation

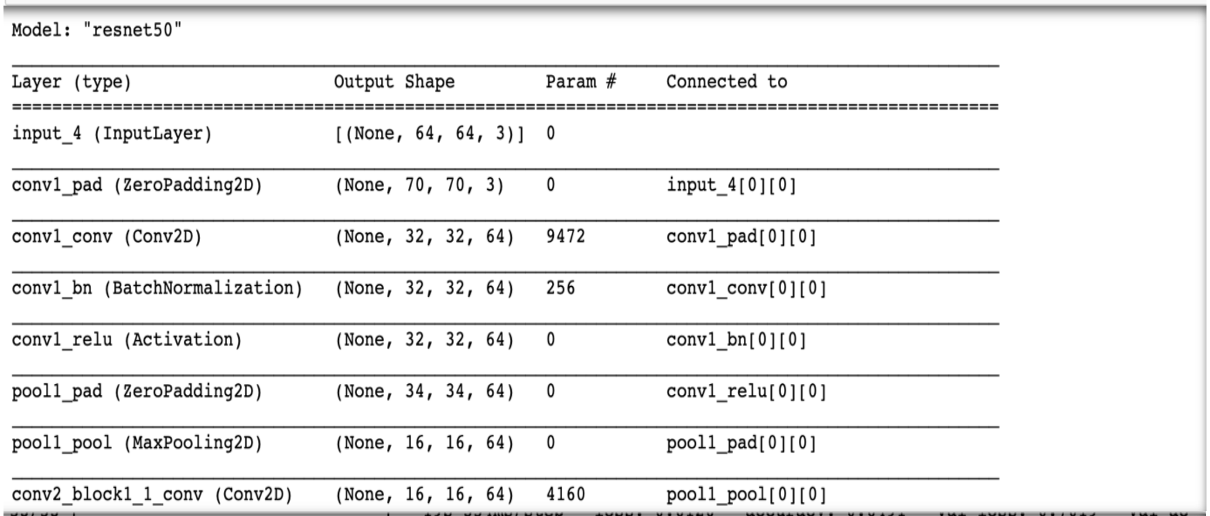
```
#ResnetNet Model
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.models import Model
from keras.applications.resnet50 import ResNet50

# MobileNet was designed to work on 224 x 224 pixel input images sizes
img_rows, img_cols = 64, 64

# Re-loads the ResN50et model without the top or FC layers
ResNet = ResNet50(weights = 'imagenet',
                 include_top = False,
                 input_shape = (img_rows, img_cols, 3))

ResNet.summary()
# Here we freeze the last 4 layers
# Layers are set to trainable as True by default
for layer in ResNet.layers:
    layer.trainable = False

# Let's print our layers
for (i,layer) in enumerate(ResNet.layers):
    print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
```

```
# Let's make a function that returns our FC Head
def addTopModelResNet(bottom_model, num_classes):
    """creates the top or head of the model that will be
    placed ontop of the bottom layers"""

    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(1,activation='sigmoid')(top_model)
    return top_model

num_classes = 2

FC_Head = addTopModelResNet(ResNet, num_classes)

model = Model(inputs = ResNet.input, outputs = FC_Head)

print(model.summary())


#Loading our Cancer Dataset

from keras.preprocessing.image import ImageDataGenerator

train_data_dir =  '/Users/vishalgoyal/Downloads/ImageProcessed/train/'
validation_data_dir ='/Users/vishalgoyal/Downloads/ImageProcessed/test/'
```

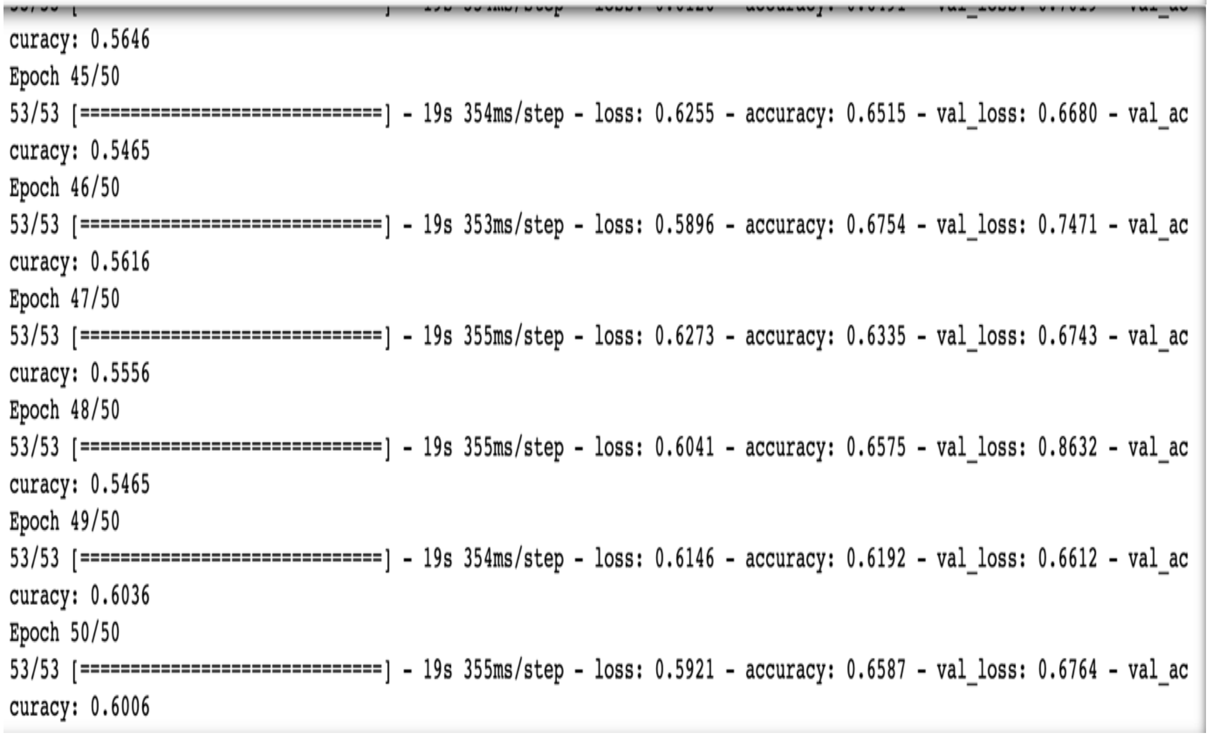
```
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=20,
      width_shift_range=0.2,
      height_shift_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# batchsize
train_batchsize = 16
val_batchsize = 1

train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=train_batchsize,
        class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=val_batchsize,
        class_mode='binary',
        shuffle=False)

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

epochs = 50
batch_size = 16
```



```
history = model.fit_generator(
    train_generator,
    epochs = epochs,
    validation_data = validation_generator)
```



```
Model: "resnet50"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)            [(None, 64, 64, 3)]  0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 70, 70, 3)    0           input_4[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 32, 32, 64)   9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 32, 32, 64)   256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 32, 32, 64)   0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 34, 34, 64)   0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 16, 16, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 16, 16, 64)   4160        pool1_pool[0][0]
```

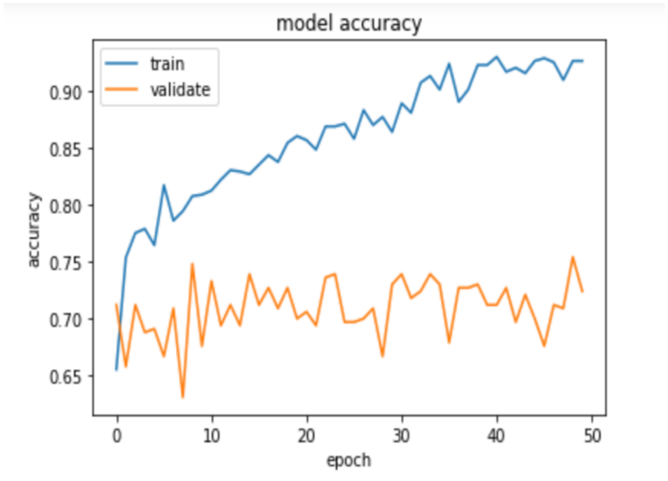


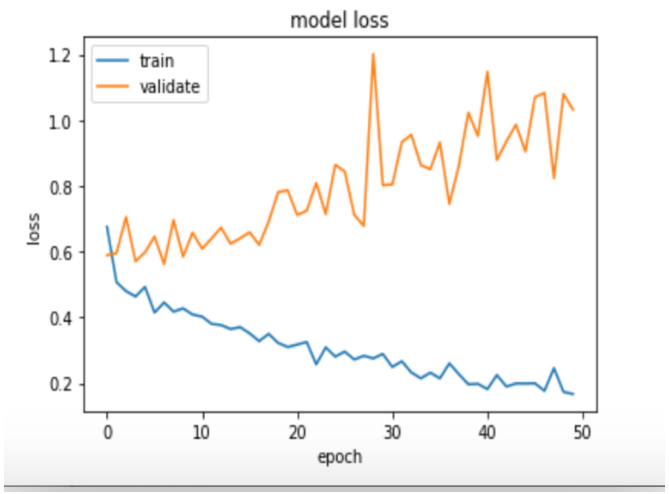
```
curacy: 0.5646
Epoch 45/50
53/53 [==============================] - 19s 354ms/step - loss: 0.6255 - accuracy: 0.6515 - val_loss: 0.6680 - val_ac
curacy: 0.5465
Epoch 46/50
53/53 [==============================] - 19s 353ms/step - loss: 0.5896 - accuracy: 0.6754 - val_loss: 0.7471 - val_ac
curacy: 0.5616
Epoch 47/50
53/53 [==============================] - 19s 355ms/step - loss: 0.6273 - accuracy: 0.6335 - val_loss: 0.6743 - val_ac
curacy: 0.5556
Epoch 48/50
53/53 [==============================] - 19s 355ms/step - loss: 0.6041 - accuracy: 0.6575 - val_loss: 0.8632 - val_ac
curacy: 0.5465
Epoch 49/50
53/53 [==============================] - 19s 354ms/step - loss: 0.6146 - accuracy: 0.6192 - val_loss: 0.6612 - val_ac
curacy: 0.6036
Epoch 50/50
53/53 [==============================] - 19s 355ms/step - loss: 0.5921 - accuracy: 0.6587 - val_loss: 0.6764 - val_ac
curacy: 0.6006
```

6. Evaluation

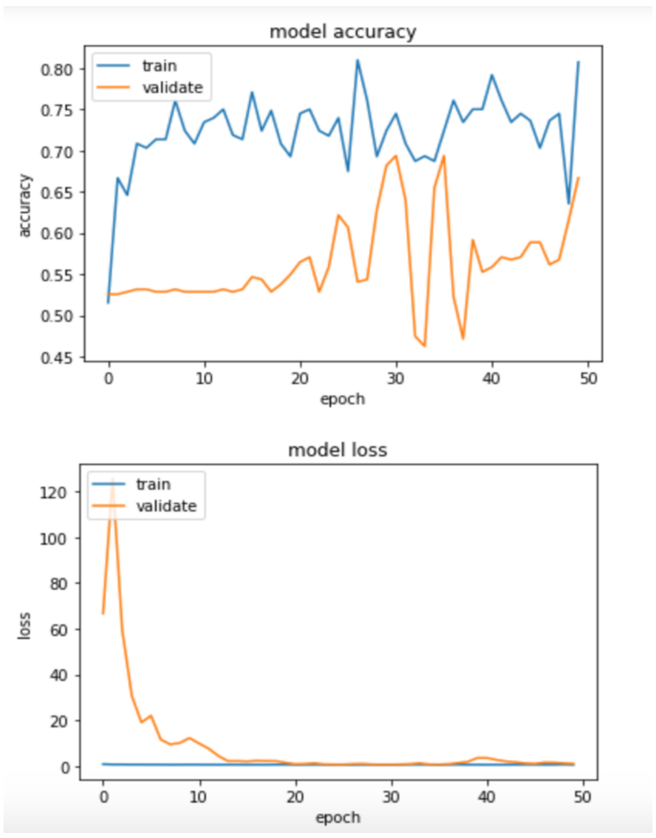
```
import plotly.express as px
import chart_studio.plotly as py
import plotly.graph_objs as go
from plotly.offline import iplot
import cufflinks
cufflinks.go_offline()
cufflinks.set_config_file(world_readable=True, theme='pearl')
import matplotlib.pyplot as plt


plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validate'], loc='upper left')
plt.show()
```

MobileNet Evaluation

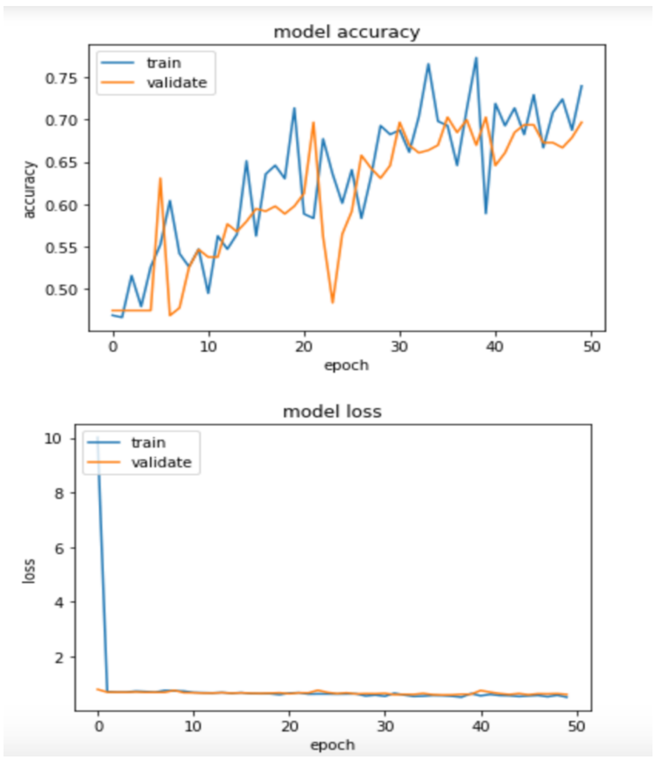




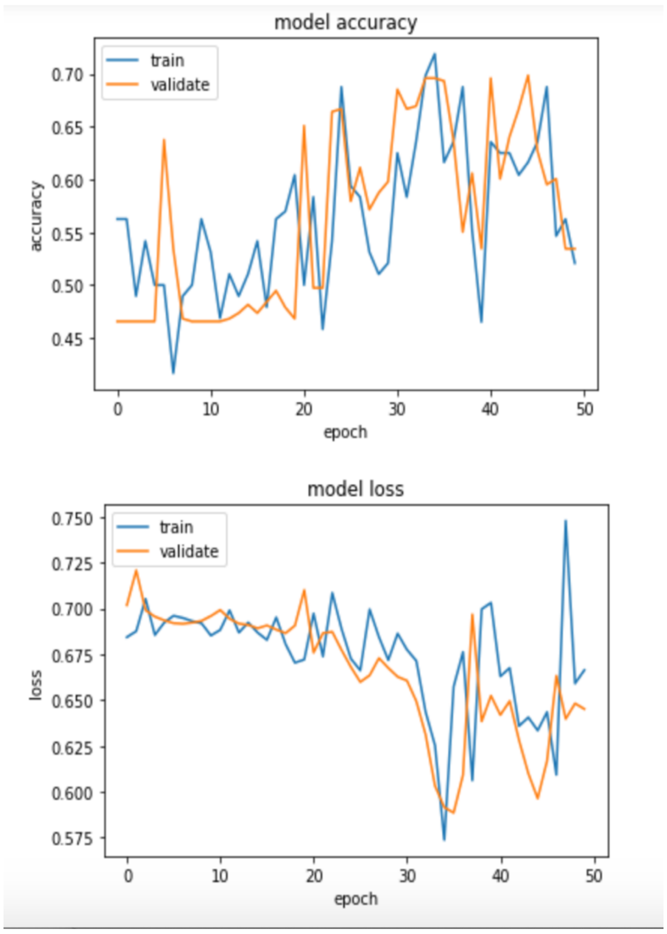
AlexNet Evaluation



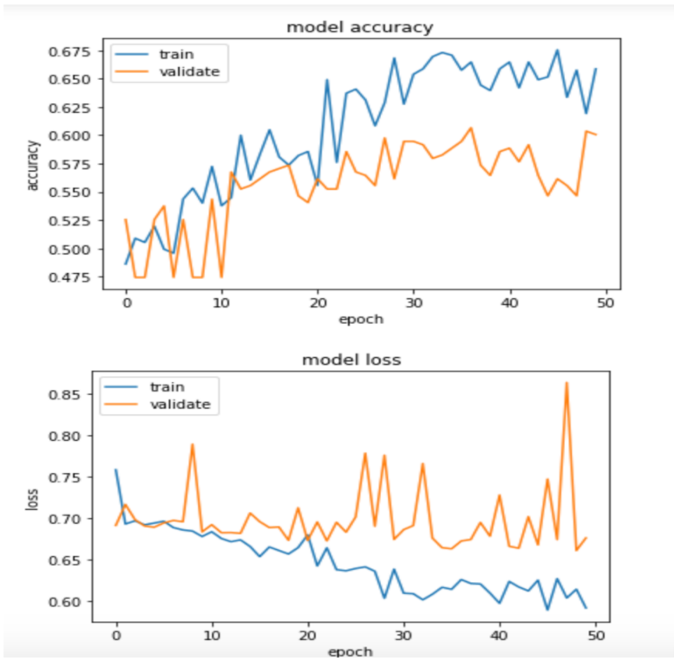
LeNet Evaluation



CNN Evaluation



ResNet50 Evaluation


model accuracy
accuracy
epoch
train
validate
model loss
loss
epoch
train
validate