International
Institute of Information
Technology Bangalore

Malaria Parasite Prediction

Project Report

November 2019

Team: ML Wizards

Aditya Bakliwal (MT2019008)

Manav Desai (MT2019060)

Shriya Kabra (MT2019142)

Table of Contents

Sr. No.	Title	Page No.
	Abstract	4
1.	Problem Statement	4
2.	Dataset Description	5
3.	Exploratory Data Analysis	6
4.	Pre-processing and Feature Extraction	7
5.	Model Building	12
6.	Observations and Comparison of Models	15

List of Figures

FIGURE 1: DISTRIBUTION OF IMAGES IN TRAINING DATASET 5
FIGURE 2: THE DIFFERENCE BETWEEN INFECTED AND UNINFECTED CELLS....5
FIGURE 3 : IMAGES MAY HAVE DIFFERENT DIMENSIONS 6
FIGURE 4 : PYTHON CODE SNIPPET FOR DIVIDING IMAGES INTO CHUNKS OF 100....7
FIGURE 5: DIRECTORIES, EACH HAVING 100 IMAGES 7
FIGURE 6: CSV FILE CONTAINING 100 ROWS FOR EACH FOLDER 8
FIGURE 7:CODE SNIPPET FOR GAUSSIAN BLURRING 8
FIGURE 8 : IMAGE AFTER APPLYING GAUSSIAN BLUR 9
FIGURE 9 : CODE SNIPPET FOR GRAY-SCALING 9
FIGURE 10: IMAGE AFTER GRAY-SCALING 9
FIGURE 11: CODE SNIPPET FOR GLOBAL THRESHOLDING 9
FIGURE 12: IMAGE AFTER THRESHOLDING 10
FIGURE 13: PRE-PROCESSING THE IMAGES 10
FIGURE 14: CREATE HISTOGRAM AND BINS 10
FIGURE 15: CODE SNIPPET FOR SIFT 11
FIGURE 16: PCA APPLIED ON FEATURE MATRIX OBTAINED BY SIFT 11
FIGURE 17: CODE SNIPPET FOR DECISION TREE MODEL 12
FIGURE 18: CODE SNIPPET FOR CNN MODEL....12
FIGURE 19: CNN MODEL....13
FIGURE 20: CODE SNIPPET FOR VGG MODEL 13
FIGURE 21: VGG MODEL....14
FIGURE 22:VALIDATION ACCURACY CURVE FOR CNN MODEL....15
FIGURE 23: VALIDATION LOSS CURVE FOR CNN MODEL....16
FIGURE 24: SNIPPET FOR VALIDATION ACCURACY AND LOSS OF VGG MODEL. 16
FIGURE 25: SNIPPET FOR VALIDATION ACCURACY AND LOSS FOR CNN MODEL....16
FIGURE 26: VALIDATION ACCURACY FOR DECISION TREE USING SIFT. 17
FIGURE 27: VALIDATION ACCURACY FOR NAIVE BAYES USING SIFT. 17
FIGURE 28: VALIDATION ACCURACY FOR RANDOM FOREST USING SIFT....17

Abstract

The aim of this project is to address the development of computer assisted **malaria parasite prediction and classification using Machine learning approaches.** We have applied a number of approaches (Traditional and Deep Learning) and Analysed their predictions. In the first approach, we have focussed on reducing noise by applying Gaussian Blurring on Grayscale images. We detected blob in the image by applying global thresholding and then passing it to the Decision Tree Model for prediction. In the second approach, we have applied Convolutional Neural Network on the thresholded images for making prediction. Also, we tried to improve by applying the VGG Pre-Trained Model for Image Classification.

1. Problem Statement

Given an image of a cell, the task is to predict whether it is parasitized or not. The images are publicly accessible from the following link:

https://www.kaggle.com/c/parasitedetection-iiitb2019/data

2. Dataset Description

- The dataset given comprises of two types of images: Uninfected and Parasitized
- Total images present in the dataset for training and validation purpose: 22046
- Total images present in the dataset for testing purpose: 1587
- There are 11023 infected and 11023 un-infected images in the training dataset.

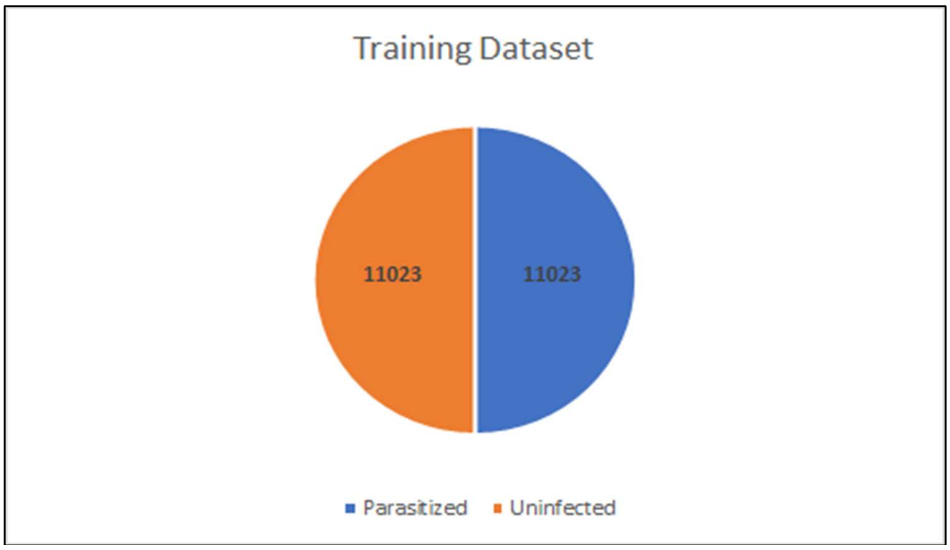


Figure 1: Distribution of Images in Training Dataset

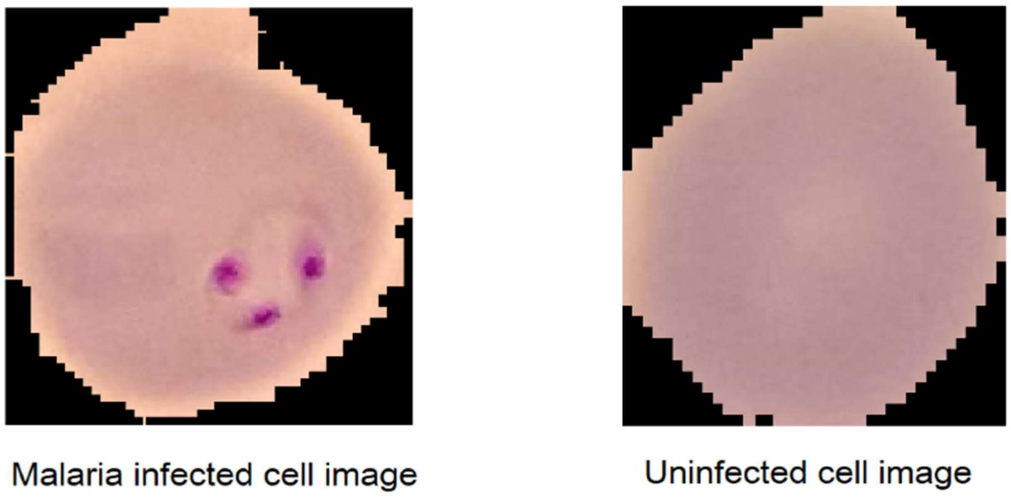
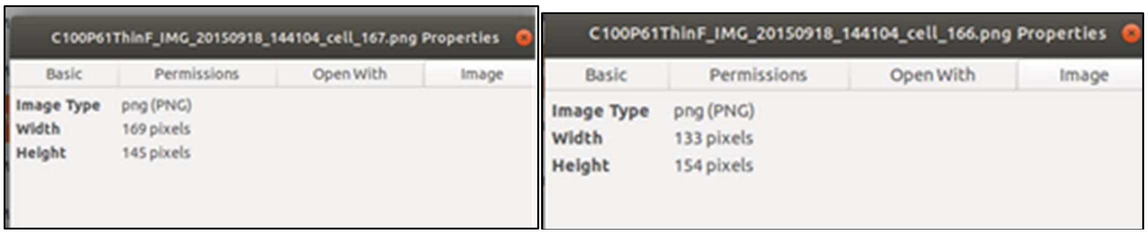


Figure 2: The Difference between Infected and Uninfected cells

3. Exploratory Data Analysis

Following are the manual observations from the dataset:

- Infected cells have a blob present in them which is having a higher intensity purple colour.
- Images are of different dimensions.



C100P61ThinF_IMG_20150918_144104_cell_167.png Properties

Image Type	png (PNG)
Width	169 pixels
Height	145 pixels

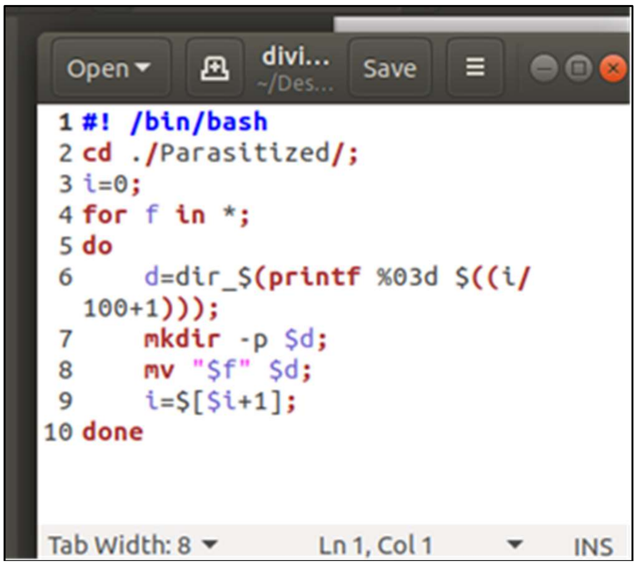
C100P61ThinF_IMG_20150918_144104_cell_166.png Properties

Image Type	png (PNG)
Width	133 pixels
Height	154 pixels

Figure 3 : Images may have different dimensions

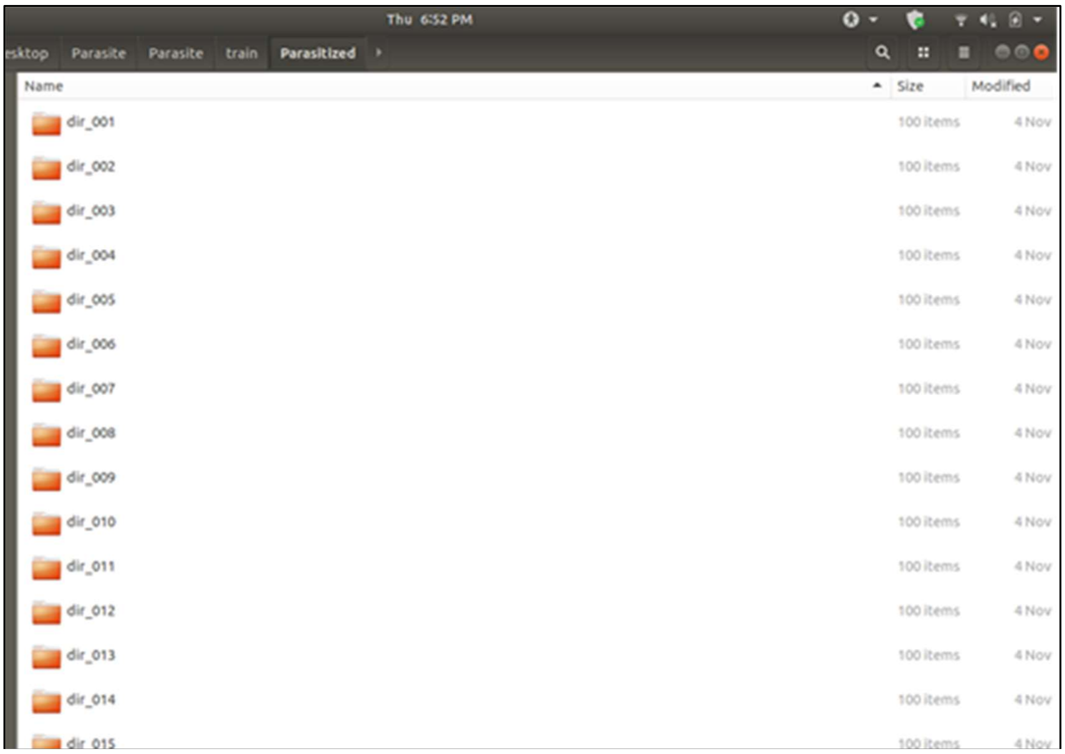
4. Pre-Processing and Feature Extraction

We divided all images in chunks of 100 images and then stored them in separate folders. For each folder, a CSV file is created with Image name, its features (black_stain = True or False) and Label. It is done to quickly load images in order to pre-process them while working on Jupyter Notebook locally.



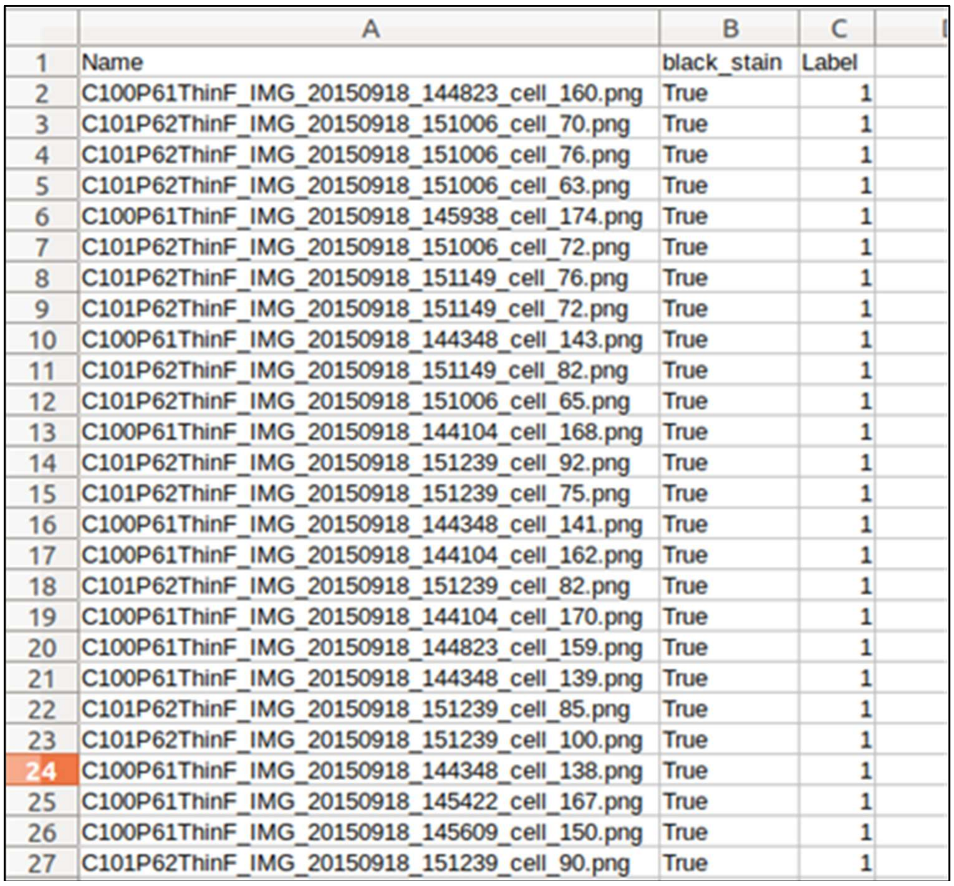
```
#! /bin/bash
cd ./Parasitized/;
i=0;
for f in *;
do
	d=dir_$(printf %03d $((i/100+1)));
	mkdir -p $d;
	mv "$f" $d;
	i=$[$i+1];
done
```

Figure 4 : Python Code Snippet for dividing images into chunks of 100



```
Name        Size        Modified
dir_001     100 items   4 Nov
dir_002     100 items   4 Nov
dir_003     100 items   4 Nov
dir_004     100 items   4 Nov
dir_005     100 items   4 Nov
dir_006     100 items   4 Nov
dir_007     100 items   4 Nov
dir_008     100 items   4 Nov
dir_009     100 items   4 Nov
dir_010     100 items   4 Nov
dir_011     100 items   4 Nov
dir_012     100 items   4 Nov
dir_013     100 items   4 Nov
dir_014     100 items   4 Nov
dir_015     100 items   4 Nov
```

Figure 5: Directories, each having 100 images



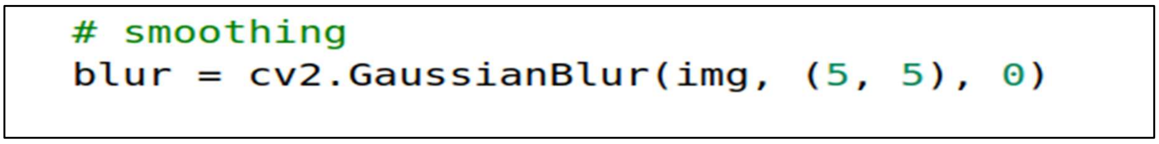
	A	B	C
1	Name	black_stain	Label
2	C100P61ThinF_IMG_20150918_144823_cell_160.png	True	1
3	C101P62ThinF_IMG_20150918_151006_cell_70.png	True	1
4	C101P62ThinF_IMG_20150918_151006_cell_76.png	True	1
5	C101P62ThinF_IMG_20150918_151006_cell_63.png	True	1
6	C100P61ThinF_IMG_20150918_145938_cell_174.png	True	1
7	C101P62ThinF_IMG_20150918_151006_cell_72.png	True	1
8	C101P62ThinF_IMG_20150918_151149_cell_76.png	True	1
9	C101P62ThinF_IMG_20150918_151149_cell_72.png	True	1
10	C100P61ThinF_IMG_20150918_144348_cell_143.png	True	1
11	C101P62ThinF_IMG_20150918_151149_cell_82.png	True	1
12	C101P62ThinF_IMG_20150918_151006_cell_65.png	True	1
13	C100P61ThinF_IMG_20150918_144104_cell_168.png	True	1
14	C101P62ThinF_IMG_20150918_151239_cell_92.png	True	1
15	C101P62ThinF_IMG_20150918_151239_cell_75.png	True	1
16	C100P61ThinF_IMG_20150918_144348_cell_141.png	True	1
17	C100P61ThinF_IMG_20150918_144104_cell_162.png	True	1
18	C101P62ThinF_IMG_20150918_151239_cell_82.png	True	1
19	C100P61ThinF_IMG_20150918_144104_cell_170.png	True	1
20	C100P61ThinF_IMG_20150918_144823_cell_159.png	True	1
21	C100P61ThinF_IMG_20150918_144348_cell_139.png	True	1
22	C101P62ThinF_IMG_20150918_151239_cell_85.png	True	1
23	C101P62ThinF_IMG_20150918_151239_cell_100.png	True	1
24	C100P61ThinF_IMG_20150918_144348_cell_138.png	True	1
25	C100P61ThinF_IMG_20150918_145422_cell_167.png	True	1
26	C100P61ThinF_IMG_20150918_145609_cell_150.png	True	1
27	C101P62ThinF_IMG_20150918_151239_cell_90.png	True	1

Figure 6: CSV File containing 100 rows for each folder

Applied traditional Image Processing techniques-

1st Approach for Pre-processing:

- **Applied Gaussian Blurring**: Used it to remove the noise from the image.



```
# smoothing
blur = cv2.GaussianBlur(img, (5, 5), 0)
```

Figure 7:Code Snippet for Gaussian Blurring

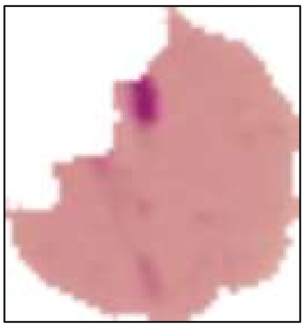


Figure 8 : Image after applying Gaussian Blur

- **Converted the Images to Grayscale:** Inherent complexity of gray level images is lower than that of coloured images. Also, we are interested in the intensity of the images, and Grayscale images are used for measuring the intensity of light in images.

```
gray_img = cv2.cvtColor(blur, cv2.COLOR_BGR2GRAY)
```

Figure 9 : Code Snippet for Gray-Scaling

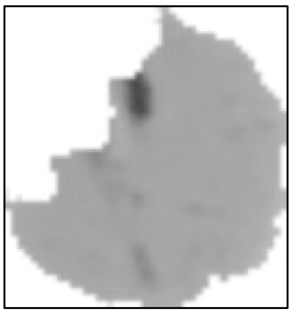


Figure 10: Image after Gray-Scaling

- **Applied Global Thresholding:** For separating out the black stain from the image as shown in the figure below.

```
ret, th1 = cv2.threshold(gray_img, 127, 255, cv2.THRESH_BINARY)
```

Figure 11: Code Snippet for Global Thresholding

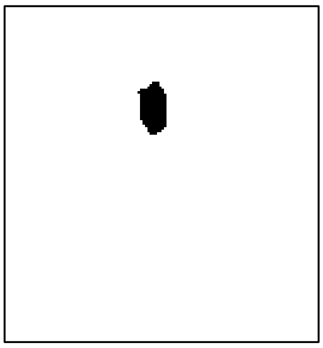


Figure 12: Image after Thresholding

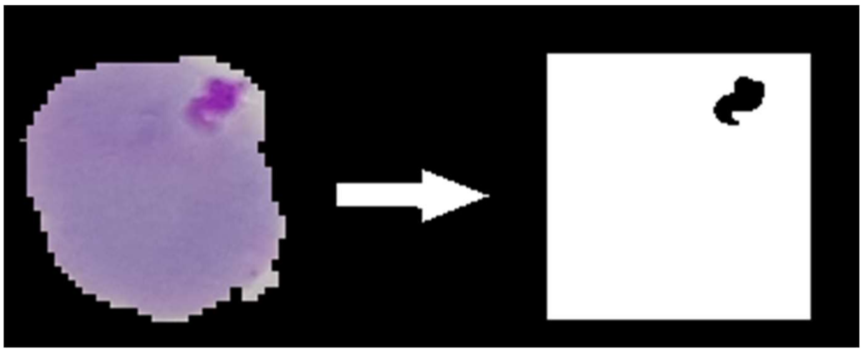
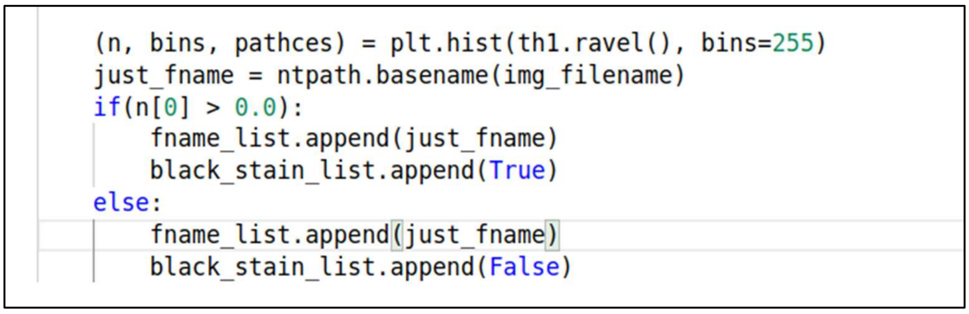


Figure 13: Pre-processing the images

- Create histogram of Black and White intensities for images, based on which we append the images with a black stain into list 'black_stain_list'. We use this as a feature that we can pass to the Decision Tree Model as explained earlier.



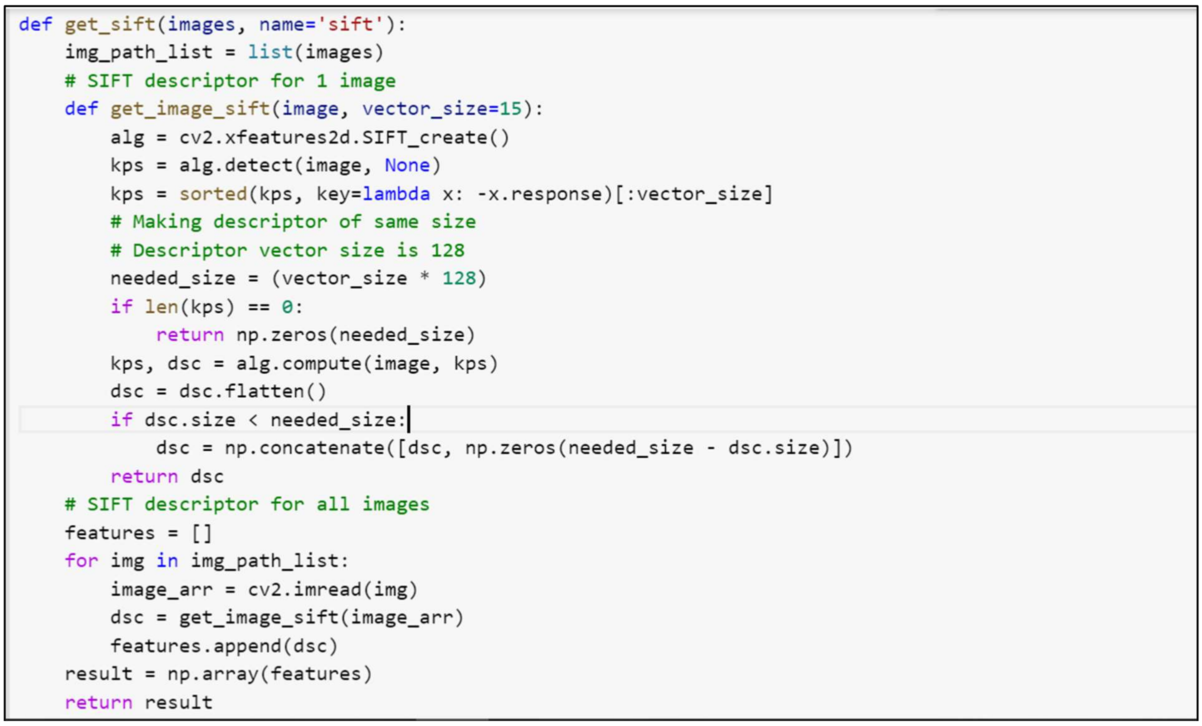
```
(n, bins, pathces) = plt.hist(th1.ravel(), bins=255)
just_fname = ntpath.basename(img_filename)
if(n[0] > 0.0):
    fname_list.append(just_fname)
    black_stain_list.append(True)
else:
    fname_list.append(just_fname)
    black_stain_list.append(False)
```

Figure 14: Create Histogram and Bins

- We **train Decision Tree Classifier** on these images and the stain as a feature to make predictions for parasitized or non-parasitized Malaria Cells.

2nd Approach for Pre-processing:

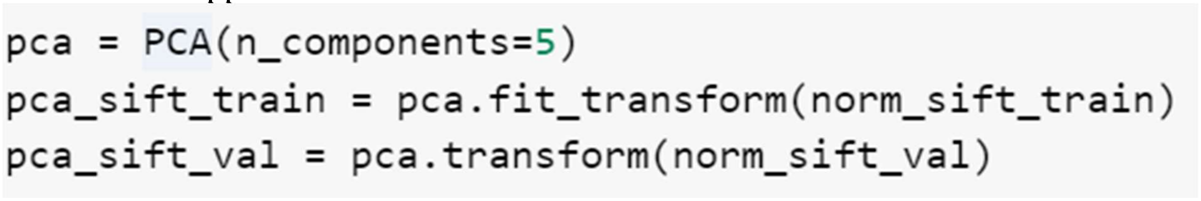
- Applied SIFT to extract features



```python
def get_sift(images, name='sift'):
    img_path_list = list(images)
    # SIFT descriptor for 1 image
    def get_image_sift(image, vector_size=15):
        alg = cv2.xfeatures2d.SIFT_create()
        kps = alg.detect(image, None)
        kps = sorted(kps, key=lambda x: -x.response)[:vector_size]
        # Making descriptor of same size
        # Descriptor vector size is 128
        needed_size = (vector_size * 128)
        if len(kps) == 0:
            return np.zeros(needed_size)
        kps, dsc = alg.compute(image, kps)
        dsc = dsc.flatten()
        if dsc.size < needed_size:
            dsc = np.concatenate([dsc, np.zeros(needed_size - dsc.size)])
        return dsc
    # SIFT descriptor for all images
    features = []
    for img in img_path_list:
        image_arr = cv2.imread(img)
        dsc = get_image_sift(image_arr)
        features.append(dsc)
    result = np.array(features)
    return result
```

Figure 15: Code Snippet for SIFT

- Applied PCA on the features extracted from SIFT.



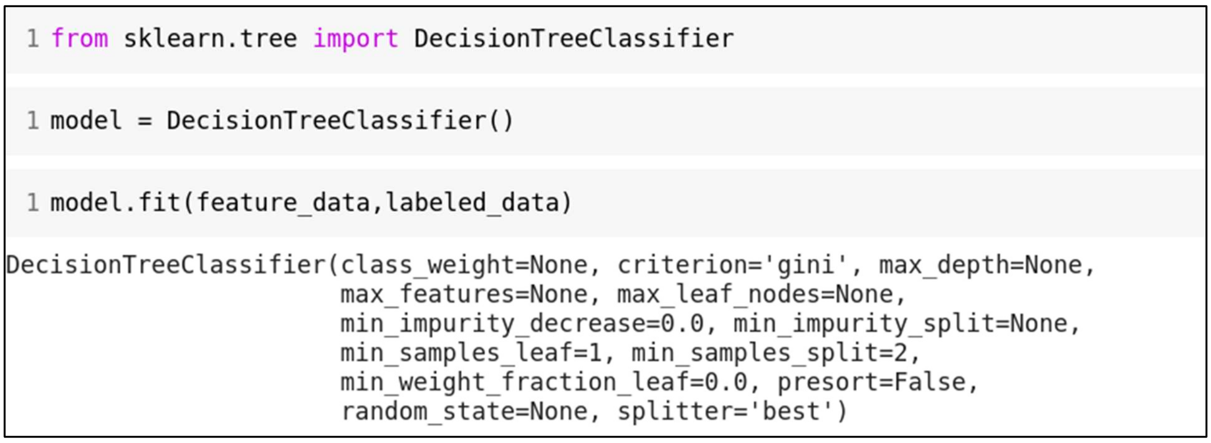
```python
pca = PCA(n_components=5)
pca_sift_train = pca.fit_transform(norm_sift_train)
pca_sift_val = pca.transform(norm_sift_val)
```

Figure 16: PCA Applied on Feature Matrix Obtained by SIFT

5. Model Building

We first split the dataset into train and test sets and then apply the following models:

- We have applied **Decision Trees** on the images obtained after applying the approach described under section 5.1 (Thresholded images). The reason for applying decision trees is that we have only one feature and accuracy obtained is also considerably good.



```
from sklearn.tree import DecisionTreeClassifier
```

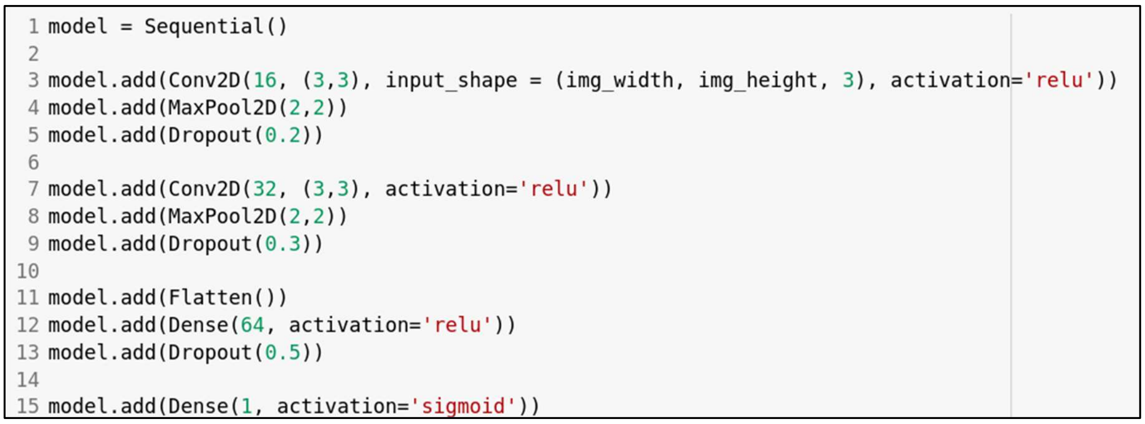
```
model = DecisionTreeClassifier()
```

```
model.fit(feature_data,labeled_data)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

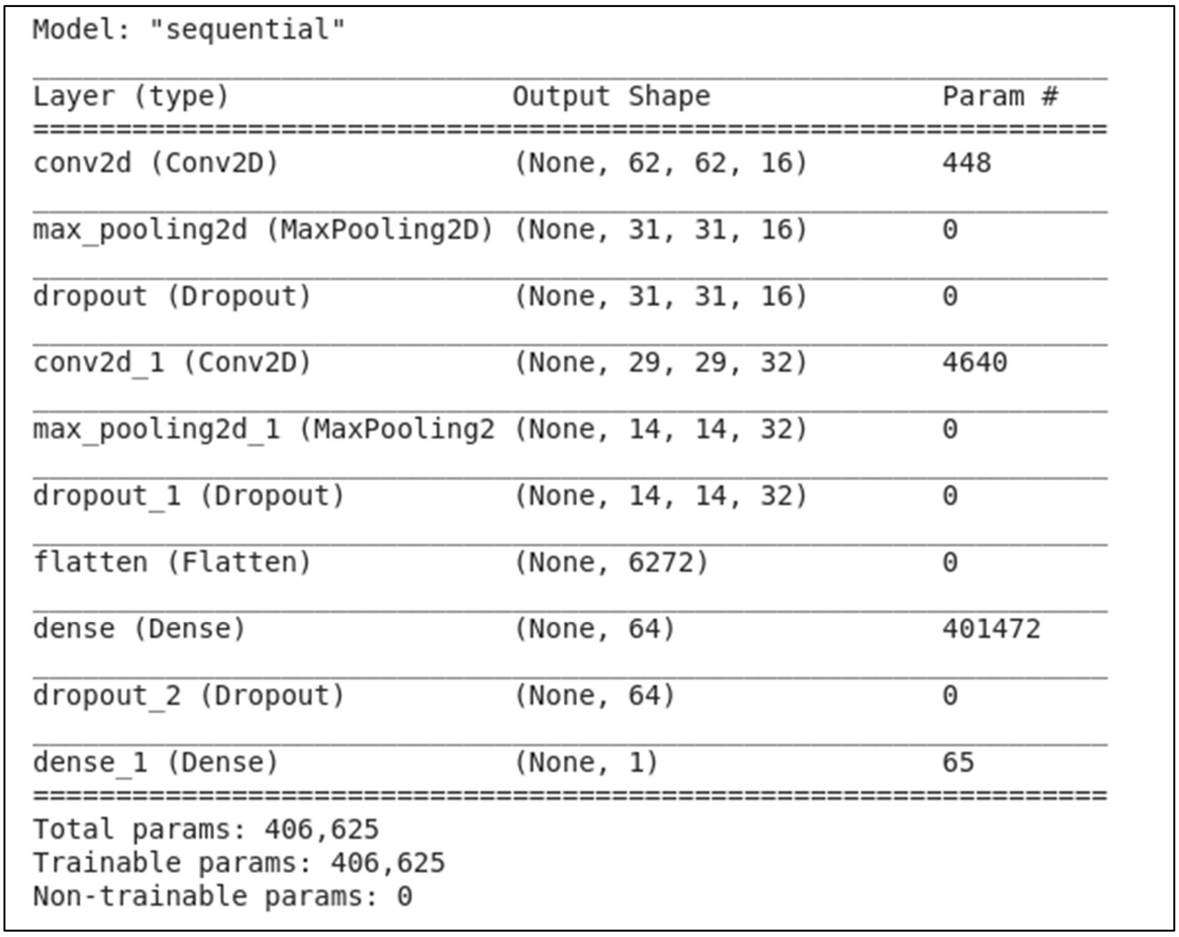
Figure 17: Code Snippet for Decision Tree Model

- We have applied **CNN Model** (Deep Learning Approach) on the original dataset (after resizing all the images to same dimensions).



```
model = Sequential()

model.add(Conv2D(16, (3,3), input_shape = (img_width, img_height, 3), activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

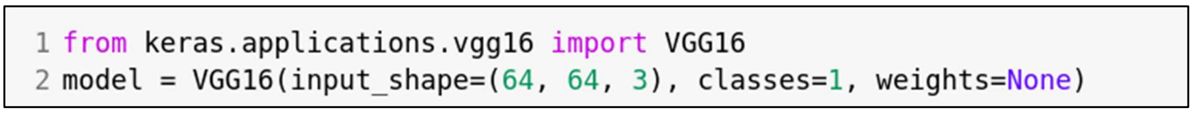
Figure 18: Code Snippet for CNN Model



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 16)        0
_________________________________________________________________
dropout (Dropout)            (None, 31, 31, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 64)                401472
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 406,625
Trainable params: 406,625
Non-trainable params: 0
```

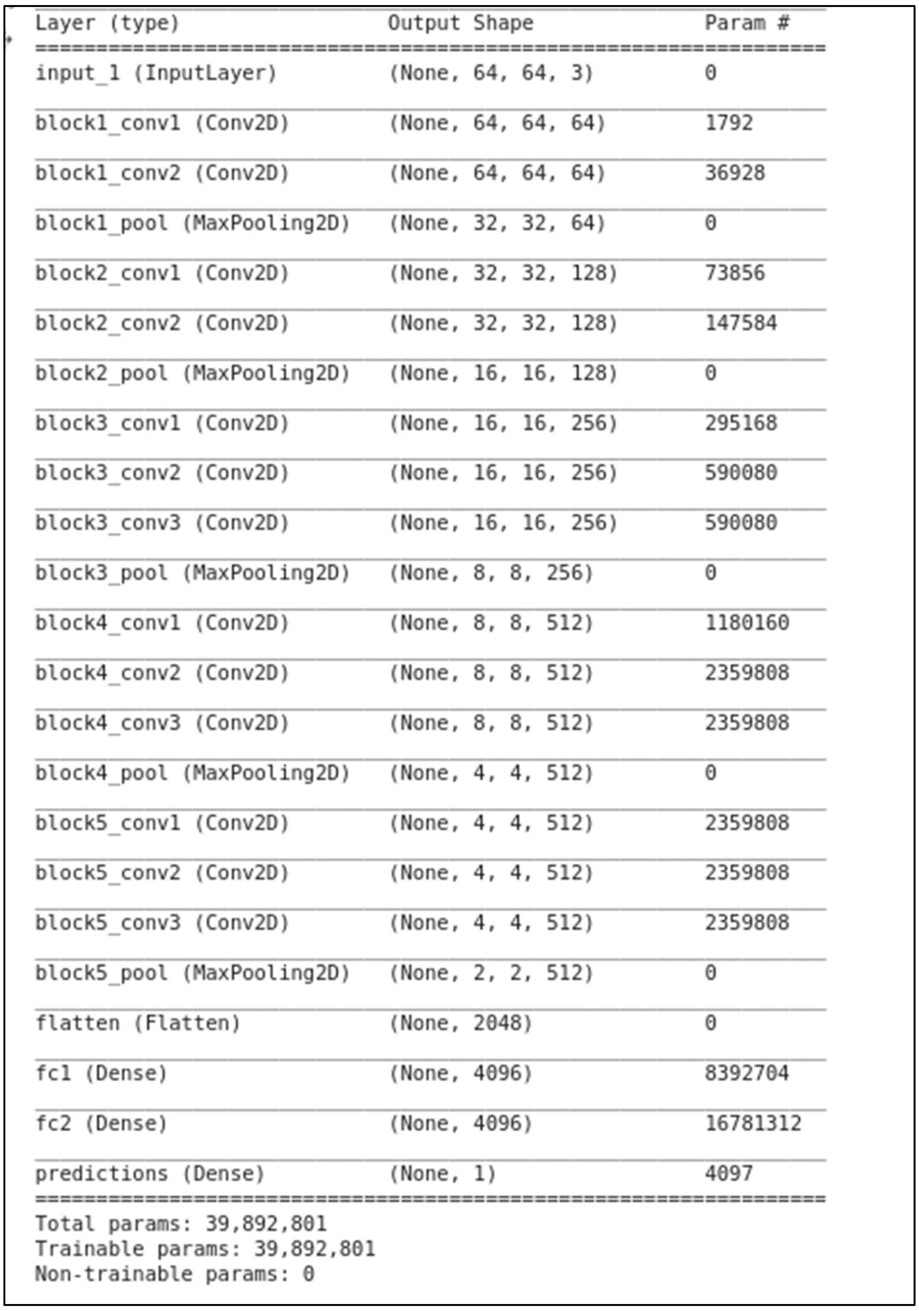
Figure 19: CNN Model

- We have applied **VGG Model** (Pre-Trained Model) on the dataset. VGG has been shown to perform well on complex datasets like the ImageNet dataset which has over 1000 categories. Since ours is a much simpler dataset, we decided to apply VGG on our dataset.



```
from keras.applications.vgg16 import VGG16
model = VGG16(input_shape=(64, 64, 3), classes=1, weights=None)
```

Figure 20: Code Snippet for VGG Model



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 64, 64, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 64, 64, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 64, 64, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 32, 32, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 32, 32, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 32, 32, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 16, 16, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 16, 16, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 16, 16, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 8, 8, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 8, 8, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 4, 4, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              8392704
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1)                 4097
=================================================================
Total params: 39,892,801
Trainable params: 39,892,801
Non-trainable params: 0
```

Figure 21: VGG Model

6. Observations and Comparison of Models

Approaches	Validation Accuracy
Thresholding + Decision Tree	92.53%
SIFT + Random Forest	89.13%
SIFT + Decision Tree	83.94%
SIFT + Naïve Bayes	82.42%
CNN	94.42%
VGG	50.00%

Results obtained from different approaches:

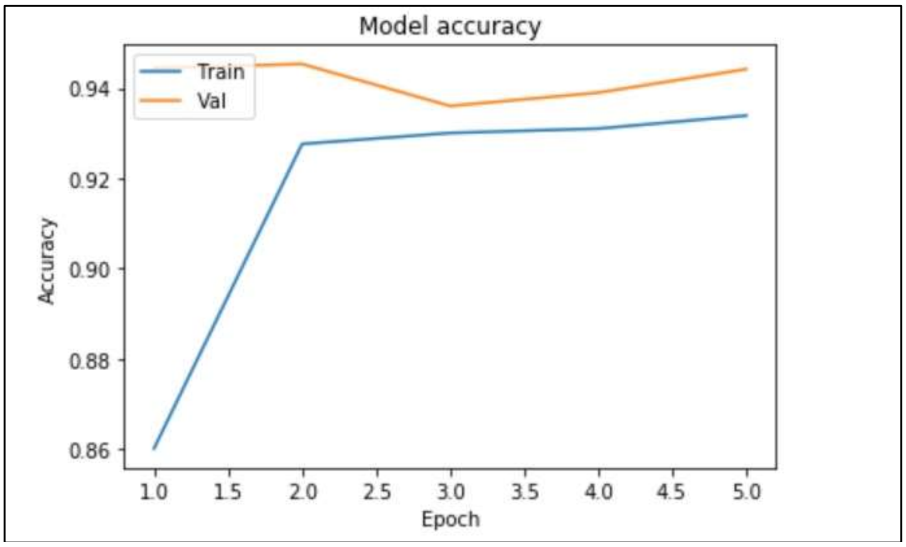


Figure 22:Validation Accuracy Curve for CNN Model

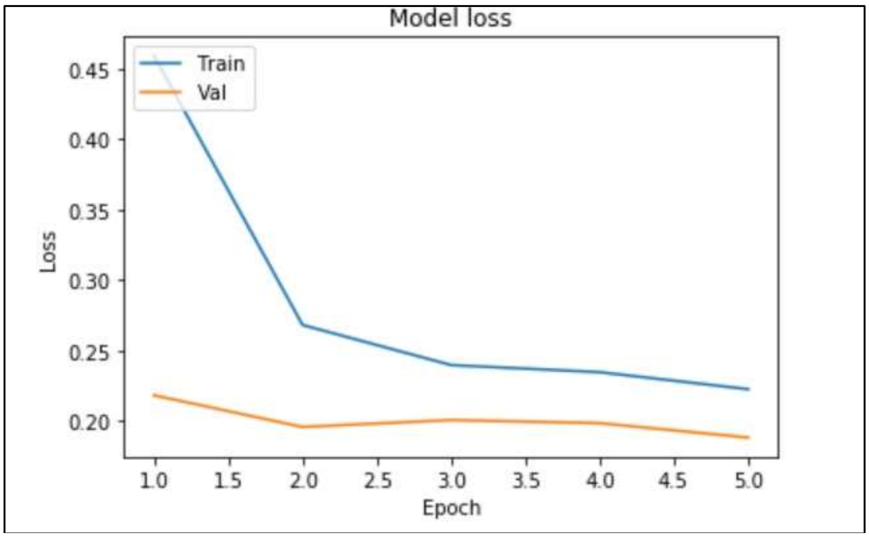
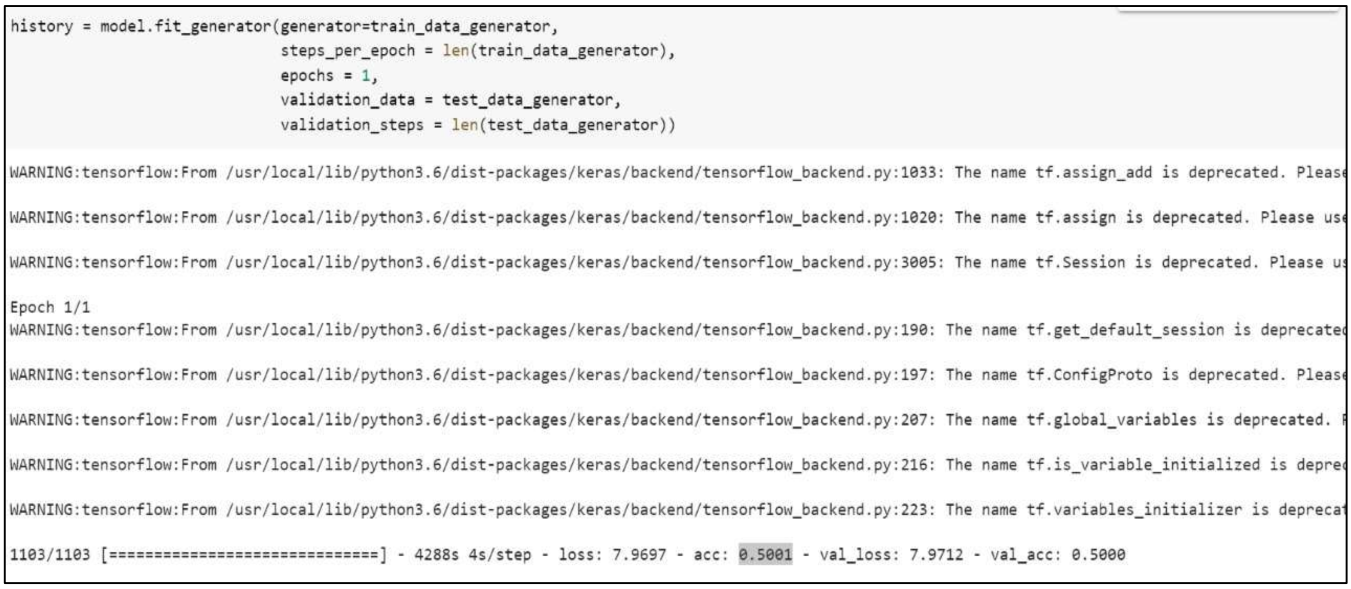
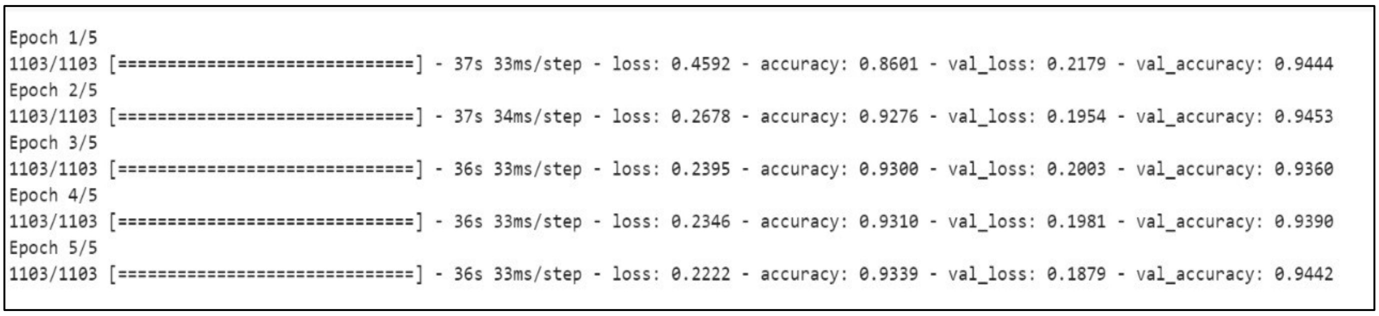


Figure 23: Validation Loss Curve for CNN Model



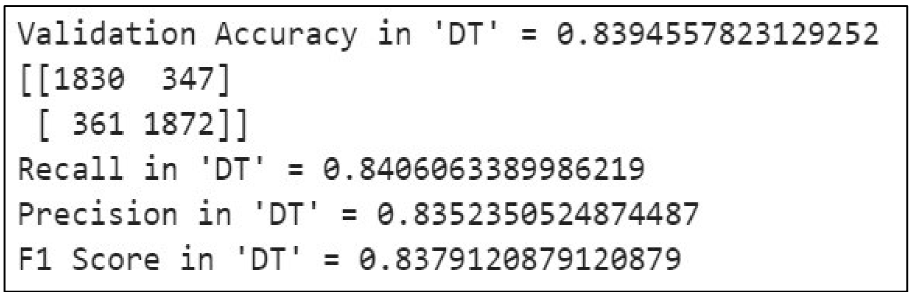
```
history = model.fit_generator(generator=train_data_generator,
                              steps_per_epoch = len(train_data_generator),
                              epochs = 1,
                              validation_data = test_data_generator,
                              validation_steps = len(test_data_generator))

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3005: The name tf.Session is deprecated. Please us

Epoch 1/1
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:190: The name tf.get_default_session is deprecated

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:207: The name tf.global_variables is deprecated. P

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:216: The name tf.is_variable_initialized is depre

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:223: The name tf.variables_initializer is deprecat

1103/1103 [==============================] - 4288s 4s/step - loss: 7.9697 - acc: 0.5001 - val_loss: 7.9712 - val_acc: 0.5000
```

Figure 24: Snippet for Validation Accuracy and Loss of VGG Model.



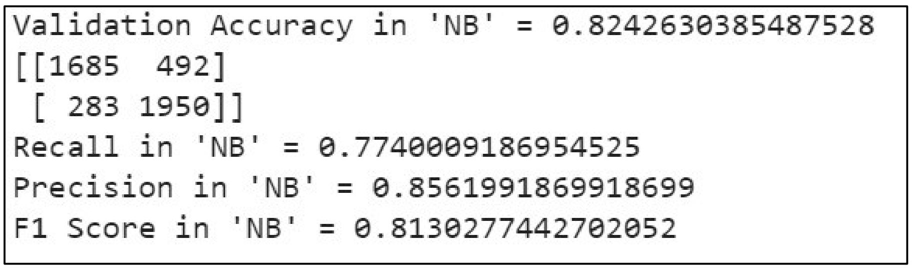
```
Epoch 1/5
1103/1103 [==============================] - 37s 33ms/step - loss: 0.4592 - accuracy: 0.8601 - val_loss: 0.2179 - val_accuracy: 0.9444
Epoch 2/5
1103/1103 [==============================] - 37s 34ms/step - loss: 0.2678 - accuracy: 0.9276 - val_loss: 0.1954 - val_accuracy: 0.9453
Epoch 3/5
1103/1103 [==============================] - 36s 33ms/step - loss: 0.2395 - accuracy: 0.9300 - val_loss: 0.2003 - val_accuracy: 0.9360
Epoch 4/5
1103/1103 [==============================] - 36s 33ms/step - loss: 0.2346 - accuracy: 0.9310 - val_loss: 0.1981 - val_accuracy: 0.9390
Epoch 5/5
1103/1103 [==============================] - 36s 33ms/step - loss: 0.2222 - accuracy: 0.9339 - val_loss: 0.1879 - val_accuracy: 0.9442
```

Figure 25: Snippet for Validation Accuracy and Loss for CNN Model.



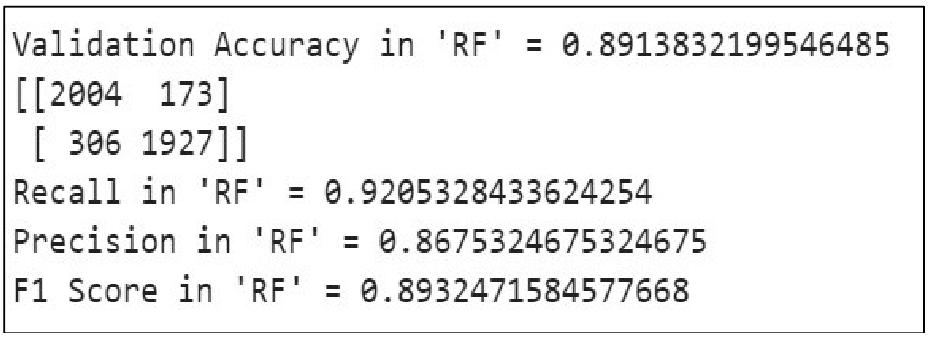
```
Validation Accuracy in 'DT' = 0.8394557823129252
[[1830  347]
 [ 361 1872]]
Recall in 'DT' = 0.8406063389986219
Precision in 'DT' = 0.8352350524874487
F1 Score in 'DT' = 0.8379120879120879
```

Figure 26: Validation Accuracy for Decision Tree using SIFT.



```
Validation Accuracy in 'NB' = 0.8242630385487528
[[1685  492]
 [ 283 1950]]
Recall in 'NB' = 0.7740009186954525
Precision in 'NB' = 0.8561991869918699
F1 Score in 'NB' = 0.8130277442702052
```

Figure 27: Validation Accuracy for Naive Bayes using SIFT.



```
Validation Accuracy in 'RF' = 0.8913832199546485
[[2004  173]
 [ 306 1927]]
Recall in 'RF' = 0.9205328433624254
Precision in 'RF' = 0.8675324675324675
F1 Score in 'RF' = 0.8932471584577668
```

Figure 28: Validation Accuracy for Random Forest using SIFT.