Image Classification of Insect Bites using Convolutional Neural Network

1. Code File for Data collection :

Step 1: GoogleImageCrawler is used to perform web scrapping from dermatology websites.

Different keyword is given to scrap the images.

```
# Download the images from websites using google image crawler if the keyword-'sandfly bite' matches
# Different keywords was used to download the sandfly bite image.
google_crawler = GoogleImageCrawler(storage={'root_dir': 'C:\\Users\\nandh\\OneDrive\\Documents\\Masters Matl\\Semester2\\AI\\As
google_crawler.crawl(keyword='sandfly bite', max_num=200)
```

```
2019-03-04 12:04:57,354 - INFO - icrawler.crawler - start crawling...
2019-03-04 12:04:57,354 - INFO - icrawler.crawler - starting 1 feeder threads...
2019-03-04 12:04:57,358 - INFO - feeder - thread feeder-001 exit
2019-03-04 12:04:57,362 - INFO - icrawler.crawler - starting 1 parser threads...
2019-03-04 12:04:57,368 - INFO - icrawler.crawler - starting 1 downloader threads...
2019-03-04 12:04:58,026 - INFO - parser - parsing result page https://www.google.com/search?q=sandfly+bite&ijn=0&start=0&tbs=
&tbm=isch
2019-03-04 12:04:58,361 - INFO - downloader - image #1  http://farnorthtraining.com.au/wp-content/uploads/2015/01/Indonesian_
sandfly_bites_on_leg.jpg
2019-03-04 12:04:58,444 - INFO - downloader - image #2  https://upload.wikimedia.org/wikipedia/commons/thumb/7/77/SandFlyBit
e.JPG/200px-SandFlyBite.JPG
2019-03-04 12:04:59,223 - INFO - downloader - image #3  https://co0069yjui-flywheel.netdna-ssl.com/wp-content/uploads/2017/0
6/Sand-Fly-Bites-300x199.jpg
2019-03-04 12:04:59,654 - INFO - downloader - image #4  https://cdn-img.health.com/sites/default/files/styles/medium_16_9/pub
lic/styles/main/public/gettyimages-586114450.jpg?itok=fFl6wLi9&1528470728
2019-03-04 12:04:59,953 - INFO - downloader - image #5  https://www.healthline.com/hlcmsresource/images/Image-Galleries/Fly-B
ites/642x361_Slide_3_Fly_bites.jpg
2019-03-04 12:05:00,591 - INFO - downloader - image #6  https://www.hiddenvalleyholidaypark.com.au/wp-content/uploads/2014/0
2/Blog-2_DF_Midges-Sand-Flies_bites_DR.jpg
```

Step 2: Images of various insect bites are also collected from medical videos in YouTube. The video is downloaded using 4K video downloader and later frames are captured from the videos. The frames are then saved in a folder and manual interpretation is done to choose the best frames with good lighting and settings.

```
# Medical images of sandfly bites is also obtained from youtube videos.
# Many frames was captured from the vidoes and the images and the images with best setting and quality is used in the dataset.
def vid_to_frames(input_path, output_path):

    try:
        os.mkdir(output_path)
    except OSError:
        pass
    time_start = time.time()
    cap = cv2.VideoCapture(input_path)
    video_length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) - 1
    print ("Number of frames: ", video_length)
    count = 0
    print ("Converting video..\n")
    while cap.isOpened():
        ret, frame = cap.read()
        cv2.imwrite(output_path + "/%#05d.jpg" % (count+1), frame)
        count = count + 5
        if (count > (video_length-1)):
            time_end = time.time()
            cap.release()
            print ("\n%d frames extracted" % count)
            print ("%d seconds forconversion." % (time_end-time_start))
            break

input_path = '''C:\\Users\\nandh\\Videos\\4K Video Downloader\\FNN Syria Aleppo al Mashed a child infected with L.mp4'''
output_path = 'C:\\Users\\nandh\\OneDrive\\Documents\\Masters Matl\\Semester2\\AI\\Assignments\\images'
vid_to_frames(input_path, output_path)
```

```
Number of frames:  1887
Converting video..


1890 frames extracted
10 seconds forconversion.
```

Step 3: All the images are resized to 224*224 as the VGG network expects an input size of 224*224.

```
# Resize image to 224*224
path = "C:\\Users\\nandh\\OneDrive\\Documents\\Masters Matl\\Semester2\\AI\\Assignments\\dataset\\test_set\\TickBites"
dirs = os.listdir( path )

def resize():
    for item in dirs:
        input_path = os.path.join(path, item)
        im = Image.open(input_path)
        f, e = os.path.splitext(input_path)
        imResize = im.resize((224,224), Image.ANTIALIAS)
        imResize.convert('RGB').save(f + ' resize224.jpg', 'JPEG', quality=90)

resize()
```

2. **Code File for Data Visualization :** In order to understand the distribution of high-dimensional dataset , i.e how close the images are related to one another t-SNE was used to visualize the image dataset.
   (Code Reference - https://medium.com/@luckylwk/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b)

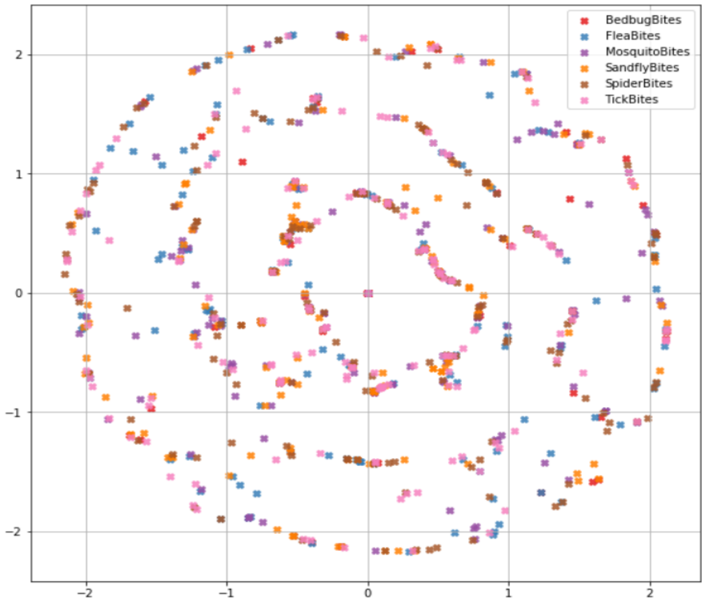
```
tsne = TSNE(n_components=2, perplexity=40.0)
```

```
tsne_result = tsne.fit_transform(pca_result)
```

```
tsne_result_scaled = StandardScaler().fit_transform(tsne_result)
```

```
def visualize_scatter(data_2d, label_ids, figsize=(10,10)):
    plt.figure(figsize=figsize)
    plt.grid()

    nb_classes = len(np.unique(label_ids))

    for label_id in np.unique(label_ids):
        plt.scatter(data_2d[np.where(label_ids == label_id), 0],
                    data_2d[np.where(label_ids == label_id), 1],
                    marker='X',
                    color= plt.cm.Set1(label_id / float(nb_classes)),
                    linewidth='1',
                    alpha=0.8,
                    label=id_to_label_dict[label_id])
    plt.legend(loc='best')
```

```
visualize_scatter(tsne_result_scaled, label_ids)
```



3. **Train/Validation and Test data split :** Three separate folder are created for train/validation and test set. 960 images are used for training (160 images per class and there are 6 class in total). 240 images are used for validation and test set (40 images per class). The below code is to view the sample images in the dataset.

- test_set
- training_set
- valid_set

```
# To view sample of images to get insight about the data.
fig = plt.figure(1, figsize=(7, 7))
grid = ImageGrid(fig, 111, nrows_ncols=(6, 6), axes_pad=0.05)
i = 0
for category_id, category in enumerate(Labels):
    for filepath in train[train['label'] == category]['file'].values[:Num_Labels]:
        ax = grid[i]
        img = read_img(filepath, (224, 224))
        ax.imshow(img / 255.)
        ax.axis('off')
        if i % Num_Labels == Num_Labels - 1:
            ax.text(250, 112, filepath.split('/')[1], verticalalignment='center')
        i += 1
plt.show();
```

4. **Code file for Image Augmentation :** ImageDataGenerator of keras was used for image augmentation. For the training dataset we first rescale the image and use transformations like feature_wise centre, feature_wise_std_normalization, rotation, zoom, width_shift, height_shift, zca whitening, horizontal and vertical flip. For validation dataset , we just rescale the image and do not use any image transformation.

```
img_width, img_height = 224, 224
batch_size = 16
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    featurewise_center=True,
    samplewise_center=False,
    featurewise_std_normalization=True,
    samplewise_std_normalization=False,
    rotation_range=90,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zca_whitening=True,
    horizontal_flip=True,
    vertical_flip=True)

train_generator = train_datagen.flow_from_directory(
    train_set,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

valid_datagen = ImageDataGenerator(rescale=1. / 255)

validation_generator = valid_datagen.flow_from_directory(
    valid_set,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
```

```
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:339: UserWarning: This ImageDataGenera
tor specifies `zca_whitening` which overrides setting of`featurewise_std_normalization`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
Found 960 images belonging to 6 classes.
Found 240 images belonging to 6 classes.
```

5. Code files for transfer learning thru VGG-16 and VGG-19

Step 1: First we extract all the features from the VGG-16 and VGG-19 pre-trained model and feed this as input to our classifier.

Code snippet to extract the features from training set:

```
model_vgg19 = applications.VGG19(include_top=False, input_shape=(img_width, img_height, 3),weights='imagenet')
model_vgg19.summary()
```

```
bottleneck_features_train_vgg19_WA = model_vgg19.predict_generator(train_generator, predict_train_size)
```

```
bottleneck_features_train_vgg19_WA.shape
```

```
(960, 7, 7, 512)
```

Code snippet to extract the features from training set:

```
bottleneck_features_valid_vgg19_WA = model_vgg19.predict_generator(validation_generator, predict_valid_size)
```

```
np.save('bottleneck_features_valid_vgg19_wa.npy', bottleneck_features_valid_vgg19_WA)
```

```
bottleneck_features_valid_vgg19_WA.shape
```

```
(240, 7, 7, 512)
```

Load the bottleneck features and save the model:

```
# load the bottleneck features saved earlier
train_data = np.load('bottleneck_features_train_vgg19_wa.npy')
```

```
# load the bottleneck features saved earlier
valid_data = np.load('bottleneck_features_valid_vgg19_wa.npy')
```

Build the classifier model where the output layer consists of 6 classes.

```
model_bn = Sequential()
model_bn.add(Flatten(input_shape=train_data.shape[1:]))
model_bn.add(Dense(512, activation='relu',kernel_initializer='he_normal'))
model_bn.add(BatchNormalization(axis=-1))
model_bn.add(Dropout(0.2))
model_bn.add(Dense(num_classes, activation='softmax'))
model_bn.summary()
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               12845568
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 3078
=================================================================
Total params: 12,850,694
Trainable params: 12,849,670
Non-trainable params: 1,024
_________________________________________________________________
```

Step 2: As we discussed in the report, we experimented with five different models which is constructed by fine-tuning and freezing certain convolutional layers.

```
vgg19 = applications.VGG19(include_top=False, weights='imagenet', input_shape=(img_width, img_height, 3))
```

```
Model_vgg19_WA1 = Model(inputs= vgg19.input, outputs= model_bn(vgg19.output))
```

Code snippet to freeze and fine-tune the convolutional layers:

Model 1: First layer is trained, and other layers are freezed.

```
for layer in Model_vgg19_WA1.layers[:-1]:
    layer.trainable = False
```

Model 2: Last 3 layers (sequential_1, block5_pool, block5_conv4) are trained and other layers are freezed.

```
for layer in Model_vgg19_WA2.layers[:-3]:
    layer.trainable = False
```

Model 3: Last 6 layers (sequential_1, block5_pool, block5_conv4, block5_conv3, block5_conv2 block5_conv1) are trained and other layers are freezed.

```
for layer in Model_vgg19_WA3.layers[:-6]:
    layer.trainable = False
```

Model 4: Last 8 layers (sequential_1, block5_pool, block5_conv4, block5_conv3, block5_conv2 block5_conv1, block4_pool, block4_conv4) are trained and other layers are freezed.

```
for layer in Model_vgg19_WA4.layers[:-8]:
    layer.trainable = False
```

Model 5: Last 9 layers (sequential_1, block5_pool, block5_conv4, block5_conv3, block5_conv2 block5_conv1, block4_pool, block4_conv4, block4_conv3) are trained and other layers are freezed.

```
for layer in Model_vgg19_WA5.layers[:-9]:
    layer.trainable = False
```

Model 5 fine-tuning and freezed layers can be viewed as follows-

```
layers = [(layer, layer.name, layer.trainable) for layer in Model_vgg19_WA5.layers]
pd.DataFrame(layers, columns=['Layer', 'Layer Name', 'Layer Train'])
```

	Layer	Layer Name	Layer Train
0	<keras.engine.input_layer.InputLayer object at...	input_3	False
1	<keras.layers.convolutional.Conv2D object at 0...	block1_conv1	False
2	<keras.layers.convolutional.Conv2D object at 0...	block1_conv2	False
3	<keras.layers.pooling.MaxPooling2D object at 0...	block1_pool	False
4	<keras.layers.convolutional.Conv2D object at 0...	block2_conv1	False
5	<keras.layers.convolutional.Conv2D object at 0...	block2_conv2	False
6	<keras.layers.pooling.MaxPooling2D object at 0...	block2_pool	False
7	<keras.layers.convolutional.Conv2D object at 0...	block3_conv1	False
8	<keras.layers.convolutional.Conv2D object at 0...	block3_conv2	False
9	<keras.layers.convolutional.Conv2D object at 0...	block3_conv3	False
10	<keras.layers.convolutional.Conv2D object at 0...	block3_conv4	False
11	<keras.layers.pooling.MaxPooling2D object at 0...	block3_pool	False
12	<keras.layers.convolutional.Conv2D object at 0...	block4_conv1	False
13	<keras.layers.convolutional.Conv2D object at 0...	block4_conv2	False
14	<keras.layers.convolutional.Conv2D object at 0...	block4_conv3	True
15	<keras.layers.convolutional.Conv2D object at 0...	block4_conv4	True
16	<keras.layers.pooling.MaxPooling2D object at 0...	block4_pool	True
17	<keras.layers.convolutional.Conv2D object at 0...	block5_conv1	True
18	<keras.layers.convolutional.Conv2D object at 0...	block5_conv2	True
19	<keras.layers.convolutional.Conv2D object at 0...	block5_conv3	True
20	<keras.layers.convolutional.Conv2D object at 0...	block5_conv4	True
21	<keras.layers.pooling.MaxPooling2D object at 0...	block5_pool	True
22	<keras.engine.sequential.Sequential object at ...	sequential_1	True

Step 3: We experimented with two different optimizer and learning rate ranging from 0.1 to 0.00001. Different learning rate techniques like Cyclic LR, SGD Warm restart , AdamW and SGDR was experimented. SGD optimizer with Cyclic LR set base learning rate of 0.0001 and maximum learning rate of 0.0009 is found to give the best results. EarlyStopping is implemented to stop the training if the validation loss does not show any improvement in 30 epochs (patience is set to 30). The best model is saved by monitoring the validation accuracy.

```
from keras.optimizers import *
earlystop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=30)
best_model_vgg19_wa5 = ModelCheckpoint('best_model_vgg19_wa5.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
#optimizer = optimizers.Adam(lr=0.1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
#optimizer = optimizers.Adam()
#optimizer = optimizers.RMSprop()
optimizer = optimizers.SGD()
#optimizer = SGDW(lr=0.0001, momentum=0., decay=0., weight_decay=1e-4, nesterov=False)
#optimizer = AdamW(lr=0.0001, beta_1=0.9, beta_2=0.999, weight_decay=1e-4, epsilon=1e-8, decay=0.)
clr = CyclicLR(mode='triangular2')
#sgdr = SGDRScheduler(min_lr=0.0001, max_lr = 0.0009,
#                     steps_per_epoch = np.ceil(train_generator.n/batch_size),
#                     lr_decay = 0.9,
#                     cycle_length=10,
#                     mult_factor = 2)

#optimizer = tf.contrib.opt.AdamWOptimizer(weight_decay=0.0)
Model_vgg19_WA5.compile(optimizer = optimizer , loss = "categorical_crossentropy", metrics=["accuracy"])
#learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
#                                            patience=20,
#                                            verbose=1,
#                                            factor=0.5,
#                                            min_lr=0.0001)
```

The code for implementing the cyclicLR is inspired from paper - https://arxiv.org/abs/1506.01186

Code reference - https://github.com/keras-team/keras-contrib/tree/master/keras_contrib/callbacks

The code for implementing the SGDR is inspired from the paper - http://arxiv.org/abs/1608.03983

Code reference - https://gist.github.com/jeremyjordan/5a222e04bb78c242f5763ad40626c452

The code for implementing SGDW and AdamW is inspired from the paper - https://arxiv.org/abs/1711.05101

Code reference - https://github.com/shaoanlu/AdamW-and-SGDW

Step 4: Train the model with fit generator.

```
Model1_VGG19_WA2=Model_vgg19_WA2.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.n//batch_size,
    epochs=300,
    validation_data=validation_generator,
    validation_steps=validation_generator.n//batch_size,
    callbacks = [earlystop,best_model_vgg19_wa2,clr])
```

```
60/60 [==============================] - 18s 293ms/step - loss: 1.1389 - acc: 0.5771 - val_loss: 1.1998 - val_acc: 0.6000

Epoch 00048: val_acc improved from 0.59583 to 0.60000, saving model to best_model_vgg19_wa2.h5
Epoch 49/300
60/60 [==============================] - 17s 290ms/step - loss: 1.0816 - acc: 0.6115 - val_loss: 1.1968 - val_acc: 0.5625
```

Step 5: The best model is saved for evaluation .

```
saved_model_baseModel_Vgg19_WA5 = load_model('best_model_vgg19_wa5.h5')
```

Step 6: Evaluate generator is used to get the training accuracy, validation accuracy and loss from the best model saved.

```
loss, accuracy = saved_model_baseModel_Vgg19_WA5.evaluate_generator(train_generator,train_generator.n//batch_size)
print("Training Accuracy: {:.4f}".format(accuracy))
print("Training Loss: {:.4f}".format(loss))
loss, accuracy = saved_model_baseModel_Vgg19_WA5.evaluate_generator(validation_generator,validation_generator.n//batch_size)
print("Validation Accuracy:  {:.4f}".format(accuracy))
print("Validation Loss:  {:.4f}".format(loss))
```

```
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:699: UserWarning: This ImageDataGenera
tor specifies `featurewise_center`, but it hasn't been fit on any training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:718: UserWarning: This ImageDataGenera
tor specifies `zca_whitening`, but it hasn't been fit on any training data. Fit it first by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '
```

```
Training Accuracy: 0.8313
Training Loss: 0.4842
Validation Accuracy:  0.8000
Validation Loss:  0.7535
```

Step 7: Predict generator is used to predict the class for the test dataset.

```
batch_size = 1
test_datagen = ImageDataGenerator(rescale=1. / 255)

test_generator = test_datagen.flow_from_directory(
    directory=test_set,
    target_size=(img_height, img_width),
    color_mode="rgb",
    batch_size=batch_size,
    class_mode=None,
    shuffle=False
)

test_generator.reset()
```

```
Found 240 images belonging to 6 classes.
```

```
pred=saved_model_baseModel_Vgg19_WA5.predict_generator(test_generator,verbose=1,steps=test_generator.samples/batch_size)
```

```
240/240 [==============================] - 8s 35ms/step
```

```
predicted_class_indices=np.argmax(pred,axis=1)
```

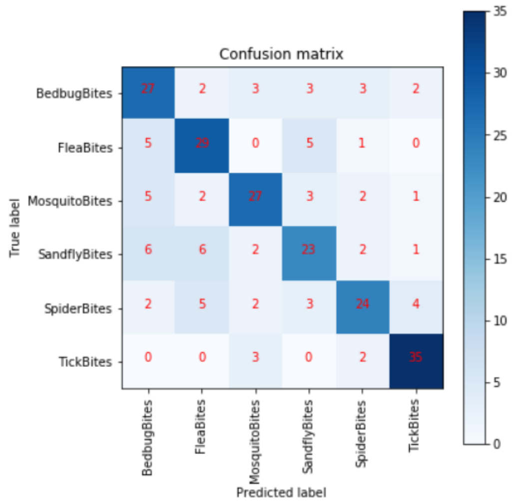
```
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]

predict_class_Label = lambda l :[0 if predictions=='BedbugBites' else 1 if predictions=='FleaBites' else 2 if predictions == 'Mo
pred_class_label = predict_class_Label(predictions)
```

```
filenames=test_generator.filenames
class_labels = list(test_generator.class_indices.keys())
true_classes = test_generator.classes
results=pd.DataFrame({"Filename":filenames,
                      "True Class":true_classes,
                      "Predict Class":pred_class_label,
                      "Predictions":predictions})
results.head(5)
```

	Filename	Predict Class	Predictions	True Class
0	BedbugBites/000023 resized resize224.jpg	0	BedbugBites	0
1	BedbugBites/000038 resized resize224.jpg	0	BedbugBites	0
2	BedbugBites/000041 resized resize224.jpg	5	TickBites	0
3	BedbugBites/000042 resized resize224.jpg	0	BedbugBites	0
4	BedbugBites/000045 resized resize224.jpg	0	BedbugBites	0

Step 8: The metrics used for testing data is confusion matrix. The code is implemented to get the confusion matrix by providing the predicted labels and true labels. This gives the result of True Positive(TP), False Positive(FP), True Negative (TN), False Negative (FN).

```
def metrics_confusion_matrix(cmtx, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    fig = plt.figure(figsize=(6,6))
    plt.imshow(cmtx, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    if normalize:
        cmtx = cmtx.astype('float') / cmtx.sum(axis=1)[:, np.newaxis]

    for i, j in itertools.product(range(cmtx.shape[0]), range(cmtx.shape[1])):
        plt.text(j, i, cmtx[i, j],
                 horizontalalignment="center",
                 color="red" )

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# confusion matrix
confusionMatrix = confusion_matrix(true_classes, pred_class_label)

# plot the confusion matrix
metrics_confusion_matrix(confusionMatrix, classes = class_labels)
```



Step 9: This code is implemented to get the values for Precision,Recall,F1-Score

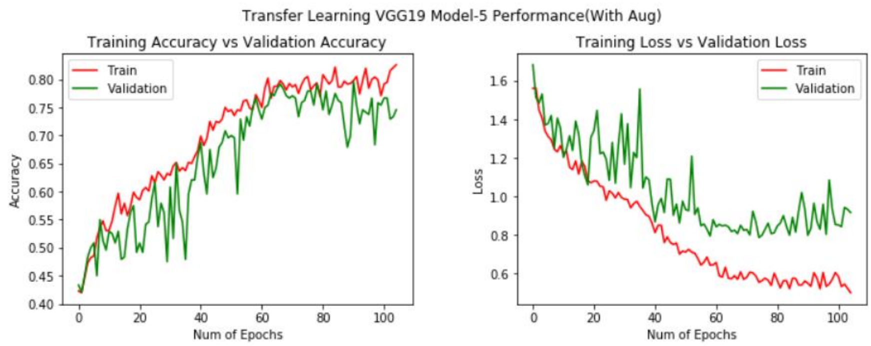
```
class_labels = list(test_generator.class_indices.keys())
report = metrics.classification_report(true_classes, pred_class_label, target_names=class_labels)
print(report)
```

```
              precision    recall  f1-score   support

  BedbugBites       0.60      0.68      0.64        40
    FleaBites       0.66      0.72      0.69        40
MosquitoBites       0.73      0.68      0.70        40
 SandflyBites       0.62      0.57      0.60        40
  SpiderBites       0.71      0.60      0.65        40
    TickBites       0.81      0.88      0.84        40

    micro avg       0.69      0.69      0.69       240
    macro avg       0.69      0.69      0.69       240
 weighted avg       0.69      0.69      0.69       240
```

Step 10: Code snippet to plot the Training Accuracy Vs Validation Accuracy and Training Loss Vs Validation Loss.

```
# summarize history for accuracy
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
t = fig.suptitle('Transfer Learning VGG19 Model-5 Performance(With Aug)', fontsize=12)
fig.subplots_adjust(top=0.85, wspace=0.3)

#plt.figure(0)
ax1.plot(Model1_VGG19_WA5.history['acc'],'r')
ax1.plot(Model1_VGG19_WA5.history['val_acc'],'g')
ax1.set_xlabel("Num of Epochs")
ax1.set_ylabel("Accuracy")
ax1.set_title("Training Accuracy vs Validation Accuracy")
ax1.legend(['Train','Validation'])

#ax2.figure(1)
ax2.plot(Model1_VGG19_WA5.history['loss'],'r')
ax2.plot(Model1_VGG19_WA5.history['val_loss'],'g')
ax2.set_xlabel("Num of Epochs")
ax2.set_ylabel("Loss")
ax2.set_title("Training Loss vs Validation Loss")
ax2.legend(['Train','Validation'])
```

```
<matplotlib.legend.Legend at 0x7fe08a6fb4a8>
```



The above code can be used for VGG-16 as well.

6. Implementation of Advanced Augmentation technique.

Apart from the augmentation parameters provided by ImageDataGenerator we also experimented by adding custom functions to improve the contrast in an image using AEH,CLAHE and smoothing the image by removing noise using Gaussian blur bilateral filter .

Inspired from the paper –

Ganesh, V. R. (1), & Ramesh, H. (2). (n.d.). Effectiveness of contrast limited adaptive histogram equalization technique on multispectral satellite imagery. ACM International Conference Proceeding Series, 234–239. https://doi.org/10.1145/3177404.3177409
Goyal, A., Bijalwan, A., & Chowdhury, M. K. (2012). A comprehensive review of image smoothing techniques. International Journal of Advanced Research in Computer Engineering & Technology, 1(4), 315-319.

```
# Histogram equalization
def AHE(img):
    img_eq = exposure.equalize_hist(img)
    return img_eq

# Adaptive histogram equalization
def CLAHE(img):
    img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
    return img_adapteq

def GaussianBlur(img):
    img_gausblur = cv2.GaussianBlur(img,(5,5),0)
    return img_gausblur

def bilateralFilter(img):
    img_bilateralfilter = cv2.bilateralFilter(img,9,75,75)
    return img_bilateralfilter
```

```
img_width, img_height = 224, 224
batch_size = 16
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    featurewise_center=True,
    samplewise_center=False,
    featurewise_std_normalization=True,
    samplewise_std_normalization=False,
    rotation_range=90,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    preprocessing_function = bilateralFilter,
    zca_whitening=True,
    horizontal_flip=True,
    vertical_flip=True)

train_generator = train_datagen.flow_from_directory(
    train_set,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

valid_datagen = ImageDataGenerator(rescale=1. / 255)

validation_generator = valid_datagen.flow_from_directory(
    valid_set,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')
```

```
## AHE:
#Training Accuracy: 0.1667
#Training Loss: 11.3102
#Validation Accuracy:  0.2208
#Validation Loss:  9.3558

# Gaussian Blur
#Training Accuracy: 0.7271
#Training Loss: 0.7521
#Validation Accuracy:  0.5375
#Validation Loss:  1.3320

#Bilateral Filter
#Training Accuracy: 0.5375
#Training Loss: 1.2259
#Validation Accuracy:  0.4875
#Validation Loss:  1.5286
```