Diabetic Retinopathy Detection

BALACHANDAR JEGANATHAN
Colorado State University

Abstract- Diabetic Retinopathy (DR) is a diabetic-induced eye disorder that can lead to vision loss or even total blindness. For people aged 20 to 64 years, diabetic retinopathy accounts for 12 percent of all new cases of blindness in the United States and is the leading cause of blindness. If caught early enough, progression to vision impairment may be delayed if not fully prevented, but this is often difficult because symptoms can occur too late to provide successful care. It has been estimated that diabetic retinopathy affects about 93 million people worldwide, though only half are aware of it. Diabetic Retinopathy has four major stages; dysfunctional blood vessels propagate on the surface of the retina at its most advanced level, which can lead to scarring and loss of cells in the retina. This paper explains how we are going to identify whether a patient is affected by Diabetic Retinopathy (DR) or not by using their retina image. This paper describes the step-by-step process to build the model, it includes the Image dataset and pre-processing, split the training and test data set, train the model, and validate the model.

I. IMAGE DATASET AND PRE-PROCESSING

A dataset of retina images from a recent Kaggle contest is being used. These are a collection of high-resolution retina pictures, including various cameras, colors, lighting, and orientations, taken in a variety of conditions. For each person, along with a DR classification diagnosed by a clinician, we have a picture of their left and right eye. Because of these varying conditions, there is significant noise and variance in the data collection. There are four stages of Diabetic Retinopathy, the level indicates the severity of the disease.

TABLE. Diagnosing Diabetic Retinopathy

DIABETIC RETINOPATHY LEVEL	RETINAL FINDINGS
Mild NPDR	MAs only
Moderate NPDR	At least one hemorrhage or MA and/or at least one of the following: • Retinal hemorrhages • Hard exudates • Cotton wool spots • Venous beading
Severe NPDR	Any of the following but no signs of PDR (4-2-1 rule): • > 20 intraretinal hemorrhages in each of four quadrants • Definite venous beading in two or more quadrants • Prominent IRMA in one or more quadrants
PDR	One of either: • Neovascularization • Vitreous/preretinal hemorrhage

Abbreviations: IRMA, intraretinal microvascular abnormality; MA, microaneurysm; NPDR, nonproliferative diabetic retinopathy; PDR, proliferative diabetic retinopathy

- Diabetic Retinopathy Detection Workflow

1. Data Preparation
   a. Download Images from data source
   b. Decode Images and prepare a dataset
   c. Generate Augmented Images
2. Model Training
   a. Load Images from Pickle Files
   b. Prepare Training and Test Samples
   c. Build Convolutional Neural Network
   d. CNN Model Training
   e. Visualize Accuracy and Loss
   f. Save model weights and training history
3. Model Validation
   a. Load trained model weights
   b. Prepare Validation Samples
   c. Perform Sample Predictions
   d. Visualize Confusion Matrix
   e. Display Accuracy Details

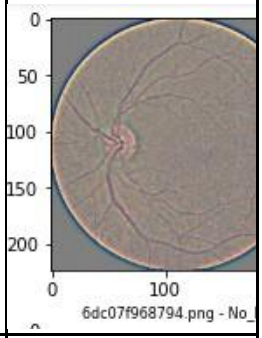
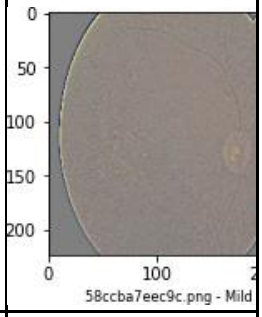
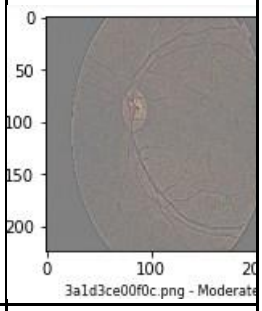
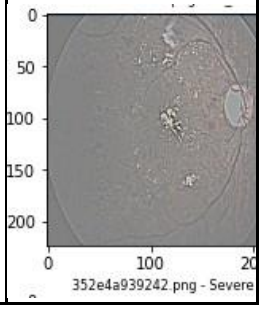
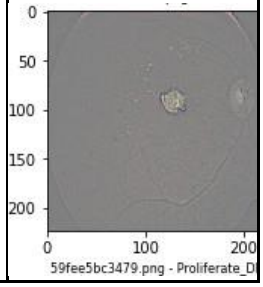
II. DATA PREPARATION

- Download Images from data source:

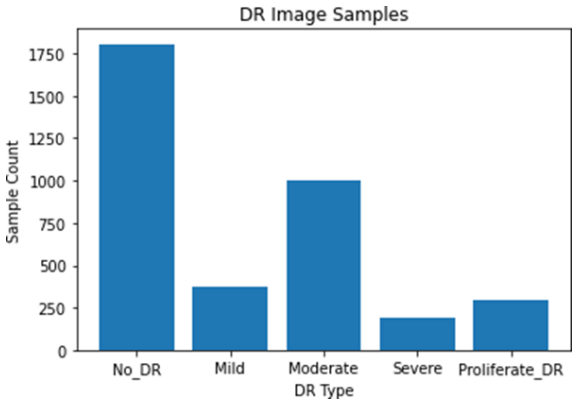
The images are downloaded from Kaggle, these images are Gaussian filtered retina scan images to

detect the diabetic retinopathy. The dimension of these images is 244 x 244 x 3 pixels, these can be used with many pre-trained deep learning models.

Source: https://www.kaggle.com/sovitrath/diabetic-retinopathy-224x224-gaussian-filtered

umber of Samples	Diabetic Retinopathy Level	Sample Image
1805	0 - No DR	
370	1 - Mild	
999	2 - Moderate	
193	3- Severe	
295	4 - Proliferate DR	

Sample Image Distribution:

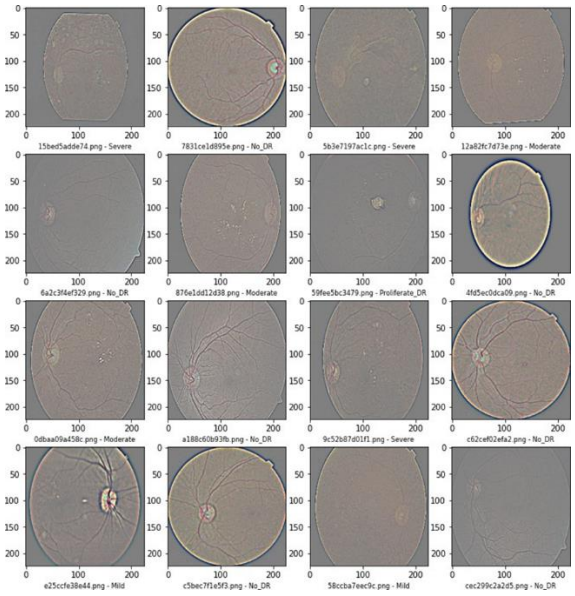


Decode Images and prepare a dataset:

The training images are 'PNG' format, it needs to be converted as a matrix (ex: 224 x 224 x 3) to use them as an input for convolutional neural networks.

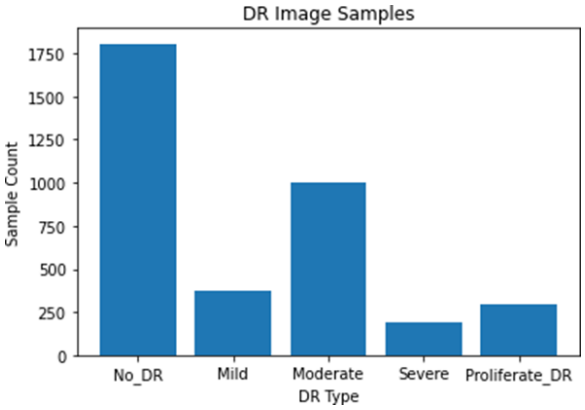
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd from PIL import
Image from glob import glob import os
import pickle

# Image file directory
file_directory="/content/drive/My
Drive/CSUG/Foundations_of_Artificial_Intelligence/So
urceCode/FinalProjec t/Gaussian_Giltered_Images/"

# Decode Image to flot32 type def
decode_png_image(file_path):
img = tf.io.read_file(file_path)
# convert the compressed string to a
3D uint8 tensor img =
tf.image.decode_png(img,
```

```
channels=3)
# Use `convert_image_dtype` to
convert to floats in the [0,1]
range. img =
tf.image.convert_image_dtype(img,
tf.float32)
return img

# Read Images from the Image folder
and create dictionary def
get_dataset_dict():

category_map={0:"No_DR",1:"Mild",2:"
Moderate",3:"Severe",4:"Proliferate_
DR"}
dataset_dict={'FileName':[],
'ImageData':[],'Level':[],'Label':[]
}

for level in category_map:
file_directory_category=file_directory +
category_map[level] + "/"
files=glob(file_directory_category + '/*.png',
recursive=True)
for file in files:
img_data=decode_png_image(file)
file_name = os.path.basename(file)
dataset_dict['FileName'].append(file
_name)
dataset_dict['ImageData'].append(img
_data)
dataset_dict['Level'].append(level)
dataset_dict['Label'].append(categor
y_map[level])
return dataset_dict

# Save Dataset into a Pickle file
dataset_dict = get_dataset_dict()
dataDF=pd.DataFrame(dataset_dict)
dataDF.to_pickle(file_directory+"Dataset.pickle")

# Read Pickle file and Create the dataset
dataset_file=file_directory+'Dataset.pickle' dataset =
pd.read_pickle(dataset_file)

# Plot Images
def
PlotImagesFromList(images,labels,fil
eNames):

n=len(labels)
img_size=images[0].shape
resized_width=int(img_size[0]/5)
resized_height=int(img_size[1]/5)
fig=plt.figure(figsize=(12, 16))
columns = 4
rows = 5
for i in range(1, n +1):
fig.add_subplot(rows, columns, i)
plt.imshow(images[i-1])
xlabel= str(fileNames[i-1]) + " - " + str(labels[i-1])
plt.xlabel(xlabel,fontsize=8)
plt.show()

# Display Sample Images
samples=dataset.sample(16)
PlotImagesFromList(list(samples['ImageData']),lis
t(samples['Label']),lis t(samples['FileName']))
```



```
# Sample distribution
def
show_sample_distrubution(df:pd.Data
Frame):
category_map={0:"No_DR",1:"Mild",2:"
Moderate",3:"Severe",4:"Proliferate_
DR"}
data_count=df.Level.value_counts()
labels = [category_map[i] for i in
```

```
data_count.index]
x_axis=data_count.index
y_axis=data_count.values fig, ax = plt.subplots() ax.bar(x_axis,y_axis)
ax.set_xlabel("DR Type")
ax.set_ylabel("Sample Count") ax.set_title("DR Image Samples") ax.set_xticks(x_axis)
ax.set_xticklabels(labels) plt.show()

show_sample_distrubution(dataset)
```



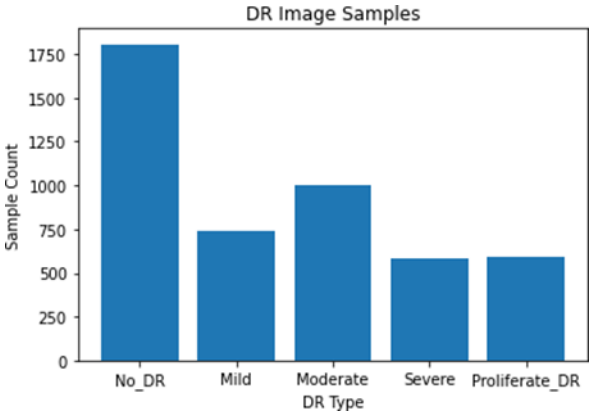
Generate Augmented Images:

Image augmentation is a technique that helps increase the diversity of data available for training models, without acquiring new data. Image augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to generate more training data.

```
# Generate Augmented images
def generate_augmented_images(df:pd.DataFrame,level:int,label:str,number_of_images:int):
filtered_images = df[df['Level']==level]
total_filtered_images=len(filtered_images)

number_of_agumented_images_per_image=number_of_images//total_filtered_images

augmented_images=[] # Augmentation
data_augmentation = tf.keras.Sequential([
tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),])

for index, row in filtered_images.iterrows():
for i in range(number_of_agumented_images_per_image): img = row["ImageData"]
augmented_image = data_augmentation(tf.expand_dims(img, 0))
augmented_images.append(augmented_image[0])

levels=[level for i in range(len(augmented_images))]
labels=[label for i in range(len(augmented_images))]
file_names=['Augmented ' + label for i in
range(len(augmented_images))]

result_df=pd.DataFrame({'ImageData':augmented_images,'Level':levels,'Label':labels,'FileName':file_names})
return result_df
category_map={0:"No_DR",1:"Mild",2:"Moderate",3:"Severe",4:"Proliferate_ DR"}
# Create ~500 augmented images for Mild Category
mild_images = generate_augmented_images(dataset,1,'Mild',500)

# Create ~500 augmented images for Moderate Category
moderate_images = generate_augmented_images(dataset,2,'Moderate',500)

# Create ~500 augmented images for Severe Category
severe_images = generate_augmented_images(dataset,3,'Severe',500)

# Create ~500 augmented images for
```

```
Proliferate_DR Category
Proliferate_DR_images =
generate_augmented_images(dataset,4,
'Proliferate_DR',500)

# Combine sample Images and Augmented
Images
final_dataset =
dataset.append(mild_images,ignore_index=True)
final_dataset =
final_dataset.append(moderate_images,ignore_inde
x=True) final_dataset =
final_dataset.append(severe_images,ignore_index=T
rue) final_dataset =
final_dataset.append(Proliferate_DR_images,ignore_i
ndex=True)
```

show_sample_distrubution(final_dataset)




```
# Save Sample Images + Augmented
Images into a Pickle file

mild_images.to_pickle(file_director
y+"Augmented_Dataset_Mild.pickle")
moderate_images.to_pickle(file_dire
ctory+"Augmented_Dataset_Moderate.p
ic kle")
severe_images.to_pickle(file_direct
ory+"Augmented_Dataset_Severe.pickl
e"
)

Proliferate_DR_images.to_pickle(file_directory+"Aug
mented_Dataset_Prolif erate_DR.pickle")
```

III. MODEL TRAINING

Load Images from Pickle Files:

There are around 4500 training images, it would be a very expensive operation for reading each file to make it ready for the model training. So, the pickle file contains multiple images and it helps to reduce the multiple file access.

```
# Read Pickle file and
Create the dataset
dataset_file=file_directory
+
'Gaussian_Giltered_Images/'
original_dataset =
pd.read_pickle(dataset_file
+ 'Dataset.pickle')
augmented_mild_images=pd.read_pickl
e(dataset_file+"Augmented_Dataset_M
il d.pickle")
augmented_moderate_images=pd.read_p
ickle(dataset_file+"Augmented_Datas
et
_Moderate.pickle")
augmented_severe_images=pd.read_pic
kle(dataset_file+"Augmented_Dataset
_S evere.pickle")
augmented_Proliferate_DR_images=
pd.read_pickle(dataset_file+"Augmented_Dataset_Pr
oliferate_DR.pickle")

# Combine Training Images and
Augmented Images dataset =
original_dataset.append(augmented_mild_images,ign
ore_index=True) dataset =
dataset.append(augmented_moderate_images,ignor
e_index=True) dataset =
dataset.append(augmented_severe_images,ignore_i
ndex=True) dataset =
dataset.append(augmented_Proliferate_DR_images,i
gnore_index=True)
```

Prepare Training and Test Samples:

```
# Import Packages import tensorflow
```

```
as tf
from sklearn.model_selection import train_test_split from keras.models import Sequential
from tensorflow.keras import layers
from tensorflow import keras
from sklearn import preprocessing

# Prepare training and testing dataset X=dataset.ImageData
Y=dataset.Level
X =np.array([X[i] for i in range(len(X))]) Y =np.array([Y[i] for i in range(len(Y))]) lb = preprocessing.LabelBinarizer()
lb.fit(Y)
lb.classes_ Y=lb.transform(Y)
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.25, random_state=0, stratify=Y)
```

- Build Convolutional Neural Network:

In machine learning, the word "convolution" is also used to refer to either a convolutional operation or a convolutional layer. A typical convolutional neural network consists of one or more of the layers such as COnvolutional layers, pooling layers and Dense layers. The CNN would be the great choice for image classification problem.

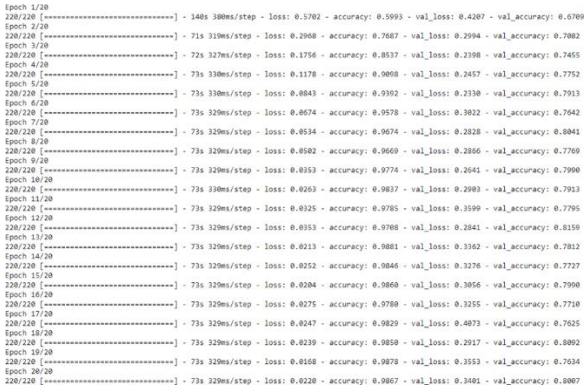
```
# Build CNN model
def build_model(backbone, lr=1e-4):
model = Sequential()
model.add(backbone)
model.add(tf.keras.layers.GlobalAveragePooling2D())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(5, activation='softmax'))

model.compile(
loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(lr=lr), metrics=['accuracy']
)
return model

# Use Pretrained Imagenet (DenseNet201) resnet =tf.keras.applications.DenseNet201(
weights='imagenet', include_top=False, input_shape=(224,224,3)

# Build CNN model
model = build_model(resnet ,lr = 1e-4)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet201 (Functional)     (None, 7, 7, 1920)        18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dropout (Dropout)            (None, 1920)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 1920)              7680
_________________________________________________________________
dense (Dense)                (None, 5)                 9605
=================================================================
Total params: 18,339,269
Trainable params: 18,106,373
Non-trainable params: 232,896
_________________________________________________________________
```

- CNN Model Training:

The model training is one of the ML development lifecycles in which practitioners try to match the best combination of weights and bias to a machine learning algorithm in order to minimize a loss function over the prediction range.

```
# Training data generator
train_generator = tf.keras.preprocessing.image.ImageDataGenerator( zoom_range=2, # set range for random zoom
rotation_range = 90,
horizontal_flip=True, # randomly flip images
vertical_flip=True, # randomly flip images
)
# Set Batch size BATCH_SIZE = 16

# Train Model
history = model.fit(X_train, Y_train,
steps_per_epoch=X_train.shape[0] / BATCH_SIZE, epochs=20,
validation_data=(X_test, Y_test))
```

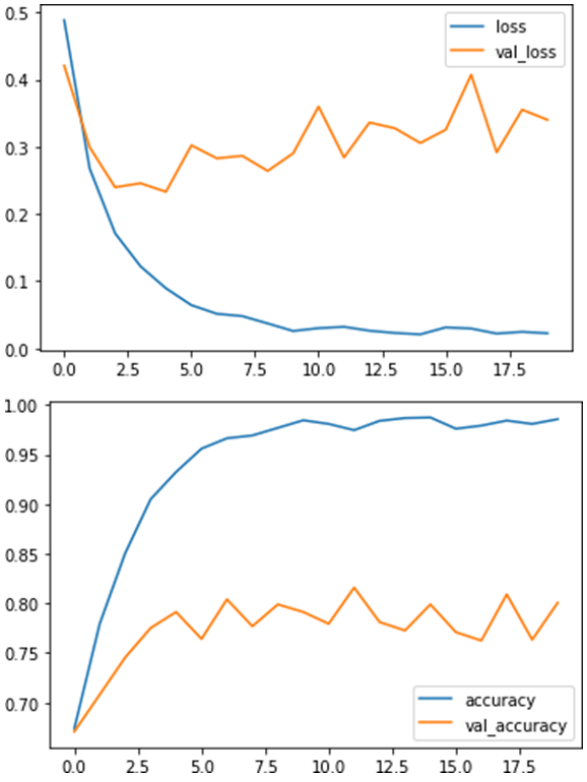


Visualize Accuracy and Loss:

The "history" object is returned from the model.fit() operation, which holds the accuracy and loss information, this information would be useful to understand the training progress and troubleshooting.

```
# Visualize Loss and Accuracy
history_df = pd.DataFrame(history.history)
history_df[['loss', 'val_loss']].plot()
```

```
history_df = pd.DataFrame(history.history)
history_df[['accuracy','val_accuracy']].plot()
```



- Save model weights and training history:

After completing the training, the model object holds the calculated weights, these weights can be used for further training progress and can be used for other classification problems.

```
# Save model weights
model.save_weights(dataset_file + "weights.best.hdf5")

# Save training history
history_df=pd.DataFrame(history.history)
history_df.to_csv(dataset_file + "history.best.csv")
```

IV. MODEL VALIDATION

- Load trained model weights:

If we want to add new training data later, we no need to perform model training for the entire training dataset, we can load the calculated weights to continue the training using new dataset.

```
# Read Pickle file and Create the dataset
dataset_file=file_directory + 'Gaussian_Giltered_Images/'
dataset = pd.read_pickle(dataset_file + 'Dataset.pickle')

# Convolutional Neural Network Model
def build_model(backbone, lr=1e-4):
model = Sequential()
model.add(backbone)
model.add(tf.keras.layers.GlobalAveragePooling2D())
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dense(5, activation='softmax'))

model.compile(
loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(lr=lr), metrics=['accuracy']
)
return model

# Use pretrained model (DenseNet201)
```

```
resnet
=tf.keras.applications.DenseNet201(
weights='imagenet',
include_top=False,
input_shape=(224,224,3)
)

# Load Model
best_model= build_model(resnet ,lr =
1e-4)

# Load Trained Weights
best_model.load_weights(dataset_file +
"weights.best.hdf5")
```

- Prepare Validation Samples:

Preparing validation samples is an important task, these samples are used to validate the trained model. We are using 500 samples from the different classes (Categories) for model validation.

```
# Generate 500 samples form original
dataset for validation
samples=dataset.sample(500)
images=np.array([img for img in samples.ImageData])
```

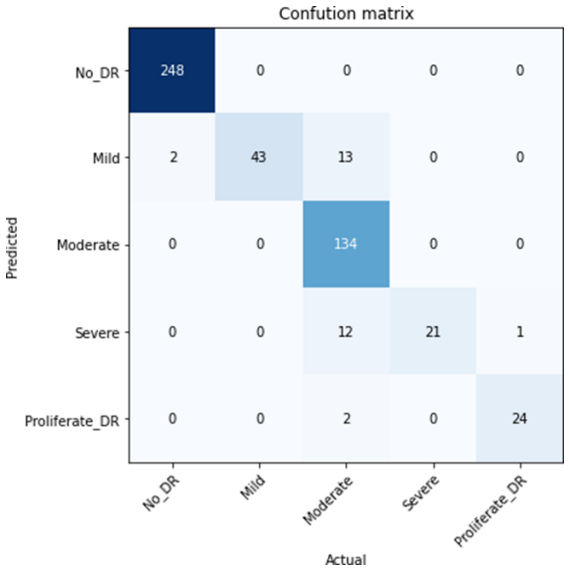
- Perform Sample Predictions:

The validation process helps to analyze the model performance, we can compare the predicted values with actual.

- Visualize Confusion Matrix:

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. The confusion matrix itself is straightforward, but the associated terms may be confusing.

```
# Plot Confusion Matrix
def plot_confusion_matrix(cMatrix,
classes,
title=None, cmap=plt.cm.Blues):

#This function plots the confusion
matrix.

cMatrix = np.array(cMatrix)
fig, ax =
plt.subplots(figsize=(6,6))
ax.imshow(cMatrix, cmap=cmap)
ax.set(xticks=np.arange(cMatrix.shap
e[1]),
yticks=np.arange(cMatrix.shape[0]),
# x & y labels with classes
xticklabels=classes,
yticklabels=classes, title=title,
ylabel='Predicted', xlabel='Actual')

# Rotate the tick labels and set
their alignment.
plt.setp(ax.get_xticklabels(),
rotation=45, ha="right",
rotation_mode="anchor")

# Loop over data dimensions and
create text annotations. thresh =
cMatrix.max() / 2.
for i in range(cMatrix.shape[0]):
for j in range(cMatrix.shape[1]):
ax.text(j, i, format(cMatrix[i, j], 'd'),
ha="center", va="center",
color="white" if cMatrix[i, j] >
thresh else
"black")
return ax

# Create Confusion Matrix
cMatrix=tf.math.confusion_matrix(actual,predictions)
plot_confusion_matrix(cMatrix,category_map.values
(),'Confusion matrix')
```



Display Accuracy Details:

There are 500 samples used for validation, the final accuracy was 94%, still the model can be improved using the trained model weights and adding new datasets.

```
# Generate Accuracy Report from sklearn import metrics
print("Accuracy Score : {0}".format(metrics.accuracy_score(actual,predictions)))
print("Precision Score : {0}".format(metrics.precision_score(actual,predictions,average='weighted')))
print("F1 Score : {0}".format(metrics.f1_score(actual,predictions,average='weighted')))
```

```
Acuracy Score    : 0.94
Precision Score  : 0.9490079006211178
F1 Socre         : 0.9371199449947756
```

CONCLUSION

The Diabetic Retinopathy is an eye disorder that can lead to vision loss. This paper explained the entire ML development lifecycle to predict the DR level, this prediction would help people to get the higher accuracy, as a result they can get an appropriate treatment and avoid vision loss. We started from downloading the training dataset, then we cleaned and organized the dataset, the organized training data fed into the CNN model. Also this paper explained how the trained model weights are stored and reused for further, the validation result shows that we would get 94% accuracy when we use this trained model, this higher accuracy gives more confidence in DR level prediction.

REFERENCES

[1] "Case Study: TensorFlow in Medicine - Retinal Imaging (TensorFlow Dev Summit 2017)." *YouTube*, uploaded by Google Developers, 16 Feb. 2017, www.youtube.com/watch?v=oOeZ7IgEN4o.

[2] "Diabetic Retinopathy (Resized)." *Kaggle*, Kaggle, www.kaggle.com/tanlikesmath/diabetic-retinopathy-resized. Accessed 2019.

[3] Felts, John W. "Deep CNNs for Diabetic Retinopathy Detection." *Stanford University*, vol. 3, no. 2, 1999, pp. 25–26. *Crossref*, doi:10.1108/err.1999.3.2.25.24.

[4] "The Four Stages of Diabetic Retinopathy." *Modern Optometry*, modernod.com/articles/2019-june/the-four-stages-of-diabeticretinopathy?c4src=article:infinite-scroll. Accessed 22 Feb. 2021.

[5] Bora, A., Balasubramanian, S., & Babenko, B. (2020, November 26). *The Lancet Digital Health*.

[6] Https://Www.Thelancet.Com/Journals/Landig/Article/PIIS2589-7500(20)30 250-8/Fulltext. https://secure.jbs.elsevierhealth.com/action/cookieAbsent