ENEL 610: Final Project Report

Intelligent Ocular Disease Recognition

by

Aditya Porwal

Sparsh Mehta

Submitted to:

Department of Electrical and Computer Engineering

Schulich School of Engineering

University of Calgary

Calgary, Alberta

1 Introduction and Objectives

Sight is the most important sense out of the five human senses. An eye can suffer a number of diseases such as Glaucoma, Myopia, Cataract, etc., depending upon several factors like age, sex or even other diseases like Diabetes, which can lead to partial or permanent blindness [1, 2, 3]. According to the World Health Organization (WHO), globally, 1 billion people have a vision impairment that could have been prevented or has yet to be addressed. This includes Cataract (65.2 million) and Glaucoma (6.9 million) among others [4]. Thus, it is evident that the early detection of Ocular diseases would have a great impact in preventing or at least reducing the severeness of eye damage.

In addition to this, the detection of eye diseases is a challenging task for Ophthalmologists. Even with support systems like computer-aided diagnosis (CADx), it can be costly if not applied to large populations [5]. Hence, a deep-learning based approach where features are automatically extracted, selected and classified without the need of domain-specific expert knowledge has great significance. This not only addresses the cost issue for expensive detection, but also cuts down on the requirement of time and professional trained personnel that might not be available in many cases.

In this project of ocular disease recognition, we have developed our own CNN model from scratch, and 3 other models (VGG19, InceptionV3, Xception) using transfer learning techniques. We compare and contrast their performance on metrics such as- Precision, Recall, F1 score etc and finally find the best suitable model giving an acceptable classification accuracy for this problem.

2 Dataset

The dataset used to train our models was generously provided by Shanggong Medical Technology Co.Ltd. [10] in a an international competition on the 'Global Grand Challenge' website organised by Peking University (PKU). The database, named as Ocular disease intelligent recognition (ODIR) database, is now publicly available on the online machine learning community- Kaggle [9]. It contains images of the Fundus of the left and right eyes of 5000 patients. Dataset is having approximately 7000 images with labels for training and 1000 images without labels for testing. Around 5000 pre-processed images are also provided, with the pre-processing techniques unknown.

3 Prior literature

This topic is very vast and yet to be explored. There are limited past studies where authors propose disease classification using modern deep-learning algorithms for ocular diseases recognition. For instance, Xiangyu Chen et al. (2015) [6] used a six layer architecture (four convolutional layers and two fully-connected layers with output fed to a soft-max classifier) for the detection of Glaucoma. Their model achieved area under Receiver Operating Characteristic (ROC) curve at 0.887.

Another work authored by Abbas Q., used an integrated approach of combining Convolutional Neural Network (CNN), Deep Belief Network (DBN) and the Softmax deep-learning classifiers [7]. They curated a dataset of 1200 images containing normal Fundi and Glaucoma diseased Fundi, and extracted the Regions of Interest (ROI)- Optic Disc (OD) and Optical Cup (CUP) for deep learning features' classification. They achieved an accuracy of 99% and precision of 84% but again concentrated their classification on one disease.

A study on Cataract detection [8] using Deep Convolutional Neural Network (DCNN) achieved 93.52% accuracy, working with only the G-filter (Green component) of RGB Fundus images, increasing the time-efficiency. They also experimented with the scalability of database effects on accuracy in DCNN classification and established that the accuracy continuously improved with the amount of available samples. As a Kaggle competition[12], a dataset for diabetic retinopathy detection was introduced and the winning solution proposed by Graham et al. used SparseConvNet CNN with min-pooling for their classification of Fundus images. Various image pre-processing and augmentation techniques were applied before providing the images as input to the architecture.

Various authors have developed deep learning models for the ODIR dataset. Islam et al. [13] implemented a CNN architecture which used the left and right eye Fundi individually to input to their model. They also considered the samples having a single label, this reduced the problem complexity and lead them to achieve an accuracy of 88% with AUC value of 80.5%. Another work [14] considered this as a multi-label and multi-class problem and compared various combinations of CNN architectures, and two approaches- considering left and right Fundus individually, and then in concatenated form. Their best model received the AUC and F1 score of around 85% and 85%, respectively.

4 Methodology

4.1 Exploratory Data Analysis

We are using an open source dataset on the Kaggle platform under the name "Ocular Disease Recognition; Right and left eye Fundus photographs of 5000 patients" [9, 10]. This is a structured ophthalmic database constructed using images of the Fundus of the eye. An eye Fundus is the interior surface of the eye behind the lens that includes the Retina, Optic Disc (OD), Macula, Fovea, and Posterior pole (see Figure 1). All these areas of interest play an important role in detecting diseases in the eye images. The dataset was created with the input from many domain experts who provided diagnostic keywords, and labeled the images with corresponding diseases. Information such as the age, gender of the patient is also given (Figure 2).

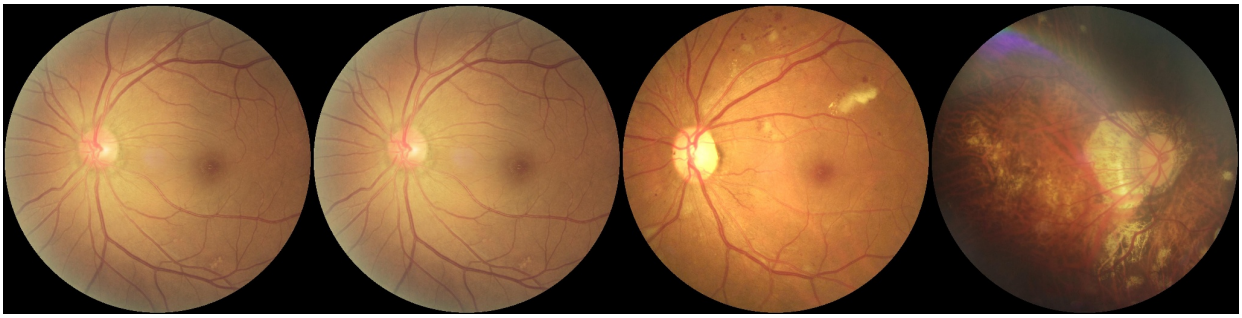


Figure 1: Eye Fundi with labels- a) Other, b) Diabetes, c) Diabetes, d) Cataract
(Note: The given dataset has 8 classes, of which our project focuses on 4)

Following is a brief introduction to the dataset:

- Train Images: 7000, Test Images : 1000 (no labels)
- Dataset includes images of both, the left and the right Fundus (with no NULL values present)
- Pre-processed Images (provided in the ODIR dataset): 5000 Images
- Eight classes: Normal, Diabetes, Glaucoma, Cataract, AMD, Hypertension, Myopia, and Others.

- Diagnostic Keywords are provided for each eye (Left and Right), along with the patient ID, age, sex and labels.
- A patient might be suffering from more than one disease. This was verified by plotting the correlation matrix during data exploration (Figure 3). Hence, a multi-label classification problem.

ID	Age	Sex	L_Fundus	R_Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	labels	filename
0	69	Female	0_left.jpg	0_right.jpg	cataract	normal fundus	['C']	0_left.jpg
5	50	Female	5_left.jpg	5_right.jpg	moderate non proliferative retinopathy	moderate non proliferative retinopathy	['D']	5_left.jpg
95	46	Male	95_left.jpg	95_right.jpg	suspected glaucoma	hypertensive retinopathy	['G']	95_left.jpg
102	73	Female	102_left.jpg	102_right.jpg	dry age-related macular degeneration	dry age-related macular degeneration	['A']	102_left.jpg
225	70	Female	225_left.jpg	225_right.jpg	pathological myopia	moderate non proliferative retinopathy	['M']	225_left.jpg
394	63	Male	394_left.jpg	394_right.jpg	normal fundus	normal fundus	['N']	394_left.jpg

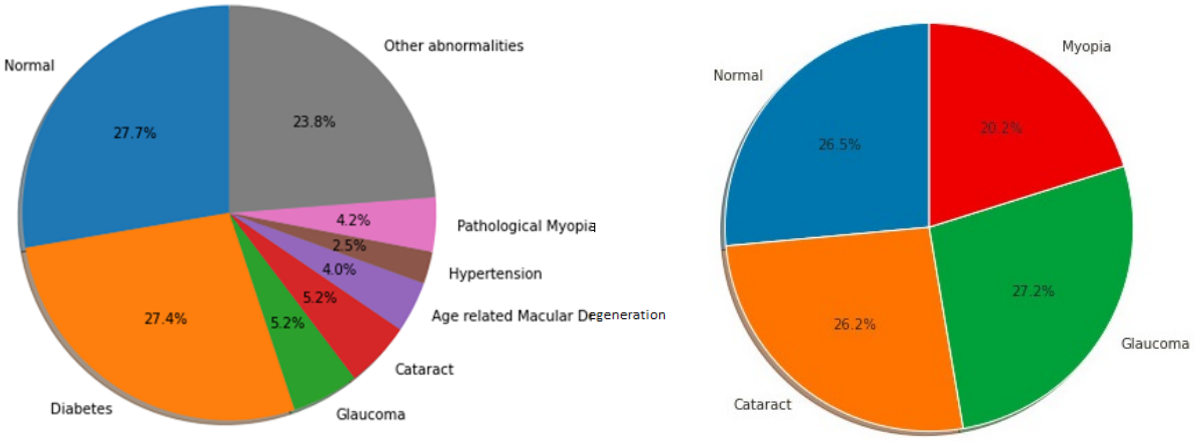
Figure 2: Six random rows from the dataset (Note: Diagnostic Keywords are given for the Left and Right Eye. Eyes are categorized under eight different classes.)

	N	D	G	C	A	H	M	O
N	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
D	0.000	1.000	0.149	0.184	0.098	0.437	0.109	0.309
G	0.000	0.028	1.000	0.014	0.067	0.078	0.040	0.050
C	0.000	0.035	0.014	1.000	0.000	0.019	0.000	0.032
A	0.000	0.014	0.051	0.000	1.000	0.039	0.017	0.016
H	0.000	0.040	0.037	0.009	0.024	1.000	0.000	0.012
M	0.000	0.017	0.033	0.000	0.018	0.000	1.000	0.045
O	0.000	0.269	0.228	0.146	0.098	0.117	0.253	1.000

Figure 3: Correlation matrix showing the intersection between various classes. For example, about 15% patients suffering from Glaucoma are also Diabetic.

4.2 Dataset Manipulation

The dataset we are using is highly imbalanced in terms of number of image samples per class of disease. For instance, normal Fundi constitute around 1140 images while for Myopia only 174 samples are provided (Figure 4a). In accordance with our final goal for this project, i.e. to classify ocular diseases into 4 categories- normal, Cataract, Glaucoma and Myopia, we extract corresponding images from the main dataset, creating a mini-dataset to feed into our model. This task was achieved using writing a python script, in which we checked all the diagnostic keywords of each and every image and shortlisted the ones which had 'cataract' mentioned in them. After that, all the labels were scanned for the label 'C' (corresponding to Cataract) and if one was found, it was stored in a data frame. This was repeated for the Normal, Glaucoma and Myopia classes as well, and finally concatenating the data frames a sub-dataset was created.



(a) Highly imbalanced ODIR dataset

(b) Our sub-dataset with similar sample distribution

Figure 4: Pie-Chart to visualize dataset distribution

This sub-dataset contains normal eye Fundus images (around 600), and images suffering from cataract (around 600), Glaucoma (around 600), and Myopia (around 450) (Figure 5). According to a published research [15], it is beneficial to use Contrast Limited Adaptive Histogram Equalisation (CLAHE) to bring out minute details of the Fundus in such datasets. We then convert the image to the 'LAB color format' (L- Lightness, A- color component ranging from Green to Magenta, B- color component ranging from Blue to Yellow [16]), and use CLAHE image processing technique on the L channel of the image, which resulted in images shown in 5.

This new dataset with processed images is then divided into the training (60%), validation (30%) and test (10%) sets, i.e. approx. 2040 images for the model development (Train and selection) and 230 images for model testing.

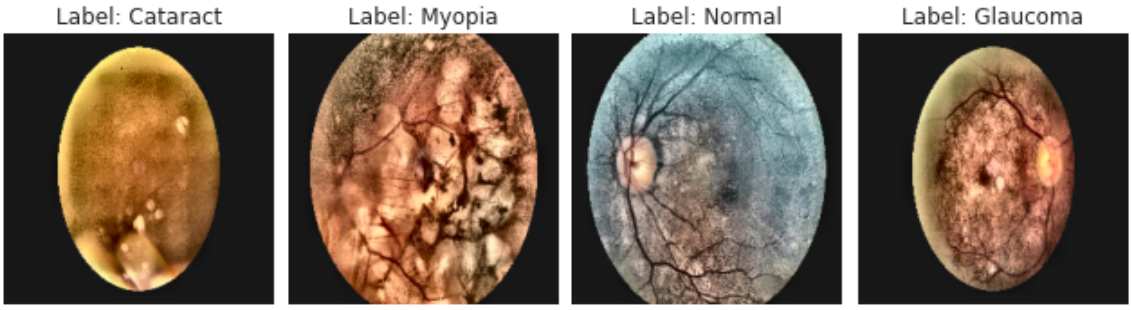


Figure 5: Four samples of Fundus of an eye (CLAHE processed) with their labels from the new sub-dataset we created.

We used Min-Max dataset normalization (normalization with mean-standard deviation was also tried but it gave slightly poor results) to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Min-max dataset normalization is done using the following formula:

$$X_{norm} = \frac{(X - X_{min})}{(X_{max} - X_{min})} \qquad \text{where X= train, validation and test samples}$$

The labels are represented using the one hot encoding, which is a type of categorical binary representation that maps the categorical data to numbers that machine learning algorithms can work with.

Deep learning models need as much data as possible to learn and then classify images. Considering our case of 4 classes and insufficient data, we employed Data Augmentation techniques such as rotating (rotation range- 15 degrees), and horizontal flipping of the images that addressed the problem of having a small dataset. Even after using these techniques, this much amount of data is not sufficient and at some point of time even data augmentation might start feeding redundant images to the model, thus, not letting the model learn new information. We avoided the use of other augmentations techniques to preserve the minutest details in the Fundus images that may affect the overall decision making if tampered. We used the batch size of 32 for this task. Data Augmentation was used on both- train and validation set.

4.3 Models and their Training

To achieve the best possible performance on our dataset, we tried 4 different models (CNN, VGG16, Inception, Xception) with many variations.

Key points (Remains same for all the models):

- Initial learning rate: $1e^{-5}$
- Learning rate for fine tuning: $1e^{-6}$
- Did not import original classification network for any of the models
- Same classification network and training parameters for transfer learning.

4.3.1 Convolutional Neural Network (CNN) model

Owing to the fact that our dataset is quite small in comparison to the ones needed to train deep learning models, we knew beforehand that training a deep neural network from scratch might not yield satisfactory results. Still, as a learning exercise, after experimenting we developed a model with the following configurations:

- Convolutional layers: 10
  Dense layers: 4 (including the output dense layer)
- Optimizer: Stochastic Gradient Descent (SGD)
- Trainable Parameters: 197,160,836
  Non-Trainable Parameters: 9,728

The model trained for 3 epochs, after which it started to over-fit and was eventually stopped by a call-back. Dropouts techniques are employed to let the model learn redundant paths to the same output. This model had its own limitations (prime issue being such a small dataset). As expected, the best accuracy achieved by CNN after varying the hyper-parameters was 30%, and it classified every image in the normal class. Thus we shifted transfer learning approaches.

Next, we implemented transfer learning on three pre-trained models (VGG19, Inception and Xception) to see if there were any improvements in the results.

4.3.2 VGG19 model

About the VGG19 model (pre-trained on the ImageNet dataset):

- VGG19 has 16 convolution layers, 3 Fully connected layers, with the final fully connected layer having Softmax activation function as the non-linear part.
- The base convolutional layers having pre-trained weights were frozen.
- For our classification network: We used two dense layers. The final dense layer has four neurons (one for each class) and the activation function as Softmax activation.
- For fine-tuning, we unfroze the base convolutional layers and ran the model for additional 20 epochs or the callbacks stopped the model from training. (Whichever is smaller)
- For classification, total trainable parameters were 100,356, while non-trainable were 20,024,384. Whereas for fine-tuning, they were 20,124,740 and 0 respectively.
- The network was trained with call-backs, to stop training the network in case of over-fitting.
- Training parameters:
    1. Loss Function: Categorical Cross-entropy
    2. Optimizer: ADAM

The pre-trained weights helped a lot in adjusting the classification network to our dataset. VGG19 showed significant improvement in the accuracy, loss values. This model clocked out at 77.7% accuracy and a loss of 0.532 after fine tuning.

4.3.3 InceptionV3 model

The third model we worked upon was InceptionV3. Again, using transfer learning, we trained this model on our dataset and noted its performance metrics. Model specifications are:

- InceptionV3: a quite complex model with concepts such as: Factorized convolutions, Smaller convolutions, Asymmetric convolutions, Grid size reduction.
- The base convolutional layers having pre-trained weights were frozen.
- For classification, total trainable parameters were 13,108,484 & non-trainable were 21,802,784. Whereas for fine-tuning they were 34,876,836 and 34,432 respectively.

With the results getting better with each epoch, this model started to over-fit at 12th epoch and was stopped by call-back. The best saved model is then fine-tuned and resulted in the accuracy score of 82.22% and loss of 0.4032 on the test set.

4.3.4 Xception model

Finally, we used transfer learning on the Xception network. Some details for this model are:

- It was first termed as the extreme version of Inception model, hence, Xception [17]
- Based entirely on depth-wise separable convolution layers
- 36 convolutional layers forming the feature extraction base of the network

- The 36 convolutional layers are structured into14 modules, all of which have linear residual connections around them, except for the first and last modules [17]
- The base convolutional layers having pre-trained weights were frozen.
- For classification, total trainable parameters were 25,691,396 & non-trainable were 20,861,480. Whereas for fine-tuning, they were 46,498,348 and 54,528 respectively.

Xception performed extremely well on our dataset and gave us decent results with the loss and accuracy values of 0.343 and 85.3%.

5 Performance Metrics and Results

We developed and trained the models using around 2040 images, and now we test them with the test-split dataset consisting approximately 230 samples. The area under receiver operating characteristics (ROC) curve is shown in Figure 6. The final performance of the models is evaluated on Accuracy, Precision , Recall and F1 score which are defined in terms of true positive (TP), false positive (FP), true negative (FP), and false negative (FN).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN} \qquad F1\ score = 2 * \frac{Precision * Recall}{(Precision + Recall)}$$

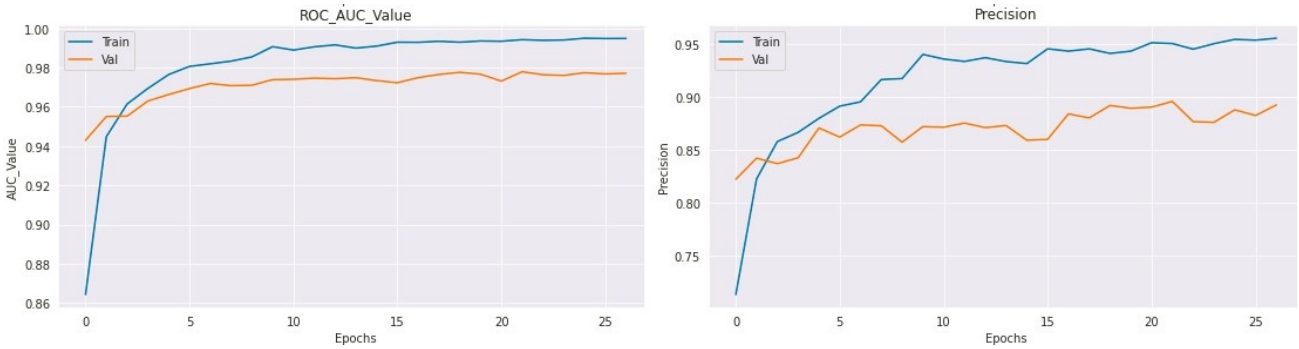
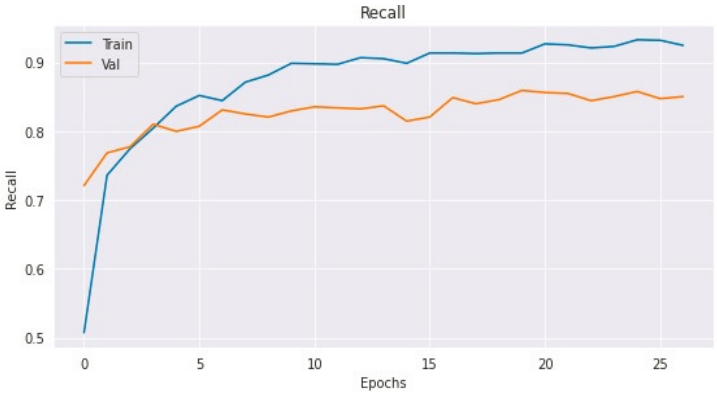


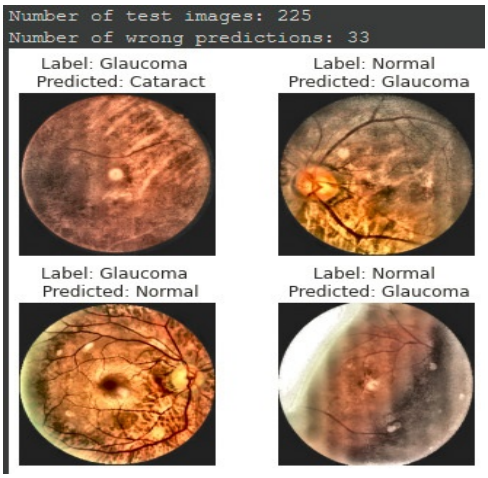
Figure 6: Behaviour of Area Under Curve (AUC) and Precision with number of epochs

Precision talks about how precise our model is. That is, out of all the predicted positives, how many are true positives. Achieving a good score on this metric is important for this problem. For instance, while deciding whether or not an eye is normal, predicting an eye suffering from cataract as a normal eye, is a case of false positive. This leads to the ignorance of the disease and ultimately results in partial/permanent blindness. Our final model (Xception) achieved a precision of 83.5% (Figure 6).

Recall on the other hand is calculated using False Negatives and True Positives (Figure 7a). Recall is the model metric used to select the best model when there is a high cost associated with False Negative, which is also a perfect metric for our case. For example, a person has a perfectly normal eye but he is diagnosed to have the cataract disease (False Negative). This might make him take unnecessary treatment which is in no condition acceptable. Recall for our model was 97.9%.



(a) Recall vs number of epochs



(b) Predictions that the model wrongly guessed

Figure 7: Other metrics for performance analysis

F1 Score is needed when we want to seek a balance between Precision and Recall. The same behaviour can be achieved using accuracy, but it can be largely contributed by many True Negatives which in many cases might not make a difference. We got a F1 score of 85.3% (Table 1).

Talking about the results, with time the loss of the network decreased and accuracy increased. Loss on test set turned out to be 0.3425 while the accuracy reached 0.8559 (Figure 8). The deep learning network with the best results in the performance metrics is found to be Xception (refer Table 1).

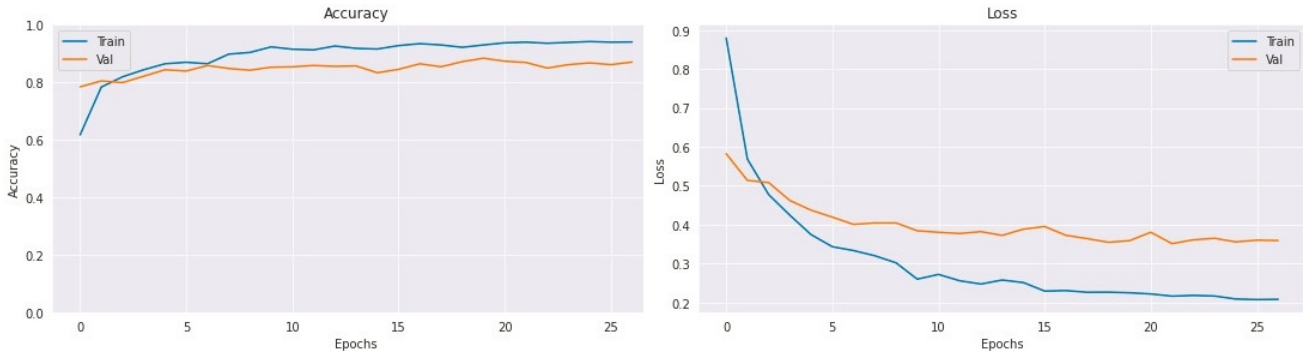


Figure 8: Plot of Loss and Accuracy against number of epochs

	CNN	VGG19	InceptionV3	Xception
Loss	1.389	0.532	0.403	**0.343**
Accuracy	0.302	0.777	0.822	**0.853**
ROC AUC Value	0.0	0.799	0.831	**0.850**
Recall	0.500	0.947	0.968	**0.979**
Precision	0.0	0.742	0.808	**0.835**
F1 Score	0.140	0.773	0.822	**0.853**

Table 1: A comparison between all the models we tried with their respective results on different metrics tested on the test set

6 Conclusion

Using transfer learning techniques on the three models in study, we compensate the requirement of more data samples to train a deep learning model. Among all the models, the best model suited for the dataset was Xception.

7 Future Work

Further work can be done for the classification of the entire dataset all at once, by reading and learning more about 'multi-label, multi-class' classification techniques. We plan to find similar data-sets having one or more of the eye diseases mentioned in this data-set and pre-process and merge them together in order to create a bigger data-set and then work on training our CNN from scratch. This will also enable us to compare more number of pre-trained models and provide a benchmark for the dataset.

References

[1] Yau JW, Rogers SL, Kawasaki R, Lamoureux EL, Kowalski JW, Bek T, Chen SJ, Dekker JM, Fletcher A, Grauslund J, Haffner S, Hamman RF, Ikram MK, Kayama T, Klein BE, Klein R, Krishnaiah S, Mayurasakorn K, O'Hare JP, Orchard TJ, Porta M, Rema M, Roy MS, Sharma T, Shaw J, Taylor H, Tielsch JM, Varma R, Wang JJ, Wang N, West S, Xu L, Yasuda M, Zhang X, Mitchell P, Wong TY; Meta-Analysis for Eye Disease (META-EYE) Study Group. Global prevalence and major risk factors of diabetic retinopathy. Diabetes Care. 2012 Mar;35(3):556-64. doi: 10.2337/dc11-1909. Epub 2012 Feb 1. PMID: 22301125; PMCID: PMC3322721.

[2] Cheung N, Mitchell P, Wong TY. Diabetic retinopathy. Lancet. 2010 Jul 10;376(9735):124-36. doi: 10.1016/S0140-6736(09)62124-3. Epub 2010 Jun 26. PMID: 20580421.

[3] Ting DSW, Cheung CY, Lim G, et al. Development and Validation of a Deep Learning System for Diabetic Retinopathy and Related Eye Diseases Using Retinal Images From Multiethnic Populations With Diabetes. JAMA. 2017;318(22):2211–2223. doi:10.1001/jama.2017.18152

[4] Bourne RRA, Flaxman SR, Braithwaite T, Cicinelli MV, Das A, Jonas JB, et al.; Vision Loss Expert Group. Magnitude, temporal trends, and projections of the global prevalence of blindness and distance and near vision impairment: a systematic review and meta-analysis. Lancet Glob Health. 2017 Sep;5(9):e888–97.

[5] Anton A, Fallon M, Cots F, Sebastian MA, Morilla-Grasa A, Mojal S, Castells X. Cost and detection rate of glaucoma screening with imaging devices in a primary care center. Clin Ophthalmol. 2017;11:337-346

[6] Xiangyu Chen, Yanwu Xu, Damon Wing Kee Wong, Tien Yin Wong, & Jiang Liu (2015). Glaucoma detection based on deep convolutional neural network. Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Annual International Conference, 2015, 715–718. https://doi.org/10.1109/EMBC.2015.7318462

[7] Abbas, Q. (2017). Glaucoma-deep: detection of glaucoma eye disease on retinal Fundus images using deep learning. Int J Adv Comput Sci Appl, 8(6), 41-5.

[8] Linglin Zhang et al., "Automatic cataract detection and grading using Deep Convolutional Neural Network," 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), Calabria, 2017, pp. 60-65, doi: 10.1109/ICNSC.2017.8000068.

[9] Larxel. *Ocular Disease Recognition: Right and left eye Fundus photographs of 5000 patients*. kaggle. Retrieved February 17, 2021, from https://www.kaggle.com/andrewmvd/ocular-disease-recognition-odir5k

[10] Shanggong Medical Technology Co. Ltd. *Peking University International Competition on Ocular Disease Intelligent Recognition (ODIR-2019)*. Grand Challenge. Retrieved February 17, 2021, from https://odir2019.grand-challenge.org/introduction/

[11] Agrawal R. *Optimization Algorithms for Deep Learning*. The Medium. Retrieved February 17, 2021, from https://medium.com/analytics-vidhya/optimization-algorithms-for-deep-learning-1f1a2bd4c46b

[12] *Diabetic Retinopathy Detection*. Kaggle. Retrieved February 17, 2021, from https://www.kaggle.com/c/diabetic-retinopathy-detection

[13] M. T. Islam, S. A. Imran, A. Arefeen, M. Hasan and C. Shahnaz, "Source and Camera Independent Ophthalmic Disease Recognition from Fundus Image Using Neural Network," 2019 IEEE International Conference on Signal Processing, Information, Communication and Systems (SPICSCON), Dhaka, Bangladesh, 2019, pp. 59-63, doi: 10.1109/SPICSCON48833.2019.9065162.

[14] N. Gour and P. Khanna, "Multi-class multi-label ophthalmological disease detection using transfer learning based convolutional neural network", Biomed. Signal Process. Control, vol. 66, Dec. 2020.

[15] Sonali, and Sahu, Sima and Singh, Amit and Ghrera, S.P. and Elhoseny, Mohamed. (2018). An approach for de-noising and contrast enhancement of retinal fundus image using CLAHE. Optics and Laser Technology. 110. 10.1016/j.optlastec.2018.06.061.

[16] Gupta, V. *Color spaces in OpenCV (C++ / Python)*. learnopencv. Retrived April 2, 2021 from https://learnopencv.com/color-spaces-in-opencv-cpp-python/

[17] Chollet, Francois. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195.

Appendix A Code Running Instructions

Our git repository can be found here: GitHub
For step by step directions to successfully run the project, follow this link : Code running guide

Appendix B Code

This is a copy of ENEL_610_Project.ipynb, original file is located at: *Google Colaboratory*

```
# ENEL 610- Biometric Technologies and Systems
## Ocular Disease Intelligent Recognition
### Normal v/s Cataract v/s Glaucoma v/s Myopia Prediction

'''## Getting the dataset'''

import os
os.environ["KAGGLE_CONFIG_DIR"] = '/content'    #Setting Environment variable
!chmod 600 /content/kaggle.json
# API command to directly import the dataset to colab.
!kaggle datasets download -d andrewmvd/ocular-disease-recognition-odir5k
!unzip *.zip && rm *.zip          # Unzipping the dataset and simultaneously
                                  # deleting the zip.

"""## Importing necessary libraries, functions """

import tensorflow as tf
import numpy as np
import pandas as pd
import seaborn as sns
sns.set_style("darkgrid")
import cv2, random
import matplotlib.pyplot as plt
import itertools

from tensorflow.keras.preprocessing.image import ImageDataGenerator, \
load_img, img_to_array
from tensorflow.keras.layers import Flatten, Dense, Input, Conv2D, \
MaxPool2D, AveragePooling2D, BatchNormalization, Dropout
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, \
LearningRateScheduler
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.metrics import AUC, Precision, Recall
from sklearn.metrics import f1_score, confusion_matrix
from tensorflow.keras.preprocessing.image import img_to_array
from tqdm import tqdm

# Importing pre-trained models for transfer learning
```

```
from tensorflow.keras.applications import VGG19, InceptionV3, Xception

AUC_value = AUC(curve = 'ROC', name = 'auc_value', multi_label = False)
# This metric creates four local variables, true_positives, true_negatives,
# false_positives and false_negatives that are used to compute the AUC.
# Here, receiver operating characteristics at default threshold is being used.
# It tells us how much area at the given threshold is the ROC curve covering.
# Higher the area, better the classifier.

# Global Variables

join = True                                     # If true, concatenate the data
image_size = 224                                # Desired Image Size
i_shape = (image_size, image_size, 3)           # Input image shape

lr_cnn = 1e-5                                   # Learning rate for training
lr_ft = 1e-6                                    # Learning rate for fine-tuning

class_names = ['Normal', 'Cataract', 'Glaucoma', 'Myopia']
classes = ['Normal', 'Diabetes', 'Glaucoma', 'Cataract', \
'Age related Macular Degeneration', 'Hypertension', 'Pathological Myopia', \
'Other abnormalities']

"""## Exploratory Data Analysis"""

data = pd.read_csv('/content/full_df.csv')       # Path to full_df.csv
print(data.head(5))                 # Displaying the top three rows and all the
                                    # columns of the dataset
print(data.columns)
print(data.info())
# It can be seen that there no null values in the dataset

DATA_PATH = '/content/ODIR-5K/ODIR-5K/data.xlsx'       # Path to data.xlsx
main_df = pd.read_excel(DATA_PATH)
sample_in_classes = main_df.iloc[:, -8:]
all_classes = sample_in_classes.sum()
print(all_classes)

# The data set is highly imbalanced.
# Only 103 images are there for Hypertension (H) while 1000+ images are there
# Diabetic(D) or Normal(N) class

# Displaying some random images from the dataset
show_images = ['/content/ODIR-5K/ODIR-5K/Training Images/1006_left.jpg', \
'/content/ODIR-5K/ODIR-5K/Training Images/0_right.jpg', \
'/content/ODIR-5K/ODIR-5K/Training Images/1537_left.jpg', \
'/content/ODIR-5K/ODIR-5K/Training Images/1613_right.jpg']
count = 0

plt.figure(figsize = (10, 5))
```

```
for iter in show_images:
    im = cv2.imread(iter)
    im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
    plt.subplot(2, 2, 1 + count)
    plt.imshow(im)
    plt.axis('off')
    count += 1
plt.tight_layout()
# It can be observed that the images are of different sizes.

# Visualizing the dataset
plt.figure(figsize = (10, 6))
plt.pie(all_classes, labels = classes, startangle = 90, autopct='%1.1f%%', \
shadow = True)
plt.axis('equal')
plt.show()

# Function to plot the countplot

def cplot(variable, data_f, hue = None):
    sns.countplot(x = data_f[variable], hue = hue, palette = 'Paired')
    plt.title("{} distribution".format(variable))
    plt.tight_layout()
    plt.show()

plt.figure(figsize = (20, 4))
cplot('Patient Age', data)      # It can be observed that most of the patients
                                # are fairly old

cplot('Patient Sex', data)      # More number of males than female patients

abbrevated = ['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']
for plot in range(8):
    plt.figure(figsize = (8, 4))
    cplot('Patient Sex', data, hue = data[abbrevated[plot]])
    plt.show()

# 0- Disease not present, 1- Disease present

# Plotting the correlation matrix to check if a patient is suffering from two
# or more than two diseases at a time.
# For example: if we consider first row, it can be observed that 15% patient
# suffering from glaucoma are also Diabetic.

columns = main_df.iloc[:,-8:]
normalize = columns.sum()                    # Finding the sum of each column
correlation = columns.T.dot(columns)         # Correlation = C.(C)^T
correlation_2 = correlation / normalize
# Normalizing the values to be in between 0 and 1
# 0- No correlation, 1- 100% correlated
```

```
correlation_2.style.background_gradient().set_precision(3)
# Rounding up to 3 decimal points.

"""## Dataset Creation
### Making a separate dataset having images from cataract and normal class.
"""

# Defining a function to find a keyword (default - cataract, can be overwrite
# by specifying the keyword) in the text

def if_keyword(text, keyword):
    if keyword in text:
        return 1
    else:
        return 0

# Finding the keyword in left and right diagnostic keywords.

data["left_cataract"] = data["Left-Diagnostic Keywords"].apply\
(lambda x: if_keyword(x, 'cataract'))
data["right_cataract"] = data["Right-Diagnostic Keywords"].apply\
(lambda x: if_keyword(x, 'cataract'))

data['left_normal'] = data['Left-Diagnostic Keywords'].apply\
(lambda x: if_keyword(x, 'normal fundus'))
data['right_normal'] = data['Right-Diagnostic Keywords'].apply\
(lambda x: if_keyword(x, 'normal fundus'))

data['left_glaucoma'] = data['Left-Diagnostic Keywords'].apply\
(lambda x: if_keyword(x, 'glaucoma'))
data['right_glaucoma'] = data['Right-Diagnostic Keywords'].apply\
(lambda x: if_keyword(x, 'glaucoma'))

data['left_myopia'] = data['Left-Diagnostic Keywords'].apply\
(lambda x: if_keyword(x, 'pathological myopia'))
data['right_myopia'] = data['Right-Diagnostic Keywords'].apply\
(lambda x: if_keyword(x, 'pathological myopia'))

# Images having 'cataract' associated with their Diagnostic Keywords

left_cataract = data.loc[(data.C == 1) & (data.left_cataract == 1)]\
["Left-Fundus"].values
right_cataract = data.loc[(data.C == 1) & (data.right_cataract == 1)]\
["Right-Fundus"].values

# 300-Images having 'normal' associated with their Diagnostic Keywords
# All or more images can be taken into consideration by removing or changing\
# the sample function

left_normal = data.loc[(data.N == 1) & (data.left_normal == 1)]\
```

```
["Left-Fundus"].sample(300, random_state = 11).values
right_normal = data.loc[(data.N == 1) & (data.right_normal == 1)]\
["Right-Fundus"].sample(300, random_state = 11).values

left_glaucoma = data.loc[(data.G == 1) & (data.left_glaucoma == 1)]\
["Left-Fundus"].values
right_glaucoma = data.loc[(data.G == 1) & (data.right_glaucoma == 1)]\
["Right-Fundus"].values

left_myopia = data.loc[(data.M == 1) & (data.left_myopia == 1)]\
["Left-Fundus"].values
right_myopia = data.loc[(data.M == 1) & (data.right_myopia == 1)]\
["Right-Fundus"].values

# Joining both the arrays (left eye's and right eye's diagnosys) into one
# single arrays for each of the classes.

if join:
    cataract = np.concatenate((left_cataract, right_cataract), axis = 0)
    normal = np.concatenate((left_normal, right_normal), axis = 0)
    glaucoma = np.concatenate((left_glaucoma, right_glaucoma), axis = 0)
    myopia = np.concatenate((left_myopia, right_myopia), axis = 0)
    join = False

print(f'Cataract: {len(cataract)} \t Normal: {len(normal)} \t Glaucoma:\
{len(glaucoma)} \t Myopia: {len(myopia)}')

# Visualizing the dataset
our_classes = [600, 594, 616, 457]
plt.figure(figsize = (10, 6))
plt.pie(our_classes, labels = class_names, startangle = 90, \
autopct='%1.1f%%', shadow = True)
plt.axis('equal')
plt.show()

def CLAHE(img, clipLimit, tileGridSize):
  clahe = cv2.createCLAHE(clipLimit = clipLimit, tileGridSize = tileGridSize)
  lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)  # convert from BGR to LAB
  l, a, b = cv2.split(lab)                    # split on 3 different channels
  l2 = clahe.apply(l)                         # apply CLAHE to the L-channel
  lab = cv2.merge((l2,a,b))                   # merge channels
  img = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)  # convert from LAB to BGR
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  return img

# Creating one single dataset by combing the images from normal, cataract,
# glaucoma and Myopia diagnosys

dataset_dir = '/content/preprocessed_images'
dataset = []
```

```
def create_dataset(image_category, label):
    for img in tqdm(image_category):          # Showing the progress bar
        image_path = os.path.join(dataset_dir, img)
        try:
            image = cv2.imread(image_path, cv2.IMREAD_COLOR)
            # cv2.IMREAD_COLOR loads a color image.
            image = cv2.resize(image, (image_size, image_size))
            # Resizing the image to(224, 224)-default input to the VGG19 model
        except:
            continue
        image = CLAHE(image, 20, (10, 10))
        dataset.append([np.array(image), np.array(label)])
    random.shuffle(dataset)
    return dataset

data_set1 = create_dataset(normal, 0)
data_set1 = create_dataset(cataract, 1)
data_set1 = create_dataset(glaucoma, 2)
data_set1 = create_dataset(myopia, 3)

print(len(data_set1))
# For Cataract v/s Normal v/s Glaucoma v\s Myopia prediction we have thus
# created a mini-dataset from the original dataset

# Displaying randomly selected nine images from the dataset

plt.figure(figsize = (10, 5))
for iter in range(8):
    im = random.choice(range(len(dataset)))
    image = dataset[im][0]
    category = dataset[im][1]
    plt.subplot(2, 4, 1 + iter)
    plt.imshow(image)
    plt.axis('off')
    plt.title("Label: %s" %class_names[category])
plt.tight_layout()

# Splitting the dataset into data and associated labels

x_dev = np.array([i[0] for i in dataset]).reshape(-1, image_size, image_size,\
3)
y_dev = np.array([i[1] for i in dataset])

"""## Train, Validation, Test - Split"""

# Shuffling the samples

indexes = np.arange(x_dev.shape[0])
np.random.shuffle(indexes)
X_dev = x_dev[indexes,:]
```

```
Y_dev = y_dev[indexes]

# Then, we split our data into train/val/test sets
train_split = np.int(0.6 * Y_dev.size)
val_split = np.int(0.9 * Y_dev.size)

X_train = X_dev[: train_split, : ]          # 60% of the data for training
Y_train = Y_dev[: train_split]

X_val = X_dev[train_split : val_split, : ]  # 30% for validation
Y_val = Y_dev[train_split : val_split]

X_test = X_dev[val_split: , :]              # 10% for testing
Y_test = Y_dev[val_split : ]

print(f'X_train: {X_train.shape} \tX_val: {X_val.shape} \tX_test: \
{X_test.shape}')

"""## Data Scaling"""

# Min-Max Normalization
Train_min = X_train.min()
Train_max = X_train.max()

X_train = (X_train - Train_min)/(Train_max - Train_min)
X_val = (X_val - Train_min)/(Train_max - Train_min)
X_test = (X_test - Train_min)/(Train_max - Train_min)

"""## One Hot Encoding"""

# Representing the labels as one-hot encoded

Y_train_oh = to_categorical(Y_train)
Y_val_oh = to_categorical(Y_val)
Y_test_oh = to_categorical(Y_test)

print(f'Labels: {Y_train[:2]}')
print(f'One hot encoded labels: \n{Y_train_oh[:2]}' )

"""## Data Augmentation"""

# Data Augmentation is used to feed the model with minor changes.
# This technique is very useful when we have a dataset is very imbalannced
# and/or small in size.
# Here, we are using techniques such as rotating the images by 15 degrees
# in a random fashion, flipping the images horizontally.
# At a time, a batch of 32 images are fed.

batch_size = 32
gen_params = {"rotation_range":15, "horizontal_flip":True, \
```

```
"fill_mode":'constant', 'cval':0 }
train_gen = ImageDataGenerator(**gen_params)
val_gen = ImageDataGenerator(**gen_params)

train_gen.fit(X_train, seed = 1)
val_gen.fit(X_val, seed = 1)

train_flow = train_gen.flow(X_train, Y_train_oh, batch_size = batch_size)
val_flow = val_gen.flow(X_val, Y_val_oh, batch_size = batch_size)

# The keras ImageDataGenerator returns a generator and thus we have to
# use getitem().

plt.figure(figsize = (14, 6))
X_batch, Y_batch = train_flow.__getitem__(0)
print(f'Minimum pixel value: {X_batch.min()} \t Maximum pixel value: \
{X_batch.max()}')
for i in range(8):
    plt.subplot(2, 4, i+1)
    plt.imshow(X_batch[i])
    plt.title("Label: %s" %class_names[int(Y_batch[i].argmax())])
    plt.axis('off')
plt.show()

"""## Defining the models"""

# Here we define our experimental model. This model was finalized after doing
# a lot of manipulations. Starting with the learning rate to the number of
# neurons, kernel size, number of kernels everything was changed. We used
# Average Pooling, Max Pooling, batch normalization- no batch normalization,
# with and without dropouts and find out this model was performing better if
# not the best.

def our_cnn(lr = lr_cnn):
  print(f'\nYou chose our cnn model. \n')

  inputs = Input(shape = i_shape)
  # Padding is used to keep the dimensions same before and after convolution
  c1 = Conv2D(256, (3, 3), activation = 'relu', padding = 'same')(inputs)
  c2 = Conv2D(256, (3, 3), activation = 'relu', padding = 'same')(c1)
  mp1 = MaxPool2D(2, 2)(c2)
  bn1 = BatchNormalization()(mp1)
  dp1 = Dropout(0.3)(bn1)

  c3 = Conv2D(512, (3, 3), activation = 'relu', padding = 'same')(dp1)
  c4 = Conv2D(512, (3, 3), activation = 'relu', padding = 'same')(c3)
  mp2 = MaxPool2D(2, 2)(c4)
  bn2 = BatchNormalization()(mp2)
  dp2 = Dropout(0.25)(bn2)
```

```
    c5 = Conv2D(1024, (3, 3), activation = 'relu', padding = 'same')(dp2)
    c6 = Conv2D(1024, (3, 3), activation = 'relu', padding = 'same')(c5)
    mp3 = MaxPool2D(2, 2)(c6)
    bn3 = BatchNormalization()(mp3)
    dp3 = Dropout(0.25)(bn3)

    c7 = Conv2D(1024, (3, 3), activation = 'relu', padding = 'same')(dp3)
    c8 = Conv2D(1024, (3, 3), activation = 'relu', padding = 'same')(c7)
    mp4 = MaxPool2D(2, 2)(c8)
    bn4 = BatchNormalization()(mp4)
    dp4 = Dropout(0.25)(bn4)

    c9 = Conv2D(2048, (3, 3), activation = 'relu', padding = 'same')(dp4)
    c10 = Conv2D(2048, (3, 3), activation = 'relu', padding = 'same')(c9)
    mp5 = MaxPool2D(2, 2)(c10)
    bn5 = BatchNormalization()(mp5)
    dp5 = Dropout(0.3)(bn5)

    # Since Dense layers or fully connected layers require input in a
    # single dimension, flatten was used.
    flat = Flatten()(dp5)
    d1 = Dense(1024, activation = 'relu')(flat)
    d2 = Dense(512, activation = 'relu')(d1)
    d3 = Dense(128, activation = 'relu')(d2)

    # Softmax outputs each class as an equivalent of probability. 4 classes
    # each corresponding to a single disease
    out = Dense(4, activation = 'softmax')(d3)

    our_model = Model(inputs = inputs, outputs = out)   # Finalizing the model
    our_model.compile(optimizer = SGD(learning_rate = lr), loss = \
    'categorical_crossentropy', metrics = ['accuracy', Precision(), Recall(), \
    AUC_value])
    # Model is compiled with loss as categorical crossentropy, optimizer being
    # stochastic gradient descent with initial learing rate = 1e-5, and four
    # metrics to monitor the training.

    return our_model

def vgg_model(train_l = False, lr = lr_cnn):
    print(f'\nYou chose VGG19. \n')
    # Importing the model with the weights achieved on Imagenet dataset
    vgg = VGG19(weights = "imagenet", include_top = False, input_shape = \
    (image_size, image_size, 3))
    # Classification network/fully connected layers are not imported.
    vgg.trainable = train_l           # Freezing the trainable paramters

    input_image = Input(shape = i_shape)
    x1 = vgg(input_image, training = False)
    x2 = Flatten()(x1)
```

```
    x3 = Dense(256, activation = 'relu')
    out = Dense(4, activation = 'softmax')(x2)

    vgg_model = Model(inputs = input_image, outputs = out)
    vgg_model.compile(optimizer = Adam(learning_rate = lr), loss = \
    'categorical_crossentropy', metrics = ['accuracy', Precision(), \
    Recall(), AUC_value])

    return vgg_model

def inceptionV3(train_l = False, lr = lr_cnn):
  print(f'\nYou chose InceptionV3. \n')
  # Importing/Calling inception from keras applications

  inception = InceptionV3(weights = 'imagenet', include_top = False, \
  input_shape = i_shape)
  inception.trainable = train_l

  input_image = Input(shape = i_shape)
  x1 = inception(input_image, training = False)
  x2 = Flatten()(x1)
  x3 = Dense(256, activation = 'relu')(x2)
  out = Dense(4, activation = 'softmax')(x3)

  inception_model = Model(inputs = input_image, outputs = out)
  inception_model.compile(optimizer = Adam(learning_rate = lr), loss = \
  'categorical_crossentropy', metrics = ['accuracy', Precision(), \
  Recall(), AUC_value])

  return inception_model

def xception(train_l = False, lr = lr_cnn):
  print(f'\nYou chose Xception. \n')
  # Here, we use the improved version of Inception model on our dataset.

  xception = Xception(weights = 'imagenet', include_top = False, \
  input_shape = i_shape)
  xception.trainable = train_l

  input_image = Input(shape = i_shape)
  x1 = xception(input_image, training = False)
  x2 = Flatten()(x1)
  x3 = Dense(256, activation = 'relu')(x2)
  out = Dense(4, activation = 'softmax')(x3)

  xception_model = Model(inputs = input_image, outputs = out)
  xception_model.compile(optimizer = Adam(learning_rate = lr), \
  loss = 'categorical_crossentropy', metrics = ['accuracy', Precision(), \
  Recall(), AUC_value])
```

```
  return xception_model

"""## Calling the model"""

# This function lets you select one model out of all the models.
# An input will be asked from the user.

def model_s():
  model_selection = int(input(f'Choose a model please- \n1: Our CNN model\
  \n2: VGG19 model \n3: InceptionV3 model \n4: Xception Model \n'))

  if model_selection == 1:
    model_name = 'our_cnn.h5'       # Path to save and load the model at.
    model = our_cnn()

    model.summary()
    return model, model_name, model_selection

  elif model_selection == 2:
    model_name = 'vgg19.h5'       # Path to save and load the vgg19 model at.
    model = vgg_model()

    model.summary()
    return model, model_name, model_selection

  elif model_selection == 3:
    model_name = 'inceptionV3.h5'  # Path to save & load the inceptionV3 model
    model = inceptionV3()

    model.summary()
    return model, model_name, model_selection

  elif model_selection == 4:
    model_name = 'xception.h5'     # Path to save & load the xception model
    model = xception()
    model.summary()
    # This function also prints the summary of the model, displaying the
    # structure, paramteres.
    return model, model_name, model_selection

  else:
    print('Wrong Choice, please choose again')
    # In case of a wrong input, you are again asked to select a proper input.
    model_s()

model, model_name, model_selection = model_s()

"""## Defining the callbacks"""

# These callbacks monitor the training. From chaning the learing rate
```

```
# throughout the training process to stopping the training if the validation
# loss starts increasing everything is being controlled by these callbacks.
# Here, we save the best model at the path specified by the variable
# 'model_name'. This saved model can be later directly used without training
# for the classifying 4 classes.

model_checkpoint = ModelCheckpoint(model_name, monitor = 'val_loss', \
save_best_only = True, save_weights_only = False, mode = 'min')

early_stop = EarlyStopping(monitor = 'val_loss', mode = 'min', \
restore_best_weights = True, patience = 5, min_delta = 0.0001)

def scheduler(epoch, lr):
    if (epoch + 1) % 5 == 0 and epoch < 20:
        lr /= 2
    elif (epoch + 1) % 15 == 0:
        lr /= 2
    return lr

lr_schedule = LearningRateScheduler(scheduler, verbose = 1)

"""## Training the model"""

# This cell starts the training of the model and saves the statistics during
# the training to the variable 'history'.

history = model.fit(train_flow, batch_size = 32, epochs = 50, validation_data\
= (val_flow), verbose = 1, callbacks = [lr_schedule, model_checkpoint, \
early_stop])

"""## Fine Tuning the Model"""

# This block of code is used to fine tune the model, i.e. we even update the
# weights of kernels of convolutional layers to let them adapt better to our
# classifier.

if model_selection == 1:
    model = load_model('our_cnn.h5')
elif model_selection == 2:
    model = load_model('vgg19.h5')
elif model_selection == 3:
    model = load_model('inceptionV3.h5')
else:
    model = load_model('xception.h5')

model.trainable = True
model.compile(optimizer = Adam(learning_rate = lr_ft), loss = \
'categorical_crossentropy', metrics = ['accuracy', Precision(), \
Recall(), AUC_value])
```

```
model.summary()

# Same callbacks are used for fine-tuning as well, so as to stop the model
# from over-fitting.
tune_history = model.fit(train_flow, batch_size = 32, epochs = 20, \
validation_data = (val_flow), verbose = 1, callbacks = [lr_schedule, \
model_checkpoint, early_stop])

"""## Testing the best saved model"""

# Evaluating the model

# We use the weights from the best saved model.
model.load_weights(model_name)
Y_pred = model.predict(X_test)
Y_pred = Y_pred.argmax(axis = 1)      # The maximum value of all the values
                                      # is chosen for each prediction
loss, accuracy, auc, precision, recall = model.evaluate(X_test, Y_test_oh, \
verbose = 0)
print(f'loss: {loss} \tAccuracy: {accuracy} \tAUC_value: {auc} \tPrecision: \
{precision} \tRecall: {recall}')

# Here, F1 score is calculated on the predicted values.

F1_score = f1_score(y_true = Y_test, y_pred = Y_pred, average = 'weighted')

print(f'F1 score: {F1_score}')

# This function to calculate confusion matrix was taken from StackOverFlow

classes=['N', 'C', 'G', 'M']             # Labels for the confusion matrix
true_classes = Y_test
print('Confusion Matrix')
cm = confusion_matrix(true_classes, Y_pred)         # From sklearn metrics

def plot_confusion_matrix(cm, classes, title = 'Confusion matrix', \
cmap = plt.cm.Oranges):
    plt.figure(figsize=(6, 6))
    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 30)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j], horizontalalignment="center",  \
        color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
```

```
    plt.xlabel('Predicted label')

plot_confusion_matrix(cm, classes)

# True positive (TP): correct positive prediction
# False positive (FP): incorrect positive prediction
# True negative (TN): correct negative prediction
# False negative (FN): incorrect negative prediction

# Calculating the error rates: TPR, TNR, FPR

def calculate_tpr_tnr_fpr(y_test, y_pred_test):
    actual_pos = y_test == 1
    actual_neg = y_test == 0

    true_pos = (y_pred_test == 1) & (actual_pos)
    false_pos = (y_pred_test == 1) & (actual_neg)
    true_neg = (y_pred_test == 0) & (actual_neg)
    false_neg = (y_pred_test == 0) & (actual_pos)

    tpr = np.sum(true_pos) / np.sum(actual_pos)
    tnr = np.sum(true_neg) / np.sum(actual_neg)
    fpr = np.sum(false_pos) / np.sum(actual_neg)

    return tpr, tnr, fpr

tpr, tnr, fpr = calculate_tpr_tnr_fpr(Y_test, Y_pred)
print(f'True postive rate, Sensitivity: {tpr}')
print(f'True negative rate, Specitivity: {tnr}')
print(f'False positive rate: {fpr}')

# Plotting the graphs for accuracy, loss, auc_value, precision, recall v/s
# epochs

plt.figure(figsize = (20, 15))
epochs = np.arange(len(history.history["accuracy"]))

# accuracy v/s epochs
plt.subplot(3, 2, 1)
plt.plot(epochs, history.history["accuracy"])
plt.plot(epochs, history.history["val_accuracy"])
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend(["Train", "Val"])
plt.ylim(0, 1)

# loss v/s epochs
plt.subplot(3, 2, 2)
plt.plot(epochs, history.history["loss"])
```

```
plt.plot(epochs, history.history["val_loss"])
plt.title("Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Train", "Val"])

# auc_value v/s epochs
plt.subplot(3, 2, 3)
plt.plot(epochs, history.history["auc_value"])
plt.plot(epochs, history.history["val_auc_value"])
plt.title("ROC_AUC_Value")
plt.xlabel("Epochs")
plt.ylabel("AUC_Value")
plt.legend(["Train", "Val"])

# precision v/s epochs
plt.subplot(3, 2, 4)
plt.plot(epochs, history.history["precision"])
plt.plot(epochs, history.history["val_precision"])
plt.title("Precision")
plt.xlabel("Epochs")
plt.ylabel("Precision")
plt.legend(["Train", "Val"])

# recall v/s epochs
plt.subplot(3, 2, 5)
plt.plot(epochs, history.history["recall"])
plt.plot(epochs, history.history["val_recall"])
plt.title("Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.legend(["Train", "Val"])
plt.show()

# Plotting 8 random images which were classified wrongly

wrong_indexes = np.where(Y_pred != Y_test)[0]
print(f'Number of test images: {Y_test.size}')
print(f'Number of wrong predictions: {wrong_indexes.size}')

sample_indexes = np.random.choice(np.arange(wrong_indexes.shape[0], dtype =\
int), size = 8, replace = False)
plt.figure(figsize = (10, 5))
for (ii,jj) in enumerate(sample_indexes):
    plt.subplot(2, 4 , ii+1)
    plt.imshow(X_test[wrong_indexes[jj]], cmap = "gray")
    plt.title(f"Label: {class_names[Y_test[wrong_indexes[jj]]]} \n \
    Predicted: {class_names[Y_pred[wrong_indexes[jj]]]}")
    plt.axis('off')
plt.tight_layout()
```

```
plt.show()

# Plotting 8 random images which were classified rightly

right_indexes = np.where(Y_pred == Y_test)[0]
print(f'Number of test images: {Y_test.size}')
print(f'Number of right predictions: {right_indexes.size}')

sample_indexes = np.random.choice(np.arange(right_indexes.shape[0], \
dtype = int), size = 8, replace = False)
plt.figure(figsize = (10, 5))
for (ii,jj) in enumerate(sample_indexes):
    plt.subplot(2, 4 , ii+1)
    plt.imshow(X_test[right_indexes[jj]], cmap = "gray")
    plt.title(f"Label: {class_names[Y_test[right_indexes[jj]]]} \n \
    Predicted: {class_names[Y_pred[right_indexes[jj]]]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```