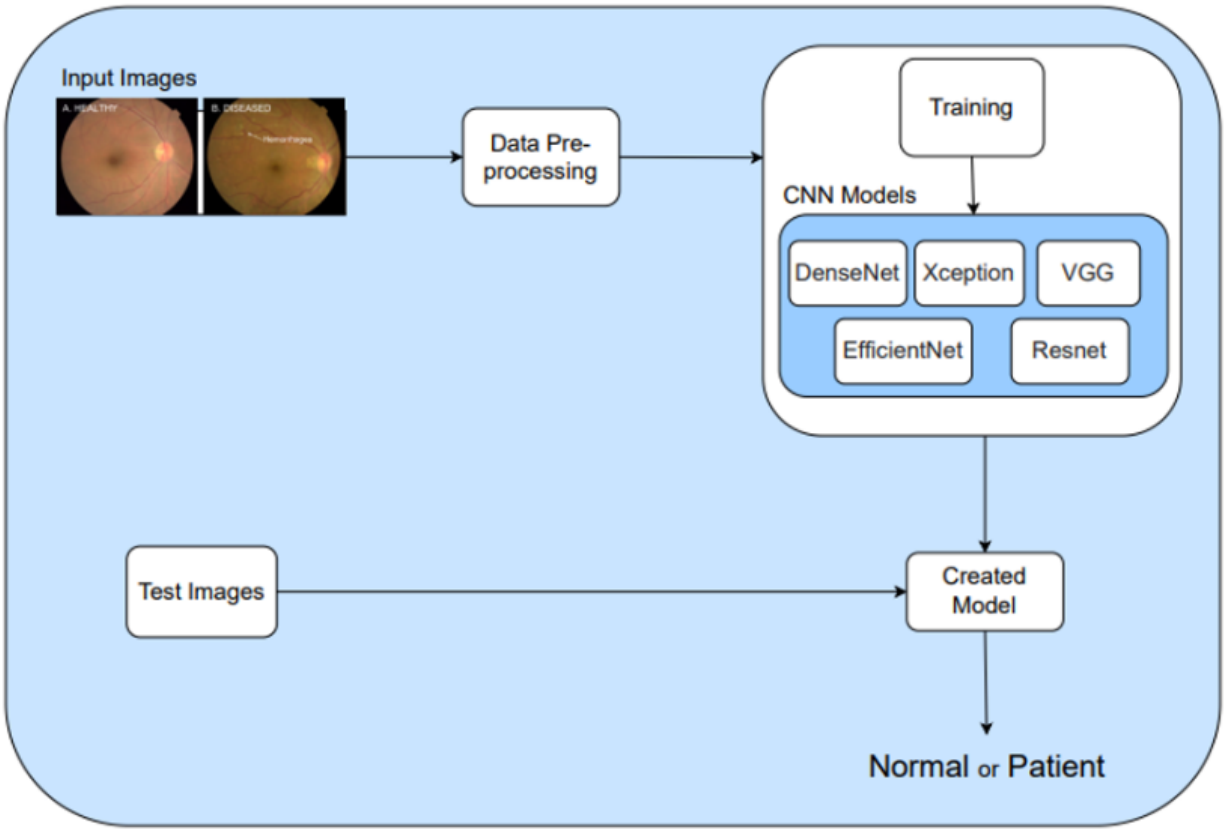
Eye Disease Detection Using Deep Learning

In this project we are classifying various types of Eye Diseases that people get due to various reasons like age, diabetes, etc. These diseases are majorly classified into 4 categories namely Normal, cataract, Diabetic Retinopathy & Glaucoma. Deep-learning (DL) methods in artificial intelligence (AI) play a dominant role as high-performance classifiers in the detection of the Eye Diseases using images.
Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in image analysis and classification. We used Transfer Learning techniques like Inception V3, VGG19, Xception V3 that are more widely used as a transfer learning method in image analysis and they are highly effective.
Technical Architecture:



Prerequisites

In order to develop this project we need to install the following software/packages:
Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning-related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook,
QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using a Jupyter notebook and Spyder
Python packages:
NumPy: NumPy is a Python package that stands for 'Numerical Python. It is the core library for scientific computing, which contains a powerful n-dimensional array of objects.
Pandas: pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool, built on top of the Python programming language.
Matplotlib: It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits
Keras: Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3, Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, R, Theano, and PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.
TensorFlow: TensorFlow is just one part of a much bigger, and growing ecosystem of libraries and extensions that help you accomplish your machine learning goals. It is a free and open-source software library for data flow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks.
Flask: Web framework used for building Web applications
If you are using anaconda navigator, follow the below steps to download the required packages:
Open anaconda prompt.
Type “pip install OpenCV-python” and click enter.
Type “pip install keras==2.2.4” and click enter.
Type “pip install imutils” and click enterpip install imutils
Type “pip install tensorflow==2.0” and click enter. (Make sure you are working on python 64-bit)
Type “pip install Flask” and click enter.

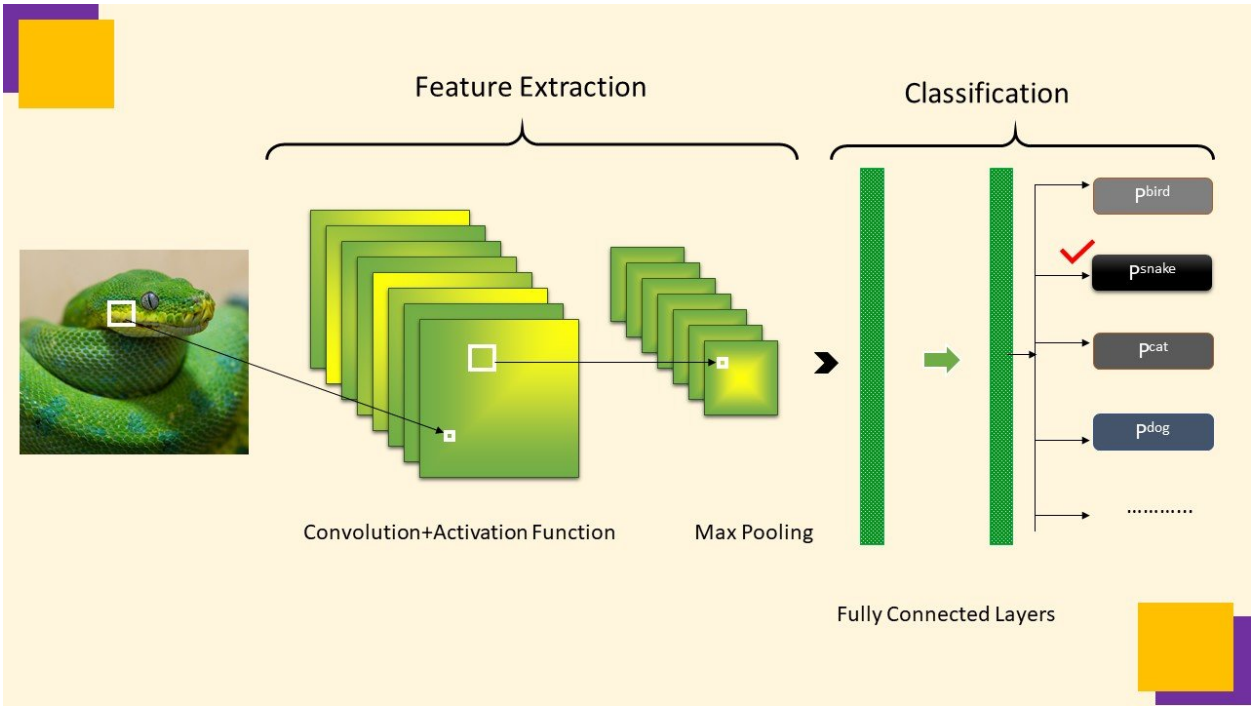
Convolutional Neural Networks (CNN) in Deep Learning

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing structured grid-like data, such as images. They are widely used in image classification, object detection, medical image analysis, and

more. CNNs automatically learn spatial hierarchies of features, making them highly effective for tasks involving visual data.

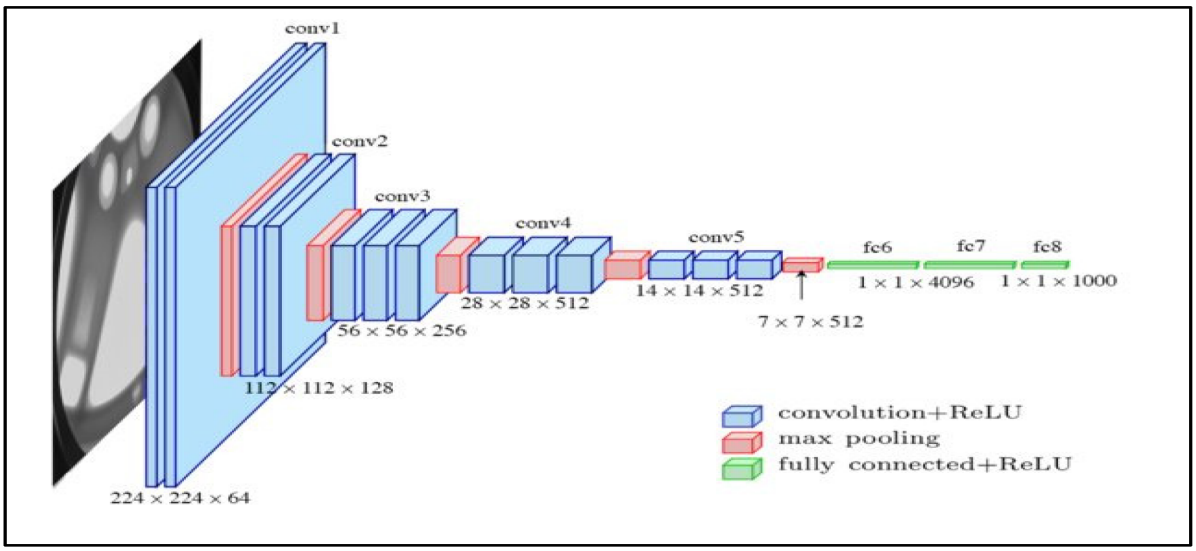
Key Components of CNN

A. Convolutional Layer
The core component of CNN.
Uses filters (kernels) to slide over the input image and extract spatial features (edges, textures, patterns).
Mathematically, it performs element-wise multiplication between the kernel and a patch of the image, followed by summation.
B. Activation Function (ReLU)
Rectified Linear Unit (ReLU) is applied after convolution to introduce non-linearity.
Formula:
$f(x)=max(0,x)$
Helps CNNs learn complex patterns by eliminating negative values.
C. Pooling Layer (Downsampling)
Purpose: Reduces dimensionality while retaining important features.
Types:
Max Pooling: Takes the maximum value in a region.
Average Pooling: Takes the average value in a region.
Example: A 2×2 max pooling reduces a 4×4 matrix to a 2×2 matrix.
D. Fully Connected Layer (FC Layer)
The extracted features from the convolution and pooling layers are flattened and passed to a fully connected neural network.
Helps in making final classifications.
E. Softmax / Sigmoid (Output Layer)
Softmax: Used for multi-class classification, converts logits into probabilities.
Sigmoid: Used for binary classification, outputs a probability between 0 and 1.



VGG16: Deep Learning Model for Image Classification

VGG16 is a deep convolutional neural network (CNN) developed by the Visual Geometry Group at Oxford. It is widely used in image classification, object detection, and medical image analysis due to its simple yet effective architecture.

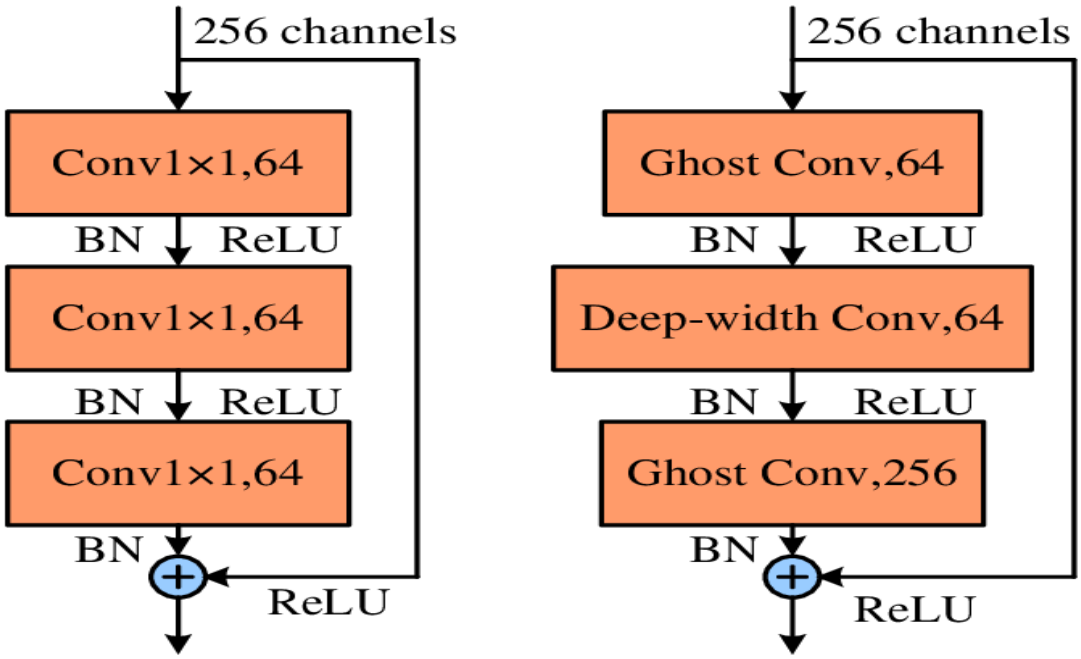


ResNet-50: A Powerful Deep Learning Model

ResNet-50 (Residual Network) is a deep convolutional neural network (CNN) with 50 layers, designed to overcome the vanishing gradient problem in deep learning. It introduced skip (residual) connections, allowing very deep networks to be trained effectively.

Key Features of ResNet-50

Skip Connections (Residual Blocks): Helps in training deeper networks without vanishing gradients.
50 Layers Deep: More powerful than shallow networks like VGG16.
Uses Batch Normalization: Stabilizes training and improves convergence.
Pretrained on ImageNet: Can be used for transfer learning in custom applications like eye disease detection.

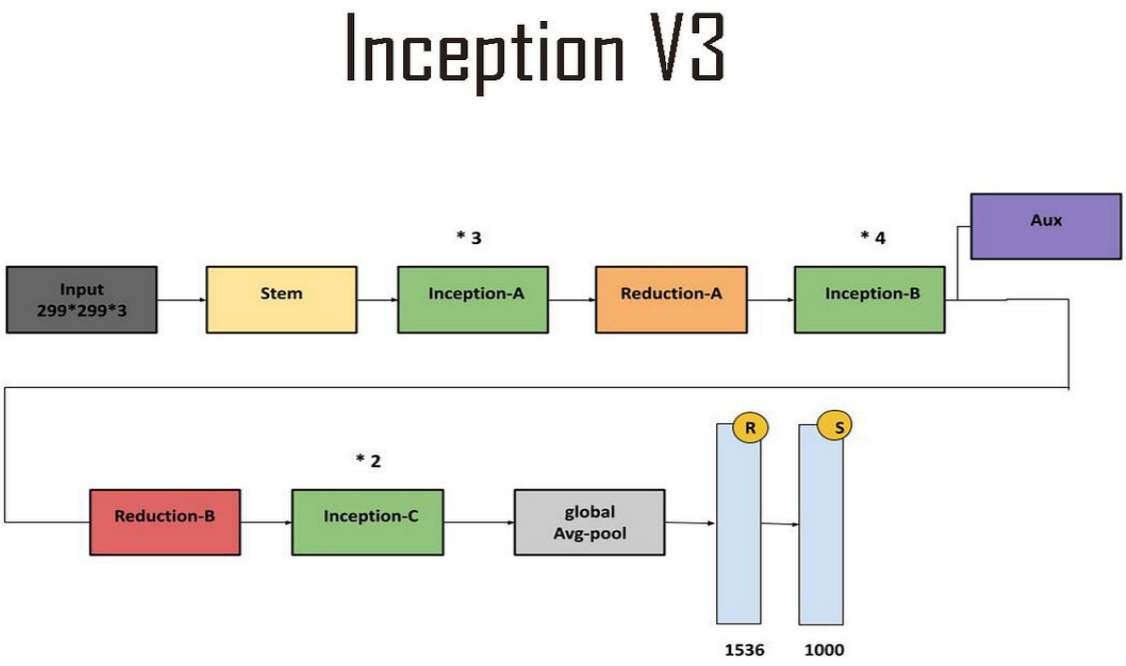


Inception-V3: A Powerful CNN for Image Classification

Inception-V3 is a deep convolutional neural network (CNN) that improves on previous models like VGG16 and ResNet-50. It was developed by Google and is known for its efficiency, accuracy, and Inception Modules, which allow the model to extract multi-scale features in a single layer.

Key Features of Inception-V3

Uses Inception Modules: Combines multiple filter sizes in parallel.
Factorized Convolutions: Reduces computational cost.
Auxiliary Classifiers: Helps in training deep networks.
Pretrained on ImageNet: Can be used for transfer learning.

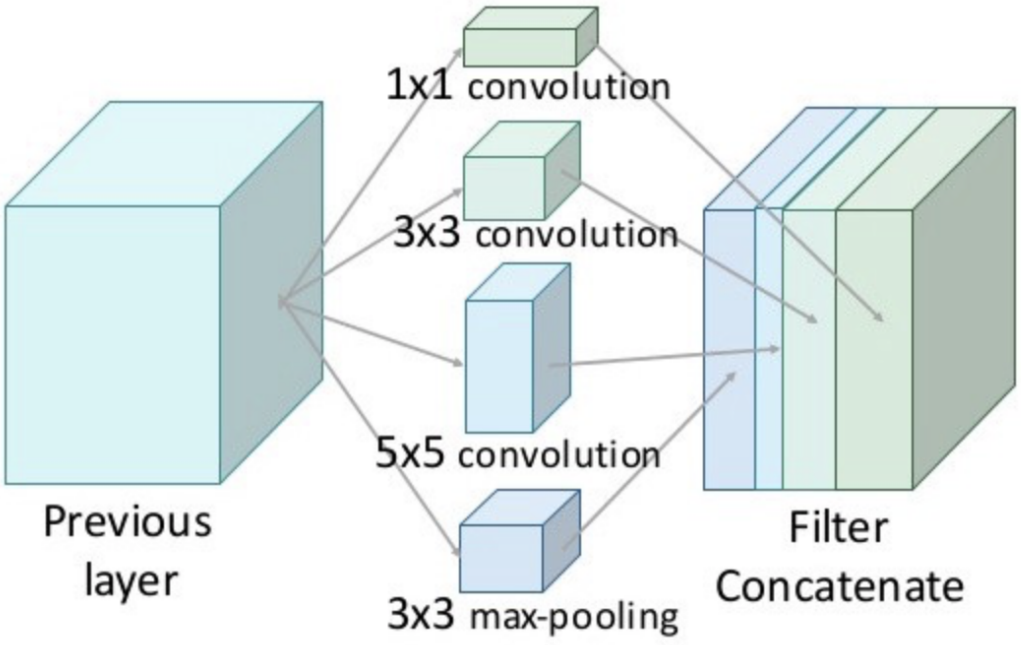


Xception: Extreme Inception - A More Efficient CNN

Xception (Extreme Inception) is a deep convolutional neural network (CNN) that improves upon Inception-V3 by replacing standard convolutions with depthwise separable convolutions. It was introduced by François Chollet (creator of Keras) in 2017.

Key Features of Xception

Uses Depthwise Separable Convolutions: Faster and more efficient than regular convolutions.
Inspired by Inception-V3: Removes fully connected layers, using global average pooling instead.
Pretrained on ImageNet: Suitable for transfer learning.
Lighter than VGG16 & ResNet-50: Only ~22.9 million parameters.



Project Objectives

By the end of this project you'll understand:
Preprocessing the images.
Applying Transfer learning algorithms on the dataset.
How deep neural networks detect the disease.
You will be able to know how to find the accuracy of the model.
You will be able to build web applications using the Flask framework.

Project Flow

The user interacts with the UI (User Interface) to choose the image.

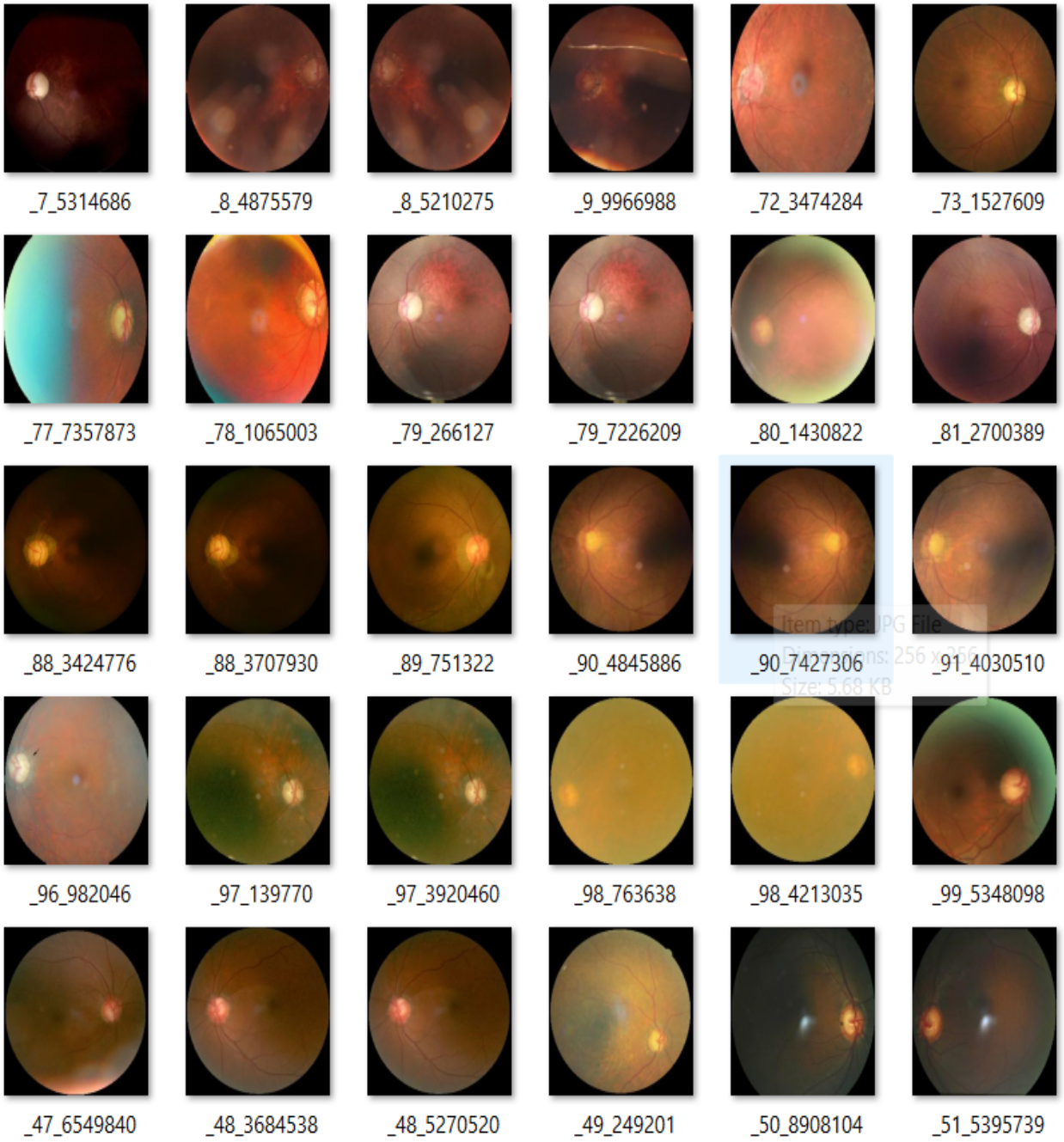
The chosen image is analyzed by the model integrated with flask application.
The Xception Model analyzes the image, then showcases the prediction on the Flask UI.
To accomplish this, we have to complete all the activities and tasks listed
Data Collection..
Data Pre-processing.
Import the required library
Configure ImageDataGenerator class
ApplyImageDataGenerator functionality to Trainset and Testset
Model Building
Pre-trained CNN model as a Feature Extractor
Adding Dense Layer
Configure the Learning Process
Train the model
Save the Model
Test the model
Application Building
Create an HTML file

Project Structure

Flask folder consists of static, templates and app.py.
IBM folder consists of trained model notebook.
Training file consist of eye_disease_detection.ipynb , model training.

Download the dataset

Collect images of Eye Diseases then organize into subdirectories based on their respective names as shown in the project structure. Create folders of types of Eye Diseases that need to be recognized. In this project, we have collected images of 4 types of Eye Diseases images like Normal, cataract, Diabetic Retinopathy & Glaucoma and they are saved in the respective sub directories with their respective names. Note: For better accuracy train on more images We are going to build our training model on Google colab. We will be connecting Kaggle with Google Colab because the dataset is too big to import using the following code:



Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation There are five main types of data augmentation techniques for image data; specifically:

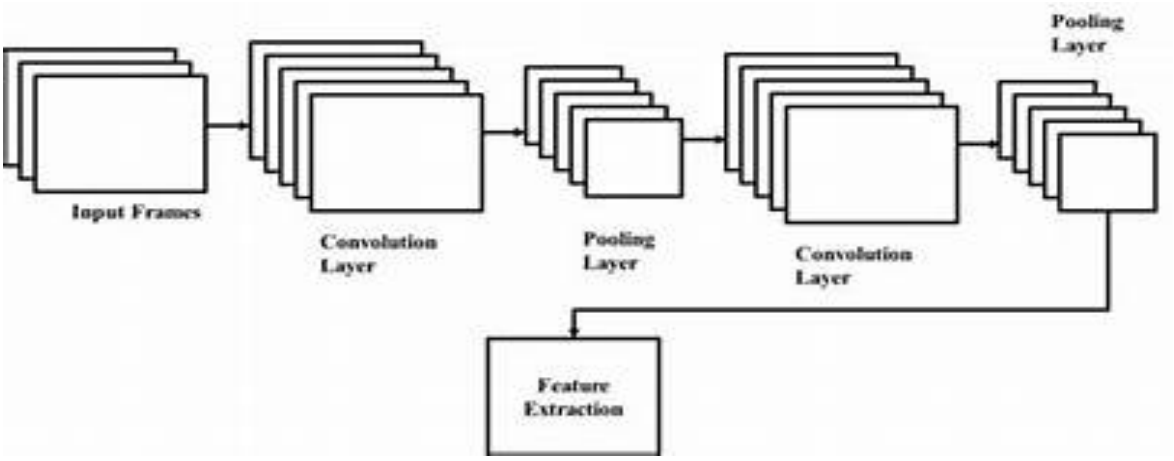
Image shifts via the width_shift_range and height_shift_range arguments.

The image flips via the horizontal_flip and vertical_flip arguments.

Image rotations via the rotation_range argument
Image brightness via the brightness_range argument.
Image zoom via the zoom_range argument.
Let us apply ImageDataGenerator functionality to the Train set and Test set by using the following code. For Training set using flow_from_directory function.
This function will return batches of images from the subdirectories
Arguments:
directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
batch_size: Size of the batches of data which is 64.
target_size: Size to resize images after they are read from disk.
class_mode:
- ‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss). - 'categorical' means that the labels are encodedas a
categorical vector (e.g. for categorical_crossentropy loss).
- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
- None (no labels).

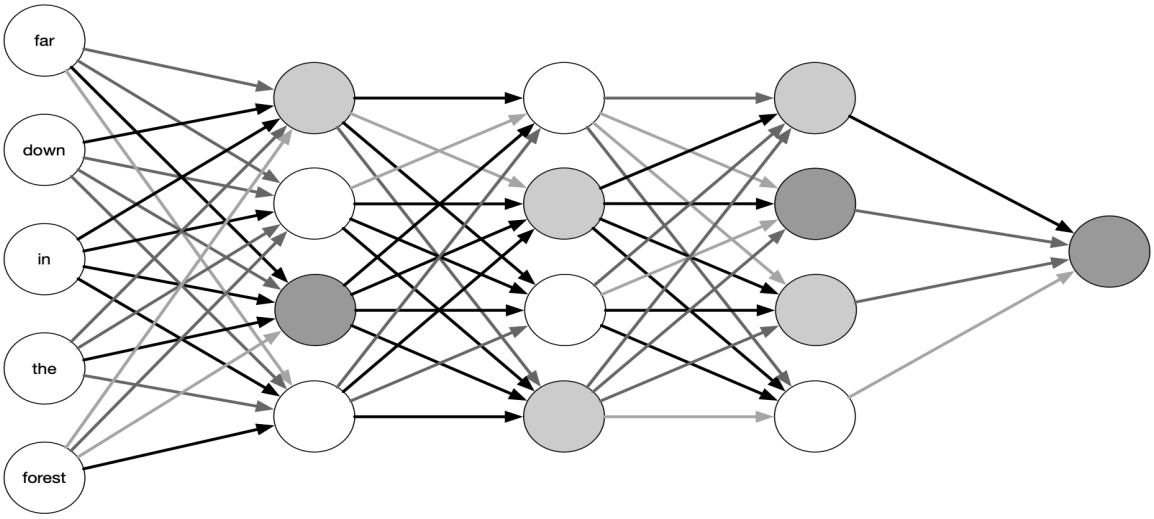
Pre-trained CNN model as a Feature Extractor

For one of the models, we will use it as a simple feature extractor by freezing all the five convolution blocks to make sure their weights don’t get updated after each epoch as we train our own model.
Here, we have considered images of dimension (224,244,3).
Also, we have assigned include_top = False because we are using convolution layer for features extraction and wants to train fully connected layer for our image classification (since it is not the part of Imagenet dataset)
Flatten layer flattens the input. Does not affect the batch size.



Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. Let us create a model object named model with inputs as VGG19.input and output as dense layer. The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities. Understanding the model is a very important phase to properly use it for training and prediction purposes.
Keras provides a simple method, summary to get the full information about the model and its layers.



Configure the Learning Process

The compilation is the final step in creating a model. Once the compilation is done,

we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Train the model

Now, let us train our model with our image dataset. The model is trained for 50 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch and probably there is further scope to improve the model.
.fit functions used to train a deep learning neural network
Arguments:
steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
Epochs: an integer and number of epochs we want to train our model for.
validation_data can be either:
- an inputs and targets list
- a generator
- an inputs, targets, and sample_weights list which can be used to
evaluate the loss and metrics for any model after any epoch has ended.
validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.
From the above run time, we can observe that at 50th epoch the model is giving the best accuracy.

Save the Model

The model is saved with .h5 extension as follows.
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

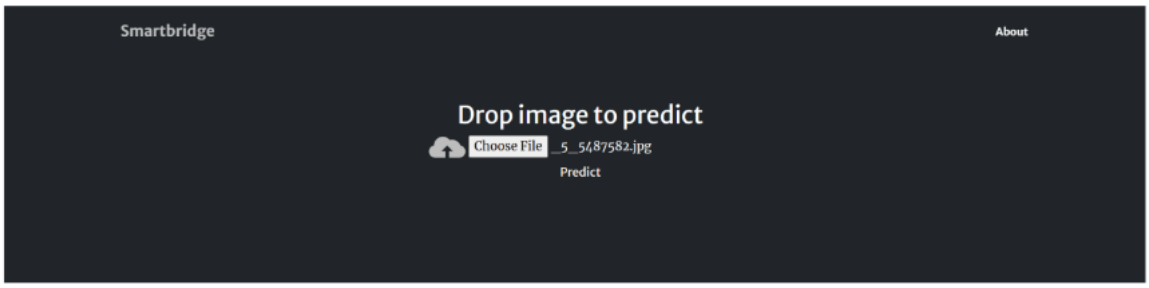
Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks
Building HTML Pages
Building server side script

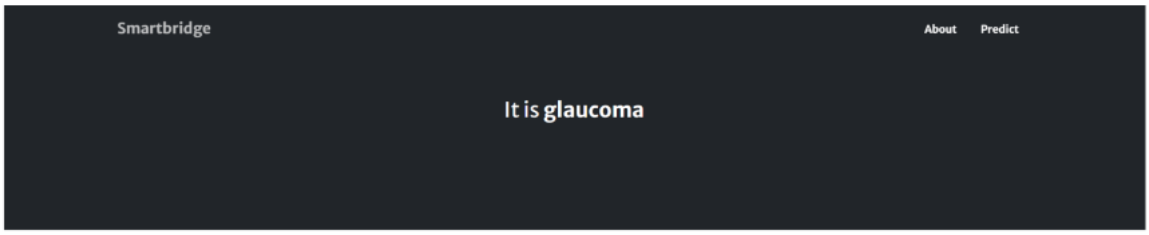
Run the application

Open Spyder
Navigate to the folder where your Python script is.
Now click on the green play button above.
Click on the predict button from the top right corner, enter the inputs, click on the Classify button, and see the result/prediction on the web.
The home page looks like this. When you click on the Predict button, you'll be redirected to the predict section

Input 2:



Output2:



```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
```

+ Code + Markdown

Add Markdown Cell

```
import os
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the VGG19 model (without top layers)
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze convolutional layers (feature extractor)
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification layers
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = Dense(4, activation='softmax')(x)  # 4 classes

# Create the model
model = Model(inputs=base_model.input, outputs=x)

# Compile model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

print(model.summary())
```

```python
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
# Training data
train_generator = datagen.flow_from_directory(
    r'C:\Users\LENOVO\Downloads\archive\dataset',
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical',
    subset='training'
)
# Validation data
val_generator = datagen.flow_from_directory(
    r'C:\Users\LENOVO\Downloads\archive\dataset',
    target_size=(224, 224),
    batch_size=64,
    class_mode='categorical',
    subset='validation'
)
```

```
Found 3376 images belonging to 4 classes.
Found 841 images belonging to 4 classes.
```

Found 3376 images belonging to 4 classes.
Found 841 images belonging to 4 classes.

```python
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs= 5 # Change as needed
)
```

[10]

```
Epoch 1/5
53/53 ━━━━━━━━━━━━━━━━━━━━ 830s 16s/step - accuracy: 0.7266 - loss: 0.7006 - val_accuracy: 0.6314 - val_loss: 0.8698
Epoch 2/5
53/53 ━━━━━━━━━━━━━━━━━━━━ 810s 15s/step - accuracy: 0.8644 - loss: 0.3784 - val_accuracy: 0.7812 - val_loss: 0.5281
Epoch 3/5
53/53 ━━━━━━━━━━━━━━━━━━━━ 809s 15s/step - accuracy: 0.8649 - loss: 0.3420 - val_accuracy: 0.7574 - val_loss: 0.5492
Epoch 4/5
53/53 ━━━━━━━━━━━━━━━━━━━━ 806s 15s/step - accuracy: 0.8964 - loss: 0.2719 - val_accuracy: 0.7883 - val_loss: 0.5070
Epoch 5/5
53/53 ━━━━━━━━━━━━━━━━━━━━ 1853s 35s/step - accuracy: 0.9035 - loss: 0.2660 - val_accuracy: 0.8145 - val_loss: 0.4636
```

```python
model.save('best_model.keras')  # New recommended format
```

[12]

```python
import numpy as np
from tensorflow.keras.preprocessing import image

# Load the trained model
model = load_model('best_model.keras')

# Load a single image for testing
img_path = r'C:\Users\LENOVO\Downloads\archive\dataset\glaucoma\_75_820784.jpg'  # Change this
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0  # Normalize

# Make prediction
predictions = model.predict(img_array)
class_names = ['cataract', 'Diabetic Retinopathy', 'Glaucoma', 'normal']
predicted_class = class_names[np.argmax(predictions)]

print(f"Predicted Class: {predicted_class}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 353ms/step
Predicted Class: Glaucoma
```