```python
# Core libraries essential for data manipulation, analysis, training,
exploration, and others
import os
import cv2
import gdown
import zipfile
import pandas as pd
import numpy as np
import seaborn as sns
import tensorflow as tf
import albumentations as A
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.utils import resample
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.utils import Sequence
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import layers, models, callbacks
from tensorflow.keras.regularizers import l2
from albumentations.core.transforms_interface import DualTransform

print(tf.__version__)
```

```
2025-03-19 19:44:54.310968: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:467] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1742438694.353348     391 cuda_dnn.cc:8579] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1742438694.362098     391 cuda_blas.cc:1407] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
W0000 00:00:1742438694.406488     391 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1742438694.406553     391 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1742438694.406555     391 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
W0000 00:00:1742438694.406557     391 computation_placer.cc:177]
computation placer already registered. Please check linkage and avoid
linking the same target more than once.
2025-03-19 19:44:54.414449: I
```

```
tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.

2.19.0
```

First, we check to confirm if the ocular disease dataset currently exists in our specified location and proceed forward to either download the file or move onto the next chunk of code.

```
# Google Drive File ID with filename zip file
file_id = "1oQ6Vy_HqZlVHnkFspgxMn0IcE__D8Kmh"
zip_filename = "ocular-disease-recognition.zip"
extract_path = "./ocular-disease-recognition"

# Checking if the file already exists
if not os.path.exists(zip_filename):
    print(f"Downloading {zip_filename}...")
    gdown.download(f"https://drive.google.com/uc?id={file_id}",
zip_filename, quiet=False)
else:
    print(f"{zip_filename} already exists. Skipping download.")

ocular-disease-recognition.zip already exists. Skipping download.
```

This is a repeated function handling the extraction of the zip file. checking if the dataset has already been unzipped to the defined folder, it extracts all files from the zip archive to our directory to preprocess and manipulate.

```
# Check if dataset extraction file exists
if not os.path.exists(extract_path):
    os.makedirs(extract_path, exist_ok=True)
    print(f"Extracting {zip_filename}...")
    with zipfile.ZipFile(zip_filename, "r") as zip_ref:
        zip_ref.extractall(extract_path)
    print(f"Extraction complete! Files extracted to: {extract_path}")
else:
    print(f"Extraction skipped: {extract_path} already exists.")

Extraction skipped: ./ocular-disease-recognition already exists.
```

Next, we load in the dataset from a csv file into a DataFrame and apply a filter to ensure that each record has both left and right fundus images available. Given the dual input nature of our model, this is essential to provide long term results. After filtering, we split the dataset into training, validation, and tests sets for model training and evaluation down the pipeline.

```
# Load the dataset, update the path for local machine
dataset_path = "processed_ocular_disease.csv"
```

```
# Read in CSV file to dataframe
df = pd.read_csv(dataset_path)

# Filter dataset for only instances having both left and right fundus
images present
df = df[
    df.apply(lambda row:

os.path.exists(os.path.join('ocular-disease-recognition/preprocessed_i
mages', row['Left-Fundus'])) and

os.path.exists(os.path.join('ocular-disease-recognition/preprocessed_i
mages', row['Right-Fundus'])),
        axis=1
    )
].reset_index(drop=True)

# Dataset split into training and temporary datasets for training and
validation
# 70% Training, 30% Temporary
train_df, temp_df = train_test_split(df, test_size=0.3,
random_state=42)
# 15% Validation, 15% Testing
val_df, test_df = train_test_split(temp_df, test_size=0.5,
random_state=42)
```

Considering the large data imbalance present in our dataset, we approach the problem with class weights via SKLearn. The weights are calculated in order to counteract any imbalance in the dataset for training stability and classification ability of our model

```
# Calculate number of unique class labels
num_classes = len(np.unique(df['labels']))
print(f"Number of Classes: {num_classes}")

# Compute class weights for class imbalance
class_labels = np.unique(df['labels'])
class_weights = compute_class_weight(class_weight="balanced",
classes=class_labels, y=df['labels'])
class_weight_dict = {i: class_weights[i] for i in
range(len(class_labels))}
print("Computed Class Weights:", class_weight_dict)
```

```
Number of Classes: 8
Computed Class Weights: {0: np.float64(2.917307692307692), 1:
np.float64(2.718637992831541), 2: np.float64(0.49285250162443145), 3:
np.float64(2.7783882783882783), 4: np.float64(6.01984126984127), 5:
np.float64(3.77363184079602), 6: np.float64(0.2791682002208318), 7:
np.float64(1.1270430906389302)}
```

We continue with preprocessing by rebalancing the training dataset, utilizing resampling with replacement on each class so that all classes have an equal number of samples. The balanced dataset is shuffled with counts of each label printing to very class balances.

```
# function to rebalance classes (?)
def balance_classes(df):
    """Resample dataset to balance classes."""
    max_size = df['labels'].value_counts().max()
    balanced_df = pd.concat([
        resample(df[df['labels'] == cls], replace=True,
n_samples=max_size, random_state=42)
        for cls in df['labels'].unique()
    ])
    # Shuffle after resampling.
    return balanced_df.sample(frac=1).reset_index(drop=True)

# Apply to training data only
train_df_balanced = balance_classes(train_df)

# Check if balancing worked
print(train_df_balanced['labels'].value_counts())  # Should now be
balanced
```

```
labels
3    1901
0    1901
6    1901
2    1901
7    1901
4    1901
5    1901
1    1901
Name: count, dtype: int64
```

Moving along, we create a custom data pipeline to handle paired image data and augmentations with our DualImageAugmentation class to ensure that both images per instance undergo the same random transformations. Alongside this, a customd ata generator using Keras Sequence class was used to load images and apply augmentations, merging the two resulting greyscale images into a 2-channel input and outputting batches of data alongwith their corresponding labels.

```
# Custom dual augmentation class inheringt albumentations
class DualImageAugmentation(DualTransform):
    def __init__(self, transforms, always_apply=False, p=0.5):
        # Initialize parent class store augmentation pipeline
        super(DualImageAugmentation, self).__init__(always_apply, p)
        self.transforms = A.Compose(transforms)

    def apply(self, img, **params):
        # Apply pipeline to image and return transformations
```

```
        return self.transforms(image=img)["image"]

    def apply_to_image1(self, img, **params):
        # Specifically apply same transformation to second paired
image
        return self.transforms(image=img)["image"]

# Ocular data generator
class OcularDatasetGenerator(Sequence):
    def __init__(self, df, batch_size=32, img_size=(128, 128),
shuffle=True, augment=True, **kwargs):
        super().__init__(**kwargs)

        # Filter dataframe to include only rows with existing left and
right image pairs
        self.df = df[df.apply(lambda row:

os.path.exists(os.path.join('ocular-disease-recognition/preprocessed_i
mages', row['Left-Fundus'])) and

os.path.exists(os.path.join('ocular-disease-recognition/preprocessed_i
mages', row['Right-Fundus'])),
            axis=1
        )].reset_index(drop=True)  # Reset index after filtering
        print(f"Dataset initialized with {len(self.df)} valid
samples.")


        # Set class attributed based on the provided parameters
        self.batch_size = batch_size
        self.img_size = img_size
        self.shuffle = shuffle
        self.augment = augment
        self.indices = np.arange(len(df))

        # Define augmentation pipeline if augmentation is enabled
        if augment:
            self.augmentation_pipeline =
self.get_augmentation_pipeline()
        else:
            self.augmentation_pipeline = None

        self.on_epoch_end()

    def __len__(self):
        # Get indices for the current batch based on batch size
        return int(np.floor(len(self.df) / self.batch_size))

    def __getitem__(self, index):
        batch_indices = self.indices[index * self.batch_size:(index +
```

```
1) * self.batch_size]
        # SElect corresponding rows
        batch = self.df.iloc[batch_indices]
        # Generate batch data for images and labels
        X, y = self.__data_generation(batch)
        return np.array(X), np.array(y)

    def __data_generation(self, batch):
        # Initialize empty lists to accumulate batch images and labels
        X_batch = []
        y_batch = []

        # Iterate over each row
        for _, row in batch.iterrows():
            # Construct full file paths for left and right fundus
images
            left_image_path = os.path.join('ocular-disease-
recognition/preprocessed_images', row['Left-Fundus'])
            right_image_path = os.path.join('ocular-disease-
recognition/preprocessed_images', row['Right-Fundus'])

            # Load images using helper method
            left_image = self.load_image(left_image_path)
            right_image = self.load_image(right_image_path)

            # Skip iteration if either image could not be loaded
            if left_image is None or right_image is None:
                continue  # Skip invalid images

            # Apply augmentation (both images get the same
transformation)
            if self.augment and self.augmentation_pipeline:
                augmented =
self.augmentation_pipeline(image=left_image, image1=right_image)
                left_image = augmented["image"]
                right_image = augmented["image1"]

            # Convert grayscale images to 3D (required for CNN)
            left_image = np.expand_dims(left_image, axis=-1)
            right_image = np.expand_dims(right_image, axis=-1)

            # Merge images into a two-channel input
            combined_image = np.concatenate((left_image, right_image),
axis=-1)

            X_batch.append(combined_image)
            y_batch.append(int(row['labels']))

        return np.array(X_batch, dtype=np.float32), np.array(y_batch,
dtype=np.int32)
```

```
    def load_image(self, image_path):
        # Read image in grayscale via CV2
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if image is None:
            return None
        # Resize image to target dimensions
        image = cv2.resize(image, self.img_size)
        # normalize pixel values to 0, 1 range
        image = image / 255.0
        return image

    def get_augmentation_pipeline(self):
        # Define a composition of augmentation transofrmations
        return A.Compose([
            A.RandomBrightnessContrast(p=0.5),
            A.GaussianBlur(blur_limit=(3, 7), p=0.4),
            A.HorizontalFlip(p=0.5),
            A.Rotate(limit=30, p=0.5),
            A.ElasticTransform(p=0.5),
            A.CoarseDropout(max_holes=3, max_height=0.2,
max_width=0.2, p=0.5),
        ], additional_targets={"image1": "image"})

    def on_epoch_end(self):
        """ Shuffle indices at the end of each epoch. """
        self.indices = np.arange(len(self.df))  # Ensure indices match
filtered dataset
        if self.shuffle:
            np.random.shuffle(self.indices)
```

Moving forward, we now compile and build our model on 64 batch sizing across 237 epochs, setting our metric of interest as accuracy.

```
# **Create Data Generators**
batch_size = 64
train_generator = OcularDatasetGenerator(train_df_balanced,
batch_size=batch_size, img_size=(224, 224), augment=True)
val_generator = OcularDatasetGenerator(val_df, batch_size=batch_size,
img_size=(224, 224))
test_generator = OcularDatasetGenerator(test_df,
batch_size=batch_size, shuffle=False, img_size=(224, 224))


# Build a sequential convolutional NN
model = models.Sequential([
    layers.Input(shape=(224, 224, 2)),
```

```
    # Convolutional block
    layers.Conv2D(32, (3, 3), activation='relu', padding='same'),
    # Batch normalization
    layers.BatchNormalization(),
    # Downsampling feature maps
    layers.MaxPooling2D((2, 2)),
    # Dropout 10% of nodes
    layers.Dropout(0.1),

    # Convolutional block 2
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.2),

    #Convolutional block 3 with increased filters
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.2),

    # Convolutional block 4: Increasing filters to 256
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    # Convolutional block 5: filter increased to 512
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.4),

    # Flatten output to 1D vector
    layers.Flatten(),
    # Dense layer with 512 units and ReLU activation
    layers.Dense(512, activation='relu'),
    layers.BatchNormalization(),
    # Highdropout of 50%
    layers.Dropout(0.5),

    layers.Dense(num_classes, activation='softmax')
])

# Defined learning rate scheduler for step decay
def step_decay(epoch):
    initial_lr = 0.001
    drop = 0.5
    epochs_drop = 5
    return initial_lr * (drop ** (epoch // epochs_drop))
```

```python
# Another learning rate scheduler with coside decay
def cosine_decay(epoch):
    initial_lr = 0.001
    return initial_lr * (0.5 * (1 + np.cos(np.pi * epoch / 100)))


lr_schedule = tf.keras.callbacks.LearningRateScheduler(cosine_decay)

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='sparse_categorical_crossentropy',  # ✅ Use sparse version
    metrics=['accuracy']
)


# **Define Early Stopping Callback**
early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',    # Stop if validation loss stops improving
    patience=8,            # Wait for 5 epochs before stopping
    restore_best_weights=True  # Restore best model weights
)

# **Train the Model with Early Stopping**
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=val_generator,
    callbacks=[early_stopping, lr_schedule],
)
```

```
Dataset initialized with 15208 valid samples.
Dataset initialized with 910 valid samples.
Dataset initialized with 911 valid samples.

/tmp/ipykernel_391/3805339739.py:165: UserWarning: Argument(s)
'max_holes, max_height, max_width' are not valid for transform
CoarseDropout
  A.CoarseDropout(max_holes=3, max_height=0.2, max_width=0.2, p=0.5),
I0000 00:00:1742438785.377546     391 gpu_device.cc:2019] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 8847 MB
memory:  -> device: 0, name: NVIDIA GeForce RTX 3080, pci bus id:
0000:06:00.0, compute capability: 8.6

Epoch 1/100

WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
I0000 00:00:1742438793.306045     752 service.cc:152] XLA service
0x7f74a4004d20 initialized for platform CUDA (this does not guarantee
that XLA will be used). Devices:
```

```
I0000 00:00:1742438793.306183     752 service.cc:160]   StreamExecutor
device (0): NVIDIA GeForce RTX 3080, Compute Capability 8.6
2025-03-19 19:46:33.430513: I
tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269]
disabling MLIR crash reproducer, set env var
`MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
I0000 00:00:1742438794.097146     752 cuda_dnn.cc:529] Loaded cuDNN
version 90300
2025-03-19 19:46:35.293984: I
external/local_xla/xla/stream_executor/cuda/subprocess_compilation.cc:
346] ptxas warning : Registers are spilled to local memory in function
'gemm_fusion_dot_4658', 4 bytes spill stores, 4 bytes spill loads


  1/237 ━━━━━━━━━━━━━━━━━━━━ 1:07:55 17s/step - accuracy: 0.0938 -
loss: 3.3140

I0000 00:00:1742438806.206409     752 device_compiler.h:188] Compiled
cluster using XLA!  This line is logged at most once for the lifetime
of the process.

237/237 ━━━━━━━━━━━━━━━━━━━━ 77s 254ms/step - accuracy: 0.2376 - loss:
2.4636 - val_accuracy: 0.0949 - val_loss: 2.7261 - learning_rate:
0.0010
Epoch 2/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 56s 235ms/step - accuracy: 0.3270 - loss:
1.9466 - val_accuracy: 0.0525 - val_loss: 3.6943 - learning_rate:
9.9975e-04
Epoch 3/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 54s 226ms/step - accuracy: 0.3961 - loss:
1.6655 - val_accuracy: 0.1908 - val_loss: 1.9448 - learning_rate:
9.9901e-04
Epoch 4/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 56s 237ms/step - accuracy: 0.4660 - loss:
1.4428 - val_accuracy: 0.2377 - val_loss: 1.7995 - learning_rate:
9.9778e-04
Epoch 5/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 220ms/step - accuracy: 0.5132 - loss:
1.3165 - val_accuracy: 0.2589 - val_loss: 1.7174 - learning_rate:
9.9606e-04
Epoch 6/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 217ms/step - accuracy: 0.5481 - loss:
1.2080 - val_accuracy: 0.3147 - val_loss: 1.6692 - learning_rate:
9.9384e-04
Epoch 7/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 217ms/step - accuracy: 0.5937 - loss:
1.0974 - val_accuracy: 0.2924 - val_loss: 1.7421 - learning_rate:
9.9114e-04
Epoch 8/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 220ms/step - accuracy: 0.6170 - loss:
```

```
1.0391 - val_accuracy: 0.2455 - val_loss: 1.9066 - learning_rate:
9.8796e-04
Epoch 9/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 219ms/step - accuracy: 0.6100 - loss:
1.0706 - val_accuracy: 0.3013 - val_loss: 1.6941 - learning_rate:
9.8429e-04
Epoch 10/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 55s 232ms/step - accuracy: 0.6457 - loss:
0.9456 - val_accuracy: 0.3359 - val_loss: 1.5632 - learning_rate:
9.8015e-04
Epoch 11/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 63s 265ms/step - accuracy: 0.6786 - loss:
0.8730 - val_accuracy: 0.3170 - val_loss: 1.6493 - learning_rate:
9.7553e-04
Epoch 12/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 60s 252ms/step - accuracy: 0.7170 - loss:
0.7669 - val_accuracy: 0.4007 - val_loss: 1.4947 - learning_rate:
9.7044e-04
Epoch 13/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 53s 223ms/step - accuracy: 0.7205 - loss:
0.7412 - val_accuracy: 0.3917 - val_loss: 1.5634 - learning_rate:
9.6489e-04
Epoch 14/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 221ms/step - accuracy: 0.7232 - loss:
0.7505 - val_accuracy: 0.4208 - val_loss: 1.4743 - learning_rate:
9.5888e-04
Epoch 15/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 219ms/step - accuracy: 0.7336 - loss:
0.7126 - val_accuracy: 0.3962 - val_loss: 1.6253 - learning_rate:
9.5241e-04
Epoch 16/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 220ms/step - accuracy: 0.7625 - loss:
0.6419 - val_accuracy: 0.3951 - val_loss: 1.5768 - learning_rate:
9.4550e-04
Epoch 17/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 220ms/step - accuracy: 0.7524 - loss:
0.6642 - val_accuracy: 0.4475 - val_loss: 1.5106 - learning_rate:
9.3815e-04
Epoch 18/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 219ms/step - accuracy: 0.7656 - loss:
0.6384 - val_accuracy: 0.4118 - val_loss: 1.5936 - learning_rate:
9.3037e-04
Epoch 19/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 53s 223ms/step - accuracy: 0.7845 - loss:
0.5806 - val_accuracy: 0.3973 - val_loss: 1.6955 - learning_rate:
9.2216e-04
Epoch 20/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 51s 217ms/step - accuracy: 0.7917 - loss:
0.5803 - val_accuracy: 0.4420 - val_loss: 1.5719 - learning_rate:
```

```
9.1354e-04
Epoch 21/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 52s 218ms/step - accuracy: 0.8120 - loss:
0.5151 - val_accuracy: 0.4531 - val_loss: 1.5709 - learning_rate:
9.0451e-04
Epoch 22/100
237/237 ━━━━━━━━━━━━━━━━━━━━ 51s 216ms/step - accuracy: 0.8060 - loss:
0.5348 - val_accuracy: 0.4754 - val_loss: 1.4930 - learning_rate:
8.9508e-04
```

```
# **Evaluate the Model on Test Set**
test_loss, test_acc = model.evaluate(test_generator)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 3s 217ms/step - accuracy: 0.4259 - loss:
1.4151
Test Accuracy: 0.4241
```

Despite best attempts, the model only attained 42% accuracy in this iteration, with a high of 47% accuracy in previous training attempts. However, this proves to be an insufficient approach, from which we developed our final model architecture present in ResNetMultiModel_v4.ipynb notebook instead.