```
!pip install kaggle
!mkdir -p ~/.kaggle
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from k
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kag
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kag
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

```

```
!kaggle datasets download -d ruhulaminsharif/eye-disease-image-dataset -p /content

```

```
Dataset URL: https://www.kaggle.com/datasets/ruhulaminsharif/eye-disease-image-dataset
License(s): Attribution 4.0 International (CC BY 4.0)
```

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split

# Step 1: Set the base path to the dataset
base_path = '/content/Original_Dataset/Original_Dataset'  # Update this path accor

# Step 2: List all image paths and their corresponding labels
```

```
classes, paths = [], []
for label in os.listdir(base_path):
    class_dir = os.path.join(base_path, label)
    if os.path.isdir(class_dir):
        for fname in os.listdir(class_dir):
            if fname.lower().endswith(('.png', '.jpg', '.jpeg')):
                classes.append(label)
                paths.append(os.path.join(class_dir, fname))

# Step 3: Create a DataFrame
df = pd.DataFrame({'Class': classes, 'Path': paths})

# Step 4: Split the data into training, validation, and testing sets
train_df, temp_df = train_test_split(df, test_size=0.3, stratify=df['Class'], rand
valid_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['Cla

print(f"Training samples: {len(train_df)}")
print(f"Validation samples: {len(valid_df)}")
print(f"Testing samples: {len(test_df)}")

```

```
Training samples: 3734
Validation samples: 800
Testing samples: 801
```

```
df
```

	Class	Path
0	Retinitis Pigmentosa	/content/Original_Dataset/Original_Dataset/Ret...
1	Retinitis Pigmentosa	/content/Original_Dataset/Original_Dataset/Ret...
2	Retinitis Pigmentosa	/content/Original_Dataset/Original_Dataset/Ret...
3	Retinitis Pigmentosa	/content/Original_Dataset/Original_Dataset/Ret...
4	Retinitis Pigmentosa	/content/Original_Dataset/Original_Dataset/Ret...
...	...	...
5330	Central Serous Chorioretinopathy-Color Fundus	/content/Original_Dataset/Original_Dataset/Cen...
5331	Central Serous Chorioretinopathy-Color Fundus	/content/Original_Dataset/Original_Dataset/Cen...
5332	Central Serous Chorioretinopathy-Color Fundus	/content/Original_Dataset/Original_Dataset/Cen...
5333	Central Serous Chorioretinopathy-Color Fundus	/content/Original_Dataset/Original_Dataset/Cen...
5334	Central Serous Chorioretinopathy-Color Fundus	/content/Original_Dataset/Original_Dataset/Cen...

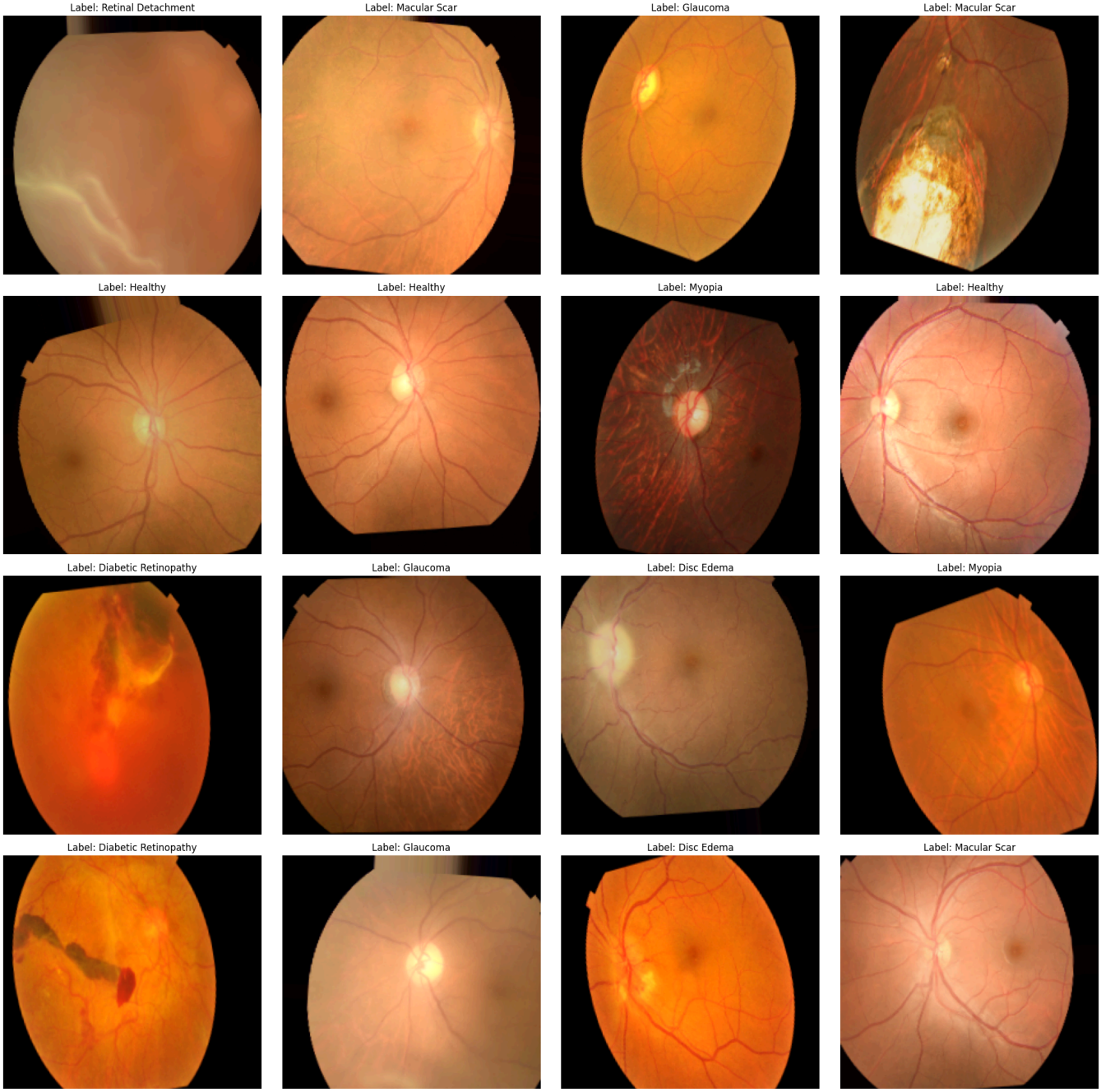
5335 rows × 2 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
#first 16 img
tr_gen = make_generator(train_df, img_size=(224, 224), batch_size=32, au
import matplotlib.pyplot as plt
import numpy as np

# Fetch a batch of images and labels
batch = next(tr_gen)
images, labels = batch[0], batch[1]

# Get class names from the generator
class_names = list(tr_gen.class_indices.keys())

# Plot the images in a 4x4 grid
plt.figure(figsize=(20, 20))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.imshow(images[i])
    class_index = np.argmax(labels[i])
    class_name = class_names[class_index]
    plt.title(f"Label: {class_name}")
```

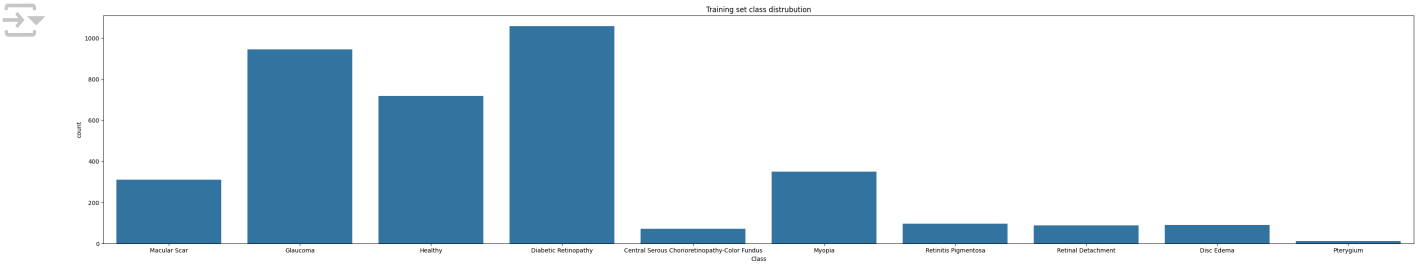
```
    plt.axis('off')
plt.tight_layout()
plt.show()

```

```
Found 3734 validated image filenames belonging to 10 classes.
```




```
import seaborn as sns

plt.figure(figsize=(40,7))
plt.title('Training set class distrubution')
#sns.countplot() is a function that creates a bar
#chart showing the counts of different categories in a dataset.
#data=tr_df specifies that the data for the plot should be taken
# from the Pandas DataFrame called tr_df
#x=tr_df['Class'] tells the function to use the values in the 'Class' column of tr
```

```
# for the categories on the x-axis of the plot.
ax=sns.countplot(data=train_df,x=train_df['Class'])
#this dataset is pretty balanced
```




```
import seaborn as sns

plt.figure(figsize=(40,7))
plt.title('Training set class distrubution')
#sns.countplot() is a function that creates a bar
#chart showing the counts of different categories in a dataset.
#data=tr_df specifies that the data for the plot should be taken
# from the Pandas DataFrame called tr_df
#x=tr_df['Class'] tells the function to use the values in the 'Class' column of tr
# for the categories on the x-axis of the plot.
ax=sns.countplot(data=test_df,x=test_df['Class'])
#this dataset is pretty balanced
```


```
# 3. ImageDataGenerators
def make_generator(df, img_size, batch_size=32, augment=False):
    if augment:
        datagen = ImageDataGenerator(
            rescale=1/255,
            rotation_range=20,
            width_shift_range=0.1,
            height_shift_range=0.1,
            brightness_range=(0.8,1.2),
            horizontal_flip=True
        )
    else:
        datagen = ImageDataGenerator(rescale=1/255)
    return datagen.flow_from_dataframe(
        df, x_col="Path", y_col="Class",
        target_size=img_size, batch_size=batch_size,
```

```
        class_mode="categorical", shuffle=augment
    )
```

```
from tensorflow.keras.applications import (
    VGG16, ResNet50, InceptionV3,
    vgg16, resnet50, inception_v3
)
from tensorflow.keras.layers import (
    GlobalAveragePooling2D, Dense, Dropout
)
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
def build_model(base_cls, input_shape, num_classes):
    base = base_cls(
        include_top=False,
        weights="imagenet",
        input_shape=input_shape,
        pooling=None
    )
    x = base.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(512, activation="relu")(x)
    x = Dropout(0.3)(x)
    outputs = Dense(num_classes, activation="softmax")(x)
    model = Model(inputs=base.input, outputs=outputs, name=base_cls.__name__)
    model.compile(
        optimizer=Adam(learning_rate=1e-4),
        loss="categorical_crossentropy",
        metrics=["accuracy"]
    )
    return model
```

```
# Settings
NUM_CLASSES = train_df["Class"].nunique()
EPOCHS = 10
BATCH_SIZE = 32

# for each model input size
gens = {
    "VGG16": {
        "train": make_generator(train_df, img_size=(224,224), batch_size=
        "valid": make_generator(valid_df, img_size=(224,224), batch_size=
        "test" : make_generator(test_df,  img_size=(224,224), batch_size=
```

```
    },
    "ResNet50": {
        "train": make_generator(train_df, img_size=(224,224), batch_size=
        "valid": make_generator(valid_df, img_size=(224,224), batch_size=
        "test" : make_generator(test_df,  img_size=(224,224), batch_size=
    },
    "InceptionV3": {
        "train": make_generator(train_df, img_size=(299,299), batch_size=
        "valid": make_generator(valid_df, img_size=(299,299), batch_size=
        "test" : make_generator(test_df,  img_size=(299,299), batch_size=
    },
}

# Build & train each model
histories = {}
for name, cls in [("VGG16", VGG16), ("ResNet50", ResNet50), ("InceptionV3
    print(f"\n=== Training {name} ===")
    model = build_model(cls, gens[name]["train"].image_shape, NUM_CLASSES
    histories[name] = model.fit(
        gens[name]["train"],
        validation_data=gens[name]["valid"],
        epochs=EPOCHS
    )
    model.save(f"{name}_eye_disease.h5")

```

```
Epoch 9/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 119s 1s/step - accuracy: 0.8365 - loss: 0.4226 - val_acc
Epoch 10/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 120s 1s/step - accuracy: 0.8446 - loss: 0.3571 - val_acc
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.s

=== Training InceptionV3 ===
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/in
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 323s 2s/step - accuracy: 0.5385 - loss: 1.3665 - val_acc
Epoch 2/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 171s 1s/step - accuracy: 0.7478 - loss: 0.7262 - val_acc
Epoch 3/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 164s 1s/step - accuracy: 0.7691 - loss: 0.6139 - val_acc
Epoch 4/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 164s 1s/step - accuracy: 0.8032 - loss: 0.5387 - val_acc
Epoch 5/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 168s 1s/step - accuracy: 0.8264 - loss: 0.4631 - val_acc
Epoch 6/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 164s 1s/step - accuracy: 0.8477 - loss: 0.4187 - val_acc
Epoch 7/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 165s 1s/step - accuracy: 0.8595 - loss: 0.3767 - val_acc
Epoch 8/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 168s 1s/step - accuracy: 0.8503 - loss: 0.3837 - val_acc
Epoch 9/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 168s 1s/step - accuracy: 0.8628 - loss: 0.3494 - val_acc
Epoch 10/10
117/117 ━━━━━━━━━━━━━━━━━━━━ 161s 1s/step - accuracy: 0.8661 - loss: 0.3282 - val_acc
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.s
```

```python
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.models import load_model

results = {}
for name in ["VGG16", "ResNet50", "InceptionV3"]:
    print(f"\n--- Evaluating {name} ---")
    model = load_model(f"{name}_eye_disease.h5")
    test_gen = gens[name]["test"]
    preds = model.predict(test_gen, verbose=1)
    y_true = test_gen.classes
    y_pred = np.argmax(preds, axis=1)

    # Metrics
    print(classification_report(
        y_true, y_pred, target_names=list(test_gen.class_indices.keys())
    ))
    cm = confusion_matrix(y_true, y_pred)
    results[name] = {
        "accuracy": np.mean(y_true == y_pred),
        "confusion_matrix": cm
```

```
    }

```

```
                                   Healthy       0.69      0.90      0.78        15
                                Macular Scar     0.63      0.64      0.63         6
                                     Myopia      0.63      0.67      0.65         7
                                  Pterygium      1.00      1.00      1.00
                           Retinal Detachment    1.00      0.68      0.81         1
                          Retinitis Pigmentosa   0.94      0.76      0.84         2

                                    accuracy                         0.78        80
                                   macro avg      0.85      0.71      0.73        80
                                weighted avg      0.79      0.78      0.77        80


--- Evaluating ResNet50 ---
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_a
  self._warn_if_super_not_called()
26/26 ━━━━━━━━━━━━━━━━━━━━ 20s 623ms/step
                                              precision    recall  f1-score   suppor

Central Serous Chorioretinopathy-Color Fundus       0.82      0.60      0.69         1
                        Diabetic Retinopathy        0.90      0.94      0.92        22
                                   Disc Edema       0.92      0.63      0.75         1
                                     Glaucoma       0.70      0.72      0.71        20
                                      Healthy       0.66      0.79      0.72        15
                                   Macular Scar     0.62      0.58      0.60         6
                                        Myopia      0.71      0.43      0.53         7
                                     Pterygium      1.00      1.00      1.00
                           Retinal Detachment       0.95      1.00      0.97         1
                          Retinitis Pigmentosa      0.84      0.76      0.80         2

                                    accuracy                         0.76        80
                                   macro avg      0.81      0.74      0.77        80
                                weighted avg      0.76      0.76      0.75        80


--- Evaluating InceptionV3 ---
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_a
  self._warn_if_super_not_called()
26/26 ━━━━━━━━━━━━━━━━━━━━ 26s 792ms/step
                                              precision    recall  f1-score   suppor

Central Serous Chorioretinopathy-Color Fundus       0.67      0.53      0.59         1
```

accuracy			0.76	80
macro avg	0.79	0.74	0.76	80
weighted avg	0.76	0.76	0.75	80

```
# Summarize test accuracies
for name, info in results.items():
    print(f"{name:10s} Test Accuracy: {info['accuracy']*100:.2f}%")

```

```
VGG16      Test Accuracy: 77.53%
ResNet50   Test Accuracy: 76.03%
InceptionV3 Test Accuracy: 75.66%
```

```
# # Compare the performance of the different models. Which one performed
# VGG16: 77.53%
# ResNet50: 76.03%
# InceptionV3: 75.66%

# here VGG16 got the highest test accuracy among the three models
# VGG16  test accuracy was well suited for this particular dataset.Becaus
# 1.VGG16's straightforward architecture may have allowed it to generaliz
# 2.Its deep convolutional layers are adept at capturing hierarchical fea

# ResNet50 and InceptionV3 are more complex architectures designed to cap
# VGG16 outperformed ResNet50 and InceptionV3 in terms of test accuracy.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```