```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Import Necessary Libraries

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

Load CSV file containing image labels (DR or Not DR)

```
# Load CSV file containing image labels
df_labels = pd.read_csv('/content/drive/MyDrive/labels_mbrset.csv')


# Convert DR labels to integers for class weight computation
# If final_icdr is 0, it's 'Not DR' (0), otherwise, it's 'DR' (1)
df_labels['dr_diagnosis_int'] = df_labels['final_icdr'].apply(lambda x: 0 if x == 0 else 1)


# Display the transformed labels
print(df_labels[['file', 'final_icdr', 'dr_diagnosis_int']].head())
```

```
      file  final_icdr  dr_diagnosis_int
0   1.1.jpg        4.0                 1
1   1.2.jpg        4.0                 1
2   1.3.jpg        4.0                 1
3   1.4.jpg        4.0                 1
4  10.1.jpg        0.0                 0
```

Define Paths and Parameters

```
# Define paths and parameters
IMG_SIZE = (299, 299)
BATCH_SIZE = 32
TRAIN_PATH = '/content/drive/MyDrive/images'
```

Data Augmentation and Generator Setup

```
# ImageDataGenerator for training and validation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # Use 20% of data for validation
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

Load training and validation data with binary targets (DR or Not DR)

```
# Load training and validation data with binary targets (DR or Not DR)
train_generator = train_datagen.flow_from_dataframe(
    dataframe=df_labels,
    directory=TRAIN_PATH,
    x_col='file',
    y_col='dr_diagnosis_int',  # Single binary output (DR or Not DR)
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='raw',
    subset='training',
    shuffle=True
)
```

```
Found 4132 validated image filenames.
```

```
valid_generator = train_datagen.flow_from_dataframe(
    dataframe=df_labels,
    directory=TRAIN_PATH,
    x_col='file',
    y_col='dr_diagnosis_int',
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='raw',
    subset='validation',
    shuffle=True
)
```

```
Found 1032 validated image filenames.
```

Create the binary classification model

```
# Step 2: Create the binary classification model
input_tensor = tf.keras.Input(shape=(299, 299, 3))


# Load pre-trained InceptionV3 without top layers
base_model = InceptionV3(weights=None, include_top=False, input_tensor=input_tensor)
base_model.load_weights('/content/drive/MyDrive/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5')



# Shared layers (the base of the model)
x = base_model.output
x = BatchNormalization()(x)
x = GlobalAveragePooling2D()(x)


# Output layer for binary classification (DR vs Not DR)
dr_output = Dense(1024, activation='relu')(x)
dr_output = Dense(1, activation='sigmoid', name='dr_output')(dr_output)



# Final model with a single output
model = Model(inputs=base_model.input, outputs=dr_output)


# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False
```

Handle Class Imbalance by Computing Class Weights

```
# Compute class weights from the original labels in the DataFrame
class_weights = compute_class_weight(
    'balanced',
    classes=np.unique(df_labels['dr_diagnosis_int']),
    y=df_labels['dr_diagnosis_int']
)
class_weight_dict = dict(enumerate(class_weights))
```

```
# Display class weights for debugging
print(f'Class weights: {class_weight_dict}')
```

```
Class weights: {0: 0.6885333333333333, 1: 1.8260254596888261}
```

```
# Compile the model
model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Train the Binary Classification Model

```
# Step 4: Train the binary classification model
checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True, mode='min')
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
```

```
history = model.fit(
    train_generator,
    epochs=15,
    validation_data=valid_generator,
    class_weight=class_weight_dict,
    callbacks=[checkpoint, early_stopping],
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    validation_steps=valid_generator.samples // BATCH_SIZE
)
```

```
Epoch 1/15
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
129/129 ━━━━━━━━━━━━━━━━━━━━ 2592s 19s/step - accuracy: 0.6102 - loss: 0.6502 - val_accuracy: 0.6943 - val_loss: 0.5902
Epoch 2/15
  1/129 ━━━━━━━━━━━━━━━━━━━━ 13:34 6s/step - accuracy: 0.6875 - loss: 0.7600/usr/lib/python3.10/contextlib.py:153: UserWarning: Your inp
  self.gen.throw(typ, value, traceback)
129/129 ━━━━━━━━━━━━━━━━━━━━ 10s 26ms/step - accuracy: 0.6875 - loss: 0.7600 - val_accuracy: 0.7500 - val_loss: 0.5290
Epoch 3/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 1416s 11s/step - accuracy: 0.7112 - loss: 0.5772 - val_accuracy: 0.7275 - val_loss: 0.5445
Epoch 4/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 11s 21ms/step - accuracy: 0.6875 - loss: 0.7118 - val_accuracy: 0.8750 - val_loss: 0.4398
Epoch 5/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 1459s 11s/step - accuracy: 0.7225 - loss: 0.5779 - val_accuracy: 0.7021 - val_loss: 0.5862
Epoch 6/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 9s 13ms/step - accuracy: 0.8125 - loss: 0.4502 - val_accuracy: 0.7500 - val_loss: 0.4853
Epoch 7/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 1408s 11s/step - accuracy: 0.7405 - loss: 0.5390 - val_accuracy: 0.6963 - val_loss: 0.5942
Epoch 8/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 43s 328ms/step - accuracy: 1.0000 - loss: 0.4390 - val_accuracy: 0.8750 - val_loss: 0.6154
Epoch 9/15
129/129 ━━━━━━━━━━━━━━━━━━━━ 1400s 11s/step - accuracy: 0.7548 - loss: 0.5296 - val_accuracy: 0.6660 - val_loss: 0.5970
```

```
# Save the final model
model.save('/content/drive/MyDrive/diabetic_retinopathy_binary_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```

Prediction Function for Diabetic Retinopathy

The graphs display the model accuracy and model loss over training epochs for both the training and validation datasets.

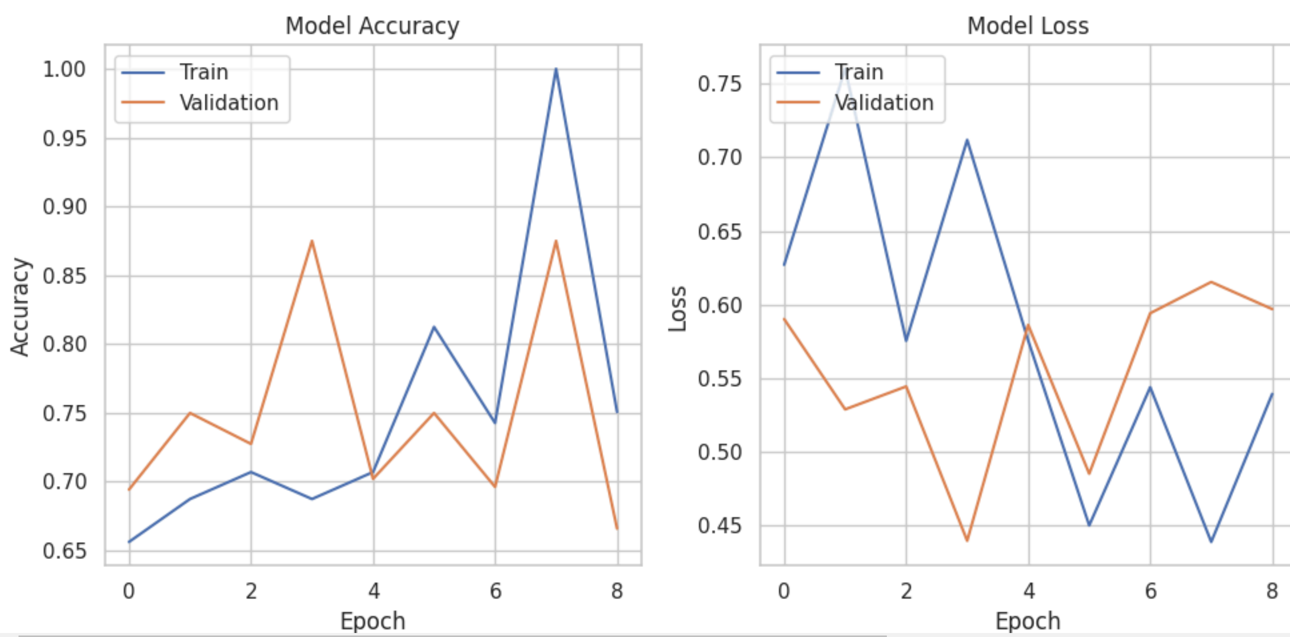
The graphs show fluctuations in both training and validation accuracy, with a peak in training accuracy at epoch 6, indicating learning but with instability. The loss curves suggest potential overfitting, as validation loss does not consistently decrease, indicating a need for further tuning.

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
sns.set(style="whitegrid")
```

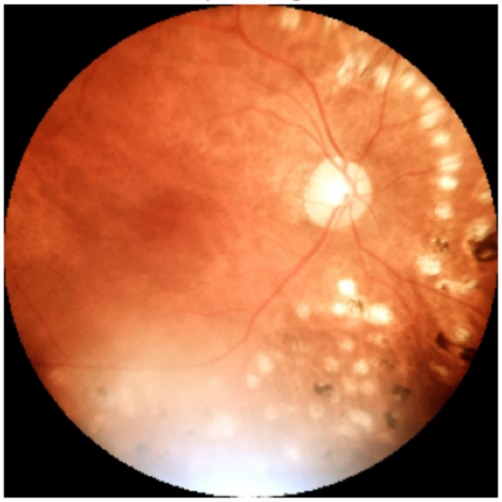
```
# Plot training and validation accuracy/loss curves
```

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

# Accuracy plot
plt.subplot(1, 2, 1)
sns.lineplot(data=history.history['accuracy'], label='Train')
sns.lineplot(data=history.history['val_accuracy'], label='Validation')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')

# Loss plot
plt.subplot(1, 2, 2)
sns.lineplot(data=history.history['loss'], label='Train')
sns.lineplot(data=history.history['val_loss'], label='Validation')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```




```
import matplotlib.pyplot as plt
# Function to display image and predict DR
def display_and_predict_image(img_path):
    # Display the image
    img = tf.keras.preprocessing.image.load_img(img_path, target_size=IMG_SIZE)
    plt.imshow(img)
    plt.axis('off')
    plt.title("Input Image")
    plt.show()

    # Preprocess the image for prediction
    img_array = tf.keras.preprocessing.image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # Get prediction for DR
    dr_prediction = model.predict(img_array)

    # Interpret prediction for DR
    dr_result = 'Not DR' if dr_prediction < 0.5 else 'DR'

    print(f'DR Prediction: {dr_result}')
    return dr_result


img_path = '/content/drive/MyDrive/images/1.1.jpg'
display_and_predict_image(img_path)
```

Input Image



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 238ms/step
DR Prediction: DR
'DR'
```

Summary of Results

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

# Evaluate the model on validation data
val_predictions = model.predict(valid_generator)
val_predictions_binary = (val_predictions > 0.5).astype(int)

# Generate classification report
true_labels = valid_generator.labels  # True labels from the validation generator
print(classification_report(true_labels, val_predictions_binary, target_names=['Not DR', 'DR']))

# Confusion matrix
cm = confusion_matrix(true_labels, val_predictions_binary)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Not DR', 'DR'], yticklabels=['Not DR', 'DR'])
plt.title('Confusion Matrix')
plt.show()

# AUC-ROC score
roc_auc = roc_auc_score(true_labels, val_predictions)
print(f'ROC AUC Score: {roc_auc:.4f}')
```

```
33/33 ━━━━━━━━━━━━━━━━━━━━ 294s 9s/step
              precision    recall  f1-score   support

      Not DR       0.73      0.74      0.73       748
          DR       0.28      0.27      0.27       284

    accuracy                           0.61      1032
   macro avg       0.50      0.50      0.50      1032
weighted avg       0.60      0.61      0.60      1032
```