INFYMA HACKATHON ‘25

Muhammad Zeeshan
Muhammad Essa Zeeshan
Manshah Hussain

MARCH 10, 2025

THE OVERFITTERS
COMSATS University Islamabad

Report on CNN Model for Image Classification

1. Introduction

This project focuses on developing a convolutional neural network (CNN) for classifying images into 5 categories. The model leverages transfer learning using a pre-trained ResNet152V2 base, combined with custom layers for fine-tuning. The approach takes advantage of data augmentation, dropout, and batch normalization to enhance generalization and performance.

2. Data Preparation

Data Augmentation & Preprocessing

To improve the robustness of the model and mitigate overfitting, the following preprocessing steps were applied using Keras' ImageDataGenerator:

- **Training Data:**
  - **Rescaling:** Pixel values normalized by dividing by 255.
  - **Augmentations:**
    - Rotation up to 30 degrees.
    - Horizontal flipping.
    - Shear transformation (range of 0.3).
    - Width and height shifts (range of 0.2).
    - Zoom (range of 0.1).
    - Filling mode set to 'nearest' to handle empty areas after transformations.
- **Validation & Test Data:**
  - **Rescaling Only:** Images are normalized by dividing pixel values by 255 without additional augmentations.

The images are resized to a target size of (224, 224) and are processed in batches of 64.

3. Model Architecture

Base Model: ResNet152V2

- **Pre-trained on ImageNet:**
  The ResNet152V2 model is imported without its top classification layer, allowing the model to utilize pre-learned feature extraction.

- **Layer Freezing:**
  All layers except the last 40 layers are frozen. This strategy retains most of the learned features while fine-tuning the deeper layers to better adapt to the new dataset.

Custom Layers

Following the base model, additional layers were added:

1. **Dropout (0.5):** Helps prevent overfitting by randomly setting input units to 0.
2. **Flatten:** Transforms the multidimensional output of the base model into a one-dimensional vector.
3. **Batch Normalization:** Normalizes the outputs, improving training stability.
4. **Dense Layer (256 units, He initialization):**
    - Followed by batch normalization, ReLU activation, and dropout.
5. **Dense Layer (128 units, He initialization):**
    - Similarly followed by batch normalization, ReLU activation, and dropout.
6. **Dense Layer (32 units, He initialization):**
    - Followed by batch normalization and ReLU activation.
7. **Output Layer:**
    - A Dense layer with 5 units and softmax activation to predict class probabilities.

Parameter Summary:

- **Total Parameters:** ~84.46 million
- **Trainable Parameters:** ~41.16 million
- **Non-trainable Parameters:** ~43.30 million

4. Frontend Implementation:

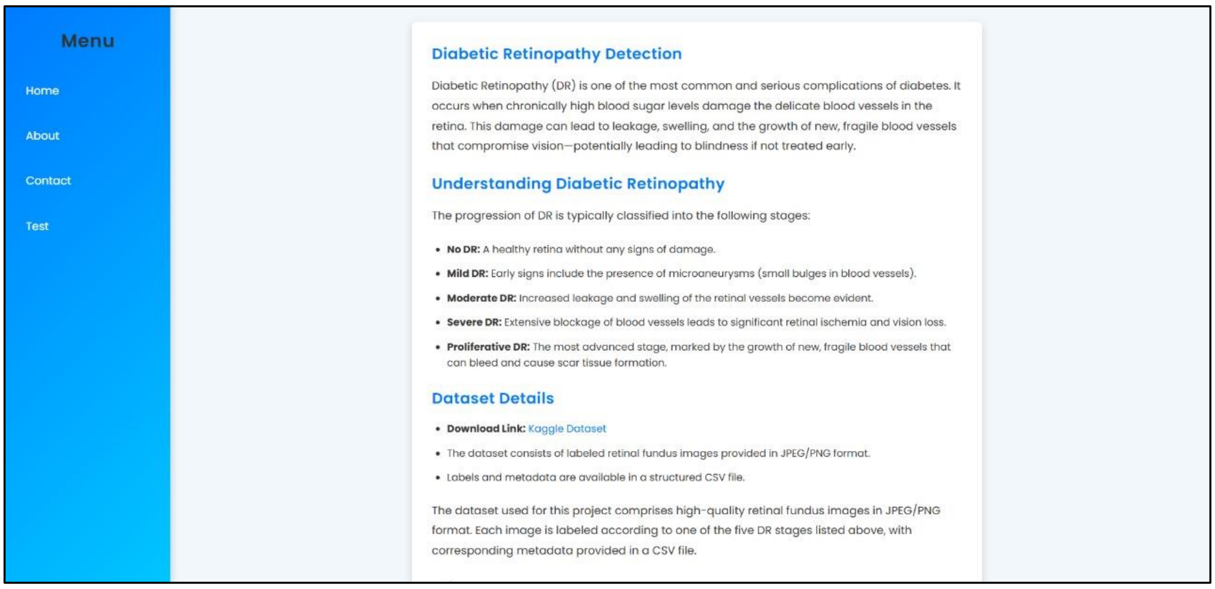
To provide a user-friendly interface for interacting with the model, we developed a frontend with four modules: **Home, About, Contact, and Test**. The **Test** module serves as the core functionality, allowing users to upload images and receive classification results.

Key Features:

- **Home:** An overview of the application, guiding users on its purpose and functionality.
- **About:** Provides details about the project, including the model architecture and objectives.
- **Contact:** Contains information for user inquiries and support.
- **Test:** Allows users to upload images, processes them through the CNN model, and displays classification results in real-time.

The frontend was designed to be **intuitive, responsive, and visually appealing**, ensuring smooth interaction with the model.

- **Home:**



Diabetic Retinopathy Detection

Diabetic Retinopathy (DR) is one of the most common and serious complications of diabetes. It occurs when chronically high blood sugar levels damage the delicate blood vessels in the retina. This damage can lead to leakage, swelling, and the growth of new, fragile blood vessels that compromise vision—potentially leading to blindness if not treated early.

Understanding Diabetic Retinopathy

The progression of DR is typically classified into the following stages:

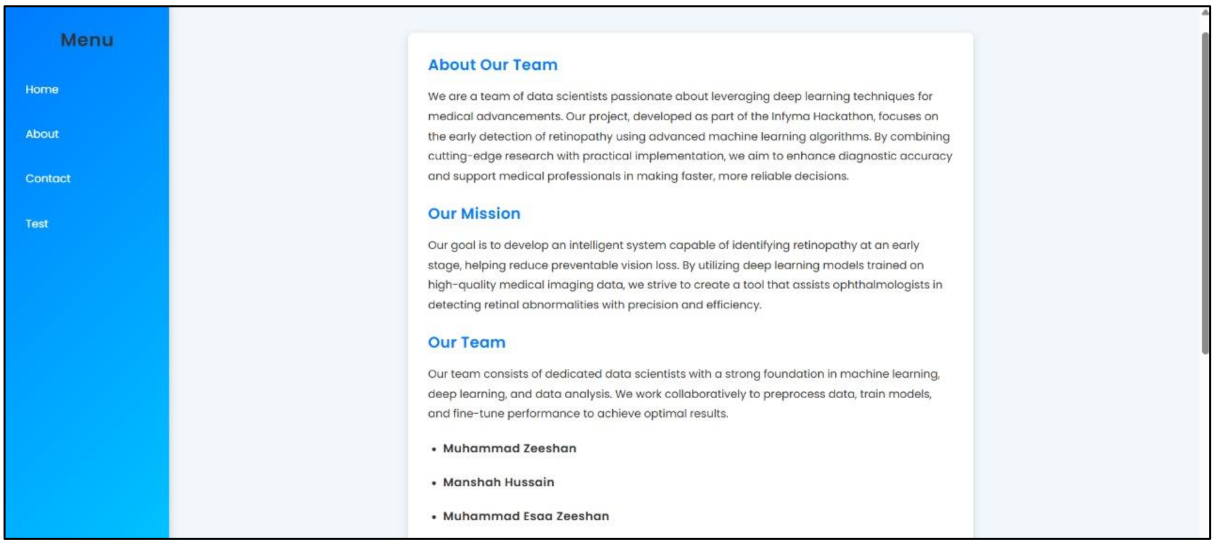
- **No DR:** A healthy retina without any signs of damage.
- **Mild DR:** Early signs include the presence of microaneurysms (small bulges in blood vessels).
- **Moderate DR:** Increased leakage and swelling of the retinal vessels become evident.
- **Severe DR:** Extensive blockage of blood vessels leads to significant retinal ischemia and vision loss.
- **Proliferative DR:** The most advanced stage, marked by the growth of new, fragile blood vessels that can bleed and cause scar tissue formation.

Dataset Details

- **Download Link:** Kaggle Dataset
- The dataset consists of labeled retinal fundus images provided in JPEG/PNG format.
- Labels and metadata are available in a structured CSV file.

The dataset used for this project comprises high-quality retinal fundus images in JPEG/PNG format. Each image is labeled according to one of the five DR stages listed above, with corresponding metadata provided in a CSV file.

- **About:**



About Our Team

We are a team of data scientists passionate about leveraging deep learning techniques for medical advancements. Our project, developed as part of the Infyma Hackathon, focuses on the early detection of retinopathy using advanced machine learning algorithms. By combining cutting-edge research with practical implementation, we aim to enhance diagnostic accuracy and support medical professionals in making faster, more reliable decisions.

Our Mission

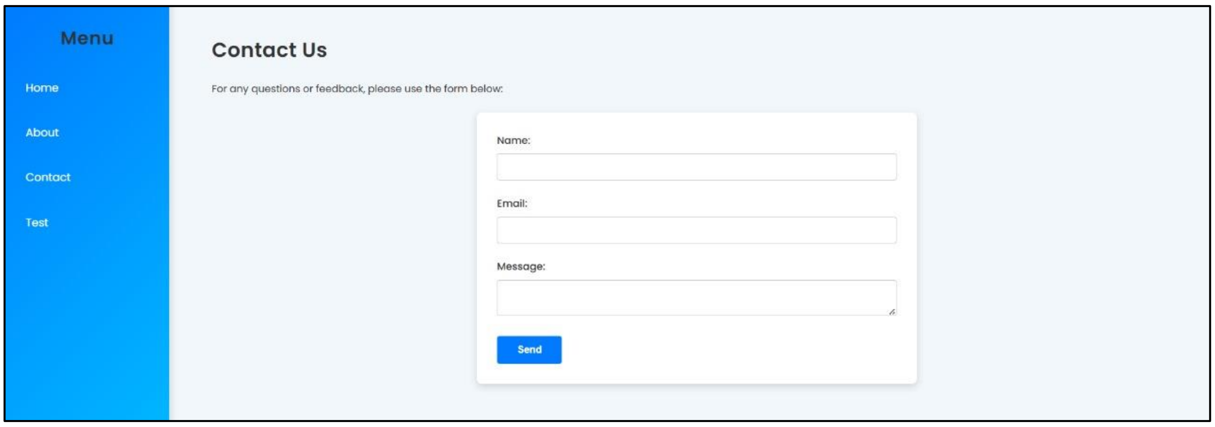
Our goal is to develop an intelligent system capable of identifying retinopathy at an early stage, helping reduce preventable vision loss. By utilizing deep learning models trained on high-quality medical imaging data, we strive to create a tool that assists ophthalmologists in detecting retinal abnormalities with precision and efficiency.

Our Team

Our team consists of dedicated data scientists with a strong foundation in machine learning, deep learning, and data analysis. We work collaboratively to preprocess data, train models, and fine-tune performance to achieve optimal results.

- **Muhammad Zeeshan**
- **Manshah Hussain**
- **Muhammad Esaa Zeeshan**

- **Contact:**



Contact Us

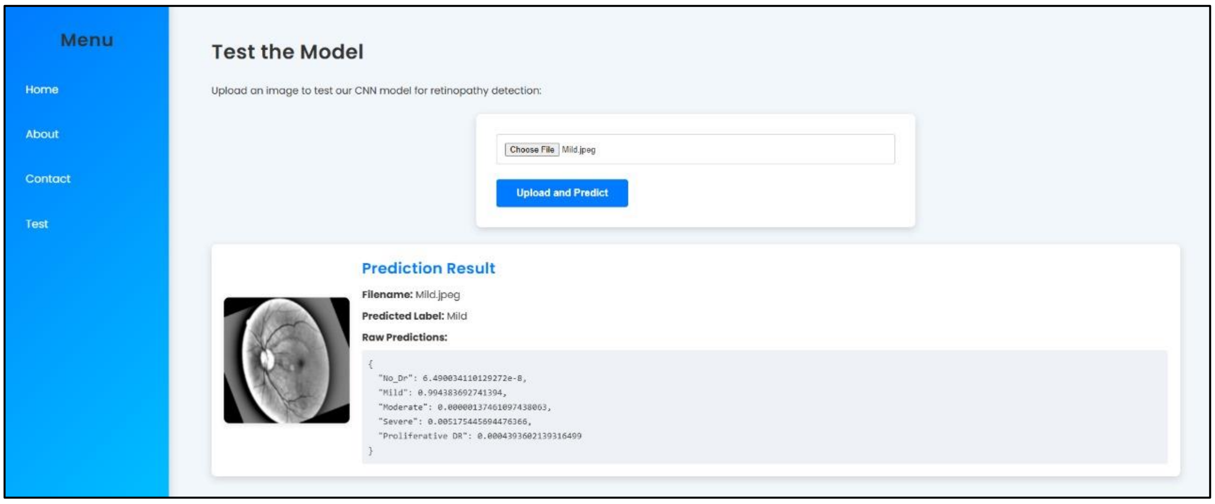
For any questions or feedback, please use the form below:

Name:

Email:

Message:

- **Test (Main Prediction):**



Test the Model

Upload an image to test our CNN model for retinopathy detection:

Mild.jpeg

Prediction Result

Filename: Mild.jpeg

Predicted Label: Mild

Raw Predictions:

```
{
  "No_Dr": 6.490034110129272e-8,
  "Mild": 0.994383692741394,
  "Moderate": 0.000001374610974380G3,
  "Severe": 0.005175445694476366,
  "Proliferative DR": 0.0004393602139316499
}
```

5. Training Process

Compilation

The model was compiled with the following settings:

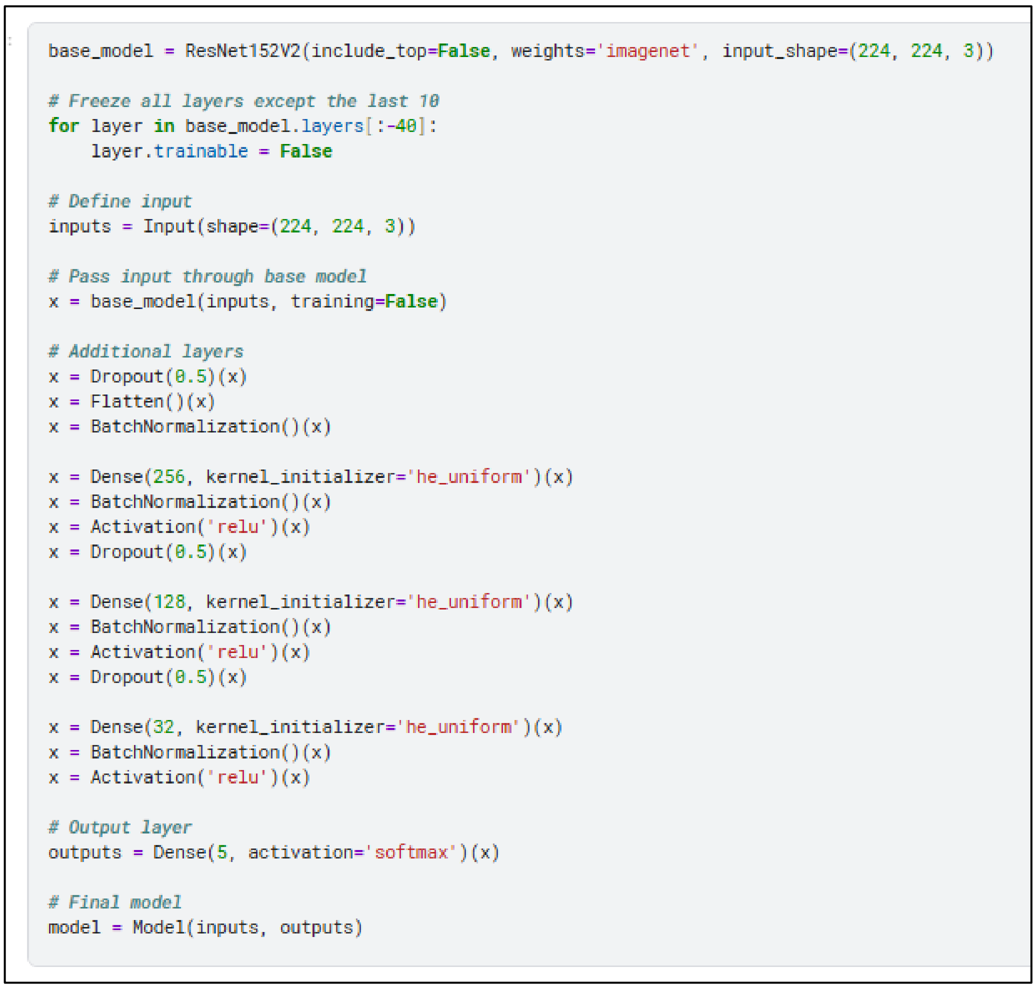
- **Optimizer:** Adam
- **Loss Function:** Categorical crossentropy
- **Metric:** Accuracy

Model Training

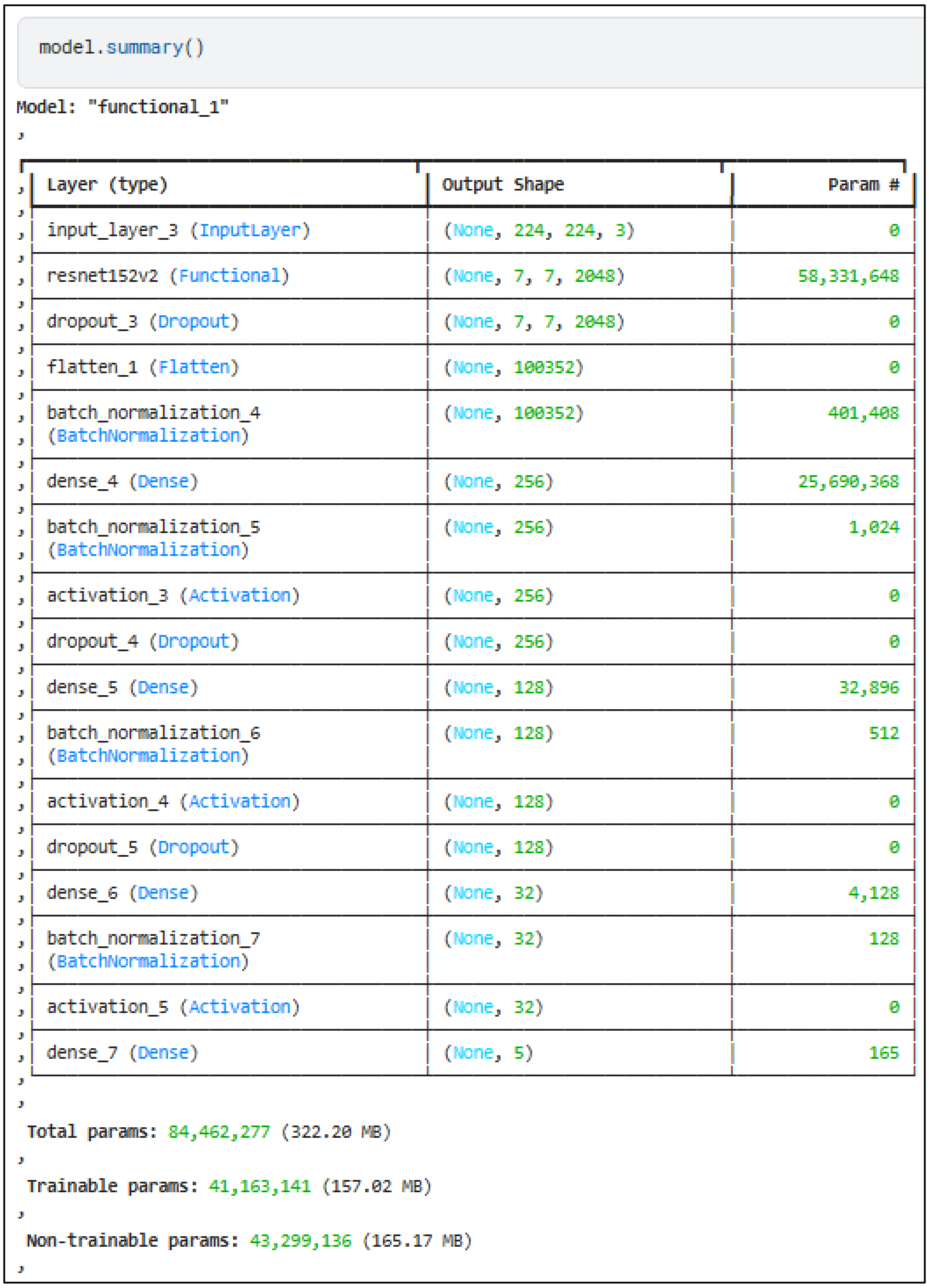
The model was trained for 100+ epochs using:

- **Training Data:** 100 steps per epoch
- **Validation Data:** 20 steps per epoch

The training process outputs training and validation metrics, which are later used to assess performance.

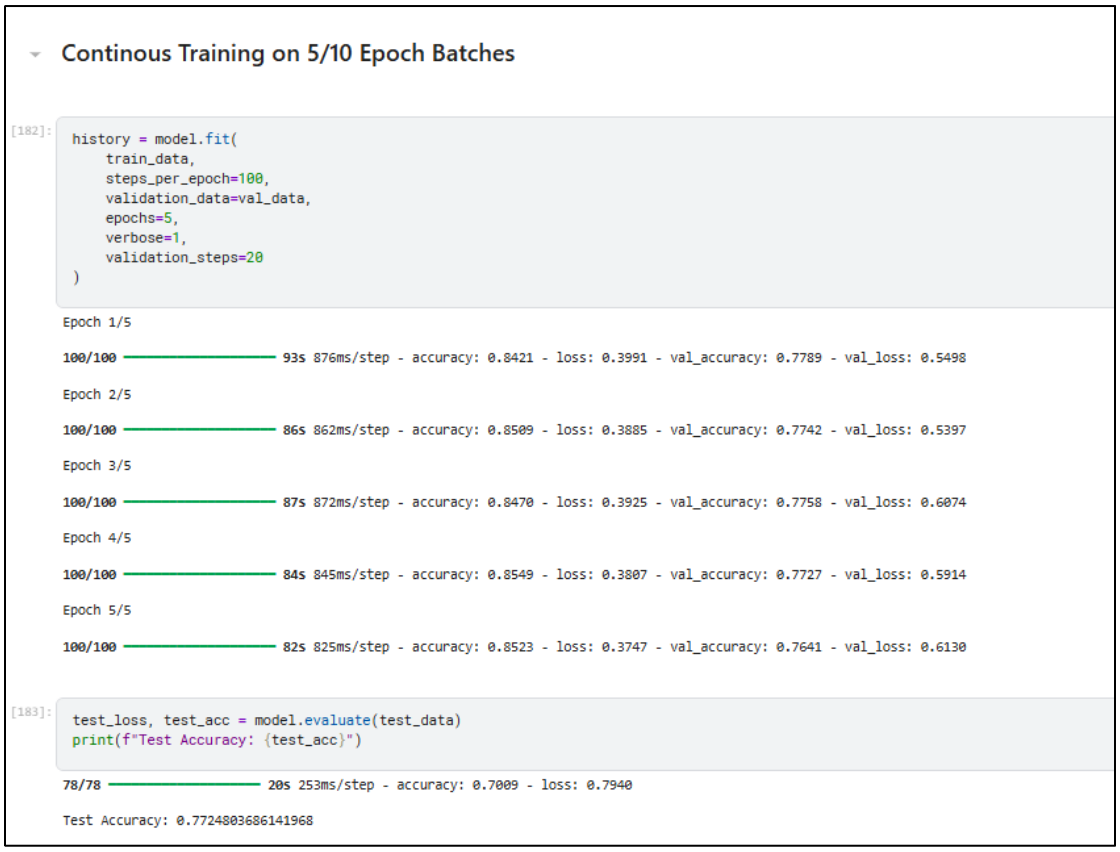


```python
base_model = ResNet152V2(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Freeze all layers except the last 10
for layer in base_model.layers[:-40]:
    layer.trainable = False

# Define input
inputs = Input(shape=(224, 224, 3))

# Pass input through base model
x = base_model(inputs, training=False)

# Additional layers
x = Dropout(0.5)(x)
x = Flatten()(x)
x = BatchNormalization()(x)

x = Dense(256, kernel_initializer='he_uniform')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

x = Dense(128, kernel_initializer='he_uniform')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)

x = Dense(32, kernel_initializer='he_uniform')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

# Output layer
outputs = Dense(5, activation='softmax')(x)

# Final model
model = Model(inputs, outputs)
```



```python
model.summary()
```

```
Model: "functional_1"
,
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
,┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
,┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
,│ input_layer_3 (InputLayer)           │ (None, 224, 224, 3)         │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ resnet152v2 (Functional)             │ (None, 7, 7, 2048)          │      58,331,648 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dropout_3 (Dropout)                  │ (None, 7, 7, 2048)          │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ flatten_1 (Flatten)                  │ (None, 100352)              │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ batch_normalization_4                │ (None, 100352)              │         401,408 │
,│ (BatchNormalization)                 │                             │                 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dense_4 (Dense)                      │ (None, 256)                 │      25,690,368 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ batch_normalization_5                │ (None, 256)                 │           1,024 │
,│ (BatchNormalization)                 │                             │                 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ activation_3 (Activation)            │ (None, 256)                 │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dropout_4 (Dropout)                  │ (None, 256)                 │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dense_5 (Dense)                      │ (None, 128)                 │          32,896 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ batch_normalization_6                │ (None, 128)                 │             512 │
,│ (BatchNormalization)                 │                             │                 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ activation_4 (Activation)            │ (None, 128)                 │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dropout_5 (Dropout)                  │ (None, 128)                 │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dense_6 (Dense)                      │ (None, 32)                  │           4,128 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ batch_normalization_7                │ (None, 32)                  │             128 │
,│ (BatchNormalization)                 │                             │                 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ activation_5 (Activation)            │ (None, 32)                  │               0 │
,├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
,│ dense_7 (Dense)                      │ (None, 5)                   │             165 │
,└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
,
 Total params: 84,462,277 (322.20 MB)
,
 Trainable params: 41,163,141 (157.02 MB)
,
 Non-trainable params: 43,299,136 (165.17 MB)
,
```



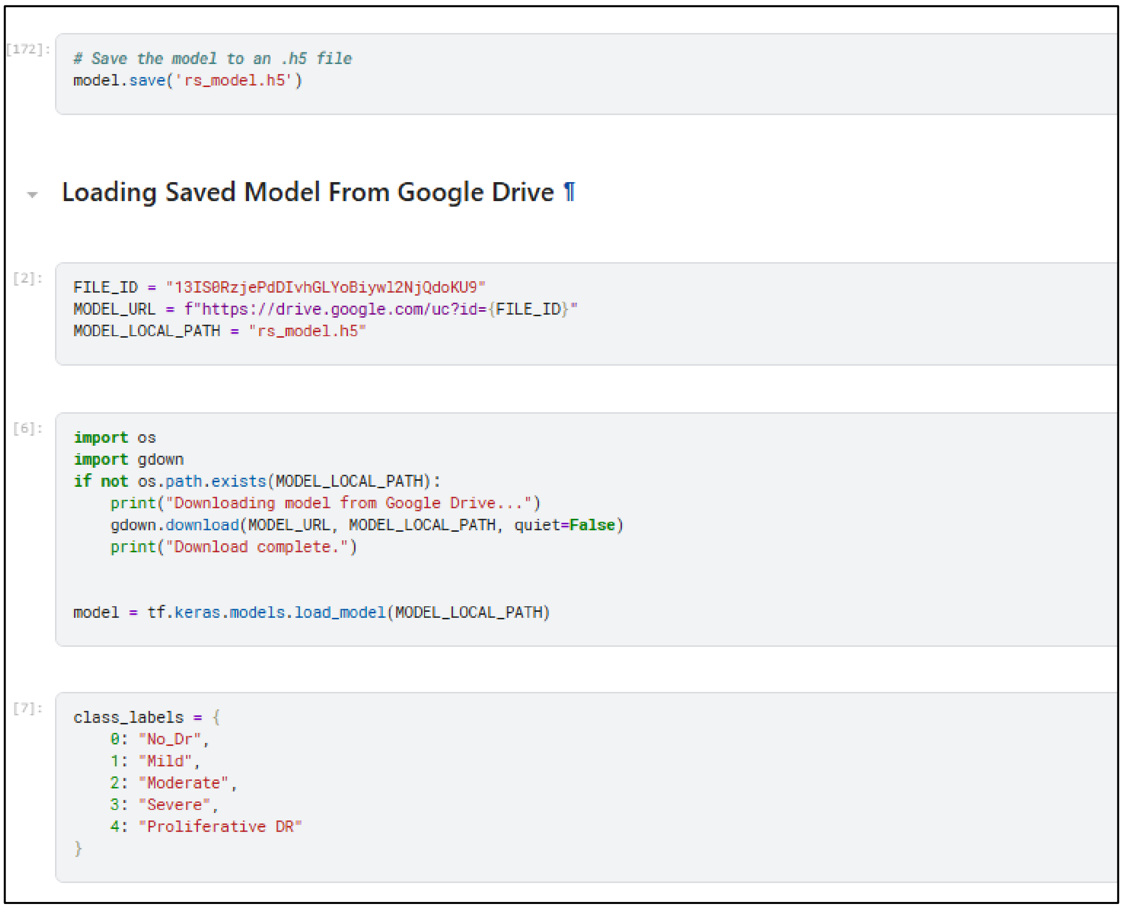
Continous Training on 5/10 Epoch Batches

```
[182]:
history = model.fit(
    train_data,
    steps_per_epoch=100,
    validation_data=val_data,
    epochs=5,
    verbose=1,
    validation_steps=20
)
```

```
Epoch 1/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 93s 876ms/step - accuracy: 0.8421 - loss: 0.3991 - val_accuracy: 0.7789 - val_loss: 0.5498
Epoch 2/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 86s 862ms/step - accuracy: 0.8509 - loss: 0.3885 - val_accuracy: 0.7742 - val_loss: 0.5397
Epoch 3/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 87s 872ms/step - accuracy: 0.8470 - loss: 0.3925 - val_accuracy: 0.7758 - val_loss: 0.6074
Epoch 4/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 84s 845ms/step - accuracy: 0.8549 - loss: 0.3807 - val_accuracy: 0.7727 - val_loss: 0.5914
Epoch 5/5
100/100 ━━━━━━━━━━━━━━━━━━━━ 82s 825ms/step - accuracy: 0.8523 - loss: 0.3747 - val_accuracy: 0.7641 - val_loss: 0.6130
```

```
[183]:
test_loss, test_acc = model.evaluate(test_data)
print(f"Test Accuracy: {test_acc}")
```

```
78/78 ━━━━━━━━━━━━━━━━━━━━ 20s 253ms/step - accuracy: 0.7009 - loss: 0.7940
Test Accuracy: 0.7724803686141968
```



```
[172]:
# Save the model to an .h5 file
model.save('rs_model.h5')
```

Loading Saved Model From Google Drive ¶

```
[2]:
FILE_ID = "13IS0RzjePdDIvhGLYoBiywl2NjQdoKU9"
MODEL_URL = f"https://drive.google.com/uc?id={FILE_ID}"
MODEL_LOCAL_PATH = "rs_model.h5"
```

```
[6]:
import os
import gdown
if not os.path.exists(MODEL_LOCAL_PATH):
    print("Downloading model from Google Drive...")
    gdown.download(MODEL_URL, MODEL_LOCAL_PATH, quiet=False)
    print("Download complete.")


model = tf.keras.models.load_model(MODEL_LOCAL_PATH)
```

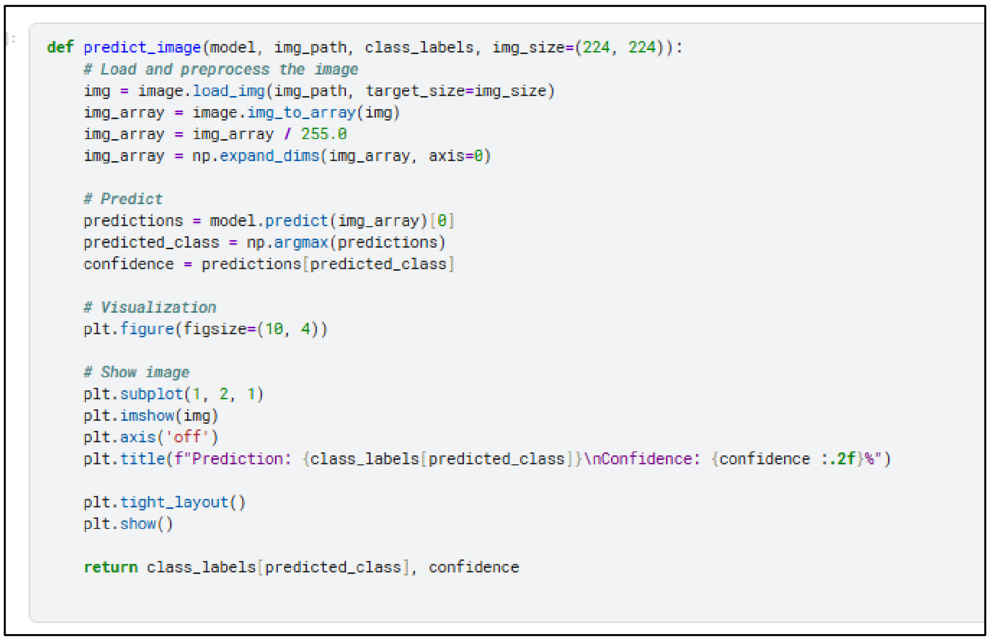
```
[7]:
class_labels = {
    0: "No_Dr",
    1: "Mild",
    2: "Moderate",
    3: "Severe",
    4: "Proliferative DR"
}
```

6. Evaluation and Results

Prediction & Metrics

After training, the model's performance was evaluated using the test dataset:

- **Predictions:**
  The model generates probability distributions for each test image, with the predicted class being the one with the highest probability.



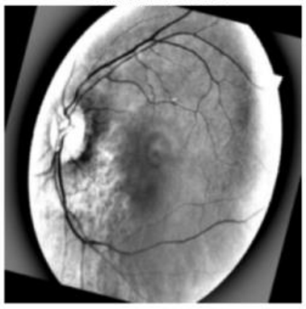
```python
def predict_image(model, img_path, class_labels, img_size=(224, 224)):
    # Load and preprocess the image
    img = image.load_img(img_path, target_size=img_size)
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    # Predict
    predictions = model.predict(img_array)[0]
    predicted_class = np.argmax(predictions)
    confidence = predictions[predicted_class]

    # Visualization
    plt.figure(figsize=(10, 4))

    # Show image
    plt.subplot(1, 2, 1)
    plt.imshow(img)
    plt.axis('off')
    plt.title(f"Prediction: {class_labels[predicted_class]}\nConfidence: {confidence :.2f}%")

    plt.tight_layout()
    plt.show()

    return class_labels[predicted_class], confidence
```

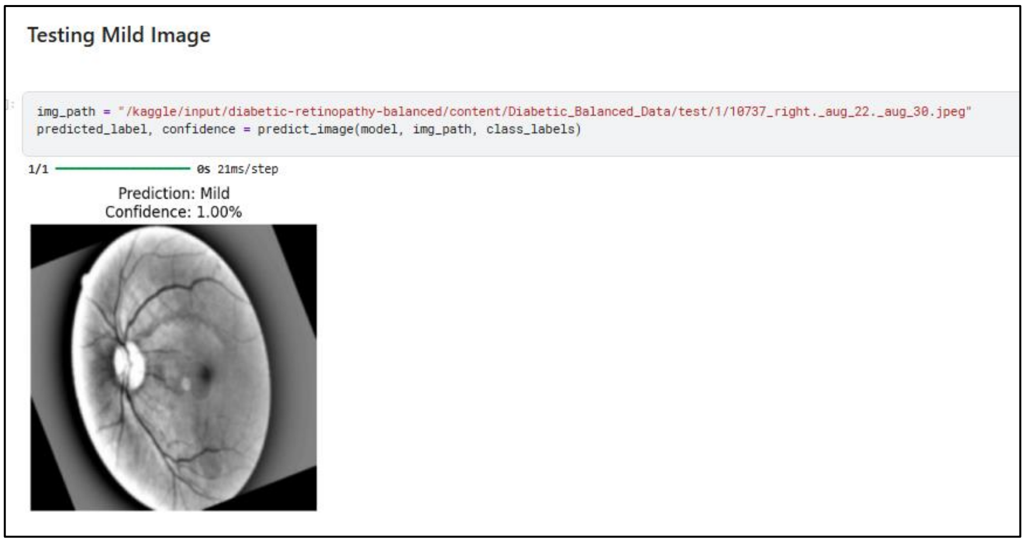
Testing No_DR Image

```python
img_path = "/kaggle/input/diabetic-retinopathy-balanced/content/Diabetic_Balanced_Data/test/0/10814_left._aug_16.jpeg"
predicted_label, confidence = predict_image(model, img_path, class_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Prediction: No_Dr
Confidence: 0.81%



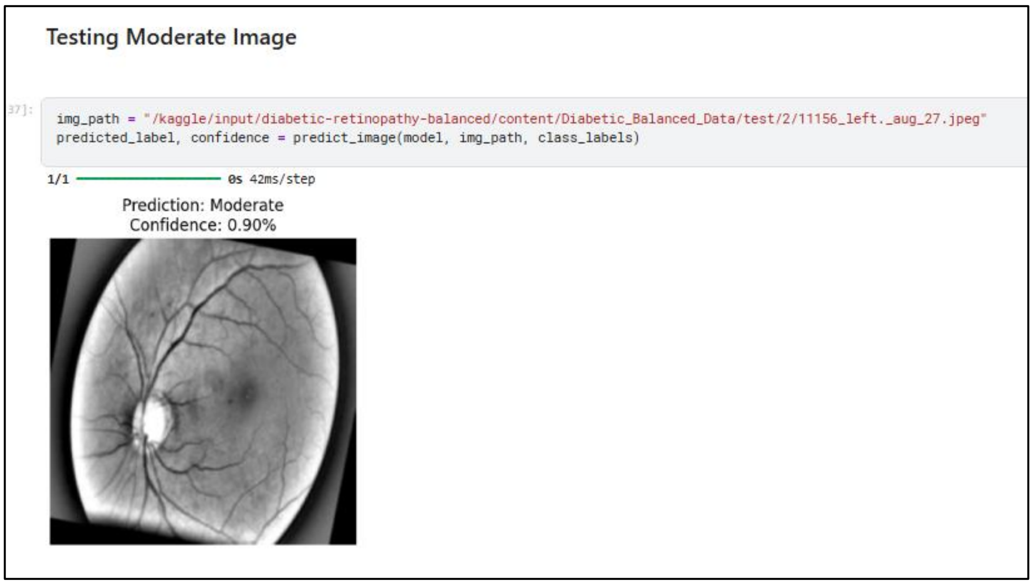


Testing Mild Image

```
img_path = "/kaggle/input/diabetic-retinopathy-balanced/content/Diabetic_Balanced_Data/test/1/10737_right._aug_22._aug_30.jpeg"
predicted_label, confidence = predict_image(model, img_path, class_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Prediction: Mild
Confidence: 1.00%

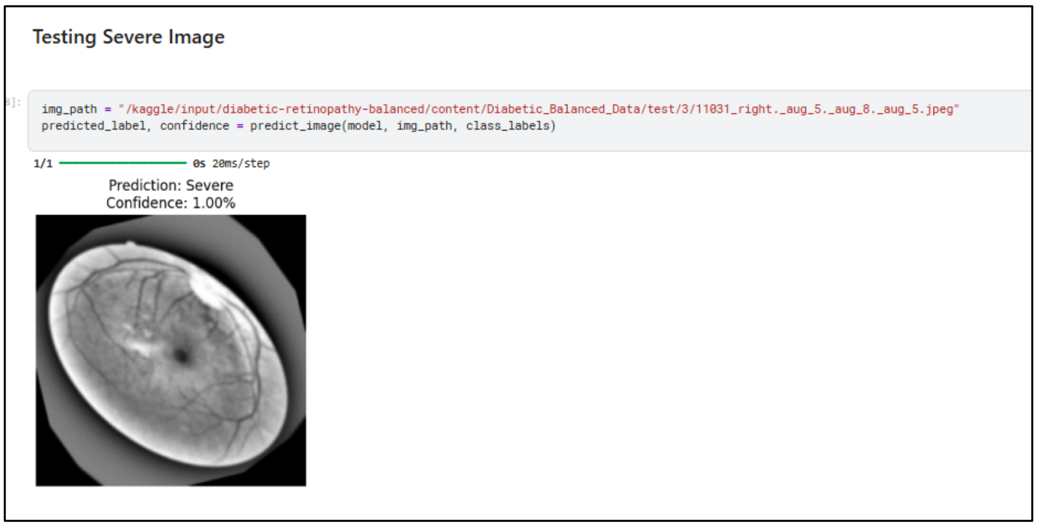


Testing Moderate Image

```
img_path = "/kaggle/input/diabetic-retinopathy-balanced/content/Diabetic_Balanced_Data/test/2/11156_left._aug_27.jpeg"
predicted_label, confidence = predict_image(model, img_path, class_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
```

Prediction: Moderate
Confidence: 0.90%

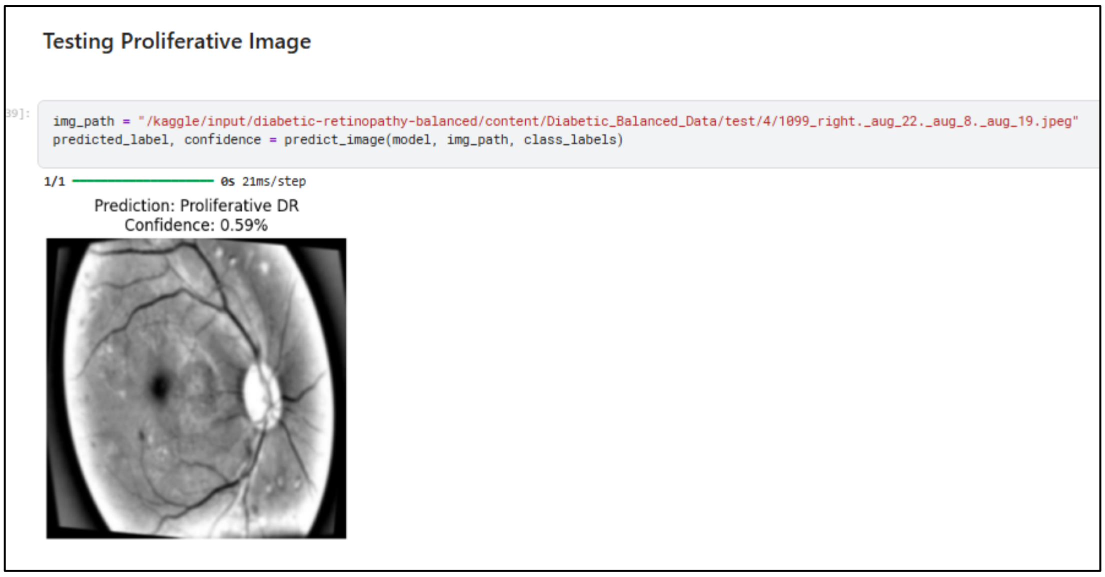


Testing Severe Image

```
img_path = "/kaggle/input/diabetic-retinopathy-balanced/content/Diabetic_Balanced_Data/test/3/11031_right._aug_5._aug_8._aug_5.jpeg"
predicted_label, confidence = predict_image(model, img_path, class_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

Prediction: Severe
Confidence: 1.00%

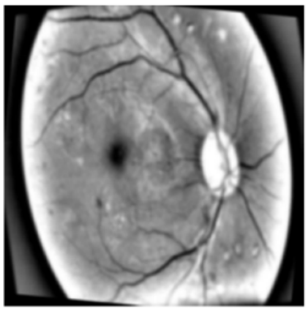


Testing Proliferative Image

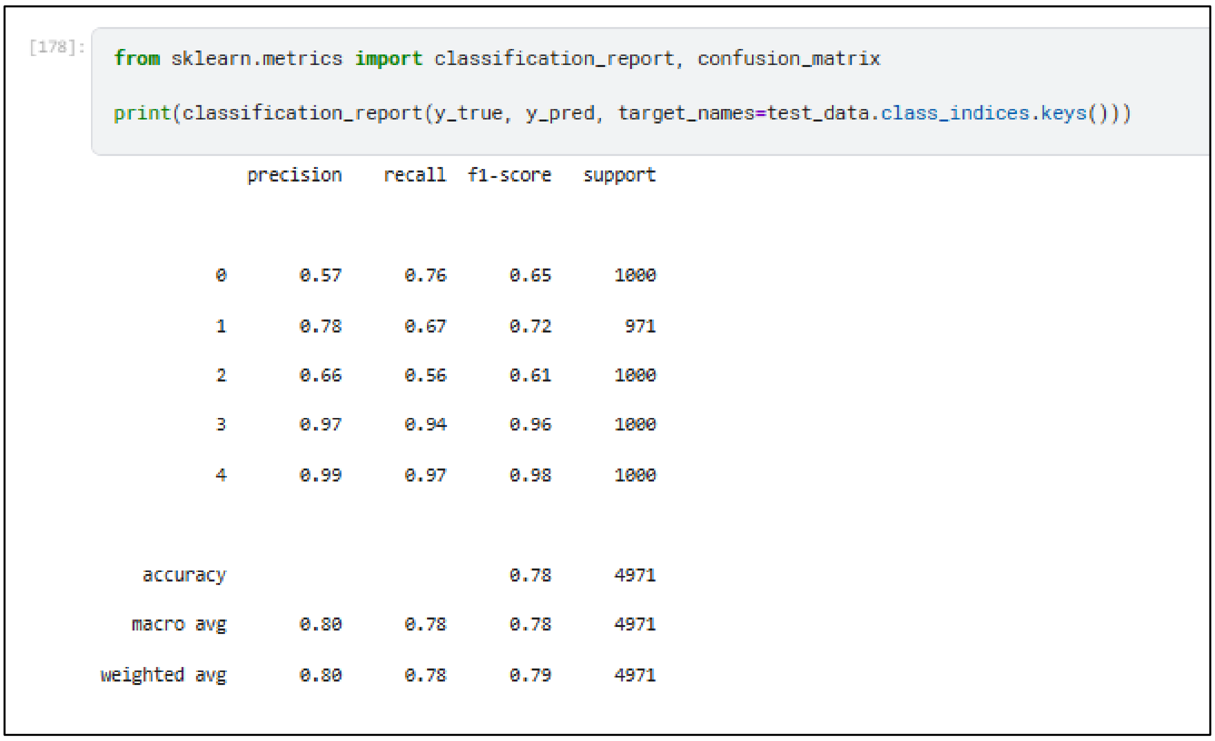
```
img_path = "/kaggle/input/diabetic-retinopathy-balanced/content/Diabetic_Balanced_Data/test/4/1099_right._aug_22._aug_8._aug_19.jpeg"
predicted_label, confidence = predict_image(model, img_path, class_labels)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

Prediction: Proliferative DR
Confidence: 0.59%



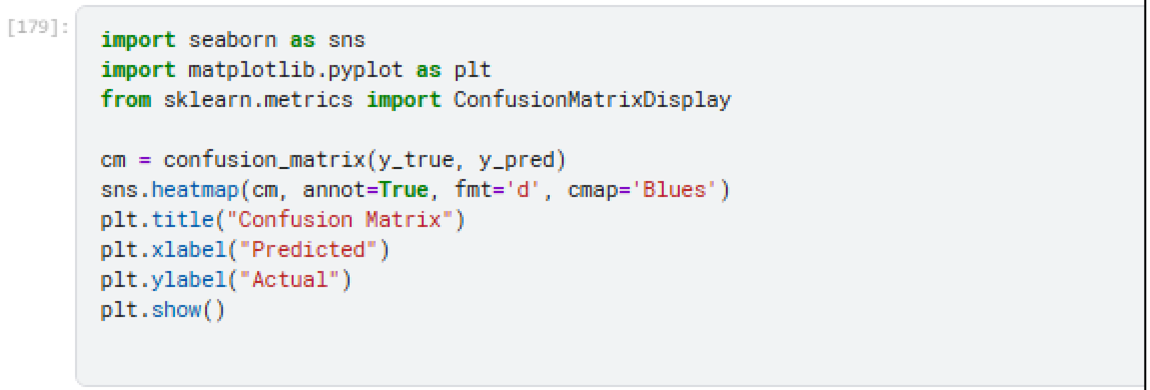
- **Classification Report:**
  Precision, recall, and F1-score for each class were computed to assess classification performance.



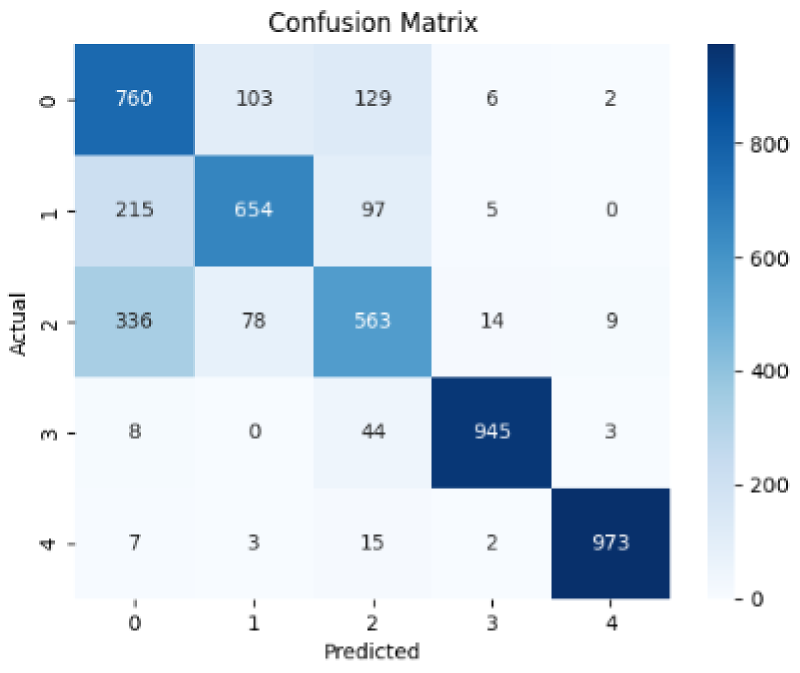
```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_true, y_pred, target_names=test_data.class_indices.keys()))
```

```
              precision    recall  f1-score   support

           0       0.57      0.76      0.65      1000
           1       0.78      0.67      0.72       971
           2       0.66      0.56      0.61      1000
           3       0.97      0.94      0.96      1000
           4       0.99      0.97      0.98      1000

    accuracy                           0.78      4971
   macro avg       0.80      0.78      0.78      4971
weighted avg       0.80      0.78      0.79      4971
```

- **Confusion Matrix:**
  A confusion matrix was generated and visualized using a heatmap. This provides insights into class-specific performance and misclassification patterns.



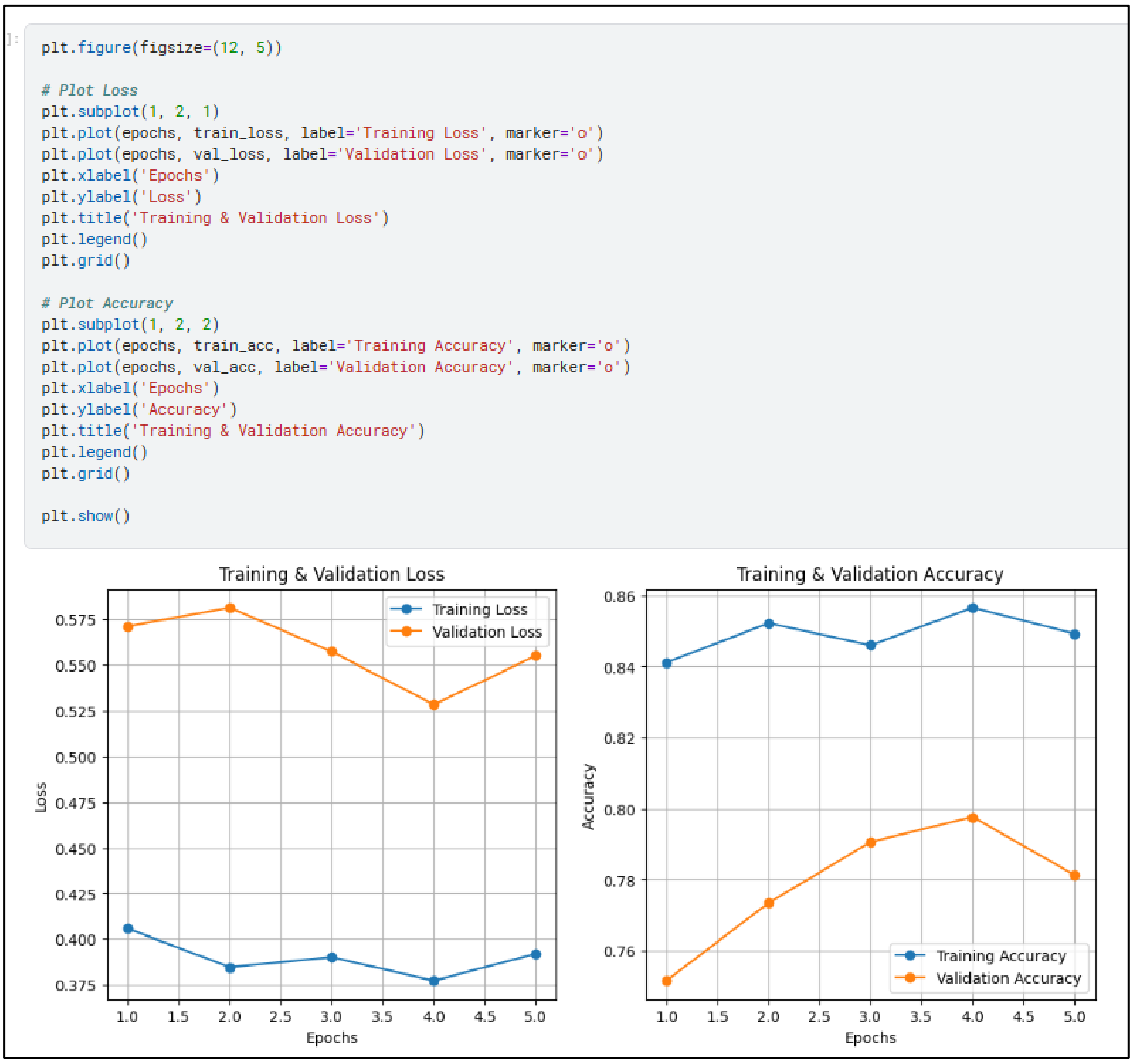
[179]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



Performance Visualization

The training process was further analyzed through visualizations:

- **Loss Curves:**
  Both training and validation loss were plotted across epochs to monitor convergence and overfitting.
- **Accuracy Curves:**
  Training and validation accuracy were similarly plotted to assess model performance improvements over epochs..



```python
plt.figure(figsize=(12, 5))

# Plot Loss
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, label='Training Loss', marker='o')
plt.plot(epochs, val_loss, label='Validation Loss', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training & Validation Loss')
plt.legend()
plt.grid()

# Plot Accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc, label='Training Accuracy', marker='o')
plt.plot(epochs, val_acc, label='Validation Accuracy', marker='o')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training & Validation Accuracy')
plt.legend()
plt.grid()

plt.show()
```

7. Conclusion

This report outlined the development of a robust image classification model using transfer learning with ResNet152V2. Key enhancements included:

- **Data Augmentation:** Increased dataset variability to improve generalization.
- **Model Fine-Tuning:** Utilized pre-trained features while refining deeper layers for better adaptation.
- **Regularization Techniques:** Applied dropout and batch normalization to prevent overfitting.
- **Evaluation:** Assessed performance using classification metrics and visualizations.

Future improvements could focus on hyperparameter tuning, experimenting with deeper architectures, or incorporating advanced augmentation techniques to further enhance accuracy. This approach demonstrates the effectiveness of transfer learning in image classification and lays a foundation for further advancements in model performance.