```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!pip install imgaug
```

```
Requirement already satisfied: imgaug in /usr/local/lib/python3.10/dist-packages (0.4.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from imgaug) (1.16.0)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist-packages (from imgaug) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from imgaug) (1.13.1)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from imgaug) (10.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from imgaug) (3.7.1)
Requirement already satisfied: scikit-image>=0.14.2 in /usr/local/lib/python3.10/dist-packages (from imgaug) (0.24.0)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (from imgaug) (4.10.0.84)
Requirement already satisfied: imageio in /usr/local/lib/python3.10/dist-packages (from imgaug) (2.35.1)
Requirement already satisfied: Shapely in /usr/local/lib/python3.10/dist-packages (from imgaug) (2.0.6)
Requirement already satisfied: networkx>=2.8 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (3.4)
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (2
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (24.1)
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.10/dist-packages (from scikit-image>=0.14.2->imgaug) (0.4
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (1.4.7)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->imgaug) (2.8.2)
```

```
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.pyplot as plt
from tqdm import tqdm
from imgaug import augmenters as iaa

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# For Capsule Network
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Layer

# For Grad-CAM
import tensorflow.keras.backend as K
import seaborn as sns



# Define the base path to your dataset
base_path = '/content/drive/MyDrive/DiabeticRetinopathyDetection/DR/data/'

train_df = pd.read_csv(base_path + 'train.csv')
test_df = pd.read_csv(base_path + 'test.csv')

train_img_path = base_path + 'train_images/'
test_img_path = base_path + 'test_images/'



# Image size for resizing (Reduced)
IMG_SIZE = 64  # Reduced image size for resource constraints

def load_and_preprocess_image(image_id, label, img_dir):
    """
    Load an image and preprocess it.
    """
    # Read the image
    file_path = os.path.join(img_dir, image_id + '.png')
    image = cv2.imread(file_path)

    # Check if image is loaded properly
    if image is None:
        print(f'Error loading image {file_path}')
        return None, label
```

```
    # Resize the image
    image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))

    # Convert to RGB (from BGR)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Do NOT normalize here; we'll do it after augmentation
    return image, label



# Define augmentation sequence
augmenter = iaa.Sequential([
    iaa.Fliplr(0.5),  # horizontal flips
    iaa.Affine(rotate=(-15, 15)),  # reduced rotation angles
    iaa.Multiply((0.9, 1.1)),  # slight brightness adjustments
    iaa.LinearContrast((0.9, 1.1)),  # slight contrast adjustments
], random_order=True)



X = []
y = []

# Handle class imbalance by calculating class counts
class_counts = train_df['diagnosis'].value_counts()
minority_classes = class_counts[class_counts < class_counts.mean()].index.tolist()

for idx, row in tqdm(train_df.iterrows(), total=train_df.shape[0]):
    image_id = row['id_code']
    label = row['diagnosis']
    image, label = load_and_preprocess_image(image_id, label, train_img_path)
    if image is not None:
        # Ensure image is uint8 before augmentation
        image = image.astype(np.uint8)

        X.append(image)
        y.append(label)

        # Data augmentation (limit to minority classes to handle imbalance)
        if label in minority_classes:
            augmented_image = augmenter(image=image)
            X.append(augmented_image)
            y.append(label)

X = np.array(X)
y = np.array(y)

# Now perform normalization
X = X.astype('float32') / 255.0
```

```
100%|██████████| 3662/3662 [1:25:27<00:00,  1.40s/it]
```

```
from sklearn.preprocessing import LabelBinarizer

lb = LabelBinarizer()
y_encoded = lb.fit_transform(y)
y_encoded = to_categorical(y)



X_train, X_val, y_train, y_val = train_test_split(X, y_encoded, test_size=0.2, random_state=42, stratify=y)



def squash(vectors, axis=-1):
    """
    The non-linear activation used in Capsule Networks.
    """
    s_squared_norm = tf.reduce_sum(tf.square(vectors), axis, keepdims=True)
    scale = s_squared_norm / (1 + s_squared_norm + K.epsilon())
    return vectors * scale / tf.sqrt(s_squared_norm + K.epsilon())



class CapsuleLayer(tf.keras.layers.Layer):
    def __init__(self, num_capsule, dim_capsule, routings=3,
                 kernel_initializer='glorot_uniform', **kwargs):
        super(CapsuleLayer, self).__init__(**kwargs)
        self.num_capsule = num_capsule      # Number of output capsules
        self.dim_capsule = dim_capsule      # Dimension of each output capsule
```

```
        self.routings = routings            # Number of routing iterations
        self.kernel_initializer = keras.initializers.get(kernel_initializer)

    def build(self, input_shape):
        # input_shape: [batch_size, input_num_capsule, input_dim_capsule]
        self.input_num_capsule = input_shape[1]
        self.input_dim_capsule = input_shape[2]

        # Initialize the transformation matrix W
        # W shape: [input_num_capsule, num_capsule, input_dim_capsule, dim_capsule]
        self.W = self.add_weight(shape=(self.input_num_capsule,
                                        self.num_capsule,
                                        self.input_dim_capsule,
                                        self.dim_capsule),
                                 initializer=self.kernel_initializer,
                                 name='W',
                                 trainable=True)
        super(CapsuleLayer, self).build(input_shape)

    def call(self, inputs):
        # inputs shape: [batch_size, input_num_capsule, input_dim_capsule]
        batch_size = tf.shape(inputs)[0]

        # Expand input dimensions
        inputs_expanded = tf.expand_dims(inputs, 2)
        # inputs_expanded shape: [batch_size, input_num_capsule, 1, input_dim_capsule]

        inputs_tiled = tf.tile(inputs_expanded, [1, 1, self.num_capsule, 1])
        # inputs_tiled shape: [batch_size, input_num_capsule, num_capsule, input_dim_capsule]

        # Expand input tensors to match W dimensions for matmul
        inputs_tiled = tf.expand_dims(inputs_tiled, -1)
        # inputs_tiled shape: [batch_size, input_num_capsule, num_capsule, input_dim_capsule, 1]

        # Tile W to match batch size
        W_tiled = tf.tile(tf.expand_dims(self.W, 0), [batch_size, 1, 1, 1, 1])
        # W_tiled shape: [batch_size, input_num_capsule, num_capsule, input_dim_capsule, dim_capsule]

        # Perform matrix multiplication between W and inputs
        # Matmul between last two dimensions: (input_dim_capsule, dim_capsule) x (input_dim_capsule, 1)
        # Resulting shape: [batch_size, input_num_capsule, num_capsule, dim_capsule, 1]
        votes = tf.matmul(W_tiled, inputs_tiled)
        # votes shape: [batch_size, input_num_capsule, num_capsule, dim_capsule, 1]

        # Squeeze the last dimension
        votes = tf.squeeze(votes, axis=-1)
        # votes shape: [batch_size, input_num_capsule, num_capsule, dim_capsule]

        # Routing algorithm
        logits = tf.zeros([batch_size, self.input_num_capsule, self.num_capsule], dtype=tf.float32)

        for i in range(self.routings):
            # Apply softmax to the logits along the num_capsule axis
            c = tf.nn.softmax(logits, axis=2)
            # c shape: [batch_size, input_num_capsule, num_capsule]

            # Expand dims for broadcasting
            c_expanded = tf.expand_dims(c, -1)
            # c_expanded shape: [batch_size, input_num_capsule, num_capsule, 1]

            # Weight the votes by the coupling coefficients
            weighted_votes = c_expanded * votes
            # weighted_votes shape: [batch_size, input_num_capsule, num_capsule, dim_capsule]

            # Sum over the input capsules
            s = tf.reduce_sum(weighted_votes, axis=1)
            # s shape: [batch_size, num_capsule, dim_capsule]

            # Apply the squash activation function
            v = squash(s)
            # v shape: [batch_size, num_capsule, dim_capsule]

            if i < self.routings - 1:
                # Update the logits for the next iteration
                v_expanded = tf.expand_dims(v, 1)
                # v_expanded shape: [batch_size, 1, num_capsule, dim_capsule]

                agreement = tf.reduce_sum(votes * v_expanded, axis=-1)
                # agreement shape: [batch_size, input_num_capsule, num_capsule]

                logits += agreement
```

```
        return v


def CapsNet(input_shape, n_class, routings):
    x = Input(shape=input_shape)

    # Layer 1: Conventional Conv2D layer
    conv1 = Conv2D(filters=32, kernel_size=9, strides=1, activation='relu', padding='valid', name='conv1')(x)

    # Layer 2: Primary Capsules
    primary_caps = Conv2D(filters=8 * 16, kernel_size=9, strides=2, activation='relu', padding='valid', name='primarycap_conv2')(conv1)
    # primary_caps shape: [batch_size, height, width, filters]

    # Reshape to combine the channels into capsules
    primary_caps = Reshape(target_shape=[-1, 16], name='primarycap_reshape')(primary_caps)
    # primary_caps shape: [batch_size, num_capsules, dim_capsule]

    # Apply the squash activation function
    primary_caps = Lambda(squash, name='primarycap_squash')(primary_caps)

    # Layer 3: Digit Capsules
    digit_caps = CapsuleLayer(num_capsule=n_class, dim_capsule=16, routings=routings, name='digit_caps')(primary_caps)
    # digit_caps shape: [batch_size, num_capsule (n_class), dim_capsule]

    # Output Layer: Compute the length of each capsule to get class probabilities
    out_caps = Length(name='out_caps')(digit_caps)
    # out_caps shape: [batch_size, n_class]

    # Define the model
    model = Model(inputs=x, outputs=out_caps)
    return model


class Length(tf.keras.layers.Layer):
    def call(self, inputs, **kwargs):
        # Compute the length of vectors along the last axis
        return tf.sqrt(tf.reduce_sum(tf.square(inputs), axis=-1) + K.epsilon())


# Define input shape
input_shape = (IMG_SIZE, IMG_SIZE, 3)
n_class = 5  # 0 to 4 severity levels
routings = 3

# Instantiate the model
model = CapsNet(input_shape=input_shape, n_class=n_class, routings=routings)
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 64, 64, 3)	0
conv1 (Conv2D)	(None, 56, 56, 32)	7,808
primarycap_conv2 (Conv2D)	(None, 24, 24, 128)	331,904
primarycap_reshape (Reshape)	(None, 4608, 16)	0
primarycap_squash (Lambda)	(None, 4608, 16)	0
digit_caps (CapsuleLayer)	(None, 5, 16)	5,898,240
out_caps (Length)	(None, 5)	0

Total params: 6,237,952 (23.80 MB)

```
def margin_loss(y_true, y_pred):
    """
    Margin loss for CapsNet
    """
    lamb, m_plus, m_minus = 0.5, 0.9, 0.1
    T = y_true
    v = y_pred

    L = T * K.square(K.maximum(0., m_plus - v)) + \
        lamb * (1 - T) * K.square(K.maximum(0., v - m_minus))
    return K.mean(K.sum(L, axis=1))



model.compile(optimizer=optimizers.Adam(learning_rate=0.0005),
              loss=margin_loss,
              metrics=['accuracy'])



from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)

history = model.fit(X_train, y_train,
                    batch_size=8,  # Small batch size for resource constraints
                    epochs=20,
                    validation_data=(X_val, y_val),
                    callbacks=[early_stopping, lr_scheduler])
```

```
Epoch 1/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - accuracy: 0.4982 - loss: 0.3828 - val_accuracy: 0.6217 - val_loss: 0.2682 - learning_ra
Epoch 2/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 15s 11ms/step - accuracy: 0.6510 - loss: 0.2471 - val_accuracy: 0.6350 - val_loss: 0.2484 - learning_ra
Epoch 3/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6668 - loss: 0.2294 - val_accuracy: 0.6626 - val_loss: 0.2324 - learning_rat
Epoch 4/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.6909 - loss: 0.2139 - val_accuracy: 0.6560 - val_loss: 0.2285 - learning_rat
Epoch 5/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.7253 - loss: 0.1972 - val_accuracy: 0.6527 - val_loss: 0.2295 - learning_rat
Epoch 6/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7255 - loss: 0.1905 - val_accuracy: 0.6549 - val_loss: 0.2363 - learning_rat
Epoch 7/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7491 - loss: 0.1688 - val_accuracy: 0.6692 - val_loss: 0.2206 - learning_rat
Epoch 8/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.7842 - loss: 0.1547 - val_accuracy: 0.6781 - val_loss: 0.2228 - learning_rat
Epoch 9/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.8093 - loss: 0.1452 - val_accuracy: 0.6792 - val_loss: 0.2225 - learning_rat
Epoch 10/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 7s 14ms/step - accuracy: 0.8323 - loss: 0.1222 - val_accuracy: 0.6869 - val_loss: 0.2162 - learning_rat
Epoch 11/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.8582 - loss: 0.1080 - val_accuracy: 0.6914 - val_loss: 0.2164 - learning_rat
Epoch 12/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.8707 - loss: 0.0992 - val_accuracy: 0.6881 - val_loss: 0.2249 - learning_rat
Epoch 13/20
452/452 ━━━━━━━━━━━━━━━━━━━━ 7s 14ms/step - accuracy: 0.8841 - loss: 0.0956 - val_accuracy: 0.7046 - val_loss: 0.2189 - learning_rat
```

```
# Predictions
y_pred = model.predict(X_val)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_val, axis=1)

# Classification report
from sklearn.metrics import classification_report

print(classification_report(y_true_classes, y_pred_classes))
```

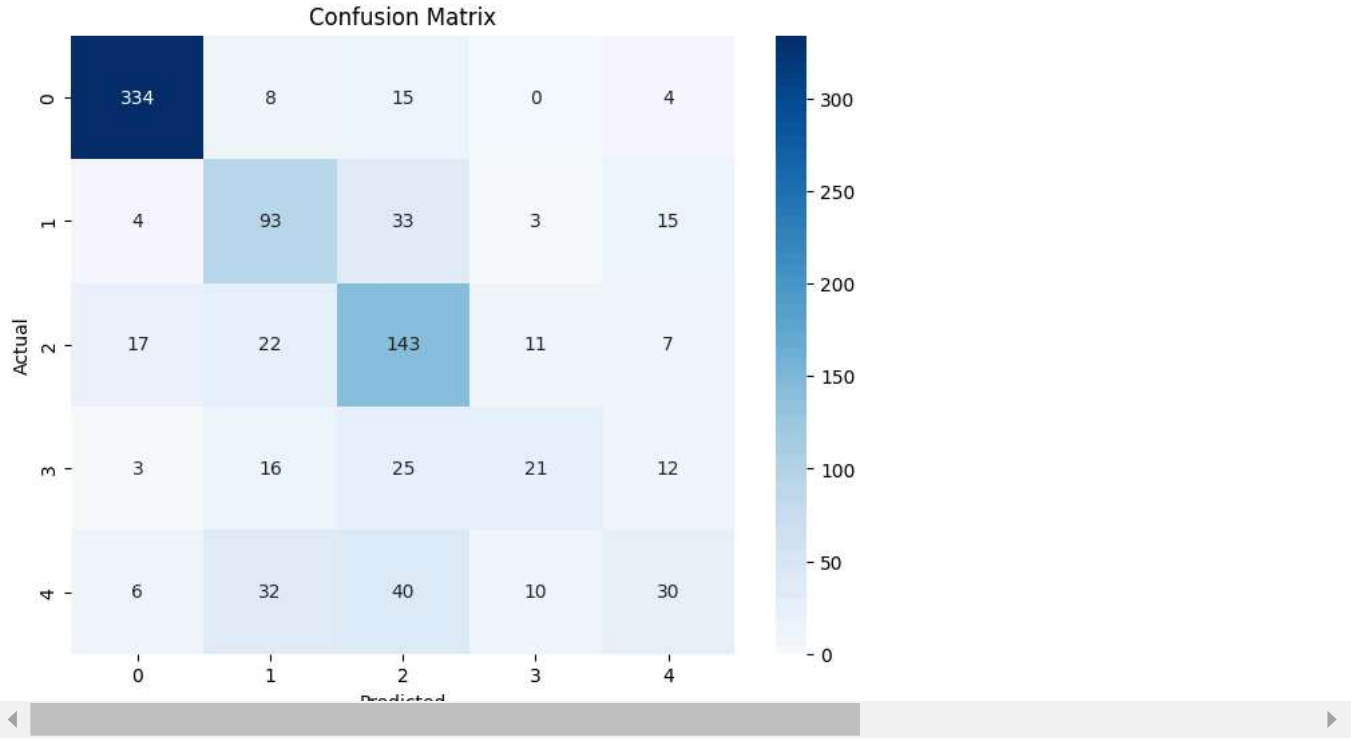
```
29/29 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step
              precision    recall  f1-score   support

           0       0.92      0.93      0.92       361
           1       0.54      0.63      0.58       148
           2       0.56      0.71      0.63       200
           3       0.47      0.27      0.34        77
           4       0.44      0.25      0.32       118

    accuracy                           0.69       904
   macro avg       0.59      0.56      0.56       904
weighted avg       0.68      0.69      0.67       904
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true_classes, y_pred_classes)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
def get_gradcam(model, image, cls, layer_name):
    """
    Grad-CAM for a specific class
    """
    grad_model = tf.keras.models.Model([model.inputs], [model.get_layer(layer_name).output, model.output])

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(np.array([image]))
        loss = predictions[:, cls]

    output = conv_outputs[0]
    grads = tape.gradient(loss, conv_outputs)[0]

    # Average gradients over the feature map
    weights = tf.reduce_mean(grads, axis=(0, 1))

    # Build a weighted combination of the feature maps
    cam = np.zeros(output.shape[0:2], dtype=np.float32)

    for i, w in enumerate(weights):
        cam += w * output[:, :, i]

    cam = cv2.resize(cam.numpy(), (IMG_SIZE, IMG_SIZE))
    cam = np.maximum(cam, 0)
    heatmap = cam / (cam.max() + K.epsilon())

    return heatmap


def display_gradcam(image, heatmap):
    """
    Display the image and overlay Grad-CAM heatmap
    """
    heatmap = cv2.applyColorMap(np.uint8(255 * heatmap), cv2.COLORMAP_JET)
    heatmap = np.float32(heatmap) / 255
    cam = heatmap + np.float32(image)
    cam = cam / np.max(cam)
    plt.figure(figsize=(6,6))
```

```
    plt.imshow(cam)
    plt.axis('off')
    plt.show()
```

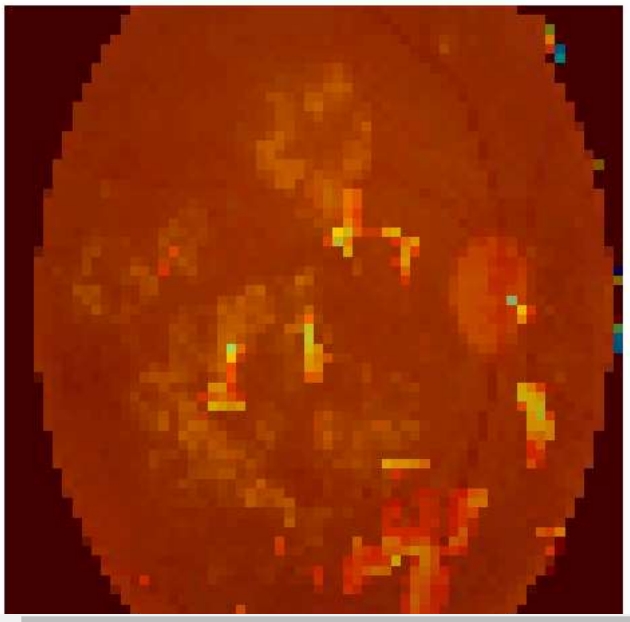
```
# Select a sample image from validation set
sample_index = 0  # Change index to view different samples
sample_image = X_val[sample_index]
sample_label = np.argmax(y_val[sample_index])

# Get model prediction
prediction = model.predict(np.array([sample_image]))
predicted_class = np.argmax(prediction[0])

# Get Grad-CAM heatmap
layer_name = 'conv1'  # Use the name of the first convolutional layer
heatmap = get_gradcam(model, sample_image, predicted_class, layer_name)

# Display Grad-CAM
display_gradcam(sample_image, heatmap)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
```



```
model.save('/content/drive/MyDrive/DiabeticRetinopathyDetection/DR/capsnet_model.keras')
```

Start coding or generate with AI.