```
# IMPORTANT: SOME KAGGLE DATA SOURCES ARE PRIVATE
# RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES.
Import kagglehub
Kagglehub.login()
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE’S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

Diabetic_retinopathy_detection_path = kagglehub.competition_download(‘diabetic-retinopathy-detection’)
Sovitrath_diabetic_retinopathy_224x224_gaussian_filtered_path = kagglehub.dataset_download(‘sovitrath/diabetic-retinopathy-224x224-gaussian-filtered’)

Print(‘Data source import complete.’)
From tensorflow import lite
Import tensorflow as tf
From tensorflow import keras
From tensorflow.keras import layers
Import numpy as np
Import pandas as pd
Import random, os
Import shutil
Import matplotlib.pyplot as plt
From matplotlib.image import imread
From keras.preprocessing.image import ImageDataGenerator
From tensorflow.keras.metrics import categorical_accuracy
From sklearn.model_selection import train_test_split
Df = pd.read_csv(r’../input/diabetic-retinopathy-224x224-gaussian-filtered/train.csv’)


Diagnosis_dict_binary = {
    0: ‘No_DR’,
    1: ‘DR’,
    2: ‘DR’,
    3: ‘DR’,
    4: ‘DR’
}


Diagnosis_dict = {
    0: ‘No_DR’,
    1: ‘Mild’,
    2: ‘Moderate’,
    3: ‘Severe’,
    4: ‘Proliferate_DR’,
}



Df[‘binary_type’] =  df[‘diagnosis’].map(diagnosis_dict_binary.get)
Df[‘type’] = df[‘diagnosis’].map(diagnosis_dict.get)
Df.head()
Df[‘type’].value_counts().plot(kind=’barh’)
Train_intermediate, val = train_test_split(df, test_size = 0.15, stratify = df[‘type’])
Train, test = train_test_split(train_intermediate, test_size = 0.15 / (1 – 0.15), stratify = train_intermediate[‘type’])
```

```
Print(“For Training Dataset :”)

Print(train[‘type’].value_counts(), ‘\n’)

Print(“For Testing Dataset :”)

Print(test[‘type’].value_counts(), ‘\n’)

Print(“For Validation Dataset :”)

Print(val[‘type’].value_counts(), ‘\n’)

Base_dir = ‘’


Train_dir = os.path.join(base_dir, ‘train’)

Val_dir = os.path.join(base_dir, ‘val’)

Test_dir = os.path.join(base_dir, ‘test’)


If os.path.exists(base_dir):

   Shutil.rmtree(base_dir)


If os.path.exists(train_dir):

   Shutil.rmtree(train_dir)

Os.makedirs(train_dir)


If os.path.exists(val_dir):

   Shutil.rmtree(val_dir)

Os.makedirs(val_dir)


If os.path.exists(test_dir):

   Shutil.rmtree(test_dir)

Os.makedirs(test_dir)

Src_dir = r’../input/diabetic-retinopathy-224x224-gaussian-
filtered/gaussian_filtered_images/gaussian_filtered_images’
```

```
For index, row in train.iterrows():
    Diagnosis = row[‘type’]
    Binary_diagnosis = row[‘binary_type’]
    Id_code = row[‘id_code’] + “.png”
    Srcfile = os.path.join(src_dir, diagnosis, id_code)
    Dstfile = os.path.join(train_dir, binary_diagnosis)
    Os.makedirs(dstfile, exist_ok = True)
    Shutil.copy(srcfile, dstfile)

For index, row in val.iterrows():
    Diagnosis = row[‘type’]
    Binary_diagnosis = row[‘binary_type’]
    Id_code = row[‘id_code’] + “.png”
    Srcfile = os.path.join(src_dir, diagnosis, id_code)
    Dstfile = os.path.join(val_dir, binary_diagnosis)
    Os.makedirs(dstfile, exist_ok = True)
    Shutil.copy(srcfile, dstfile)

For index, row in test.iterrows():
    Diagnosis = row[‘type’]
    Binary_diagnosis = row[‘binary_type’]
    Id_code = row[‘id_code’] + “.png”
    Srcfile = os.path.join(src_dir, diagnosis, id_code)
    Dstfile = os.path.join(test_dir, binary_diagnosis)
    Os.makedirs(dstfile, exist_ok = True)
    Shutil.copy(srcfile, dstfile)
Train_path = ‘train’
Val_path = ‘val’
```

```
Test_path = ‘test’


Train_batches = ImageDataGenerator(rescale = 1./255).flow_from_directory(train_path,
target_size=(224,224), shuffle = True)

Val_batches = ImageDataGenerator(rescale = 1./255).flow_from_directory(val_path,
target_size=(224,224), shuffle = True)

Test_batches = ImageDataGenerator(rescale = 1./255).flow_from_directory(test_path,
target_size=(224,224), shuffle = False)

Model = tf.keras.Sequential([

  Layers.Conv2D(8, (3,3), padding=”valid”, input_shape=(224,224,3), activation = ‘relu’),

  Layers.MaxPooling2D(pool_size=(2,2)),

  Layers.BatchNormalization(),


  Layers.Conv2D(16, (3,3), padding=”valid”, activation = ‘relu’),

  Layers.MaxPooling2D(pool_size=(2,2)),

  Layers.BatchNormalization(),


  Layers.Conv2D(32, (4,4), padding=”valid”, activation = ‘relu’),

  Layers.MaxPooling2D(pool_size=(2,2)),

  Layers.BatchNormalization(),


  Layers.Conv2D(64, (4,4), padding=”valid”, activation = ‘relu’),

  Layers.MaxPooling2D(pool_size=(2,2)),

  Layers.BatchNormalization(),


  Layers.Flatten(),

  Layers.Dense(64, activation = ‘relu’),

  Layers.Dropout(0.15),

  Layers.Dense(2, activation = ‘softmax’)
])


Model.compile(optimizer=tf.keras.optimizers.Adam(lr = 1e-5),
          Loss=tf.keras.losses.BinaryCrossentropy(),
          Metrics=[‘acc’])


History = model.fit(train_batches,
              Epochs=15,
              Validation_data=val_batches)


# Assuming you have a TensorFlow model named ‘model’
Model_json = model.to_json()


# Save the model architecture in JSON format
With open(“model.json”, “w”) as json_file:
    Json_file.write(model_json)


Weights = [np.array(w) for w in model.get_weights()]


# Save weights to a binary file
With open(“model_weights.bin”, “wb”) as binary_file:
    For weight in weights:
        Binary_file.write(weight.tobytes())
# Load Json
# Load the model architecture from the JSON file
With open(“model.json”, “r”) as json_file:
    Loaded_model_json = json_file.read()
```

```
Loaded_model = tf.keras.models.model_from_json(loaded_model_json)


# Load the weights into the model
With open("model_weights.bin", "rb") as bin_file:
  For layer in loaded_model.layers:
    If isinstance(layer, tf.keras.layers.BatchNormalization):
      # For BatchNormalization layers, load gamma and beta
      Gamma_beta = np.fromfile(bin_file, dtype=np.float32, count=2 * layer.input_shape[-1])
      Gamma = gamma_beta[:layer.input_shape[-1]]
      Beta = gamma_beta[layer.input_shape[-1]:]
      Moving_mean = np.fromfile(bin_file, dtype=np.float32, count=layer.input_shape[-1])
      Moving_variance = np.fromfile(bin_file, dtype=np.float32, count=layer.input_shape[-1])

      Layer.set_weights([gamma, beta, moving_mean, moving_variance])
    Else:
      # For other layers, load weights as usual
      Layer_weights = [np.fromfile(bin_file, dtype=np.float32, count=np.prod(param.shape)).reshape(param.shape)
              For param in layer.trainable_variables]
      Layer.set_weights(layer_weights)
Loaded_model.compile(optimizer=tf.keras.optimizers.Adam(lr = 1e-5),
       Loss=tf.keras.losses.BinaryCrossentropy(),
       Metrics=['acc'])
Print("Original: -\n")
Loss, acc = model.evaluate_generator(test_batches, verbose=1)
Print("Loss: ", loss)
Print(“Accuracy: “, acc)
Print(“Loaded: -\n”)
Loss, acc = loaded_model.evaluate_generator(test_batches, verbose=1)
Print(“Loss: “, loss)
Print(“Accuracy: “, acc)
Import tensorflow as tf
Import cv2
Import numpy as np
Import matplotlib.pyplot as plt


Def predict_class(path):
  Img = cv2.imread(path)

  RGBImg = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
  RGBImg= cv2.resize(RGBImg,(224,224))
  Plt.imshow(RGBImg)
  Image = np.array(RGBImg) / 255.0
#    new_model = tf.keras.models.load_model(“64x3-CNN.model”)
  Predict=loaded_model.predict(np.array([image]))
  Per=np.argmax(predict,axis=1)
  If per==1:
    Print(‘Diabetic Retinopathy Not Detected’)
  Else:
    Print(‘Diabetic Retinopathy Detected’)
Predict_class(‘/kaggle/input/diabetic-retinopathy-224x224-gaussian-filtered/gaussian_filtered_images/gaussian_filtered_images/Severe/1b495ac025b7.png’)
```