```
# Ocular Disease Recognition
from google.colab import drive
drive.mount('/content/drive')
```

```
import os
os.chdir('/content/drive/MyDrive/ODIR-5K')

!pip install tensorflow scikit-learn matplotlib seaborn
```

Show hidden output

```
import numpy as np
import pandas as pd
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.model_selection import train_test_split

data = pd.read_excel('/content/drive/MyDrive/ODIR-5K/data.xlsx')
```

```
print("Sample of the DataFrame:")
print(data.head())

train_image_dir = '/content/drive/MyDrive/ODIR-5K/Training Images/'
test_image_dir = '/content/drive/MyDrive/ODIR-5K/Testing Images/'
```

```
def load_images(image_dir, image_paths, target_size=(224, 224)):
    images = []
    valid_image_paths = []
    for path in image_paths:
        full_path = os.path.join(image_dir, path)
        if os.path.exists(full_path):
            img = load_img(full_path, target_size=target_size)
            img_array = img_to_array(img) / 255.0
            images.append(img_array)
            valid_image_paths.append(path)
    return np.array(images), valid_image_paths
```

```
train_image_paths = data['Left-Fundus'].values
X_train, valid_train_image_paths = load_images(train_image_dir, train_image_paths)

valid_indices = data[data['Left-Fundus'].isin(valid_train_image_paths)].index
y_train = data.loc[valid_indices, ['N', 'D', 'G', 'C', 'A', 'H', 'M', 'O']].sum(axis=1).values

print(f"Number of loaded images: {X_train.shape[0]}")
print(f"Number of corresponding labels: {y_train.shape[0]}")

X_train_split, X_val, y_train_split, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)
```

```
print("Training data shape:", X_train_split.shape)
print("Validation data shape:", X_val.shape)
```

```
Sample of the DataFrame:
   ID  Patient Age Patient Sex Left-Fundus Right-Fundus  \
0   0           69      Female  0_left.jpg  0_right.jpg
1   1           57        Male  1_left.jpg  1_right.jpg
2   2           42        Male  2_left.jpg  2_right.jpg
3   3           66        Male  3_left.jpg  3_right.jpg
4   4           53        Male  4_left.jpg  4_right.jpg

                          Left-Diagnostic Keywords  \
0                                         cataract
1                                   normal fundus
2  laser spot, moderate non proliferative retinopathy
3                                   normal fundus
4                     macular epiretinal membrane

               Right-Diagnostic Keywords  N  D  G  C  A  H  M  O
0                           normal fundus  0  0  0  1  0  0  0  0
1                           normal fundus  1  0  0  0  0  0  0  0
2  moderate non proliferative retinopathy  0  1  0  0  0  0  0  1
```

```
3        branch retinal artery occlusion  0  0  0  0  0  0  0  1
4       mild nonproliferative retinopathy  0  1  0  0  0  0  0  1
Number of loaded images: 157
Number of corresponding labels: 157
Training data shape: (125, 224, 224, 3)
Validation data shape: (32, 224, 224, 3)
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator


train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)


val_test_datagen = ImageDataGenerator()

train_generator = train_datagen.flow(X_train_split, y_train_split, batch_size=32)
val_generator = val_test_datagen.flow(X_val, y_val, batch_size=32)



# A. Convolutional Neural Network (CNN)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# CNN model
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # For binary classification
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history_cnn = cnn_model.fit(train_generator, epochs=10, validation_data=val_generator)


model_save_path = '/content/drive/MyDrive/ODIR-5K/trained_cnn_model.h5'
cnn_model.save(model_save_path)
```

```
ython3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument
t__(activity_regularizer=activity_regularizer, **kwargs)

ython3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `sup
_super_not_called()
━━━━━━━━━━━━━━━━━━━━ 18s 3s/step - accuracy: 0.8349 - loss: -3.4027 - val_accuracy: 0.9062 - val_loss: -24.2503

━━━━━━━━━━━━━━━━━━━━ 20s 4s/step - accuracy: 0.8904 - loss: -35.8254 - val_accuracy: 0.9062 - val_loss: -91.8782

━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - accuracy: 0.9363 - loss: -100.5947 - val_accuracy: 0.9062 - val_loss: -253.1686

━━━━━━━━━━━━━━━━━━━━ 20s 3s/step - accuracy: 0.9176 - loss: -257.8715 - val_accuracy: 0.9062 - val_loss: -596.6794

━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - accuracy: 0.9337 - loss: -605.6632 - val_accuracy: 0.9062 - val_loss: -1244.9421

━━━━━━━━━━━━━━━━━━━━ 20s 3s/step - accuracy: 0.9351 - loss: -1158.1207 - val_accuracy: 0.9062 - val_loss: -2341.5513

━━━━━━━━━━━━━━━━━━━━ 15s 3s/step - accuracy: 0.9098 - loss: -2548.9866 - val_accuracy: 0.9062 - val_loss: -4232.3760

━━━━━━━━━━━━━━━━━━━━ 20s 3s/step - accuracy: 0.8933 - loss: -4545.1738 - val_accuracy: 0.9062 - val_loss: -7215.5483

━━━━━━━━━━━━━━━━━━━━ 20s 3s/step - accuracy: 0.9350 - loss: -6627.0698 - val_accuracy: 0.9062 - val_loss: -11396.5420

━━━━━━━━━━━━━━━━━━━━ 15s 4s/step - accuracy: 0.9146 - loss: -12296.3848 - val_accuracy: 0.9062 - val_loss: -17803.4492
u are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We
```

```
# B. Support Vector Machine (SVM)

from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score

# Flatten the image data for SVM
X_train_flat = X_train_split.reshape(X_train_split.shape[0], -1)
X_val_flat = X_val.reshape(X_val.shape[0], -1)

scaler = StandardScaler()
X_train_flat = scaler.fit_transform(X_train_flat)
X_val_flat = scaler.transform(X_val_flat)


svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train_flat, y_train_split)


svm_predictions = svm_model.predict(X_val_flat)
svm_accuracy = accuracy_score(y_val, svm_predictions)
print(f"SVM Validation Accuracy: {svm_accuracy}")
```

```
SVM Validation Accuracy: 0.90625
```

```
# C. Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier


rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_flat, y_train_split)


rf_predictions = rf_model.predict(X_val_flat)
rf_accuracy = accuracy_score(y_val, rf_predictions)
print(f"Random Forest Validation Accuracy: {rf_accuracy}")
```

```
Random Forest Validation Accuracy: 0.875
```

```
# D. Logistic Regression

from sklearn.linear_model import LogisticRegression

lr_model = LogisticRegression()
lr_model.fit(X_train_flat, y_train_split)

lr_predictions = lr_model.predict(X_val_flat)
lr_accuracy = accuracy_score(y_val, lr_predictions)
print(f"Logistic Regression Validation Accuracy: {lr_accuracy}")
```

```
Logistic Regression Validation Accuracy: 0.75
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
# E. K-Nearest Neighbors (KNN)

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_flat, y_train_split)

knn_predictions = knn_model.predict(X_val_flat)
knn_accuracy = accuracy_score(y_val, knn_predictions)
```

```
print(f"KNN Validation Accuracy: {knn_accuracy}")
```

```
KNN Validation Accuracy: 0.90625
```

```
# F. Gradient Boosting Classifier

from sklearn.ensemble import GradientBoostingClassifier

gb_model = GradientBoostingClassifier()
gb_model.fit(X_train_flat, y_train_split)

gb_predictions = gb_model.predict(X_val_flat)
gb_accuracy = accuracy_score(y_val, gb_predictions)
print(f"Gradient Boosting Validation Accuracy: {gb_accuracy}")
```

Show hidden output

```
# Model Comparison

# Collect and compare accuracy of all models
model_accuracies = {
    'CNN': max(history_cnn.history['val_accuracy']),
    'SVM': svm_accuracy,
    'Random Forest': rf_accuracy,
    'Logistic Regression': lr_accuracy,
    'KNN': knn_accuracy,

}


for model, accuracy in model_accuracies.items():
    print(f"{model} Validation Accuracy: {accuracy * 100:.2f}%")
```

```
CNN Validation Accuracy: 90.62%
SVM Validation Accuracy: 90.62%
Random Forest Validation Accuracy: 87.50%
Logistic Regression Validation Accuracy: 75.00%
KNN Validation Accuracy: 90.62%
```

```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import StandardScaler

print(f"Test Image Directory: {test_image_dir}")


def load_images_safe(image_dir, image_paths, target_size=(224, 224)):
    images = []
    valid_paths = []

    for path in image_paths:
        image_path = os.path.join(image_dir, path)
        if os.path.exists(image_path):
            try:
                img = load_img(image_path, target_size=target_size)
                img_array = img_to_array(img) / 255.0
                images.append(img_array)
                valid_paths.append(path)
            except Exception as e:
                print(f"Error loading {image_path}: {e}")
        else:
            print(f"Warning: Image not found - {image_path}")


    return np.array(images), valid_paths


test_image_paths = data['Right-Fundus'].values
X_test, valid_test_paths = load_images_safe(test_image_dir, test_image_paths)


print(f"Number of images loaded: {len(valid_test_paths)}")
```

```
if X_test.shape[0] > 0:

    scaler = StandardScaler()
    X_test_flat = scaler.fit_transform(X_test.reshape(X_test.shape[0], -1))

    # Make predictions using the trained CNN model
    cnn_predictions = cnn_model.predict(X_test)
    print("Predictions:", cnn_predictions)
else:
    print("No valid images loaded. Please check file paths.")
```

Show hidden output

```
import os
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import StandardScaler

data_path = '/content/drive/MyDrive/ODIR-5K/data.xlsx'
df = pd.read_excel(data_path)

test_image_dir = '/content/drive/MyDrive/ODIR-5K/Testing Images/'

def load_images_safe(image_dir, image_paths, target_size=(224, 224)):
    images = []
    valid_paths = []

    for path in image_paths:
        image_path = os.path.join(image_dir, path)
        if os.path.exists(image_path):
            try:
                img = load_img(image_path, target_size=target_size)
                img_array = img_to_array(img) / 255.0
                images.append(img_array)
                valid_paths.append(path)
            except Exception as e:
                print(f"Error loading {image_path}: {e}")
        else:
            print(f"Warning: Image not found - {image_path}")


    return np.array(images), valid_paths

test_image_paths = df['Right-Fundus'].values
X_test, valid_test_paths = load_images_safe(test_image_dir, test_image_paths)

print(f"Number of images loaded: {len(valid_test_paths)}")

if X_test.shape[0] > 0:
    scaler = StandardScaler()
    X_test_flat = scaler.fit_transform(X_test.reshape(X_test.shape[0], -1))

    cnn_predictions = cnn_model.predict(X_test)
    print("Predictions:", cnn_predictions)
else:
    print("No valid images loaded. Please check file paths.")
```

Show hidden output

```
import os


def normalize_filename(filename):

    return filename.lower().replace('.jpeg', '.jpg').replace('.PNG', '.jpg')


actual_image_files = os.listdir(test_image_dir)


normalized_image_files = [normalize_filename(f) for f in actual_image_files]
```

```
print("Normalized image files in the directory:", normalized_image_files[:10])

def load_images_safe(image_dir, image_paths, target_size=(224, 224)):
    images = []
    valid_paths = []
    for path in image_paths:
        normalized_path = normalize_filename(path)
        full_path = os.path.join(image_dir, normalized_path)
        if normalized_path in normalized_image_files:
            img = load_img(full_path, target_size=target_size)
            img_array = img_to_array(img) / 255.0
            images.append(img_array)
            valid_paths.append(normalized_path)
        else:
            print(f"Warning: Image {normalized_path} not found. Skipping.")

    return np.array(images), valid_paths


X_test, valid_test_paths = load_images_safe(test_image_dir, data['Right-Fundus'].values)


print(f"Loaded {len(valid_test_paths)} valid test images.")
print(f"Shape of X_test: {X_test.shape}")

if X_test.shape[0] > 0:
    cnn_predictions = cnn_model.predict(X_test)
    print("Predictions:", cnn_predictions)
else:
    print("No valid images loaded.")
```

Show hidden output

```
files.upload()
```

Show hidden output

```
import cv2
image = cv2.imread("Diabetic-Retina.jpg")
target_size=(224, 224)
img = cv2.resize(image, target_size)
```

```
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import load_model

model_path = '/content/drive/MyDrive/ODIR-5K/trained_cnn_model.h5'


model = load_model(model_save_path)


def preprocess_image(img, target_size=(224, 224)):
    img_array = img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    return img_array


new_image = preprocess_image(img)


prediction = model.predict(new_image)

# Interpret the prediction (assuming one-hot encoded output)
# Example: Assuming classes are ['N', 'M', 'C', 'D', 'G', 'H', 'AMD', 'DR']
```