

Training and Evaluating Pre-trained CNN Models

Lecture 04

Prepared by

Dr. Hager Saleh

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.layers import Dense, Flatten, Dropout,
GlobalAveragePooling2D
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
import matplotlib.pyplot as plt
import os
import shutil
```

link to download dataset

<https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification/data>

```

original_dataset_dir = 'eye dataset' # Path to the dataset with
subfolders
base_dir = 'dataset' # Where you want to save new directories
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Create directories if they do not exist
for dir_path in [train_dir, validation_dir, test_dir]:
    os.makedirs(dir_path, exist_ok=True)

# Split ratios
train_ratio = 0.7
validation_ratio = 0.1
test_ratio = 0.2

# Function to split and copy files
def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in
os.listdir(class_dir) if os.path.isfile(os.path.join(class_dir, f))]
    train_images, temp_images = train_test_split(all_images,
test_size=(1 - train_ratio), random_state=42)
    validation_images, test_images = train_test_split(temp_images,
test_size=test_ratio/(validation_ratio + test_ratio), random_state=42)

    for images, folder_name in zip([train_images, validation_images,
test_images], ['train', 'validation', 'test']):
        dest_dir = os.path.join(dest_base_dir, folder_name,
os.path.basename(class_dir))
        os.makedirs(dest_dir, exist_ok=True)
        for file in images:
            shutil.copy(file, dest_dir)

# Iterate over each class directory and split the files
for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

def create_data_generators(train_dir, val_dir, test_dir,
img_size=(240, 240), batch_size=32):
    # Data augmentation for training dataset
    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=360,
        horizontal_flip=True,
        vertical_flip=True,
        brightness_range=(0.75, 1.25)
    )

```

```

# Only rescale for validation and testing
val_test_datagen = ImageDataGenerator(rescale=1./255)

# Generate augmented training images
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True
)

# Generate validation images
val_generator = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

# Generate test images
test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

return train_generator, val_generator, test_generator

```

Function to evaluate model

```

def evaluate_model(model_name, y_true, y_pred):

    labels = list(test_gen.class_indices.keys())
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    print("Accuracy:", accuracy)
    print("Precision :", precision)
    print("Recall:", recall)
    print("F1-score:" , f1)

```

```

print("Classification Report:")
print(classification_report(y_true, y_pred_classes,
target_names=labels))

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=labels, yticklabels=labels)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()

# Directories for datasets
train_dir = 'dataset/train'
val_dir = 'dataset/validation'
test_dir = 'dataset/test'
# Get input shape and number of classes
input_shape = (240, 240, 3)

# Create data generators
train_gen, val_gen, test_gen = create_data_generators(train_dir,
val_dir, test_dir)

num_classes = len(train_gen.class_indices)

Found 1519 images belonging to 2 classes.
Found 217 images belonging to 2 classes.
Found 436 images belonging to 2 classes.

```

ResNet50

```

def build_resnet8_model(input_shape, num_classes):
    base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',

```

```
loss='categorical_crossentropy',  
metrics=['accuracy'])
```

```
return model
```

```
model_resnet8 = build_resnet8_model(input_shape, num_classes)  
history = model_resnet8.fit(train_gen, epochs=50,  
validation_data=val_gen)
```

```
Epoch 1/50
```

```
48/48 _____ 44s 846ms/step - accuracy: 0.4944 - loss:  
0.8244 - val_accuracy: 0.6406 - val_loss: 0.6813
```

```
Epoch 2/50
```

```
48/48 _____ 39s 802ms/step - accuracy: 0.5228 - loss:  
0.6953 - val_accuracy: 0.5760 - val_loss: 0.6780
```

```
Epoch 3/50
```

```
48/48 _____ 50s 1s/step - accuracy: 0.4952 - loss:  
0.6972 - val_accuracy: 0.5622 - val_loss: 0.6889
```

```
Epoch 4/50
```

```
48/48 _____ 43s 883ms/step - accuracy: 0.5054 - loss:  
0.6918 - val_accuracy: 0.4931 - val_loss: 0.6916
```

```
Epoch 5/50
```

```
48/48 _____ 39s 804ms/step - accuracy: 0.5016 - loss:  
0.6930 - val_accuracy: 0.4931 - val_loss: 0.6863
```

```
Epoch 6/50
```

```
48/48 _____ 39s 810ms/step - accuracy: 0.5459 - loss:  
0.6907 - val_accuracy: 0.6498 - val_loss: 0.6898
```

```
Epoch 7/50
```

```
48/48 _____ 42s 874ms/step - accuracy: 0.5026 - loss:  
0.6934 - val_accuracy: 0.5438 - val_loss: 0.6922
```

```
Epoch 8/50
```

```
48/48 _____ 41s 846ms/step - accuracy: 0.5170 - loss:  
0.6928 - val_accuracy: 0.4931 - val_loss: 0.6895
```

```
Epoch 9/50
```

```
48/48 _____ 41s 844ms/step - accuracy: 0.5068 - loss:  
0.6917 - val_accuracy: 0.6498 - val_loss: 0.6868
```

```
Epoch 10/50
```

```
48/48 _____ 39s 803ms/step - accuracy: 0.5177 - loss:  
0.6935 - val_accuracy: 0.6083 - val_loss: 0.6850
```

```
Epoch 11/50
```

```
48/48 _____ 40s 839ms/step - accuracy: 0.5272 - loss:  
0.6909 - val_accuracy: 0.5300 - val_loss: 0.6818
```

```
Epoch 12/50
```

```
48/48 _____ 40s 837ms/step - accuracy: 0.5075 - loss:  
0.6910 - val_accuracy: 0.6406 - val_loss: 0.6850
```

```
Epoch 13/50
```

```
48/48 _____ 39s 810ms/step - accuracy: 0.5460 - loss:  
0.6916 - val_accuracy: 0.6498 - val_loss: 0.6781
```

```
Epoch 14/50
```

```
48/48 _____ 39s 804ms/step - accuracy: 0.5944 - loss:
```

0.6809 - val_accuracy: 0.6682 - val_loss: 0.6731
Epoch 15/50
48/48 _____ 39s 809ms/step - accuracy: 0.5391 - loss:
0.6868 - val_accuracy: 0.6037 - val_loss: 0.6724
Epoch 16/50
48/48 _____ 39s 813ms/step - accuracy: 0.5616 - loss:
0.6852 - val_accuracy: 0.6682 - val_loss: 0.6696
Epoch 17/50
48/48 _____ 39s 802ms/step - accuracy: 0.5876 - loss:
0.6849 - val_accuracy: 0.6452 - val_loss: 0.6749
Epoch 18/50
48/48 _____ 39s 802ms/step - accuracy: 0.5627 - loss:
0.6867 - val_accuracy: 0.4977 - val_loss: 0.6761
Epoch 19/50
48/48 _____ 38s 791ms/step - accuracy: 0.5160 - loss:
0.6911 - val_accuracy: 0.5760 - val_loss: 0.6859
Epoch 20/50
48/48 _____ 38s 795ms/step - accuracy: 0.5166 - loss:
0.6928 - val_accuracy: 0.6590 - val_loss: 0.6687
Epoch 21/50
48/48 _____ 39s 811ms/step - accuracy: 0.5686 - loss:
0.6824 - val_accuracy: 0.6544 - val_loss: 0.6662
Epoch 22/50
48/48 _____ 40s 821ms/step - accuracy: 0.5825 - loss:
0.6790 - val_accuracy: 0.6359 - val_loss: 0.6661
Epoch 23/50
48/48 _____ 39s 810ms/step - accuracy: 0.5679 - loss:
0.6829 - val_accuracy: 0.6590 - val_loss: 0.6649
Epoch 24/50
48/48 _____ 39s 812ms/step - accuracy: 0.5601 - loss:
0.6866 - val_accuracy: 0.6590 - val_loss: 0.6648
Epoch 25/50
48/48 _____ 40s 825ms/step - accuracy: 0.5527 - loss:
0.6800 - val_accuracy: 0.6590 - val_loss: 0.6636
Epoch 26/50
48/48 _____ 40s 838ms/step - accuracy: 0.5851 - loss:
0.6758 - val_accuracy: 0.6313 - val_loss: 0.6644
Epoch 27/50
48/48 _____ 40s 832ms/step - accuracy: 0.5989 - loss:
0.6776 - val_accuracy: 0.6636 - val_loss: 0.6610
Epoch 28/50
48/48 _____ 41s 847ms/step - accuracy: 0.5526 - loss:
0.6854 - val_accuracy: 0.6590 - val_loss: 0.6624
Epoch 29/50
48/48 _____ 40s 837ms/step - accuracy: 0.5945 - loss:
0.6762 - val_accuracy: 0.6313 - val_loss: 0.6635
Epoch 30/50
48/48 _____ 41s 843ms/step - accuracy: 0.5881 - loss:
0.6800 - val_accuracy: 0.6636 - val_loss: 0.6602

Epoch 31/50
48/48 ————— 42s 865ms/step - accuracy: 0.5934 - loss: 0.6756 - val_accuracy: 0.6498 - val_loss: 0.6624
Epoch 32/50
48/48 ————— 41s 846ms/step - accuracy: 0.5851 - loss: 0.6806 - val_accuracy: 0.6590 - val_loss: 0.6622
Epoch 33/50
48/48 ————— 41s 853ms/step - accuracy: 0.5834 - loss: 0.6816 - val_accuracy: 0.5991 - val_loss: 0.6679
Epoch 34/50
48/48 ————— 40s 837ms/step - accuracy: 0.5524 - loss: 0.6861 - val_accuracy: 0.6083 - val_loss: 0.6661
Epoch 35/50
48/48 ————— 40s 841ms/step - accuracy: 0.5894 - loss: 0.6804 - val_accuracy: 0.6636 - val_loss: 0.6617
Epoch 36/50
48/48 ————— 40s 849ms/step - accuracy: 0.5812 - loss: 0.6854 - val_accuracy: 0.6406 - val_loss: 0.6642
Epoch 37/50
48/48 ————— 40s 833ms/step - accuracy: 0.5808 - loss: 0.6781 - val_accuracy: 0.6636 - val_loss: 0.6601
Epoch 38/50
48/48 ————— 40s 826ms/step - accuracy: 0.5854 - loss: 0.6762 - val_accuracy: 0.6590 - val_loss: 0.6610
Epoch 39/50
48/48 ————— 41s 860ms/step - accuracy: 0.5815 - loss: 0.6792 - val_accuracy: 0.6636 - val_loss: 0.6609
Epoch 40/50
48/48 ————— 41s 854ms/step - accuracy: 0.5881 - loss: 0.6750 - val_accuracy: 0.6590 - val_loss: 0.6599
Epoch 41/50
48/48 ————— 41s 847ms/step - accuracy: 0.5930 - loss: 0.6762 - val_accuracy: 0.6037 - val_loss: 0.6660
Epoch 42/50
48/48 ————— 40s 832ms/step - accuracy: 0.5756 - loss: 0.6778 - val_accuracy: 0.6636 - val_loss: 0.6590
Epoch 43/50
48/48 ————— 40s 838ms/step - accuracy: 0.5899 - loss: 0.6821 - val_accuracy: 0.6221 - val_loss: 0.6636
Epoch 44/50
48/48 ————— 42s 879ms/step - accuracy: 0.5789 - loss: 0.6787 - val_accuracy: 0.6590 - val_loss: 0.6609
Epoch 45/50
48/48 ————— 41s 850ms/step - accuracy: 0.5918 - loss: 0.6759 - val_accuracy: 0.6498 - val_loss: 0.6617
Epoch 46/50
48/48 ————— 41s 844ms/step - accuracy: 0.5987 - loss: 0.6725 - val_accuracy: 0.6037 - val_loss: 0.6645
Epoch 47/50

```
48/48 _____ 41s 851ms/step - accuracy: 0.5650 - loss: 0.6869 - val_accuracy: 0.6590 - val_loss: 0.6622
```

```
Epoch 48/50
```

```
48/48 _____ 42s 875ms/step - accuracy: 0.5563 - loss: 0.6817 - val_accuracy: 0.5991 - val_loss: 0.6688
```

```
Epoch 49/50
```

```
48/48 _____ 42s 874ms/step - accuracy: 0.6002 - loss: 0.6734 - val_accuracy: 0.5945 - val_loss: 0.6690
```

```
Epoch 50/50
```

```
48/48 _____ 44s 907ms/step - accuracy: 0.5828 - loss: 0.6841 - val_accuracy: 0.6636 - val_loss: 0.6620
```

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

```
# Plot training & validation accuracy values
```

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

```
# Accuracy curve
```

```
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
```

```
plt.title('Model Accuracy')
```

```
plt.legend()
```

```
# Loss curve
```

```
plt.subplot(1, 2, 2)
```

```
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Validation Loss')
```

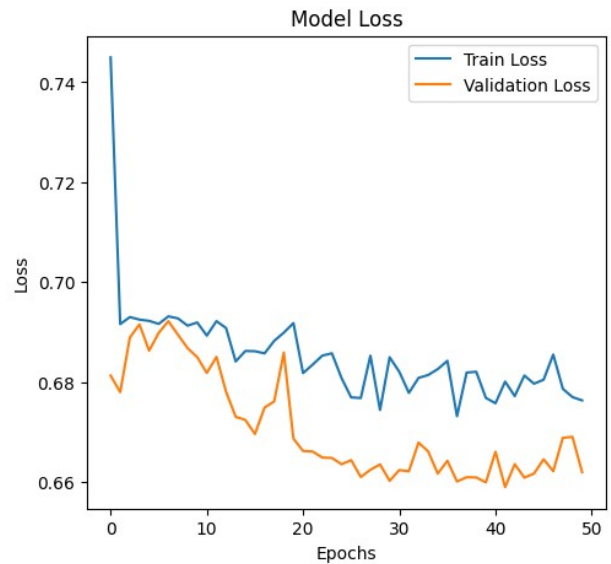
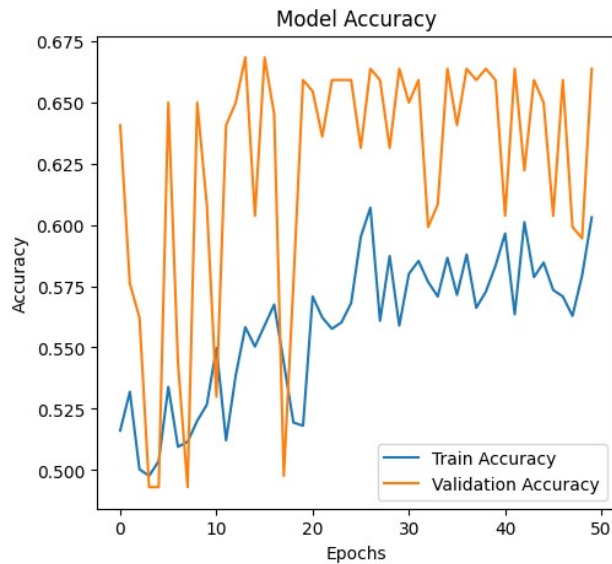
```
plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')
```

```
plt.title('Model Loss')
```

```
plt.legend()
```

```
plt.show()
```



```
y_pred=model_resnet8.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('ResNet50', y_true, y_pred_classes)
```

14/14 ————— 11s 740ms/step

Accuracy: 0.6834862385321101

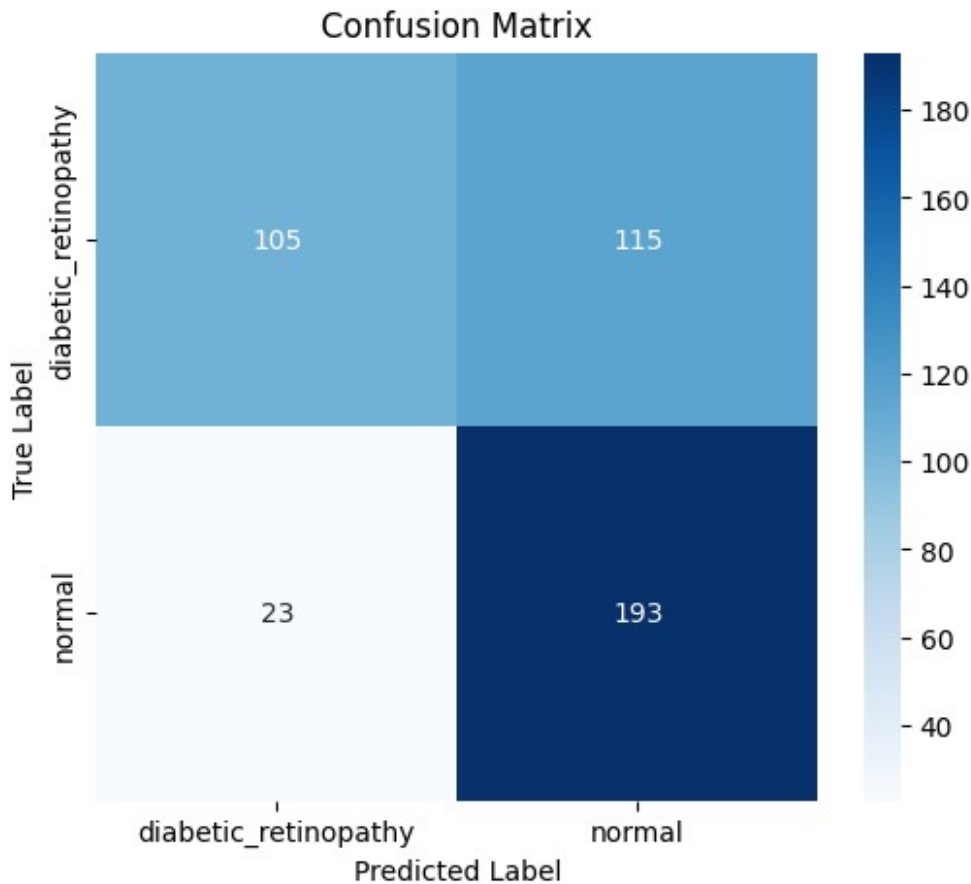
Precision : 0.7243564205290123

Recall: 0.6834862385321101

F1-score: 0.6694337718037822

Classification Report:

	precision	recall	f1-score	support
diabetic_retinopathy	0.82	0.48	0.60	220
normal	0.63	0.89	0.74	216
accuracy			0.68	436
macro avg	0.72	0.69	0.67	436
weighted avg	0.72	0.68	0.67	436



```
def build_efficientnetb0_model(input_shape, num_classes):
    base_model = EfficientNetB0(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False # Freeze the base model

    model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

efficientnetb0_model = build_efficientnetb0_model(input_shape,
num_classes)
```

```
history = efficientnetb0_model.fit(train_gen, epochs=50,  
validation_data=val_gen)
```

Epoch 1/50

48/48 _____ 30s 524ms/step - accuracy: 0.5406 - loss:
0.7557 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 2/50

48/48 _____ 26s 534ms/step - accuracy: 0.4780 - loss:
0.6963 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 3/50

48/48 _____ 24s 503ms/step - accuracy: 0.4899 - loss:
0.6945 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 4/50

48/48 _____ 24s 497ms/step - accuracy: 0.5039 - loss:
0.6936 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 5/50

48/48 _____ 25s 513ms/step - accuracy: 0.4935 - loss:
0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 6/50

48/48 _____ 24s 507ms/step - accuracy: 0.5043 - loss:
0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 7/50

48/48 _____ 24s 504ms/step - accuracy: 0.5110 - loss:
0.6930 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 8/50

48/48 _____ 25s 510ms/step - accuracy: 0.4920 - loss:
0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 9/50

48/48 _____ 24s 508ms/step - accuracy: 0.4927 - loss:
0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 10/50

48/48 _____ 25s 521ms/step - accuracy: 0.5042 - loss:
0.6931 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 11/50

48/48 _____ 27s 557ms/step - accuracy: 0.4993 - loss:
0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 12/50

48/48 _____ 25s 513ms/step - accuracy: 0.4943 - loss:
0.6935 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 13/50

48/48 _____ 24s 502ms/step - accuracy: 0.5000 - loss:
0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 14/50

48/48 _____ 28s 587ms/step - accuracy: 0.5046 - loss:
0.6931 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 15/50

48/48 _____ 26s 550ms/step - accuracy: 0.5051 - loss:
0.6935 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 16/50

48/48 _____ 25s 514ms/step - accuracy: 0.5071 - loss:

0.6931 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 17/50
48/48 _____ 25s 517ms/step - accuracy: 0.4822 - loss:
0.6938 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 18/50
48/48 _____ 25s 520ms/step - accuracy: 0.5134 - loss:
0.6929 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 19/50
48/48 _____ 25s 511ms/step - accuracy: 0.5020 - loss:
0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 20/50
48/48 _____ 25s 519ms/step - accuracy: 0.5199 - loss:
0.6928 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 21/50
48/48 _____ 25s 510ms/step - accuracy: 0.4905 - loss:
0.6935 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 22/50
48/48 _____ 25s 523ms/step - accuracy: 0.5102 - loss:
0.6930 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 23/50
48/48 _____ 24s 508ms/step - accuracy: 0.4901 - loss:
0.6935 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 24/50
48/48 _____ 25s 522ms/step - accuracy: 0.5236 - loss:
0.6927 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 25/50
48/48 _____ 25s 511ms/step - accuracy: 0.4938 - loss:
0.6934 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 26/50
48/48 _____ 25s 510ms/step - accuracy: 0.4948 - loss:
0.6934 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 27/50
48/48 _____ 25s 514ms/step - accuracy: 0.4970 - loss:
0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 28/50
48/48 _____ 25s 521ms/step - accuracy: 0.5104 - loss:
0.6930 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 29/50
48/48 _____ 24s 505ms/step - accuracy: 0.4994 - loss:
0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 30/50
48/48 _____ 24s 507ms/step - accuracy: 0.4865 - loss:
0.6936 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 31/50
48/48 _____ 24s 509ms/step - accuracy: 0.5144 - loss:
0.6929 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 32/50
48/48 _____ 25s 512ms/step - accuracy: 0.5210 - loss:
0.6927 - val_accuracy: 0.5069 - val_loss: 0.6931

Epoch 33/50
48/48 _____ 25s 528ms/step - accuracy: 0.5180 - loss: 0.6928 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 34/50
48/48 _____ 26s 536ms/step - accuracy: 0.4993 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 35/50
48/48 _____ 29s 594ms/step - accuracy: 0.5108 - loss: 0.6930 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 36/50
48/48 _____ 27s 562ms/step - accuracy: 0.5178 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 37/50
48/48 _____ 28s 581ms/step - accuracy: 0.5024 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 38/50
48/48 _____ 25s 514ms/step - accuracy: 0.5225 - loss: 0.6927 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 39/50
48/48 _____ 24s 508ms/step - accuracy: 0.4988 - loss: 0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 40/50
48/48 _____ 25s 511ms/step - accuracy: 0.5029 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 41/50
48/48 _____ 24s 507ms/step - accuracy: 0.5095 - loss: 0.6931 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 42/50
48/48 _____ 25s 519ms/step - accuracy: 0.4943 - loss: 0.6933 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 43/50
48/48 _____ 25s 526ms/step - accuracy: 0.5170 - loss: 0.6929 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 44/50
48/48 _____ 25s 517ms/step - accuracy: 0.5030 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 45/50
48/48 _____ 24s 508ms/step - accuracy: 0.5046 - loss: 0.6931 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 46/50
48/48 _____ 24s 509ms/step - accuracy: 0.4994 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 47/50
48/48 _____ 24s 506ms/step - accuracy: 0.5018 - loss: 0.6932 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 48/50
48/48 _____ 25s 510ms/step - accuracy: 0.5245 - loss: 0.6927 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 49/50

```
48/48 ————— 25s 515ms/step - accuracy: 0.5105 - loss:
0.6930 - val_accuracy: 0.5069 - val_loss: 0.6931
Epoch 50/50
48/48 ————— 24s 502ms/step - accuracy: 0.4926 - loss:
0.6935 - val_accuracy: 0.5069 - val_loss: 0.6931
```

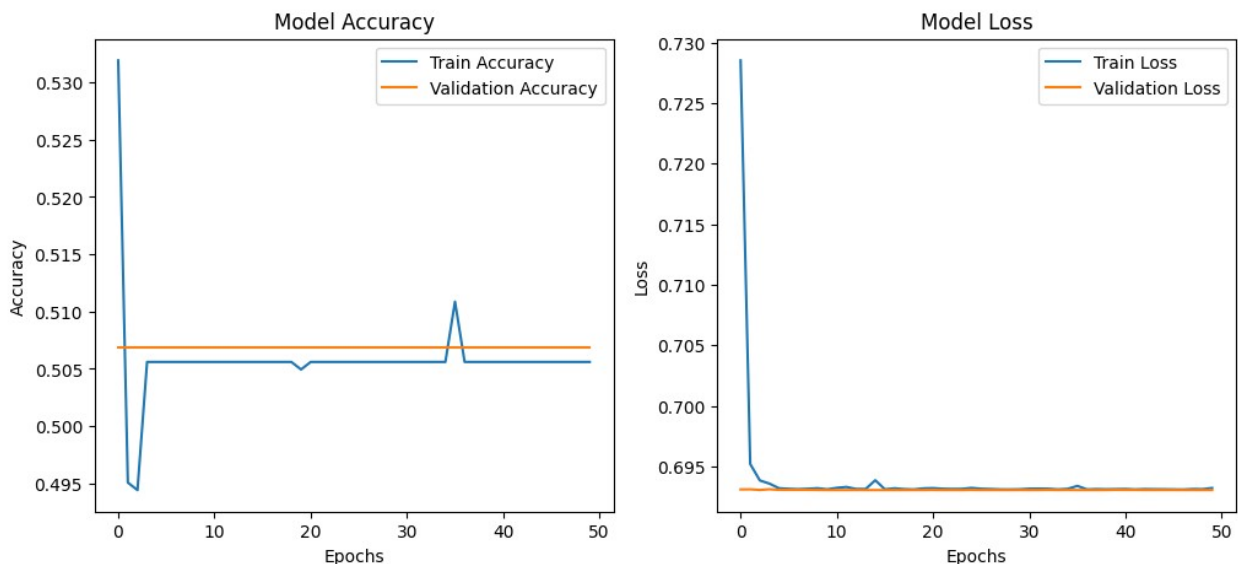
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 5))

# Accuracy curve
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()

# Loss curve
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.legend()

plt.show()
```



```

y_pred=efficientnetb0_model.predict(test_gen)
y_true = test_gen.classes
y_pred_classes = np.argmax(y_pred, axis=1)
evaluate_model('EfficientNetB0', y_true, y_pred_classes)

```

14/14 ————— 7s 414ms/step

Accuracy: 0.5045871559633027

Precision : 0.2546081979631344

Recall: 0.5045871559633027

F1-score: 0.33844260460953235

Classification Report:

	precision	recall	f1-score	support
diabetic_retinopathy	0.50	1.00	0.67	220
normal	0.00	0.00	0.00	216
accuracy			0.50	436
macro avg	0.25	0.50	0.34	436
weighted avg	0.25	0.50	0.34	436

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

```

C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\sklearn\metrics_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```

_warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

```

