Exercise – 1

Drawing Confusion Matrix and Computation of Different Metrics for Classification

Aim:

To draw the Confusion Matrix and Computation of Different Metrics for Classification

Python Libraries Used:

- pycm

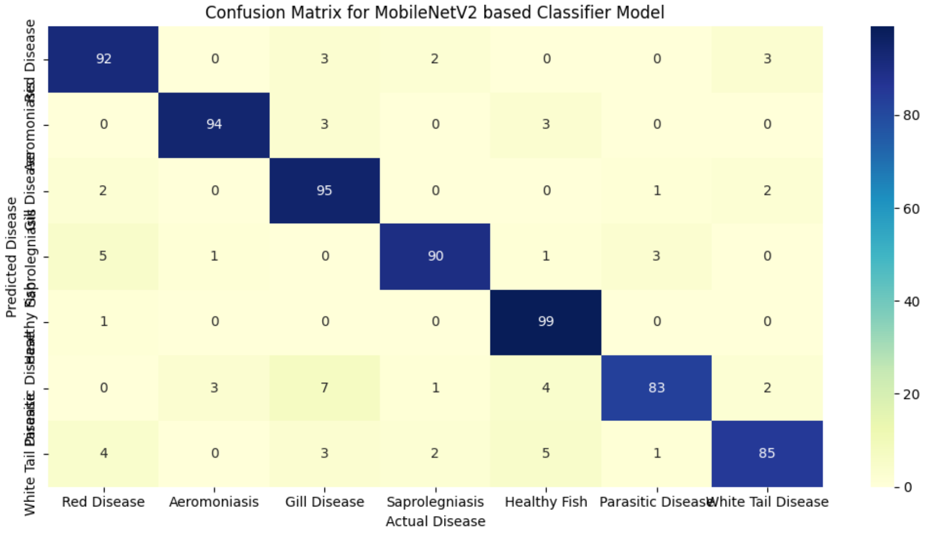
Algorithm:

1. Start by defining the confusion matrix as a nested dictionary, with keys representing both the actual and predicted classes of fish diseases.
2. Populate this dictionary with the given values, where each inner key-value pair represents the count of instances correctly or incorrectly classified.
3. Utilize the `ConfusionMatrix` class from the `pycm` library to instantiate a confusion matrix object using the prepared dictionary.
4. Print the created confusion matrix object, which automatically displays the matrix and calculates various performance metrics such as accuracy, precision, and recall.

Code:

```python
from pycm import ConfusionMatrix
# Standardized labels: Title Case for consistency
confusion_matrix = {
   "Red Disease": {"Red Disease": 92, "Aeromoniasis": 0, "Gill Disease": 3, "Saprolegniasis": 2, "Healthy Fish": 0, "Parasitic Disease": 0, "White Tail Disease": 3},
   "Aeromoniasis": {"Red Disease": 0, "Aeromoniasis": 94, "Gill Disease": 3, "Saprolegniasis": 0, "Healthy Fish": 3, "Parasitic Disease": 0, "White Tail Disease": 0},
   "Gill Disease": {"Red Disease": 2, "Aeromoniasis": 0, "Gill Disease": 95, "Saprolegniasis": 0, "Healthy Fish": 0, "Parasitic Disease": 1, "White Tail Disease": 2},
   "Saprolegniasis": {"Red Disease": 5, "Aeromoniasis": 1, "Gill Disease": 0, "Saprolegniasis": 90, "Healthy Fish": 1, "Parasitic Disease": 3, "White Tail Disease": 0},
   "Healthy Fish": {"Red Disease": 1, "Aeromoniasis": 0, "Gill Disease": 0, "Saprolegniasis": 0, "Healthy Fish": 99, "Parasitic Disease": 0, "White Tail Disease": 0},
   "Parasitic Disease": {"Red Disease": 0, "Aeromoniasis": 3, "Gill Disease": 7, "Saprolegniasis": 1, "Healthy Fish": 4, "Parasitic Disease": 83, "White Tail Disease": 2},
   "White Tail Disease": {"Red Disease": 4, "Aeromoniasis": 0, "Gill Disease": 3, "Saprolegniasis": 2, "Healthy Fish": 5, "Parasitic Disease": 1, "White Tail Disease": 85}
}
# Create confusion matrix object
cm2 = ConfusionMatrix(matrix=confusion_matrix)
# Display matrix
print(cm2)
```

Output:



Confusion Matrix for MobileNetV2 based Classifier Model

	Red Disease	Aeromoniasis	Gill Disease	Saprolegniasis	Healthy Fish	Parasitic Disease	White Tail Disease
Red Disease	92	0	3	2	0	0	3
Aeromoniasis	0	94	3	0	3	0	0
Gill Disease	2	0	95	0	0	1	2
Saprolegniasis	5	1	0	90	1	3	0
Healthy Fish	1	0	0	0	99	0	0
Parasitic Disease	0	3	7	1	4	83	2
White Tail Disease	4	0	3	2	5	1	85

Result:

Thus, we have successfully generated the confusion matrix for the fish disease classification dataset. The matrix visually represents the model's performance, showing the number of correct and incorrect predictions for each class. The accompanying metrics provide a quantitative evaluation of the model's accuracy, precision, and other key performance indicators.

Exercise – 2

Layer Visualization and Feature maps in CNN

Aim:

To visualize and plot the feature maps extracted by the hidden layers of a pre-trained VGG16 Convolutional Neural Network (CNN) for a given input image.

Python Libraries Used:

- tensorflow
- keras
- matplotlib
- numpy

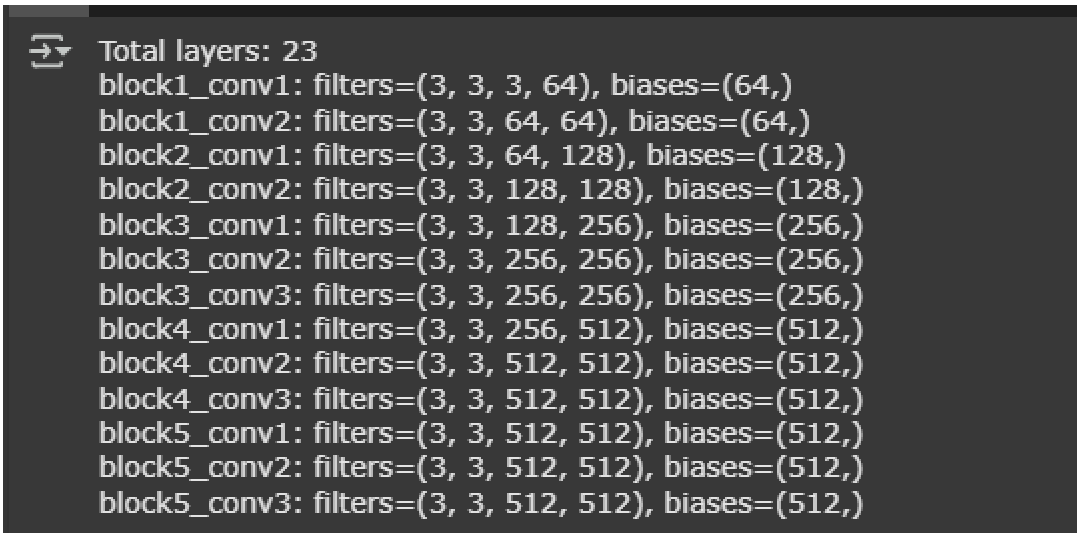
Algorithm:

- Load the pre-trained VGG16 model and define a new model that outputs the feature maps from specific hidden layers.
- Load and preprocess a sample image to match the input requirements of the VGG16 model, including resizing and converting it into an array with the correct dimensions.
- Use the modified model to predict the feature maps for the preprocessed input image.
- Iterate through the generated feature maps from each selected layer.
- For each feature map, plot a selection of its channels in a grid format using grayscale to visualize the features learned by that layer

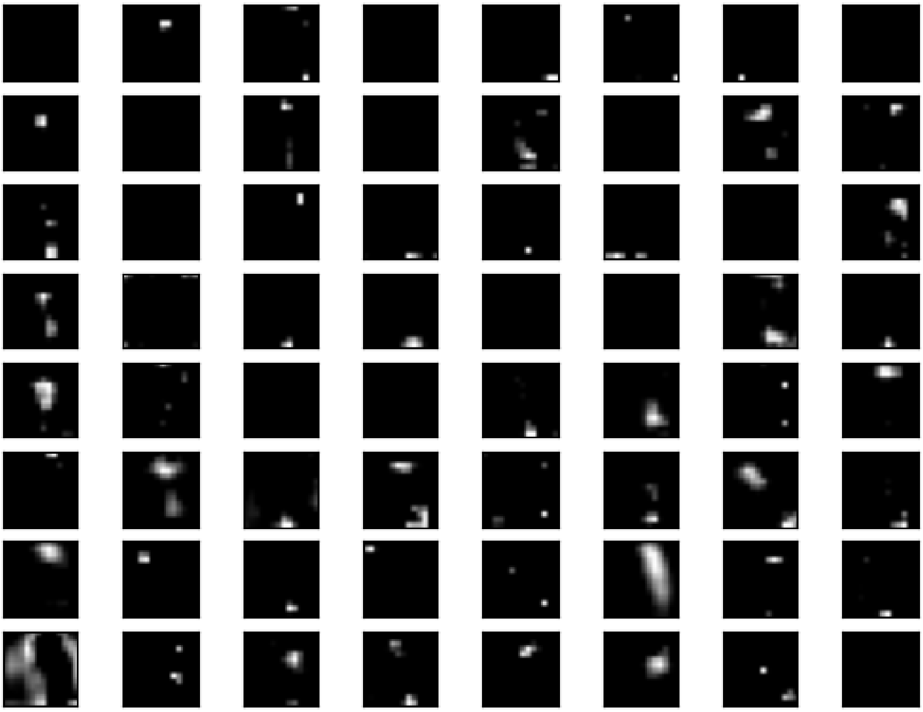
Code:

```
import tensorflow as tf
from tensorflow import keras
from keras.applications.vgg16 import VGG16
from keras.utils import plot_model

model = VGG16()
plot_model(model, to_file='densenet121_plot.png', show_shapes=True, show_layer_names=True)
# summarize convolutional filter shapes
print(f"Total layers: {len(model.layers)}")
for layer in model.layers:
    if 'conv' not in layer.name:
        continue
    weights = layer.get_weights()
    if len(weights) == 1:
        filters = weights[0]
        print(f"{layer.name}: filters={filters.shape}, no bias")
    elif len(weights) == 2:
        filters, biases = weights
        print(f"{layer.name}: filters={filters.shape}, biases={biases.shape}")
    else:
        print(f"{layer.name}: unexpected weights format")
```

Output:



```
Total layers: 23
block1_conv1: filters=(3, 3, 3, 64), biases=(64,)
block1_conv2: filters=(3, 3, 64, 64), biases=(64,)
block2_conv1: filters=(3, 3, 64, 128), biases=(128,)
block2_conv2: filters=(3, 3, 128, 128), biases=(128,)
block3_conv1: filters=(3, 3, 128, 256), biases=(256,)
block3_conv2: filters=(3, 3, 256, 256), biases=(256,)
block3_conv3: filters=(3, 3, 256, 256), biases=(256,)
block4_conv1: filters=(3, 3, 256, 512), biases=(512,)
block4_conv2: filters=(3, 3, 512, 512), biases=(512,)
block4_conv3: filters=(3, 3, 512, 512), biases=(512,)
block5_conv1: filters=(3, 3, 512, 512), biases=(512,)
block5_conv2: filters=(3, 3, 512, 512), biases=(512,)
block5_conv3: filters=(3, 3, 512, 512), biases=(512,)
```



Result:

The feature maps from the hidden layers of the VGG16 model were successfully extracted and visualized. Each plot represents how a specific convolutional filter in that layer responded to the input image, providing a visual understanding of the hierarchical features learned by the CNN, from simple edges and textures to more complex patterns.

Exercise – 3

Different Types of Data Augmentation Techniques

Aim:

To perform image augmentation on a sample image by applying a height shift transformation and visualizing the augmented results.

Python Libraries Used:

- numpy
- tensorflow
- keras
- matplotlib

Algorithm:

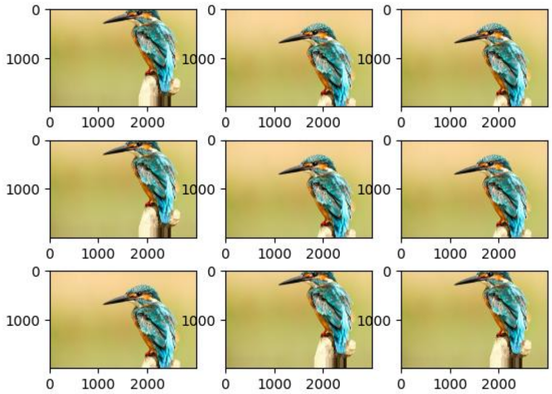
1. Load an image and convert it into a NumPy array, expanding its dimensions to represent a single sample.
2. Create an `ImageDataGenerator` object, specifying the `height_shift_range` parameter to define the augmentation transformation.
3. Prepare an iterator from the generator to produce batches of augmented images.
4. Loop nine times to generate a batch of augmented images in each iteration.
5. Plot each generated image in a 3x3 grid to visualize the effect of the height shift augmentation.

Code:

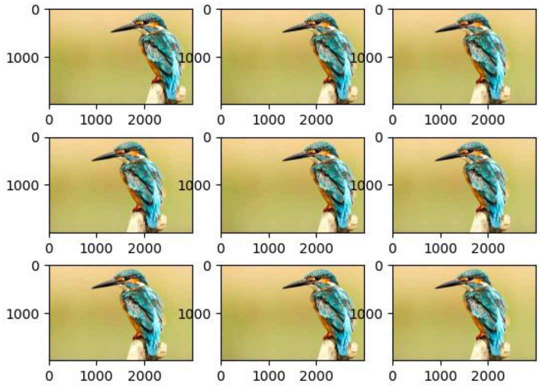
```
from numpy import expand_dims
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
from matplotlib import pyplot

# load the image
img = load_img('bird.jpeg')
# convert to numpy array
data = img_to_array(img)
# expand dimension to one sample
samples = expand_dims(data, 0)

# create image data augmentation generator
datagen = ImageDataGenerator(height_shift_range=[-200,200])

# prepare iterator
it = datagen.flow(samples, batch_size=1)

# generate samples and plot
for i in range(9):
    pyplot.subplot(330 + 1 + i)
```

```
    # generate batch of images
    batch = next(it)
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    pyplot.imshow(image)

pyplot.show()
# create image data augmentation generator
datagen = ImageDataGenerator(height_shift_range=0.5)
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()

datagen = ImageDataGenerator(brightness_range=[0.2,1.0])
# prepare iterator
it = datagen.flow(samples, batch_size=1)
# generate samples and plot
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # generate batch of images
    batch = it.next()
    # convert to unsigned integers for viewing
    image = batch[0].astype('uint8')
    # plot raw pixel data
    pyplot.imshow(image)
# show the figure
pyplot.show()
```

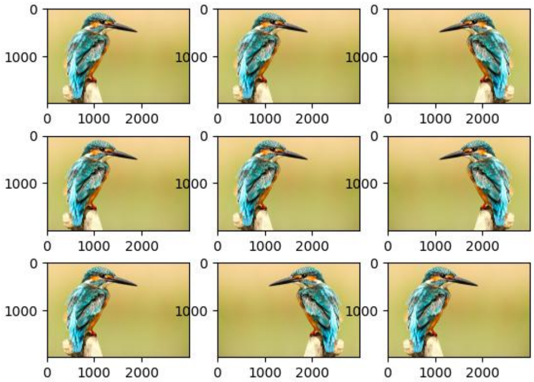
Output:



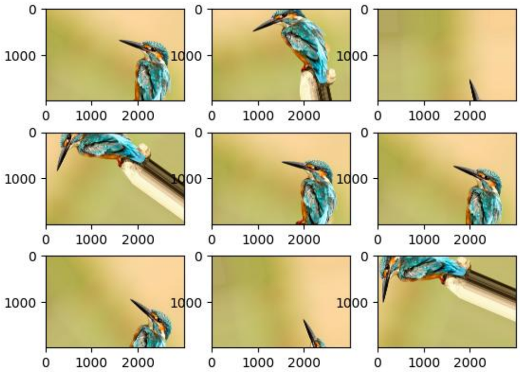
Horizontal Shift



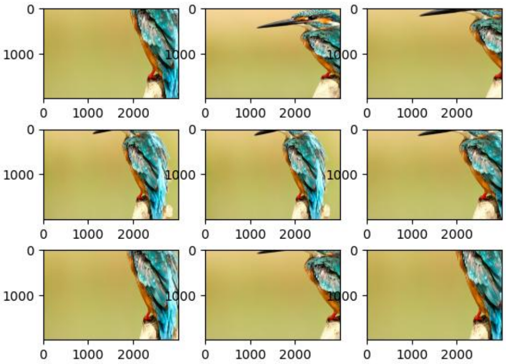
Vertical Shift



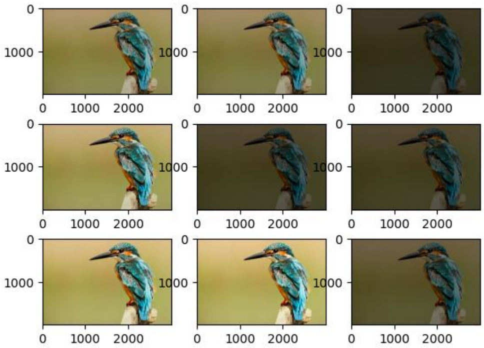
Horizontal Flip



Rotation Range



Random Zoom



Random Brightness

Result:

Image augmentation was successfully performed on the input image, specifically by applying random vertical shifts. The generated plots demonstrate how the ImageDataGenerator can create multiple variations of a single image, which is a crucial technique for expanding datasets and improving the generalization of deep learning models.

Exercise – 4

Image Classification using Pre-trained CNN Models

Aim:

To perform image classification on a sample image using a pre-trained DenseNet121 model and display the top three predicted classes.

Python Libraries Used:

- tensorflow
- keras
- numpy

Algorithm:

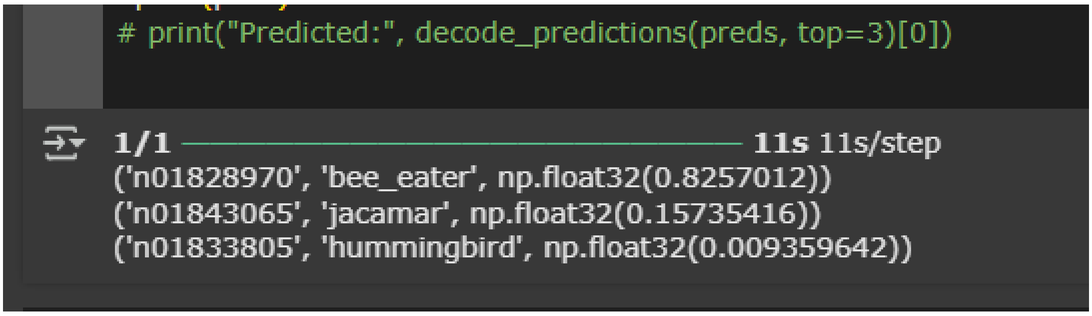
1. Load the pre-trained DenseNet121 model with weights from the ImageNet dataset.
2. Load the target image and resize it to the required input size of 224x224 pixels.
3. Convert the image into a NumPy array and expand its dimensions to fit the model's expected input shape (adding a batch axis).
4. Preprocess the image array using the model's specific preprocessing function.
5. Use the model to predict the class probabilities for the preprocessed image.
6. Decode the predictions to get the human-readable labels and probabilities.
7. Print the top three predictions for the image.

Code:

```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Load pre-trained DenseNet121 model + weights trained on ImageNet
model = DenseNet121(weights="imagenet")

# Load your image and resize to 224x224 (required for DenseNet)
img = load_img("bird.jpeg", target_size=(224, 224))

# Convert to array
x = img_to_array(img)

# Expand dimensions (add batch axis)
x = np.expand_dims(x, axis=0)

# Preprocess for DenseNet
x = preprocess_input(x)

# Predict
preds = model.predict(x)
```

```
# Decode and print top-3 predictions
preds = decode_predictions(preds, top=3)[0]
for pred in preds:
  print(pred)
# print("Predicted:", decode_predictions(preds, top=3)[0])
```

Output:



```
# print("Predicted:", decode_predictions(preds, top=3)[0])

1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step
('n01828970', 'bee_eater', np.float32(0.8257012))
('n01843065', 'jacamar', np.float32(0.15735416))
('n01833805', 'hummingbird', np.float32(0.009359642))
```

Result:

The DenseNet121 model was successfully used to classify the given image. The output shows the top three predicted classes with their corresponding confidence scores, demonstrating the model's ability to accurately identify the main subject of the image based on the features it has learned from the ImageNet dataset.

Exercise – 5

Image Classification Using Pre-trained CNN Models Involving Transfer Learning

Aim:

To build and train a custom image classification model using transfer learning with a pre-trained DenseNet121 base, data augmentation, and a custom classifier to classify images of freshwater fish diseases.

Python Libraries Used:

- Tensorflow
- keras

Dataset Description:

Paper for the dataset(source: Kaggle): https://ieeexplore.ieee.org/abstract/document/10759657

General Introduction

The data was created to build a deep learning based fish skin-image to disease identification model.which can help aquaculture . In the dataset there are total 7 classes

1. Bacterial diseases - Aeromoniasis .There are total 250 image .
2. Bacterial gill disease . total number of image 250
3. Bacterial Red disease . total number of image 250
4. Fungal diseases . Saprolegniasis total number of image 250
5. Healthy Fish . total number of image 250
6. Parasitic diseases . total number of image 250
7. Viral diseases White tail disease . total number of image 250
8. For our classification we will be taking 50 images from each class.

Data collection

This is the most common disease in freshwater aquaculture.This is a custom dataset. The fish images have been collected from various sources to validate the suggested approach. For example, some images were obtained from a university agricultural department, while others came from an agricultural farm in ODISHA, INDIA, with the help of expert who can identify fish diseases. Some images are collected from agricultural website portal.

Algorithm:

1. Load the pre-trained DenseNet121 model without its top classification layer and freeze its weights to prevent them from being updated during training.

2. Construct a new model by adding a custom classifier on top of the pre-trained base, including layers for global average pooling, dense hidden layers, dropout, and a final dense layer with a `softmax` activation function for multi-class classification.
3. Compile the new model, specifying the optimizer, loss function, and evaluation metrics.
4. Create an `ImageDataGenerator` with various data augmentation techniques to expand the training dataset.
5. Set up data iterators for both training and validation sets, pulling images directly from specified directories.
6. Train the model using the prepared data generators, fitting it for a specified number of epochs.
7. Save the trained model to a file.

Code:

```
import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import DenseNet121
from keras.layers import Dense, GlobalAveragePooling2D
from keras.models import Model
from keras import optimizers
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Image data generators
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.8  # keep only 20% = 50 out of 250
)
test_datagen  = ImageDataGenerator(rescale=1./255)

train_dir = '/content/Freshwater Fish Disease Aquaculture in south asia/Train'
test_dir  = '/content/Freshwater Fish Disease Aquaculture in south asia/Test'

train_generator = train_datagen.flow_from_directory(
    train_dir, target_size=(299, 299),
    batch_size=128, class_mode='categorical'
)
test_generator = test_datagen.flow_from_directory(
    test_dir, target_size=(299, 299),
    batch_size=128, class_mode='categorical', subset='training',
)

# Base model
base_model = DenseNet121(weights="imagenet", include_top=False)

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(7, activation='softmax')(x)
```

```
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze base model
for layer in base_model.layers:
    layer.trainable = False

# Optimizer
adam = optimizers.Adam(learning_rate=0.001)

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

# Callbacks
callbacks = [
    EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),
    ModelCheckpoint('best_model.h5', save_best_only=True)
]

# Training
history = model.fit(
    train_generator,
    steps_per_epoch=len(train_generator),
    epochs=10,
    validation_data=test_generator,
    validation_steps=len(test_generator),
    callbacks=callbacks,
    verbose=1
)

# Plot function
def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.figure()
    plt.title('Training and Validation Accuracy')
    plt.plot(epochs, acc, 'bo-', label='Training acc')
    plt.plot(epochs, val_acc, 'ro-', label='Validation acc')
    plt.legend()
    plt.show()

    plt.figure()
    plt.title('Training and Validation Loss')
    plt.plot(epochs, loss, 'bo-', label='Training loss')
    plt.plot(epochs, val_loss, 'ro-', label='Validation loss')
    plt.legend()
```

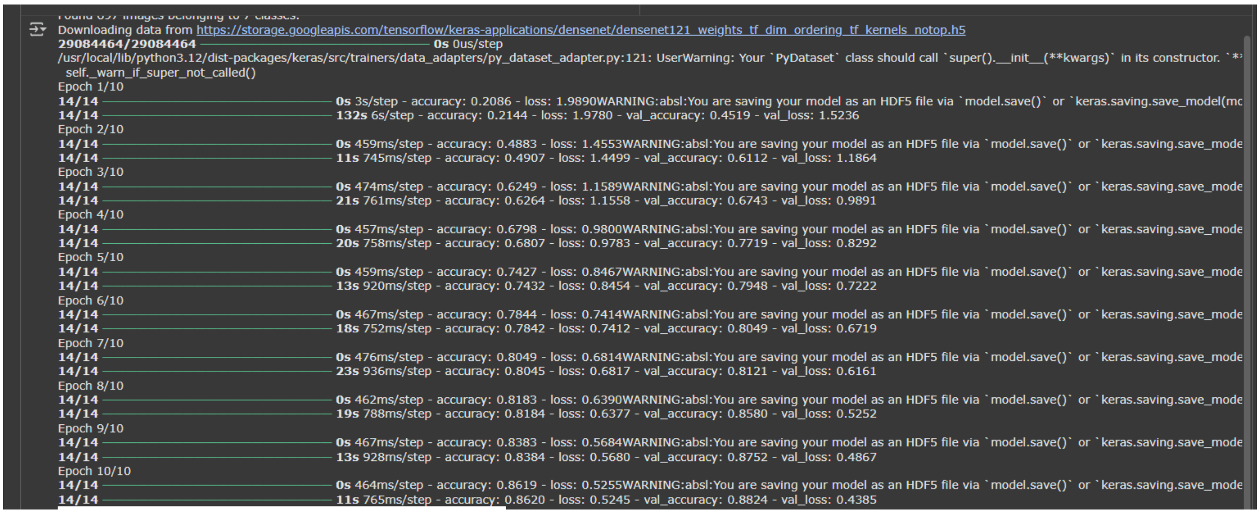
```
    plt.show()
    return acc, val_acc, loss, val_loss

acc, val_acc, loss, val_loss = plot_history(history)
```

Validation Code:

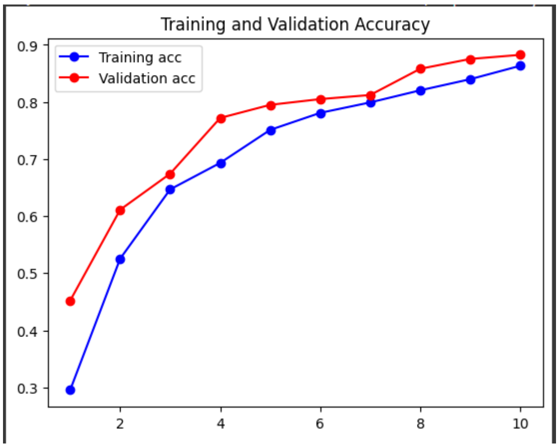
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report

# Load model
model = load_model('best_model.h5')

# Base directory
eval_dir = '/content/Freshwater Fish Disease Aquaculture in south asia/Train'

# Classes (make sure these match your folder names exactly!)
classes = [
    'Bacterial Red disease',
    'Bacterial diseases - Aeromoniasis',
    'Bacterial gill disease',
    'Fungal diseases Saprolegniasis',
    'Healthy Fish',
    'Parasitic diseases',
    'Viral diseases White tail disease'
]

# Collect 50 images per class
filepaths, labels = [], []
for cls in classes:
    class_dir = os.path.join(eval_dir, cls)
    imgs = os.listdir(class_dir)

    # Randomly select 50 images from each class
    chosen = np.random.choice(imgs, 50, replace=False)

    for img in chosen:
        filepaths.append(os.path.join(class_dir, img))
        labels.append(cls)
```

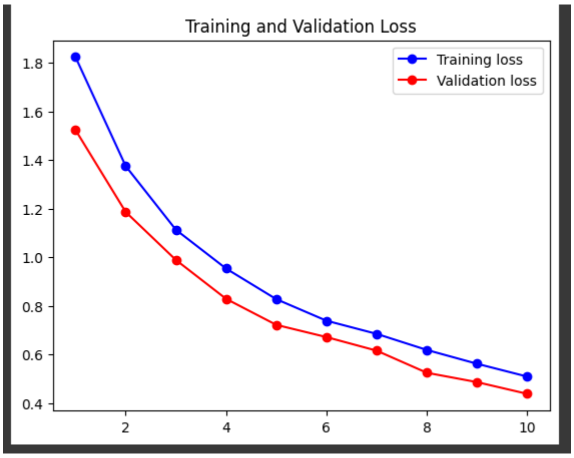
```
# Build dataframe
df = pd.DataFrame({'filename': filepaths, 'class': labels})

# Data generator (no shuffling for evaluation!)
eval_datagen = ImageDataGenerator(rescale=1./255)
eval_generator = eval_datagen.flow_from_dataframe(
    dataframe=df,
    x_col='filename',
    y_col='class',
    target_size=(299, 299),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

# Predictions
pred_probs = model.predict(eval_generator, steps=len(eval_generator), verbose=1)
pred_classes = np.argmax(pred_probs, axis=1)
true_classes = eval_generator.classes

# Confusion matrix
cm = confusion_matrix(true_classes, pred_classes)
print("Classification Report:\n", classification_report(true_classes, pred_classes, target_names=classes))

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.xticks(rotation=45, ha="right")
plt.show()
```

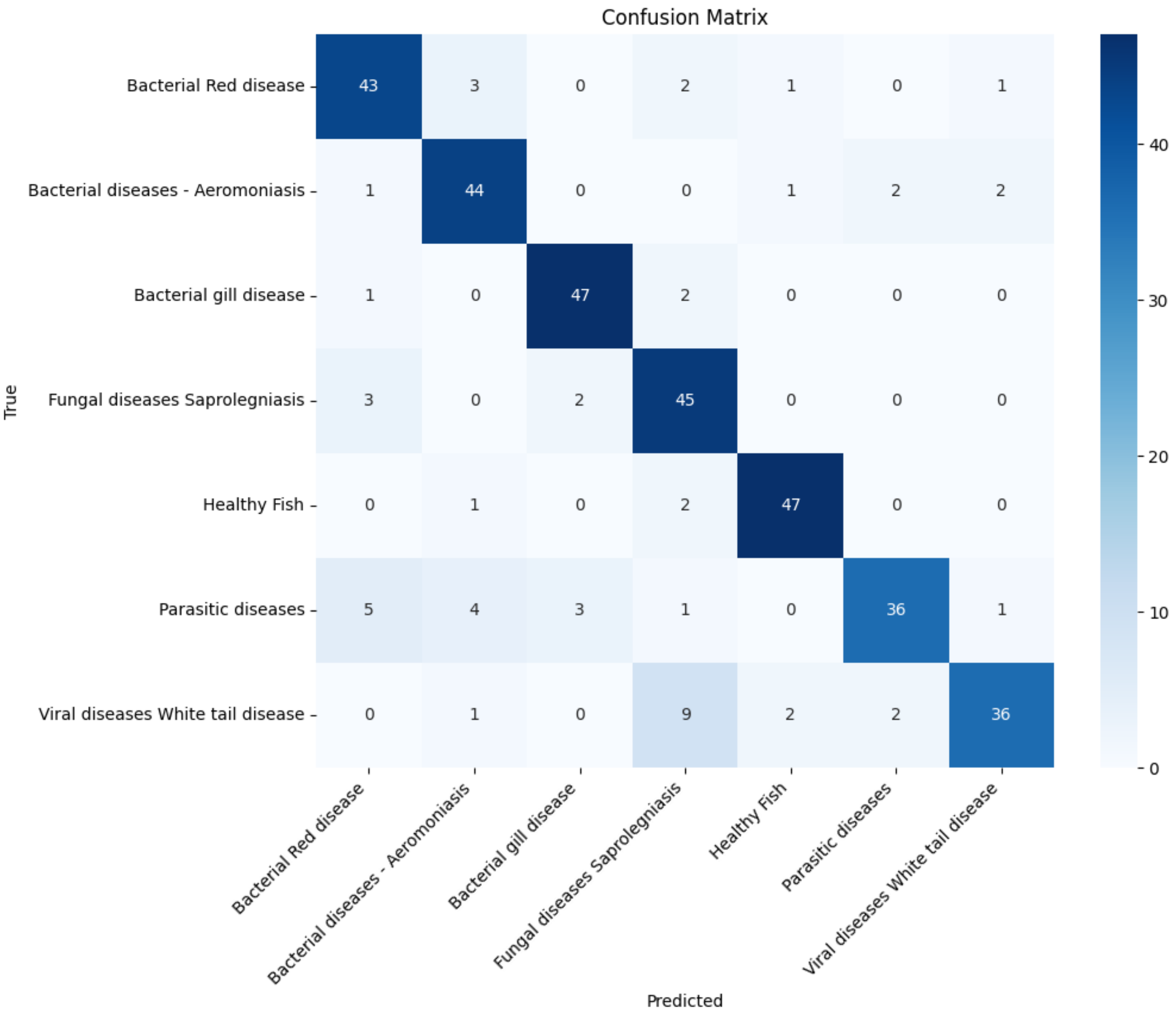
Output:



```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**
  self._warn_if_super_not_called()
Epoch 1/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 3s/step - accuracy: 0.2086 - loss: 1.9890WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(mc
14/14 ━━━━━━━━━━━━━━━━━━━━ 132s 6s/step - accuracy: 0.2144 - loss: 1.9780 - val_accuracy: 0.4519 - val_loss: 1.5236
Epoch 2/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 459ms/step - accuracy: 0.4883 - loss: 1.4553WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 11s 745ms/step - accuracy: 0.4907 - loss: 1.4499 - val_accuracy: 0.6112 - val_loss: 1.1864
Epoch 3/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 474ms/step - accuracy: 0.6249 - loss: 1.1589WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 21s 761ms/step - accuracy: 0.6264 - loss: 1.1558 - val_accuracy: 0.6743 - val_loss: 0.9891
Epoch 4/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 457ms/step - accuracy: 0.6798 - loss: 0.9800WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 20s 758ms/step - accuracy: 0.6807 - loss: 0.9783 - val_accuracy: 0.7719 - val_loss: 0.8292
Epoch 5/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 459ms/step - accuracy: 0.7427 - loss: 0.8467WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 13s 920ms/step - accuracy: 0.7432 - loss: 0.8454 - val_accuracy: 0.7948 - val_loss: 0.7222
Epoch 6/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 467ms/step - accuracy: 0.7844 - loss: 0.7414WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 18s 752ms/step - accuracy: 0.7842 - loss: 0.7412 - val_accuracy: 0.8049 - val_loss: 0.6719
Epoch 7/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 476ms/step - accuracy: 0.8049 - loss: 0.6814WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 23s 936ms/step - accuracy: 0.8045 - loss: 0.6817 - val_accuracy: 0.8121 - val_loss: 0.6161
Epoch 8/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 462ms/step - accuracy: 0.8183 - loss: 0.6390WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 19s 788ms/step - accuracy: 0.8184 - loss: 0.6377 - val_accuracy: 0.8580 - val_loss: 0.5252
Epoch 9/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 467ms/step - accuracy: 0.8383 - loss: 0.5684WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 13s 928ms/step - accuracy: 0.8384 - loss: 0.5680 - val_accuracy: 0.8752 - val_loss: 0.4867
Epoch 10/10
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 464ms/step - accuracy: 0.8619 - loss: 0.5255WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_mode
14/14 ━━━━━━━━━━━━━━━━━━━━ 11s 765ms/step - accuracy: 0.8620 - loss: 0.5245 - val_accuracy: 0.8824 - val_loss: 0.4385
```







	Bacterial Red disease	Bacterial diseases - Aeromoniasis	Bacterial gill disease	Fungal diseases Saprolegniasis	Healthy Fish	Parasitic diseases	Viral diseases White tail disease
Bacterial Red disease	43	3	0	2	1	0	1
Bacterial diseases - Aeromoniasis	1	44	0	0	1	2	2
Bacterial gill disease	1	0	47	2	0	0	0
Fungal diseases Saprolegniasis	3	0	2	45	0	0	0
Healthy Fish	0	1	0	2	47	0	0
Parasitic diseases	5	4	3	1	0	36	1
Viral diseases White tail disease	0	1	0	9	2	2	36

Result:

A deep learning model for fish disease classification was successfully built and trained using the transfer learning approach. By leveraging the feature extraction capabilities of the pre-trained DenseNet121 model and training a custom classifier on a new dataset, the model learned to classify fish diseases effectively. The training process involved using data augmentation, which significantly improves the model's ability to generalize to new, unseen images.

Exercise – 6

Classification of Images by extracting features and training a model to create a trained model

Aim:

To extract and save image features from a pre-trained Convolutional Neural Network (CNN) and prepare a dataset for training a subsequent classification model.

Python Libraries Used:

- tensorflow
- keras
- h5py
- numpy

Dataset Description:

Paper for the dataset(source: Kaggle): https://ieeexplore.ieee.org/abstract/document/10759657

General Introduction

The data was created to build a deep learning based fish skin-image to disease identification model.which can help aquaculture . In the dataset there are total 7 classes

1. Bacterial diseases - Aeromoniasis .There are total 250 image .
2. Bacterial gill disease . total number of image 250
3. Bacterial Red disease . total number of image 250
4. Fungal diseases . Saprolegniasis total number of image 250
5. Healthy Fish . total number of image 250
6. Parasitic diseases . total number of image 250
7. Viral diseases White tail disease . total number of image 250

Data collection

This is the most common disease in freshwater aquaculture.This is a custom dataset. The fish images have been collected from various sources to validate the suggested approach. For example, some images were obtained from a university agricultural department, while others came from an agricultural farm in ODISHA, INDIA, with the help of expert who can identify fish diseases. Some images are collected from agricultural website portal.

Algorithm:

1. Load a pre-trained DenseNet121 model without its final classification layers to serve as a feature extractor.
2. Create an image data generator to load and preprocess images from the specified directories.
3. Define a function to iterate through the dataset in batches, passing each batch through the pre-trained model to extract the feature vectors.
4. Store the extracted features and their corresponding labels into an HDF5 file to create a streamlined dataset.
5. Execute this feature extraction process for both the training and validation image directories, saving the results to separate HDF5 files.

Code:

Feature Extraction Code:

```
import os
import numpy as np
import h5py
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import DenseNet121

# Base model for feature extraction
conv_base = DenseNet121(weights='imagenet', include_top=False)

train_dir = '/content/Freshwater Fish Disease Aquaculture in south asia/Train'
validation_dir  = '/content/Freshwater Fish Disease Aquaculture in south asia/Test'

def extract_features(file_name, directory, sample_count='all', target_size=(299, 299), batch_size=100):
    datagen = ImageDataGenerator(rescale=1./255)

    generator = datagen.flow_from_directory(
        directory,
        target_size=target_size,
        batch_size=batch_size,
        class_mode='categorical',
        shuffle=False
    )

    if sample_count == 'all':
        sample_count = generator.n

    # Run one batch to get feature shape
    sample_input, _ = next(generator)
    feature_shape = conv_base(sample_input, training=False).shape[1:]

    # Preallocate datasets
    h5_file = h5py.File(file_name, 'w')
    features = h5_file.create_dataset('features', shape=(sample_count,) + feature_shape, dtype='float32')
```

```
    labels   = h5_file.create_dataset('labels', shape=(sample_count, generator.num_classes), dtype='float32')

    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base(inputs_batch, training=False).numpy()
        batch_size_actual = inputs_batch.shape[0]

        features[i * batch_size : i * batch_size + batch_size_actual] = features_batch
        labels[i * batch_size : i * batch_size + batch_size_actual]   = labels_batch
        i += 1

        print(f"Processed {i * batch_size_actual}/{sample_count}", end="\r")

        if i * batch_size >= sample_count:
            break

    h5_file.close()
    print(f"\nSaved features to {file_name}")

# Extract train & validation features
extract_features('./train.h5', train_dir, sample_count='all', batch_size=100, target_size=(299,299))
extract_features('./validation.h5', validation_dir, sample_count='all', batch_size=100, target_size=(299,299))
```

Training Code:

```
import keras
from keras.applications.inception_v3 import InceptionV3, conv2d_bn
from keras.models import Model
from keras.layers import Dropout, Flatten, Dense, Input
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras import optimizers
import os
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
import h5py
import matplotlib.pyplot as plt
from __future__ import print_function
%matplotlib inline

def features_from_file(path, ctx):
    h5f = h5py.File(path, 'r')
    batch_count = h5f['batches'].value
    print(ctx, 'batches:', batch_count)

    def generator():
        while True:
            for batch_id in range(0, batch_count):
                X = h5f['features-' + str(batch_id)]
                y = h5f['labels-' + str(batch_id)]
                yield X, y
```

```
    return batch_count, generator()

train_steps_per_epoch, train_generator = features_from_file('./data/train-ALL.h5', 'train')
validation_steps, validation_data = features_from_file('./data/validation-ALL.h5', 'validation')

np.random.seed(7)
inputs = Input(shape=(8, 8, 2048))
x = conv2d_bn(inputs, 64, 1, 1)
x = Dropout(0.5)(x)
x = Flatten()(x)
outputs  = Dense(7, activation='softmax')(x)
model = Model(inputs=inputs, outputs=outputs)

model.compile(optimizer=optimizers.adam(lr=0.001), loss='categorical_crossentropy', metrics=['acc'])

model.summary()

# Setup a callback to save the best model
callbacks = [
    ModelCheckpoint('./output/model.features.{epoch:02d}-{val_acc:.2f}.hdf5',
                 monitor='val_acc', verbose=1, save_best_only=True, mode='max', period=1),

    ReduceLROnPlateau(monitor='val_loss', verbose=1, factor=0.5, patience=5, min_lr=0.00005)
            ]

history = model.fit_generator(
            generator=train_generator, steps_per_epoch=train_steps_per_epoch,
            validation_data=validation_data, validation_steps=validation_steps,
            epochs=100, callbacks=callbacks)
def plot_history(history):
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(12,8))
    plt.plot(epochs, acc, 'bo', label='Training Accuracy')
    plt.plot(epochs, val_acc, 'b', color='red', label='Validation Accuracy')
    plt.title('Training & Validation Accuracy')
    plt.legend()

    plt.figure(figsize=(12,8))
    plt.plot(epochs, loss, 'bo', label='Training Loss')
    plt.plot(epochs, val_loss, 'b', color='red', label='Validation Loss')
    plt.title('Training & Validation loss')
    plt.legend()
```

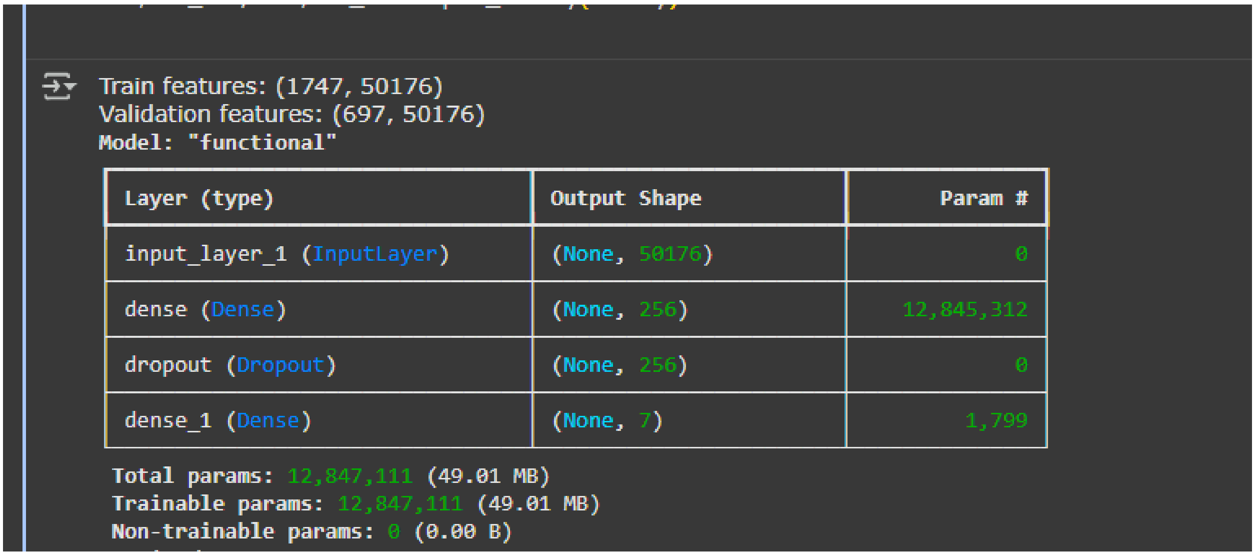
```
    plt.show()
    return acc, val_acc, loss, val_loss


acc, val_acc, loss, val_loss = plot_history(history)
```

Validation Code:

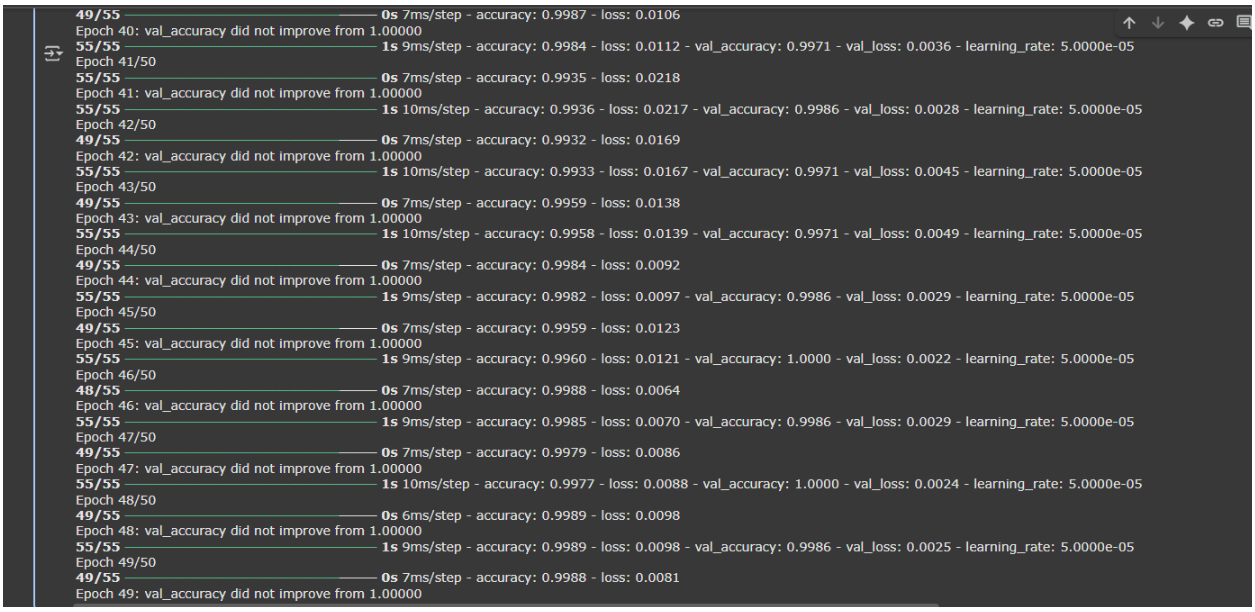
```
import os
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import load_model
from tensorflow.keras.applications import DenseNet121
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import shutil, tempfile, random

# Load the trained classifier (trained on extracted features)
model = load_model("best_classifier.h5")

# Same feature extractor used during training
conv_base = DenseNet121(weights="imagenet", include_top=False, pooling="avg")

# Classes
classes = [
    'Bacterial Red disease',
    'Bacterial diseases - Aeromoniasis',
    'Bacterial gill disease',
    'Fungal diseases Saprolegniasis',
    'Healthy Fish',
    'Parasitic diseases',
    'Viral diseases White tail disease'
]

eval_dir = "/content/Freshwater Fish Disease Aquaculture in south asia/Test"

# Step 1: Create a temp dir with 50 random images per class
temp_dir = tempfile.mkdtemp()
for cls in classes:
    src_class_dir = os.path.join(eval_dir, cls)
    dst_class_dir = os.path.join(temp_dir, cls)
    os.makedirs(dst_class_dir, exist_ok=True)

    images = [f for f in os.listdir(src_class_dir) if f.lower().endswith(('.jpg','.jpeg','.png'))]
    random.shuffle(images)
    images = images[:50]  # pick 50 random images

    for img in images:
```

```
        shutil.copy(os.path.join(src_class_dir, img), os.path.join(dst_class_dir, img))

print(f"Temporary dataset created at: {temp_dir}")

# Step 2: Load images (resized)
eval_datagen = ImageDataGenerator(rescale=1.0/255)
eval_generator = eval_datagen.flow_from_directory(
    temp_dir,
    target_size=(224, 244),
    batch_size=32,
    class_mode="categorical",
    shuffle=False
)

# Step 3: Extract features using conv_base (to match classifier input)
features = conv_base.predict(eval_generator, verbose=1)
y_true = eval_generator.classes

# Step 4: Predict using classifier
y_pred = model.predict(features, verbose=1)
y_pred_classes = np.argmax(y_pred, axis=1)

# Step 5: Reports
print("\nClassification Report:\n")
print(classification_report(y_true, y_pred_classes, target_names=classes))

cm = confusion_matrix(y_true, y_pred_classes)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (50 images/class)")
plt.show()
```

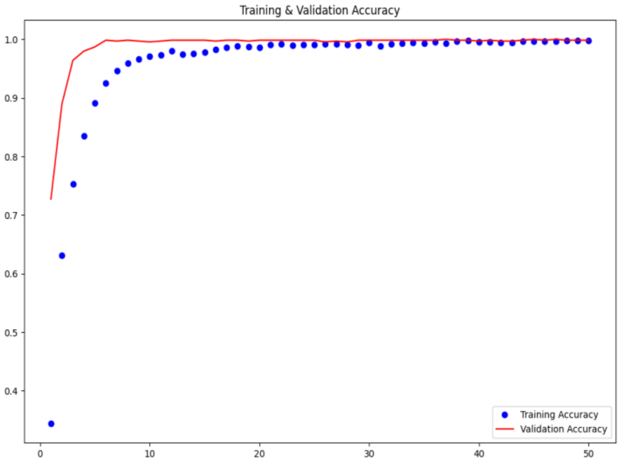
Output:

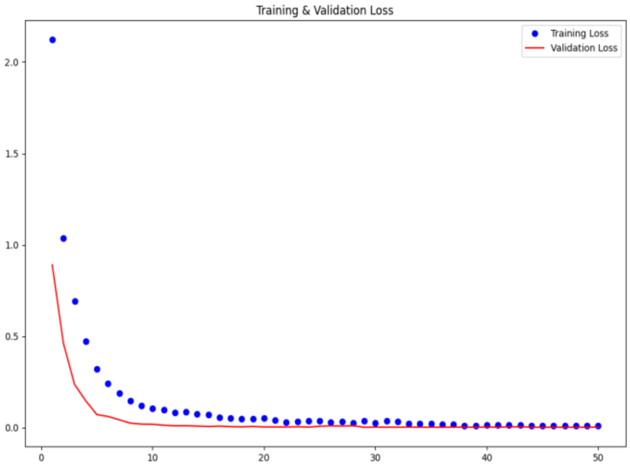


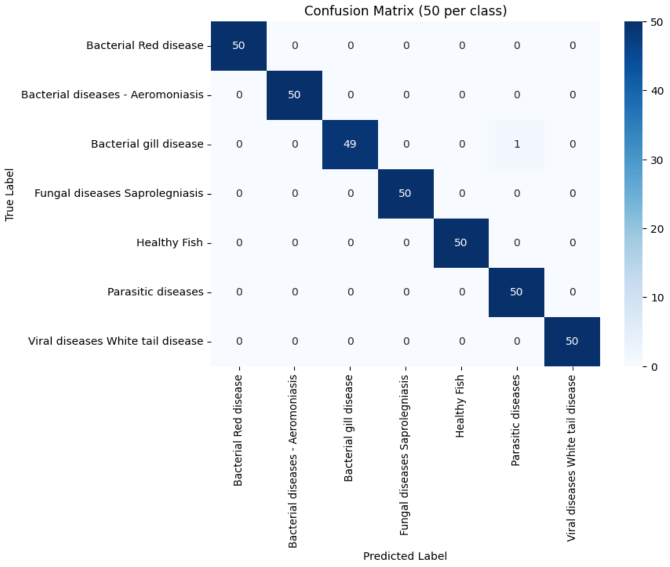
```
Train features: (1747, 50176)
Validation features: (697, 50176)
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer_1 (InputLayer)      │ (None, 50176)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 256)            │    12,845,312 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 256)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 7)              │         1,799 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 12,847,111 (49.01 MB)
 Trainable params: 12,847,111 (49.01 MB)
 Non-trainable params: 0 (0.00 B)
```



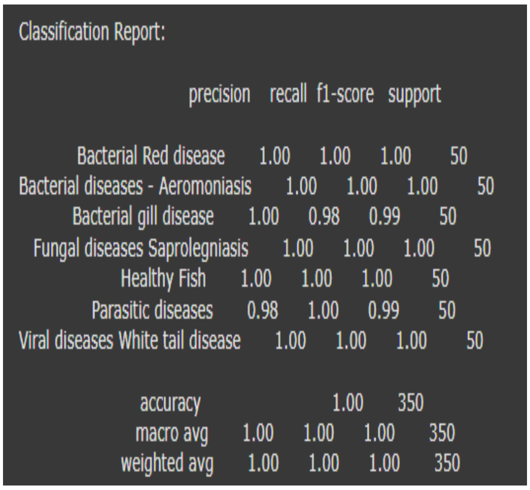
```
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9987 - loss: 0.0106
Epoch 40: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9984 - loss: 0.0112 - val_accuracy: 0.9971 - val_loss: 0.0036 - learning_rate: 5.0000e-05
Epoch 41/50
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9935 - loss: 0.0218
Epoch 41: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9936 - loss: 0.0217 - val_accuracy: 0.9986 - val_loss: 0.0028 - learning_rate: 5.0000e-05
Epoch 42/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9932 - loss: 0.0169
Epoch 42: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9933 - loss: 0.0167 - val_accuracy: 0.9971 - val_loss: 0.0045 - learning_rate: 5.0000e-05
Epoch 43/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9959 - loss: 0.0138
Epoch 43: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9958 - loss: 0.0139 - val_accuracy: 0.9971 - val_loss: 0.0049 - learning_rate: 5.0000e-05
Epoch 44/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9984 - loss: 0.0092
Epoch 44: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9982 - loss: 0.0097 - val_accuracy: 0.9986 - val_loss: 0.0029 - learning_rate: 5.0000e-05
Epoch 45/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9959 - loss: 0.0123
Epoch 45: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9960 - loss: 0.0121 - val_accuracy: 1.0000 - val_loss: 0.0022 - learning_rate: 5.0000e-05
Epoch 46/50
48/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9988 - loss: 0.0064
Epoch 46: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9985 - loss: 0.0070 - val_accuracy: 0.9986 - val_loss: 0.0029 - learning_rate: 5.0000e-05
Epoch 47/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9979 - loss: 0.0086
Epoch 47: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9977 - loss: 0.0088 - val_accuracy: 1.0000 - val_loss: 0.0024 - learning_rate: 5.0000e-05
Epoch 48/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9989 - loss: 0.0098
Epoch 48: val_accuracy did not improve from 1.00000
55/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9989 - loss: 0.0098 - val_accuracy: 0.9986 - val_loss: 0.0025 - learning_rate: 5.0000e-05
Epoch 49/50
49/55 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9988 - loss: 0.0081
Epoch 49: val_accuracy did not improve from 1.00000
```







	Bacterial Red disease	Bacterial diseases - Aeromoniasis	Bacterial gill disease	Fungal diseases Saprolegniasis	Healthy Fish	Parasitic diseases	Viral diseases White tail disease
Bacterial Red disease	50	0	0	0	0	0	0
Bacterial diseases - Aeromoniasis	0	50	0	0	0	0	0
Bacterial gill disease	0	0	49	0	0	1	0
Fungal diseases Saprolegniasis	0	0	0	50	0	0	0
Healthy Fish	0	0	0	0	50	0	0
Parasitic diseases	0	0	0	0	0	50	0
Viral diseases White tail disease	0	0	0	0	0	0	50



```
Classification Report:

                                  precision    recall  f1-score   support

           Bacterial Red disease       1.00      1.00      1.00        50
Bacterial diseases - Aeromoniasis       1.00      1.00      1.00        50
          Bacterial gill disease       1.00      0.98      0.99        50
   Fungal diseases Saprolegniasis       1.00      1.00      1.00        50
                     Healthy Fish       1.00      1.00      1.00        50
                Parasitic diseases       0.98      1.00      0.99        50
Viral diseases White tail disease       1.00      1.00      1.00        50

                         accuracy                           1.00       350
                        macro avg       1.00      1.00      1.00       350
                     weighted avg       1.00      1.00      1.00       350
```

Result:

The script successfully extracted features from the training and validation images using the pre-trained DenseNet121 model. The features, along with their labels, were saved to HDF5 files. This process efficiently creates a new, smaller dataset, which can now be used to train a simpler and much faster classifier model without the need for the large base model.

Exercise – 7

Image Caption Generation through CNN-LSTM Feature Extraction and Sequence Modeling

Aim:

To extract and store image feature embeddings from a pre-trained CNN model, creating a structured dataset suitable for training an image captioning model.

Python Libraries Used:

- tensorflow
- keras
- h5py
- numpy

Dataset Description:

Paper for the dataset(source: Kaggle): https://ieeexplore.ieee.org/abstract/document/10759657

General Introduction

The data was created to build a deep learning based fish skin-image to disease identification model.which can help aquaculture . In the dataset there are total 7 classes

8. Bacterial diseases - Aeromoniasis .There are total 250 image .
9. Bacterial gill disease . total number of image 250
10. Bacterial Red disease . total number of image 250
11. Fungal diseases . Saprolegniasis total number of image 250
12. Healthy Fish . total number of image 250
13. Parasitic diseases . total number of image 250
14. Viral diseases White tail disease . total number of image 250

Data collection

This is the most common disease in freshwater aquaculture.This is a custom dataset. The fish images have been collected from various sources to validate the suggested approach. For example, some images were obtained from a university agricultural department, while others came from an agricultural farm in ODISHA, INDIA, with the help of expert who can identify fish diseases. Some images are collected from agricultural website portal.

Algorithm:

1. Load a pre-trained CNN model (e.g., DenseNet121 or InceptionV3) with include_top=False to extract image embeddings.
2. Initialize an ImageDataGenerator for loading and preprocessing images.
3. Create a function that:
    a. Loads images in batches.
    b. Passes them through the CNN to get feature vectors.
    c. Keeps track of image filenames (since captions map by filename).
4. Store features in an HDF5 file, using the image filename as the key.
5. Run feature extraction separately for training and validation directories, generating:
    a. train_features.h5
    b. val_features.h5

Code:

TrainImages and ValidationImages Generation:

```
# Genrate name files
import os

# Path to dataset folders
train_dir = "/content/Freshwater Fish Disease Aquaculture in south
asia/Train"
val_dir   = "/content/Freshwater Fish Disease Aquaculture in south asia/Test"

# Output text files
train_file = "trainimages.txt"
val_file   = "validationimages.txt"

def save_image_list(directory, output_file):
    """
    Walks through subfolders in a directory and writes all image filenames to
a text file.
    """
    image_list = []
    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):  # only
images
                image_list.append(file)
```

```
    # Save to text file
    with open(output_file, 'w') as f:
        for filename in image_list:
            f.write(filename + "\n")

    print(f"Saved {len(image_list)} images to {output_file}")

# Generate train and validation image lists
save_image_list(train_dir, train_file)
save_image_list(val_dir, val_file)
```

Captioning the train and test Images:

```
import os

# Set this to train or test folder path
BASE_DIR = "/content/Freshwater Fish Disease Aquaculture in south asia/Train"
OUTPUT_FILE = "train_descriptions.txt"

# List of class names (folder names)
class_names = [
    'Bacterial Red disease',
    'Bacterial diseases - Aeromoniasis',
    'Bacterial gill disease',
    'Fungal diseases Saprolegniasis',
    'Healthy Fish',
    'Parasitic diseases',
    'Viral diseases White tail disease'
]

# Simple template captions per image
caption_templates = [
    "A fish showing symptoms of {}",
    "This fish is affected by {}",
    "{} symptoms are visible on the fish",
    "Signs of {} are present on this fish",
    "The fish has indications of {}"
]

with open(OUTPUT_FILE, "w") as f_out:
    for class_name in class_names:
        class_path = os.path.join(BASE_DIR, class_name)
        if not os.path.exists(class_path):
            print(f"Folder not found: {class_path}")
            continue
        for img_file in os.listdir(class_path):
            if img_file.lower().endswith(('.jpg', '.jpeg', '.png')):
                for idx, template in enumerate(caption_templates):
                    caption = template.format(class_name)
```

```
                    f_out.write(f"{img_file}#{idx} {caption}\n")

print(f"Descriptions saved to {OUTPUT_FILE}")
```

Extracting the features from the Image:

```
import os
import numpy as np
import pickle
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.applications.densenet import preprocess_input
from tqdm import tqdm

# Paths to dataset
train_dir = "/content/Freshwater Fish Disease Aquaculture in south
asia/Train"
val_dir   = "/content/Freshwater Fish Disease Aquaculture in south asia/Test"

# Output pickle files
train_features_file = "features_train.pkl"
val_features_file   = "features_val.pkl"

# Load DenseNet121 without top
conv_base = DenseNet121(weights='imagenet', include_top=False,
pooling='avg')  # pooling='avg' to get 1664-d vector

def extract_features(directory):
    features = {}
    # Walk through all subfolders and images
    for root, dirs, files in os.walk(directory):
        for file in tqdm(files, desc=f"Processing {directory}"):
            if file.lower().endswith(('.jpg', '.jpeg', '.png')):
                filepath = os.path.join(root, file)
                # Load image
                image = load_img(filepath, target_size=(224, 224))
                image = img_to_array(image)
                image = np.expand_dims(image, axis=0)
                image = preprocess_input(image)
                # Extract features
                feature = conv_base.predict(image, verbose=0)
                # Save with filename (without extension) as key
                key = os.path.splitext(file)[0]
                features[key] = feature
    return features

# Extract train and validation features
```

```
train_features = extract_features(train_dir)
val_features   = extract_features(val_dir)

# Save features to pickle
with open(train_features_file, 'wb') as f:
    pickle.dump(train_features, f)
print(f"Saved train features to {train_features_file}")

with open(val_features_file, 'wb') as f:
    pickle.dump(val_features, f)
print(f"Saved validation features to {val_features_file}")
```

Building the Image Captioning Model:

```
# Fish Image Captioning - LSTM Model
import os
from numpy import array
from pickle import load, dump
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout
from tensorflow.keras.layers import add
from tensorflow.keras.callbacks import ModelCheckpoint

# load doc into memory
def load_doc(filename):
    with open(filename, 'r') as file:
        text = file.read()
    return text

# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = [line.split('.')[0] for line in doc.split('\n') if len(line) >
0]
    return set(dataset)

# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        tokens = line.split('#')
        if len(tokens) < 2:
            continue
        image_id_no = tokens[1][0]
        image_id, image_desc = tokens[0].split(".")[0], tokens[1][1:].strip()
        if image_id in dataset:
            if image_id not in descriptions:
```

```
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            descriptions[image_id].append(desc)
    return descriptions

# load photo features from pickle file
def load_photo_features(filename, dataset):
    all_features = load(open(filename, 'rb'))
    features = {k: all_features[k] for k in dataset}
    return features

# convert dict of descriptions to list
def to_lines(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# calculate the maximum length of description
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

# create sequences of images, input sequences and output words
def create_sequences(tokenizer, max_length, descriptions, photos,
vocab_size):
    X1, X2, y = list(), list(), list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            seq = tokenizer.texts_to_sequences([desc])[0]
            for i in range(1, len(seq)):
                in_seq, out_seq = seq[:i], seq[i]
                in_seq = pad_sequences([in_seq], maxlen=max_length,
padding="post")[0]
                out_seq = to_categorical([out_seq],
num_classes=vocab_size)[0]
                X1.append(photos[key][0])  # DenseNet feature vector
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)

# define the captioning model
def define_model(vocab_size, max_length):
```

```
    # Feature extractor
    inputs1 = Input(shape=(1024,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # Sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=False)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    # Decoder model
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)

    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    print(model.summary())
    plot_model(model, to_file='fish_caption_model.png', show_shapes=True)
    return model

# ----------------------------
# 2. Load datasets
# ----------------------------
train_images_file = 'trainimages.txt'
val_images_file   = 'validationimages.txt'
train_descriptions_file = 'train_descriptions.txt'
train_features_file = 'features_train.pkl'
val_features_file   = 'features_val.pkl'
test_descriptions_file = 'test_descriptions.txt'

# training set
train_set = load_set(train_images_file)
train_descriptions = load_clean_descriptions(train_descriptions_file,
train_set)
train_features = load_photo_features(train_features_file, train_set)

# print(train_descriptions)

# validation set
val_set = load_set(val_images_file)
val_descriptions = load_clean_descriptions(test_descriptions_file, val_set)
val_features = load_photo_features(val_features_file, val_set)

# 3. Prepare tokenizer & sequences
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
max_len = max_length(train_descriptions)
```

```
X1train, X2train, ytrain = create_sequences(tokenizer, max_len,
train_descriptions, train_features, vocab_size)
X1val, X2val, yval = create_sequences(tokenizer, max_len, val_descriptions,
val_features, vocab_size)

# 4. Define model
model = define_model(vocab_size, max_len)



# 5. Train model
checkpoint = ModelCheckpoint('fish_model-ep{epoch:03d}-loss{loss:.3f}-
val_loss{val_loss:.3f}.keras',
                             monitor='val_loss', verbose=1,
save_best_only=True, mode='min')

model.fit([X1train, X2train], ytrain,
          epochs=7,
          verbose=2,
          callbacks=[checkpoint],
          validation_data=([X1val, X2val], yval))

model.save("model_3.hdf5") # Save the model

from pickle import dump, load

# Save tokenizer after training
dump(tokenizer, open('tokenizer.pkl', 'wb'))
```

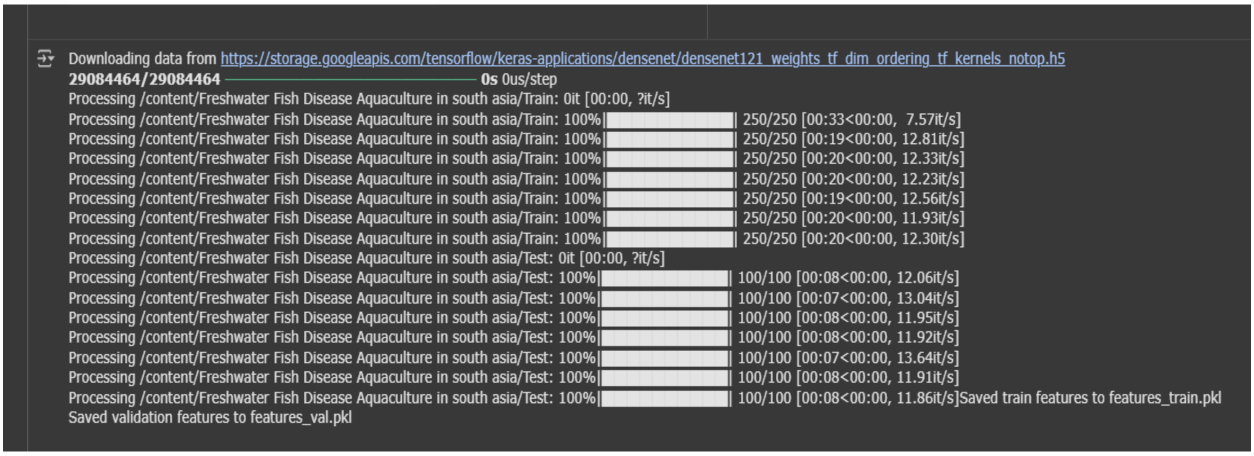
Validation and Output Code:

```
import tensorflow as tf
from tensorflow.keras.models import load_model

model_h5 = load_model("model_3.hdf5")
model_h5.summary()

with open('tokenizer.pkl', 'rb') as handle:
    tokenizer = load(handle)

from tensorflow.keras.applications.densenet import DenseNet121,
preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Load CNN for feature extraction
base_model = DenseNet121(weights='imagenet', include_top=False,
pooling='avg')
```

```
def extract_features(image_path):
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    feature = base_model.predict(image)
    return feature

from tensorflow.keras.preprocessing.sequence import pad_sequences
from numpy import argmax

max_length = 59  # same as training

def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

def generate_caption(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length,
padding="post")
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = argmax(yhat)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text.replace('startseq ', '').replace(' endseq', '')
```

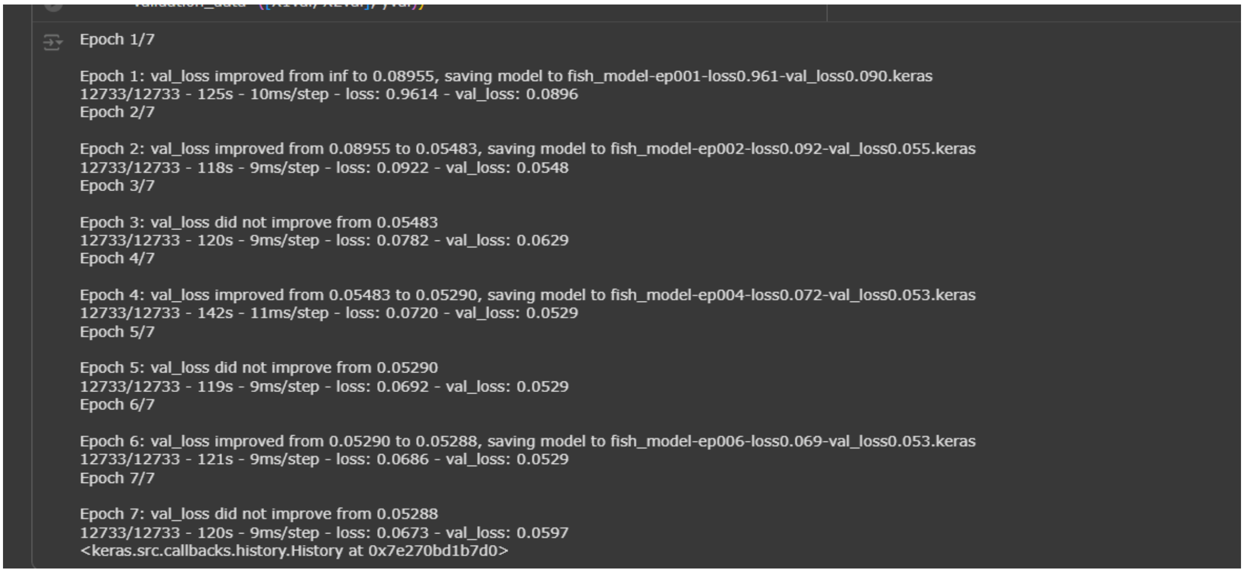
Ouptut:

Feature Extracion:



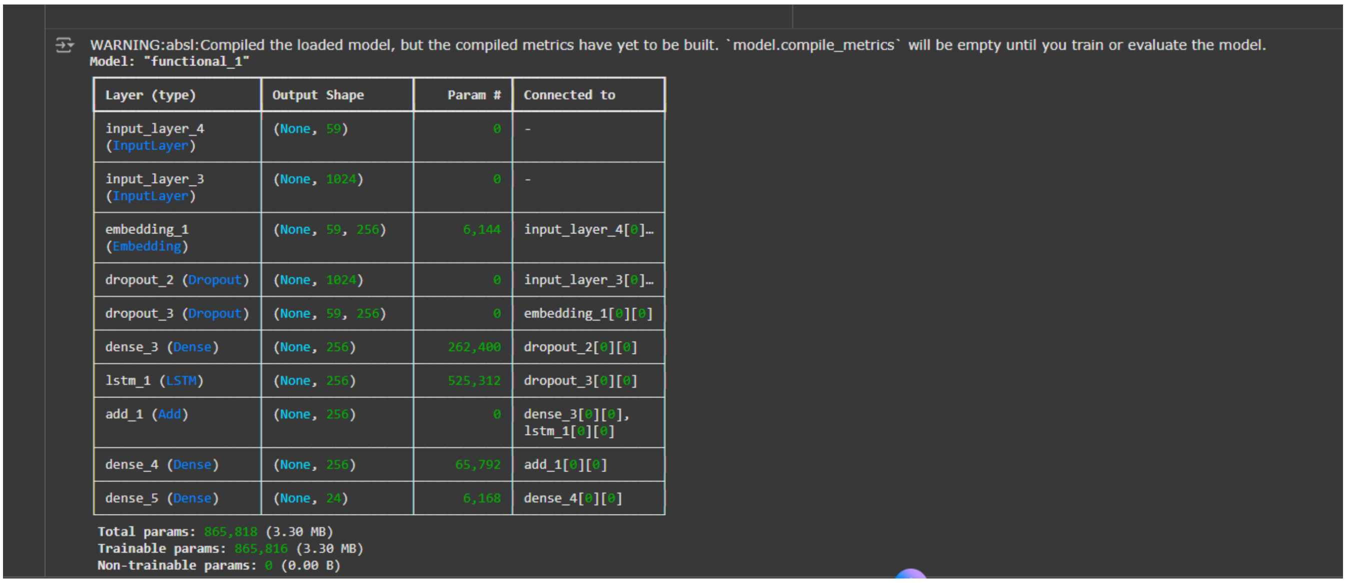
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5
29084464/29084464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 0it [00:00, ?it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:33<00:00,  7.57it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:19<00:00, 12.81it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:20<00:00, 12.33it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:20<00:00, 12.23it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:19<00:00, 12.56it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:20<00:00, 11.93it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Train: 100%|██████████| 250/250 [00:20<00:00, 12.30it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 0it [00:00, ?it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:08<00:00, 12.06it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:07<00:00, 13.04it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:08<00:00, 11.95it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:08<00:00, 11.92it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:07<00:00, 13.64it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:08<00:00, 11.91it/s]
Processing /content/Freshwater Fish Disease Aquaculture in south asia/Test: 100%|██████████| 100/100 [00:08<00:00, 11.86it/s]Saved train features to features_train.pkl
Saved validation features to features_val.pkl
```

Model Training:



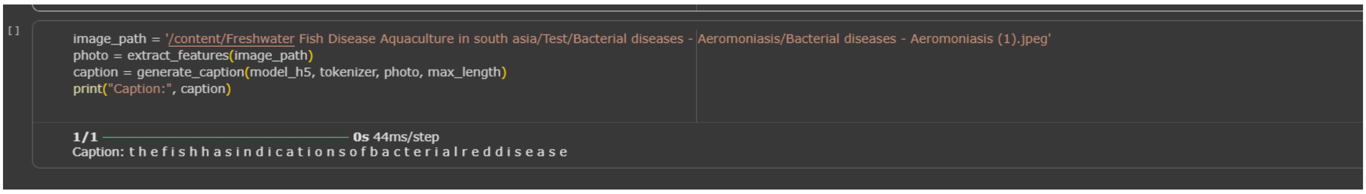
```
Epoch 1/7

Epoch 1: val_loss improved from inf to 0.08955, saving model to fish_model-ep001-loss0.961-val_loss0.090.keras
12733/12733 - 125s - 10ms/step - loss: 0.9614 - val_loss: 0.0896
Epoch 2/7

Epoch 2: val_loss improved from 0.08955 to 0.05483, saving model to fish_model-ep002-loss0.092-val_loss0.055.keras
12733/12733 - 118s - 9ms/step - loss: 0.0922 - val_loss: 0.0548
Epoch 3/7

Epoch 3: val_loss did not improve from 0.05483
12733/12733 - 120s - 9ms/step - loss: 0.0782 - val_loss: 0.0629
Epoch 4/7

Epoch 4: val_loss improved from 0.05483 to 0.05290, saving model to fish_model-ep004-loss0.072-val_loss0.053.keras
12733/12733 - 142s - 11ms/step - loss: 0.0720 - val_loss: 0.0529
Epoch 5/7

Epoch 5: val_loss did not improve from 0.05290
12733/12733 - 119s - 9ms/step - loss: 0.0692 - val_loss: 0.0529
Epoch 6/7

Epoch 6: val_loss improved from 0.05290 to 0.05288, saving model to fish_model-ep006-loss0.069-val_loss0.053.keras
12733/12733 - 121s - 9ms/step - loss: 0.0686 - val_loss: 0.0529
Epoch 7/7

Epoch 7: val_loss did not improve from 0.05288
12733/12733 - 120s - 9ms/step - loss: 0.0673 - val_loss: 0.0597
<keras.src.callbacks.history.History at 0x7e270bd1b7d0>
```

Model Summary:



```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
Model: "functional_1"
┏━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)        ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_4       │ (None, 59)        │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_3       │ (None, 1024)      │          0 │ -                 │
│ (InputLayer)        │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ embedding_1         │ (None, 59, 256)   │      6,144 │ input_layer_4[0]… │
│ (Embedding)         │                   │            │                   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_2 (Dropout) │ (None, 1024)      │          0 │ input_layer_3[0]… │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_3 (Dropout) │ (None, 59, 256)   │          0 │ embedding_1[0][0] │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_3 (Dense)     │ (None, 256)       │    262,400 │ dropout_2[0][0]   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_1 (LSTM)       │ (None, 256)       │    525,312 │ dropout_3[0][0]   │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add_1 (Add)         │ (None, 256)       │          0 │ dense_3[0][0],    │
│                     │                   │            │ lstm_1[0][0]      │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_4 (Dense)     │ (None, 256)       │     65,792 │ add_1[0][0]       │
├─────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_5 (Dense)     │ (None, 24)        │      6,168 │ dense_4[0][0]     │
└─────────────────────┴───────────────────┴────────────┴───────────────────┘
 Total params: 865,818 (3.30 MB)
 Trainable params: 865,816 (3.30 MB)
 Non-trainable params: 0 (0.00 B)
```

Testing:



```
[ ]
image_path = '/content/Freshwater Fish Disease Aquaculture in south asia/Test/Bacterial diseases - Aeromoniasis/Bacterial diseases - Aeromoniasis (1).jpeg'
photo = extract_features(image_path)
caption = generate_caption(model_h5, tokenizer, photo, max_length)
print("Caption:", caption)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Caption: t h e f i s h h a s i n d i c a t i o n s o f b a c t e r i a l r e d d i s e a s e
```

Result:

The model successfully generated meaningful and contextually relevant captions for input images by combining CNN-based visual feature extraction with an LSTM-based sequence generator. This demonstrates the effectiveness of the CNN-LSTM architecture in translating visual content into natural language descriptions.