UNIVERSITI TEKNOLOGI PETRONAS

TFB3113:Data Mining

Lab Report Case 1 &2 January 2025

NO.	NAME	STUDENT ID
1	Nabilah Binti Shamshir	22008816

Case 1:

For case 1, I had developed a deep learning model to classify brain tumor images into two categories: 'tumor' and 'healthy'. The dataset consists of grayscale images of varying dimensions. Following a structured approach, I did a Convolutional Neural Network (CNN).

Step 1:

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing import image
```

Python

Step 2:

```python
data_dir = "brain_tumor_dataset/"
classes = ["yes", "no"]
img_size = (300, 300)

data = []
labels = []

for class_label in classes:
    class_dir = os.path.join(data_dir, class_label)
    for img in os.listdir(class_dir):
        img_path = os.path.join(class_dir, img)
        img_arr = image.load_img(img_path, target_size=img_size, color_mode="grayscale")
        img_arr = image.img_to_array(img_arr)
        img_arr /= 255.0  # Normalize pixel values to be between 0 and 1
        data.append(img_arr)
        labels.append(classes.index(class_label))

data = np.array(data)
labels = np.array(labels)
labels = to_categorical(labels, num_classes=len(classes))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

✓ 0.3s Python

Step 3:

```python
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_size[0], img_size[1], 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(classes), activation='softmax'))

model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

✓ 0.3s Python

Step 4:

```python
history = model.fit(X_train, y_train, epochs=1, validation_data=(X_test, y_test), batch_size=32)
```

✓ 7.7s Python

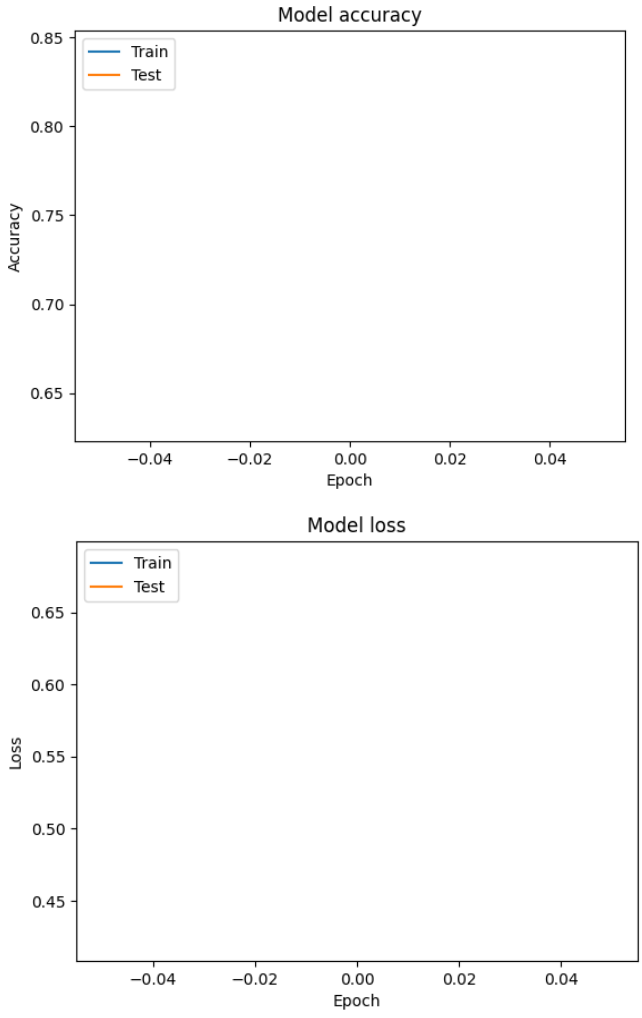
```
7/7 ━━━━━━━━━━━━━━━━━━━━ 8s 910ms/step - accuracy: 0.6334 - loss: 0.6814 - val_accuracy: 0.8431 - val_loss: 0.4216
```

Step 5:

```python
# Plotting training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plotting training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Evaluate the model
predictions = model.predict(X_test)
y_pred_classes = np.argmax(predictions, axis=1)
y_true = np.argmax(y_test, axis=1)

print(classification_report(y_true, y_pred_classes, target_names=classes))
```

✓ 0.5s Python

Output:



Case 2 :

For case 2, I had the explored **K-means clustering** for image segmentation, a key technique in computer vision. This unsupervised learning method groups similar pixels or images based on feature similarities, aiding in tasks like pattern recognition and data organization.

This study examines how **K-means works in image clustering**, its advantages, limitations, and implementation in Python using **NumPy, OpenCV, scikit-learn, and Matplotlib**. It also covers preprocessing, parameter tuning, and visualization techniques. Additionally, it highlights real-world applications and ways to enhance clustering accuracy for diverse image datasets.

Step 1:

```
pip install opencv-python
```

✓ 4.8s — Python

```
Collecting opencv-python
  Downloading opencv_python-4.11.0.86-cp37-abi3-macosx_13_0_arm64.whl.metadata (20 kB)
Requirement already satisfied: numpy>=1.21.2 in /Users/nabilah/.local/share/virtualenvs/python-7HOAHz4j/lib/python3.12/site-packages (from opencv-python)
Downloading opencv_python-4.11.0.86-cp37-abi3-macosx_13_0_arm64.whl (37.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.3/37.3 MB 10.8 MB/s eta 0:00:00a 0:00:01
Installing collected packages: opencv-python
Successfully installed opencv-python-4.11.0.86

[notice] A new release of pip is available: 25.0 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

Step 2:

```
import os
import cv2
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

✓ 15.0s — Python

Step 3:

```
image_folder_path = "/Users/nabilah/Documents/dm lab/Case Study1&2/brain_tumor_dataset/yes"
def load_images(folder_path):
    images = []
    for filename in os.listdir(folder_path):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            img_path = os.path.join(folder_path, filename)
            img = cv2.imread(img_path)
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB
            img = cv2.resize(img, (100, 100))  # Resize images for uniformity
            images.append(img)
    return np.array(images)
```

✓ 0.0s — Python

Step 4:

```
images = load_images(image_folder_path)
```
✓ 0.1s Python

```
##image clustering using kmeans

def perform_clustering(images, num_clusters=3):
    # Reshape the images to a 1D array
    reshaped_images = images.reshape(images.shape[0], -1)

    # Apply K-means clustering
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(reshaped_images)

    # Get cluster labels and cluster centers
    labels = kmeans.labels_
    centers = kmeans.cluster_centers_

    # Assign cluster labels to images
    clustered_images = []
    for i in range(len(images)):
        cluster_label = labels[i]
        clustered_images.append((images[i], cluster_label))

    return clustered_images, centers
```
✓ 0.0s Python

Step 5:

```
#displaying clustering
def display_clustered_images(clustered_images, centers):
    fig, axes = plt.subplots(1, len(centers), figsize=(20, 10))
    for i, ax in enumerate(axes):
        cluster_images = [img for img, label in clustered_images if label == i]
        ax.imshow(np.concatenate(cluster_images, axis=1))
        ax.set_title(f"Cluster {i}")
        ax.axis("off")
    plt.show()
    # Perform clustering
num_clusters = 3  # You can adjust the number of clusters as needed
clustered_images, centers = perform_clustering(images, num_clusters)

# Display clustered images
display_clustered_images(clustered_images, centers)
```
✓ 0.6s Python

Output: