| **CO4** | **Use the concept of neural networks for learning linear and non-linear activation functions** |
| --- | --- |
| **Task7**: | Apply back propagation neural network on image data. The idea is to build an Artificial Neural Network model that can effectively analyse and extract features from an image.  **Platform: Google co-lab Language: Python** |

**Use Case: Handwritten Digit Recognition**

**Problem Statement:**

The task is to build an Artificial Neural Network (ANN) that can effectively analyze and extract features from image data. The goal is often image classification, where the model learns to classify images into different categories based on the features it learns during training. We'll use a popular image dataset, such as the Fashion MNIST dataset, to demonstrate the process.

**Solution:**

The solution is a simple ANN using backpropagation for image classification. The steps include loading and preprocessing the image data, designing the neural network architecture, compiling the model, training the model, and evaluating its performance.

**Algorithm:**

1. Loading and Preprocessing Data

2. Building the Neural Network Model.

3. Choose an optimizer (e.g., Adam), a loss function (e.g., sparse categorical crossentropy for classification), and evaluation metric (e.g., accuracy).

4. Train the model on the training dataset using the fit method.

5. Specify the number of epochs and batch size.

6. Optionally, use validation data to monitor the model's performance during training.

7. Evaluate the trained model on a test dataset using the evaluate method. Assess metrics such as accuracy.

8. Plot the training and validation accuracy over epochs.

**Program:**

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import fashion\_mnist

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load and preprocess the Fashion MNIST dataset

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

# Normalize pixel values to be between 0 and 1

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

# Flatten the images to one-dimensional arrays

train\_images = train\_images.reshape((60000, 28 \* 28))

test\_images = test\_images.reshape((10000, 28 \* 28))

# Split the data into training and validation sets

train\_images, val\_images, train\_labels, val\_labels = train\_test\_split(

train\_images, train\_labels, test\_size=0.2, random\_state=42

)

# Standardize the features

scaler = StandardScaler()

train\_images = scaler.fit\_transform(train\_images)

val\_images = scaler.transform(val\_images)

test\_images = scaler.transform(test\_images)

# Build the neural network model

model = models.Sequential()

model.add(layers.Dense(128, activation='relu', input\_shape=(28 \* 28,)))

model.add(layers.Dropout(0.2))

model.add(layers.Dense(10, activation='softmax'))

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(train\_images, train\_labels, epochs=10, validation\_data=(val\_images, val\_labels))

# Evaluate the model on the test set

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f'Test accuracy: {test\_acc}')

# Plot the training and validation accuracy over epochs

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

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

def display\_images(images, labels, predictions=None):

plt.figure(figsize=(10, 4))

for i in range(5): # Displaying 5 examples

plt.subplot(2, 5, i + 1)

plt.imshow(images[i].reshape(28, 28), cmap='gray')

plt.title(f"Digit: {np.argmax(labels[i])}")

plt.axis('off')

if predictions is not None:

plt.subplot(2, 5, i + 6)

plt.bar(range(10), predictions[i])

plt.title(f"Prediction: {np.argmax(predictions[i])}")

plt.xticks(range(10))

plt.show()

display\_images(test\_images, test\_labels)

subset\_indices = np.random.choice(len(test\_images), size=5, replace=False)

subset\_images = test\_images[subset\_indices]

subset\_labels = test\_labels[subset\_indices]

predictions = model.predict(subset\_images)

display\_images(subset\_images, subset\_labels, predictions)

**Output:**

Epoch 1/10

1500/1500 [==============================] - 8s 5ms/step - loss: 0.5037 - accuracy: 0.8246 - val\_loss: 0.3805 - val\_accuracy: 0.8639

Epoch 2/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.3811 - accuracy: 0.8622 - val\_loss: 0.3562 - val\_accuracy: 0.8698

Epoch 3/10

1500/1500 [==============================] - 7s 5ms/step - loss: 0.3420 - accuracy: 0.8755 - val\_loss: 0.3397 - val\_accuracy: 0.8800

Epoch 4/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.3203 - accuracy: 0.8810 - val\_loss: 0.3510 - val\_accuracy: 0.8737

Epoch 5/10

1500/1500 [==============================] - 7s 5ms/step - loss: 0.3050 - accuracy: 0.8868 - val\_loss: 0.3312 - val\_accuracy: 0.8832

Epoch 6/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.2877 - accuracy: 0.8938 - val\_loss: 0.3533 - val\_accuracy: 0.8793

Epoch 7/10

1500/1500 [==============================] - 7s 5ms/step - loss: 0.2757 - accuracy: 0.8972 - val\_loss: 0.3518 - val\_accuracy: 0.8773

Epoch 8/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.2655 - accuracy: 0.9010 - val\_loss: 0.3478 - val\_accuracy: 0.8839

Epoch 9/10

1500/1500 [==============================] - 8s 5ms/step - loss: 0.2596 - accuracy: 0.9036 - val\_loss: 0.3402 - val\_accuracy: 0.8850

Epoch 10/10

1500/1500 [==============================] - 6s 4ms/step - loss: 0.2457 - accuracy: 0.9081 - val\_loss: 0.3493 - val\_accuracy: 0.8857

313/313 [==============================] - 1s 2ms/step - loss: 0.3816 - accuracy: 0.8776

Test accuracy: 0.8776000142097473



