EXPERIMENT 5

OBJECTIVE : WAP to train and evaluate a convolutional neural network using Keras Library to classify MNIST fashion dataset. Demonstrate the effect of filter size, regularization, batch size and optimization algorithm on model performance.

DESCRIPTION OF MODEL

The model is a CNN(Convolutional Neural Network) implemented to classify the MNIST Fashion dataset. The Fashion-MNIST dataset, consists of 70,000 grayscale images (60,000 for training, 10,000 for testing) of 10 different fashion product categories, each image being 28x28 pixels.

Input size : (28,28,1) (1 channel for grayscale images)

First Convolutional Layer: 32 filters with (3×3) kernel size , activation='relu'

Second Convolutional Layer: 64 filters with (3×3) kernel size , activation='relu'

MaxPooling is applied after each convolutional layer

Flatten layer : converts 2D vector to 1D for input to fully connected layer

First fully connected layer : 128 neurons , activation='relu'

Output fully connected layer : 10 neurons , activation='softmax'

Batch size : 32 , 64 ,128

Optimisation Algorithm : Adam , Stochastic gradient descent with momentum

Regularisation : L1 , L2

Loss function : Sparse categorical cross entropy

PYTHON IMPLEMENTATION

""" Maximum accuracy case : Batch size = 64 , filter size = (3,3),(3,3) , Optimiser algorithm = Adam , Regularisation = l2(0.0001) """

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers, regularizers

import matplotlib.pyplot as plt

import numpy as np
```

```
import seaborn as sns
import itertools
import os
from sklearn.metrics import confusion_matrix
from google.colab import files

# Check GPU availability
print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))
if tf.config.list_physical_devices('GPU'):
    print("Using GPU:", tf.config.list_physical_devices('GPU')[0])
else:
    print("No GPU found, training on CPU.")

# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Normalize the images to [0,1] range
x_train, x_test = x_train / 255.0, x_test / 255.0

# Reshape data for CNN (Adding channel dimension)
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# Create directory to save plots
os.makedirs("plots", exist_ok=True)

# Force TensorFlow to use GPU if available
with tf.device('/GPU:0'):
```

```
    # Define CNN model with regularization
    model = keras.Sequential([
      layers.Conv2D(32,(3,3),activation='relu',kernel_regularizer=regularizers.l2(0.0001), input_shape=(28,28,1)),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.0001)),
        layers.Dropout(0.5),
        layers.Dense(10, activation='softmax')
    ])

    # Compile the model with Adam optimizer
    optimizer = keras.optimizers.Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Train the model
    history = model.fit(x_train, y_train, epochs=10, batch_size=64, validation_data=(x_test, y_test))

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")

# Plot training accuracy & testing accuracy
plt.figure(figsize=(8,5))
```

```
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.legend()
plt.title("Accuracy Curve")
plt.savefig("plots/accuracy_curve4.png")  # Save the plot

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.legend()
plt.title("Loss Curve")
plt.savefig("plots/loss_curve4.png")  # Save the plot

plt.show()

y_pred = np.argmax(model.predict(x_test), axis=1)
cm = confusion_matrix(y_test, y_pred)

# Plotting the Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
```

```
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.savefig("plots/confusion_matrix4.png")  # Save plot

plt.show()


# Download Plots

files.download("plots/accuracy_curve4.png")

files.download("plots/loss_curve4.png")

files.download("plots/confusion_matrix4.png")
```

DESCRIPTION OF CODE

The code implements Convolutional Neural Network to classify the MNIST Fashion dataset.

- keras.datasets.fashion_mnist.load_data() – downloads the MNIST fashion dataset with image and label pair and splits it into training and testing samples.
- Normalisation : Image pixel values are normalized in range [o,1].
- Reshape : (28,28,1) (height, width, channels) format is input format for CNN
- CNN Model :
  - 2 Convolutional layers followed by a MaxPooling layer is used. Layer 1 consists of 32 filters with (3×3) kernel size , activation='relu' and Regularisation is applied to prevent overfitting and layer 2 consists of 64 filters with (3×3) kernel size , activation='relu' .
  - MaxPool reduces the dimension of the images in convolutional layers for feature extraction .
  - Flatten layer converts the 2D vector images to 1D for input to the fully connected layer .
  - In first fully connected layer regularisation is applied and dropout is used to prevent overfiting.
  - Output fully connected layer uses softmax activation function .
- Loss is calculated using the sparse cross entropy loss function , optimizer is used to updates the weights of the model to minimize the loss function. It adjusts the weights using gradient descent and backpropagation to improve the model's accuracy.

- model.compile() - function prepares the model for training , specifies optimiser ,loss function and metrics.
- model.fit() – Trains the model for 10 epochs and specified batch size , and at end of each epoch it checks performance on test set.\
- model.evaluate() – Evaluates the accuracy of the model.
- Accuracy curve , loss curve and confusion matrix are plotted to evaluate the performance of the model .

PERFORMANCE EVALUATION

Performance of the model is evaluated using Accuracy curve , loss curve and confusion matrix .

Maximum accuracy achieved is 91.06 % .

MY COMMENTS

- Maximum accuracy achieved is **91.06 %** for Batch size = 64 , filter size = (3,3),(3,3) , Optimiser algorithm = **Adam** , Regularisation = L2(0.0001) .
- For **Adam optimiser**
    - Regularisation L2(0.0001) ,increase in batch size from 32 to 64 increases the accuracy slightly , but for 128 , accuracy decreases , but for L2(0.001) accuracy is more for **64 batch size** in comparison with 32.
    - (Batch 64 , filter size (3,3), (5,5)) : Accuracy for **L2 regulariser** is more than for L1 regulariser.
    - (Batch 32,64) (L2(0.0001)): For Filter size (3,3) in conv layer 2 accuracy is slightly more than for (5,5) .
    - (Batch 64) Accuracy for L2(0.0001) is more than L2(0.001) .
- For **SGD** optimizer
    - (Batch size 64) : More accuracy for Filter (3,3) in second conv layer than for (5,5).
    - (Batch size 32, filter size (3,3), (3,3)) : Much more accuracy for **L2 regulariser** than for L1 regulariser.
    - Regularisation L2(0.0001) ,increase in batch size from 32 to 64 increases the accuracy slightly .
- Overall : Accuracy increases by using Adam optimizer , taking batch size 64 , filter size (3,3) , (3,3) , L2 regulariser.
- Further accuracy can be increased by using more number of convolutional layers , increasing number of filters in convolutional layers , varying different filter sizes , manipulating dropout .