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 classes, each image being 28x28 pixels.

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

Convolutional Blocks :2

- Block 1 : 2 Conv2D(32,(3,3)) layers ,activation = ReLU and Batch Normalization.
- Block 2 : 2 Conv2D(64,(3,3)) layers ,activation = ReLU and Batch Normalization.
- Block 3 : 1 Conv2D(128,(3,3)) layers ,activation = ReLU and Batch Normalization.

MaxPooling is applied after each convolutional block to reduce spatial dimention.

Dropout Layers (0.2) after each convolutional block to prevent overfitting.

Flatten layer : converts 3D(h,w,channels) vector to 1D for input to fully connected layer.

Fully connected layer :

- First layer : 512 neurons , activation= ReLU , batch normalisation.
- Dropout 0.2.
- Output fully connected layer : 10 neurons , activation='softmax'

Batch size cases : 64 ,128

Optimisation Algorithm cases : Adam , Stochastic gradient descent with momentum

Regularisation : None , Used Batch Normalisation.

Loss function : Sparse categorical cross entropy

PYTHON IMPLEMENTATION

```
import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

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

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

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

# Function to create the CNN model
def create_model():
    model = keras.Sequential([
        layers.Conv2D(32,(3,3),padding='same',activation=None, input_shape=(28,28,1)),
        layers.Activation('relu'),
        layers.BatchNormalization(),

        layers.Conv2D(32, (3,3), padding='same', activation=None),
        layers.Activation('relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Dropout(0.2),
        layers.Conv2D(64, (3,3), padding='same', activation=None),
        layers.Activation('relu'),
        layers.BatchNormalization(),

        layers.Conv2D(64, (3,3), padding='same', activation=None),
        layers.Activation('relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Dropout(0.2),

        layers.Conv2D(128, (3,3), padding='same', activation=None),
        layers.Activation('relu'),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Dropout(0.2),

        layers.Flatten(),
        layers.Dense(512, activation=None),
        layers.Activation('relu'),
        layers.BatchNormalization(),
        layers.Dropout(0.2),

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

# Define hyperparameter cases
cases = [
    (keras.optimizers.Adam(learning_rate=0.0005), 64, "Adam, Batch 64"),

    (keras.optimizers.Adam(learning_rate=0.0005), 128, "Adam, Batch 128"),

(keras.optimizers.SGD(learning_rate=0.01,momentum=0.9),64,"SGDw/Momentum,B
atch 64"),

(keras.optimizers.SGD(learning_rate=0.01,momentum=0.9),128,"SGDw/Momentum,
Batch 128")

]

# Training and evaluation for each case

for optimizer, batch_size, case_name in cases:

    print(f"\n{'='*20}\nTraining Case: {case_name}\n{'='*20}")


    model = create_model()

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


    history=model.fit(x_train,y_train,epochs=18,batch_size=batch_size,validation_data
= (x_test, y_test))


    test_loss, test_acc = model.evaluate(x_test, y_test)

    print(f"Test Accuracy ({case_name}): {test_acc:.4f}\n")


    # Plot Accuracy & Loss Curves

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


    plt.subplot(1,2,1)

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

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

    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title(f'Accuracy Curve - {case_name}')
    plt.legend()

    plt.subplot(1,2,2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Loss Curve - {case_name}')
    plt.legend()

    plt.show()

    # Confusion Matrix
    y_pred = np.argmax(model.predict(x_test), axis=1)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5,5))
    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(f'Confusion Matrix - {case_name}')
    plt.show()
```

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 [0,1].
- Reshape : (28,28,1) (height, width, channels) format is input format for CNN
- CNN Model :
  - Block 1 : 2 Conv2D layers with 32 filters and kernel size(3,3) ,activation = ReLU and Batch Normalization .
  - Block 2 : 2 Conv2D layers with 64 filters and kernel size (3,3) ,activation = ReLU and Batch Normalization.
  - Block 3 : 1 Conv2D layers with 128 filters and kernel size (3,3) ,activation = ReLU and Batch Normalization.
  - MaxPool after each convolutional block reduces the dimension of the images in convolutional layers for feature extraction .
  - Dropout Layers (0.2) after each convolutional block to prevent overfitting.
  - Flatten layer converts the 3D vector images to 1D for input to the fully connected layer .
  - In first fully connected layer has 512 neurons activation= ReLU , batch normalisation, dropout is used to prevent overfiting.
  - Output fully connected layer has 10 neurons 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 18 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 **93.44 %** .

MY COMMENTS

- Maximum accuracy achieved **is 93.44 %** for Adam optimizer and batch size 64.

- The accuracy for 2 Convolutional layers was less , max was 91.06 % , 1st having 32 filters with (3,3) kernel size , activation 'relu' and regularization , 2nd layer having 64 filters with (3,3) kernel size , activation 'relu' , the maximum accuracy achieved was just 91.06 % after changing batch sizes(32 , 64 , 128) , optimizer (Adam , Stochastic gradient descent with momentum) , kernel sizes ((3,3) , (5,5)) , regularization (L1 , L2).
- Maximum accuracy achieved is **91.06 %** for Batch size = 64 , filter size = (3,3),(3,3) , Optimiser algorithm = **Adam** , Regularisation = L2(0.0001) .

- To improve accuracy to **93.44 %** , the convolutional layers needed to be increased from 2 to 5 , increased the number of neurons in afirst fully connected layer from 128 to 512 , used batch normalization instead of regularization , used dropout after each convolutional block to reduce overfitting .
- Increasing the number of epochs for best case increased test accuracy initially gradually , but as number of epochs increased , the model overfits . For 130 epochs accuracy achieved is **94.29 %** , but further training the model for more number of epochs decreases the test accuracy or shows no change .
- Overall : Accuracy increases by using Adam optimizer , taking batch size 64 .

- 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 , using a different pooling methods .