**Objective:**

To train and evaluate a Convolutional Neural Network (CNN) using the Keras library to classify the Fashion MNIST dataset. The objective is to analyze the effect of different hyperparameters, such as filter size, regularization, batch size, and optimization algorithm, on model performance.

**Description of the Model:**

The CNN model consists of:

* Two convolutional layers with configurable filter sizes (3x3, 5x5).
* Max pooling layers for downsampling to reduce dimensionality.
* Fully connected layers for classification.
* Softmax output layer for multi-class classification.
* L2 regularization to prevent overfitting.
* Various optimization algorithms (Adam, SGD) for comparison.

**Code:**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers, regularizers

import numpy as np

import matplotlib.pyplot as plt

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.fashion\_mnist.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0 # Normalize pixel values

x\_train = x\_train.reshape(-1, 28, 28, 1)

x\_test = x\_test.reshape(-1, 28, 28, 1)

def create\_model(filter\_size=3, regularization=0.001, optimizer='adam'):

model = keras.Sequential([

layers.Conv2D(32, (filter\_size, filter\_size), activation='relu', kernel\_regularizer=regularizers.l2(regularization), input\_shape=(28, 28, 1)),

layers.MaxPooling2D(2, 2),

layers.Conv2D(64, (filter\_size, filter\_size), activation='relu'),

layers.MaxPooling2D(2, 2),

layers.Flatten(),

layers.Dense(128, activation='relu', kernel\_regularizer=regularizers.l2(regularization)),

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

])

model.compile(optimizer=optimizer, loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

return model

filter\_sizes = [3, 5]

regularization\_values = [0.001, 0.01]

batch\_sizes = [32, 64]

optimizers = ['adam', 'sgd']

results = {}

for filter\_size in filter\_sizes:

for reg in regularization\_values:

for batch\_size in batch\_sizes:

for optimizer in optimizers:

print(f"Training model with filter\_size={filter\_size}, reg={reg}, batch\_size={batch\_size}, optimizer={optimizer}")

model = create\_model(filter\_size, reg, optimizer)

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=batch\_size, validation\_data=(x\_test, y\_test), verbose=0)

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=0)

results[(filter\_size, reg, batch\_size, optimizer)] = (history, test\_acc)

plt.figure(figsize=(12, 6))

for key, (history, acc) in results.items():

filter\_size, reg, batch\_size, optimizer = key

plt.plot(history.history['val\_accuracy'], label=f'fs={filter\_size}, reg={reg}, bs={batch\_size}, opt={optimizer} ({acc:.2f})')

plt.xlabel('Epochs')

plt.ylabel('Validation Accuracy')

plt.legend()

plt.title('Effect of Hyperparameters on Validation Accuracy')

plt.show()

**Description of Code:**

* The create\_model function constructs a CNN with:
  + Two convolutional layers with configurable filter sizes.
  + Max pooling layers for feature reduction.
  + A fully connected layer followed by a softmax layer for classification.
  + L2 regularization to prevent overfitting.
* A nested loop iterates over different hyperparameter values to test multiple configurations.
* Models are trained for 10 epochs each using the given hyperparameters.
* The evaluate() function measures accuracy on the test dataset.
* Visualization using Matplotlib:
  + A line plot shows validation accuracy trends across different configurations.
  + A training loss vs validation loss plot helps identify overfitting and generalization trends.

**Performance Evaluation:**

* The model's accuracy is tested with different hyperparameters to observe their effects.
* **Filter Size**: Affects feature extraction, with larger filters capturing more details but requiring more computations.
* **Regularization**: Helps in preventing overfitting but may reduce training accuracy.
* **Batch Size**: Smaller batch sizes may improve generalization but increase training time.
* **Optimizer Comparison**:
  + Adam generally provides better accuracy compared to SGD.
  + SGD may take longer to converge but can generalize well in some cases.
* **Loss Visualization**:
  + Training loss vs validation loss is plotted over epochs to understand if the model overfits.
  + Helps determine the best epoch count for optimal performance.
* A line plot visualizes how validation accuracy changes with different configurations, helping identify the best-performing setup.

**My Comments:**

1. Adam Optimizer Performs Better than SGD

* The curves corresponding to Opt=adam consistently reach higher validation accuracy compared to Opt=sgd.
* Adam adapts the learning rate dynamically, leading to faster and more stable convergence.

2. Lower Regularization (0.0001) Leads to Higher Accuracy

* Models with Reg=0.0001 perform better than those with Reg=0.001.
* Higher regularization (0.001) imposes more constraints, possibly restricting the model’s learning ability.

3. Batch Size 64 vs. 32

* Larger batch size (64) stabilizes training but may lead to slightly lower validation accuracy compared to batch size 32.
* Smaller batch size (32) allows for more weight updates per epoch, which can help generalization.

-The highest validation accuracy is (~90%) after 5 epochs.