In this task, we build a simple neural network using TensorFlow and Keras for classifying images from the Fashion MNIST dataset, which consists of grayscale images of clothing articles. We start by preparing and normalizing the data, then construct a neural network with one hidden layer and compile it with appropriate loss functions and optimizers. The model is trained with validation and evaluated on a test set, where performance is further analyzed using metrics like accuracy and a confusion matrix.

WEEK 4

Building a Simple Neural Network

ML-WEEK 4.1

Noor Ul Ain

Task 4.1: Building a Simple Neural Network

Objective

Introduce the basics of neural networks and build a simple model using TensorFlow and Keras for image classification.

Dataset

Fashion MNIST dataset, which is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

- **Dataset URL:** Directly available in TensorFlow/Keras datasets. You can load it using the following command:

```
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Activities

1. **Data Preparation:**
    - Load the dataset.
    - Normalize the image data to the range [0, 1].
    - Reshape the data if necessary and prepare it for training.
2. **Model Building:**
    - Create a simple neural network architecture using Keras with at least one hidden layer.
    - Compile the model with an appropriate loss function and optimizer.
3. **Training:**
    - Train the model on the training data with validation using a portion of the training set.
    - Use callbacks like ModelCheckpoint or EarlyStopping to enhance training.
4. **Evaluation:**
    - Evaluate the model on the test set to check its performance.
    - Use metrics such as accuracy and create a confusion matrix to visualize the classification performance.

ACTIVITIES

1. DATA PREPARATION

Load the Dataset

First, let's load the dataset. The fashion_mnist.load_data() function returns two tuples: one for the training data and one for the test data.

```
from tensorflow.keras.datasets import fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Normalize the Image Data

Neural networks work better when input data is scaled to a range of [0, 1]. To normalize the data, divide the pixel values by 255.

```
# Normalize pixel values to be between 0 and 1

train_images = train_images / 255.0

test_images = test_images / 255.0
```

Reshape the Data

The images are already in the correct shape (28x28), but if needed, ensure the data is in the format (number of samples, height, width, number of channels). For grayscale images, there is only one channel.

```
# Adding a channel dimension (for grayscale images)

train_images = train_images[..., np.newaxis]

test_images = test_images[..., np.newaxis]
```

2. MODEL BUILDING

Create a Simple Neural Network

We'll use Keras to build a simple neural network with one hidden layer.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Initialize the model
model = Sequential()

# Flatten the input image to 1D
model.add(Flatten(input_shape=(28, 28, 1)))

# Add a hidden layer with 128 neurons and ReLU activation
model.add(Dense(128, activation='relu'))

# Add the output layer with 10 neurons (one for each class) and softmax activation
model.add(Dense(10, activation='softmax'))
```

Compile the Model

Compile the model with an appropriate loss function, optimizer, and evaluation metrics.

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

- **Optimizer:** Adam - a popular optimization algorithm.
- **Loss Function:** Sparse Categorical Crossentropy - suitable for multi-class classification problems.

- **Metrics:** Accuracy - to evaluate the performance.

3. TRAINING

Train the Model

Train the model using the training data. We'll also use a validation split to monitor the model's performance on a portion of the training data.

```
history = model.fit(train_images, train_labels,
                    epochs=10,
                    validation_split=0.2,
                    callbacks=[tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)])
```

- **Epochs:** Number of times to iterate over the training dataset.
- **Validation Split:** Fraction of the training data to be used for validation.
- **EarlyStopping:** Stops training when a monitored metric has stopped improving.

4. EVALUATION

Evaluate the Model

Evaluate the model's performance on the test set.

```
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

Confusion Matrix

To visualize the classification performance, create a confusion matrix.

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

```
# Predict the labels for the test set
predictions = model.predict(test_images)
predicted_labels = predictions.argmax(axis=1)


# Compute confusion matrix
cm = confusion_matrix(test_labels, predicted_labels)


# Plot the confusion matrix
plt.figure(figsize=(10, 7))
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.show()
```

LIBRARIES USED

- **TensorFlow:** An open-source library for machine learning and deep learning. We use it for model building and training.
- **Keras:** An API within TensorFlow for building and training neural networks. We use it for defining the model architecture and compiling it.
- **NumPy:** Used for numerical operations, such as normalizing the image data and reshaping arrays.
- **Scikit-learn:** Provides tools for creating confusion matrices and evaluating model performance.
- **Matplotlib and Seaborn:** Used for visualizing the confusion matrix.