Alden A. Quiñones
MIT 266 – Special Topic
Machine Learning
Laboratory Exercise 6

Classification with Neural Networks

Code Snippet #1: Code Snippet for downloading historical data of Apple stock from Yahoo Finance

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
fashion = keras.datasets.fashion_mnist
(xtrain, ytrain), (xtest, ytest) = fashion.load_data()
```

Function:

This code snippet loads the Fashion MNIST dataset from TensorFlow's Keras API. It first imports the necessary libraries, including TensorFlow, Keras, and NumPy. Then, it loads the dataset into training and testing sets (xtrain, ytrain, xtest, and ytest) through fashion.load_data(), which automatically fetches the dataset from the provided URL if not already available locally.

Figure#1: Output

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

Description:

The output shows that the Fashion MNIST dataset is being downloaded from the TensorFlow servers. The data consists of 60,000 training images and 10,000 testing images, each labeled with a category of clothing. The downloaded data includes both images and labels, which are unpacked and stored into xtrain, ytrain for the training set and xtest, ytest for the test set. The data is ready for further processing and use in training machine learning models.

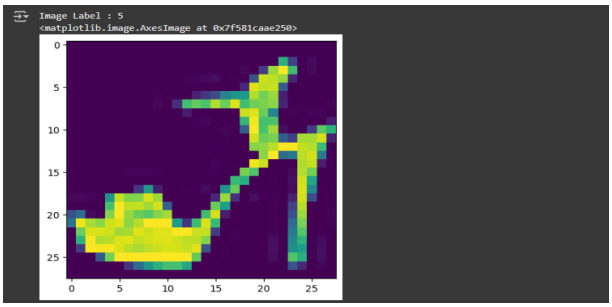
Code Snippet #2: Displaying an Image from the Fashion MNIST Dataset

```
imgIndex = 9
image = xtrain[imgIndex]
print("Image Label :",ytrain[imgIndex])
plt.imshow(image)
```

Function:

This code snippet selects an image from the training set (xtrain) by specifying an index (imgIndex). It then prints the label of that image from ytrain and displays the image using Matplotlib's imshow() function.

Figure#2: Output



Description:

The output shows that the label of the image at index 9 is "5," which corresponds to a specific category in the Fashion MNIST dataset (in this case, "Sandal"). The image is then displayed using plt.imshow(), providing a visual representation of the selected clothing item. The displayed image will be shown in the output window (typically a plot area), with the label printed alongside it.

Code Snippet #3: Inspecting the Shape of Training and Test Data

```
print(xtrain.shape)
print(xtest.shape)
```

Function:

This code snippet prints the shapes of the training and test datasets (xtrain and xtest). These datasets contain grayscale images from the Fashion MNIST dataset. The shape provides insight into the dimensions of the data arrays.

Figure#3: Output

```
(60000, 28, 28)
(10000, 28, 28)
```

Description:

The output shows that the training dataset (xtrain) contains 60,000 images, each of size 28x28 pixels, and the test dataset (xtest) contains 10,000 images of the same 28x28 pixel size. Specifically, the shape (60000, 28, 28) for xtrain indicates that there are 60,000 images, each with 28 rows and 28 columns of pixel values. Similarly, (10000, 28, 28) for xtest indicates that there are 10,000 test images with the same dimensions. These shapes are typical for the Fashion MNIST dataset, where each image is a grayscale image (with pixel values between 0 and 255) representing a clothing item.

Alden A. Quiñones
MIT 266 – Special Topic
Machine Learning
Laboratory Exercise 6

Code Snippet #4: Defining a Neural Network Model and Inspecting Its Summary

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
print(model.summary())
```

Function:

This code snippet defines a neural network model using Keras' Sequential API. The model consists of a Flatten layer that reshapes the input images from a 28x28 shape into a flat vector, followed by two Dense layers with ReLU activation, and a final Dense layer with softmax activation for classification.

Figure#4: Output

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100

 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
None
```

Description:

The output shows that the model has been successfully created with four layers. The Flatten layer reshapes the 28x28 input image into a vector of size 784, with no trainable parameters. The first Dense layer has 300 neurons and uses ReLU activation, resulting in 235,500 trainable parameters. The second Dense layer has 100 neurons with ReLU activation, contributing 30,100 trainable parameters. Finally, the Dense output layer has 10 neurons with softmax activation, producing class probabilities, and has 1,010 trainable parameters. In total, the model has 266,610 trainable parameters that will be adjusted during training to minimize the loss.

Alden A. Quiñones
MIT 266 – Special Topic
Machine Learning
Laboratory Exercise 6

Code Snippet #5: Training the Model on the Fashion MNIST Dataset

```
[ ] xvalid, xtrain = xtrain[:5000]/255.0, xtrain[5000:]/255.0
    yvalid, ytrain = ytrain[:5000], ytrain[5000:]

[ ] model.compile(loss="sparse_categorical_crossentropy",
                  optimizer="sgd",
                  metrics=["accuracy"])
    history = model.fit(xtrain, ytrain, epochs=30,
                        validation_data=(xvalid, yvalid))
```

Function:

This code snippet splits the training data into training and validation sets, normalizes the pixel values, compiles the model with the sparse categorical crossentropy loss function and SGD optimizer, and trains the model for 30 epochs while monitoring validation accuracy.

Figure#5: Output

```
Epoch 1/30
1719/1719 [==============================] - 11s 3ms/step - loss: 0.7248 - accuracy: 0.7631 - val_loss: 0.5160 - val_accuracy: 0.8264
Epoch 2/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.4928 - accuracy: 0.8283 - val_loss: 0.4568 - val_accuracy: 0.8378
Epoch 3/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.4478 - accuracy: 0.8430 - val_loss: 0.4309 - val_accuracy: 0.8550
Epoch 4/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.4211 - accuracy: 0.8517 - val_loss: 0.4034 - val_accuracy: 0.8600
Epoch 5/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.4000 - accuracy: 0.8594 - val_loss: 0.3906 - val_accuracy: 0.8672
Epoch 6/30
...
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2435 - accuracy: 0.9129 - val_loss: 0.2979 - val_accuracy: 0.8942
Epoch 27/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2387 - accuracy: 0.9149 - val_loss: 0.3014 - val_accuracy: 0.8944
Epoch 28/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2346 - accuracy: 0.9156 - val_loss: 0.3087 - val_accuracy: 0.8882
Epoch 29/30
1719/1719 [==============================] - 5s 3ms/step - loss: 0.2303 - accuracy: 0.9181 - val_loss: 0.3238 - val_accuracy: 0.8848
```

Description:

The output shows the training process over 30 epochs, where the model is trained on the xtrain data and validated on the xvalid set. During each epoch, the model's loss and accuracy on both the training set (accuracy) and the validation set (val_accuracy) are reported. Over time, the training loss decreases, indicating the model is learning, while the validation accuracy increases, showing that the model is generalizing well. By the end of 30 epochs, the model achieves an accuracy of 91.88% on the training set and 88.80% on the validation set. The validation accuracy fluctuates slightly between epochs, but overall, the model shows significant improvement over the course of the training.

Code Snippet #6: Making Predictions on New Data

```
[ ] new = xtest[:5]
    predictions = model.predict(new)
    print(predictions)
```

Function:

This code snippet selects the first five test images (new = xtest[:5]), uses the trained model to predict the class probabilities for each image, and prints the resulting predictions. The predict() method of the model returns the predicted probability distributions for each class (0 to 9).

Figure#6: Output

```
1/1 [==============================] - 0s 78ms/step
[[0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

Description:

The output shows the model's predicted probabilities for each of the five test images. Each prediction is a vector of 10 values corresponding to the 10 classes in the Fashion MNIST dataset, with the highest value indicating the predicted class. For example:

Code Snippet #7: Converting Prediction Probabilities to Class Labels

```
[ ] classes = np.argmax(predictions, axis=1)
    print(classes)
```

Function:

This code snippet uses np.argmax() to convert the predicted class probabilities (from the previous predictions array) into the final class labels by selecting the index of the highest value for each prediction.

Figure#6: Output

```
[9 2 1 1 0]
```

Description:

The output shows the class labels for each of the five test images, which are derived from the predicted probabilities. Using np.argmax(predictions, axis=1), the model returns the index of the maximum value in each row, representing the predicted class label. For example, the first image is predicted to belong to class 9 (likely "Ankle boot"), the second image to class 2 (likely "Pullover"), and both the third and fourth images to class 1 (likely "Trouser"). The fifth image is predicted to belong to class 0 (likely "T-shirt/top"). This step provides the final predicted labels, which can be directly compared to the true labels for evaluation or used for further processing.