CS5330 Spring 2024
Lab 3 - Image Classification using Machine Learning
Kay (Mengxian) Cai

Project Report on Image Classification using Machine Learning

I. Introduction

For this lab, we will design, train and test three classifiers to classify the images in the Fashion-MNIST dataset (https://github.com/zalandoresearch/fashion-mnist) which consists 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.

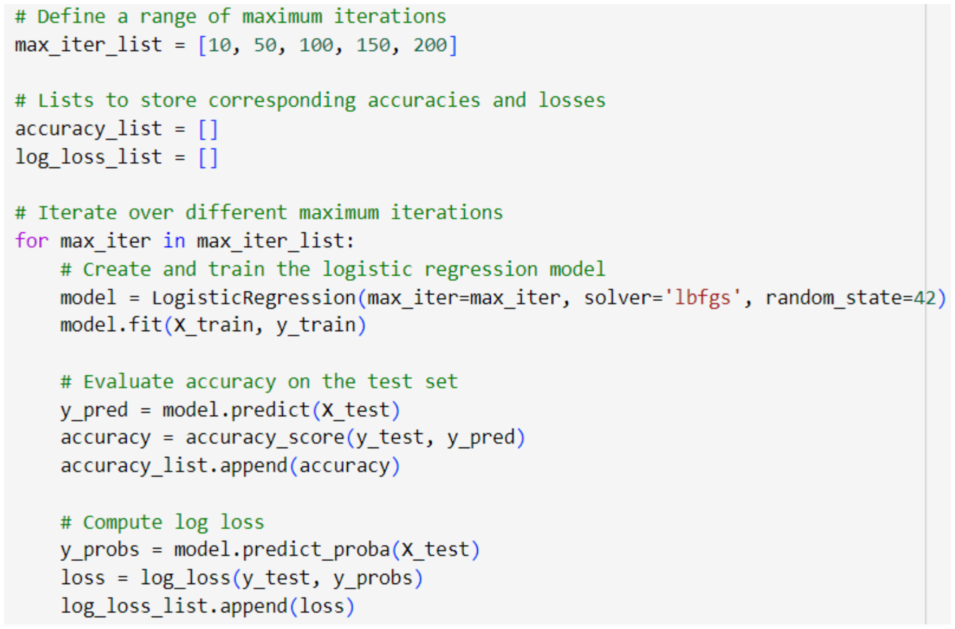
The three classifiers we will train include (a) linear classifier, (b) neural network classifier, and (c) convolutional neural network (CNN) classifier.

II. Methods

A. Linear classifier

For the linear classifier, we choose Scikit-learn's Logistic Regression model, which uses the one-vs-rest (OvR) strategy for multiclass problems like Fashion-MNIST, and the class with the highest probability is chosen as the final prediction.

We scaled the features of images and tested adjusting different parameters including maximum iterations and regularization strength:



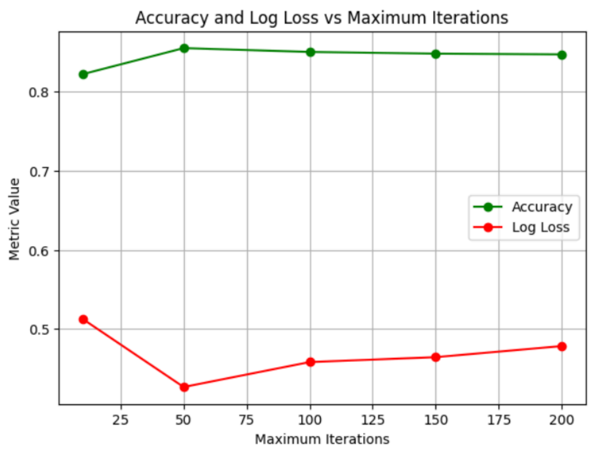
```python
# Define a range of maximum iterations
max_iter_list = [10, 50, 100, 150, 200]

# Lists to store corresponding accuracies and losses
accuracy_list = []
log_loss_list = []

# Iterate over different maximum iterations
for max_iter in max_iter_list:
    # Create and train the logistic regression model
    model = LogisticRegression(max_iter=max_iter, solver='lbfgs', random_state=42)
    model.fit(X_train, y_train)

    # Evaluate accuracy on the test set
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_list.append(accuracy)

    # Compute log loss
    y_probs = model.predict_proba(X_test)
    loss = log_loss(y_test, y_probs)
    log_loss_list.append(loss)
```



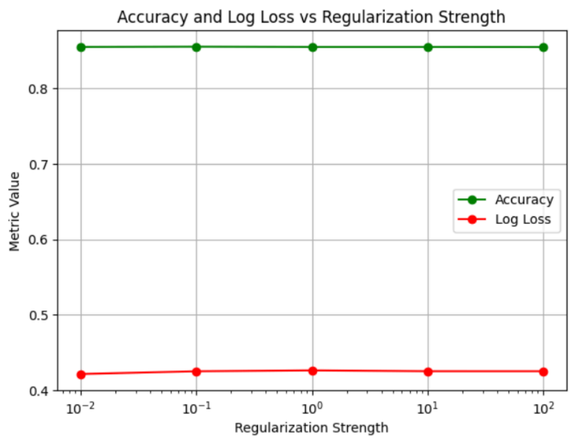
```python
# Define a range of regularization strengths
C_values_list = [0.01, 0.1, 1, 10, 100]

# Lists to store corresponding accuracies
C_accuracy_list = []
C_log_loss_list = []

# Iterate over different regularization strengths
for C in C_values_list:
    # Create and train the logistic regression model
    model = LogisticRegression(max_iter=50, solver='lbfgs', random_state=42, C=C)
    model.fit(X_train, y_train)

    # Evaluate accuracy on the test set
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    C_accuracy_list.append(accuracy)

    # Compute log loss
    y_probs = model.predict_proba(X_test)
    loss = log_loss(y_test, y_probs)
    C_log_loss_list.append(loss)
```



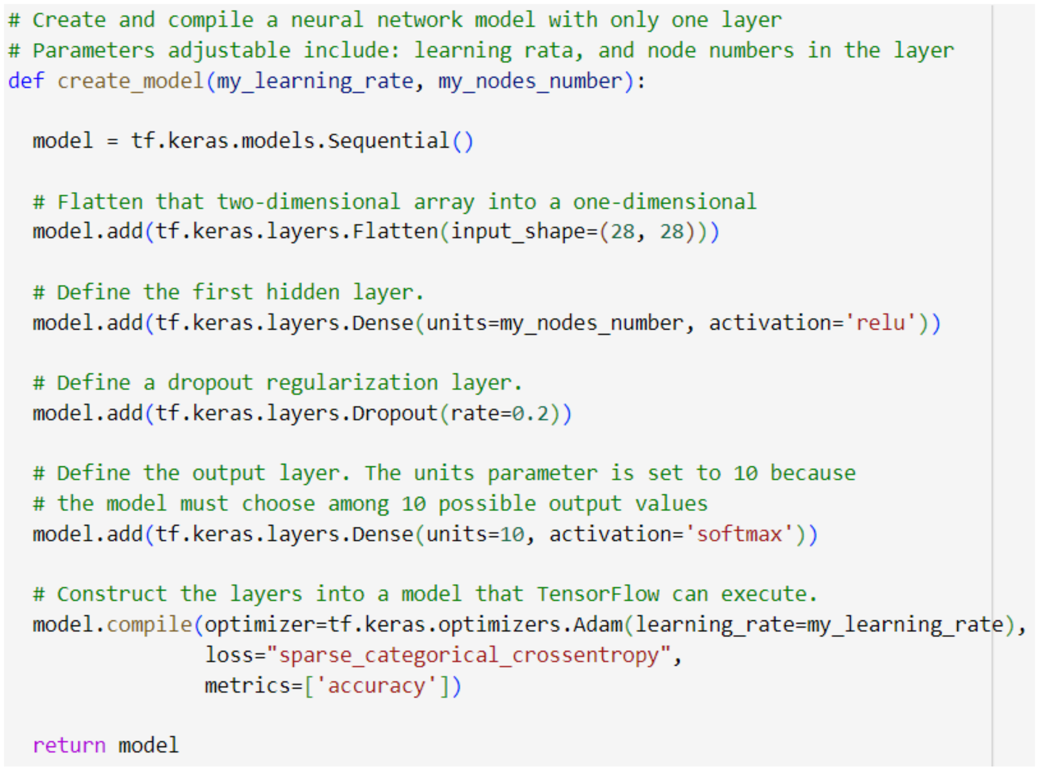
The model with the best performance has the following parameters:

- Maximum iterations: 50;
- Solver: lbfgs;
- Regularization strength: 0.1 (the difference brought by regularization strength is so trivial that could be disregarded).

B. Neural network classifier

We use TensorFlow Keras library to build the neural network classifier, this classifier has only one layer and we adjusted the parameters including learning rate, number of nodes in the layer, batch size, and epochs to understand the impact of different parameters on the model's performance.

Below is one of the code snippets of building the model:



```python
# Create and compile a neural network model with only one layer
# Parameters adjustable include: learning rata, and node numbers in the layer
def create_model(my_learning_rate, my_nodes_number):

  model = tf.keras.models.Sequential()

  # Flatten that two-dimensional array into a one-dimensional
  model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))

  # Define the first hidden layer.
  model.add(tf.keras.layers.Dense(units=my_nodes_number, activation='relu'))

  # Define a dropout regularization layer.
  model.add(tf.keras.layers.Dropout(rate=0.2))

  # Define the output layer. The units parameter is set to 10 because
  # the model must choose among 10 possible output values
  model.add(tf.keras.layers.Dense(units=10, activation='softmax'))

  # Construct the layers into a model that TensorFlow can execute.
  model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=my_learning_rate),
                loss="sparse_categorical_crossentropy",
                metrics=['accuracy'])

  return model
```

The table below summarizes the impact of various parameters on the model. We can note that the number of neurons plays a vital role in improving the performance, while the influence of the learning rate appears less consistent, occasionally leading to lower accuracy with higher values. For this dataset, an increased batch size demonstrates a positive correlation with improved results.

Learning rate	***Nodes number***	***Batch size***	***Epochs***	***Validation accuracy***	***Validation loss***
0.003	32	64	10	0.8648	0.3786
0.004	32	64	10	0.8611	0.3909
0.003	128	64	10	0.8808	0.3491
0.004	128	64	10	0.8798	0.3479
0.003	256	64	10	0.8820	0.3350
0.003	256	500	10	0.8884	0.3129
0.004	256	64	10	0.8871	0.3342
0.003	256	64	30	0.8838	0.3612
0.004	256	64	30	0.8838	0.3697
0.003	256	500	50	0.8971	0.3387
0.004	256	500	50	0.8937	0.3549
0.005	256	64	10	0.8712	0.3609
0.005	256	500	10	0.8881	0.5697

We highlight the group of parameters achieving the best results (both in terms of the validation accuracy and loss) in red above. This model has the following parameters and its accuracy on the test set is 89.07%.

- Learning rate: 0.03;
- Neurons in dense layer: 256;
- Batch size: 500;
- Epochs: 50.

However, when applying the models to the test set, the model with a learning rate of 0.004 and other parameters same as the above achieved the highest accuracy rate of 89.27%.

C. CNN classifier

We use the TensorFlow Keras library to build four models for the CNN classifier, as detailed below:

Model Description	***Validation accuracy***	***Validation loss***
LeNet 2 Conv + 3 FC (15,626 params)	0.8658	0.3637
Model 1 3 Conv + 2 FC (93,322 params) Reference: Deep Learning with Python Notebooks, First edition, Chapter 5.1	0.9115	0.4528
Model 2 3 Conv + 2 FC (294,922 params) Based on Model 1, but doubled output channels in the third convolutional layer Add one dropout layer before the final output layer to address Model 1's overfitting	0.8995	0.3482
Model 3 2 Conv +3 FC (493,902 params) 3 Dropout layers With data augmentation and batch normalization Reference: zalando-fashion-mnist by Christopher Masch	0.9268	0.2051

The best performing Model 3's detailed structure and parameters are as below:

- First batch normalization layer
- First convolutional layer: window shape 4 x 4, 64 output channels + 2 x 2 maxpooling operation
- Dropout layer: 0.1 dropout rate
- Second convolutional layer: window shape 4 x 4, 64 output channels + 2 x 2 maxpooling operation
- Dropout layer: 0.3 dropout rate
- Three fully connected layers, with 256, 64, and 10 outputs, respectively
- One dropout layer at the rate of 0.5 between the first and second fully connected layers
- One batch normalization layer before the final dense layer
- Data augmentation on the train set generate over 40,000 additional images

III. Results and Limitations

The table below shows the highest accuracy rate of each classifier on the test set:

Classifier	***Test accuracy***
Linear	85.56%

Neuron network	89.27%
CNN	91.88%

From the table, we can tell CNN apparently outperforms the other two classifiers, which aligns with the strength of CNN in capturing spatial relationships and patterns within images. The convolutional layers, with their ability to learn local features and hierarchies, make CNN well-equipped to discern intricate patterns within the images and achieve the highest accuracy.

In contrast, the linear classifier provides the least favourable performance. Its limitation lies in its linear decision boundaries, which may struggle to capture the complex relationships present in image data.

The neural network classifier worked better than the linear classifier because of its capacity to model non-linear relationships within the data, and therefore can extract meaningful features for image classification. On the other hand, it still lags behind CNN because CNN incorporates convolutional layers, which are adept at learning spatial hierarchies and enable the network to recognize local patterns irrespective of their location in the image.

Nevertheless, due to the limited timetable and knowledge about machine learning, we are not able to spend sufficient effort to fine-tune all the models, which may result in imprecise observations. Adjustments on a limited number of parameters are also not comprehensive enough for thorough tests. Moreover, the size and characteristics of the dataset also affect the model generalization. In this study, the dataset's specific features may have favoured CNN, and further exploration is needed to determine whether additional feature engineering or alternative models could mitigate the limitations.

IV. Conclusion

The Fashion-MNIST dataset is more challenging than the traditional MNIST dataset because (a) clothing items are more complex and diverse than simple numbers, and (b) some fashion items may share visual similarities (e.g., sneakers and boots), making it harder for models to distinguish between them based on subtle differences.

Undoubtedly, CNN is the best classifier for this task because it is highly effective for capturing spatial dependencies in images. Some other key findings include: (i) parameters, especially critical parameters, could have a significant influence on the model’s test, but it takes time and requires experience to identify those key parameters and how to adjust them; (ii) there is potential variability in model performance between the validation set and the test set. While the performance on the validation set serves as a valuable benchmark during model development, our tests in the neural network models have shown that it may not unequivocally predict the model's performance on unseen data, particularly the test set.

As such, we should exercise careful consideration to the unpredictability associated with model performance across different datasets, and should be cautious when interpreting the results and making predictions about real-world performance. This also serves as a reminder that running additional tests on various subsets of the test set can provide a more nuanced understanding of the model's generalization capabilities.