CS770 Machine Learning

Assignment 3-2: Fashion MNIST and Neural Networks

04/30/2025

Submitted by: Logan Schraeder (x356t577)

Introduction

This report details the development and evaluation of a deep learning model for image classification on the Fashion MNIST dataset. The Fashion MNIST dataset consists of 70,000 grayscale images of clothing items, categorized into 10 distinct classes. Each image is 28×28 pixels. The primary objective of this part of the assignment is to implement and assess a specific neural network architecture incorporating techniques such as batch normalization and dropout to potentially enhance performance and generalization.

Let's build such a neural network to tackle the California housing problem:

```
input = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.Concatenate()([input, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input], outputs=[output])
```

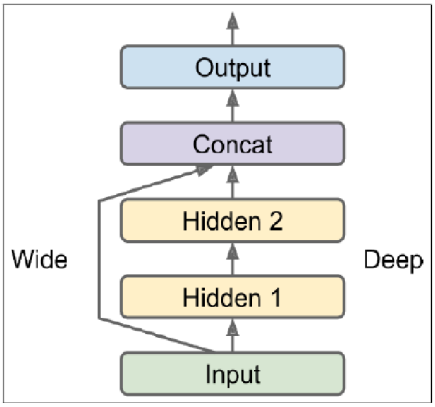

Figure 1. Base Neural Network Configuration. The neural network architecture as dictated by the assignment

Methodology

Data Preparation

The Fashion MNIST dataset was loaded using the tensorflow.keras.datasets module. The dataset was initially split into training and testing sets, comprising 60,000 and 10,000 images respectively. Image pixel values, originally in the range of 0 to 255, were normalized to the range [0, 1] by casting the image data to float32 and dividing by 255. This normalization step is crucial for neural networks as it helps in faster convergence during training. The labels were kept in their original integer format, suitable for use with the sparse_categorical_crossentropy loss function.

Model Architecture

The neural network architecture implemented for this task follows the specific structure outlined in the assignment instructions, incorporating batch normalization and dropout layers. The model is a Keras Model defined using the Functional API, allowing for more flexible network designs, including the concatenation of layers.
The model architecture is as follows:

- **Input Layer:** An Input layer is defined with the shape matching the input images (28×28).
- **Hidden Layer 1:** A Dense layer with 30 units and ReLU activation is connected to the input layer. ReLU is chosen for its computational efficiency and effectiveness in mitigating the vanishing gradient problem.
- **Batch Normalization 1:** A BatchNormalization layer is applied after the first hidden layer. Batch normalization helps stabilize and accelerate the training process by normalizing the inputs to layers, reducing the dependence on the initialization and learning rates.
- **Hidden Layer 2:** Another Dense layer with 30 units and ReLU activation is connected to the output of the first batch normalization layer.
- **Batch Normalization 2:** A second BatchNormalization layer is applied after the second hidden layer.
- **Dropout:** A Dropout layer with a rate of 0.5 is added after the second batch normalization layer. Dropout is a regularization technique that randomly sets a fraction of input units to 0 at each training step, which helps prevent overfitting.
- **Concatenation Layer:** A Concatenate layer merges the output of the initial Input layer and the output of the dropout layer. This creates a "wide and deep" aspect to the network, allowing it to potentially learn both simple patterns from the raw input and more complex features from the hidden layers.
- **Flatten Layer:** A Flatten layer is applied to the concatenated output to convert the 2D output into a 1D vector, which is required for the final dense layer.
- **Output Layer:** A Dense layer with 10 units (corresponding to the 10 Fashion MNIST classes) and a softmax activation function is used. The softmax activation provides a probability distribution over the classes.

The model was compiled using the 'adam' optimizer and sparse_categorical_crossentropy as the loss function, suitable for multi-class classification with integer labels. 'accuracy' was used as the evaluation metric.

Training and Evaluation

The model was trained on the normalized training data for 10 epochs. Validation was performed using the normalized test data to monitor performance and identify potential overfitting. Following training, the model's performance was evaluated on the test dataset. Predictions were generated for the test images. A confusion matrix and a classification report

were computed using sklearn.metrics to provide a detailed analysis of the model's performance across each class, including precision, recall, and F1-score. The training performance can be seen in Table 1.

Results

The model was trained for 10 epochs. The training process showed a steady improvement in both training accuracy and validation accuracy, while training and validation loss decreased.

Epoch	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss
1	0.7631	0.6698	0.8440	0.4229
2	0.8523	0.4070	0.8654	0.3879
3	0.8669	0.3725	0.8712	0.3691
4	0.8725	0.3513	0.8727	0.3669
5	0.8793	0.3353	0.8789	0.3537
6	0.8825	0.3253	0.8726	0.3616
7	0.8860	0.3143	0.8750	0.3441
8	0.8882	0.3103	0.8665	0.3773
9	0.8865	0.3097	0.8807	0.3370
10	0.8903	0.3015	0.8786	0.3489

The final accuracy on the test dataset was approximately 87.86%. The confusion matrix and classification report below provide a more detailed breakdown of the model's performance per class.

```
---Q2 CNN Classification Report---
              precision    recall  f1-score   support

           0       0.80      0.86      0.83      1000
           1       0.99      0.96      0.97      1000
           2       0.78      0.84      0.81      1000
           3       0.83      0.92      0.87      1000
           4       0.77      0.85      0.81      1000
           5       0.97      0.96      0.97      1000
           6       0.80      0.50      0.62      1000
           7       0.94      0.95      0.94      1000
           8       0.96      0.97      0.97      1000
           9       0.95      0.96      0.96      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.87     10000
weighted avg       0.88      0.88      0.87     10000
```

Figure 2. Neural Network Classification Report. Despite being a relatively vanilla neural network, the model performed well in training on the MNIST fashion dataset.

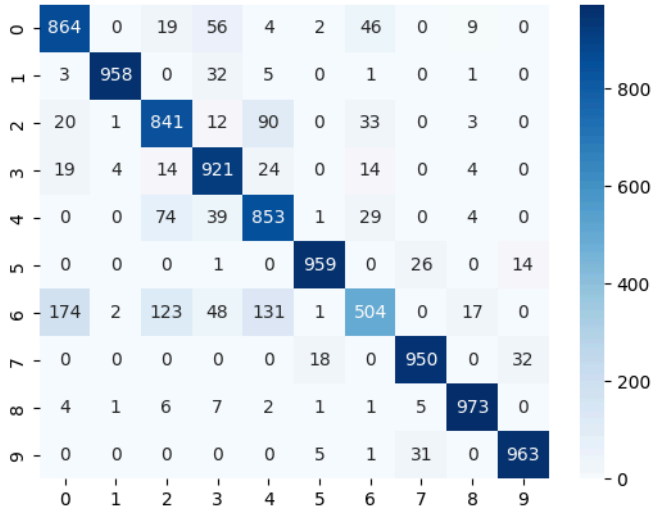

Figure 3. Neural Network Confusion Matrix. The network performed well and tended to mis-classify clothing articles as a false negative more than false positive.

Analysis

The overall test accuracy of approximately 87.86% indicates that the model performs reasonably well on the Fashion MNIST classification task. However, the confusion matrix and classification report reveal significant variations in performance across different classes. Classes 1 (Trouser), 5 (Sandal), 7 (Sneaker), 8 (Bag), and 9 (Ankle boot) show high precision, recall, and F1-scores (all above 0.94), suggesting the model is highly effective at identifying these items with minimal confusion. This is likely due to these items having more distinct visual features compared to others. Conversely, Class 6 (Shirt) exhibits the lowest recall (0.50) and F1-score (0.62). The confusion matrix shows that a large number of shirts (Class 6) are misclassified as Class 0 (T-shirt/top), Class 2 (Pullover), and Class 4 (Coat).

This is an expected challenge with the Fashion MNIST dataset, as these clothing items share similar shapes and textures, making differentiation difficult for the model. Similarly, Class 0 (T-shirt/top), Class 2 (Pullover), and Class 4 (Coat) also show lower performance metrics compared to the well-classified classes, and the confusion matrix indicates significant misclassification among these categories as well. For instance, Class 2 (Pullover) is often confused with Class 4 (Coat) and Class 0 (T-shirt/top), and Class 4 (Coat) is frequently misclassified as Class 2 (Pullover).

The inclusion of batch normalization appears to have contributed to stable training, as evidenced by the smooth decrease in loss and increase in accuracy over epochs. Batch normalization helps in handling the internal covariate shift, allowing for higher learning rates and faster convergence. The dropout layer, applied with a rate of 0.5, acts as a regularizer. While the training accuracy is higher than validation accuracy (as expected with dropout), the gap is not excessively large, suggesting that overfitting is being mitigated to some extent. The

"wide" connection from the input layer to the concatenation layer allows the model to directly use raw pixel information, potentially helping with simpler patterns, while the "deep" path through the dense layers allows for learning more complex, abstract features.

Conclusion

The implemented deep learning model, incorporating batch normalization and dropout, achieved a test accuracy of approximately 87.86% on the Fashion MNIST dataset. The model demonstrated strong performance on easily distinguishable classes like trousers, sandals, sneakers, bags, and ankle boots. However, it struggled to differentiate between visually similar items such as shirts, t-shirts/tops, pullovers, and coats, leading to lower recall and F1-scores for these categories.

Batch normalization likely aided in the stable and relatively fast convergence of the model. Dropout appears to have provided some regularization against overfitting, although a performance gap between training and validation accuracy remains. The "wide and deep" architecture aimed to leverage both raw and abstract features. For future improvements, exploring techniques specifically designed to handle fine-grained classification or visual similarity could be beneficial. This might include:

- Using convolutional layers, which are typically more effective at capturing spatial and image-based features from sample data
- Implementing data augmentation to expose the model to a wider variety of image variations (rotations, shifts, zooms), which could improve robustness and generalization, especially for the classes that are currently misclassified.
- Experimenting with different network depths and widths, optimizer parameters, and regularization strengths.
- Investigating more advanced architectures or ensemble methods, as explored in Q1 of the assignment.