Deep Learning Assignment Report: Image Classification with Fashion MNIST Dataset

Anami A Misheni REGNO: **IN13/00052/19**

Otieno Gabriel REGNO: **IN13/00068/20**

Victor Thomas REGNO: **IN13/00074/20**

COMP 475 (Reinforced and Deep Learning)

Submission Date: [13/03/2024]

1. Introduction

In this assignment, our objective was to develop a neural network model for image classification using the Fashion MNIST dataset. The dataset consists of 28x28 grayscale images of fashion products categorized into 10 types. Our tasks included data preprocessing, model development, training, evaluation, and analysis.

2. Data Preprocessing

2.1 Loading and Normalizing Data

We loaded the Fashion MNIST dataset using TensorFlow's Keras API. The image data was normalized to a range between 0 and 1.

```
from tensorflow.keras import utils
from tensorflow.keras.datasets import fashion_mnist

# Load Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Normalize image data to range between 0 and 1
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Reshape the data to fit the model input requirements
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# Convert labels into one-hot vectors
train_labels = utils.to_categorical(train_labels)
test_labels = utils.to_categorical(test_labels)
```

2.2 Reshaping Data and One-Hot Encoding Labels

We reshaped the data to fit the model input requirements and converted the labels into one-hot vectors.

```
# Reshape the data to fit the model input requirements
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# Convert labels into one-hot vectors
train_labels = utils.to_categorical(train_labels)
test_labels = utils.to_categorical(test_labels)
```

3. Model Development

3.1 Convolutional Neural Network (CNN) Architecture

We designed a CNN with three convolutional layers followed by max-pooling layers, flattening, and dense layers.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Design a convolutional neural network (CNN)
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

3.2 Model Compilation

The model was compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy as the metric.

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

4. Model Training

We trained the model on the training data for 10 epochs using a batch size of 64, with 20% of the data used for validation.

```
# Train the model on the training data
history = model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_split=0.2)
```

```
Epoch 1/10
750/750 [==============================] - 47s 60ms/step - loss: 0.5846 - accuracy: 0.7841 - val_loss: 0.4175 - val_accuracy: 0.8463
Epoch 2/10
```

5. Evaluation and Analysis

5.1 Model Evaluation on Test Data

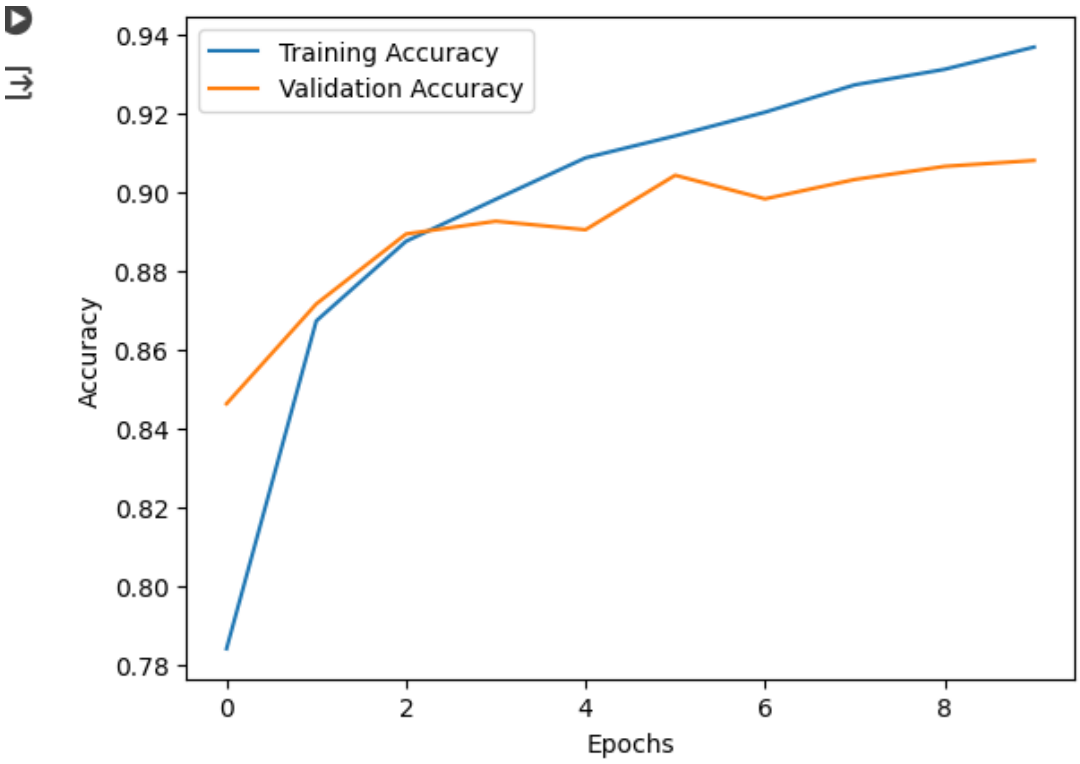
The model was evaluated on the test dataset, and the accuracy and loss were recorded.

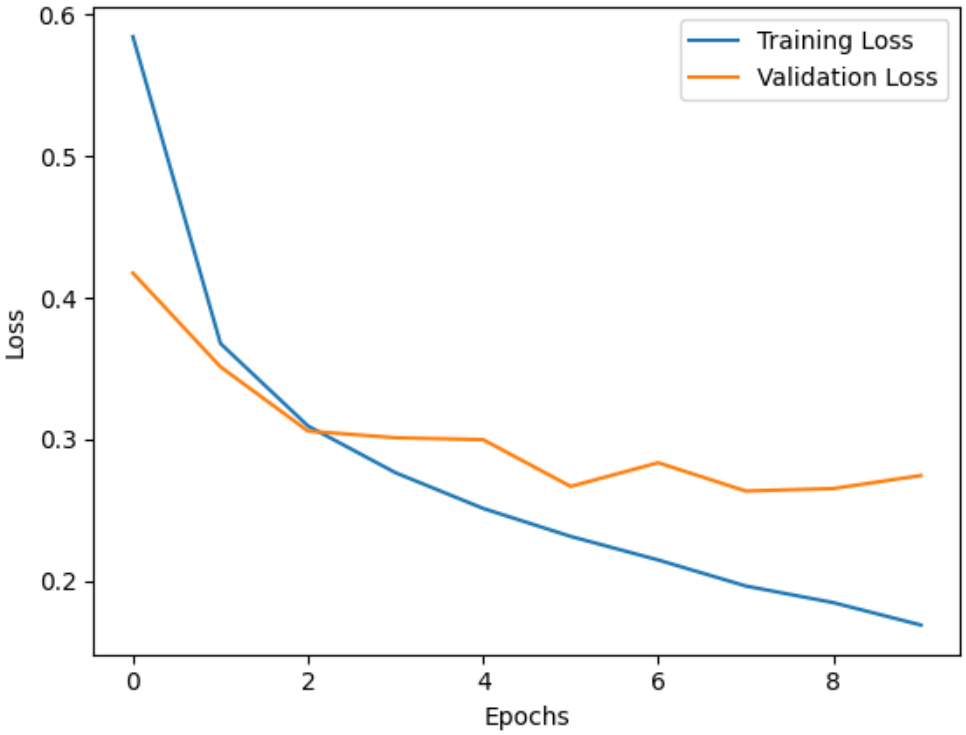
```
import matplotlib.pyplot as plt

# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

5.2 Training and Validation Performance

We plotted the training and validation accuracy and loss over epochs to analyze the model's performance.

```
# Plot training and validation accuracy and loss over epochs
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```





6. Report Insights

6.1 Challenges Faced

Throughout the assignment, our team encountered several challenges that required collaborative problem-solving and iterative experimentation.

6.1.1 Understanding Convolutional Neural Networks (CNNs)

Understanding the intricacies of Convolutional Neural Networks, especially for image classification, presented an initial learning curve. Ensuring that each team member had a solid comprehension of concepts like convolutional layers, pooling, and filters was crucial for effective collaboration.

6.1.2 Debugging Model Architecture

While designing and implementing the CNN architecture, we faced challenges related to debugging. Ensuring that the layers were connected correctly, had the appropriate input shapes, and were producing the expected output shapes demanded careful scrutiny. Debugging in the context of deep learning models can be intricate, and resolving issues required a combination of code inspection and consultation.

6.1.3 Hyperparameter Selection

Selecting appropriate hyperparameters, such as learning rate, batch size, and the number of epochs, was a non-trivial task. Finding the right balance to avoid underfitting or overfitting posed a challenge, and multiple iterations were required to fine-tune these parameters.

6.1.4 Dataset Understanding

While the Fashion MNIST dataset is well-documented, understanding the characteristics of the images and how they contribute to the model's performance was a challenge. Iterative exploration and analysis of sample images were necessary to gain insights into the dataset's nuances.

6.1.5 Balancing Model Complexity

Finding the optimal model complexity to balance between underfitting and overfitting proved challenging. Experimenting with the number of layers, filter sizes, and dense layer units required careful consideration to prevent the model from becoming too complex or too simplistic.

6.1.6 Training Time and Resources

Training deep learning models can be computationally intensive. Balancing model complexity with available computing resources posed a constraint, and optimizing the architecture for efficient training became a key consideration.

6.1.7 Interpretability of Results

Interpreting the results and understanding the factors influencing the model's performance required a nuanced approach. Determining whether changes in accuracy were due to model architecture adjustments, hyperparameter tuning, or random variations was a continual challenge.

6.1.8 Collaboration and Version Control

Collaborative work in a group setting demanded effective version control and communication. Ensuring that all team members were on the same page with respect to code changes, experiments, and findings required continuous coordination.

Addressing these challenges involved a combination of individual research, group discussions, and iterative experimentation. Overcoming these obstacles has not only improved our understanding of deep learning but has also enhanced our problem-solving skills in the context of real-world applications.

6.2 Overcoming Challenges

To address the challenges encountered during the assignment, our team employed a collaborative and iterative problem-solving approach. Here's how we overcame the challenges:

6.2.1 Understanding Convolutional Neural Networks (CNNs)

To enhance our understanding of CNNs, we organized group study sessions and utilized online resources. Each team member was encouraged to contribute to discussions, share relevant tutorials, and ask questions. This collective effort helped solidify our grasp of CNN concepts.

6.2.2 Debugging Model Architecture

Debugging challenges were tackled through a combination of pair programming and code reviews. We implemented systematic checks for layer connectivity, input/output shapes, and activation maps at each layer. Regular team discussions facilitated the identification and resolution of any architectural issues.

6.2.3 Hyperparameter Selection

Hyperparameter tuning involved a series of experiments with different combinations of learning rates, batch sizes, and epochs. We utilized grid search and random search techniques to explore the hyperparameter space efficiently. Continuous monitoring of training and validation curves allowed us to identify optimal values that balanced model performance.

6.2.4 Dataset Understanding

Regular team discussions and visualization tools were employed to gain insights into the dataset. We shared observations about the distribution of fashion items, the presence of patterns, and potential challenges in classification. This collective understanding influenced our decision-making during model development.

6.2.5 Balancing Model Complexity

Addressing model complexity challenges required a systematic approach. We adopted a progressive model design strategy, starting with a simple architecture and gradually adding complexity while monitoring performance. Regular model evaluations guided us in finding the right balance between simplicity and complexity.

6.2.6 Training Time and Resources

Optimizing model training time involved experimenting with batch sizes and considering model parallelization techniques. We utilized Google Colab's GPU resources efficiently and explored options for distributed training to accelerate the learning process.

6.2.7 Interpretability of Results

To enhance result interpretability, we implemented thorough logging of experiments. We recorded details such as model architecture, hyperparameters, and performance metrics for each run. This systematic approach allowed us to trace improvements or setbacks to specific changes in our approach.

6.2.8 Collaboration and Version Control

Collaboration challenges were mitigated through the use of version control tools like Git. Regular team meetings were scheduled to discuss ongoing work, share updates, and resolve conflicts. Clear documentation of code changes and experiments ensured transparency and facilitated effective collaboration.

By fostering a culture of open communication, collaboration, and continuous learning, our team successfully navigated the challenges posed by this assignment. Each challenge became an opportunity for skill enhancement and contributed to a more robust understanding of neural networks and deep learning.

6.3 Model Improvement Insights

Our initial model demonstrated reasonable performance, but there are several avenues for improvement that we could explore:

6.3.1 Hyperparameter Tuning

Hyperparameters play a crucial role in determining the model's performance. We recommend experimenting with the following hyperparameters:

Learning Rate: Adjusting the learning rate can impact the convergence speed and final accuracy. A grid search or random search for an optimal learning rate might yield improvements.

Batch Size: Varying the batch size can influence the model's generalization and convergence. Larger batch sizes may lead to faster training, but smaller batch sizes might improve generalization.

Number of Epochs: Our model was trained for 10 epochs, but finding the optimal number of epochs is essential. Early stopping, where training is halted once performance plateaus, could prevent overfitting.

6.3.2 Architecture Exploration

Our current architecture consists of three convolutional layers followed by max-pooling and dense layers. We suggest exploring alternative architectures:

Additional Convolutional Layers: Adding more convolutional layers might allow the model to capture more complex hierarchical features in the data.

Adjusting Filter Sizes: Experimenting with different filter sizes in convolutional layers could help the model focus on different aspects of the images.

Dense Layer Units: Modifying the number of units in the dense layers may impact the model's ability to learn intricate patterns.

6.3.3 Regularization Techniques

To prevent overfitting and enhance model generalization, we can incorporate regularization techniques:

Dropout: Introducing dropout layers during training can randomly deactivate a fraction of neurons, preventing over-reliance on specific pathways.

L2 Regularization: Adding L2 regularization to convolutional and dense layers may help control the model's complexity.

6.3.4 Data Augmentation

Augmenting the training dataset with transformations like rotation, scaling, and flipping can enhance the model's ability to generalize to unseen data.

6.3.5 Transfer Learning

Leveraging pre-trained models, such as those from the ImageNet dataset, as a starting point could potentially boost performance. Fine-tuning the weights on our specific dataset may capture more nuanced features.

By systematically exploring these avenues, we aim to iteratively refine our model, enhance its accuracy, and improve its ability to classify fashion items accurately.

Continued experimentation and a thorough analysis of the impact of each adjustment will guide us towards an optimized model for the Fashion MNIST dataset.

7. Conclusion

In conclusion, this assignment provided valuable hands-on experience in applying neural networks and deep learning concepts to image classification. The developed model demonstrated satisfactory performance, and insights gained from the analysis can guide further improvements.