Assignment: Exploring Neural Networks with MNIST and Color MNIST

Overview

In this assignment, you will explore and implement various neural network architectures, including feedforward neural networks (FNNs) and convolutional neural networks (CNNs). You will also deep dive into techniques like hyperparameter tuning and transfer learning. This assignment will help you better understand how these methods can be applied to image classification tasks using the MNIST dataset (grayscale) and Color MNIST dataset (from Kaggle).

You are required to submit one or more Python notebooks that include code, visualizations, and comments for each task. Ensure that your code is well-documented, and all results are clearly explained.

Datasets

- **MNIST (Grayscale)**: The classic handwritten digit dataset with 60,000 training images and 10,000 test images, where each image is 28x28 pixels.
- **Color MNIST**: A colorized version of the MNIST dataset available on Kaggle. It contains the same digits as MNIST but in RGB format (3 channels).

Assignment Breakdown

1. Feedforward Neural Network on MNIST

- **Task**: Build a simple feedforward neural network (FNN) to classify digits from the grayscale MNIST dataset.
    - Use fully connected layers (dense layers) with activation functions like ReLU and softmax.
    - The network should include at least two or more hidden layers.
    - Train the model and evaluate its performance on the test set.
    - Experiment with techniques to prevent overfitting (we have seen a couple in class), e.g., regularization techniques like dropout or early stopping techniques.

2. Convolutional Neural Network on grayscale MNIST

- **Task**: Build a convolutional neural network (CNN) to classify digits from the grayscale MNIST dataset.
    - Use at least two convolutional layers, followed by max-pooling layers.
    - Include fully connected layers before the output layer.
    - Experiment with techniques to prevent overfitting (we have seen a couple in class), e.g., regularization techniques like dropout or early stopping techniques.
    - Compare your results to those in the previous section using a feed forward NN, which approach gave you fastest results? And with which one did you achieve better accuracy?

3. Implement a Residual Block in TensorFlow

- **Task**: Implement a basic residual block in TensorFlow and incorporate it into the best CNN model that you got in the previous section, for grayscale MNIST.
    - A residual block is a fundamental building block of **ResNet**, a popular deep neural network architecture that helps alleviate the vanishing gradient problem in deep networks.
    - The residual block should contain two convolutional layers and a skip connection that adds the input of the block to the output of the convolutional layers.

Hint: You can implement the residual block using the following structure:

```
def residual_block(x):
    shortcut = x  # Save input for the skip connection
    x = Conv2D(filters, kernel_size, padding='same', activation='relu')(x)
    x = Conv2D(filters, kernel_size, padding='same')(x)
    x = Add()([x, shortcut])  # Skip connection
    x = ReLU()(x)
    return x
```

Residual networks (ResNets) have been instrumental in improving deep learning models' ability to generalize, especially in very deep networks. This exercise will give you hands-on experience with how residual blocks are implemented.

4. Hyperparameter Tuning

- **Task**: Tune the hyperparameters of your CNN for optimal performance on the grayscale MNIST dataset.
    - Experiment with different values for learning rate, batch size, number of layers, dropout rates, etc.
    - Use techniques such as grid search, random search, or manual tuning.
    - Save your best model (you can use the documentation here)

Hyperparameter tuning is essential in ensuring that your model achieves the best possible performance. Experimenting with different values will help you understand the sensitivity of your models to various hyperparameters.

5. Transfer Learning

What is Transfer Learning? Transfer learning involves taking a model that has already been trained on one task (usually a large dataset like ImageNet) and adapting it to a different, but related task. This is especially useful when you do not have enough data to train a model from scratch. Instead of starting with random weights, transfer learning allows you to start with weights that have already been fine-tuned on a large amount of data.

Popular Architectures for Transfer Learning:

- **VGG16**: A popular CNN architecture known for its simplicity. It consists of 16 layers, including convolutional layers followed by max-pooling and dense layers.
- **ResNet (Residual Networks)**: These networks solve the issue of vanishing gradients by using skip connections (or residual connections) that allow the network to learn identity mappings. This is particularly useful in very deep networks.

For this task, you will load a pretrained model (either it is yours, VGG16 or ResNet), freeze some of the layers, and train the remaining layers on the Color MNIST dataset.

- **Task**: Use a pretrained CNN model (you can use your model from exercise 4, or you can look for a pre-trained VGG16 or ResNet from Tensorflow) to classify digits from the **Color MNIST** dataset.
    - Load your pre-trained model
    - Adapt the model if needed (input/output)
    - Fine-tune the model by freezing the lower layers (the earlier layers in the network, don't freeze the input layer if you had to change it) and retraining the top layers.

General Submission Guidelines

1. **Submission Format**: Submit a Jupyter notebook (.ipynb) file that includes all the code, explanations, and visualizations. Code should be structured and organized according to best practices.
2. **Documentation**: Ensure that all sections of your notebook are well-documented. Code should be properly discussed and commented.
3. **Plots and Visualizations**: All graphs, tables, and confusion matrices must be clearly labelled and explained.
4. For this assignment, please note that we will be using software tools to check for originality and ensure that all submitted work is free of plagiarism. Be sure that all code and content provided **is your own or properly cited** if external sources were referenced.