CMPSC 448 Final Project

Eric Chen

Task, Data, and Preprocessing

For this project, I decided that I wanted to try an image classification problem: something that I have not done in this course yet. I decided to use the CIFAR-10 dataset, which is a popular dataset used to train image models, consisting of 60,000 color images across 10 different classes.

Using this dataset, I will be comparing the accuracy of image classification in a convolutional neural network with a recurrent neural network. I also decided to use the Tensorflow package from Python to acquire the functionality for these algorithms.

To load the dataset, I simply called the CIFAR-10 dataset included from the Tensorflow package. The variables *x_train* and *y_train* contain the training images and labels respectively for the training set, while the variables *x_test* and *y_test* contain the training images and labels for the test set.

To preprocess the data, for both CNN and RNN methods, I divided *x_train and x_test* by 255. Dividing these variables by 255 normalizes the pixel values of the images, thus, the values will be between zero and one.

For the CNN model, I added another step to set the labels to categorical variables. This normalizes the labels and converts them to binary vectors of length 10.

Implementation and Architecture

To accomplish the task at hand, I tried to optimize both models as best as possible.

The CNN model is comprised of two convolutional layers with max-pooling, followed by a flatten layer and two dense layers. The convolutional layers learn features from the input images, and the dense layers act as classifiers.

The RNN model has a single layer, which utilizes long short-term memory (LSTM) cells. This layer is followed by a dropout layer, which is a regularization technique used to prevent overfitting. The recurrent layers are also followed by a flatten layer and two dense layers.

Both models are compiled with the Adam optimizer and categorical crossentropy loss, which are then trained on the CIFAR-10 data that I preprocessed earlier.

Training Details

Below is an overview of each model's architecture, parameters, and training procedures.

CNN Training Details:

- **Architecture**:
    - Convolutional Layer 1: 32 filters with a kernel size of (3, 3) and ReLU activation.
    - MaxPooling Layer 1: Pooling with a (2, 2) window.
    - Convolutional Layer 2: 64 filters with a kernel size of (3, 3) and ReLU activation.
    - MaxPooling Layer 2: Pooling with a (2, 2) window.
    - Flatten Layer: Flattens the output for the fully connected layers.
    - Dense Layer 1: 128 units with ReLU activation.
    - Dense Layer 2: 10 units with softmax activation for classification.
- **Training Parameters**:
    - Optimizer: Adam optimizer.
    - Loss Function: Categorical Cross-entropy.
    - Batch Size: 32.
    - Epochs: 10.
- **Training Procedure**:
    - The CNN model is trained for 10 epochs on the CIFAR-10 training data.
    - During each epoch, the model is updated using batches of size 32.
    - The Adam optimizer minimizes the categorical crossentropy loss.
    - Training progress and validation accuracy are monitored.

RNN Training Details:

- **Architecture**:
    - LSTM Layer: 128 units.
    - Dropout Layer: 0.2 dropout rate.
    - Flatten Layer: Flattens the output for the fully connected layers.
    - Dense Layer 1: 128 units with ReLU activation.
    - Dense Layer 2: 10 units with softmax activation for classification.
- **Training Parameters**:

    - Optimizer: Adam optimizer.
    - Loss Function: Sparse Categorical Cross-entropy.
    - Batch Size: 32.
    - Epochs: 10.
- **Training Procedure**:
    - The RNN model is trained for 10 epochs on the CIFAR-10 training data.
    - During each epoch, the model is updated using batches of size 32.
    - The Adam optimizer minimizes the categorical crossentropy loss.
    - Training progress and validation accuracy are monitored.

Results, Observations, and Conclusions

After running both neural networks multiple times, the maximum accuracy I found for CNN was 87.25%. The maximum accuracy that I found for RNN was 59.69%.

I also noticed that Tensorflow was able to measure each model's time efficiency accurately. For my CNN model, it took 5 milliseconds to execute a single step, leading to 8 seconds per epoch. The overall time it took to run was 80 seconds. For my RNN model, it took 13 milliseconds to execute a single step, leading to 21 seconds per epoch. As a result, the overall time the RNN model took to run was 210 seconds.

Overall, I discovered that the CNN model performed much more efficiently and accurately compared to the RNN model. This was not surprising, as RNN models are more suited for sequence data like NLP. CNN models are specifically designed for image classification, and they should outperform RNN models at this task.

Challenges and Obstacles

The hardest part of this assignment was fitting and optimizing an RNN model to the task. Initially, I wanted to make an RNN model that was as effective as a CNN model. Of course, since RNN models are not optimal for image classification, it would have been difficult to create an RNN model that was as good as a CNN model for this task. As a result, finding hyperparameters that could improve accuracy of the RNN model also proved to be difficult. I ended up creating a simpler RNN model than I wanted to.

I also had difficulty selecting a package to use for this assignment. Originally, I wanted to use PyTorch to create the models, but I ran into a compatibility issue when trying to run the code on my graphics card. I ended up creating the models on Tensorflow instead, and I had no issue running the code afterwards.