ID:80585887
CS 5363
Dr. Olac Fuentes

I. Problem Description

For Lab 5 our task was to implement different neural network architectures to classify the samples included from the CIFAR and Fashion MNIST datasets. In addition, we were suppose to attempt to find the best modifications in order to maximize the accuracy of the predictions. For example, learning rate, batch size, batch regularization, dropout, etc. In order to do this we are given some code in the form of a Google Colab notebook and that acted as our starter code.

II. Algorithms Implemented

Problem 1

For this problem (CIFAR 10) I used a custom made architecture inspired in the implementation provided by Dr. Fuentes. As change, I modified the dropout applied to the layers and I also included the some layers to increase the complexity of the representations learned at earlier layers of the network. In addition, I decided to keep the mirrored-image augmentation provided in the implementation by Dr. Fuentes. Following is the best architecture in terms of accuracy for validation and training:

```
def cnn_model(input_shape=(28,28,1)):
    model = tf.keras.models.Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.3))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.5))
    model.add(Conv2D(96, (3, 3), activation='relu'))
    model.add(Conv2D(96, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    return model
```

Problem 2

For this problem (Fashion MNIST) I decided to work, once again, with the augmentations provided but did modify the architecture used for this problem. I used as a template the architecture used in the last problem/provided as starter code and eventually found the following architecture which provides the best results in terms of accuracy:

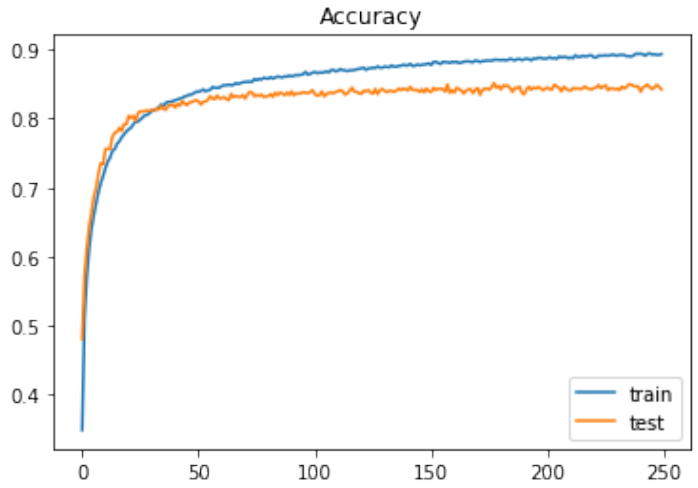
```
def cnn_model(input_shape=(28,28,1)):
    model = tf.keras.models.Sequential()
    model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.3))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Dropout(0.5))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    return model
```

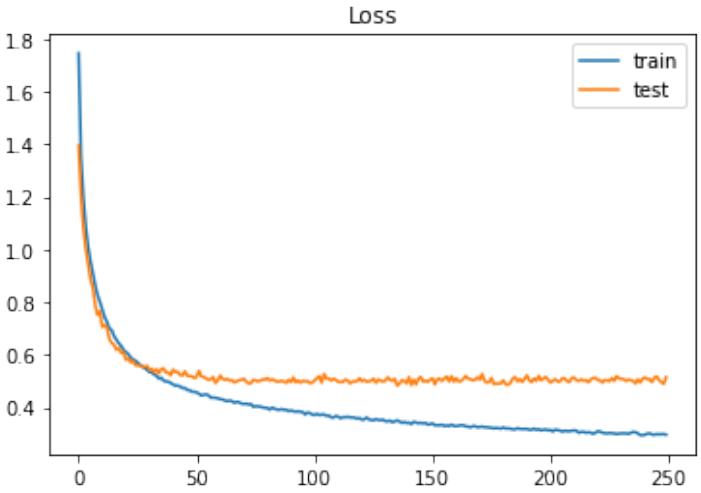
III. Experimental Results

Problem 1:

- Results after training for 250 epochs

  Lowest training loss = 0.294459, in epoch 240
  Lowest validation loss = 0.482680, in epoch 136
  Highest training accuracy = 0.894650, in epoch 245
  Highest validation accuracy = 0.851700, in epoch 178

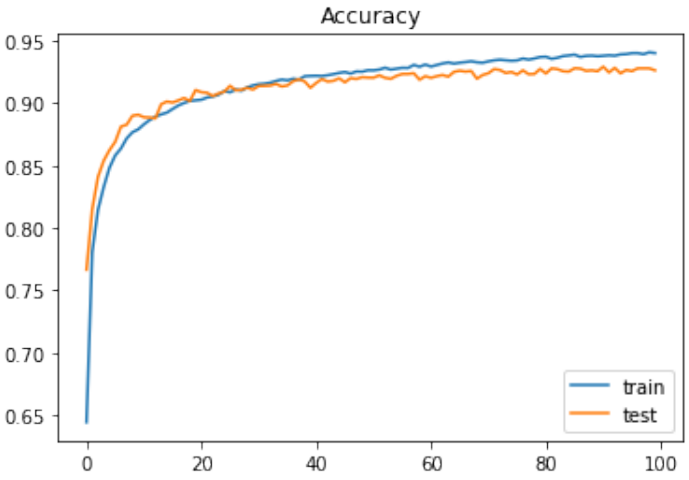


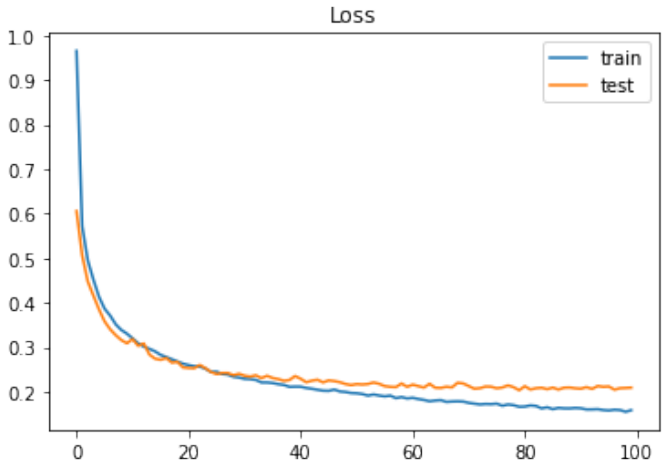


Problem 2

- Results after training for 100 epochs

Lowest training loss = 0.155296, in epoch 99
Lowest validation loss = 0.203618, in epoch 80
Highest training accuracy = 0.940850, in epoch 99
Highest validation accuracy = 0.929300, in epoch 91





IV. Discussion of Results

In running experiments to “solve” these problems I tried multiple other architectures. For example, in the case of CIFAR 10 I ran experiments with the following architectures:

Architecture 1:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_22 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 14, 14, 32)        0
_________________________________________________________________
dropout_16 (Dropout)         (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
dropout_17 (Dropout)         (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 4, 4, 96)          55392
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 2, 2, 96)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 384)               0
_________________________________________________________________
dense_8 (Dense)              (None, 64)                24640
_________________________________________________________________
dense_9 (Dense)              (None, 10)                650
=================================================================
```

Results:

Lowest training loss = 0.328093, in epoch 250
Lowest validation loss = 0.474922, in epoch 190
Highest training accuracy = 0.882720, in epoch 249
Highest validation accuracy = 0.846700, in epoch 216

Architecture 2:

```
def cnn_model(input_shape=(28,28,1)):
  model = tf.keras.models.Sequential()
  model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
  model.add(Dropout(0.2))
  model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(0.3))
  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
  model.add(Dropout(0.5))
  model.add(Conv2D(96, (3, 3), activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
  model.add(Flatten())
  model.add(Dense(64, activation='relu'))
  model.add(Dense(10, activation='softmax'))
  return model
```

Results:

Lowest training loss = 0.385936, in epoch 494
Lowest validation loss = 0.455321, in epoch 436
Highest training accuracy = 0.862610, in epoch 443
Highest validation accuracy = 0.846800, in epoch 344

This is not an all-inclusive list but rather I am just including those experiments that had successful runs. I decided to report as main results the network that had the absolute best score for both training and testing accuracies. I discover with my experimentations that it was fundamental to have a harsh dropout between the last two dropouts, my intuition behind that is that it is fundamental to force the network to learn more fine grained representations of the classes in the CIFAR 10 dataset. In addition, I discover it was fundamental to include a secondary convolutional layer after each of the start code's layers. My intuition behind that is that the representations learned from the dataset are fundamental to be "checked" or reaffirmed before a maximum pooling is applied, this indeed provided around 5% increase in the accuracy of the validation set compared to other experimentations that only made use of the "harsh" dropout that I had mentioned.

In terms of the Fashion MNIST experimentations I feel It was important to increase the number of features that are produced in the last convolutional layer with 128 filters since there is a good chance that the filters applied at that last layer did not focus enough on the edges, changes, etc. as seen in the samples of the training data. The results of my experimentation seem to be around the state of the art which is around 96%. An accuracy of 92% percent is a good result but I have concluded that most likely the dataset is not a hard “problem” for a CNN implementation, but rather just a “toy” dataset created to benchmark deep learning implementations vs. what a vanilla CNN would look like.

V. Appendix

Best for MNIST:

```
def cnn_model(input_shape=(28,28,1)):
        model = tf.keras.models.Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.3))
        model.add(Conv2D(64, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
        model.add(Dropout(0.5))
        model.add(Conv2D(128, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
        model.add(Flatten())
        model.add(Dense(64, activation='relu'))
        model.add(Dense(10, activation='softmax'))
        return model
```

Best for CIFAR:

```
  def cnn_model(input_shape=(28,28,1)):
      model = tf.keras.models.Sequential()
      model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
      model.add(Dropout(0.2))
      model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      model.add(Dropout(0.3))
      model.add(Conv2D(64, (3, 3), activation='relu'))
      model.add(Conv2D(64, (3, 3), activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
      model.add(Dropout(0.5))
      model.add(Conv2D(96, (3, 3), activation='relu'))
      model.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
      model.add(Flatten())
      model.add(Dense(64, activation='relu'))
      model.add(Dense(10, activation='softmax'))
      return model
```

VI. Conclusion

I believe this lab was a nice introduction to Deep Learning, although probably starting without augmentations could have done the problem a little more challenging. I believe other augmentations of the data like random zooms, cutouts could have helped to train the data but I didn't have the time to implement. In addition, I have concluded that having the idea of "reaffirmation" of the learned patters in "earlier" layers of the networks is quite good at least for the CIFAR 10, further investigation would be interesting to read/see.

In relation to improving the results of the CIFAR 10 and MNIST datasets I believe that CIFAR10 could use some augmentations but I believe there are more implementations that have succeeded in achieving 100% of accuracy in the dataset so the problem is already "solved". Similarly, Fashion MNIST is also a solved problem, that is, there are networks that have gotten close to 100% of accuracy in MNIST. Nevertheless, for a small assignment like this I got to develop network that could have been state of the art networks in 2012 and 2020 respectively.

Statement of Academic Honesty:

“I certify that this project is entirely my own work. I
wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.”