CS 577 - F22 - Assignment 2 Programming Portion

Due date: 10/4/2022

Anas Puthawala - A20416308

Professor Gady Agam

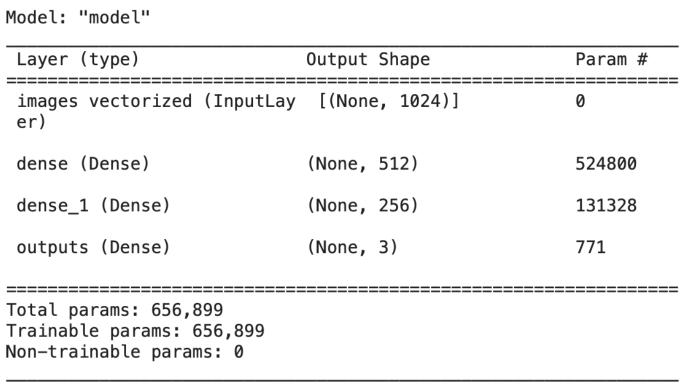
Precursor - Data Preparation

The data used in the programming questions was taken from the previous assignment. In summary, it is from the CIFAR-10 dataset, and was sampled to take only images from within **three** classes from the dataset (dogs, cars, and airplanes). Vectorization of the images was performed in the following manner:

1. Collapsing the color channel by taking the mean of the three channels
2. Dividing by 255 to normalize
3. Reshaping the array so that it goes from 32 x 32 matrix to a single vector of 32*32 = 1024 elements.

Problem 1

The purpose for problem 1 was to implement gradient descent by using tf.GradientTape() as opposed to simply calling the .fit() method to train the network. The current network was modeled using the functional method of building a model. The model architecture is as follows:



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 images vectorized (InputLay  [(None, 1024)]           0
 er)

 dense (Dense)               (None, 512)               524800

 dense_1 (Dense)             (None, 256)               131328

 outputs (Dense)             (None, 3)                 771

=================================================================
Total params: 656,899
Trainable params: 656,899
Non-trainable params: 0
_________________________________________________________________
```

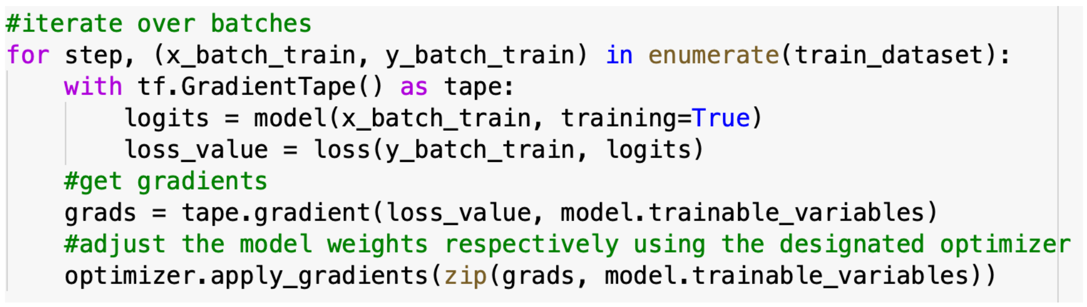
Figure 1.0 - Network Summary

The dense layers ('dense' and 'dense_1') used sigmoid activation whereas the output consisted of the softmax activation.

The optimizer used was Adam with a learning rate of 1e-3 and categorical cross entropy loss was used with the parameters 'from_logits' set to True since we're going to be manually building the training loop and working with logits.

Tensorflow datasets were used to effectively compile the training and validation datasets into two distinct datasets. And the training and validation metrics used to assess the model were 'CategoricalAccuracy'. In addition, there were four empty vectors I instantiated with the purpose of collecting the metric results as the model trains so we can visualize the training performance and see if there is any over-fitting, etc.

The process for training the model is as follows:



```python
#iterate over batches
for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):
    with tf.GradientTape() as tape:
        logits = model(x_batch_train, training=True)
        loss_value = loss(y_batch_train, logits)
    #get gradients
    grads = tape.gradient(loss_value, model.trainable_variables)
    #adjust the model weights respectively using the designated optimizer
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

1. Instantiate the gradient tape in order to take the gradients to update the parameters easily
2. Push the x_batch_train through the network and collect the logits
3. Compute a loss value using a loss function of yourself, compare the y_batch_train to the logits you just received
4. Use tape.gradient with the loss value and the model's trainable variables to compute the gradients
5. Using an optimizer of your choice, apply the gradients to update the weights of the model
6. Update metrics
7. Repeat until desired performance is achieved

The models' training accuracy and loss were tracked in the empty arrays instantiated at the beginning of the training loop as-well as the training & validation loss in order to plot them to visualize the model's training.

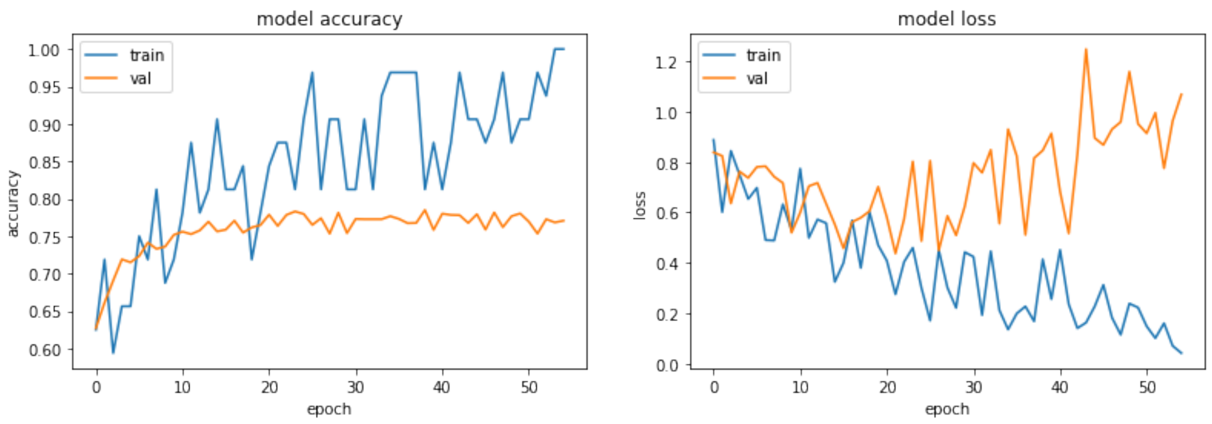


Figure 2.0 - Loss and Accuracy curves

Few things to note:

- The model took longer to train since it wasn't optimized as well as the .fit() command is. There are approaches to speed it up by casting @tf.function which enables Tensorflow to optimize things using a computational graph
- The model learns fairly quickly, starting off at accuracies of around 65% and improving close to 75%
- The model is seem to over-fitting after around the 20 - 25 epoch range as the accuracy begins decreasing and the loss begins to increase, regularization and the like can be applied in order to prevent regularization and boost accuracies.
- Ultimately, a convolutional network is the optimal method to train a model with feature set as this (which are images).

Problem 2

The purpose of this problem was to train an entire neural network by scratch without using libraries like Tensorflow or Keras or Pytorch. The goal is to implement the necessary classes and functions in order to train the network from scratch. The network to be implemented is a three layer network for three class classification. The loss function is categorical cross entropy, relu activation for the first hidden layer, sigmoid for the second and softmax for the output. Unfortunately due to time constraints and other commitments I was unable to complete the implementation of back-propagation as I was facing many difficulties with my implementation of the Layer and Node class. I tried many alternatives but due to time I was unable to fully complete this. Forward propagation however, was indeed working. I also implemented an evaluation function which was simply the accuracy. I.e. it would count the number of correct predictions when comparing the prediction labels and the actual truth labels in the test set.

In the implementation I have developed two classes:

1. Layer Class

The layer class contains weights for the layer the forward and backward prop (backward is incomplete unfortunately) functions as-well as a few activation functions.

2. Node Class

The node class is (as it's named) contains the node properties including the weight, bias, rules for forward prop and back prop.
There are a few more functions defined that aid in the forward prop process such as manually one hot encoding the labels. In addition, I began the process of defining functions that would aid in the backprop process such as the derivative of relu (an activation function used) and the 'gradient_calc' function which goes through and calculates the gradients.