# Homework 5

Emily Teng, Niaoniao Ma, Susannah Schulze, Xiaochen Guo

## Problem 1: Softmax Properties

### Question 1

The softmax function is invariant to constant offsets because the constant term disappears when the equation is simplified (based on exponent rules and properties).

For instance, if ***a*** is a vector if 4 elements *a1, a2, a3*, and *a4*, the simplification process will look like this:

$$\text{Softmax}(a+c) = \frac{e^{a+c}}{e^{a1+c} + e^{a2+c} + e^{a3+c} + e^{a4+c}}$$

$$= \frac{e^{a} \times e^{c}}{(e^{a1} \times e^{c}) + (e^{a2} \times e^{c}) + (e^{a3} \times e^{c}) + (e^{a4} \times e^{c})}$$

$$= \frac{e^{a} \times \cancel{e^{c}}}{\cancel{e^{c}} \times (e^{a1} + e^{a2} + e^{a3} + e^{a4})}$$

$$= \frac{e^{a}}{e^{a1} + e^{a2} + e^{a3} + e^{a4}}$$

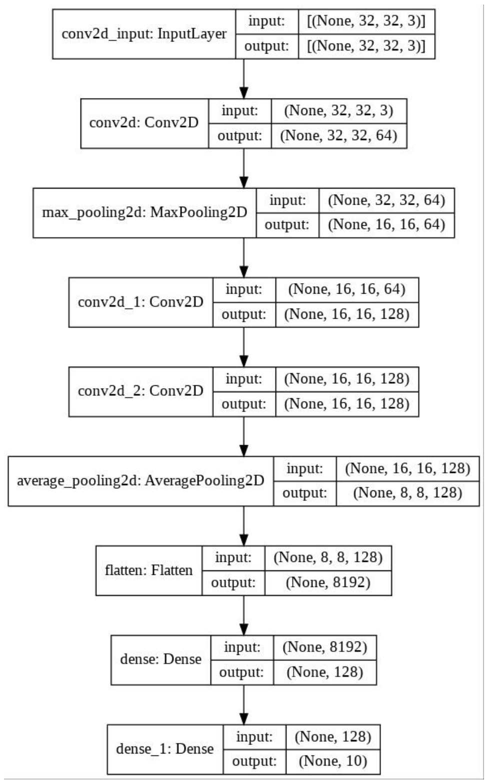
### Question 2

In practice, we can use this property to prevent the product of data and weights (in a neural network) from getting too big. When a constant is elevated to the exponent, the product could get so large that the result would not be computed. Knowing that the softmax function is invariant to constant offsets, we can address this potential problem by subtracting the constants from the product.

## Problem 2: CNN with CIFAR Data

### Question 1

## Question 2

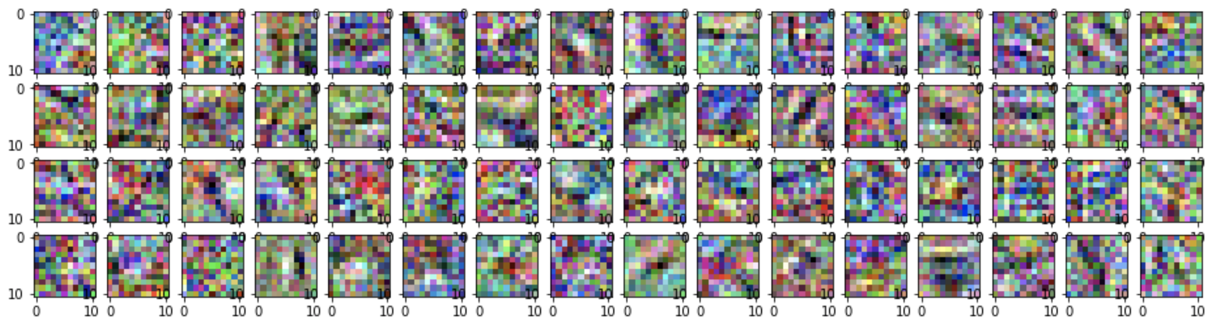

conv2d_input: InputLayer
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 64)
max_pooling2d: MaxPooling2D
input: (None, 32, 32, 64)
output: (None, 16, 16, 64)
conv2d_1: Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 128)
conv2d_2: Conv2D
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
average_pooling2d: AveragePooling2D
input: (None, 16, 16, 128)
output: (None, 8, 8, 128)
flatten: Flatten
input: (None, 8, 8, 128)
output: (None, 8192)
dense: Dense
input: (None, 8192)
output: (None, 128)
dense_1: Dense
input: (None, 128)
output: (None, 10)

We used the Sequential API() to create the model. The convolutional layers will extract the features of the image. We begin by using 64 filters of size 11x11, but with subsequent layers decrease the filter size and increase the number of filters so that we can capture more details of the features. We used the He weight initialization, which works well with the Relu activation function. The max pooling layer will reduce the dimensionality of the image and only retain important parameters. Our max pooling layer will take the maximum parameter value, while the average pooling layer will take the average of the parameter values. After adding the convolutional and pooling layers, a flatten layer will convert our output into a one-dimensional array that can be fed to the fully connected dense layer. Our output layer uses the softmax activation function to predict the probability that image belongs to each of the 10 classes.

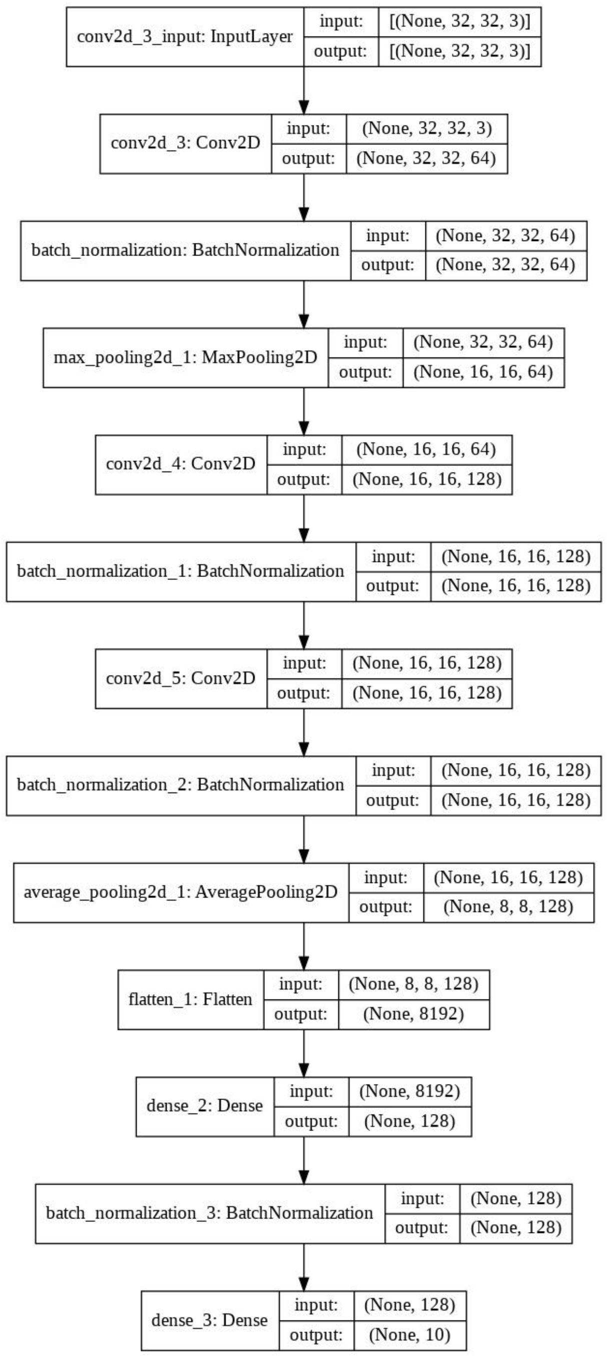
We use the sparse categorical cross-entropy loss function because our images are separated into 10 distinct classes. If one image could belong to more than one class, we would use the categorical cross-entropy loss function.

Due to the time needed to train such a large model, we started out with only training it on 15 epochs. This gave us a test accuracy of 66%. In the future, we would increase the number of epochs and train the model overnight to get better results.

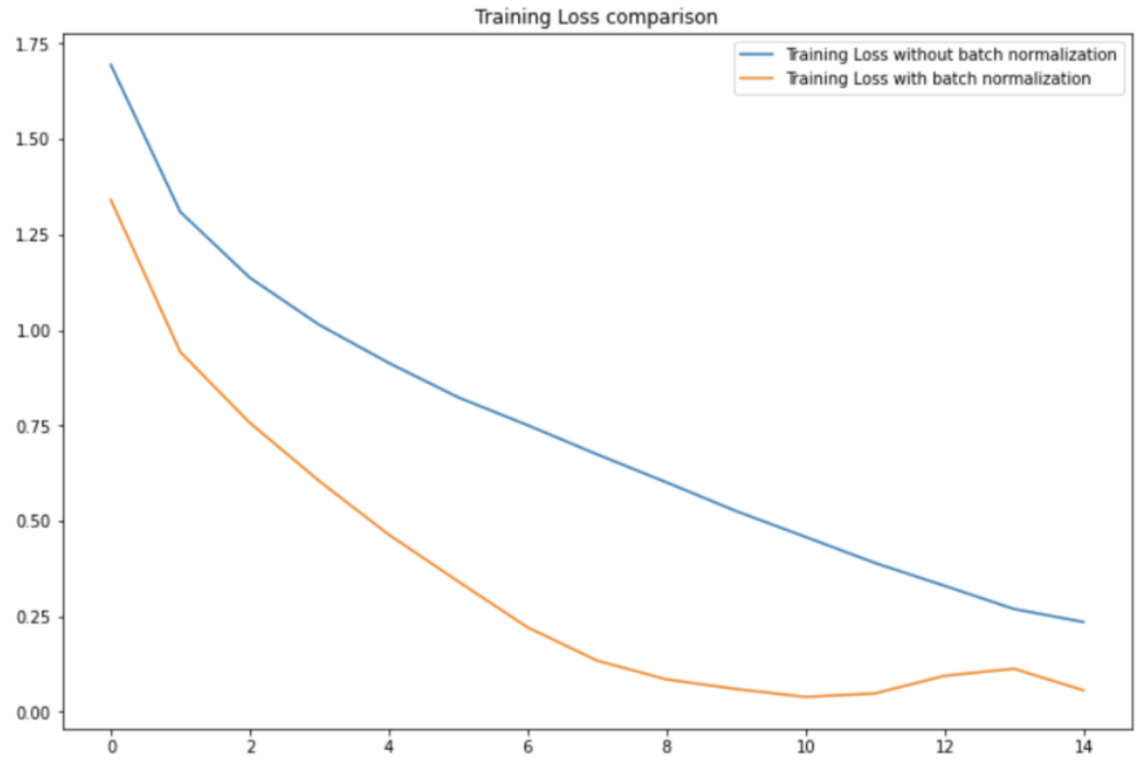
## Question 3



## Question 4

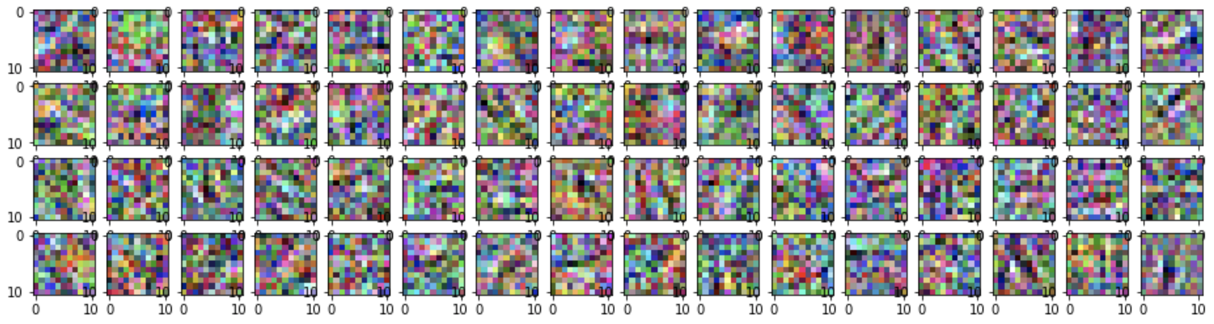

conv2d_3_input: InputLayer
input: [(None, 32, 32, 3)]
output: [(None, 32, 32, 3)]
conv2d_3: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 64)
batch_normalization: BatchNormalization
input: (None, 32, 32, 64)
output: (None, 32, 32, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 32, 32, 64)
output: (None, 16, 16, 64)
conv2d_4: Conv2D
input: (None, 16, 16, 64)
output: (None, 16, 16, 128)
batch_normalization_1: BatchNormalization
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
conv2d_5: Conv2D
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
batch_normalization_2: BatchNormalization
input: (None, 16, 16, 128)
output: (None, 16, 16, 128)
average_pooling2d_1: AveragePooling2D
input: (None, 16, 16, 128)
output: (None, 8, 8, 128)
flatten_1: Flatten
input: (None, 8, 8, 128)
output: (None, 8192)
dense_2: Dense
input: (None, 8192)
output: (None, 128)
batch_normalization_3: BatchNormalization
input: (None, 128)
output: (None, 128)
dense_3: Dense
input: (None, 128)
output: (None, 10)

```
Text(0.5, 1.0, 'Training Loss comparison')
```



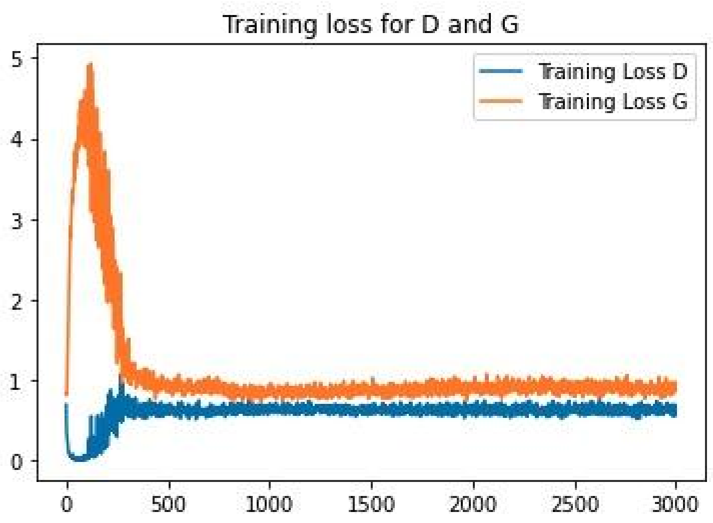
Above is how we would set up the architecture for the model with Batch Normalization and the Comparison of the training loss with and without batch normalization as a function of epochs. The test accuracy of the model with batch normalization is 69.7%, which is greater than the 65%. What's more, the training loss of the new model along the epochs is always less than the training model without batch normalization. And the final test error is 0.3033.

One of the benefits of Batch Normalization is that it can potentially speed up the training time of a model. We hypothesize that the Batch Normalization will slightly improve our model performance through a small regularization effect.

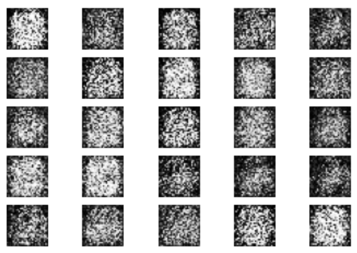
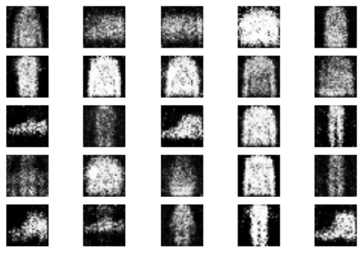
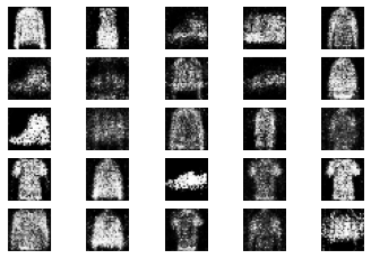


## Problem 3: GAN with Fashion-MINST Data

**Question 1**

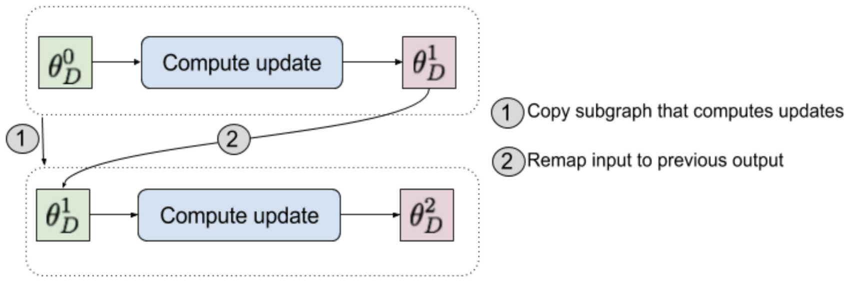


3 stages of samples are(1 2 3 from left to right) :

1  2  3 

**Question 2**

We have tried developing the unrolled GAN and plotting the results, which were not as expected. In our initial attempt, we are trying to code for graph rewriting, which involves copying the graph and replacing the initial weights with the last iteration's weights.



Source: https://notebook.community/poolio/unrolled_gan/Unrolled%20GAN%20demo

Theoretically, the unrolled GAN would reduce the mode collapse problem because the unrolled GAN approach introduces a generator loss function that incorporates both the current discriminator's classifications and future discriminator outputs, thereby preventing the over-optimizing of a single discriminator.