ELEC 475 Lab 3

Lucas Coster

20223016

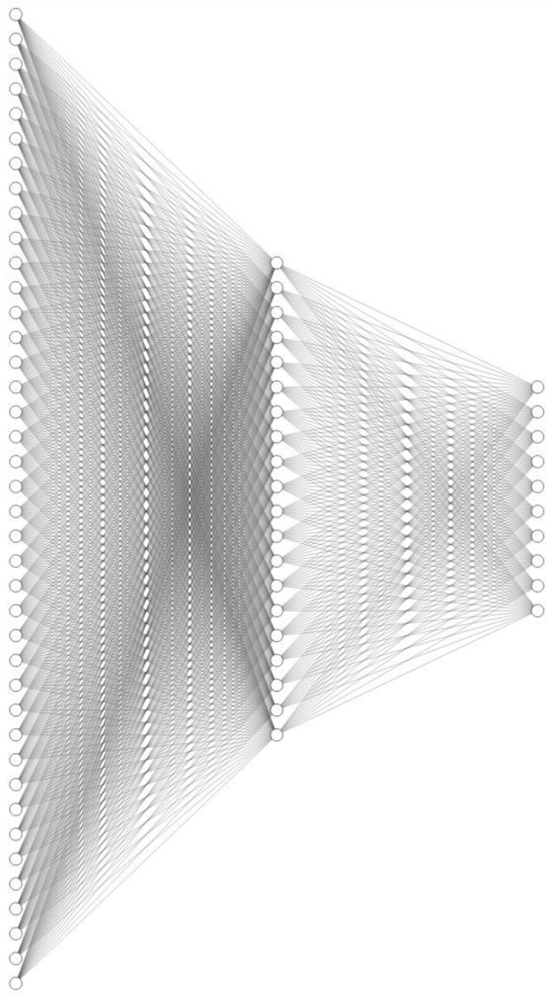
November 6th, 2023

Section 1: Vanilla Architecture

The vanilla architecture was designed to be a simple implementation of a classification model which would be able to detect and classify images from the CIFAR dataset. It used two main parts, the feature extractor which used VGG architecture and had predetermined weights along with a frontend classifier which was to be designed.

The simple architecture designed consisted of three main features, linear layers, RELU activation functions and dropout layers which took in the extraction features and classified them into one of 10 classes for CIFAR10 and one of 100 classes for CIFAR100.

We chose to use linear layers because we thought this would be the simplest approach to the problem and a fast solution train. The RELU activation layers are a common part of neural networks and were added to improve performance throughout the model. The dropout layers were the final layers added and should probably have been added in the modified architecture. These layers produced the greatest change in model accuracy and were included due to the fact they had been recently discussed in class and were simple to implement. The model outputs the number of classes being predicted which provides a prediction as to what the input image was.



On the left you can see a simple diagram of the fully connected architecture described. The diagram is not exact as the number of features in the initial layers is much larger, but it shows the process of cutting the features in half layer by layer until the final layer is equal to the number of possible classes.

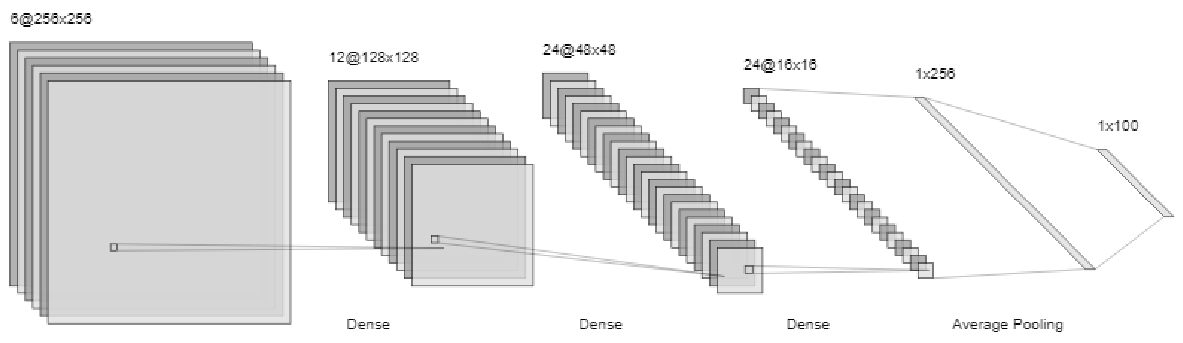
The diagram has no representation for dropout but all that would look like is losing half of the weights between layers each loop through the model.

Section 2: Mod Architecture

For the modified architecture an implementation of a dense residual network was done which involved 4 main improvements. The addition of dense connectivity, residual connections, custom growth and layer configuration and adaptive average pooling. The motivation for this specific model was pulled from a collection of model implementations learned through class material. The most influential model was ResNet which provided inspiration for the residual layers which are implemented throughout the model along with several other decisions made.

The encoder stayed the same still implementing the VGG architecture provided with the weights predetermined.

The dense connectivity refers to each layer receiving inputs from all the previous layers promoting feature use and improving training and generalization. Residual connections or skip connections were added to mitigate gradient loss and improve network depth. This model improvement involved providing a shortcut around one layer within the model to allow it to make residual predictions without providing the desired output directly. The custom growth rates within each layer allow the model to learn the number of weights which best suit the specific task at hand and be more adaptable. The final addition to the modified architecture was adaptive average pooling which ensured the model could accept a few input images and produce a fixed output image no matter what. The entire model was implemented using pytorch features and included several classes to easily create the multitude of improvements described.



In the diagram you can see an overview of the model architecture including the number of dense layers, the average pooling in the fully connected layer and the transitional blocks which change the convolutional dimensions. As shown in the diagram the final output is a 100 output long fully connected layer which provides the predicted class for the input image.

Section 3: Experiments

After both the vanilla and modified models had been created training and testing were required to improve and track improvement of the models. The train function included several hyper parameters to improve performance and allow for small tweaks to the training itself. The hyperparameters include the learning rate, the learning rate decay, the epoch number, batch size, and the optimizer which were all chosen after a multitude of tests based on the best model created.

The learning rate hyper parameter was chosen to be 0.001 due to the loss accuracy it provided and the speed at which the weights were able to find the most effective model. The optimizer decided upon was SGD, the reason we chose this over the Adam optimizer is it showed better results when testing on the CIFAR10 dataset. Within the SGD optimizer the learning decay value of 5e-4 was used to provide small incremental changes to the learning rate. The batch size was chosen to be 10 to improve model generalization and performance well, also staying within the capabilities of the GPU. A step LR scheduler was used within the training loop. The loss function decided upon was cross entropy loss, several loss functions including MSE were experimented with but this one provided the strongest analysis. The number of epochs was 20 chosen to minimize time of training but still allow the model to reach a minimum possible loss value.

The loss plot for both the vanilla and upgraded model can be seen below showing their descent over time.

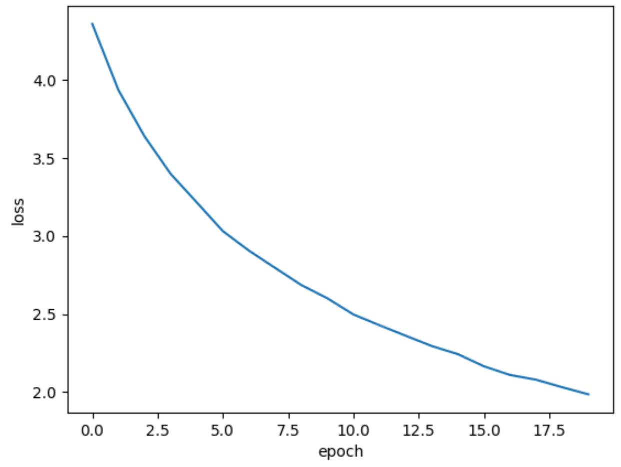


Figure 1: Vanilla Model Loss

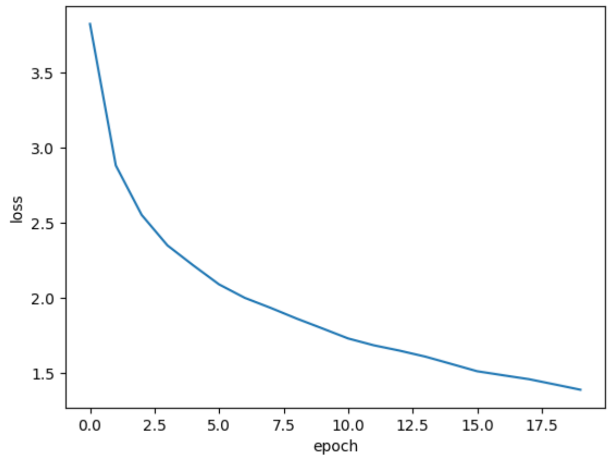
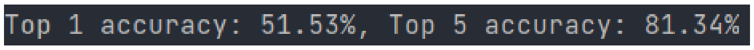


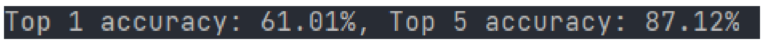
Figure 2: Upgraded Model Loss

As seen in the loss plots the upgraded model has a slightly better training curve with a lower final loss value over the 20-epoch training time. The upgraded model also had a longer training time taking around 320 seconds per epoch for a full time of around 1.7 hours whereas the vanilla model trained in around 60 minutes or 180 seconds per epoch.



```
Top 1 accuracy: 51.53%, Top 5 accuracy: 81.34%
```

Figure 3: Vanilla Model Accuracy



```
Top 1 accuracy: 61.01%, Top 5 accuracy: 87.12%
```

Figure 4: Upgraded Model Accuracy

Shown above are the vanilla and upgraded models' accuracy values when run through testing. Clearly the upgraded model has a much better top 1 accuracy and a slightly better top 5 accuracy which is mostly likely due to all the modifications made based on in class learning.

Section 4: Discussion

My system performed adequately, results were not exceptional but clearly the model was learning and predicting what images were with decent accuracy. The vanilla model performed much better than expected due to the fact it was of simple design and created very quickly compared to the modified model. The main difference the group noticed was the inclusion of dropout layers in the vanilla model and not in the modified model which may have had a larger impact on performance than initially expected. We expected the modified model to provide a larger improvement in accuracy than it did, although it did have slight improvements the complexity of the modified model was much higher than the results reflect. This lack of improvement could be attributed to several factors, the first would be overfitting. In the group's first few attempts at creating a modified model the model was found to be overfitting in each case and the lack of dropout layers could have been a key factor in this development. Another likely answer could be hyperparameter tuning, by the time the group had moved onto the modified model there was not enough time to properly tune the intricate features as much as possible and the hyperparameters used were the same as the vanilla model which may not have been the ideal values for the modified model. The upgraded model also took much longer to train than the vanilla, which makes sense due to the higher level of complexity but should have once again provided a larger improvement in accuracy.

The upgraded model traded efficiency for accuracy but not at an efficient rate, it nearly doubled training time but only improved accuracy around 10% for top 1 and 6% for top 5. In the future an ideal model would be further adapted off the vanilla model to find the strongest balance between efficiency and accuracy.