Josue Casco-Rodriguez
ELEC 576

ELEC 576 Assignment 2

1 Visualizing a CNN with CIFAR10

a) CIFAR10 Dataset

b) Train LeNet5 on CIFAR10

The best performance achieved on the CIFAR10 classification task using the LeNet5 architecture was roughly 60% accuracy on both the validation set (which was a held-out set of 10% of the training data) and test set, and was achieved using the following hyperparameters:

- Glorot Normal (Xavier) weight initialization
- Bias initialization to a positive constant
- Learning rate of 1.7e-4
- Nadam optimizer (Adam with Nesterov momentum)
- Dropout rate of 80% applied to the first fully connected (Dense) layer
- 25 epochs of training
- Batch size of 50
- Batch normalization (before max-pooling) on each convolutional layer
- L2 regularization with weight 0.01 on each convolutional layer
- L2 regularization with weight 0.005 on the first fully connected layer

While the maximum training accuracy below is roughly 80%, the maximum training accuracy achievable was above 80% when training for more than 25 epochs, but the additional training did not improve accuracy on the validation and test sets, meaning that the network began overfitting. Plotted below are the loss and accuracy for the LeNet5 model using the aforementioned hyperparameters:

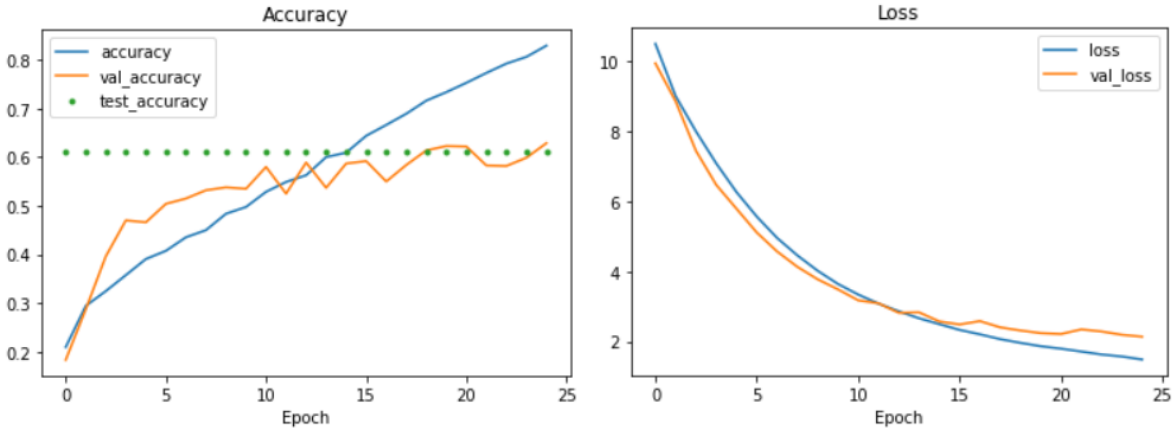

Figure 1: Accuracy and Loss for LeNet5 on CIFAR10 Dataset

c) Visualize the Trained Network

Shown below are the weights of each of the 32 5x5 filters from the first convolutional layer:

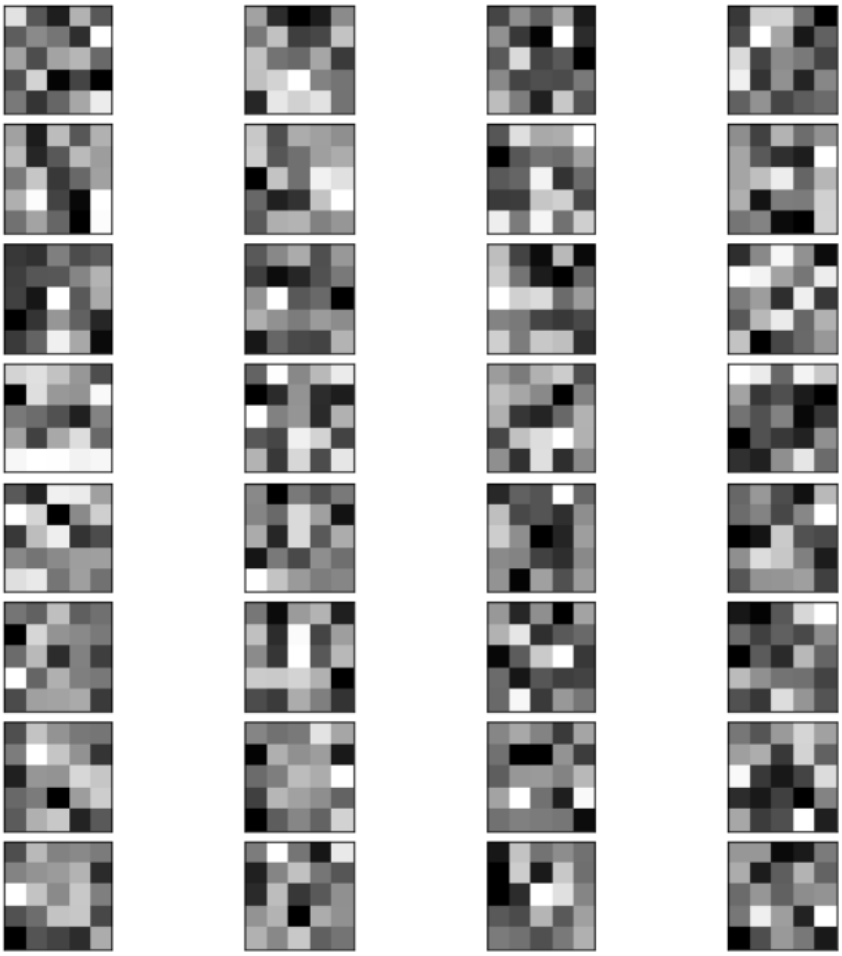

Figure 2: Weights of the First Convolutional Layer

Additionally, the activations of each convolutional layer, before batch normalization and max-pooling, are plotted below. While not shown, batch normalization (before max pooling) significantly helped the output of the first convolutional layer to be more normally distributed, but only slightly affected the distribution of the output of the second convolutional layer.

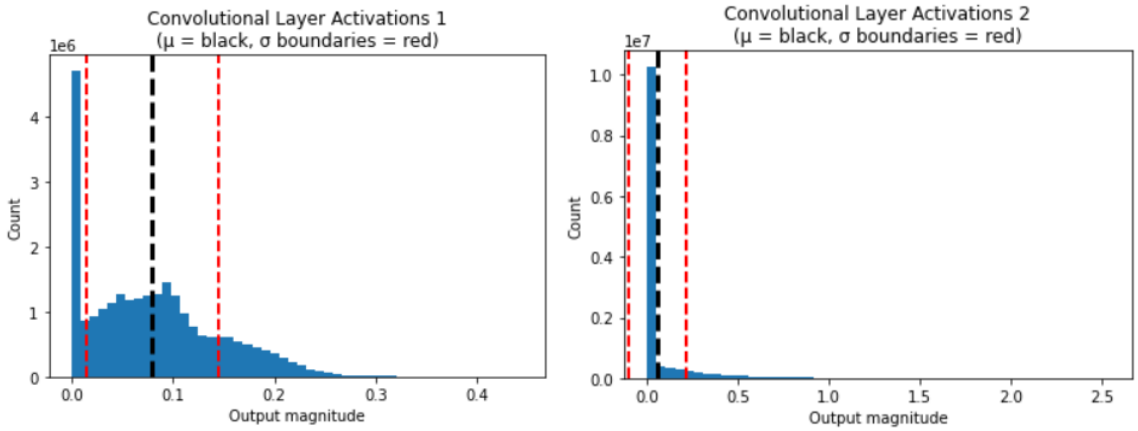

Figure 3: Activations of Each Convolutional Layer (Before Max-Pooling and Batch Normalization)

2 Visualizing and Understanding Convolutional Networks

The authors set out to better understand convolutional neural networks (CNNs)—specifically, the features that they actually learn and how important each of those features are in image classification tasks. Their primary novel method for CNN inspection involved a deconvolutional network that took high-level activations from the CNN and performed the reverse of the operations that the CNN used to produce the activations (i.e., the deconvolutional network performed deconvolution and unpooling operations). After training their CNN on the ImageNet 2012 training set, they ran all the images from the corresponding validation set through their CNN and, for each filter of each convolutional layer, identified the set of images (often 9 images) that most strongly activated the filter. Next, they ran the aforementioned strongest filter activations through their deconvolutional network, so as to convert the activations back into pixel space, so that readers can visually observe patches of real images that most strongly activated each filter.

The authors found that the first convolutional layer's filters detected edges of various frequencies and colors, such as patches with a single edge in them, or patches of one solid color. The second layer was found to identify geometric combinations of structures from the first layer (such as parallel lines, blobs, and circles), while the third layer was found to begin identifying parts of an animal (such as eyes, legs, and faces). Similarly to the third layer, the fourth layer was found to identify parts of animals, except at a more categorical and semantic level (i.e., distinguishing between dogs' faces and humans' faces), with the top image patches that activated each filter sharing not necessarily the same orientation or pose in space, but the same object in the patch (the authors found

that such shift invariance was likely due to operations such as max-pooling). The fifth and final convolutional layer was found to identify different objects (e.g., keyboards, specific dog breeds).

The authors used their deconvolutional network to inspect the CNN that won the ImageNet 2012 challenge, and then used their findings to diagnose that the neural network needed improved filter normalization strategies, smaller filter sizes in order to improve performance, and a smaller convolutional stride size. Upon implementing such changes, their new CNN model won the 2013 ImageNet challenge. They also showed that the image features learned by their CNN were applicable to other computer vision tasks and datasets, such as the Caltech 256 dataset.

3 Build and Train an RNN on MNIST

a) Setup an RNN

When using a simple RNN on the MNIST dataset, the following hyperparameters, when used together, achieved a top accuracy on the validation, test, and training sets of between 96% and 97%:

- Learning rate of 0.001
- Nadam optimizer (Adam + Nesterov momentum)
- Batch normalization between the RNN layer and the Dense feedforward layer
- 64 hidden layer units
- 10 epochs
- Batch size of 32
- Sparse categorical cross-entropy (from logits)

b) How about using an LSTM or GRU

Surprisingly, the LSTM architecture performs better than the RNN architecture, even when using many fewer hidden units. However, the price of such improvements in performance is significantly more time spent training. The best accuracy achieved on the validation, test, and training sets was between 98% and 99%, made possible by using 128 hidden units in an LSTM architecture. Plotted below are the accuracies and loss for the RNN and LSTM architectures, using 16, 32, 64, and 128 hidden units:

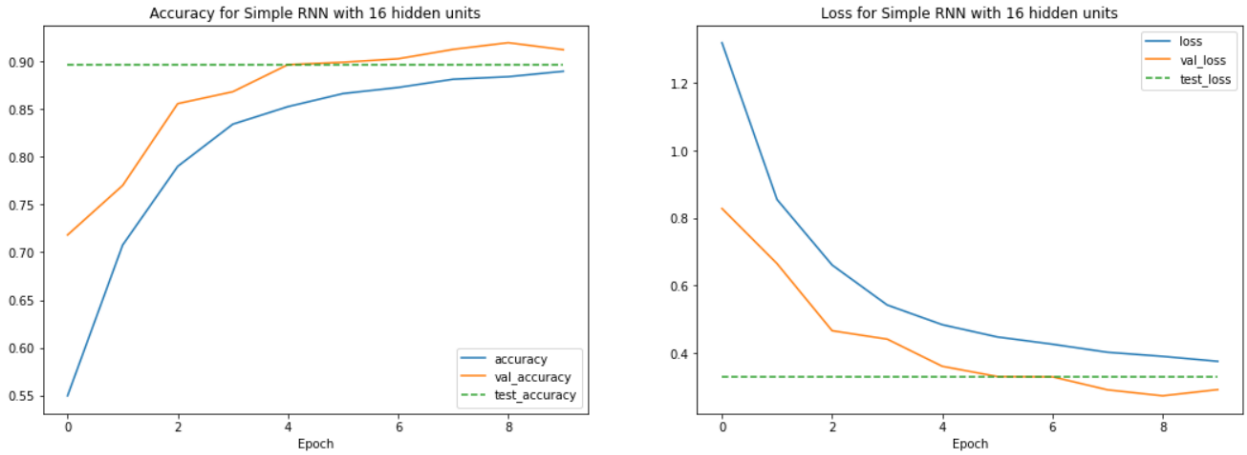

Figure 4: Accuracy and Loss for Simple RNN with 16 Hidden Units

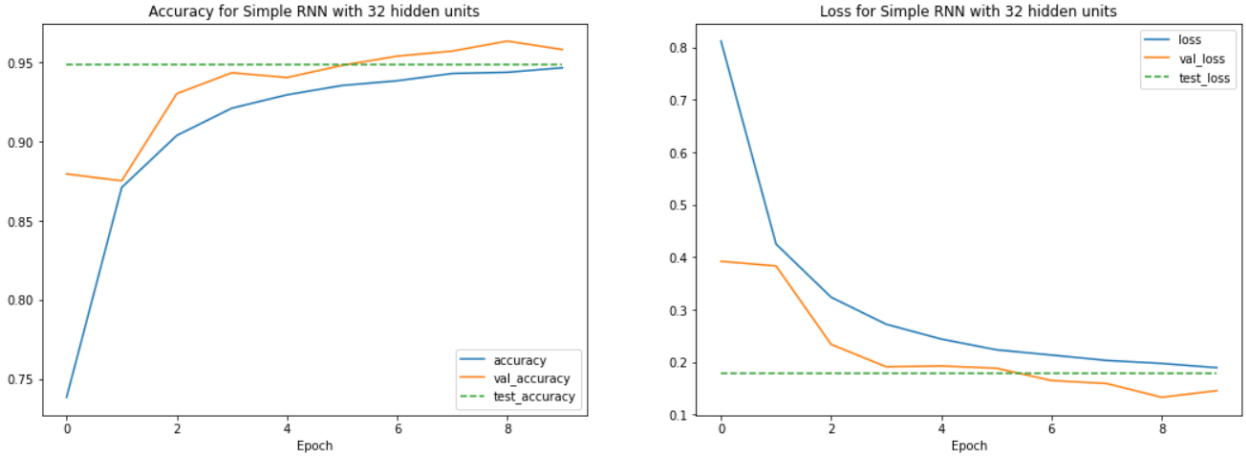

Figure 5: Accuracy and Loss for Simple RNN with 32 Hidden Units

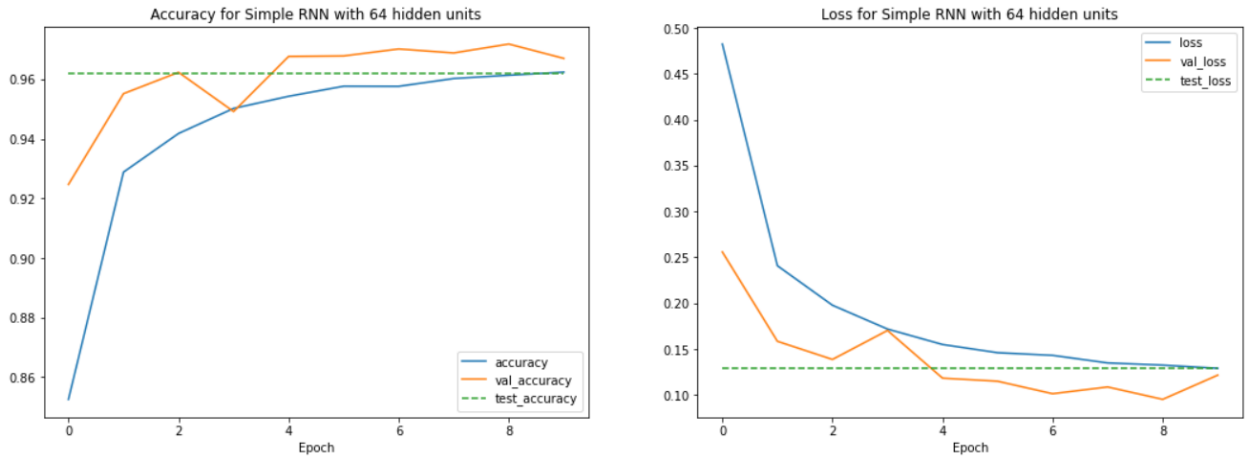

Figure 6: Accuracy and Loss for Simple RNN with 64 Hidden Units

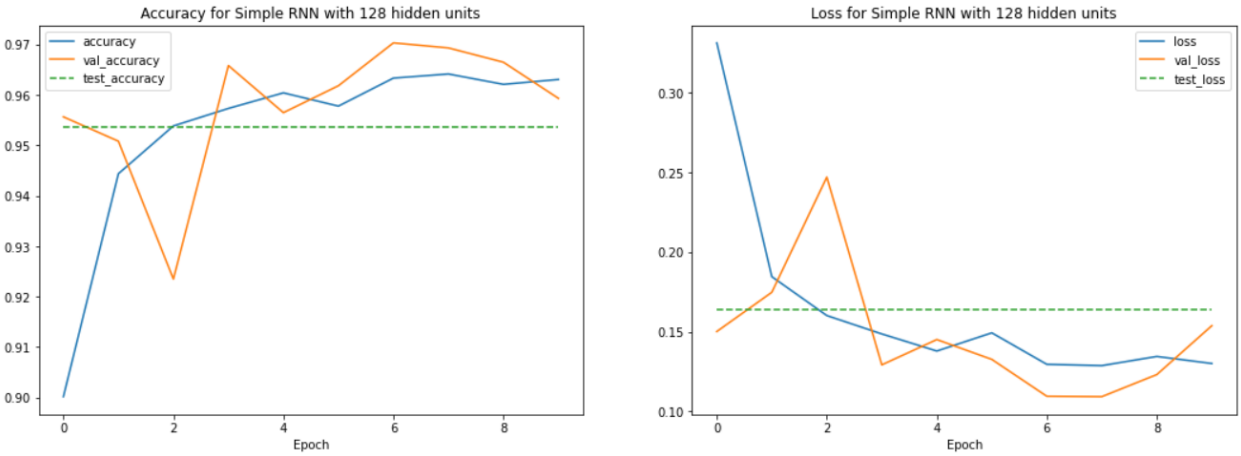

Figure 7: Accuracy and Loss for Simple RNN with 128 Hidden Units

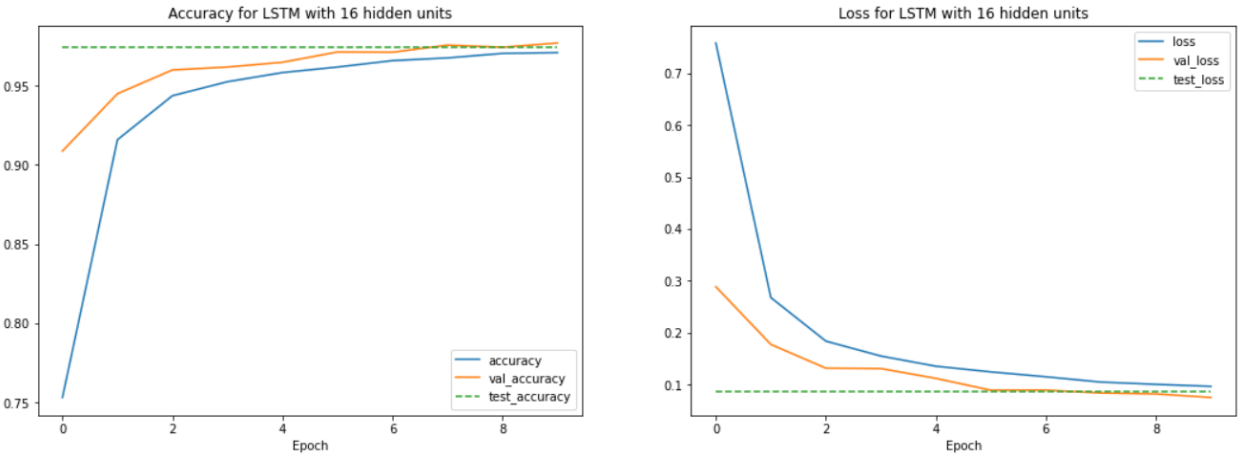

Figure 8: Accuracy and Loss for LSTM with 16 Hidden Units

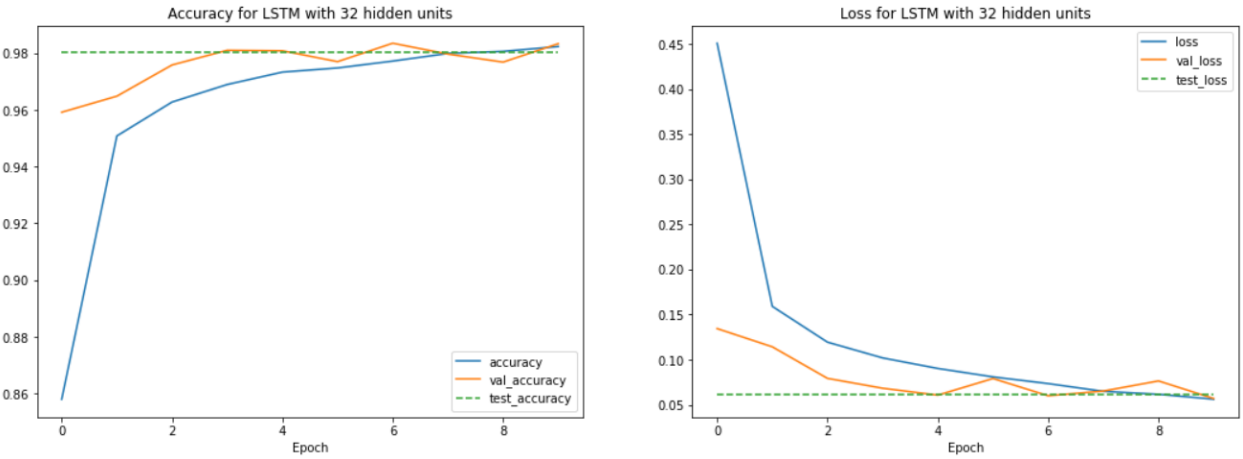

Figure 9: Accuracy and Loss for LSTM with 32 Hidden Units

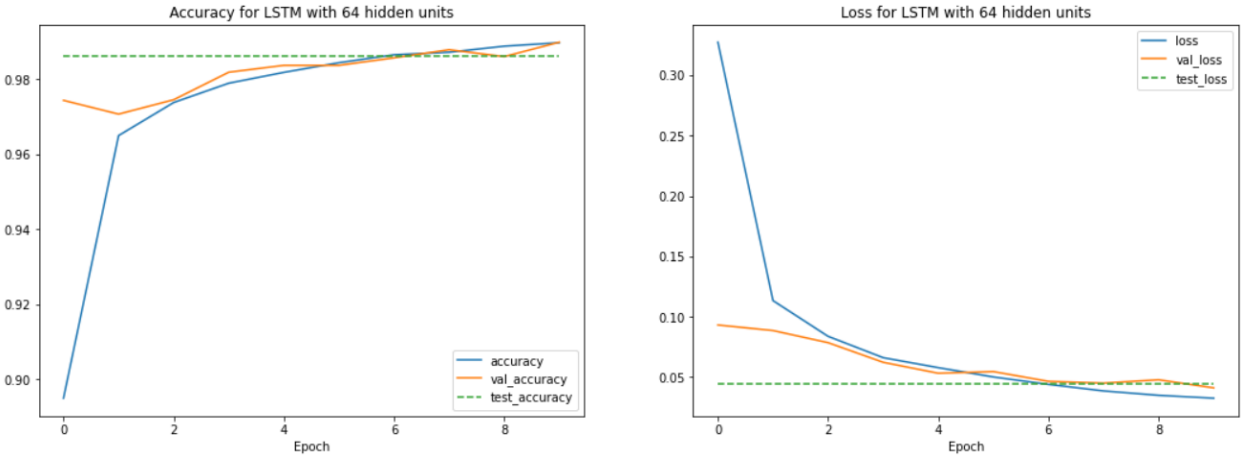

Figure 10: Accuracy and Loss for LSTM with 64 Hidden Units

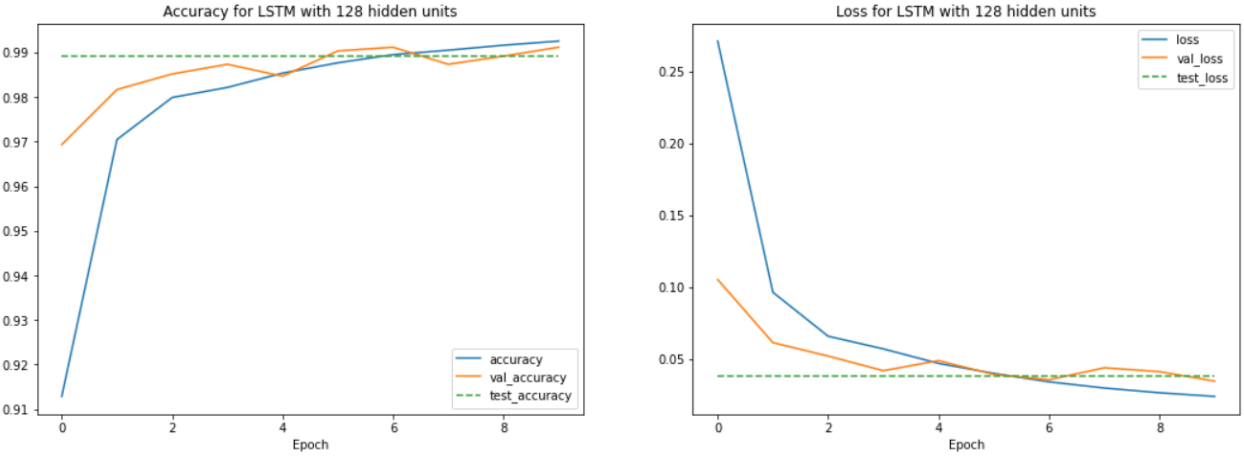

Figure 11: Accuracy and Loss for LSTM with 128 Hidden Units

c) Compare against the CNN

Compared to training RNNs, CNN training is usually fairly straightforward, not suffering from any issues with vanishing gradients. Similarly to RNN training, the main challenges in CNN training involve hyperparameter tuning and devising regularization and gradient descent strategies to achieve accuracy while avoiding overfitting. However, RNN training usually also involves additional effort spent optimizing how the recurrent cells should process incoming data—for example, tuning the number of hidden units in the recurrent cells. Surprisingly, the RNN training in this assignment was much easier to optimize than the CNN training from earlier in the assignment, perhaps because the images in the MNIST dataset had a higher resolution compared to the shrunken images in the CIFAR10 dataset, even though both datasets' images were of the same shape (28 x 28 pixels).