1 Fully-Connected Feed-Forward Neural Network Model

1.1 Mean-Squared Error

As the section title suggests, this is for the comparisons of all of my Mean-Squared Error loss function models. My base, or vanilla, model had the following hyper-parameters:

- Mean-squared error to monitor loss.
- Two hidden layers: 64 and 32 nodes.
- TanH activation functions for every node.
- Default learning rate (0.001).
- Default momentum rate (0.001).

The vanilla model had accuracy and losses in each epoch, shown in Figure 1.

1.1.1 Learning Rate

One of the parameters I changed for comparison was learning rate. I adjusted the learning rate to be 0.05. This is significantly more than the default of 0.001. After the modification, we see in Figure 2, that there is more of a bow in the curves which shows that the model is a little better training since its diminishing accuracy and loss is exponential instead of linear.
The importance of it being exponentially diminishing is that if we changed our early stopping conditions to stop early, the model would have a better accuracy than its vanilla counterpart. We see this trend continued in the overall ending accuracy of both models: the base ending at 85% accuracy and the learning rate modified ending at 90% accuracy.

1.1.2 Hidden Layers

Another parameter I changed while using the tanh activation function units was the number of nodes in the first hidden layer from 64 nodes to 32 (after resetting the learning rate to the default rate). I thought that the dimension reduction from the input layer might help speed up the model and train it in less epochs. We see in Figure 3, that this hypothesis was correct as it trained in 260 epochs. The trade-off is evidently the accuracy as it ends training at 73% which is worse than both of its sister models, up to this point.
Perhaps it would be worth investigating the combination of the 0.05 learning rate and reduced first hidden layer, but not in this lab, I just want to get it done with as little models as possible (and the naming structure I made will get out of hand with more than a few parameter changes).

1.1.3 Rectified Linear Unit

The next modification I made, after resetting the model to the base model, was change the activation function of all of the hidden layers' nodes to be ReLU. This activation function is known for being 'sticky'–once the node gets a value less than zero, the function's derivative is also zero so it never adjusts–we see this in action with its multi-level shaping. It starts to level off around 75% accuracy, then something happens and it gets some large gains in accuracy after. I cannot tell if it is cyclical or random from the graph, but we see something similar happening around 35% and 47% accuracy.
This can make it not ideal for other early stopping parameters, especially if they stop more early than ours – it could be settling for an accuracy less than its true potential. It seems that our early stopping parameters worked well since we get to 89% accuracy in about the same number of epochs as the hidden layer modified tanh model. This is the best model so far.

1.1.4 Swish

The final modification I made for the mean-squared error model was trying out a different activation function I found while browsing my machine learning FaceBook or Reddit groups/subreddits. The function is called swish and it is defined as:

$$z = \frac{x}{1 + e^{-x}}.$$

It had a cool name and was easy enough to implement. It did pretty well, as we see in Figure 5. It takes the least epochs to train, but it only gets to 78% accuracy. Estimating the number of epochs for ReLU and Swish to be 260 and 190, respectively, and estimating the accuracy of ReLU and Swish to be 89% and 79%, respectively. I compared the ratios of accuracy per epoch and got that ReLU was $\frac{89}{260} = 0.342$ and Swish was $\frac{79}{190} = 0.415$. This means that Swish was more efficient so it takes the crown for best mean-squared error model.

1.2 Categorical Cross-Entropy

Same base model as 1.1 Mean-Squared Error, but this time we are using categorical cross-entropy as our loss function. The base model for CCE had accuracy and losses in each epoch, shown in Figure 6. Right away, we can see that this loss function is significantly better since it ended at a 100% accuracy.

1.2.1 Learning Rate

As stated before, same modifications as section 1.1.1. We see that there is a little bit of instability from the wiggles in the accuracy curve, but the model gets 99% accuracy like the base model, but in slightly less epochs, making it more efficient at training.

1.2.2 Hidden Layers

Same modifications as section 1.1.2, see Figure 8. This model one-ups the learning rate's efficiency by getting to 98% accuracy with approx. 66 epochs. Good job rock.
It is very evident by now what an improvement it is to use loss functions that make sense, this set of models are much more competitive, whereas MSE had a very clear hierarchy of efficiency. Before I had to re-adjust because I saw we had defined loss functions we had to use, I was using binary cross-entropy and got some really great accuracy and stability (the training and validation never separated, the curves were smooth) because of how I had my labels transformed, but I digress.

1.2.3 Rectified Linear Unit

Different loss, same modifications as 1.1.3, see Figure 9. As we see... how do I say this professionally... It looks like hot garbage. This is likely due to the 'sticky-ness' of ReLU units I was talking about before, the stability is obviously not stable at all.
Despite it all, it gets a perfect accuracy in only 46 epochs. I am gobsmacked. I suppose that still makes it the best model so far.

1.2.4 Swish

Do I have to keep saying it? Look at Figure 10. I had high hopes for this activation function, but because of the similarity of function to ReLU, it is also unstable. While unstable, it is not *as* unstable. It takes only about 1 more epoch than ReLU, but without all of the craziness, so I am going to give it the best-model title for aesthetics. I am biased towards better looking graphs.

2 Convolutional Neural Network Model

2.1 Base Model

My base model for the convoluted model had the following hyper-paramters:

- Two hidden layers with ReLU activation functions:
  - Sixteen 4x4 filter nodes.
  - Eight 2x2 filter nodes.
- Default learning rate (0.001).
- Default momentum rate (0.001).

The base model had accuracy and losses each epoch, shown in Figure 11.
It seems to over-fit judging by the separation of the training and validation accuracy, which is a new look considering that the fully-connected models all had practically the same accuracy for validation and training and over-fitting was not an issue.

2.1.1 Hyperbolic Tangent

I was very original and modified the activation function for both layers to be TanH fucntions. We see in Figure 12 that it does not do better than our base model, it does not get a perfect accuracy for training, it is still over-fitted, and it takes more epochs to train. I am rather disappointed by these results.

2.1.2 Learning Rate

You already know, look at Figure 13. I changed the learning rate to 0.05, it performed about the same in training, but way less epochs and the validation performs much better than it did in the previous base and modified versions. And it is significantly less epochs, nearly half the previous versions.

2.1.3 Filter sizes

This is where my testing differs for this model than the fully connected model(s). I wanted to really play with the filter size, especially in a way that it changes the 'compression' of the hidden layers, so instead of going from more nodes into less nodes, I went from less nodes to more nodes. I changed the first hidden layer to be 9 6x6 filter nodes, then the second hidden layer to be 16 3x3 filter nodes. Obviously, it worked or I would not be talking about it, I'm sure you can do the math and see that it checks out on paper. Anyway, I did that, shown in Figure 14, and it performed near identically to TanH model, but with a some more epochs.

3 Conclusions

We can see from Section 1, that the best fully-connected feed-forward neural network model is the categorical cross-entropy swish activation function model. It takes the crown for its efficiency and accuracy; Its stability causes some concerns for me, but as we learned recently in class, for ensemble learning models, this stability would be acceptable, or even preferred.// The convolutional models left a lot on the table, I really thought they would be better for these applications but we see that they quickly overfitted and would not be preferable in production. I'm sure with more work and more tweaking, they could be better, but I think the size of our data just is not big enough for it to make sense. I remember writing the code and being like ”if it's only 8 by 8, do I even need a filter?” I think that since it is already normalized and so small, that it is hard for the model to abstract the info further without clear losses, and I think the results support that.// I would like to make a note of one of the major differences I saw while running the fully connected and convolutional models. I saw that the epochs for the CNN had 400 items(?) – each epoch took longer. This frightened me since the epochs sizes that I saw for the fully-connected were always 1. I don't know what caused this, especially because I copy-pasted the fully-connected into the CNN so the compile and fit and everything were the same, the layers and how I transformed the data to be 8 by 8 were the only things I changed. When you go through my code, pay attention to the epoch sizes and be on the lookout for what in my code might be causing it because I could not find it.

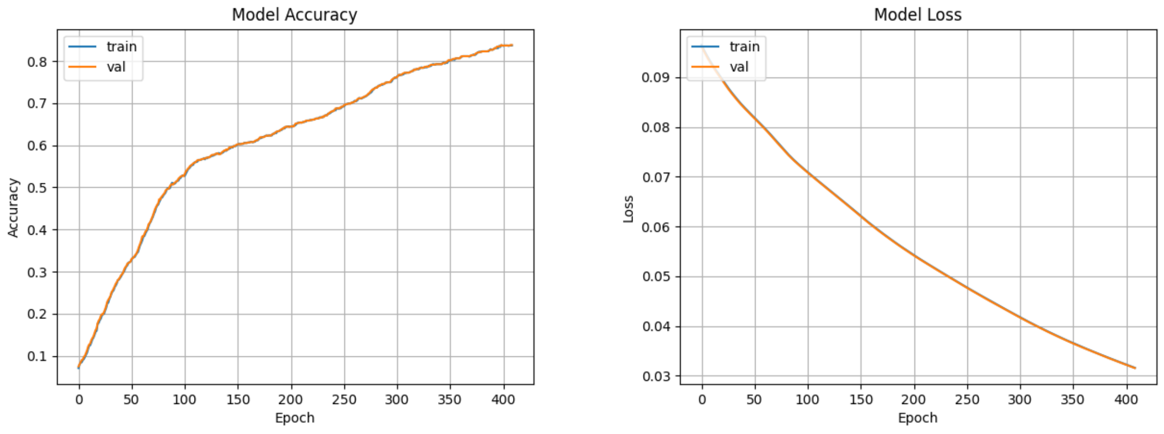

Figure 1: Acccuracy and Loss plots versus epoch for the base tanh model.

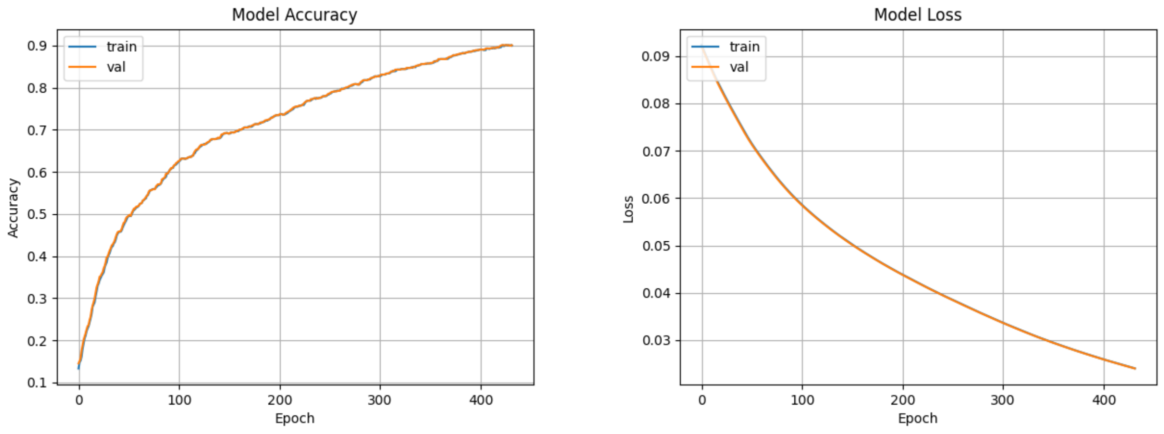

Figure 2: Acccuracy and Loss plots versus epoch for the adjusted learning rate tanh model.

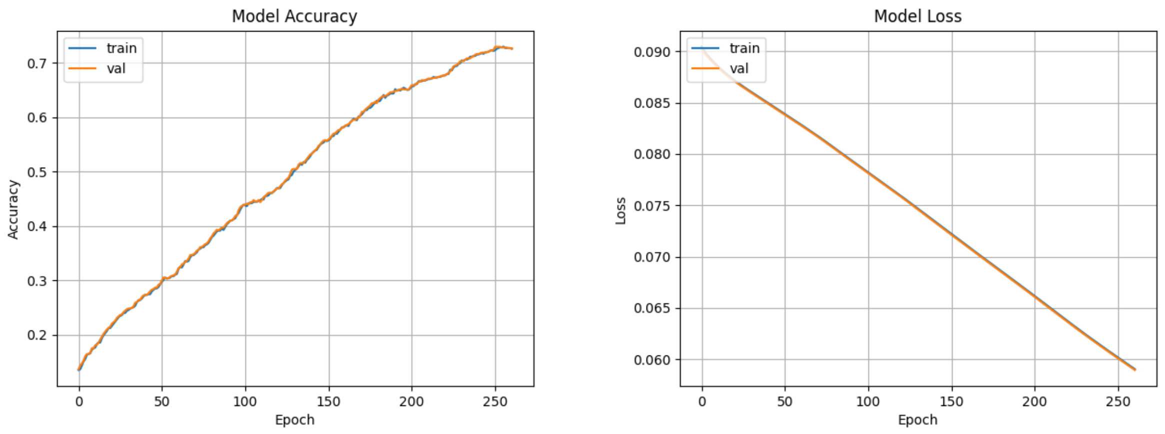

Figure 3: Acccuracy and Loss plots versus epoch for the adjusted hidden layer size tanh model.

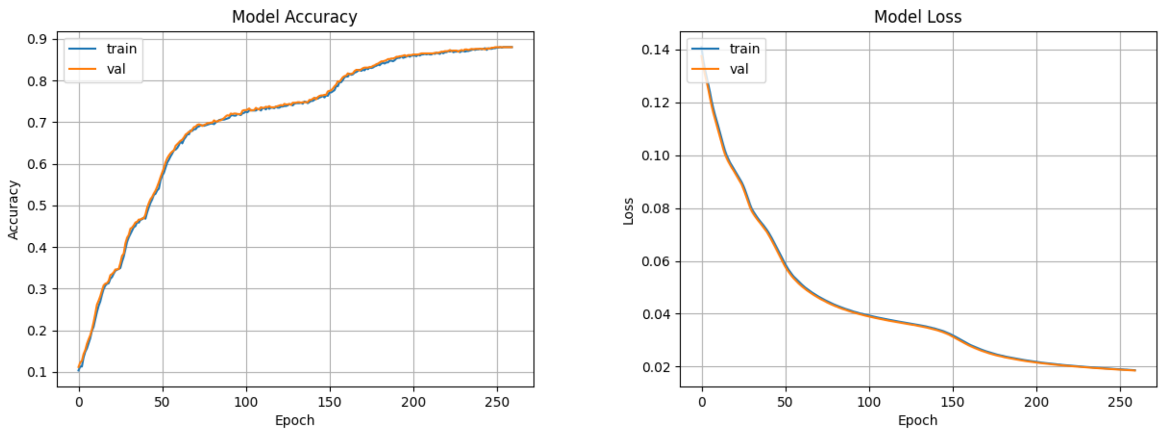

Figure 4: Acccuracy and Loss plots versus epoch for the ReLU activation function model.

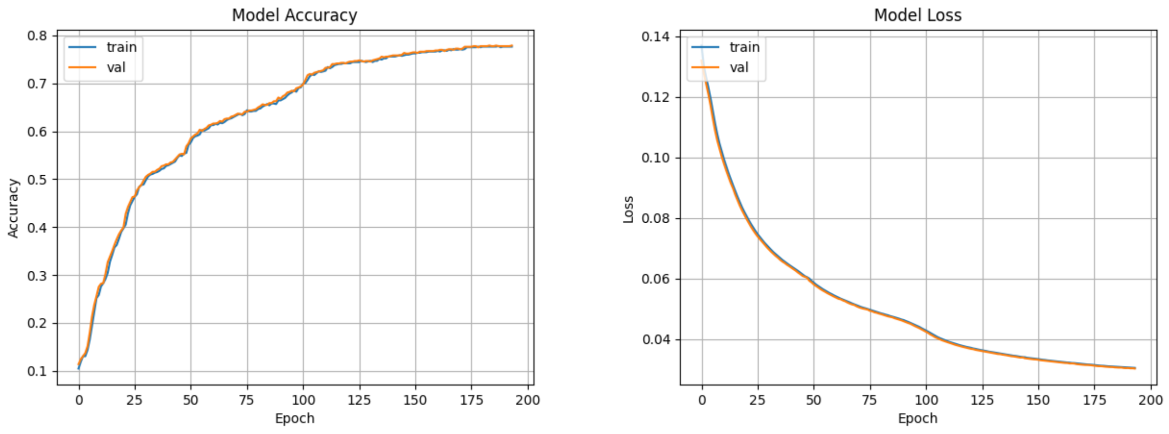

Figure 5: Acccuracy and Loss plots versus epoch for the swish activation function model.

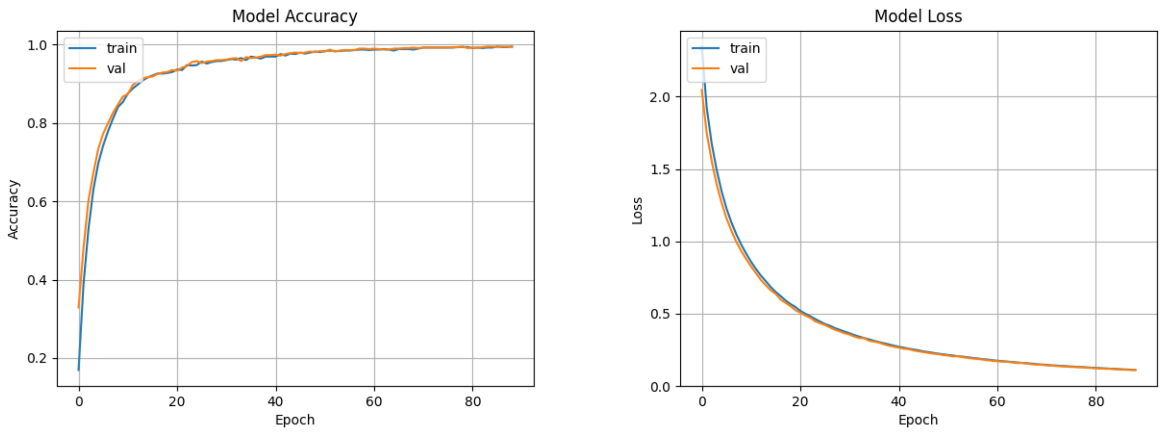

Figure 6: Acccuracy and Loss plots versus epoch for the base categorical cross-entropy tanh model.

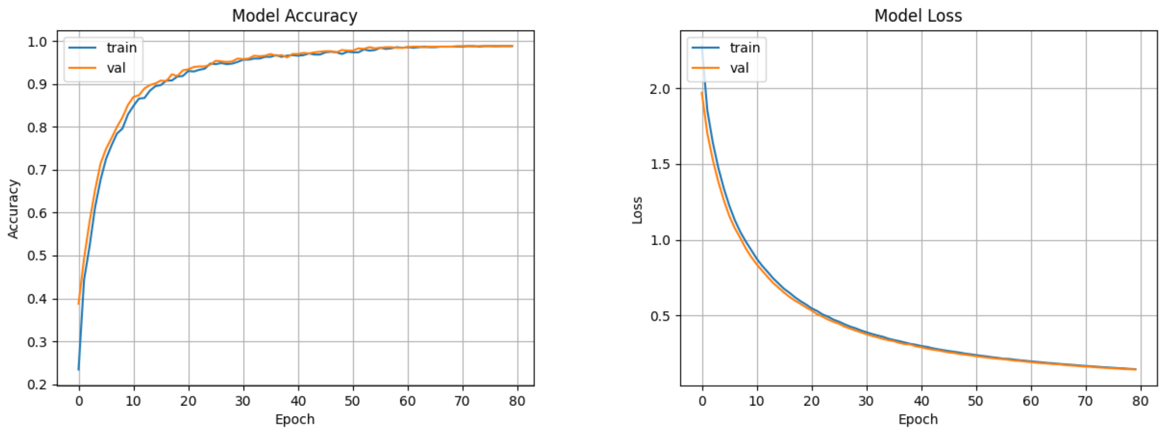

Figure 7: Acccuracy and Loss plots versus epoch for the modified learning rate tanh CCE model.

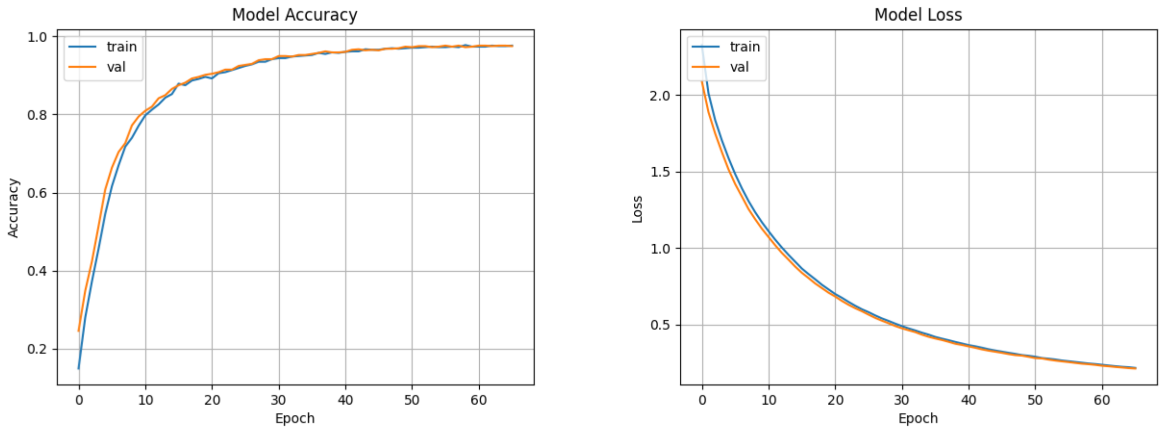

Figure 8: Acccuracy and Loss plots versus epoch for the modified number of nodes in the first hidden layer for the tanh CCE model.

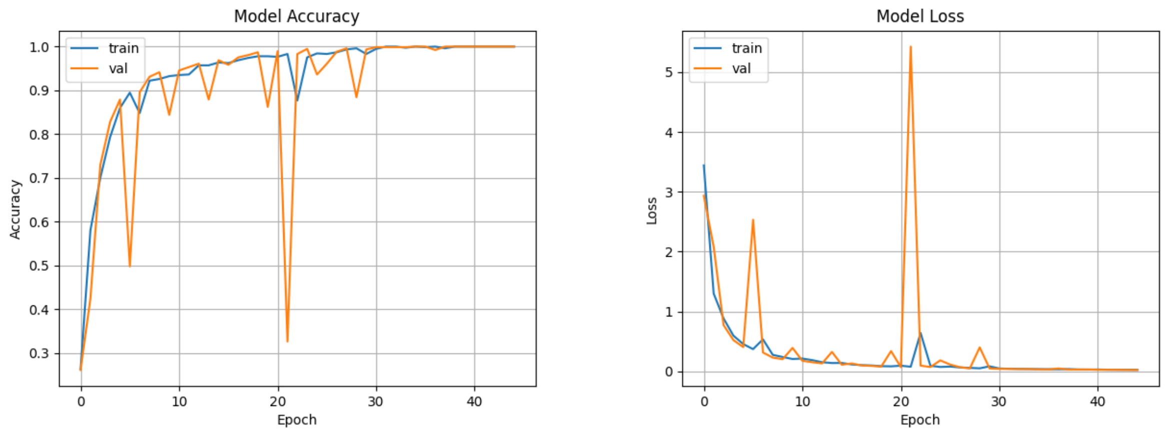

Figure 9: Acccuracy and Loss plots versus epoch for the ReLU CCE model.

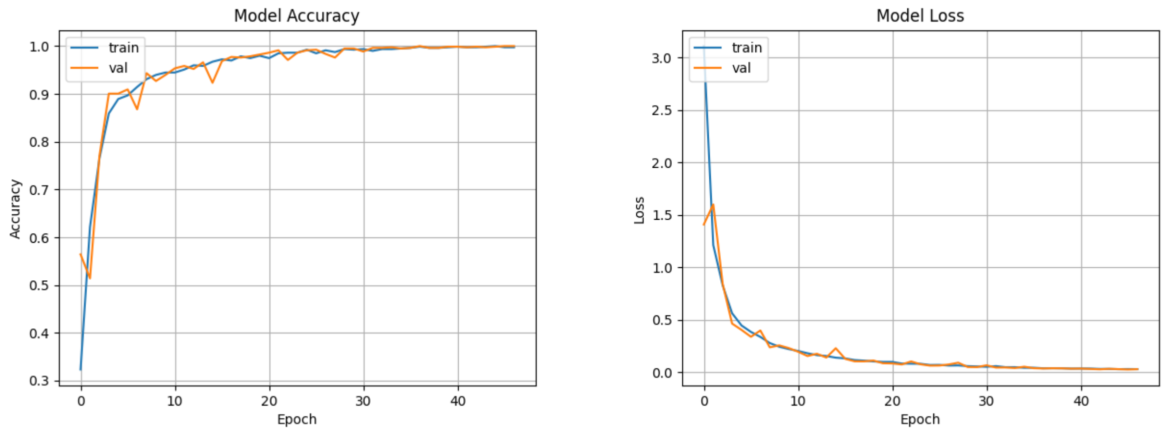

Figure 10: Acccuracy and Loss plots versus epoch for the Swish CCE model.

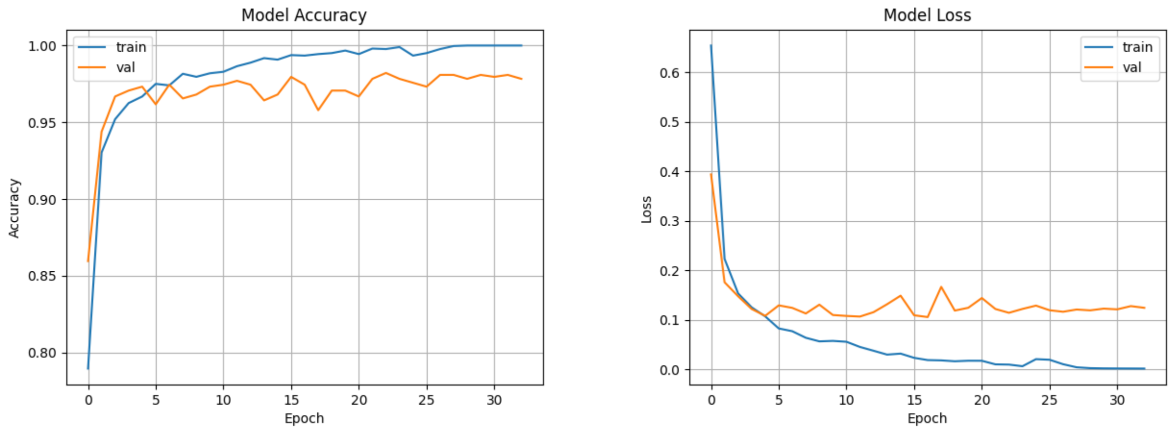

Figure 11: Acccuracy and Loss plots versus epoch for the base CNN model.

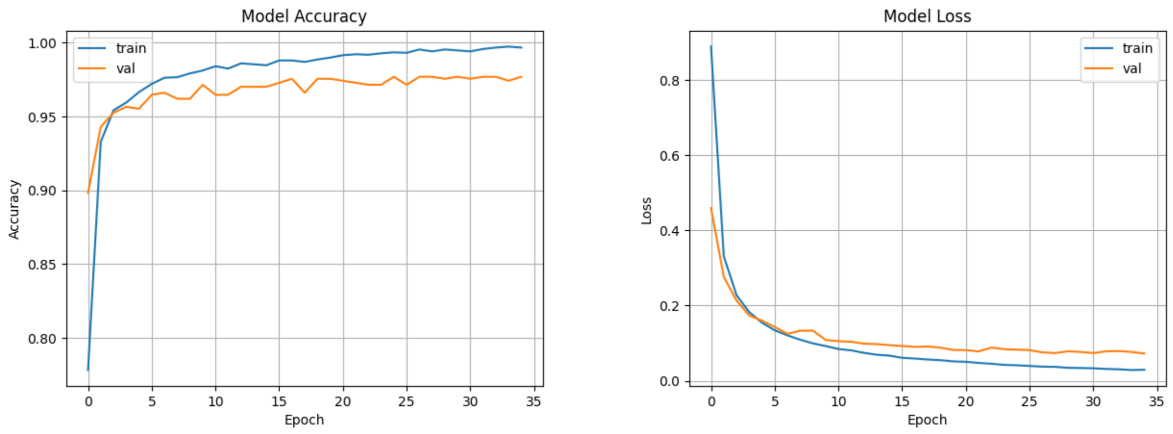

Figure 12: Acccuracy and Loss plots versus epoch for the TanH CNN model.

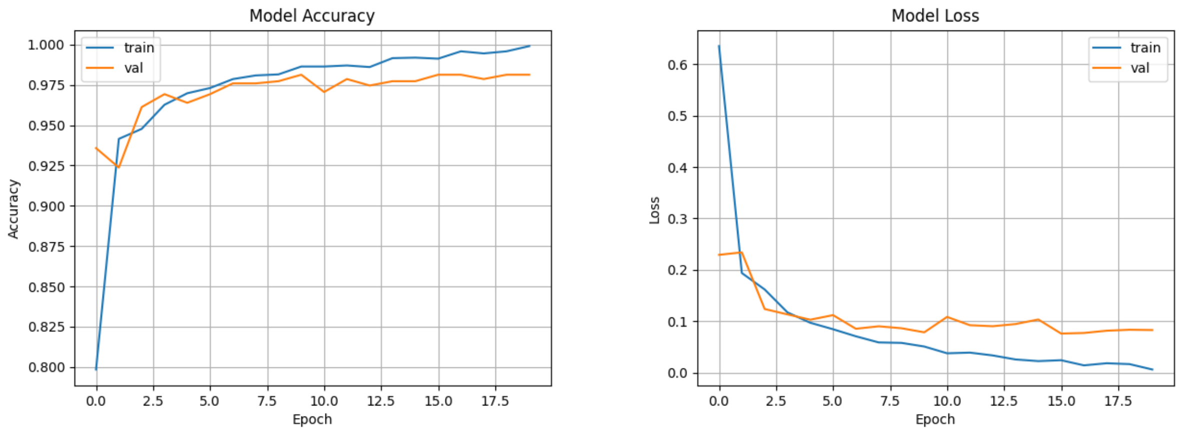

Figure 13: Acccuracy and Loss plots versus epoch for the modified learning rate CNN model.

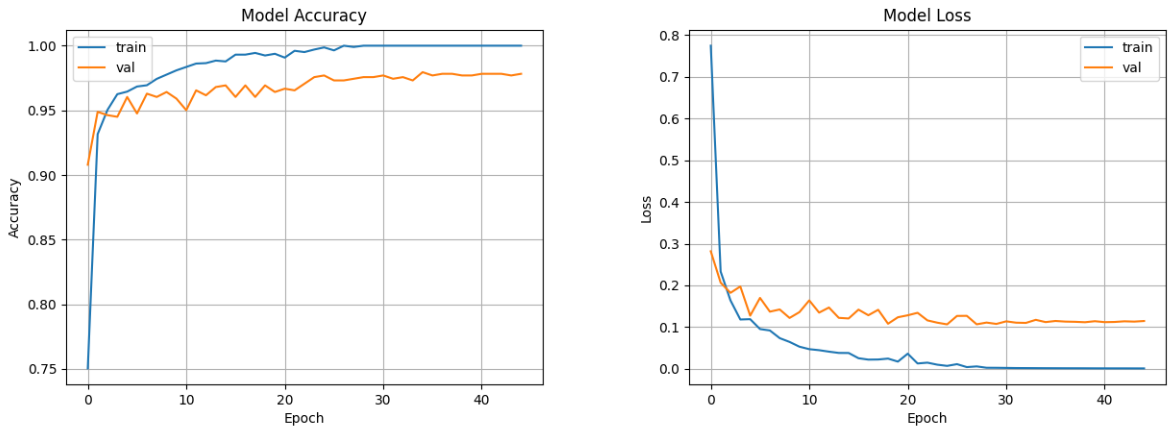

Figure 14: Acccuracy and Loss plots versus epoch for the modified filters CNN model.