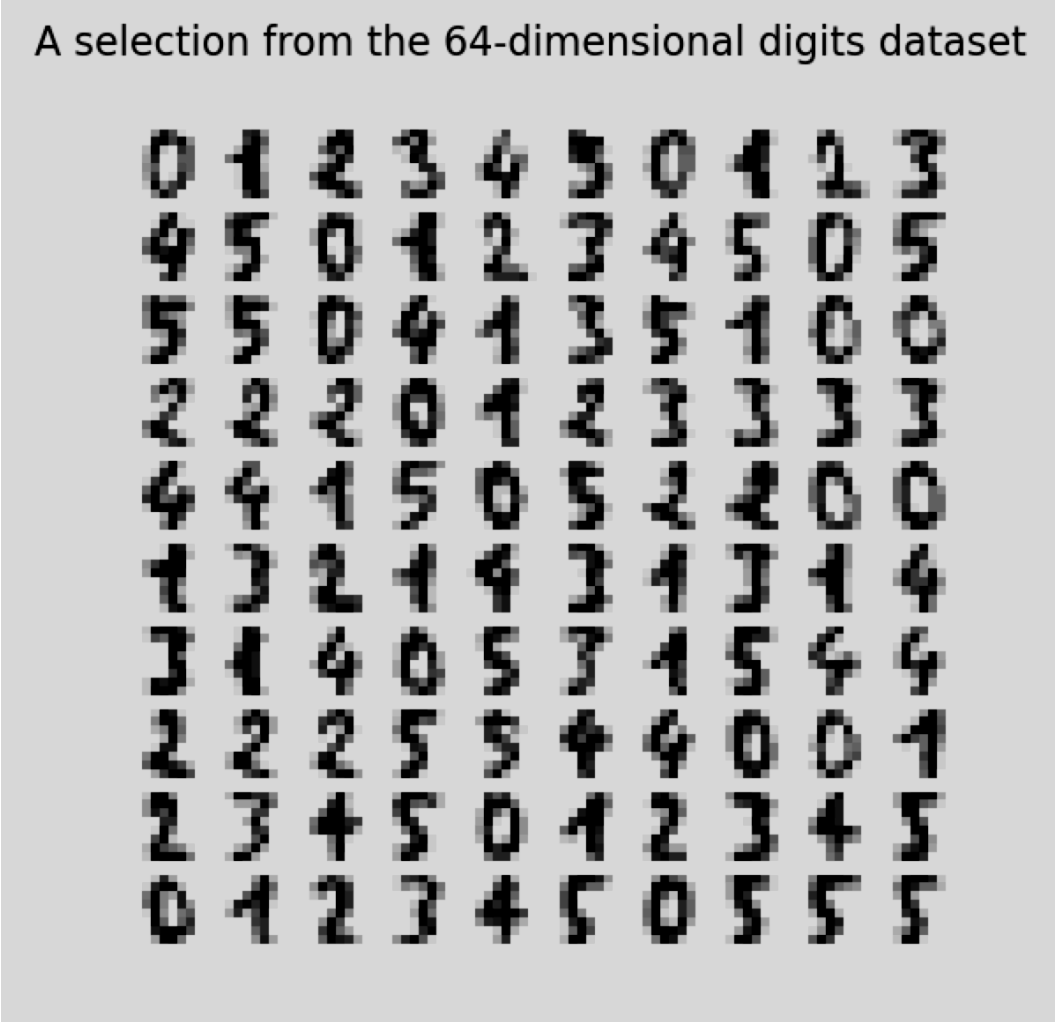
HW2: Effect of Optimizers and Learning Rate on MNIST Dataset

By Sundar Swaminathan

Dataset

The data set that will be used for this assignment is the popular MNIST dataset. We will be using the MNIST dataset provided by the scikit learn library(https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html, https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits). This is a dataset that contains 1797 images of shape 8*8. Although the original MNIST dataset has more images(60,000 28*28 images), I have used this because of faster processing and 2d images instead of 3d which reduces the flattening time.



Preprocessing

I have preprocessed the data in different steps as follows:

- **Conversion:** The data is converted into a Numpy Array as Keras expects Numpy arrays as input and does not take structured data
- **Precision:** The data is initially in an int based structured array, this is converted to float 32 format to improve precision and avoid overflow issues

- **Train-Test Split:** I have used a training and split ratio of 90%-10%. This ensures enough data is available for training while just enough is available for testing. This is an optimal split for small datasets like the one we're using in the project. If large data is set aside for testing, it would result in overfitting
- **Scaling:** I have used StandardScaler here to maintain a standard deviation of 1 to improve model convergence
- **One-hot encoding:** Since we're using softmax in the output layer, using one hot encoding of the input layer ensures compatibility.

Architecture

Input

Layer 1: 64 neuron input layer(data set)
Layer 2: Hidden Layer with 100 neurons with activation function tanh
Layer 3: Optional Hidden Layer with 100 neurons with activation function tanh

---

Output

10 neuron output layer with activation function softmax

Optimizers

Training 1: SGD with 3 learning rates(0.001, 0.01, 1.0) and 2 validation splits(0.1, 0.2)
Training 2: Adam with Default learning rate and validation split of 0.2

Epochs

I've used a standard number of epochs of 15 for equal comparison

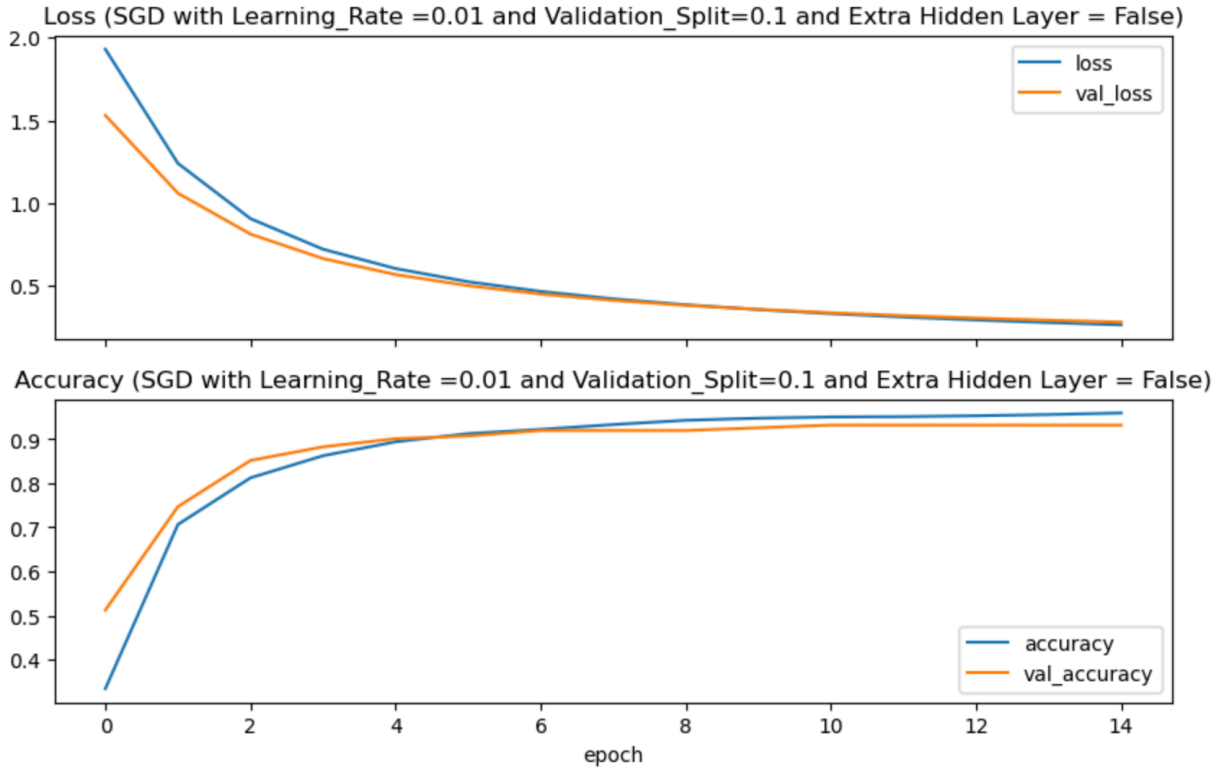
Training

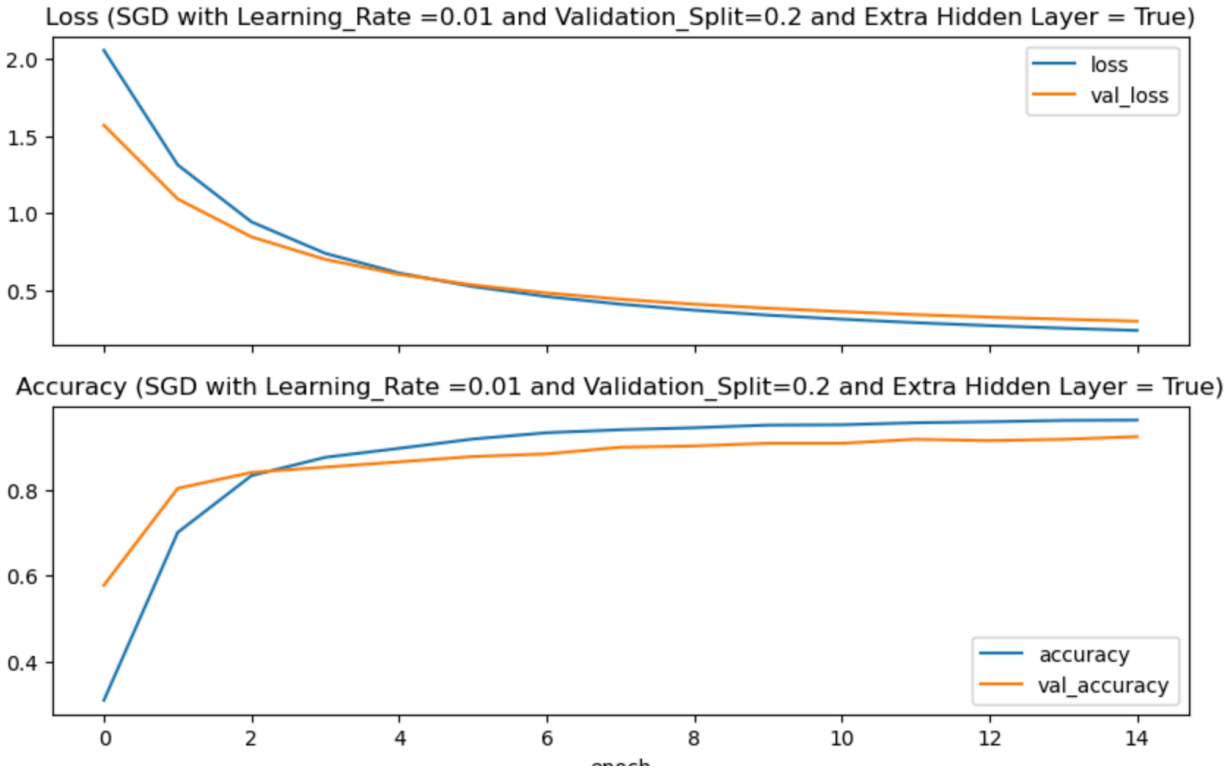
For the training model, which is the most pivotal part of the whole process, I created a total of 3 different learning rates combined with 2 different validation splits, which resulted in a total of 6 different modes of training each with their own outputs. I also introduced a third parameter, called hidden_layer which will toggle between true and false and introduce an additional hidden layer in the training depending on the toggle. This was done to contrast the difference in learning and performance of the model with additional input layer. For both the input layers, I used a tanh activation function, which was the requirement and also it generates an output in the range (-1, 1) which is smoothened later.

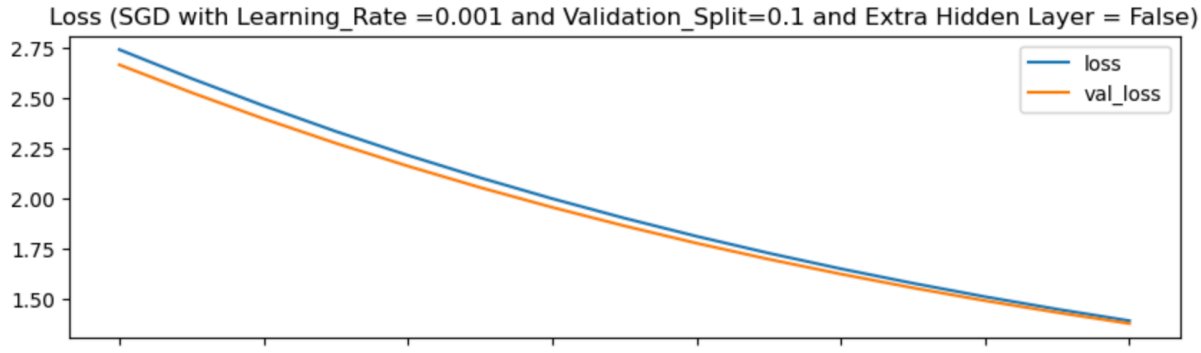
Impact of Learning Rate

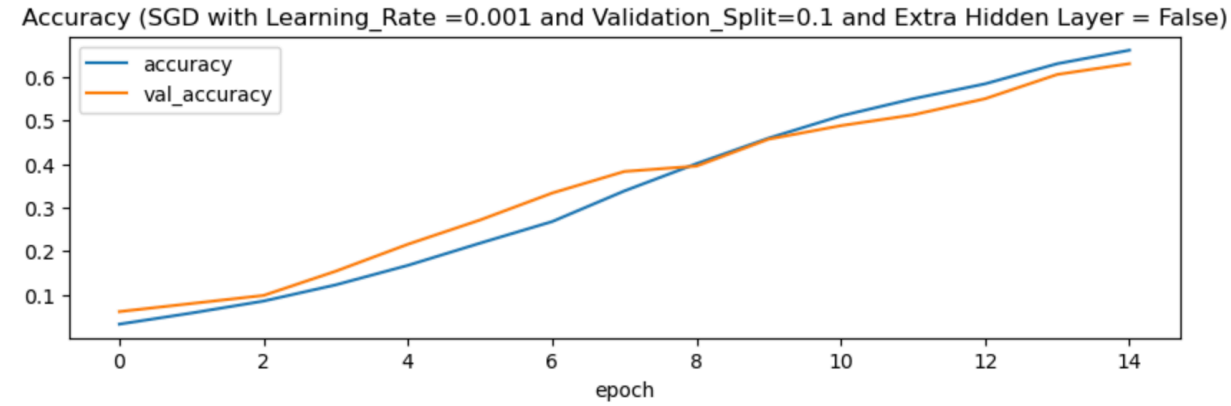
With SGD, I observed that learning rate significantly impacts the performance of the model. The lower the learning rate, the better the convergence between loss, accuracy and val_loss, val_accuracy respectively. Also I observed that having a higher validation split resulted in lower convergence compared to having a lower validation split(0.2 vs 0.1). This observation can be correlated with the train and test split discussed in the Preprocessing section.

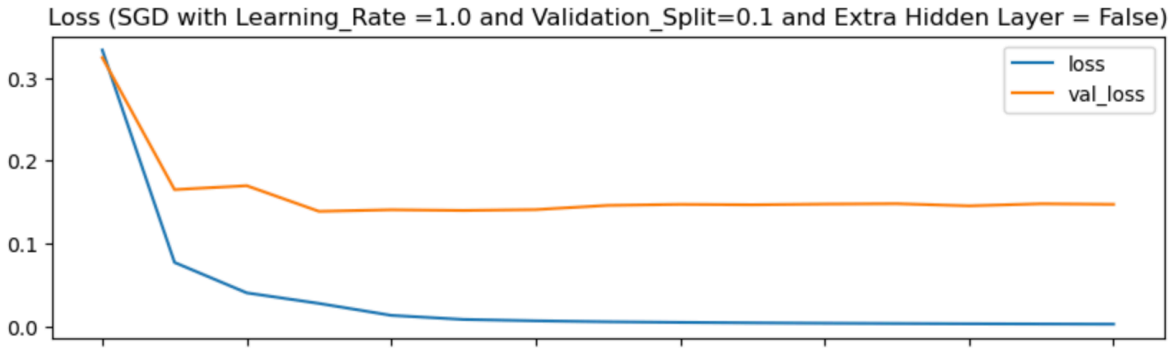
I also observed that using a very high learning rate led to the overfitting problem where extremely high divergence was found. Even this divergence could be controlled with adding an extra hidden layer between the input and the output.
Below are a few plots for the loss and val_loss and their corresponding convergence and divergence with different learning_rate, validation_split and hidden_layers

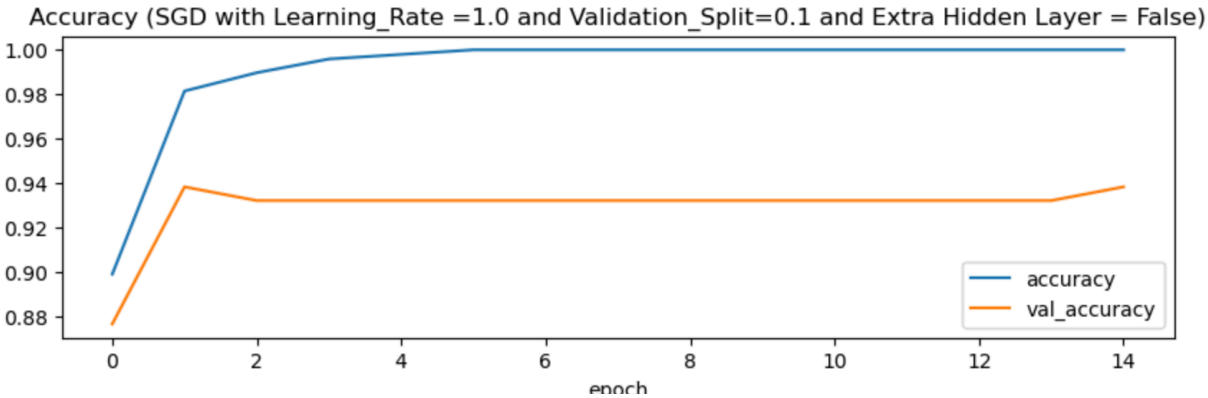

Loss (SGD with Learning_Rate =0.01 and Validation_Split=0.1 and Extra Hidden Layer = False)
loss
val_loss
Accuracy (SGD with Learning_Rate =0.01 and Validation_Split=0.1 and Extra Hidden Layer = False)
accuracy
val_accuracy
epoch


Loss (SGD with Learning_Rate =0.01 and Validation_Split=0.2 and Extra Hidden Layer = True)
loss
val_loss
Accuracy (SGD with Learning_Rate =0.01 and Validation_Split=0.2 and Extra Hidden Layer = True)
accuracy
val_accuracy





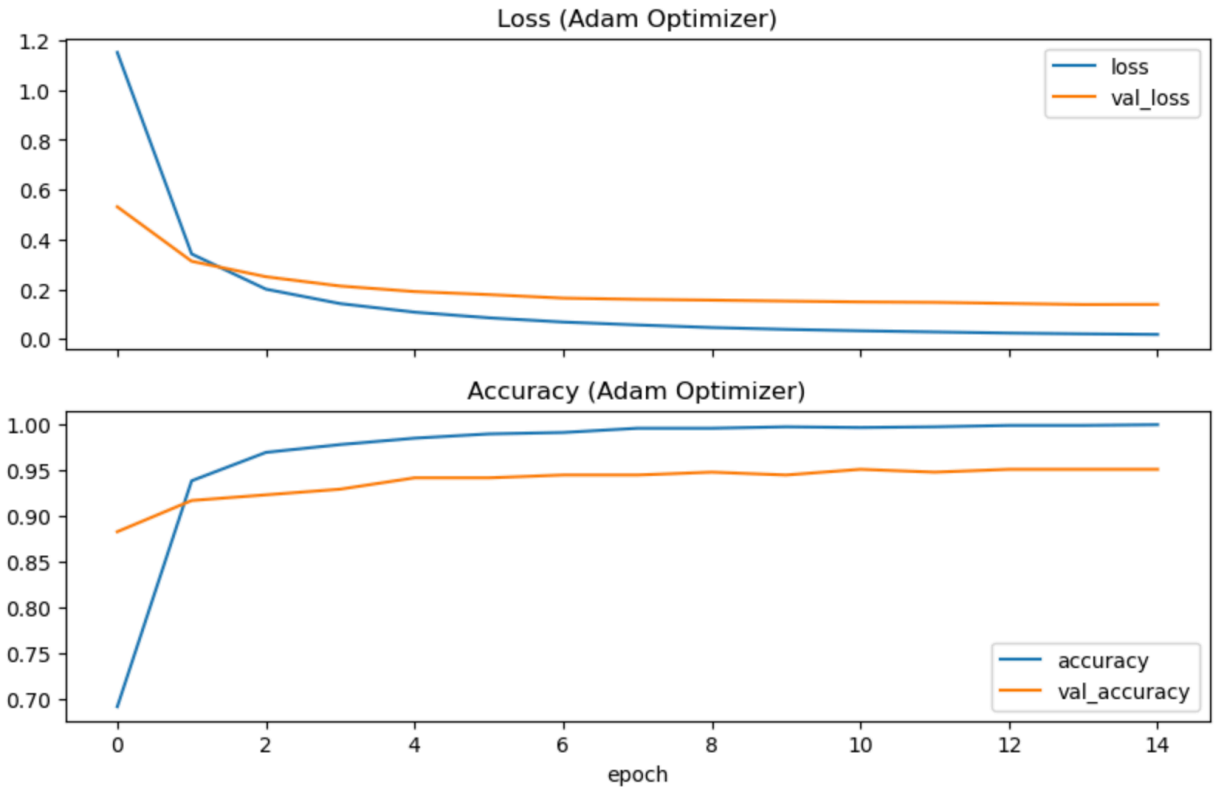




Impact of Optimizer

For the previous experiments I had been using the SGD optimizer. For this step, I decided to use the Adam optimizer as its known to work well with adaptive inputs. Adam is also well suited for faster convergence compared to SGD and its learning has an inbuilt auto adaptability which helps

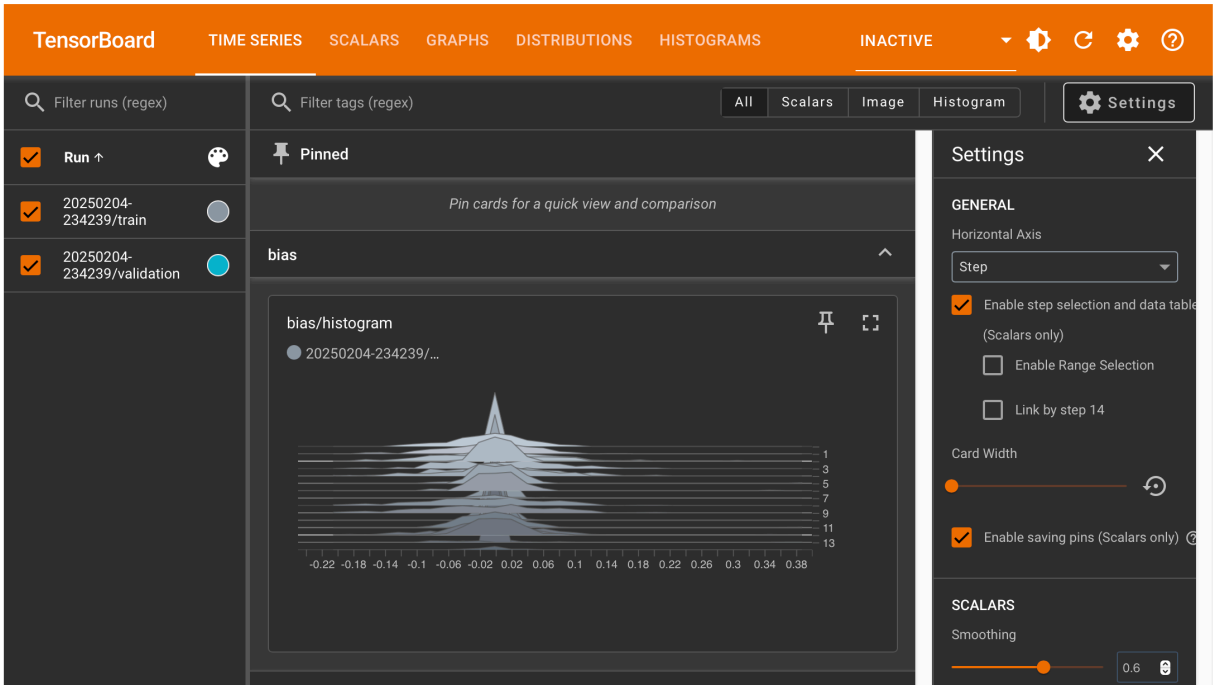
avoid underfitting. As expected Adam Optimizer performed much better than SGD with fewer parameter changes. Here's the convergence chart of Adam optimizer with default learning rate and 0.2 validation split with 2 hidden layers



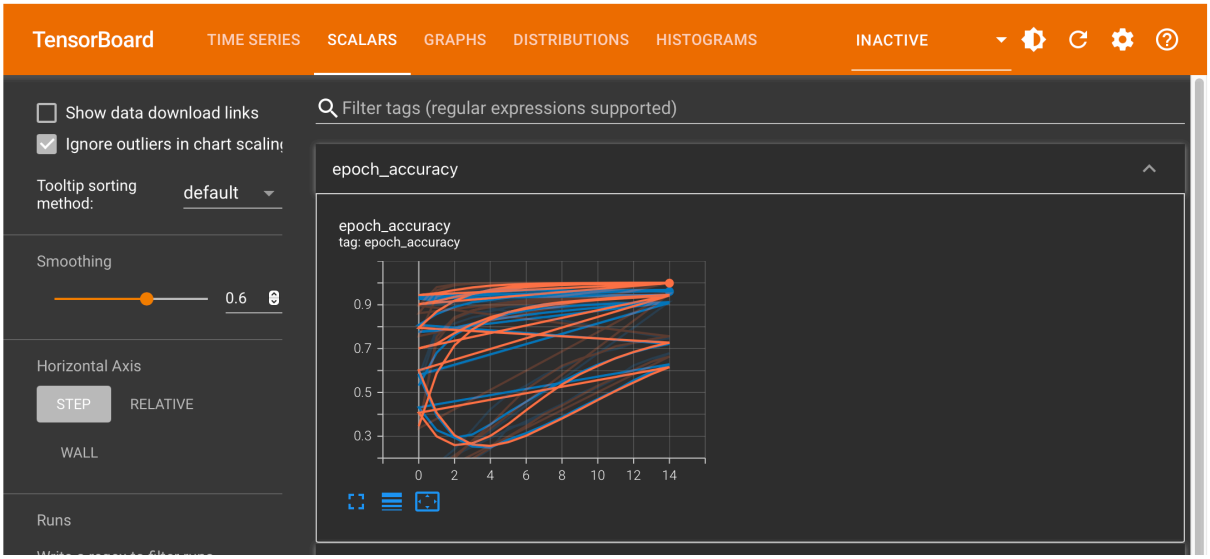
TensorBoard

Finally, I integrated tensor board into all the training steps to map out the performance of each architecture. Tensor Board is a metrics monitoring dashboard that comes in built with Keras. I used the histogram to effectively compare the different hyperparameter tuning effects. The chart can be seen here:

```
[4]: # Create tensorboard with histogram and epoch_accuracy
     %tensorboard --logdir tensorboard_logs --port=6007
```



```
[4]: # Create tensorboard with histogram and epoch_accuracy
%tensorboard --logdir tensorboard_logs --port=6007
```



Conclusion

After tuning the different hyper parameters, I feel that Adam optimizer is better than SGD optimizer for this kind of use case because of the adaptive nature of its input and the faster convergence for small datasets. When we're using SGD, we need to choose an optimal low learning rate since choosing a large learning rate will lead to extremely high divergence and overfitting of the model. Based on this we can conclude that choosing an optimal training and optimizing strategy is essential for having a good deep learning model.