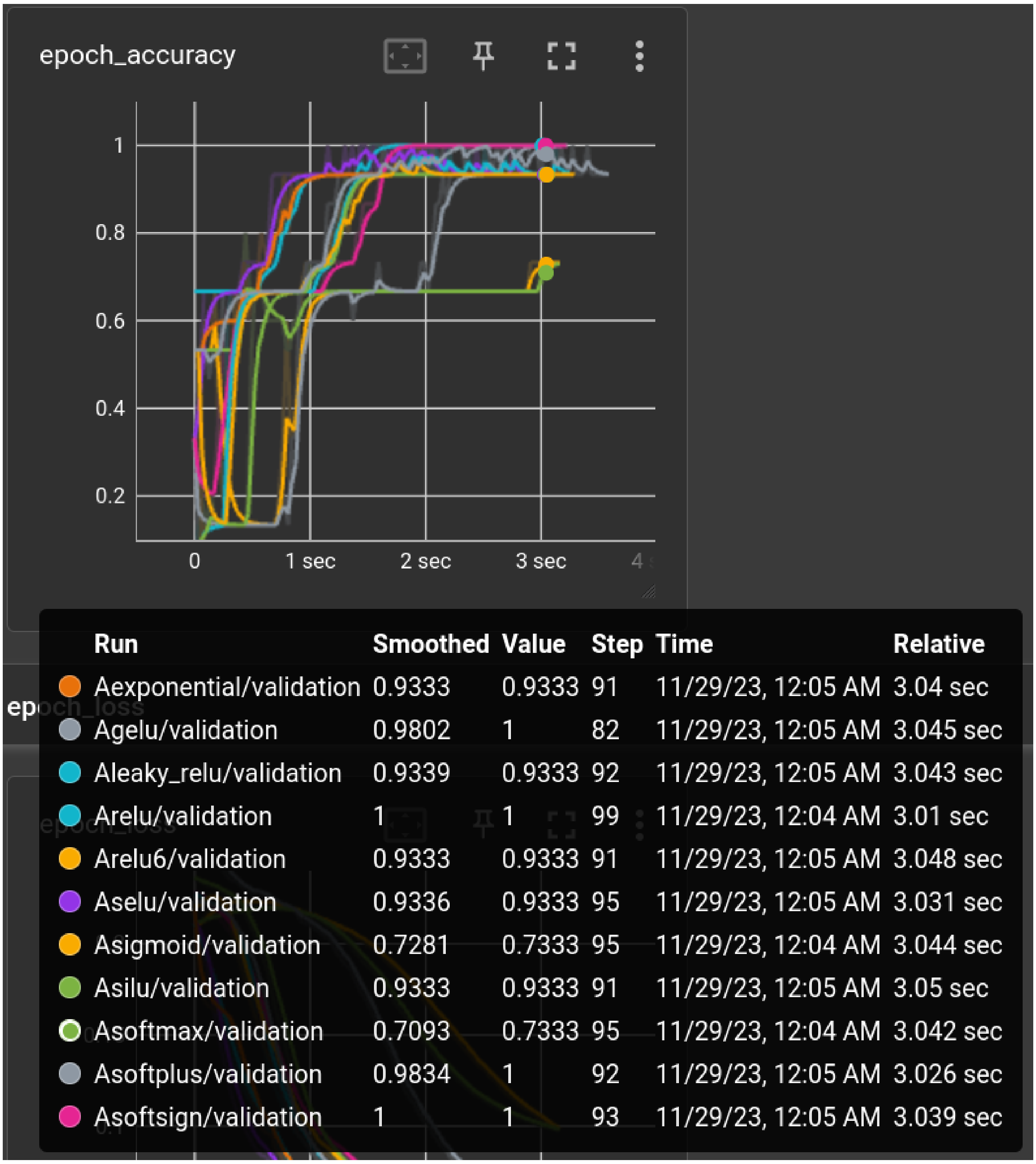
Multi Layer Networks

Similar to a single layer (0 hidden layers) network, a multi-layer network attempts to create a network such that when an input is passed through, it can accurately predict the output.

When before with a single layer network, each output would be the sum of weights multiplied by every characteristic (input) plus a bias. This number will then be fed through a non-linearity. Whilst in a single layer network, this is all that happens, for a multi layer network imagine the network deriving and tuning new characteristics from your original ones, then training the remaining of the model based on these created characteristics.

Multiple Non-linearities

Based off the first project, I discovered that the sigmoid often causes training issues where if the input data is far off the end of the sigmoid, the derivative will be so close to 0 that very little training will be done. All tanh, sigmoid, and softmax struggle from this issue as they have horizontal asymtotes. This is when I attempted to use the ReLU function, where $y = x$ when $x > 0$, but $y = 0$ when $x < 0$. This did not look promising to me as the left side of the nonlinearity would suffer from the same issue but worse: where the derivative would completely equal 0 meaning no training would be done. This is when I switched to a leaky ReLU for all the hidden layers as it does not suffer from the same diminishing derivative problem whilst still introducing a nonlinearity. For the last layer I still opted to use a softmax as the output vector should be a one-hot, and the softmax function scales outputs to roughly fit this requirement.



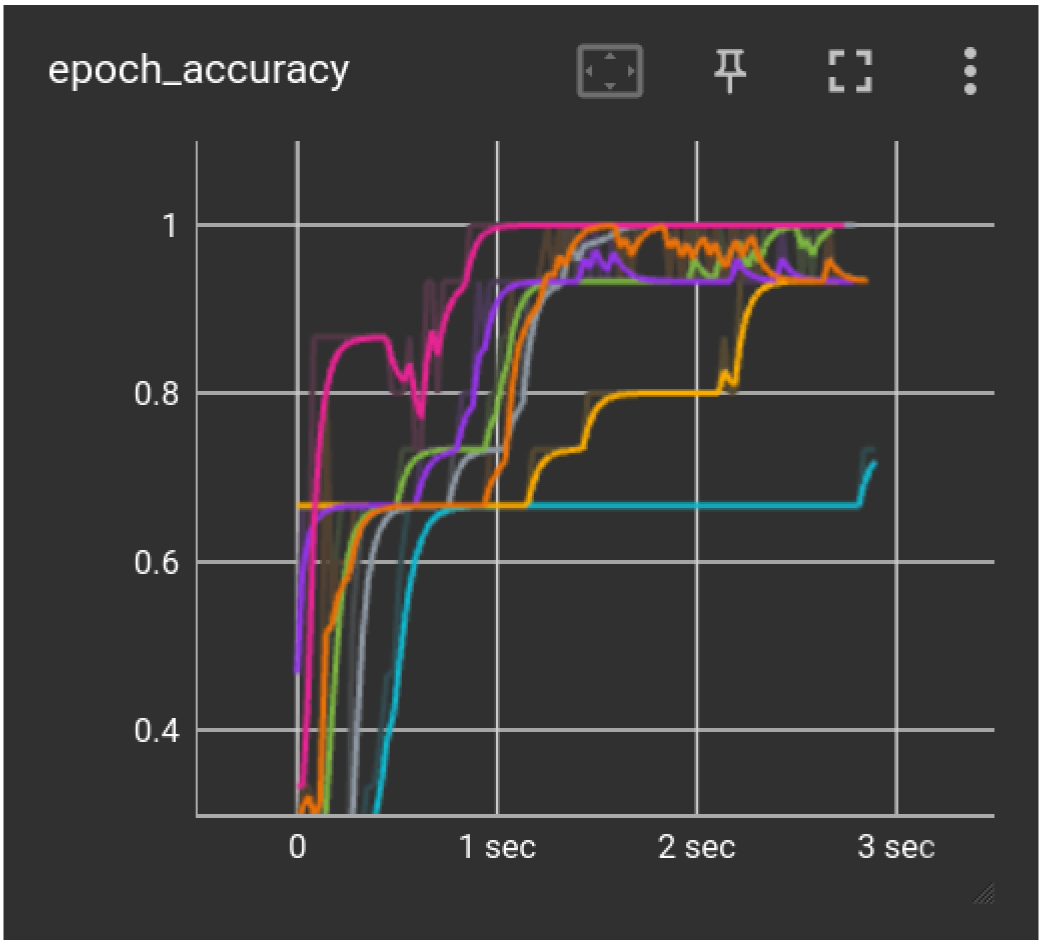
	Run	Smoothed	Value	Step	Time	Relative
●	Aexponential/validation	0.9333	0.9333	91	11/29/23, 12:05 AM	3.04 sec
●	Agelu/validation	0.9802	1	82	11/29/23, 12:05 AM	3.045 sec
●	Aleaky_relu/validation	0.9339	0.9333	92	11/29/23, 12:05 AM	3.043 sec
●	Arelu/validation	1	1	99	11/29/23, 12:04 AM	3.01 sec
●	Arelu6/validation	0.9333	0.9333	91	11/29/23, 12:05 AM	3.048 sec
●	Aselu/validation	0.9336	0.9333	95	11/29/23, 12:05 AM	3.031 sec
●	Asigmoid/validation	0.7281	0.7333	95	11/29/23, 12:04 AM	3.044 sec
●	Asilu/validation	0.9333	0.9333	91	11/29/23, 12:05 AM	3.05 sec
●	Asoftmax/validation	0.7093	0.7333	95	11/29/23, 12:04 AM	3.042 sec
●	Asoftplus/validation	0.9834	1	92	11/29/23, 12:05 AM	3.026 sec
●	Asoftsign/validation	1	1	93	11/29/23, 12:05 AM	3.039 sec

Here is the same dataset trained with an architecture of 8 and 8 nodes per layer, the adam optimizer, and various activator functions.

They all perform well except for the softmax and sigmoid functions. All in all, I still chose the leaky-ReLU as it appears to have the least training issues and most consistent accuracy growth.

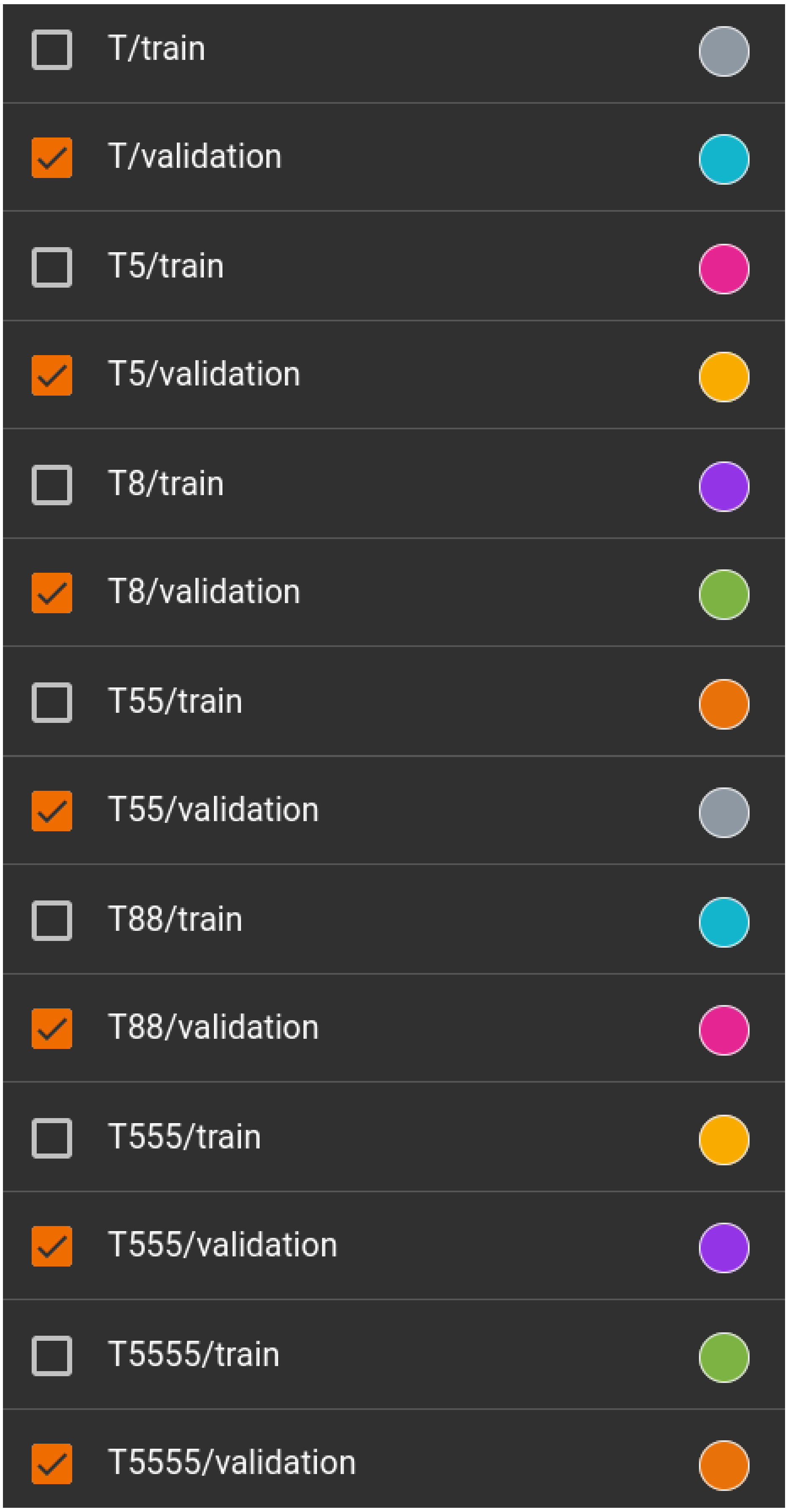
Multiple Network architectures

Here is a graph of time taken training vs. accuracy



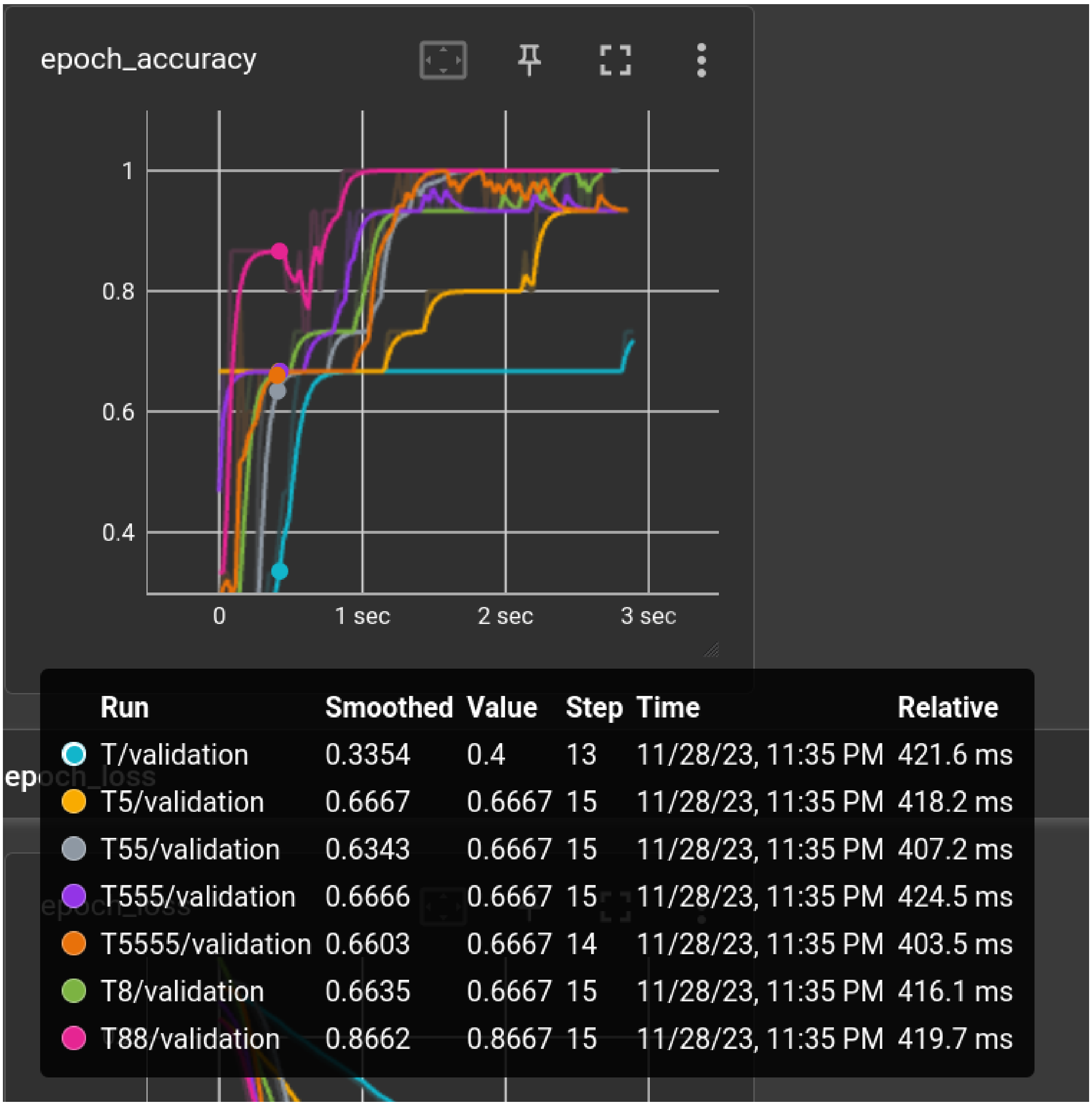
(Generated with TensorBoard)

With the following color scheme for different network architectures:



☐	T/train	●
✓	T/validation	●
☐	T5/train	●
✓	T5/validation	●
☐	T8/train	●
✓	T8/validation	●
☐	T55/train	●
✓	T55/validation	●
☐	T88/train	●
✓	T88/validation	●
☐	T555/train	●
✓	T555/validation	●
☐	T5555/train	●
✓	T5555/validation	●

As you can see in the data, the architecture of two hidden layers with size 8 and 8 generalized the fastest and also converged the fastest.



	Run	Smoothed	Value	Step	Time	Relative
●	T/validation	0.3354	0.4	13	11/28/23, 11:35 PM	421.6 ms
●	T5/validation	0.6667	0.6667	15	11/28/23, 11:35 PM	418.2 ms
●	T55/validation	0.6343	0.6667	15	11/28/23, 11:35 PM	407.2 ms
●	T555/validation	0.6666	0.6667	15	11/28/23, 11:35 PM	424.5 ms
●	T5555/validation	0.6603	0.6667	14	11/28/23, 11:35 PM	403.5 ms
●	T8/validation	0.6635	0.6667	15	11/28/23, 11:35 PM	416.1 ms
●	T88/validation	0.8662	0.8667	15	11/28/23, 11:35 PM	419.7 ms

It even beat a network with more layers but less characteristics per layer: 5555, and 555.

The worst performing in the bunch was the model with no hidden layers, it generalized slow and never converges to an acceptable error.

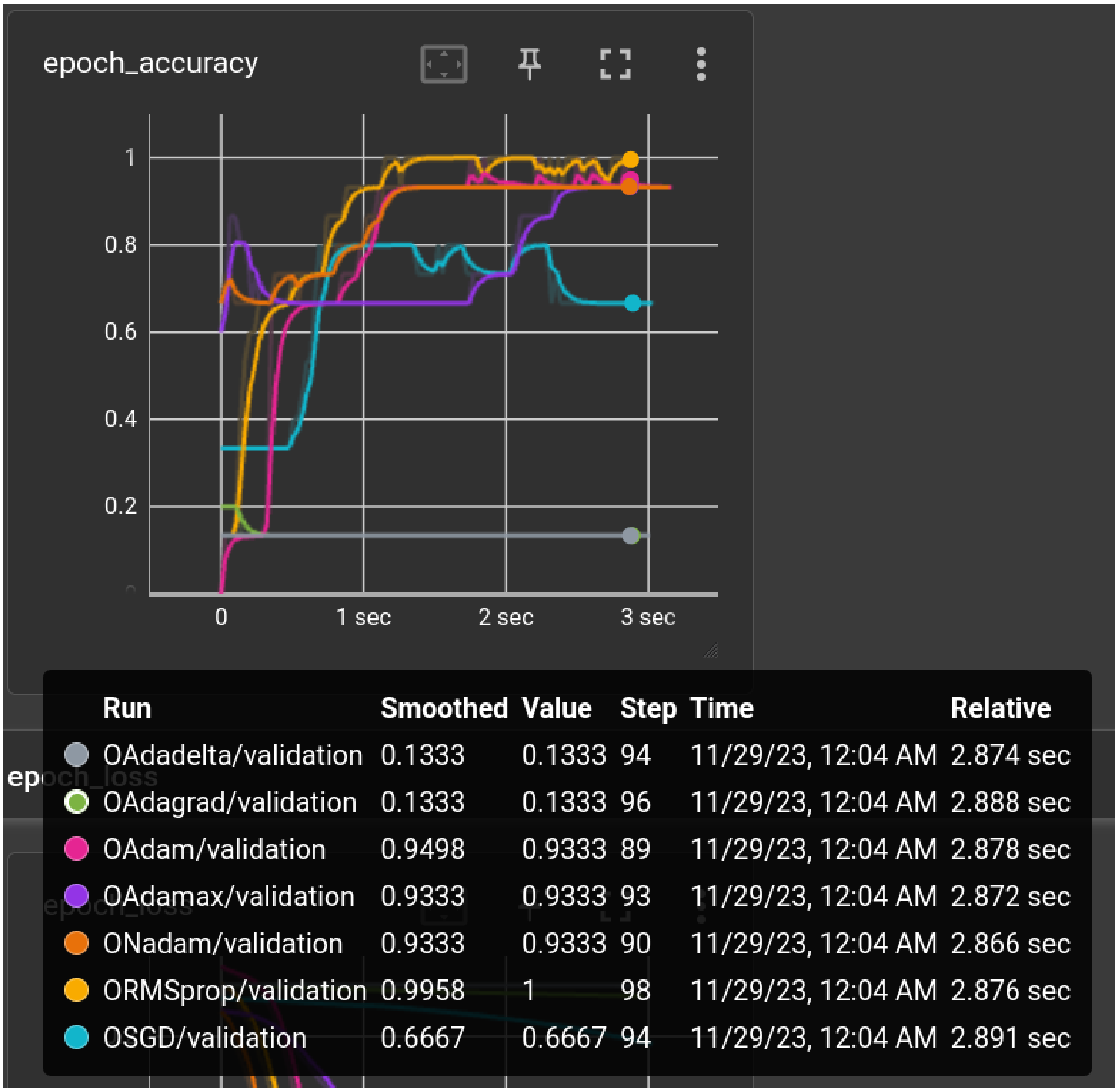
K-means clustering

K-means clustering will only ever be able to characterize models by their proximity to a generated mean. This means that K-means clustering is not possible to classify a dataset where different classes are not clustered around the same area, possibly intertwining and creating other patterns with the other classes.

Based off the previous K-means clustering on the iris dataset in P2, it quickly is able to find and create clusters on the dataset that roughly resemble that of the truth. Although it appears to have problems nearer to the decision boundaries where it will converge to a boundary that does not match that of the truth, because k-means clustering does not have feedback from known truths.

K-means clustering converges quickly and is easy to compute however, but will not offer as much accuracy as possible compared to neural networks. In the case of this dataset, since the classification boundaries are relatively simple, k-means works comparatively well, but NNs do perform better in a similar training time.

Optimizers / Algorithmic Learning Rate



	Run	Smoothed	Value	Step	Time	Relative
●	OAdadelta/validation	0.1333	0.1333	94	11/29/23, 12:04 AM	2.874 sec
●	OAdagrad/validation	0.1333	0.1333	96	11/29/23, 12:04 AM	2.888 sec
●	OAdam/validation	0.9498	0.9333	89	11/29/23, 12:04 AM	2.878 sec
●	OAdamax/validation	0.9333	0.9333	93	11/29/23, 12:04 AM	2.872 sec
●	ONadam/validation	0.9333	0.9333	90	11/29/23, 12:04 AM	2.866 sec
●	ORMSprop/validation	0.9958	1	98	11/29/23, 12:04 AM	2.876 sec
●	OSGD/validation	0.6667	0.6667	94	11/29/23, 12:04 AM	2.891 sec

After training the same model over all the different optimizers, with layers of 8 and 8, I found that the best optimizers were ORMSprop, Adam, then Nadam in that order.

Sources

https://keras.io/api/