MATH 198
Spring 2020

Exploring the Role of Activation Functions in Deep Learning

Gobind Puniani

May 28, 2020

Contents

1 Introduction

1.1 Background

Within the past decade, tremendous strides have been made in the field known as "deep learning." State-of-the-art results are often surpassed within a year or two, and progress in the field still remains steady. Indeed, "deep neural networks are the basis of the state-of-the-art results for image recognition, object detection, face recognition, speech recognition, machine translation, image caption generation, and driverless car technology" [4]. With so many potentially revolutionary technologies enabled by deep learning, we hope to contribute to this progress by helping to discover new ways to optimize the performance of artificial neural networks.

The artificial neural network is so named because each "neuron" delivers binary output; either a 1 or a 0 (activated or remaining inactive) is the result. Neurons can represent components of vectors, so that a network can be created with layers of neurons. Each neuron produces a real number to be used by the next layer, where each of those neurons creates its own weighted combination of values from the previous layer, adds its own bias, and applies the activation function. See Figure 1.1 for a visual diagram of a neural network, where w represents a weight vector and b represents a bias vector. The activation function determines whether a neuron passes output, and it also introduces non-linearity into the model [5]. Introducing non-linearity is important because linear classifiers may not be appropriate. A cost function is used to determine the category to classify the input data based on the values of the components of the final vector [1]. The network is optimized in a process called "training the network," in which the weights and biases are adjusted to minimize the cost function.

The intermediate layers between the input layer and the output layer are called "hidden layers." For simplicity, the diagram in Figure 1.1 only shows one hidden layer. Typically, many hidden layers are used in deep learning (hence the "deep" part). An optimization method known as "steepest descent" or "gradient descent" is an iterative process that tries to converge to a vector that minimizes the cost function. The "gradient" is the vector of partial derivatives, and the stepsize with each iteration is the "learning rate" [1]. The "stochastic gradient" method uses the gradient of a random training point instead of the mean of all gradients in order to save computational resources. A small sample mean could also be used to preserve accuracy.

Training requires the computation of partial derivatives with respect to each weight and bias for each neuron. The partial derivative of the cost at each neuron within each layer is known as the "error," although it is difficult to assign errors in hidden layers to errors in the final output [3]. In order to minimize the final cost, each partial derivative should be as close to zero as possible.

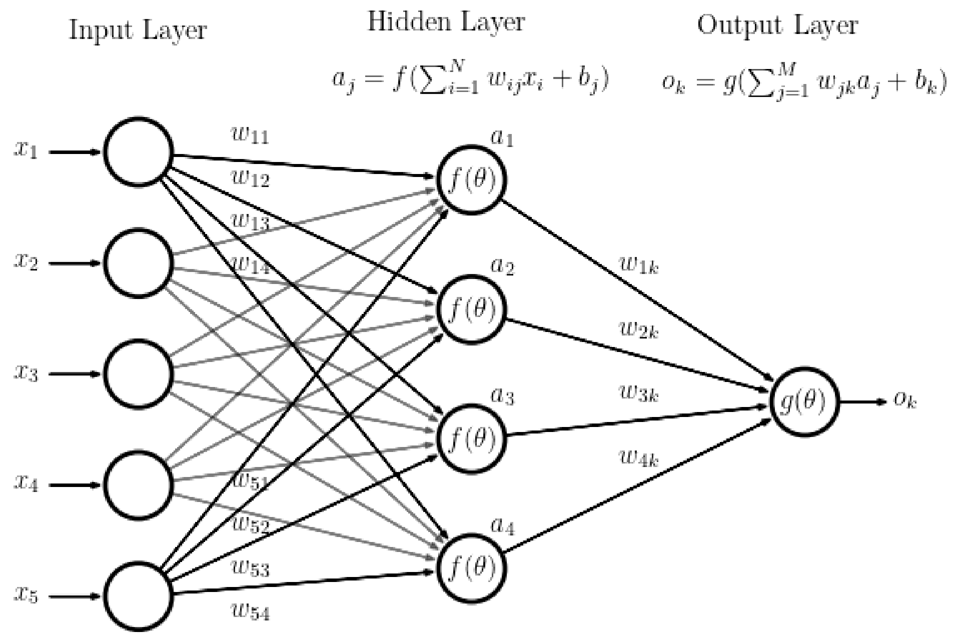

Figure 1.1: Depiction of a Neural Network

1.2 Objectives and Methods

Dr. Banuelos and his previous team of student researchers have created two new activation functions, and we are now testing the efficiency and accuracy of these against more traditional activation functions. Activation functions are generally overlooked, but they "strongly affect the network's speed of convergence, capacity, and overall performance" [5]. In the future, we may also consider the performance of neural networks that use hyperactivations, which have shown great promise.

2 Methods and Results

We ran our experiments on the CIFAR-10 dataset. The CIFAR-10 collection contains thousands of images, with 10 distinct categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck [2]. Each image is 32 by 32 pixels, which means each data point consists of 32 * 32 * 3 = 3,072 values [2]. The training set consists of 50,000 images and the validation set consists of 10,000 images (1,000 images for each category) [2]. An example of an image from each of the ten categories of CIFAR-10 is included below as Figures 2.1 - 2.10.

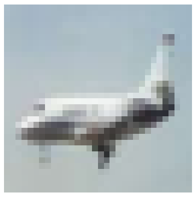

Figure 2.1: An example image of an airplane from CIFAR-10

Figure 2.2: An example image of an automobile from CIFAR-10

Figure 2.3: An example image of a bird from CIFAR-10

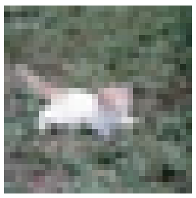

Figure 2.4: An example image of a cat from CIFAR-10

Figure 2.5: An example image of a deer from CIFAR-10

Figure 2.6: An example image of a dog from CIFAR-10

Figure 2.7: An example image of a frog from CIFAR-10

Figure 2.8: An example image of a horse from CIFAR-10

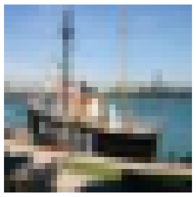

Figure 2.9: An example image of a ship from CIFAR-10

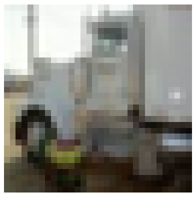
Figure 2.10: An example image of a truck from CIFAR-10

The activation function most popularly used in deep learning is ReLU (Rectified Linear Unit). Other activation functions are also commonly used, such as Swish and Tanh, but most researchers typically don't give much consideration to the choice of activation function. Last year, Dr. Banuelos and his team of research students created two new activation functions: TAct and mTAct. The TAct function is designed to be flexible, so it can emulate ReLU, Swish, or Tanh, depending on what seems to be the best for the model. The mTAct function is very similar, with just a slight modification in its parameter definitions.

All experiments were run on TLALOC (Training Learning Algorithms Landscape for Optimization and Computation), the server located in Dr. Banuelos' office. TLALOC has a much higher computational capacity than any personal computer, and it has dedicated GPU processing that could run these models uninterrupted. It often took anywhere from 1-4 hours to do a given run, so running all the experiments so far has probably taken several hundred hours.

2.1 Baseline Tests on fastai Models

There were a total of 12 runs for our baseline tests (four models each run at 25, 50, and 100 epochs). The four pre-trained models from the fastai library are xresnet18, xresnet34, xresnet50, and wrn22. The Python script to run each model was copied directly from the fastai library. Figure 2.11 depicts the accuracy of all four models over 25 epochs, Figure 2.12 depicts the accuracy of all four models over 50 epochs, and Figure 2.13 depicts the accuracy of all four models over 100 epochs. The accuracy at each epoch is an average value taken over five runs, and the windows on each graph depict the standard deviation at each epoch. All of the graphs in this paper were created using the Chartify library.

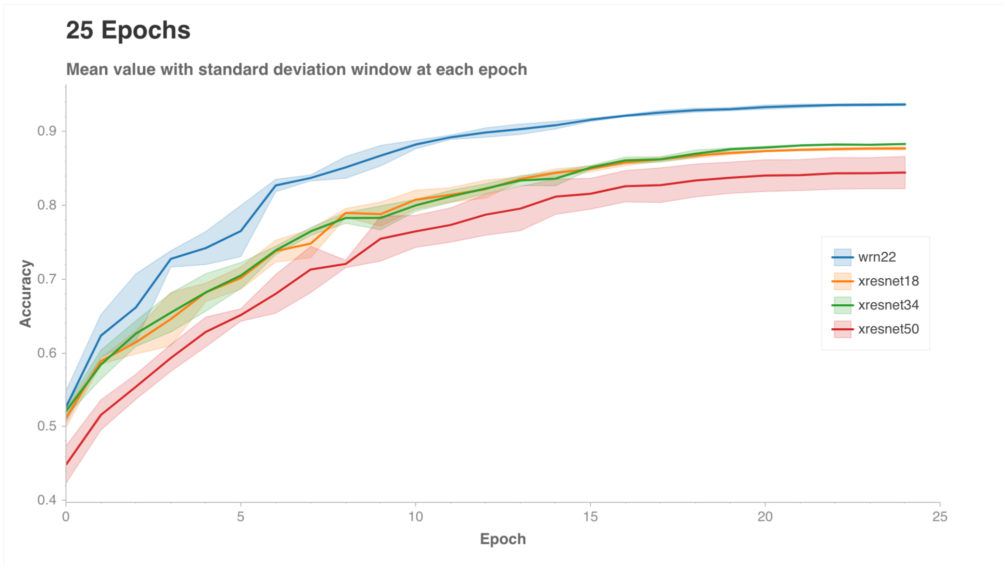

Figure 2.11: Baseline Plot across 25 epochs

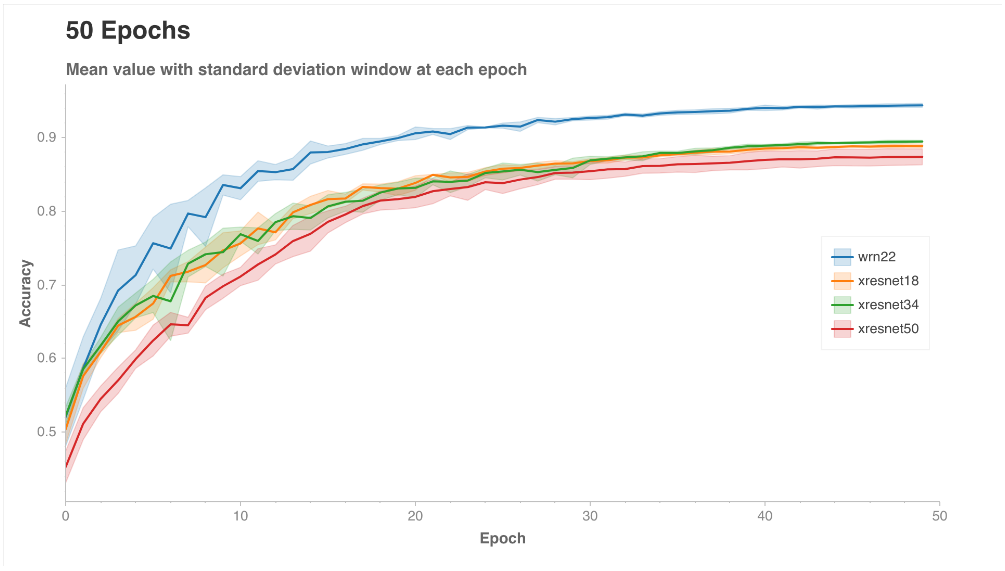

Figure 2.12: Baseline Plot across 50 epochs

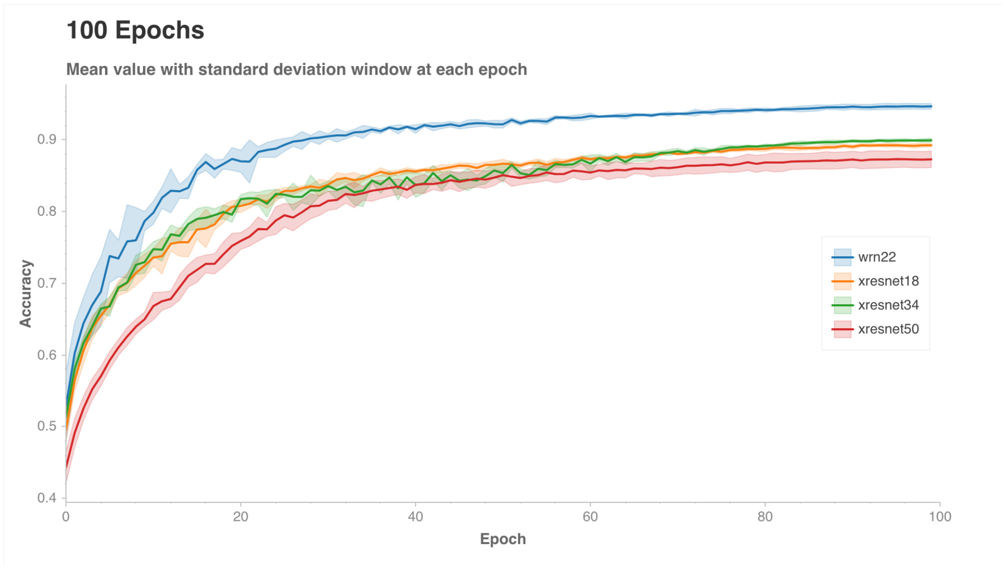

Figure 2.13: Baseline Plot across 100 epochs

Of the fastai library models, wrn22 clearly performed the best on all three epoch levels, while xresnet50 consistently lagged behind the rest. This disparity might be due to the “width” of the models [1]. This would explain why wrn22 achieved the highest accuracy (because each layer contains more neurons), whereas xresnet50 might be suffering from overfitting because of the extraneous layers (increased “depth” over “width”).

The baseline results using the fastai models allowed us to test TLALOC’s capabilities against those of the fastai team. However, all of those models were run with the ReLU activation function, so those tests are not very useful for the purposes of this project until we run the same models with different activation functions.

2.2 Testing on More Complex Models

Diganta Misra, whose paper has been cited in this project, created an activatation function called Mish, which is a modified version of Swish. Misra ran many experiments with many activation functions on many datasets and posted these results on his GitHub page (https://github.com/digantamisra98/Mish/tree/master/Notebooks). We borrowed Misra’s code for three models: DenseNet-121, SE-Net 18, and MobileNet v2. We then ran five activation functions at three times each: ReLU, Swish, Mish, TAct, and mTAct. Our results for ReLU, Swish, and Mish closely match those of Misra on the three models with the CIFAR-10 dataset. This would indicate that our results are reliable.

Table 2.1: Results for DenseNet-121

Activation	Test loss	Test accuracy	Test top-3 accuracy
Mish	0.498909	0.904964	0.985924
ReLU	**0.453389**	**0.910733**	**0.987507**
Swish	0.474808	0.908788	0.986452
TAct	0.485258	0.907700	0.986880
mTAct	0.475675	0.908986	0.987276

Table 2.2: Results for SE-Net 18

Activation	Test loss	Test accuracy	Test top-3 accuracy
Mish	0.500509	0.904964	**0.985858**
ReLU	0.502865	0.901833	0.984375
Swish	**0.496490**	**0.905558**	0.985727
TAct	0.613268	0.889076	0.980914
mTAct	0.677317	0.880142	0.978672

Table 2.3: Results for MobileNet v2

Activation	Test loss	Test accuracy	Test top-3 accuracy
Mish	0.542931	0.863199	0.974486
ReLU	0.527468	0.859935	0.975277
Swish	0.548409	0.866627	0.975541
TAct	**0.515840**	**0.867880**	**0.976200**
mTAct	0.534020	0.852420	0.971585

None of the activation functions dominated over all models. ReLU was best for DenseNet-121, Swish and Mish were best for SE-Net 18, and TAct was best for MobileNet v2. Nearly all activation functions performed comparably across all models, with the exception of TAct and mTAct on SE-Net 18. The relatively poor performance of these two activation functions on SE-Net 18 may be due to a slight incompatibility in coding architecture that was not present on the other two models.

Next, we tested DenseNet-121, SE Net-18, and MobileNet v2 with different learning rates. Our original tests of these three models used the learning rate of 0.0010, so we tried four more learning rates that differed by orders of magnitude: 0.0001, 0.0100, 0.1000, and 1.0000. We wanted to see if any particular learning rate was best suited for each of the models, and we were interested to discover whether any of the activation functions combined with a particular learning rate to yield the best accuracy performances.

After running each of the three models on each of the five activation functions on each of the four new learning rate rates three times each, we found that our original learning rate of 0.0010 was optimal for nearly all activation functions on DenseNet-121 and SE Net-18. Occasionally, the learning rates closest to 0.0010, namely 0.0001 and 0.0100, were best for some activation functions in these two models. The highest learning rate tested (1.0000) almost always yielded the worst results, which is in accordance with Misra's claim that these three complex models typically perform better with small learning rates [3]. Although the original learning rate produced consistently high accuracy values among the activation functions in MobileNet v2, the learning rate of 0.0100 seemed to produce slightly better results, making it the best learning rate for that model. But it is quite surprising to see that the next learning rate above 0.0100 (namely, 0.1000) yields such dramatically low results, as seen in Tables 2.6 and 2.7. The following six tables (Tables 2.4 - 2.9) summarize the results for our tests on varying learning rates for each of the three models on test top-1 accuracy and test top-3 accuracy. For CIFAR-10 data, test top-1 accuracy refers to the accuracy of the top prediction (as a percentage) being correct in assigning an input image from the testing set to the correct category, while test top-3 accuracy refers to the accuracy for any of the top 3 predictions (in percentages) being the correct image category for a test image. In these tables, the best result in each column (for an activation function) is italicized, while the best result in each row (for a learning rate) is in bold. From these results, it seems that the learning rate plays a more significant role than the activation function, since no activation function is capable of compensating for a deficient learning rate.

Table 2.4: Results for DenseNet-121 Learning Rate Tests (Test Top-1 Accuracy)

Learning Rate	Mish	ReLU	Swish	TAct	mTAct
0.0001	0.903349	0.900481	0.904437	0.904635	**0.906909**
0.0010	*0.904964*	***0.910733***	*0.908788*	*0.907700*	*0.908986*
0.0100	0.882021	0.877637	0.882351	**0.885153**	0.877703
0.1000	0.862902	**0.885614**	0.867220	0.874571	0.885450
1.0000	0.250494	0.099848	**0.311940**	0.100046	0.100178

Table 2.5: Results for DenseNet-121 Learning Rate Tests (Test Top-3 Accuracy)

Learning Rate	Mish	ReLU	Swish	TAct	mTAct
0.0001	***0.986056***	0.984342	0.985265	0.985825	0.985661
0.0010	0.985924	***0.987507***	*0.986452*	*0.986880*	*0.987276*
0.0100	0.979397	0.977749	0.980749	**0.981145**	0.980024
0.1000	0.974585	0.980353	0.977024	0.978277	**0.981309**
1.0000	0.499637	0.299941	**0.507384**	0.299215	0.299875

Table 2.6: Results for MobileNet v2 Learning Rate Tests (Test Top-1 Accuracy)

Learning Rate	Mish	ReLU	Swish	TAct	mTAct
0.0001	0.738133	0.718585	0.725541	0.770866	**0.776174**
0.0010	0.863199	0.859935	0.866627	***0.867880***	0.852420
0.0100	***0.880505***	*0.862342*	*0.879747*	0.866858	*0.862540*
0.1000	0.178995	0.157404	**0.260252**	0.122594	0.098233
1.0000	**0.101925**	0.099585	0.099288	0.099914	0.099255

Table 2.7: Results for MobileNet v2 Learning Rate Tests (Test Top-3 Accuracy)

Learning Rate	Mish	ReLU	Swish	TAct	mTAct
0.0001	0.929688	0.923259	0.923919	0.943730	0.947191
0.0010	0.974486	0.975277	0.975541	**0.976200**	0.971585
0.0100	***0.979727***	*0.976431*	*0.978540*	*0.977255*	*0.974321*
0.1000	0.415084	0.440961	**0.476266**	0.324862	0.222805
1.0000	0.299974	**0.300831**	0.298919	0.298820	0.297139

Table 2.8: Results for SE Net-18 Learning Rate Tests (Test Top-1 Accuracy)

Learning Rate	Mish	ReLU	Swish	TAct	mTAct
0.0001	**0.882747**	0.874637	0.881659	0.876846	0.874604
0.0010	*0.904964*	*0.901833*	***0.905558***	*0.889076*	*0.880142*
0.0100	0.883801	0.886142	**0.887889**	0.879219	0.878165
0.1000	0.861650	**0.868473**	0.829608	0.832311	0.710311
1.0000	0.099585	0.100508	0.099420	0.098563	**0.157470**

Table 2.9: Results for SE Net-18 Learning Rate Tests (Test Top-3 Accuracy)

Learning Rate	Mish	ReLU	Swish	TAct	mTAct
0.0001	**0.980320**	0.977683	0.978573	0.977453	0.976925
0.0010	***0.985858***	*0.984375*	*0.985727*	*0.980914*	0.978672
0.0100	0.980353	**0.981705**	0.979694	0.978771	*0.979661*
0.1000	0.972211	**0.973662**	0.964662	0.963739	0.867979
1.0000	0.299875	0.301325	0.300666	0.310160	**0.427314**

The charts for the two best learning rates on each of the three models averaged over three runs each are included below as Figures 2.14 - 2.19.

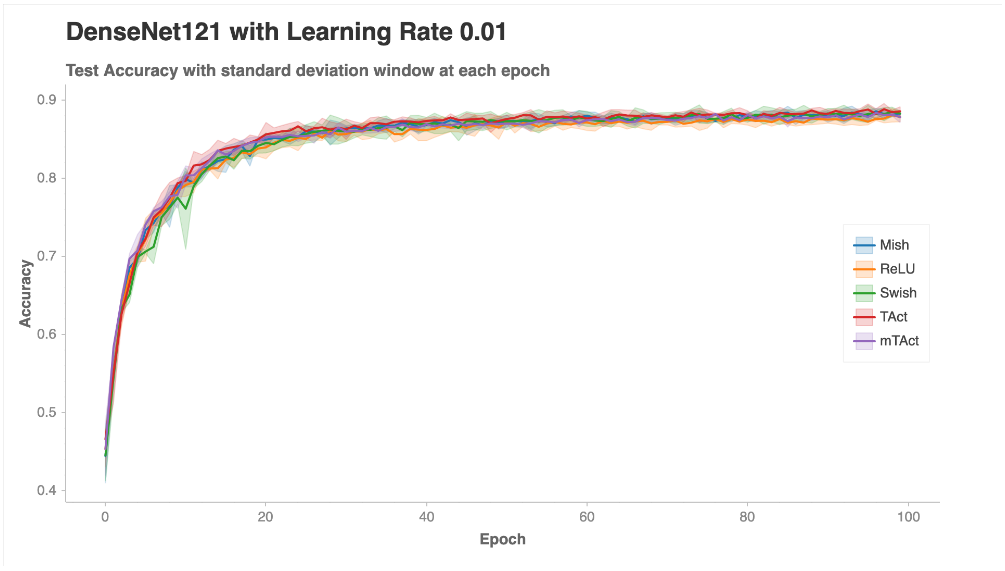

Figure 2.14: DenseNet-121 with a Learning Rate of 0.0100

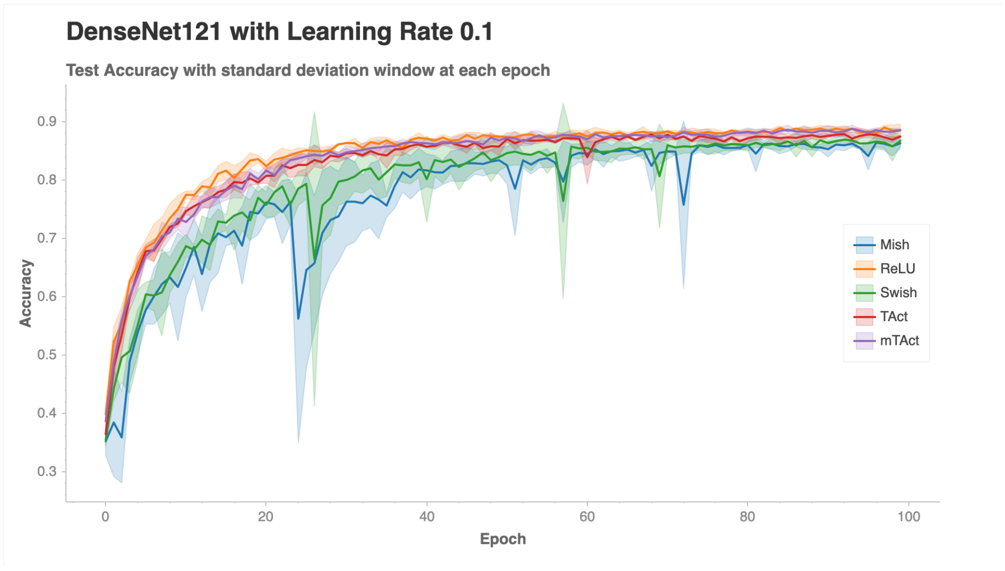

Figure 2.15: DenseNet-121 with a Learning Rate of 0.1000

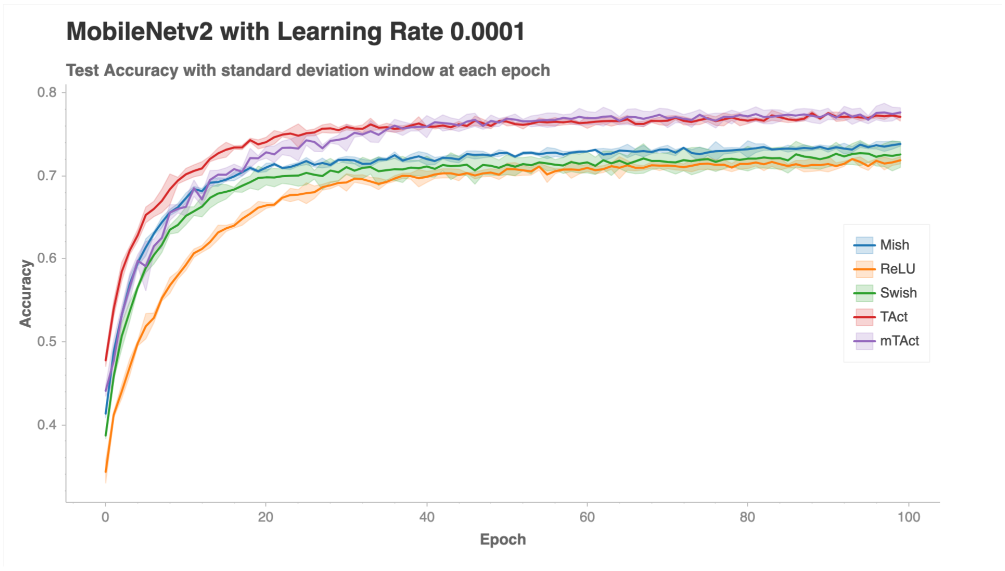

Figure 2.16: MobileNet v2 with a Learning Rate of 0.0001

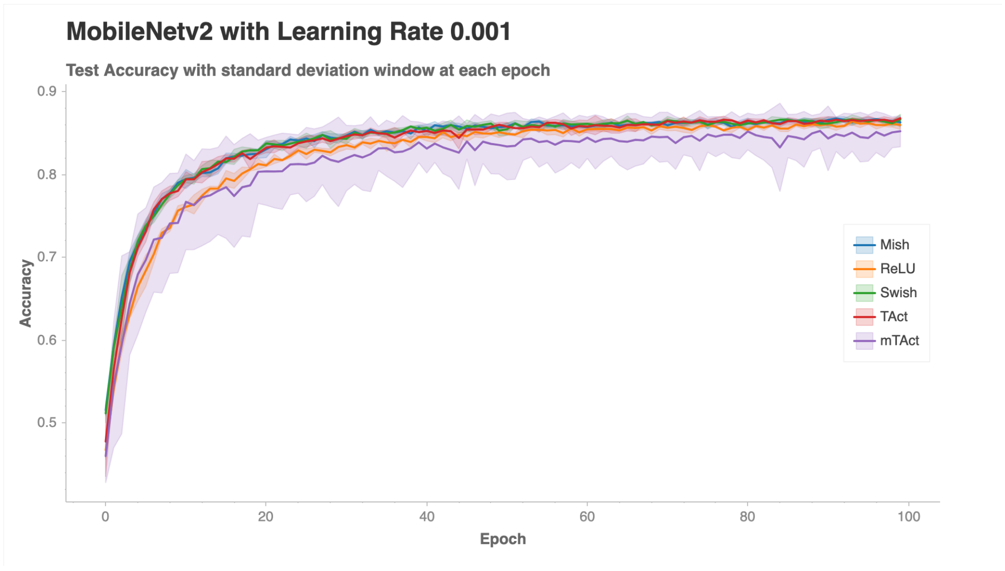

Figure 2.17: MobileNet v2 with a Learning Rate of 0.0010

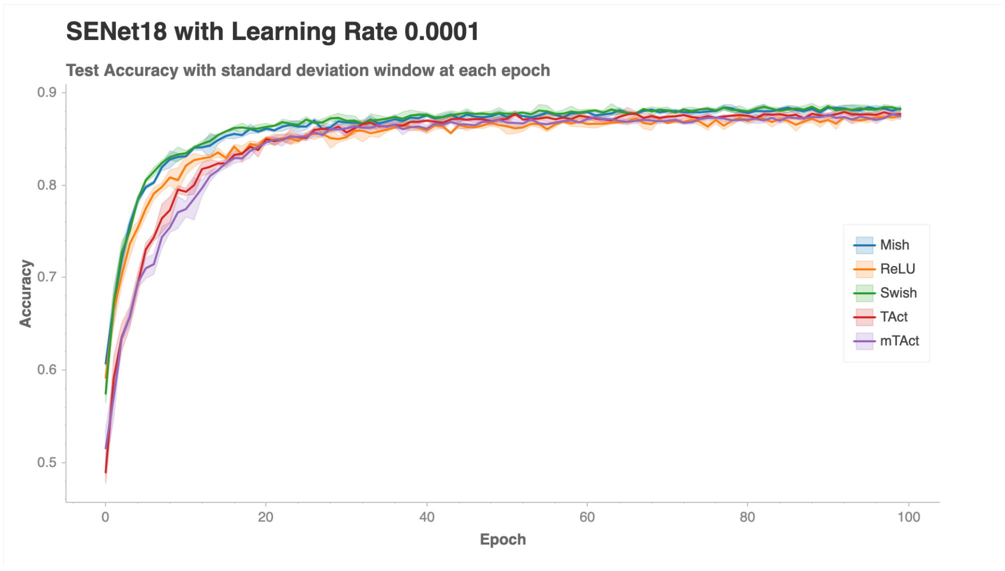

Figure 2.18: SE Net-18 with a Learning Rate of 0.0001

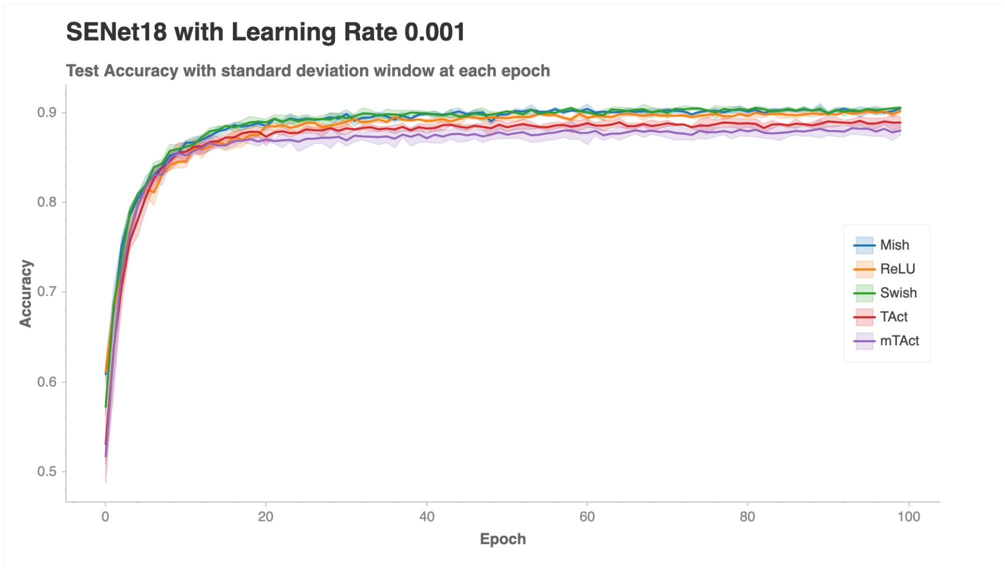

Figure 2.19: SE Net-18 with a Learning Rate of 0.0010

2.3 Simple ResNet-18 Model

We achieved similar results among all five activation functions with the three complex models (DenseNet-121, SE-Net 18, and MobileNet v2). At this point, we decided to revise our hypothesis: perhaps TAct and mTAct perform better on shallower, simpler models. Such models would have fewer layers and fewer parameters, which might convey an advantage to TAct and mTAct, which are more flexible in nature than the other activation functions. We constructed a simple ResNet-18 model using code from the PyTorch website and then modified it to accommodate our activation functions.

Our hypothesis turned out to be correct in the case of using a triangular learning rate, as shown in Figure 2.20. None of the graphs of the Simple ResNet-18 model results include standard deviation windows to improve visibility.

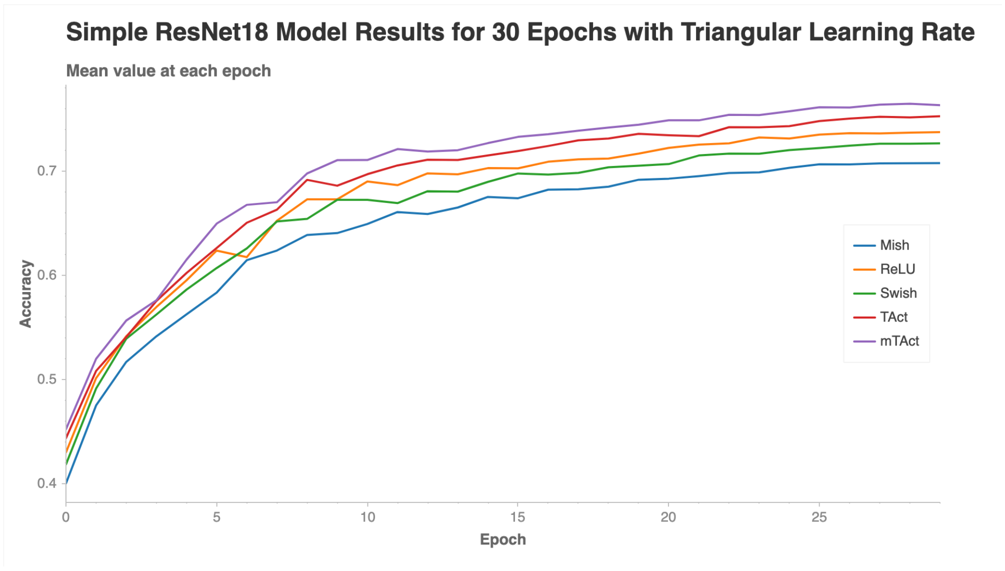

Figure 2.20: Simple ResNet 18 Plot with 30 epochs and Triangular Learning Rate, where mTAct and TAct perform the best (average of 3 runs)

We tested two versions of the simple model: one with a triangular learning rate and one with a fixed learning rate. The triangular learning rate is a type of cyclical learning rate policy in which a model is run for a few epochs with a learning rate varying linearly between a minimum value and a maximum value in order to find the optimal value for the actual run [4]. A fixed learning rate policy, on the other hand, simply uses a constant value for the duration of the run. While mTAct and TAct performed the best, respectively, on the Simple ResNet18 model with a triangular learning rate policy, TAct surpassed mTAct as best performer on a fixed learning rate policy (Figure 2.21).

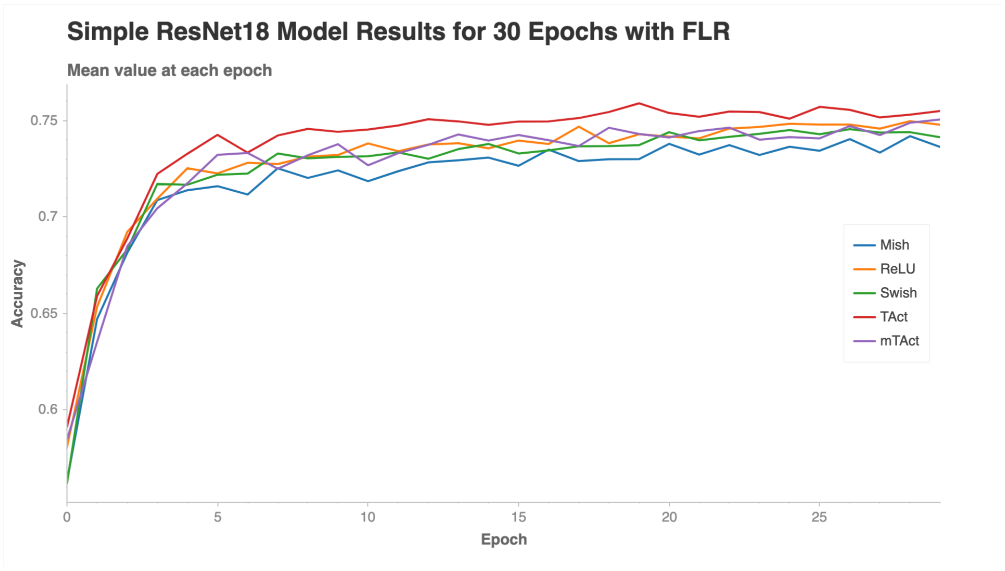

Figure 2.21: Simple ResNet 18 Plot with 30 epochs and Fixed Learning Rate (average of 3 runs)

These results were promising for TAct and mTAct, because our revised hypothesis seemed to be correct.

2.4 Testing with Noisy Datasets

We next turned our attention to noisy datasets. Manuel Chacón, a collaborator on this research project, developed four corrupted versions of the CIFAR-10 dataset. The four noisy datasets are psi8, psi16, psi32, and psi64. The psi coefficient indicates the degree of noise/corruption introduced to the dataset, with psi8 possessing the most amount of noise in its images and psi64 possessing the least amount. Examples of the corrupted images are included below as Figures 2.22 - 2.29.

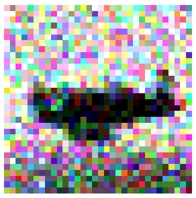

Figure 2.22: A corrupted image depicting an airplane, from the psi8 dataset

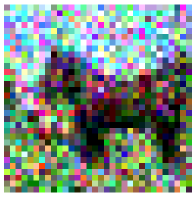

Figure 2.23: A corrupted image depicting a horse, from the psi8 dataset

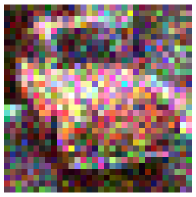

Figure 2.24: A corrupted image depicting a car, from the psi16 dataset

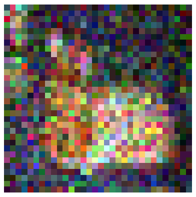

Figure 2.25: A corrupted image depicting a cat, from the psi16 dataset

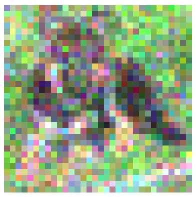

Figure 2.26: A corrupted image depicting a dog, from the psi32 dataset

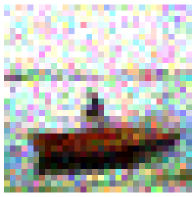

Figure 2.27: A corrupted image depicting a ship, from the psi32 dataset

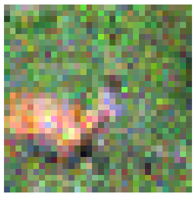

Figure 2.28: A corrupted image depicting a deer, from the psi64 dataset

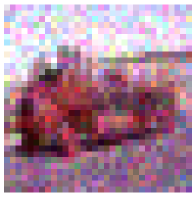

Figure 2.29: A corrupted image depicting a truck, from the psi64 dataset

First, we tested our Simple ResNet-18 model with the noisy datasets. There were two learning policies (Triangular Learning Rate and Fixed Learning Rate) and four noisy datasets (psi8, psi16, psi32, and psi64) for each of the five activation functions, which were averaged over three runs with 30 epochs per run. The graphs are displayed in Figures 2.30 - 2.37.

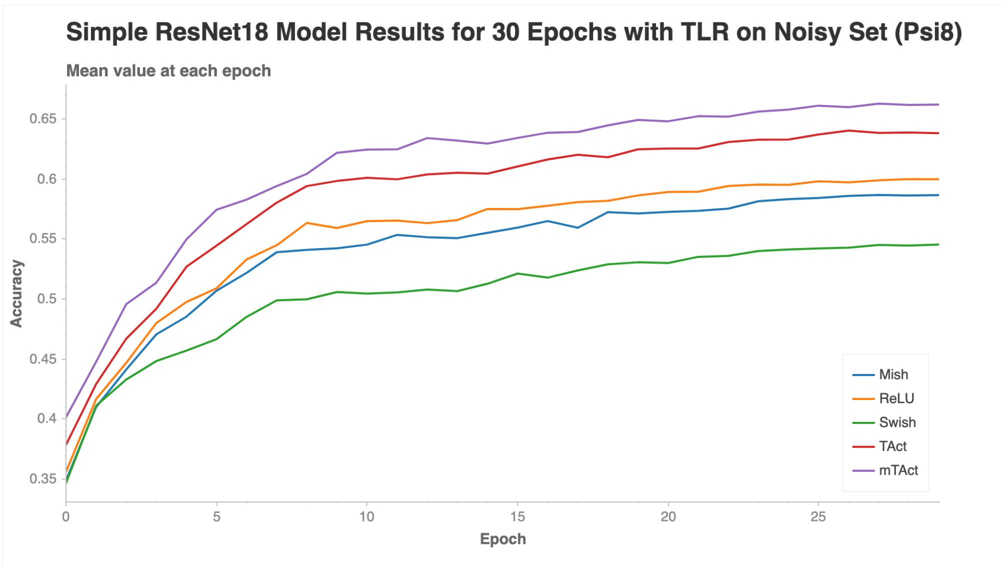

Figure 2.30: Simple ResNet-18 on the psi8 dataset with Triangular Learning Rate

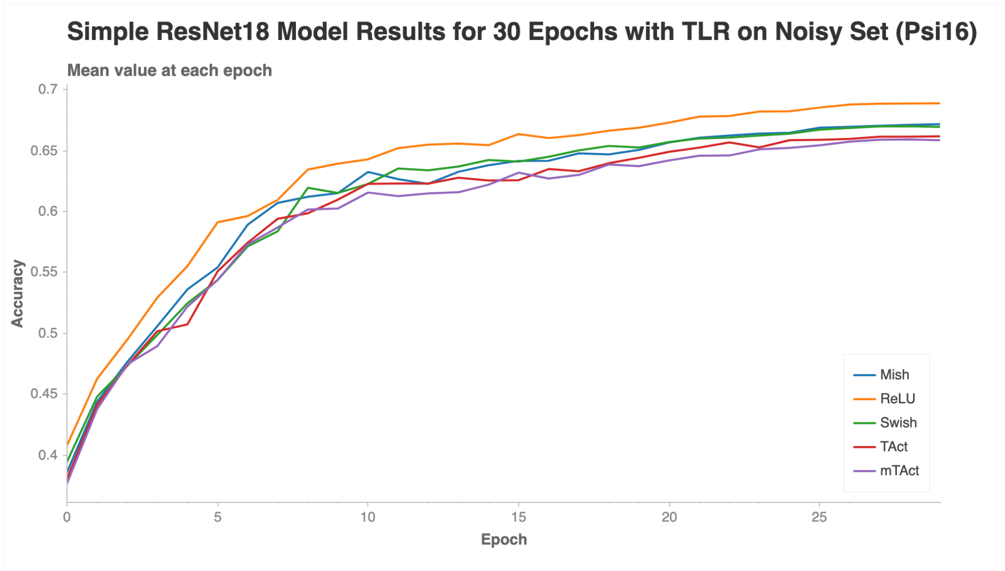

Figure 2.31: Simple ResNet-18 on the psi16 dataset with Triangular Learning Rate

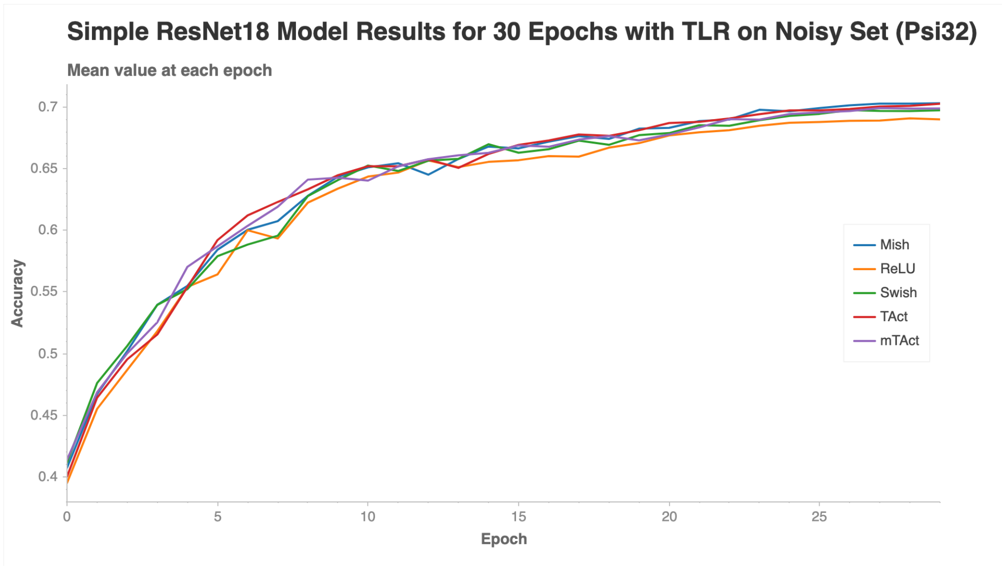

Figure 2.32: Simple ResNet-18 on the psi32 dataset with Triangular Learning Rate

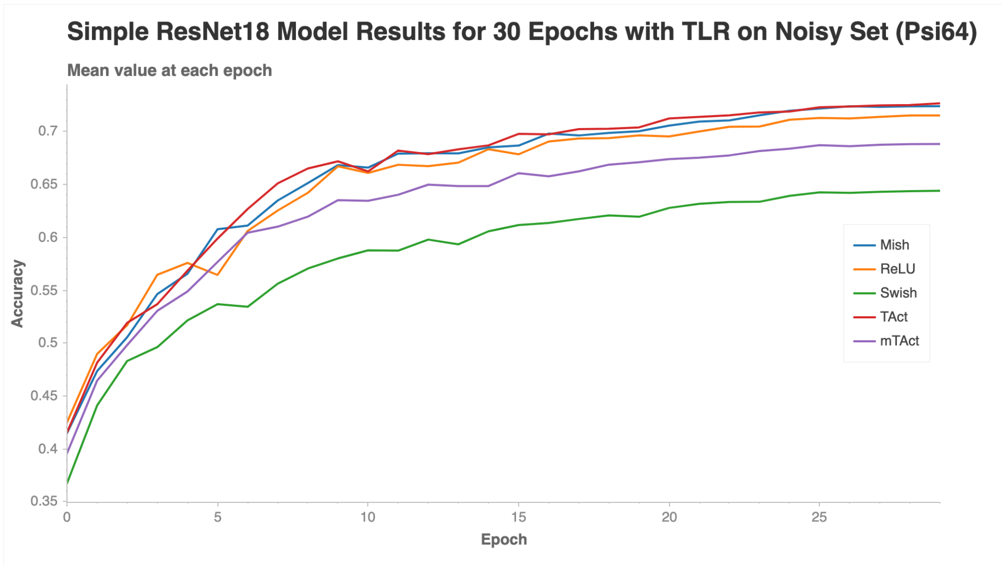

Figure 2.33: Simple ResNet-18 on the psi64 dataset with Triangular Learning Rate

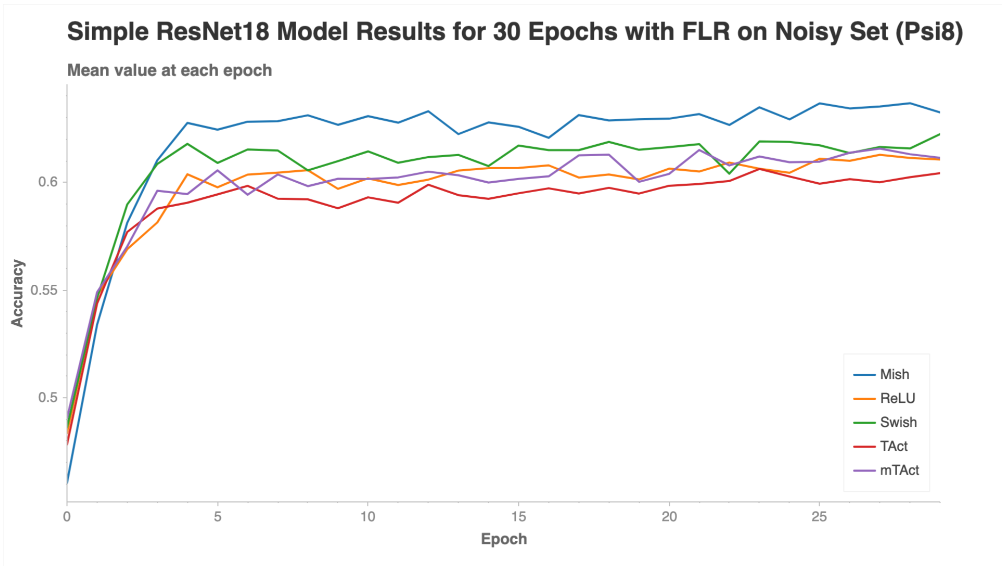

Figure 2.34: Simple ResNet-18 on the psi8 dataset with Fixed Learning Rate

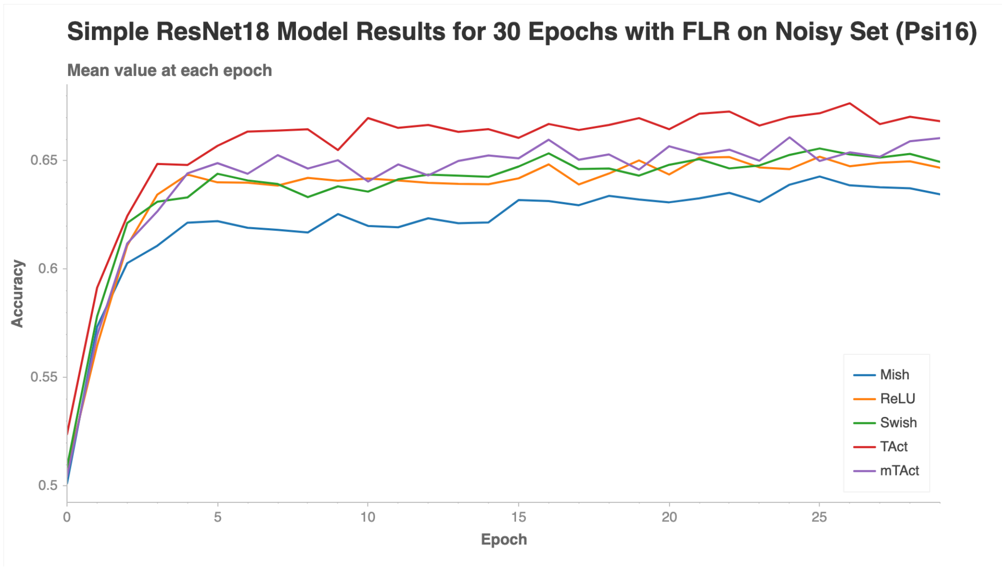

Figure 2.35: Simple ResNet-18 on the psi16 dataset with Fixed Learning Rate

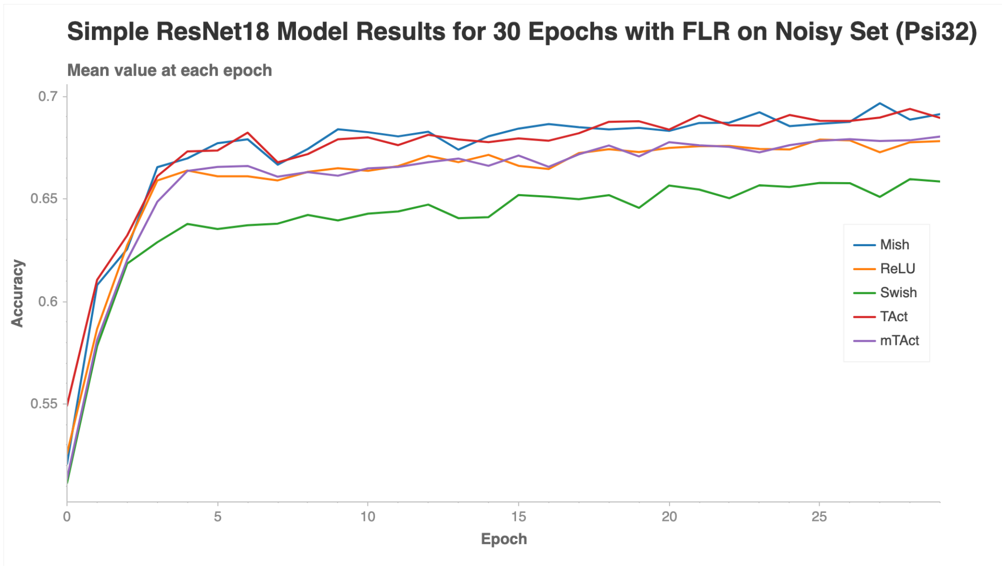

Figure 2.36: Simple ResNet-18 on the psi32 dataset with Fixed Learning Rate

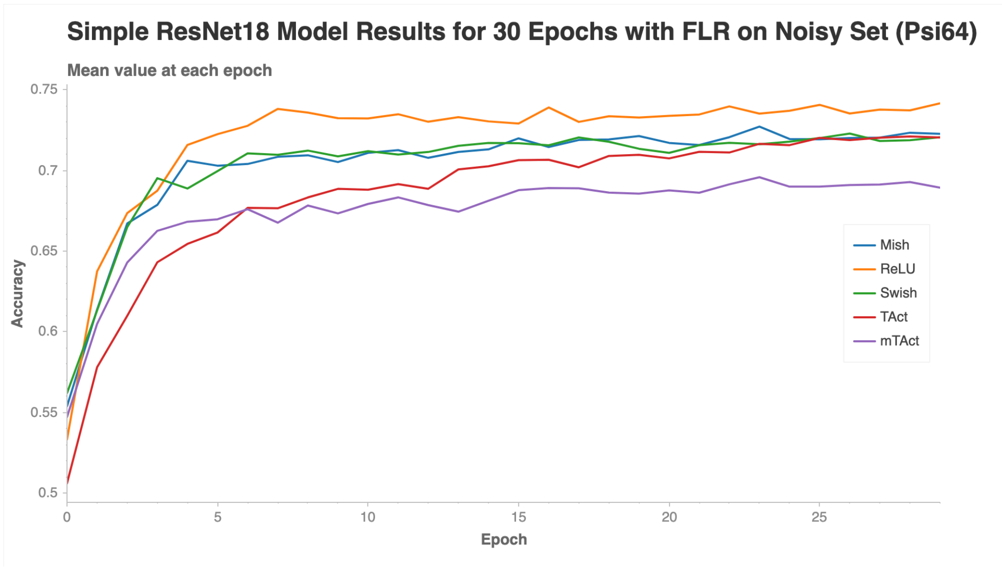

Figure 2.37: Simple ResNet-18 on the psi64 dataset with Fixed Learning Rate

It seems that there is no consistent pattern in the behavior of activation functions with respect to degree of noise corruption or learning rate policy. Thus, none of the activation functions prevailed over the others on the simple architecture with the corrupted images.

We then decided to test the noisy data on the complex models that we had previously studied: DenseNet-121, MobileNet v2, and SE Net-18. All five activation functions were tested on each of these three models with each of the four noisy datasets. The results of these tests (averaged over 5 runs per activation function per model per dataset) are included in the 12 graphs below (Figures 2.38 - 2.49).

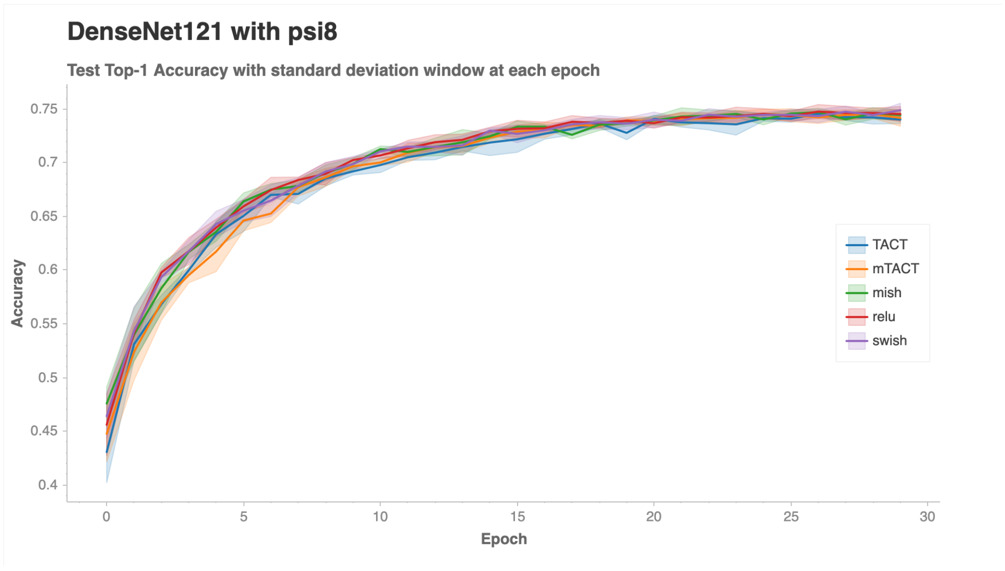

Figure 2.38: DenseNet-121 on the psi8 dataset

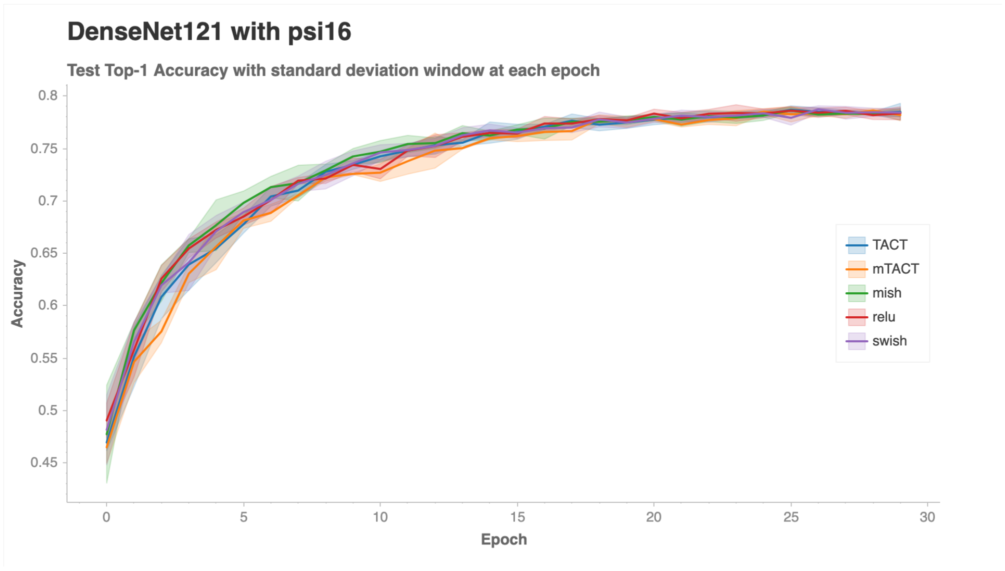

Figure 2.39: DenseNet-121 on the psi16 dataset

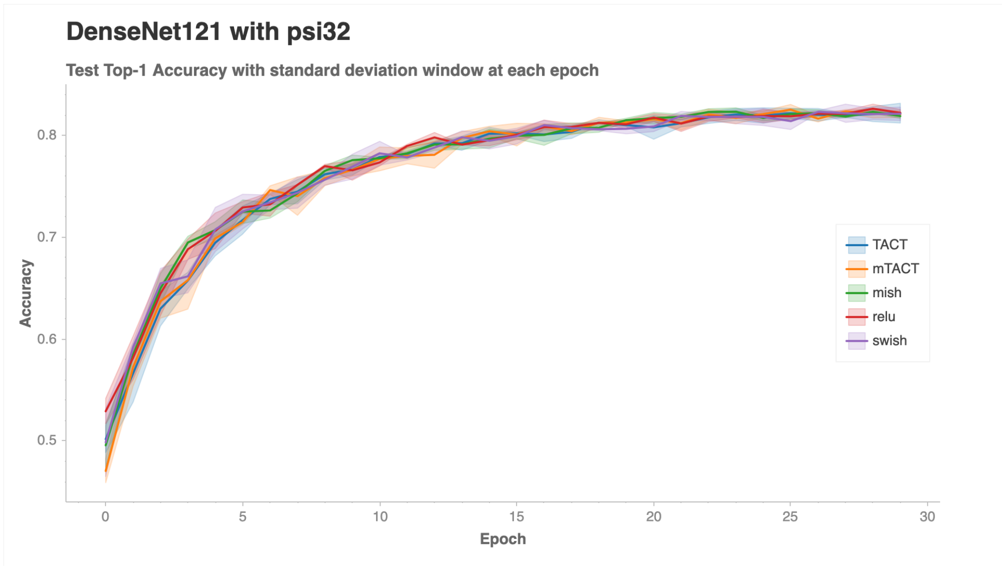

Figure 2.40: DenseNet-121 on the psi32 dataset

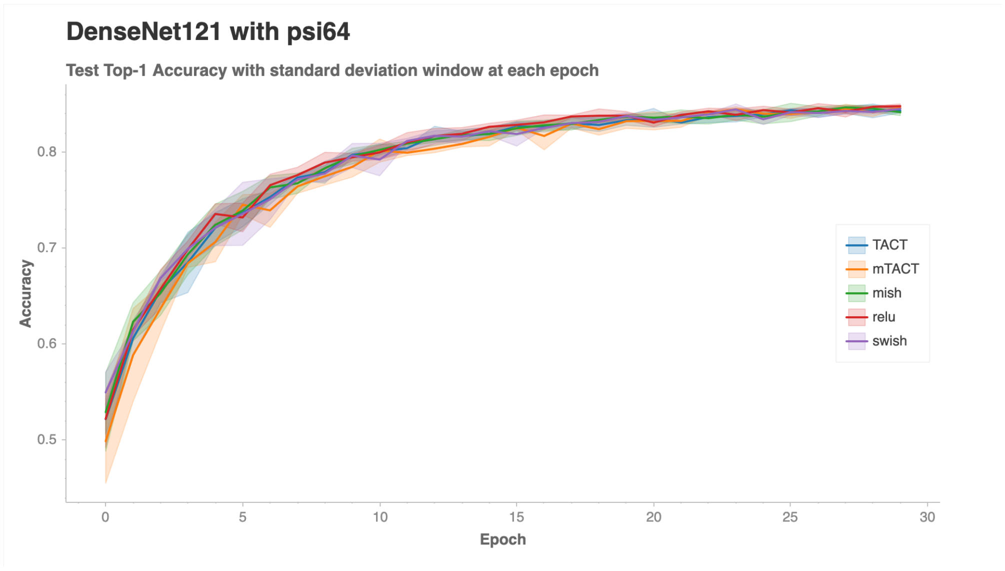

Figure 2.41: DenseNet-121 on the psi64 dataset

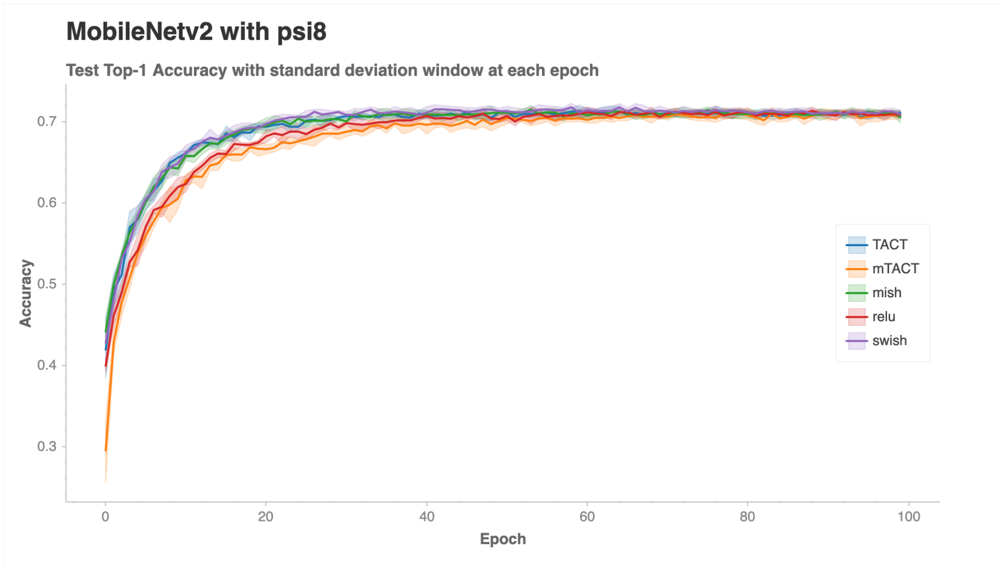

Figure 2.42: MobileNet v2 on the psi8 dataset

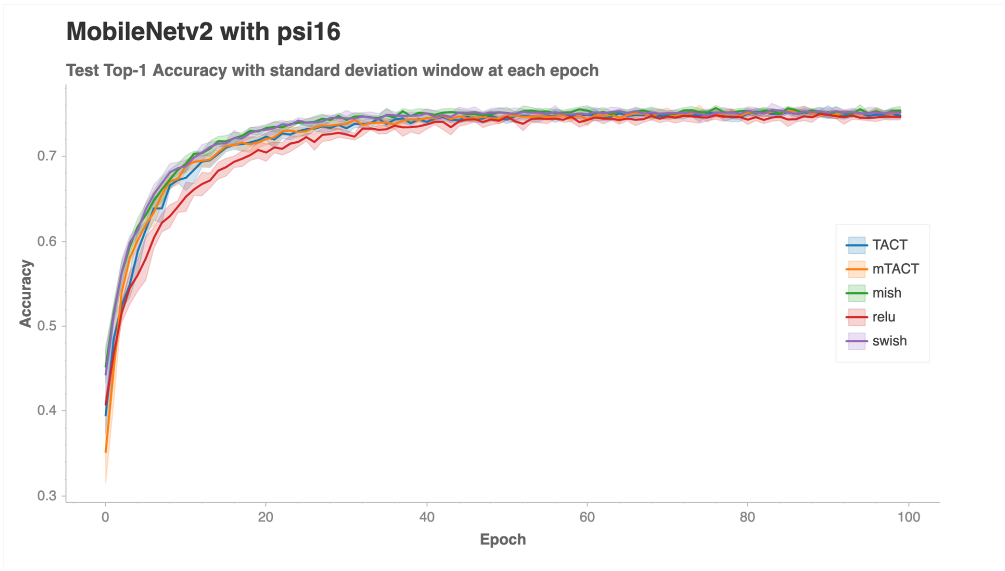

Figure 2.43: MobileNet v2 on the psi16 dataset

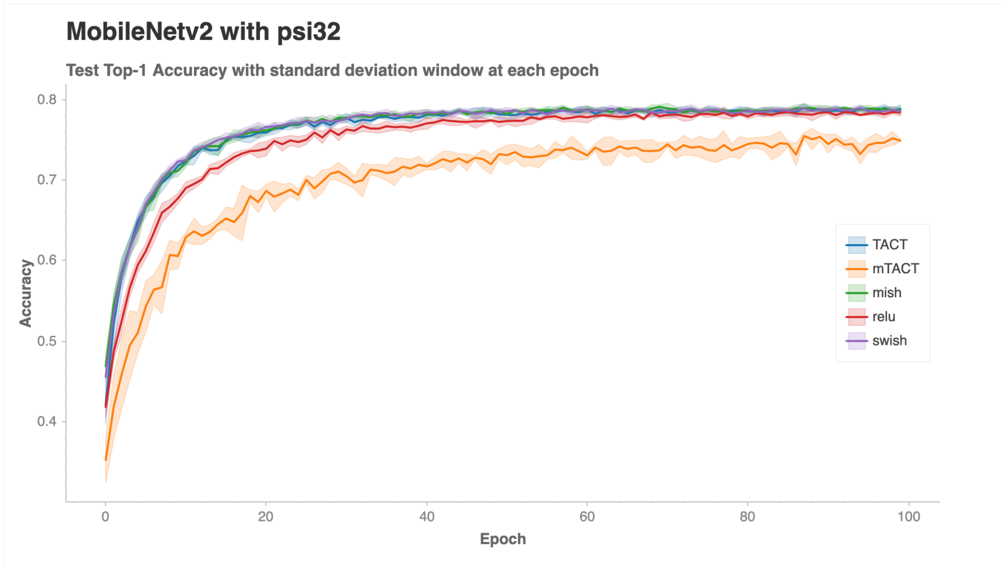

Figure 2.44: MobileNet v2 on the psi32 dataset

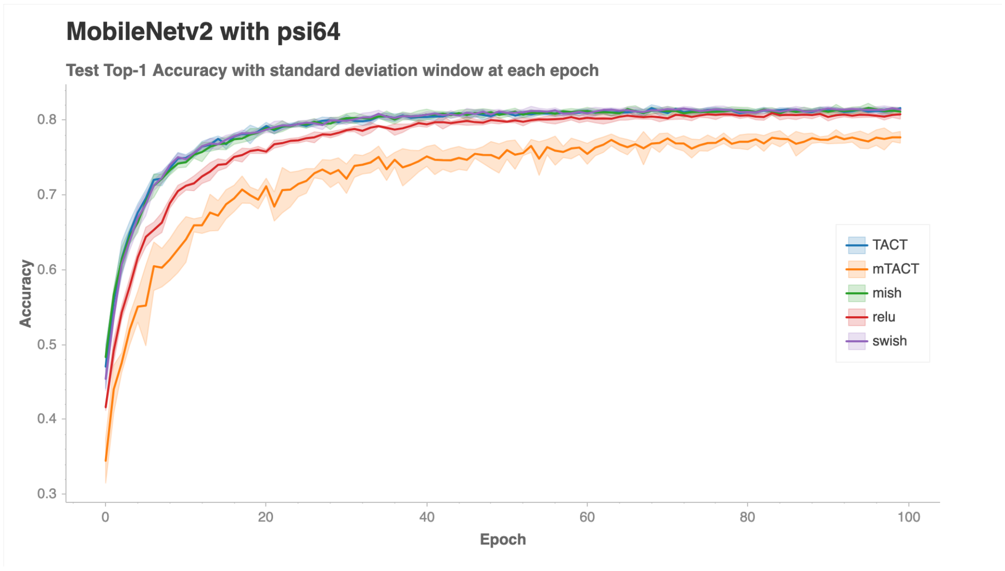

Figure 2.45: MobileNet v2 on the psi64 dataset

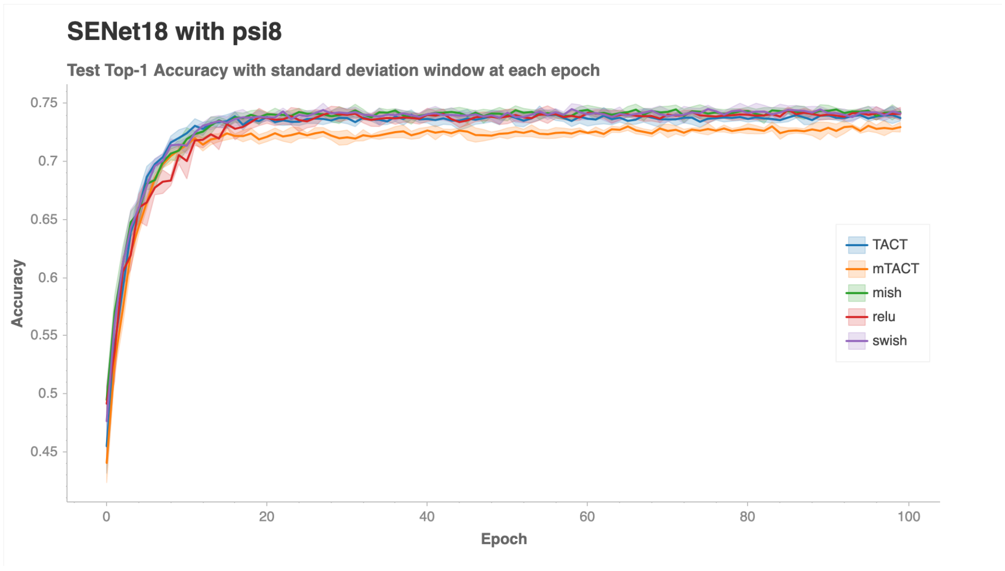

Figure 2.46: SE Net-18 on the psi8 dataset

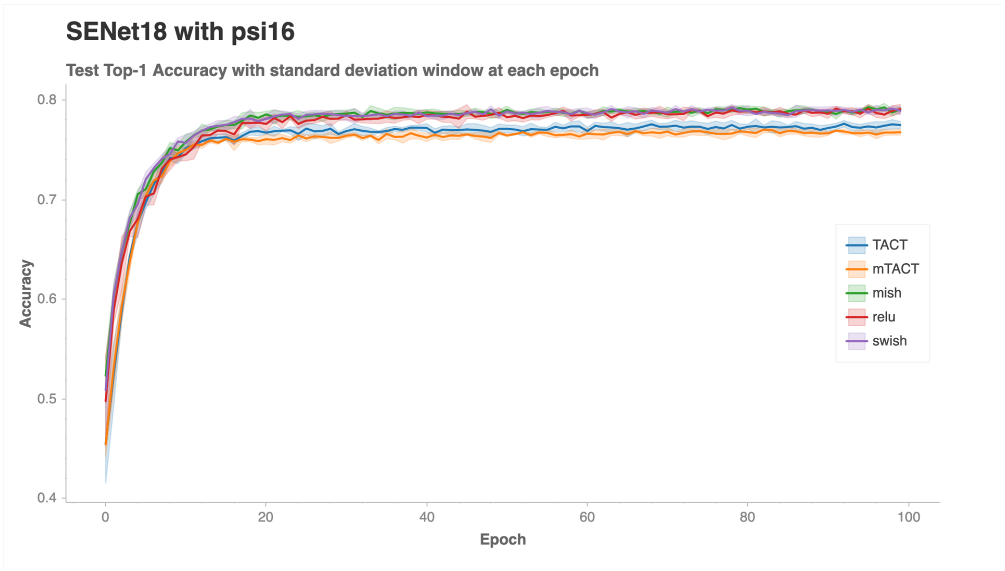

Figure 2.47: SE Net-18 on the psi16 dataset

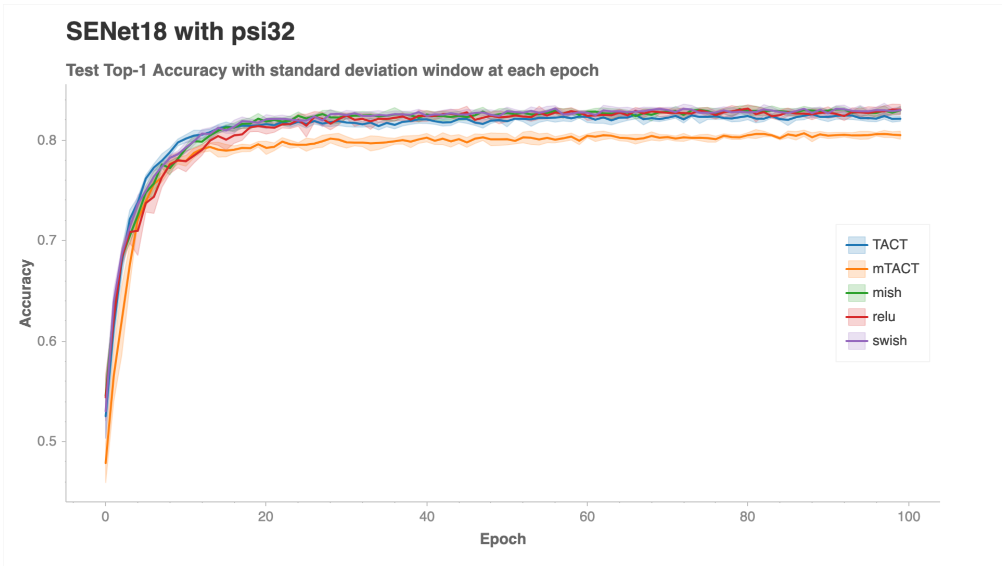

Figure 2.48: SE Net-18 on the psi32 dataset

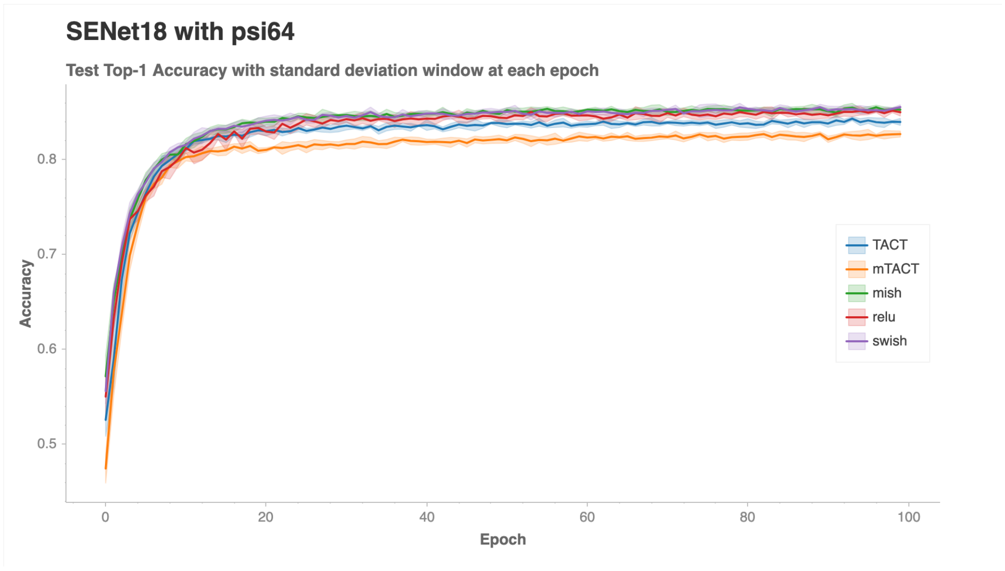

Figure 2.49: SE Net-18 on the psi64 dataset

Yet again, there was no clear pattern in the relationships between activation function, model complexity, and degree of image corruption.

Next, we tested seven more runs of each of the two learning rate policies (TLR and FLR) on each of the four noisy datasets (psi8, psi16, psi32, and psi64) on the Simple ResNet-18 Model. Our previous three runs had been sanitized, since "null runs" were discarded. A null run occurred when the error reached a value that was far too high at some point during the run, which led to the test accuracy to drop to 0.100000 for the rest of the epochs in that run. The cause was unknown, but we suspected that some activation functions might be more prone to producing null runs than others on the simple model. The complex models probably had some code to protect against null runs, because only the simple model ever produced null runs. With a total of 10 runs averaged on each of the two learning rate policies on each of the four noisy datasets, we wanted to produce graphs to compare with the santized three-run versions to see how frequently null runs occurred for a given activation function. These graphs with the 10-run averages are still being finalized.

3 Discussion

We do not have a definitive conclusion yet, since we still need to run many more experiments with different variations. Given the data we currently have, it would be scientifically irresponsible to to comment on the effect of activation functions on the performance of artificial neural networks. Hopefully, we will have sufficient results in the next few months. We still have hope for TAct and mTAct, since their unique formulations will at least provide some insight into the ideal properties of activation functions, even if these two particular activation functions do not consistently outperform other activation functions for a specific task. As of now, TAct and mTAct seem to be on par with the other activation functions tested in terms of performance, but these two activation functions may be best suited for a particular task that we have not tested yet. Given that TAct and mTAct did not outperform the other activation functions using the simple model on the noisy datasets, we are reluctant to claim that our revised hypothesis on TAct and mTAct with simple models is correct, even though TAct and mTAc did indeed perform the best on the simple model using the unmodified version of the CIFAR-10 dataset.

We are currently investigating the performances of activation functions with the other kinds of models running on the noisy datasets. Without any conclusive results, we must now consider new pathways of investigation for this research project. For example, we may look at hyperactivations, which are activation functions that are constructed during the training of the model.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–12, Dec 2015.

[2] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. pages 1–57, April 2009.

[3] D. Misra. Mish: A Self Regularized Non-Monotonic Neural Activation Function. pages 1–13.

[4] L. Smith. Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Apr 2017.

[5] C. J. Vercellino and W. Y. Wang. Hyperactivations for Activation Function Exploration. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.