Model Design for Deep Learning in Practice

Ali Turfah
Department of Statistics
Columbia University
New York, NY 10027
at3473@columbia.edu

1 Introduction

1.1 Project Goals

The goal of this project is to understand how different configuration choices affect the convergence, robustness, and performance of neural networks. In order to accomplish this, I will fit models on several datasets while varying the architecture, regularization, optimization, and initialization used. Based on these results, I will assess not only which configuration choices lead to strong performance on certain tasks, but also if there is consistency in performance across the different tasks.

1.2 Background

Architecture In lecture we discussed the use of pooling operations and increasing stride as downsampling methods in convolutional networks to ignore redundant local information. In [8], the authors evaluate different components of convolutional image classification models and find that the max-pooling operation can be replaced by a convolution without compromising performance on the CIFAR-10 dataset (they did not conduct the downsampling experiments on CIFAR-100, citing time constraints). In fact, they propose an all-convolutional model for image classification which—as the name would suggest—lacks any dense or max pooling layers and only make use of convolutional operations. This serves to guide the choice of architectures in this report; besides the two-dense-layer architecture I have chosen to adapt the different versions of Model-C and compare their performance across different datasets.

Regularization Results such as [6] argue that applying Dropout before Batch Normalization causes a shift between the variance of the training and evaluation datasets which degrades performance as the Batch Normalization won't be able to "normalize" the data correctly. Thus, they recommend applying Dropout after the Batch Normalization. In support of this, other papers such as [1] observe improvements in model convergence and performance when using this ordering on models trained using CIFAR-10 and CIFAR-100 data. This leads to the choice of regularizers in this project, which will be different orderings and combinations of Dropout and Batch Normalization models.

Optimization As discussed in lecture, Adam [5] is a good starting point for the model optimizer. Adam uses bias-corrected estimates of the first and second moments of the loss function to adjust the learning rate in the gradient descent updates. However, [7] identifies high variance in the initial stages of Adam's learning rate updates and proposes an extension called Rectified-Adam (or RAdam) which results in a constant variance of the adaptive learning rate. They demonstrate theoretically aned empirically that this should lead to better optima being achieved in less epochs. In addition, I will fit models using Stochastic Weight Averaging [4] (SWA) on Stochastic Gradient Descent (SGD). The main idea behind SWA is to average model weights along the descent trajectory to hopefully find 'flatter' solutions which should generalize better. In the original paper SWA is verified against optimizers with constant or cyclic learning rates such as SGD, which means Adam is not an ideal candidate for the optimizer since it adaptively sets its learning rate. In order to assess the benefits of SWA, I will also fit models using SGD alone in addition to SWA using the same learning rate.

Initialization Finally, for the initialization, I will test the variations in overall model performance when using different Normal and Uniform distributions. Specifically, I will test distributions whose "spread" is fixed regardless of layer size versus the Glorot versions of these distributions [3], whose range/variance depends on the input and output dimensions of the layer; "larger" dimensions will result in a more constrained distribution. The purpose of this initialization scheme is to control the layer activations and gradient variances throughout layers of larger and deeper networks.

1.3 Datasets

In this project I will evaluate model performance on four datasets: MNIST, Fashion-MNIST, Kuzushiji-MNIST, and Kuzushiji-49[1] [2]. Data augmentation by randomly rotating an image between $\pm 5^\circ$ and also shifting an image vertically or horizontally by up to five pixels was applied during training. 5-fold cross validation was used to evaluate model performance on each data set, where the test set was kept separate and 20% of the training set is used as a validation set in each fold. Folds were split sequentially, so the first fold uses the first 20% of the training data as the validation set, the second uses the next 20%, and so on. A description of each dataset is included below, with summary statistics and examples included in Table 1 and Figure 1.3 respectively.

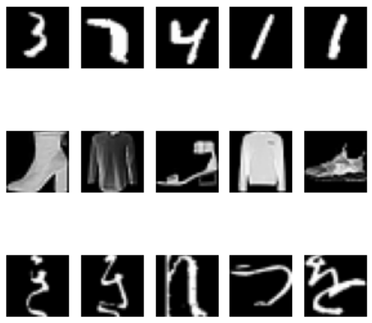


Figure 1: Sample images where the first row comes from MNIST, the second from Fashion-MNIST, and the third from the Kuzushiji datasets.

- **MNIST:** MNIST consists of black-and-white handwritten digits which comprise 10 classes (0-9). This dataset is balanced across classes, meaning there is an equal number of each class in the training, validation, and testing sets.
- **Fashion-MNIST:** Fashion-MNIST consists of black-and-white images of different articles of clothing. There are 10 diffierent classes: T-Shirts, Trousers, Pullovers, Dresses, Coats, Sandals, Shirts, Sneakers, Bags, and Ankle boots. This dataset is balanced across classes.
- **Kuzushiji-MNIST:** Kuzushiji-MNIST consists of black-and-white Japanese Hiragana characters written in the Kuzushiji script (a cursive writing style used until the 20^{th} century). There are 10 different characters represented in this dataset: お (o), き (ki), す (su), つ (tsu), な (na), は (ha), ま (ma), や (ya), れ (re), を (wo). Note that one character is taken from each consonant sound of the Hiragana letters (this corresponds to a column in Table 3 in Appendix A). This dataset is balanced across classes.
- **Kuzushiji-49:** Kuzushiji-49 is an extension of Kuzushiji-MNIST which includes examples for all Hiragana characters (see Appendix A for the complete set of characters). This dataset is not balanced across classes.

[1] `https://github.com/rois-codh/kmnist`

Table 1: Summary statistics for the MNIST, Fashion-MNIST, Kusuzhiji-MNIST, and Kuzushiji-49 datasets.

Dataset	Num. Classes	Num. Train	Num. Test	Image Dimension
MNIST	10	60,000	10,000	28x28x1
Fashion-MNIST	10	60,000	10,000	28x28x1
Kuzushiji-MNIST	10	60,000	10,000	28x28x1
Kuzushiji-49	49	232,365	38,547	28x28x1

2 Methods

2.1 Architectures

I will consider five architectures in this project denoted: Model C, Strided CNN, All CNN, ConvPool CNN, and 2FC. All of these models include a classification head which is a dense (or convolutional) layer that computes the predicted class logits. The first four models are analogous to the corresponding models in [8], with the number of filters and layers reduced for fitting time concerns over all the different configurations. The convolutional model architectures are specified in Table 2 below.

Table 2: Architecture specifications for convolutional models, where <N> is the number of classes in the chosen task.

Model C	**Strided CNN**	**ConvPool CNN**	**All CNN**
3x3; 32 Filters	3x3; 32 Filters; Stride 2	3x3; 32 Filters	3x3; 32 Filters
–	–	3x3; 32 Filters	–
3x3 M.P. Stride 2	–	3x3 M.P. Stride 2	3x3; 32 Filters; Stride 2
3x3; 64 Filters	3x3; 64 Filters; Stride 2	3x3; 64 Filters	3x3; 64 Filters
–	–	3x3; 64 Filters	–
3x3 M.P. Stride 2	–	3x3 M.P. Stride 2	3x3; 64 Filters; Stride 2
3x3; 64 Filters			
1x1; 64 Filters			
1x1; <N> Filters (Classification Head)			

Taking Model C ($\sim 60K$ parameters) as the default convolutional model, the Strided CNN ($\sim 60K$ parameters) removes the pooling operation and increases the stride of the previous convolutional layer. The All CNN ($\sim 110K$ parameters) replaces the max pooling operation of Model C with a new convolutional layer having a stride of 2. Finally, the ConvPool CNN model($\sim 110K$ parameters) adds a convolutional layer before the max pooling operation, and is meant to account for the increase in the number of parameters in the All CNN model when compared to Model C. All architecture-level regularization will occur after the pooling (or corresponding strided convolution) layers.

The 2FC model ($\sim$ 13K parameters) consists of two dense layers with 16 units each, and is meant to correspond to the "base model" specified in the project description. Experiments on this model will be considered separately from the convolutional models—as the difference in model size compared to the larger CNN models is roughly an order of magnitude—and are meant to see how much the base model's performance can be improved upon with augmentation and different choices of regularization, optimization, and initialization.

Also note that all convolutional and dense layers except for the "classification head" have a ReLU activation applied.

2.2 Optimization Techniques

As mentioned earlier, the four optimizers chosen for this project are Adam, Rectified Adam (RAdam), Stochastic Gradient Descent (SGD), and SGD with Stochastic Weight Averaging (SGD-SWA). The exact parameter specifications are provided below.

- **Adam**: Learning rate of 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$ where β_1 and β_2 are the exponential decay rates for the first and second moments respectfully. These are the default parameter specifications.

- **RAdam**: The learning rate and β values are exactly as in Adam, with a warm-up portion of 0.1 and a minimum learning rate of 10^{-10}.
- **SGD**: Learning rate of 10^{-2}, momentum = 0.9, where momentum is used as an additional force to stabilize the descent direction.
- **SMA**: Starting point of 0 and averaging period of 10, where the starting point defines how many epochs will have passed before starting to average models and the averaging period defines how often to take model averages.

Learning rates were set using the LR Finder[2] on Model C described in [9], taking a value roughly in the middle of the section of steep descent as the learning rate. As the same optimizer configuration was meant to be tested across all datasets (to test robustness and generalizability of a model), the learning rate for a specific optimizer "class" (Adam or SGD-based) were set to a value that would be reasonable on all the datasets. The results from Figure 4 suggest that learning rates of 10^{-2} and 10^{-3} would be appropriate for SGD and for Adam respectively.

The base model was fit over 100 epochs using a batch size of 128 and the Adam optimizer with default specifications. All remaining models were fit using a training batch size of 384, with a maximum training length of 200 epochs.

2.3 Regularization Techniques

All models except the base model were implicitly regularized using early stopping if 5 epochs passed without improvement in the validation loss. In the event that the model's training was terminated before the maximum allowed number of epochs, the best performing model's weights were restored.

As far as architecture-level regularization goes, models were regularized using a combination of Dropout and Batch Normalization layers. The Dropout layer probability was set to 0.5 whereas the Batch Normalization layers were left with the default parameters (momentum in the moving average of 0.99, centering and scaling the data). In the 2FC model the regularization layer block is placed after the first fully connected layer, for the convolutional networks the regularization is placed after the pooling (or convolution with increased stride) layers.

In order to test the effects of the ordering of Batch Normalization and Dropout, four architecture-level regularization techniques will be tested: Dropout alone and Batch Normalization alone—in order to determine the effects of each regularization technique independently–in addition to Dropout followed by Batch Normalization and Batch Normalization followed by Dropout.

2.4 Initialization Techniques

The convolution/dense layer weights were initialized from the Random Uniform, Random Normal, Glorot Uniform, and Glorot Normal distributions specified in the Tensorflow library. Tensorflow will use the Glorot Uniform distribution for the weights by default. The bias vector/matrices are initialized to be zero in all models, as is the Tensorflow default. The exact parameters for the chosen distributions can be found below.

- **Random Uniform**: Initializes weights $\sim Unif(-0.05, 0.05)$
- **Random Normal**: Initializes weights $\sim N(0, 0.05^2)$
- **Glorot Uniform** (Tensorflow Default): Initializes weights $\sim Unif(-limit, limit)$, where

$$limit = \sqrt{\frac{6}{(\textbf{[input units]} + \textbf{[output units]})}}$$

- **Glorot Normal**: Initializes weights $\sim N(0, limit^2)$, where

$$limit = \sqrt{\frac{2}{(\textbf{[input units]} + \textbf{[output units]})}}$$

[2] `https://github.com/surmenok/keras_lr_finder/`

3 Results

The results will be split between the 2FC and the convolutinal architectures as mentioned earlier. Overall performance for a given configuration choice will be determined by the mean performance across all folds of the data for models trained with those choices. In addition, the 5^{th} and 95^{th} percentiles of the results (i.e. not aggregated across folds) are provided to get a better idea of the range. There are a number of models which do not improve early on in their training and thus terminate training too early, resulting in random or slightly-better-than-random performance. This behavior results in the possibility of wildly different performance for a single model across folds which skews the results for the range when averaging over folds. If there is not a significant difference in the configuration choices for a given category (architecture, initializer, optimizer, regularizer) or the table is too large, the table will be omitted from the main body of the report and can be found in Appendices C and D for the 2FC and convolutional architectures respectively.

3.1 Fully Connected Models

3.1.1 Base Model Performance

Based on the convergence graphs in Figure 2, we see the training loss and accuracy improve over the course of training while the validation loss and accuracy tend to decay or stabilize after a certain point early on in training. This behavior is consistent across all datasets, but is less pronounced in Kuzushiji-49, where the training and validation accuracy fluctuate by $\sim$ 1-2% for the last 80 epochs. This could be suggestive of minor overfitting, as the loss in most cases increases steadily after reaching a minimum, but this does not seem to adversely affect the validation accuracy on the different datasets.

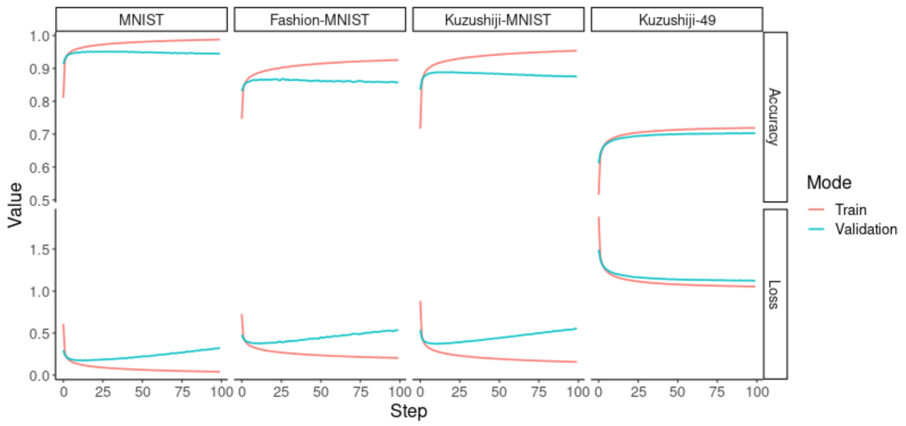


Figure 2: On non-Kuzushiji-49 datasets, Validation accuracy and loss reach their optimal values early on in training then degrade while training accuracy and loss tend to improve past that point.

Oddly, when compared to the models with data augmentation and different regularization, optimization, and initialization set-ups, even the best performing models falling behind the base model by as much as 10% (see Appendix C for the top performing models). These results are consistent with the values observed in the 90% range of Table 13, so the top performing models aren't outliers. Besides the relatively high dropout probabilities hindering this small of a model's ability to fit the data, this could be in part due to the data augmentation scheme overburdening this small of a network (relative to the number of data points). In addition, this could also be due to early stopping terminating the training before the model could reach better optima, however more experiments would be necessary to definitively verify these ideas.

Of the top performing 2FC models, Batch Normalization is consistently the best-performing regularizer, however initializer and optimizer choice seem to have much more variation.

Table 3: Overall performance of the Base Model and different 2FC model configurations across datasets.

Dataset	Base Model Acc.	2FC Model Acc.	90% Range
MNIST	0.949	0.8685	(0.8056, 0.9327)
Fashion-MNIST	0.853	0.6936	(0.649, 0.7523)
Kuzushiji-MNIST	0.768	0.5703	(0.5085, 0.6717)
Kuzushiji-49	0.587	0.3495	(0.2803, 0.4555)

3.1.2 Initializers and Optimizers

Based on the results in Tables 10 and 11 (presented in Appendix C), there does not appear to be much difference in the overall performance in the 2FC models when splitting by the initializers or optimizers; the results for the choices are within 1% of each other across all datasets. This is also observed with the 90% range of values, the bounds of which are also incredibly close. This is not that surprising for the initializers, as [3] mentions their initialization is meant to stabilize large and deep networks, neither of which describe this family of models. I am unsure why there is not a difference in performance across optimizers.

3.1.3 Regularizers

Based on the results in Table 4, it would appear that the regularizer plays a large role in the model performance. Batch Normalization consistently yields the highest results across the datasets. There does appear to be a significant difference in performance between the orderings of Batch Normalization and Dropout, as the lower end of the Batch Norm / Dropout range corresponds to the upper end of the Dropout / Batch Normalization range. Oddly enough Dropout / Batch Normalization preformed worse on average than Dropout on all but the MNIST dataset, however the results were close enough that this is probably due to random chance. This is supported by the range of performances, which are usually within 1% of each other on the bounds. These results support the claims from [6], that the ordering of Batch Normalization and Dropout affects network performance.

Table 4: Overall performance of the 2FC model architectures across regularizers.

	MNIST		Fashion-MNIST	
Regularization	Test Acc.	90% Range	Test Acc.	90% Range
BatchNorm	**0.928**	(0.916, 0.938)	**0.746**	(0.734, 0.759)
BatchNorm / Dropout	0.878	(0.863, 0.891)	0.697	(0.688, 0.707)
Dropout / BatchNorm	0.838	(0.804, 0.856)	0.663	(0.644, 0.681)
Dropout	0.831	(0.799, 0.858)	0.668	(0.645, 0.686)

	Kuzushiji-MNIST		Kuzushiji-49	
Regularization	Test Acc.	90% Range	Test Acc.	90% Range
BatchNorm	**0.661**	(0.644, 0.676)	**0.451**	(0.443, 0.459)
BatchNorm / Dropout	0.565	(0.55, 0.577)	0.351	(0.344, 0.36)
Dropout / BatchNorm	0.524	(0.503, 0.548)	0.295	(0.277, 0.312)
Dropout	0.531	(0.51, 0.555)	0.301	(0.278, 0.32)

3.2 CNN Models

3.2.1 Architectures

Based on the results in Table 13 (presented in Appendix D), there is no clear-cut ‘best architecture‘ across the different datasets. As far as the comparative performance based on down-sampling operation, there does not appear to be a trend favoring one over the other either in terms of mean performance or in the range of performances. For example, the Strided CNN architectures may outperform the Model C ones on MNIST and Fashion MNIST, but Model C performs better on Kuzushiji-MNIST and Kuzushiji-49. It is worth noting that, while the larger architectures (All CNN and ConvPool CNN) did not seem to perform better on the 10-class datasets, a performance gap is observed in Kuzushiji-49. In addition, the top performing models from Table 12—when accounting for performance across all folds—seems to all have the ConvPool CNN architecture, but based on overall performance I am hesitant to claim it to the the "best" architecture.

3.2.2 Initializers

Based on the results in Table 5 we see that the Glorot distributions consistently outperform the corresponding Random distributions, in addition to having a tighter performance range which supports the results from [3]. The Glorot Normal and Uniform distribution mean performance are within less than half a percentage point of each other on all tasks, which suggests that overall their performance is identical if not nearly identical. This is further supported when considering the range of values observed, which are within 1-2% points of the other at the endpoints.

When looking to the Random Normal and Uniform models, we see a much more pronounced difference in model performance. The models initialized using a Random Uniform distribution performed anywhere between 5-18% worse than their Random Normal or Glorot counterparts. More telling than the performance is the range of observed values, which dips consistently lower for the Random Uniform values. On all but Kuzushiji-49, as much as 5% of all the total fit models were not able to improve past a random guess, which suggests being terminated very early on during training due to not being able to improve performance from the initial state.

Table 5: Overall performance of the CNN architectures across initializers.

	MNIST			Fashion-MNIST		
Initialization	# Epochs	Test Acc.	90% Range	# Epochs	Test Acc.	90% Range
Glorot Normal	37.3	**0.982**	(0.971, 0.993)	36.6	**0.847**	(0.782, 0.892)
Glorot Uniform	36.1	0.981	(0.970, 0.993)	34.5	0.843	(0.770, 0.889)
Random Normal	43.8	0.954	(0.785, 0.992)	37.0	0.817	(0.739, 0.879)
Random Uniform	41.3	0.800	(0.113, 0.991)	33.3	0.674	(0.100, 0.865)
	Kuzushiji-MNIST			Kuzushiji-49		
Initialization	# Epochs	Test Acc.	90% Range	# Epochs	Test Acc.	90% Range
Glorot Normal	50.0	**0.914**	(0.864, 0.952)	47.3	0.854	(0.771, 0.916)
Glorot Uniform	49.3	0.913	(0.848, 0.954)	50.3	**0.855**	(0.771, 0.914)
Random Normal	56.6	0.894	(0.820, 0.950)	55.8	0.846	(0.771, 0.905)
Random Uniform	50.3	0.734	(0.100, 0.948)	60.5	0.798	(0.671, 0.902)

As far as rate of convergence, it is difficult to tell at which point models actually converged and which were terminated in the early stages; looking to Fashion-MNIST and Kuzushiji-MNIST we see that models initialized with the Random Uniform scheme had less epochs until convergence, but also had more than 5% of the models terminate training before they could improve past a random guess (which usually happens within 10 epochs). Looking at the other initializers, the Glorot Distribution models converge faster (and to better optima) than the Random Normal ones. Based on these results there does not appear to be much of a significant difference between models initialized with either of the Glorot distributions, however in order to make more definitive claims the analysis would need to account for the early-terminated models.

Table 6: Overall performance of the CNN models across optimizers.

	MNIST			Fashion-MNIST		
Optimization	# Epochs	Test Acc.	90% Range	# Epochs	Test Acc.	90% Range
Adam	34.6	**0.947**	(0.693, 0.993)	33.0	0.824	(0.684, 0.894)
RAdam	36.7	0.933	(0.591, 0.993)	36.1	**0.829**	(0.730, 0.889)
SGD	44.0	0.920	(0.507, 0.991)	36.5	0.768	(0.194, 0.878)
SGD-SWA	43.0	0.917	(0.476, 0.990)	35.7	0.760	(0.100, 0.875)
	Kuzushiji-MNIST			Kuzushiji-49		
Optimization	# Epochs	Test Acc.	90% Range	# Epochs	Test Acc.	90% Range
Adam	45.9	0.894	(0.741, 0.955)	47.4	0.822	(0.693, 0.914)
RAdam	49.4	**0.897**	(0.834, 0.953)	49.6	**0.848**	(0.772, 0.914)
SGD	55.2	0.836	(0.100, 0.941)	58.3	0.842	(0.760, 0.908)
SGD-SWA	55.7	0.829	(0.100, 0.943)	58.6	0.841	(0.755, 0.907)

3.2.3 Optimizers

Based on the results in Table 6, we see that the Adam family of optimizers tend to outperform the SGD ones. RAdam and Adam both perform similarly to each other (less than a 2% difference, which seems to be due to the lower tail from the 90% Range). SGD and SGD-SWA tend to perform very similarly to each other, with mean performance and range boundaries usually within 1%, which suggests that SGD outperforming SGD-SWA may be due to random chance. RAdam tends to be the best choice across the datasets, however performed worse than Adam on MNIST. Besides this dataset, RAdam has a tighter 90% range which suggests that the model training is more consistent as far as reaching better optima is concerned. It is possible that the RAdam models terminated training while the warm-up procedure was still happening, which could be why the aggregated accuracies are similar between them and Adam in spite of a “better" range of values.

As far as time to convergence goes, much like the 2FC models it is difficult to make much out of results which do not take into account whether a model terminated before the chance to learn much or if it terminated after training. This is especially true for the SGD and SGD-SWA models on Kuzushiji-MNIST, of which both had at least 5% of their results not improving past a random guess. However, it does appear that RAdam consistently takes longer than Adam to converge, which may in part be due to the warm-up period, granted only by a few epochs.

3.2.4 Regularizers

Based on the results in Table 7, it is apparent that Batch Normalization alone outperforms the other regularization techniques by 2-6% on average, and converges faster. On the other side, Dropout alone tends to perform poorly and takes much more time to converge, even when you consider the portion of models which terminate training early (as indicated by the lower bound of the 90% range being around 10%). I do not believe that this is a result of Dropout over-regularizing the models, as the upper bounds of the performance ranges are all very close; it may just be that the actual training process is more unstable (with respect to early termination) which pulls the lower bounds further down on some of the methods.

With respect to the ordering of the two regularizers, these results do not indicate one ordering is superior. Average performance does not seem to favor one over the other, however the convergence time seems to favor Batch Normalization followed by Dropout when they had similar results (in MNIST, Kuzushiji-MNIST, and Kuzushiji-49). This contradicts what was seen on the 2FC models, which makes drawing definitive conclusions difficult.

Table 7: Overall performance of the CNN models across regularizers.

	MNIST			Fashion-MNIST		
Regularization	# Epochs	Test Acc.	90% Range	# Epochs	Test Acc.	90% Range
BatchNorm	25.4	**0.980**	(0.964, 0.991)	27.6	**0.845**	(0.777, 0.888)
BatchNorm / Dropout	34.5	0.934	(0.762, 0.993)	33.9	0.820	(0.723, 0.888)
Dropout / BatchNorm	39.4	0.952	(0.858, 0.993)	33.1	0.782	(0.407, 0.889)
Dropout	59.2	0.851	(0.113, 0.992)	46.7	0.735	(0.100, 0.878)
	Kuzushiji-MNIST			Kuzushiji-49		
Regularization	# Epochs	Test Acc.	90% Range	# Epochs	Test Acc.	90% Range
BatchNorm	32.7	**0.902**	(0.848, 0.939)	44.0	**0.864**	(0.786, 0.916)
BatchNorm / Dropout	48.1	0.881	(0.765, 0.951)	48.9	0.844	(0.762, 0.914)
Dropout / BatchNorm	55.0	0.882	(0.747, 0.952)	53.0	0.834	(0.751, 0.905)
Dropout	70.4	0.791	(0.100, 0.954)	68.0	0.812	(0.671, 0.902)

3.3 Variable Importance

In order to attempt to quantify the effect of the configuration choice categories (Architecture, Initializer, Optimizer, and Regularizer), I performed Random Forest regression of these categories onto the final test accuracy rate in order to compute the variable importance. This quantity is measured from the percent change in mean squared error of the predictions when using a column where the data is randomly shuffled as opposed to undisturbed. Variables for which there is a large decrease in accuracy are considered to be highly important, whereas ones with little to no decrease (or even improvement) are considered to be less important. The results are shown in Table 8.

Table 8: Random forest variable importance analysis for models across the datasets.

	2FC Models			
	MNIST	Fashion-MNIST	Kuzushiji-MNIST	Kuzushiji-49
Pct. Variance Explained	*90.0%*	*92.1%*	*94.4%*	*95.4%*
Regularizer	**57.7%**	**60.3%**	**58.4%**	**64.4%**
Architecture	–	–	–	–
Optimizer	-7.2%	-9.2%	-8.0%	-8.5%
Initializer	-9.2%	-10.2%	-8.0%	-8.8%
	CNN Models			
	MNIST	Fashion-MNIST	Kuzushiji-MNIST	Kuzushiji-49
Pct. Variance Explained	*46.3%*	*66.3%*	*68.6%*	*59.2%*
Regularizer	29.5%	31.5%	30.2%	13.2%
Architecture	-0.1%	6.6%	-1.4%	**48.3%**
Optimizer	0.4%	11.7%	19.0%	0.5%
Initializer	**39.7%**	**44.4%**	**41.7%**	22.2%

The results on the 2FC models are consistent with the results presented above, namely that the regularization technique is the most informative way to determine model performance. Interestingly enough the forests performed better when using randomly shuffled optimizer and initializer information. For the CNN models, the architecture does not appear to be a strong indicator of performance until we reach the significantly larger Kuzushiji-49 dataset, which is consistent with the results presented above. This is not particularly surprising, even with data augmentation the size of these models should be more than enough to handle MNIST and similar-sized 10-class datasets; a much larger 49-class dataset should see differentiation based on the architecture sizes as it may push the smaller ones closer to their limit. On the remaining datasets Initialization and Regularization appeared to be the most significant factors affecting performance, which as mentioned before could be driven by difference in the rate of the early termination of models using certain initialization/regularization schemes. The forests generally were able to capture most of the variance observed in the data, which suggests overall good fit for these measurements.

4 Discussion

In this work, I conducted a study on the effects of different model configuration choices on the overall mean and range of performance on four datasets. When comparing the 2FC architecture models against the base model configuration, a consistent performance decrease is observed in all datasets which could be in part due to the early stopping criterion or the data augmentation schemes used. In general, the 2FC architecture performance seems in large part driven by the regularization, which is supported by the Random Forest analysis and can be seen by the different schemes occupying mostly distinct ranges of values. The extent to which these results are driven by the dropout inhibiting the model's ability to get out of a 'bad' initialization state before having training terminated by the early stopping criterion, or the performance degradation due to data augmentation, remains to be determined.

With respect to the convolutional architectures, the initialization technique appears to be a much stronger factor in determining model performance due to these models being much larger. On the other hand, there did not appear to be a significant difference between the striding and pooling convolutional architectures, which suggests that the choice of down-sampling operation does not significantly impact performance. For regularization, the ordering of the Batch Normalization and Dropout layers does not appear to significantly impact model performance, however there does appear to be an overall performance gain from including Batch Normalization with the Dropout regardless of the order in which they are applied. There did not appear to be any significant difference between the performance of the optimizers, with SGD and SGD-SWA performing nearly identically, however RAdam did appear to have more consistency in terms of model performance than Adam alone (as observed from the tighter performance range). Once again, the extent to which these results are due to the early stopping criterion interacting with ‘bad’ initialization states (or a model’s inability to escape from them) cannot be definitively determined form this analysis.

In future work I would like to extend this analysis to account for the effects of the early stopping criterion and data augmentation on the model performances. As indicated above I believe that a significant portion of the poorly-performing models were caused by the early stopping criterion terminating training too early, as a 100K parameter model should be able to at least somewhat learn MNIST. In addition, I would like to explore the CNN model performances across more ‘complex’ datasets such as CIFAR-10 and CIFAR-100, in order to better characterize the differences between the architectures. Finally, it may be more informative of optimizers do more thorough tests such as testing the optimizer across different learning rates, or using a cyclic learning rate for SGD, which may improve or at least differentiate performance when using SWA somewhat.

More of a meta-concern, but exploring all these different possible configurations in the same set of experiments causes the number of models to be fit to increase very quickly; the results presented in this paper come from more than 6,000 models which took several days to fit after being split across multiple machines. That being said, more efficient and directed experiments to assess the individual effects of the model configuration choices will be necessary if this isto scale up past a few variations of the choices.

References

[1] Guangyong Chen, Pengfei Chen, Yujun Shi, Chang-Yu Hsieh, Benben Liao, and Shengyu Zhang. Rethinking the usage of batch normalization and dropout in the training of deep neural networks. *CoRR*, abs/1905.05928, 2019.

[2] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep learning for classical japanese literature. *CoRR*, abs/1812.01718, 2018.

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*. JMLR Workshop and Conference Proceedings, 2010.

[4] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization, 2019.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[6] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift, 2018.

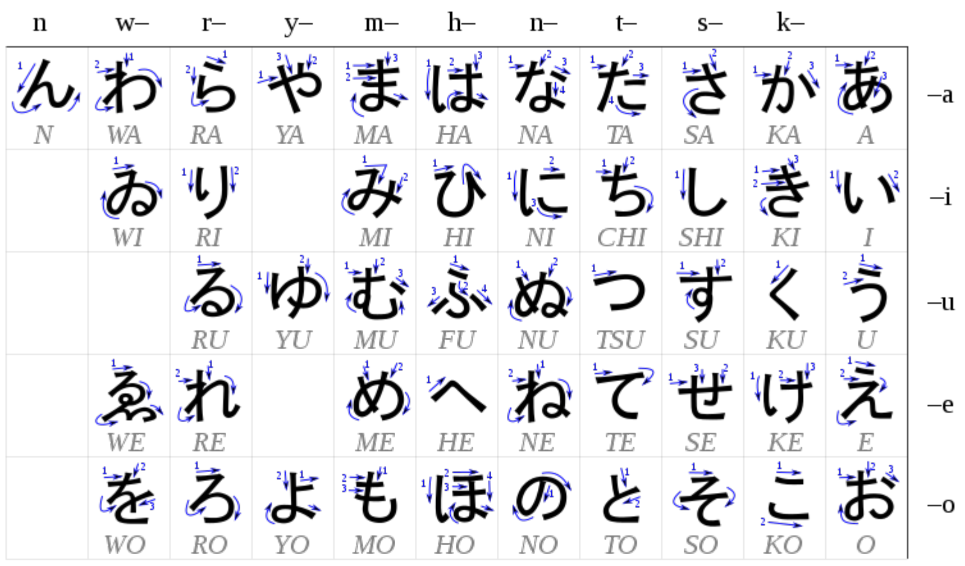
[7] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond, 2020.

[8] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2015.

[9] Paul Surmenok. Estimating an optimal learning rate for a deep neural network, 2017.

Appendices

A Hiragana Table



n	w–	r–	y–	m–	h–	n–	t–	s–	k–		
ん *N*	わ *WA*	ら *RA*	や *YA*	ま *MA*	は *HA*	な *NA*	た *TA*	さ *SA*	か *KA*	あ *A*	–a
	ゐ *WI*	り *RI*		み *MI*	ひ *HI*	に *NI*	ち *CHI*	し *SHI*	き *KI*	い *I*	–i
		る *RU*	ゆ *YU*	む *MU*	ふ *FU*	ぬ *NU*	つ *TSU*	す *SU*	く *KU*	う *U*	–u
	ゑ *WE*	れ *RE*		め *ME*	へ *HE*	ね *NE*	て *TE*	せ *SE*	け *KE*	え *E*	–e
	を *WO*	ろ *RO*	よ *YO*	も *MO*	ほ *HO*	の *NO*	と *TO*	そ *SO*	こ *KO*	お *O*	–o

Figure 3: Hiragana characters and corresponding consonant/vowel pronunciations.

B Learning Rate Graphs

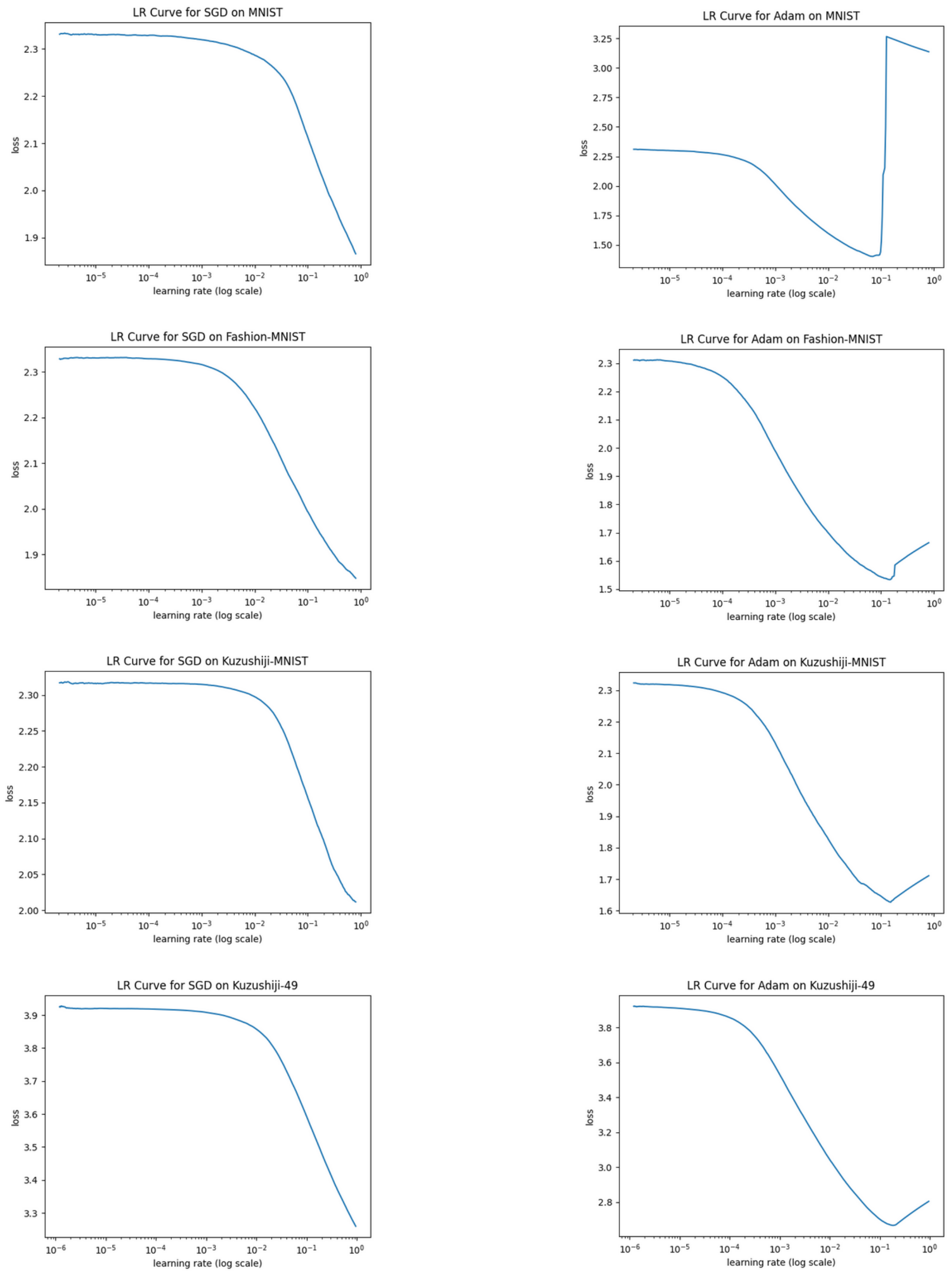


Figure 4: LR Finder suggests 10^{-2} and 10^{-3} as learning rates for SGD and Adam on the various datasets

C 2FC Model Cofiguration Tables

Table 9: Best performing model configurations for the 2FC models across all datasets averaged across folds.

MNIST			
Regularizer	Initializer	Optimizer	Test Acc.
BatchNorm	Glorot Normal	RAdam	0.9329
BatchNorm	Random Uniform	SGD-SWA	0.9323
BatchNorm	Random Normal	SGD	0.9320
BatchNorm	Glorot Normal	SGD-SWA	0.9303
BatchNorm	Glorot Normal	SGD	0.9298
Fashion-MNIST			
Regularizer	Initializer	Optimizer	Test Acc.
BatchNorm	Glorot Normal	RAdam	0.7549
BatchNorm	Random Normal	SGD-SWA	0.7523
BatchNorm	Random Normal	RAdam	0.7502
BatchNorm	Random Normal	SGD	0.7502
BatchNorm	Glorot Normal	SGD-SWA	0.7493
Kuzushiji-MNIST			
Regularizer	Initializer	Optimizer	Test Acc.
BatchNorm	Random Uniform	SGD-SWA	0.6711
BatchNorm	Random Uniform	RAdam	0.6702
BatchNorm	Random Normal	SGD-SWA	0.6689
BatchNorm	Random Uniform	Adam	0.6660
BatchNorm	Random Normal	RAdam	0.6639
Kuzushiji-49			
Regularizer	Initializer	Optimizer	Test Acc.
BatchNorm	Glorot Normal	RAdam	0.4562
BatchNorm	Random Normal	Adam	0.4550
BatchNorm	Glorot Normal	SGD-SWA	0.4544
BatchNorm	Glorot Uniform	Adam	0.4540
BatchNorm	Random Uniform	SGD-SWA	0.4519

Table 10: 2FC model performance across the datasets broken down by initialization technique.

	MNIST		Fashion-MNIST	
Initialization	Test Acc.	90% Range	Test Acc.	90% Range
Glorot Normal	0.8706	(0.8147, 0.9349)	0.6909	(0.6448, 0.7546)
Glorot Uniform	0.8662	(0.8019, 0.9317)	**0.6952**	(0.6558, 0.7497)
Random Normal	0.8650	(0.8022, 0.9324)	0.6940	(0.6513, 0.7538)
Random Uniform	**0.8723**	(0.8318, 0.9323)	0.6943	(0.6550, 0.7490)
	Kuzushiji-MNIST		Kuzushiji-49	
Initialization	Test Acc.	90% Range	Test Acc.	90% Range
Glorot Normal	0.5670	(0.5032, 0.6671)	0.3477	(0.2792, 0.4557)
Glorot Uniform	0.5679	(0.5099, 0.666)	0.3461	(0.2774, 0.4559)
Random Normal	0.5726	(0.5124, 0.6709)	**0.3525**	(0.2908, 0.4537)
Random Uniform	**0.5736**	(0.5194, 0.6743)	0.3519	(0.2928, 0.4553)

Table 11: 2FC model performance across datasets broken down by the optimizer.

	MNIST		Fashion-MNIST	
Optimization	Test Acc.	90% Range	Test Acc.	90% Range
Adam	0.8686	(0.8191, 0.9295)	0.6942	(0.6462, 0.7476)
RAdam	**0.8733**	(0.8281, 0.933)	**0.6968**	(0.6592, 0.7529)
SGD	0.8650	(0.8022, 0.9324)	0.6906	(0.6468, 0.7523)
SGD-SWA	0.8672	(0.7991, 0.9357)	0.6928	(0.6489, 0.7544)
	Kuzushiji-MNIST		**Kuzushiji-49**	
Optimization	Test Acc.	90% Range	Test Acc.	90% Range
Adam	0.5718	(0.514, 0.6707)	**0.3524**	(0.289, 0.4559)
RAdam	**0.5732**	(0.5073, 0.6729)	0.3519	(0.2887, 0.4555)
SGD	0.5698	(0.5104, 0.6743)	0.3467	(0.2773, 0.4537)
SGD-SWA	0.5662	(0.508, 0.6628)	0.3472	(0.2786, 0.4552)

D CNN Model Configuration Tables

Table 12: Top performing CNN models across the datasets averaged across the folds.

MNIST				
Architecture	Regularizer	Initializer	Optimizer	Test Acc.
All CNN	Dropout / BatchNorm	Glorot Uniform	Adam	0.9928
ConvPool CNN	Dropout / BatchNorm	Glorot Uniform	Adam	0.9926
ConvPool CNN	Dropout / BatchNorm	Glorot Normal	Adam	0.9925
All CNN	BatchNorm / Dropout	Glorot Uniform	Adam	0.9924
ConvPool CNN	BatchNorm / Dropout	Glorot Normal	Adam	0.9923
Fashion-MNIST				
Architecture	Regularizer	Initializer	Optimizer	Test Acc.
ConvPool CNN	BatchNorm / Dropout	Glorot Normal	Adam	0.8981
ConvPool CNN	Dropout / BatchNorm	Glorot Normal	Adam	0.8943
ConvPool CNN	BatchNorm	Glorot Uniform	Adam	0.8895
ConvPool CNN	BatchNorm	Glorot Normal	Adam	0.8882
ConvPool CNN	BatchNorm / Dropout	Glorot Uniform	RAdam	0.8882
Kuzushiji-MNIST				
Architecture	Regularizer	Initializer	Optimizer	Test Acc.
ConvPool CNN	Dropout	Glorot Uniform	RAdam	0.9550
ConvPool CNN	BatchNorm / Dropout	Glorot Normal	Adam	0.9538
ConvPool CNN	Dropout	Glorot Normal	Adam	0.9537
ConvPool CNN	Dropout	Glorot Uniform	Adam	0.9519
ConvPool CNN	Dropout / BatchNorm	Glorot Uniform	RAdam	0.9517
Kuzushiji-49				
Architecture	Regularizer	Initializer	Optimizer	Test Acc.
ConvPool CNN	BatchNorm	Glorot Uniform	Adam	0.9196
ConvPool CNN	BatchNorm	Glorot Uniform	RAdam	0.9182
ConvPool CNN	BatchNorm / Dropout	Glorot Normal	Adam	0.9163
ConvPool CNN	BatchNorm	Glorot Normal	RAdam	0.9143
ConvPool CNN	BatchNorm	Glorot Normal	Adam	0.9140

Table 13: Overall performance for the convolutional architectures across the datasets.

	MNIST		Fashion-MNIST	
Architecture	Test Acc.	90% Range	Test Acc.	90% Range
Model C	0.9255	(0.5861, 0.9900)	0.7609	(0.2414, 0.8697)
Strided CNN	**0.9441**	(0.8572, 0.9882)	0.7991	(0.7429, 0.8508)
All CNN	0.9308	(0.5888, 0.9929)	0.8002	(0.5874, 0.8833)
ConvPool CNN	0.9172	(0.5044, 0.9933)	**0.8212**	(0.6323, 0.8950)
	Kuzushiji-MNIST		**Kuzushiji-49**	
Architecture	Test Acc.	90% Range	Test Acc.	90% Range
Model C	**0.8789**	(0.7863, 0.9294)	0.8270	(0.7651, 0.8818)
Strided CNN	0.8311	(0.5706, 0.911)	0.7823	(0.7195, 0.8375)
All CNN	0.8679	(0.2518, 0.9507)	0.8579	(0.7589, 0.9089)
ConvPool CNN	0.8783	(0.1000, 0.9578)	**0.8860**	(0.8081, 0.9197)