Assessing the Robustness of Classifiers on Noisy Datasets of Handwritten Digits

Reuben Gardos Reid, Gaoxiang Luo, Max Schickert, Gustav Baumgart*
Department of Computer Science and Engineering
University of Minnesota - Twin Cities
Minneapolis, MN 55455
{gardo007, luo00042, schic188, baumg260}@umn.edu
https://github.umn.edu/MMNISTs/5523-group

Abstract

Many classification algorithms have been popular within machine learning communities, but in real-world settings it's labor-intensive to have perfectly clean datasets. To reproduce the real-world situation, in this project we performed single flips on class labels to make a noisy but balanced dataset of handwritten digits with scaling noise levels, to assess the robustness of four common classifiers – K Nearest Neighbor, Support Vector Machine, Fully Connected Neural Network and Convolutional Neural Network – on noisy datasets of handwritten digits. We found that the most robust model in terms of overall noise introduced is the convolutional neural network because of it's higher accuracy all the way up to around the 90^{th} percentile of noise introduction. Other interesting results have been noted in the process.

1 Introduction

The real-world datasets outside of educational purposes are usually full of noise, and the occurrences of noisy data can significantly impact the meaningfulness of classification. Many studies including the work of Nazari et al. [2018], Gupta and Gupta [2019] have shown that noise in a dataset can significantly decrease classification accuracy and produce poor prediction result.

In general, there are two types of noise analyzed by Zhu and Wu [2004], which are attribute noise and class noise. In this project, we aim to introduce class noise to an existing dataset by various degrees for a handwritten digit recognition task, and explore the robustness of two modern neural networks (Artificial Neural Network and Convolution Neural Network) and two common machine learning classifiers (K-Nearest Neighbor, Support Vector Machine). The other classifiers are out of the scope of this project. To alleviate the effect of class imbalance after introducing noise to a balanced dataset, we perform downsampling to each class, so as to remove class imbalance from the scope of this project.

2 Methods

2.1 Dataset

The dataset we are using is MNIST created by LeCun et al. [1998], which consists of 70,000 images of handwritten digits. Among the 70,000 images, 60,000 are designated as training, and represent the handwriting of 250 individuals. The remaining 10,000 are designated for testing, and the sets of

*These authors contributed equally to this work

Preprint. Under review.

individuals in training and testing are disjoint. In addition, each image is 28x28 gray-scale pixels, and the digits are all centered at their center of visual mass. While MNIST only provides 10,000 images in the testing set, which is often considered too small to provide meaningful confidence intervals, we also use QMNIST proposed by Yadav and Bottou [2019] with 60,000 training images and 60,000 testing images.

2.2 Preliminary

Since we don't want the experiments to be influenced by variants such as the number of samples in the datasets and the hyper-parameters of the classifiers, we firstly explore how many samples are necessary for each classifier to reach a stabilized classification accuracy. Also, we will pre-determine the hyper-parameters of the classifiers in the following sections, as the goal is not to reach highest accuracy but discover the robustness of these classifiers on noisy datasets.

2.2.1 K-Nearest Neighbor(KNN)

As one example of a common classifier we've deployed one nearest neighbor classifier using the sci-kit learn implementation. The number of nearest neighbors was set to $k = 10$. Hyper-parameter tuning was not performed because, as mentioned previously, the absolute performance of the model was not one of our main concerns. Euclidean distance was used as the model's comparison metric. We fit the model to the same 50,000 samples used to train the other models in the following sections.

2.2.2 Support Vector Machine(SVM)

We utilized SVM as one of the common machine learning classifiers. We used a linear SVM without tuning any hyperparameters for simplicity in repeated testing and because accuracy was not the main objective in our analysis. The default parameters in the scikit-learn package include using an RBF kernel. SVM used 50000 images for training and 10,000 images for testing, similar to the other common algorithm, KNN.

2.2.3 Neural Networks (NNs)

We've deployed two modern neural network architectures, artificial network network (ANN) (i.e., fully connected network) and convolutional neural network (CNN), to compare and contrast with the other traditional machine learning algorithms. The ANN architecture is shown in Figure 1. It takes the flattened 28x28 pixel image as an input, and uses ReLU as activation in the first layer, hyper-tangent function as activation in the second layer, followed by a Softmax function to classify from the 10-class output.

The CNN architecture is shown in figure 2. Firstly, it takes a 28x28 pixel image as an input, goes through downsampling by a series of convolution with kernel size of 3x3, ReLU activation and max pooling with filter size of 2x2, to generate the low-level feature maps of the images. Then, those features are input into the fully connected layers for classification. Finally, a dropout operation was used to reduce overfitting right before the output layer.

During Training, we were using 50,000 images for training, 10,000 for validation and 10,000 images for testing. We used cross entropy as our loss function and Adam optimizer with a learning rate of 0.001. Then, we trained both our CNN and ANN models and saved the best model among 10 epochs on Google Colab using Nvidia Tesla P100 GPU. The reason for us to use 10 epochs is that during pre-training of 100 epochs we found after around 10 epochs the validation loss of both networks started to increase, which likely would introduce overfitting. All hyper-parameters mentioned above as the baseline are fixed for the later experiments, since we're not trying to improve the accuracy of the neural networks but to explore how the performance will be effected by noisy dataset.

[3]This figure is generated by an online tool from `http://alexlenail.me/NN-SVG/LeNet.html`.

[4]This figure is generated by adapting the code from `https://github.com/gwding/draw_convnet`

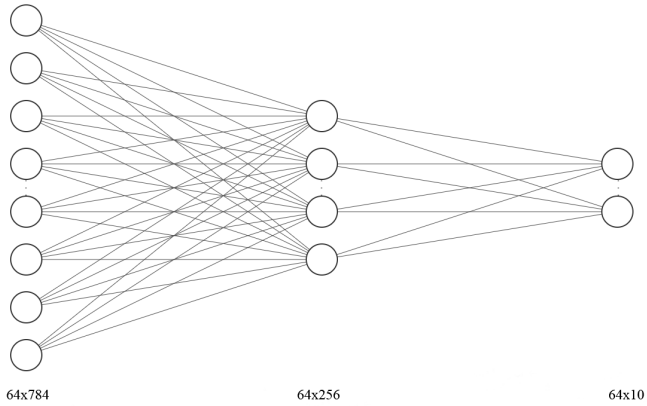

Figure 1: A 2-layer (1-hidden layer) fully connected neural network with batch size of 64, made by online tool NNSVG[3].

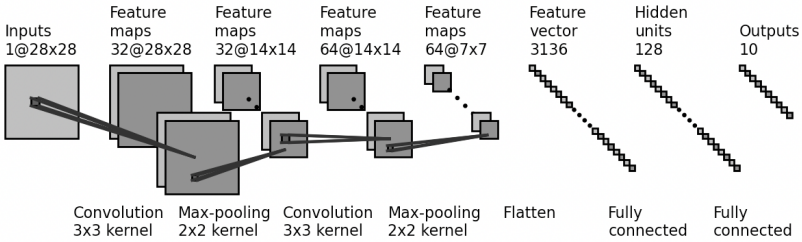

Figure 2: Neural Network Architecture of ConvNet[4].

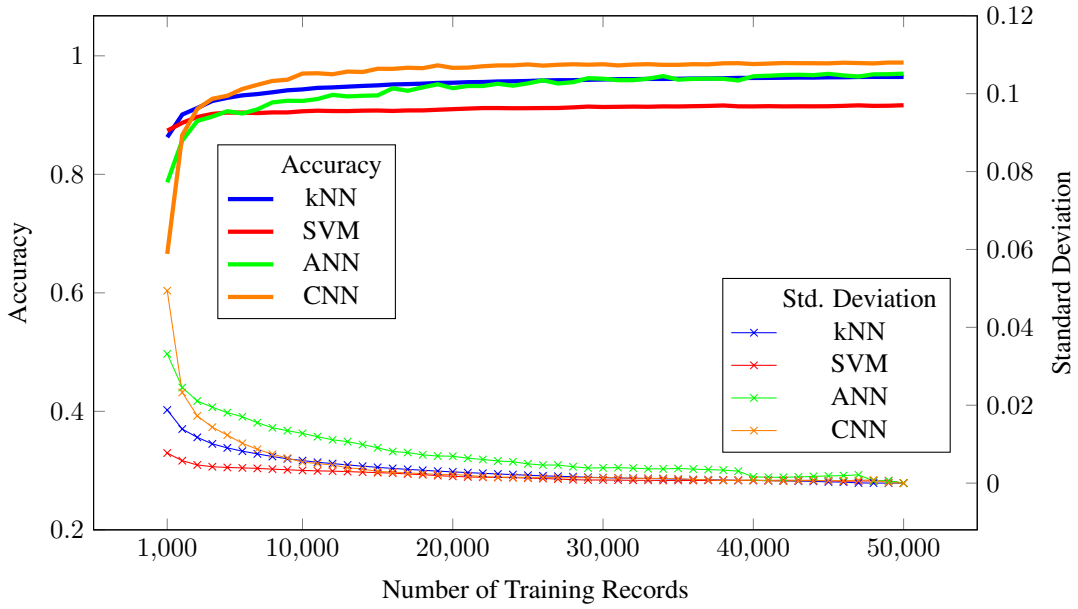

Figure 3: Classifier accuracy and running standard deviation of accuracy vs. number of training records. Each point is the average of 10 runs where for each run a random subset of the training set is selected. The calculation for running standard deviation is described in 2.3

2.3 Training Set Size

Prior to introducing noise, we had to discern the number of training samples at which the accuracies of each of our models stabilize. Doing so allowed us to isolate any changes in accuracy after adding noise from changes in accuracy that might have occurred due to small training sets. In order to determine the target threshold we trained each model on varying sizes of training sets (from 1,000 to 50,000 records), and recorded the average of 10 samples of the accuracy of each model at each training set size. The resulting plot is shown in Figure 3.

Alongside the accuracy, we calculated a running standard deviation to reference when deciding on a threshold. This measure was created by taking the standard deviation of the accuracy from each size of training set $n \in [1000, 50000]$ to the maximum 50,000. To decide on a target threshold we calculated the point at which 90% of the change in standard deviation had occurred. The point at which the 90% condition was met was between 23,000 and 24,000 records. To leave room for error, we took 25,000 as the minimum threshold for training dataset size across all four models in our experiments.

2.4 Noisy Dataset

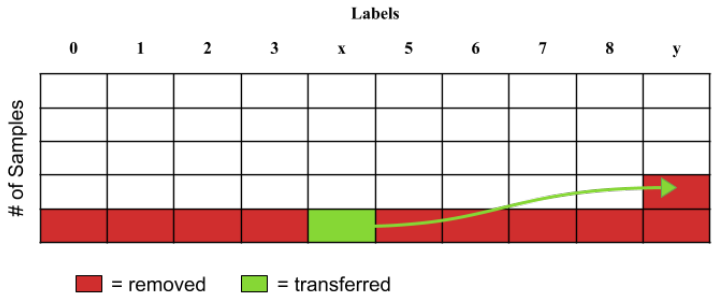

Figure 4: Graphical representation of class balancing with single-flip noise.

2.4.1 Single-Flip

We created single-flip noisy datasets by first truncating the QMNIST dataset of 60,000 samples to 5,000 samples each class in order to balance them (for a total of 50,000 samples). We kept them in an order that was maintained throughout the subsequent processes. Next, to introduce noise for testing a flip from label x to label y (note: $x \neq y$) but keep classes balanced in the training set, we separated n samples from each class except class y, from which we separated $2n$ samples. Then the n samples which we had separated for class y were labeled as class x and reintroduced to the dataset. The resulting dataset had a total of $50,000 - 10n$ records and $5,000 - n$ records for each class. All records are in their corresponding true class except for n records in class y. Thus, the percentage of noise introduced to that class is $\frac{n}{5,000-n} \cdot 100\%$. We did this for $n \in \{0, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500\}$.

Figure 5: Two examples of the digit 4 from the dataset.

The problem with choosing random sets at every iteration is that flipping certain samples could lead to more damage to the model than others. For example, we know that models had a relatively high rate of labeling true 4s as 9s. Since most models create a decision boundary based on the the set of features as (normalized) pixel-values themselves, the way a certain sample is written matters. As

shown in Figure 5, 4s have two primary subtypes. However, one of these is closer to the image of a 9. Flipping the subtype on the left in 5 would have a lesser influence on the decision boundaries of 9 than flipping the subtype on the right.

A possible solution to this problem is to take the average of a certain number of runs. However, time limitations did not allow us to do this, so we were not able to do repetitions for each n value chosen (with the exception of the neural networks in some cases). To fix the problem described earlier, we performed these tests with overlapping sets of samples removed or flipped. That is, the samples removed and flipped in an earlier iteration were also flipped or removed in the subsequent iterations. Thus, we expected to measure a monotonic decrease in performance, similar to the usual effect of noise on models if we were to average multiple runs.

2.4.2 Random-Flip

Additionally, to compare with the single flip datasets we created a series of random-flip datasets. Each of these were constructed in a similar way to the single-flip, by separating n samples and changing their associated labels. Instead of selecting one class to flip from and another to flip to, each class (0 through 9) was selected from, and each group n was distributed randomly across all other classes. In this way, each dataset remained balanced since an equal group n from each class was distributed randomly across all other classes. We did this for $n \in \{0, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 1000, 2000, 3000, 4000, 5000\}$.

3 Results

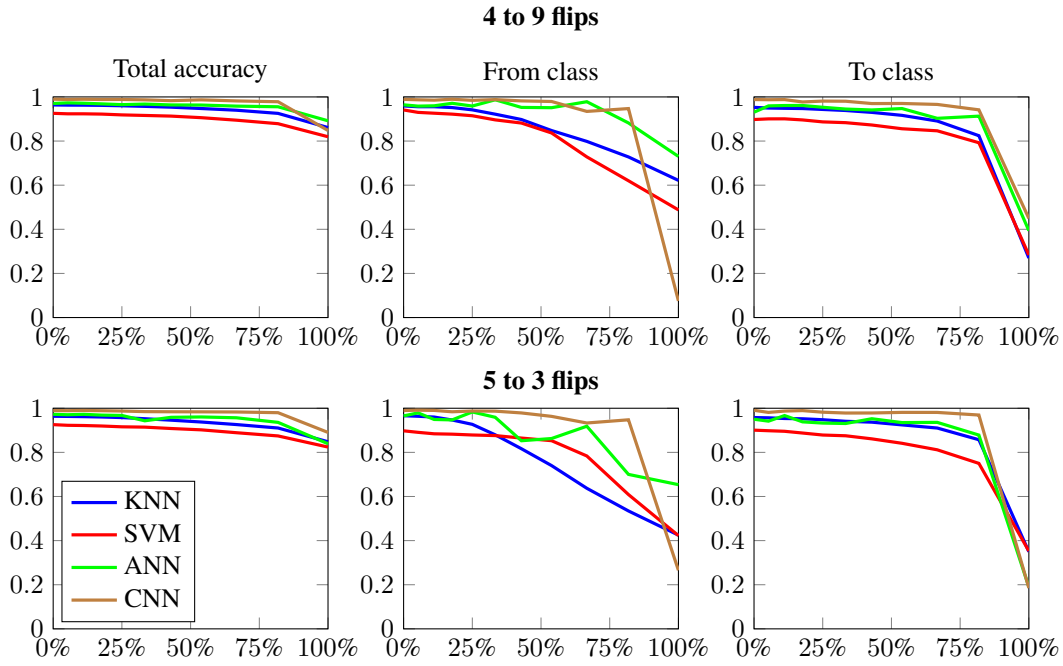

Figure 6: Accuracy of classifiers over increasing percentages of introduced noise from flipping labels from 4 to 9, and 5 to 3 (see section 2.4.1).

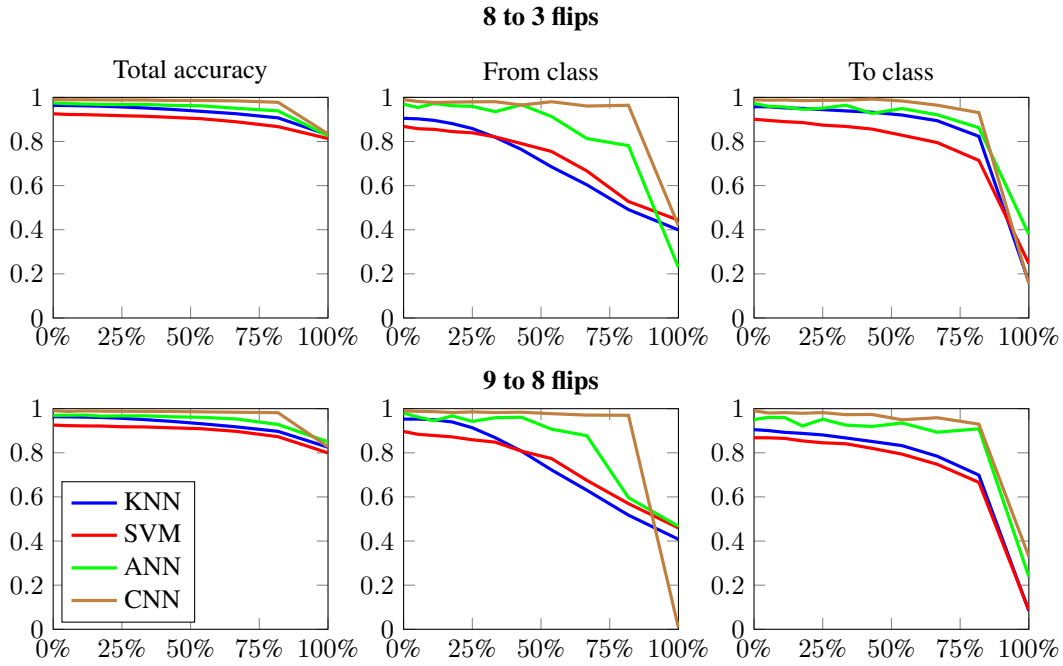

Figure 7: Accuracy of classifiers over increasing percentages of introduced noise from flipping labels from 8 to 3, and 9 to 8 (see section 2.4.1).

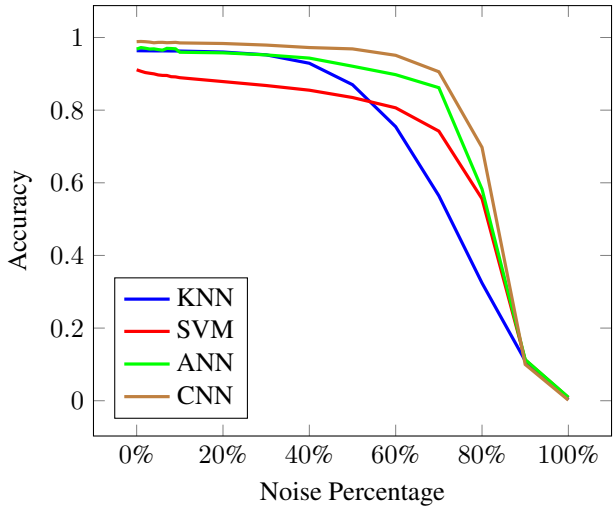

Figure 8: Accuracy of classifiers over increasing percentages of noise introduced by random flipping as described in 2.4.2.

These accuracies are tested on the testing set (50K samples) of QMNIST, which is 5 times larger than original MNIST (10K), to enhance generality of our results. Figures 6 and 7 show the results of total classification accuracies for all classes, the accuracies of a single class whose labels are flipped, and the accuracy of a noisy class (i.e., the class to which those labels are flipped), with respect to different scales of noise that was introduced. Since this is a multi-class classification task, introducing noise to a particular class will not impact the overall performance too much. As expected, the more noise that is introduced, the accuracies of SVM and KNN decrease steadily with tiny steps. Notably, neural networks almost have no performance decline until having more than around 85% of noise is introduced to a certain class. In addition, for the class from which the label is flipped, the accuracies

of KNN and SVM decline smoothly as expected, while the neural networks (especially CNN) don't get negatively impacted too much until more than a half of noise introduced. However, we noticed that there is a tremendous difference for CNN between completely noisy data (i.e., 100% noise) and almost completely noisy data (i.e., 90% noise) for all flips. For the noisy class, all classifiers have a similar trend of curve, where in terms of accuracy the highest is CNN, followed by ANN and KNN, and the lowest one is SVM.

In most situation, we observed that neural networks handled noise better than the others. The reason maybe that they have the ability to iterate through the training set multiple times (epochs) and use the most promising model among those epochs in prediction, having compared to KNN and SVM which only iterate through the dataset once. This may explain with while 80% of labels being flipped for a class, neural networks are capable to attain fairly high accuracy with only that 20% of useful and real information. On the other hand, when the labels of a class was entirely flipped, the classification ability of CNN for this class declined dramatically, because it's trained based on completely misleading information; meanwhile, both SVM and KNN attained around 50% of accuracy for this class, which is significantly higher than a random guess (i.e., 10%).

4 Discussion

The previous results were to be expected because when the 2500 samples in the class that samples are being taken from when noise is 100 percent should be classified well. Then there should be some error in mistakenly classifying the class that samples were transferred to as the class they were taken from. There were some surprising results in Figures 7 and 6 where the CNN model shows sharp decreases towards 100% noise. We think this might have been caused by the neural network optimizing for a decrease in validation loss for all of the classes. In doing so, it inadvertently decreases the accuracy for the class in question. While the networks can perform especially well on data that does not have an overabundance of flipped labels, it also performs especially poorly on data that does. If the network is able learn weights that produce a lower validation loss on the noisy dataset (by learning the misclassification), it means that the network will perform especially poorly at correctly classifying the records of the class whose labels were flipped. This may be the reason that two of the graphs for CNN have accuracies that are similar to the other models (5 to 3, 8 to 3, Figures 7, and 6) while the other two display results that are much lower (4 to 9, 9 to 8, Figures 7, and 6).

Figure 3 describes some of the characteristics of the decision boundaries of the models. Overall, we did see a clear plateau, but none of the models showed this as soon and consistently as the SVM model did. In fact, it was also the best model for training on 1,000 samples. This shows that the RBF kernel's interpretation of space worked well with separating the spaces where different classes were. Thus, SVM is a favorable model for a low number of records.

The consistent performance of the SVM model shows how its decision boundaries do not tend to be altered as easily by choice of records. The standard deviation values from the kNN model are noticeably higher until well over 20,000. This confirms our intuition about the kNN model's decision (based on a small subset of the records). Even with a large number of records, the accuracy is still affected by the choice of records, since the kNN algorithm takes an average of k (=10 in this case) neighbors.

Table 1: Cosine similarities between the frequencies of the 90 possible classification errors.

Cosine Similarity				
	kNN	SVM	ANN	CNN
kNN	1.0	0.73684	0.68894	0.57987
SVM	0.73684	1.0	0.71636	0.68786
ANN	0.68894	0.71636	1.0	0.65372
CNN	0.57987	0.68786	0.65372	1.0

From Table 1 we can tell that since the cosine similarities of each pair is mostly high (always >0.5), the errors that each of these classifiers makes is comparable in shape to the others. This suggests that we could create a general signature of errors for the digits. That is, it may be effective to describe these errors in a summarized fashion across all models. We decided to produce the following summary,

where the error frequency of a certain flip in a model is weighted so that all the entries for that model sum to 1. Then we average these values across the four models and rank them, as shown in Table 2.

Table 2: The errors ranked by weight so as to represent average relative abundance of this error in a model. (1 = incorrectly classified, 0 = correctly classified).

Rank	Predicted	Actual	Weight
1	9	4	0.062938
2	3	5	0.052571
3	8	9	0.043028
4	2	7	0.038202
5	4	9	0.033853
6	3	8	0.032394
7	7	9	0.031950
8	1	7	0.031323
9	0	2	0.030827
10	5	8	0.028819
11	7	2	0.027428
12	3	9	0.023665
13	0	6	0.020628
14	5	3	0.020551

Table 2 provides a summary of key classification errors to focus on in order to improve the accuracy of the models most productively. As noted earlier, it represents the the most frequent errors on average (we allowed all four models an equal total sum when averaging them). This could be helpful in future use cases when you have a different model that has errors have different frequencies to compare the advantages and disadvantages of error handling in that model compared to the ones present.

Table 3: Jaccard similarities between samples classification correctness (1 = incorrectly classified, 0 = correctly classified).

Jaccard Similarity				
	kNN	SVM	ANN	CNN
kNN	1.0	0.15247	0.16819	0.053790
SVM	0.15247	1.0	0.13192	0.034341
ANN	0.16819	0.13192	1.0	0.067449
CNN	0.053790	0.034341	0.067449	1.0

The Jaccard similarities in Table 3 between the vectors generated from the classification correctness (1 = incorrectly classified, 0 = correctly classified) tell a different story. Since these numbers remain low (always <.2 when not on the diagonal) the overlaps are actually small. This builds a strong case for stacking different models for improvement of accuracy.

Taken together, these tables show that while there is a predominant source of mistakes on a higher level (e.g. actual 4s predicted as 9s), a look at the lower level (individual samples) shows that the pairwise intersection of the samples incorrectly classified is relatively small compared to its corresponding pairwise union of incorrectly classified samples. As a result, we can say that these models, similar to us, may make classification mistakes of a certain kind, but tend to make mistakes on different samples.

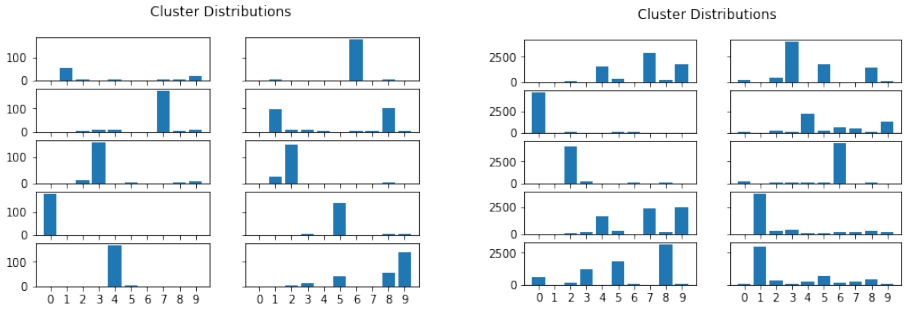

Figure 9: Clustering on MNIST data set with 8x8 resolution (left) and 28x28 resolution (right).

We performed k-means clustering on a small dataset of 8x8 resolution and a larger dataset of 28x28 resolution images. While we were hoping to gain insight on the machine learning model errors by investigating individual clusters, the 28x28 resolution images were poorly separated. As shown in Figure 9, the clusters on the left are better separated by label than the clusters on the right. This is an example of the k-means algorithm performing worse in a higher-dimensional setting, and it also exposes a flaw in the methodology of using clusters to understand errors better.

The idea of using k-means seemed favorable initially because we could take an SSE measure to describe clusters (or even individual point distances from the center), but the SSE measure becomes less meaningful in $28 \cdot 28 = 784$ dimensions and k-means will bias towards globular shaped clusters. If we were to continue to pursue k-means and similar algorithms, we must first notice that it will take effort to separate the clusters. This, unfortunately, does not lead to the desired outcome of using clustering in this project. Ideally, there would be a natural formation of clusters, and adequate measures would be taken to identify how close samples are to a prototype (which is difficult for density-based clusters). Once we attempt to cluster the data points to better separate the classes, the measures we may use for evaluation are no longer the product of an untouched algorithm, but instead the effect of our attempts to improve clustering. Thus, we ruled out the idea of using clusters to predict samples that are difficult to classify. We would also like to note that there may very well be a fundamental incompatibility with clustering and decision boundaries like those from kNN, which can take on arbitrary shape. This different interpretation of boundary formation can likely render this analysis of little practical value.

5 Conclusion

Although any model is bound to have some amount of error when extrapolated to other data sets, we can identify what errors are commonplace overall and which errors appear in certain models. Additionally, we can figure out which model is more robust in handling these errors by flipping classifications of the identified errors and testing accuracy of the models against the amount of noise present. Thus, we find how these models are affected by, and their robustness to, the noise. In the end, we can identify the most robust model, the frequency of the specific types of errors we can look into to increase the models accuracy in the future, and whether or not different avenues (e.g. model stacking) are desirable paths for further research.

Extensions of this project include completing the set of figures for individual flips. Although we built figures and compared four single-flip errors that were in at least one pairwise intersection of the top five errors by frequency of each models, our set of figures is by no means exhaustive. It also does not accurately describe the errors which are unusual in a model. We know that from the cosine similarities in 1 we can create a summary of the most common errors. However, this similarity measure may prioritize the alignment of larger error values to a greater extent (small value-0 pairs that do not intersect do not play as much of a significant role since the square of a considerably small value will not contribute significantly to the norm of a vector, thereby not significantly decreasing its similarity). Thus, further projects could attempt to better identify unique types of errors for a model. These projects could develop a fingerprint for each model by paying special attention to particular errors and analyzing why a certain kind of decision boundary provokes them. Moreover, these projects should also seek to identify how beneficial fixing these (presumably easier to catch) less frequent errors would be compared to the more frequent ones identified in this paper.

We also suggest bi-directional flips (no need to balance classes). By investigating these, we can track the interaction between the two classes as their decision boundaries are altered by these changes. This may also show cases where certain models (such as SVM because of its rigid decision boundaries) may work particularly well for one class instead of another. That is, if the bidirectional interchange of samples between two classes results in one decision boundary gaining more space than the other.

References

Shivani Gupta and Atul Gupta. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474, 01 2019. doi: 10.1016/j.procs.2019.11.146.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

Zahra Nazari, Masooma Nazari, Mir Sayed Shah Danish, and Dongshik Kang. Evaluation of class noise impact on performance of machine learning algorithms. *IJCSNS*, 18(8):149, 2018.

Chhavi Yadav and Léon Bottou. Cold Case: The Lost MNIST Digits. *arXiv*, 2019.

Xingquan Zhu and Xindong Wu. Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review*, 22(3):177–210, 2004. ISSN 0269-2821. doi: 10.1007/s10462-004-0751-8.

Contributions

- *Subhankar* Emotional support. Guided with a helpful paternal hand as to where we should focus our efforts. Increased our chances of success with inherent luckiness. Always asked interesting questions.
- *Reuben Gardos Reid* Created the KNN model scripts, and ran than to collect all data for the KNN model. Wrote the script to generate overall noisy dataset. Gathered data and created figures 3, 6, 7 and 8 for the final report. Gathered code to organize and present on GitHub. Contributed to mainly the methods section of the final report, and added portions to the results and discussion sections as well.
- *Gaoxiang Luo* Came up with this project and started the project proposal. Read papers for literature review for overall structure of the project. Implemented the baselines of ANN and CNN, as well as the initial dataset script of MNIST and QMNIST and distributed to other group mates.
- *Max Schickert* Worked on the SVM model to get data and significant results: Training samples vs accuracy and errors. Helped put together Overleaf report: Introduction, Discussion, Conclusion. Sometimes asked interesting questions.
- *Gustav Baumgart* Contributed to the analysis of this paper (primarily the discussion and conclusion). Worked with the clustering algorithms. Wrote two scripts for clustering different datasets (8x8 and 28x28 resolution images) and a script to retrieve information for tables 1-3. Most significant contribution: drafting the methodology and organizing the team's next steps in order to best compare models.