University *of Ljubljana*
Faculty *of Mathematics and Physics*

Department of Physics

Machine Learning: Classification with Neural Networks

13. Task for Model Analysis I, 2023/24

Author: Marko Urbanč
Professor: prof. Dr. Simon Širca
Advisor: doc. dr. Miha Mihovilovič

Ljubljana, August 2024

Contents

1 Introduction

Machine learning is one of those things that seem like a futuristic, near-magical, technology, however that is not the case anymore. Use of machine learning algorithms has become routine and common place in many fields of science and industry. Many people are not even aware that they constantly encounter machine learning algorithms in their everyday life, despite the fact that more and more people talk about the fabled *algorithms* that are prevalent on the internet and social media. It is indeed true that most search algorithms like Google's or recommendation algorithms like Netflix's are based on machine learning, but the use of machine learning is not limited to these applications. Machine learning is now practically omnipresent.

I consider myself a proponent of data privacy and online safety so I'd like to perhaps further elaborate on the use of machine learning algorithms in the above mentioned contexts of search and recommendation algorithms. Take Netflix for example. Netflix is an online streaming service for movies and TV shows. Due to its massive library of content it can be hard for users to find something they would like to watch. This is where recommendation algorithms come in. These work by analyzing the user's watch history and preferences from which they can try to predict what the user would like to watch next [1]. This on its own is not a bad thing but the issue lies in what other data is gathered for use in these algorithms. Netflix for example tracks not only which content you watch and for how long but also from which device you view the content, at what time of day and so on. All this data is used to create something Netflix calls *Context* [2]. The algorithms that power these recommendations are called *contextual bandits*, which are a type of reinforcement learning algorithm. We'll discuss the specifics of machine learning algorithms a little later. But Netflix plans to be even more invasive than just that. For example in the paper *Using Navigation to improve recommendations in real-time* [3], authors describe how they'd like to forecast something called the user's *intent* with which they would attempt to know what the user would like to watch before the user actually provides any input. Since this paper was published in 2016, it is likely that Netflix has already implemented this feature to some degree. This is just one example of how machine learning algorithms are used to gather data on users. Oh it's worth mentioning for all those that share Netflix accounts with their friends or family, that Netflix is also actively developing and improving machine learning algorithms to recognize account sharing [4].

Moving on to Google. Google is foremost a search engine which is used by billions of people every day. Google uses machine learning algorithms to provide the best search results for the user and has been doing so for quite some time. In the year 2015 they introduced a deep learning system called *RankBrain* which was designed to improve search results for queries that Google had never seen before. RankBrain works by evaluating past search queries and the results users clicked on to determine how best to improve search results for future queries [5]. From 2018 onwards, Google uses neural networks in its search algorithms and in 2019 Google introduced *BERT* [6], a neural network-based technique for natural language

processing.

As the last contribution to my small rant on machine learning algorithms in everyday life, I'd like to mention possibly the worst offender of them all, TikTok. TikTok is a social media platform where users can upload short videos of themselves. It's content is served up entierly by machine learning algorithms on the *For You* page. The *For You* page is a feed of videos that the algorithm thinks you would like to watch. The service again collects copious amounts of data on its users and since it is owned by a Chinese company, there are concerns about data misuse and privacy [7]. So much so, that the US government is trying to push a bill that would force the sale of TikTok to an American company, which would better ensure that data on US citizens does not leave the country [8]. Besides the privacy concerns there are also concerns about the content that is served up by the algorithm. In a sense the algorithm is maybe too good at what it does, which results in TikTok's mostly young users being exposed to harmful or even violent content, mostly in the form of videos that depict self-harm and suicidal ideation or their glorification as stated in a report by Amnesty International [9]. Another interesting phenomenon is what has come to be known as *algospeak*, where users that are aware of the algorithm's censorship of certain words and phrases, try to circumvent the censorship by using different words or phrases that mean the same thing. For example, "unalive" instead of "kill", "le$bean" instead of "lesbian", "sewer slide" instead of "suicide". Evading censorship is nothing new but it is interesting to see how the use of such words is spreading to platforms that do not censor them and even to everyday language [10]. These are examples of the most harmful uses of machine learning algorithms and with the advent of Large Language Models (LLMs) like GPT-3 and GPT-4, the potential for misuse and misinformation is even greater.

1.1 Machine Learning and Neural Networks

Enough about the bad things. It's good to make people aware of the potential dangers of machine learning algorithms but it's also important to talk about the good things. Machine learning has the potential to revolutionize many fields or already has. Today's example will be about classification with neural networks using the MNIST dataset. Let's briefly list the basic types of machine learning algorithms. They are:

- Supervised learning:
  - Classification: Used for sorting data into different classes/categories. For example: Recognition of handwritten digits.
  - Regression: Used for modeling the relationship between variables. For example: Predicting the price of a house based on its size.
- Unsupervised learning: Used for finding patterns in data. For example: Clustering of data points.
- Reinforcement learning: Used for training agents to make decisions. For example: Training a robot to walk.

In physics we mostly use supervised learning algorithms for data analysis. One of the most important discoveries in which supervised learning was used is the discovery of the Higgs boson at the Large Hadron Collider in 2012. We've even had a go at this ourselves at the end of the pre-graduate subject *Matematical Physics Practicum* where we studied the use of neural networks and decision trees for the classification of the Higgs boson.

How do machine learning algorithms even work? In general we can imagine we have a set of parameters $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^{N}$ where $x_k = (x_k^1, \ldots, x_k^M)$ is a vector of M features (representation of the data) and y_k is the corresponding label (class/category). Values of (x_k, y_k) are called samples and are independent and statistically distributed according to some unknown distribution $\mathcal{P}(\cdot,\ \cdot)$. The goal of machine learning is to find/learn a function $h : \mathbb{R}^Q \to \mathbb{R}$ that minimizes the loss function with witch we determine how well h performs on the data. The loss function can be defined in general as:

$$\mathcal{L}(h) = \frac{1}{N} \sum_{k=1}^{N} \mathrm{L}(\mathrm{y_k}, \mathrm{h}(\mathrm{x_k}))\,, \tag{1}$$

where L is some smooth function that describes the difference between the true label y_k and the predicted label $\boldsymbol{h}(x_k)$. We sample the data $\mathcal{D}$ randomly and divide it into two sets: the training set and the test set. The training set is used to train the model and the test set is used to evaluate the model's performance.

The model is trained by minimizing the loss function using various optimization algorithms. The most common optimization algorithm is the gradient descent algorithm.

In the case of neural networks, we can imagine we have a mesh of interconnected neurons that are organized into layers. Each neuron is connected to every neuron in the previous layer and every neuron in the next layer. The first layer is called the input layer, the last layer is called the output layer and all layers in between are called hidden layers. Hidden layers unfortunately present somewhat of a *black box* to us, although there are methods we can use to try and understand what is happening in the hidden layers. We'll take a look at the end of this task. There exists a plethora of different subtypes of neural networks such as convolutional neural networks (CNNs), which are good at recognizing patterns in images, recurrent neural networks (RNNs), which are good at understanding speech and so on.

The basic building block of a neural network is the *perceptron*, which is described by the equation:

$$h_{\boldsymbol{w},b} = \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \,, \tag{2}$$

where $\boldsymbol{w}$ is the weight vector, b is the bias and σ is the activation function. The activation function is a non-linear function that introduces non-linearity into the model. The most common activation functions are the sigmoid function, the hyperbolic tangent function and the rectified linear unit (ReLU) function [11], which are given as:

$$\sigma(x) = \frac{1}{1+e^{-x}} \,, \tag{3}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \,, \tag{4}$$

$$\mathrm{ReLU(x)} = \max(0, \mathrm{x}) \,. \tag{5}$$

Nowadays the ReLU function is the most commonly used activation function since it is computationally efficient and is more resistant to the so-called *vanishing gradient problem* that can occur during training of the model.
However it is more susceptible to the opposite problem known as the *exploding gradient problem*. We won't go into the details of these problems here as they are not really that important for the understanding of the task. We train the neural network with an optimization algorithm such as the gradient descent algorithm and backpropagation which is used to calculate the gradient of the loss function with respect to the weights and biases of the network. The last thing we have to specify is a loss function. The most common loss function for classification tasks is the cross-entropy loss function which is given as:

$$\mathcal{L}(\boldsymbol{w}, b) = -\frac{1}{N} \sum_{k=1}^{N} \left[y_k \log(\boldsymbol{h}_{\boldsymbol{w},b}(x_k)) + (1 - y_k) \log(1 - \boldsymbol{h}_{\boldsymbol{w},b}(x_k)) \right] \,. \tag{6}$$

We can also use something simpler like the mean squared error loss function, given as:

$$\mathcal{L}(\boldsymbol{w}, b) = \frac{1}{N} \sum_{k=1}^{N} (y_k - \boldsymbol{h}_{\boldsymbol{w},b}(x_k))^2 \,. \tag{7}$$

2 Task

2.1 Classification with Neural Networks

The instructions task us with classifying the MNIST dataset of handwritten digits using a neural network. We are to explore various architectures of the neural network and values for hyperparameters such as the learning rate, the number of hidden layers etc. We need to look at which numbers are the hardest to classify and why. We can also try to feed the trained model with images that are completely different to see how the model responds. Figure 1 shows the average image of each digit in the MNIST training set.

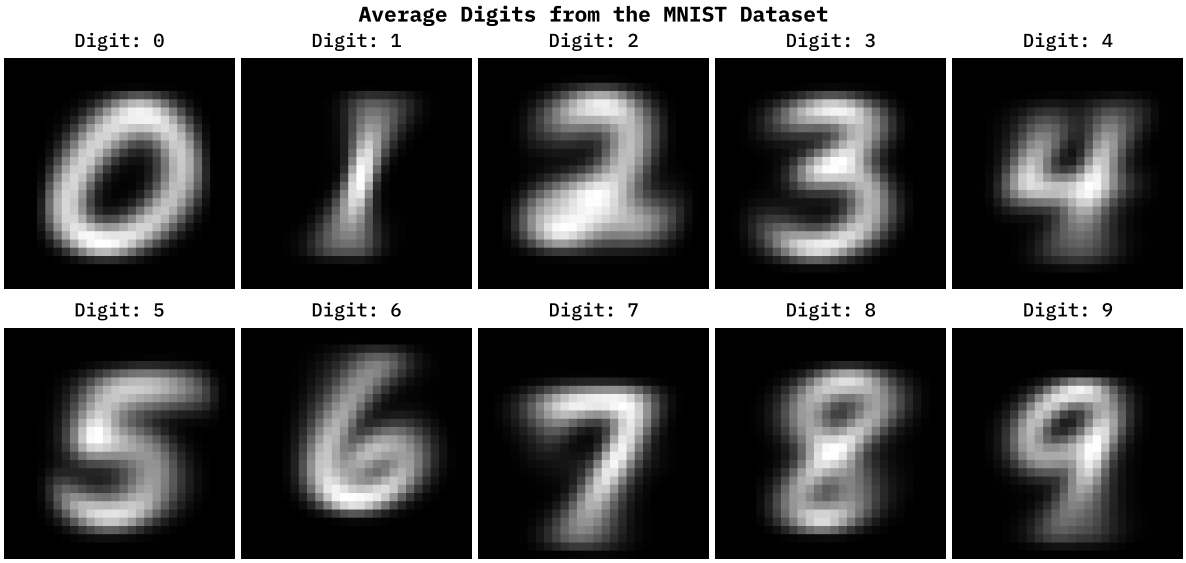

Figure 1: Average image of each digit in the MNIST training set.

2.2 Deep Dream and Class Maximization

The second part of the task is to use the trained model to generate images that maximize the output of a certain class. This is done by using the gradient of the output of the model with respect to the input image. This is called class maximization. We can also study which input minimizes the loss function of the model the most. This is called deep dream, where we can see which features the model is most sensitive to.

3 Solution Overview

I started this project very motivated back during the semester. I was certain that I was going to make my own implementation of a neural network and that it was going to be done in CUDA C++. This turned out to be very naive of me as I certainly did not think about the scope of such a project. Thus the Task was put on hold until the summer, where I promised myself to take a more pragmatic approach. Besides using the standard libraries for data manipulation and visualization like `numpy`, `matplotlib` and `scipy` I also used `torch` for the neural network implementation. and `scikit-learn` for model evaluation. I used the `torch.nn` module to define the neural network and the `torch.optim` module to define the optimization algorithm. I tried to make coherent information dense plots to present the results of the various models. The results are presented in the next section.

4 Results

4.1 Classification with Neural Networks

Jumping directly into the results I've defined models with various values for hyperparameters. All these models share the same architecture which is a neural network with one hidden layer. All the layers are linear layers with a ReLU activation function. The loss function used is the cross-entropy loss function and the optimization algorithm used is the Adam optimization algorithm. The models were trained for 10 epochs, split into different batch sizes. The models are presented in Table 1 as well as their training and testing accuracies rounded to three decimal places. Figures 9 to 19 show the ROC curves for each model for each digit with it's corresponding AUC score. Added are also the micro and macro average ROC curves. The micro average gives equal weight to all classes and shows average performance across all predictions while the macro average shows the average performance for each class.

The ROC (Receiver Operator Characteristic) curves are a good way to visualize the performance of the model at binary classification. Since we have multiple classes I plotted a ROC Curve for each class. The AUC score is a measure of how well the model can distinguish between classes. The AUC

score ranges from 0 to 1 where 0.5 is random guessing and 1 is perfect classification. The AUC score is calculated by taking the area under the ROC curve (hence the name).

Model	Learning Rate	Hidden L. Size	Batch Size	Training Acc.	Testing Acc.
0	0.001	784	64	0.998	0.982
1	0.001	784	256	0.998	0.982
2	0.1	784	64	0.703	0.706
3	0.1	784	256	0.918	0.915
4	0.1	784	16	0.404	0.407
5	0.001	10	256	0.926	0.923
6	0.001	5	256	0.776	0.784
7	0.001	1	256	0.257	0.257
8 (S)	0.001	1	256	0.187	0.189
9	0.001	7840	256	0.998	0.982
10	0.001	78400	256	0.995	0.979

Table 1: Hyperparameters and Results of the various models.

I was shocked to see how well the models performed. So much so that I had doubts about the validity of the results, but upon further inspection I did not find any errors in the code. THe models with the best performance were models 0 and 1. These models had the highest training and testing accuracies. Model 9 performs exactly the same at the cost of a much larger hidden layer size which meant that training took longer. We can see that hidden layer size does not directly translate to accuracy as in model 10 (which took even longer to train) the accuracy is lower than in model 9. From this we can conclude that it does not make sense to have a massive hidden layer, at least not for this dataset.

Commenting more on the topic of hidden layer size we can see how reducing the layer size to very small numbers greatly diminishes the performance of the model. It is interesting that model 5 performed as well as it did but what was more shocking was model 6, with a hidden layer size of only 5 neurons, yet still managed to reach $\sim 77\%$ accuracy. Even having just one neuron in the hidden layer (model 7) the model still managed to reach $\sim 25\%$ accuracy which is still better than random guessing. Model 8 is a special case. The "S" I've added next to its name is there to signify that the output layer has a sigmoid activation function. Clamping the values of the output layer further worsened the performance of the model.

In the case of models 2, 3 and 4 we see that we can somewhat improve the performance of a model with a higher learning rate by increasing the batch size. Model 3 already performs quite well with a learning rate of 0.1 and a batch size of 256. Model 4 was created to further test this hypothesis and we can see that reducing the batch size to 16 severely impacts the accuracy of the model lowering it to around 40%.

From the ROC Curves we can identify that some of the poorer performing models practically have no idea on how to classify certain digits. For example in Figure 13 we can see that the model really has no idea on how to classify the digits $2, 5, 6$ and 8. I assume this is a consequence of these digits being quite similar to each other which causes the model to get confused pretty quickly. This point is taken to the extreme if we take a look at model 8 in Figure 17 where the model doesn't really have a clue on how to classify most of the digits. It only managed to somewhat correctly classify the digits 1 ad 8 which are the two most distinct digits in the dataset. This means that the testing accuracy of 18.9% can actually be quite deceptive as the model is only really marginally useful for the classification of 2 digits. We can of course also see the opposite effect in well performing models, especially in model 0 in Figure 9 where the model reaches an AUC score of 1.0 for the digit 1 and 0.99 for the others. This means that the model is very good at distinguishing between the digits.

I plotted the values of the loss function for most of the models in Figure 2. We can see that the loss function decreases very rapidly in almost all cases. The exceptions are models 7 and 8 where the optimization algorithm has a difficult time efficiently minimizing the loss function due to there being only one neuron in the hidden layer. This issue with optimization is no doubt the reason behind the poor performance of these models, which makes perfect sense. One could imagine that we can boil down the features from the input layer of size 784 to a hidden layer with say 10 neurons but boiling all those features down to just one neuron is not really feasible. Another interesting effect can be seen in model

4 where the loss function decreases rapidly but then starts to show signs of increasing again. This is most likely a sign of overfitting. While the effect is not as pronounced I still find that explanation very plausible, since this was the model with the smallest batch size and the highest learning rate. This means that the model went through many training steps with a high learning rate which can quickly change the loss function.

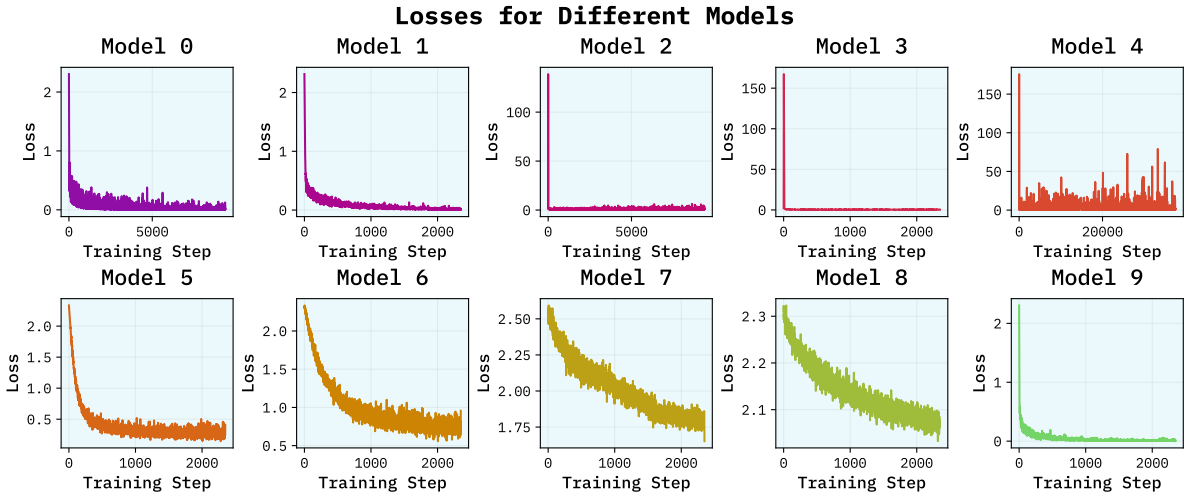

Figure 2: Loss function values for the various models, trained for 10 epochs at different batch sizes.

Since we've discussed the performance of the models at different values of hyperparameters I performed a quick parameter scan for different values of the learning rate and size of the hidden layer. The results of which are presented in Figure 3. The heatmap shows the training and testing accuracies for each combination of hyperparameters.

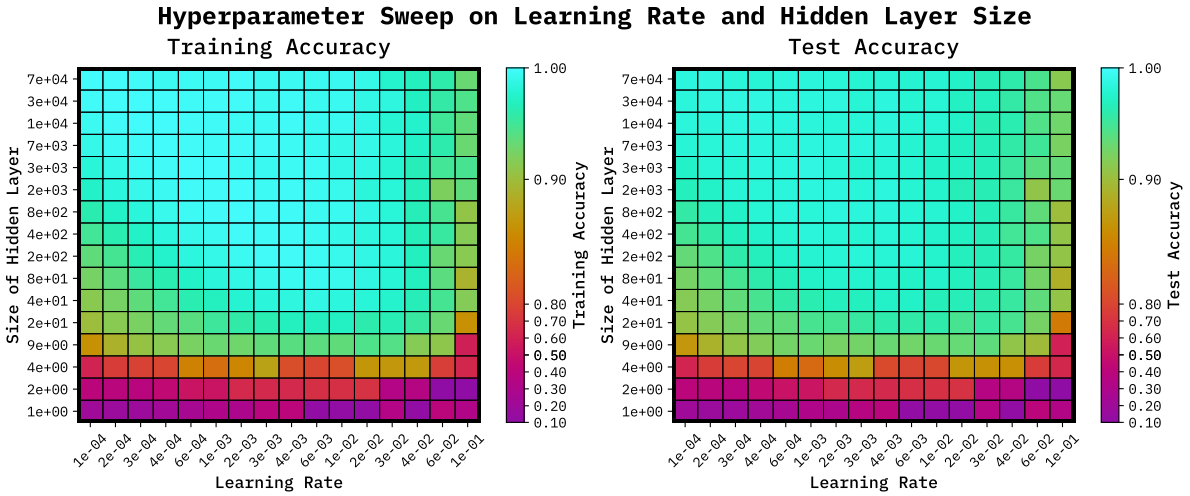

Figure 3: Heatmap of the training and testing accuracies for different values of the learning rate and hidden layer size.

From the heatmaps we can see that it is important to have a reasonably large enough hidden layer size. The model's accuracy seems to be mostly independent of the middle values of the learning rate range, provided the hidden layer has enough neurons. It was very difficult to normalize the colormap in such a way that this can be easily seen that is why I decided to use a combination of a linear and logarithmic normalization. This way we can se that very low learning rates still can yield fantastic results but only for very large hidden layers. Large values of the learning rate seem to consistently diminish the performance of the model. The best tresults were obtained with a learning rate of 0.0004 and a hidden layer size of 7131 neurons.

I also tried a few models with more layers but the performance increases were marginal if any. The results of these models are presented in Table 2, with the ROC curves in Figures 20 to 22. The missing

models are not presented as they were attempts at using the MSE loss function as well as Negative Log Likelihood for the loss function. The models performed quite poorly and I did not have the time to further investigate the issue.

Model	L. 1 Size	L. 2 Size	L. 3 Size	Training Accuracy	Testing Accuracy
12	784	392	196	0.996	0.979
13	784	1568	3136	0.997	0.981
15	784	784	784	0.995	0.979

Table 2: Hyperparameters and Results of a 3 Hidden Layer Neural Network.

Now that we've discussed the performance of the models we can have a look at how the models classified the digits. I evaluated this for models 0 and 7 as they represent the best and worst performing models respectively. The results are presented in Figures 4 and 5. The histograms show which as which digits each digit was incorrectly classified as (essentially a look at all the predicted labels for a certain true label, without the number of times it was correct).

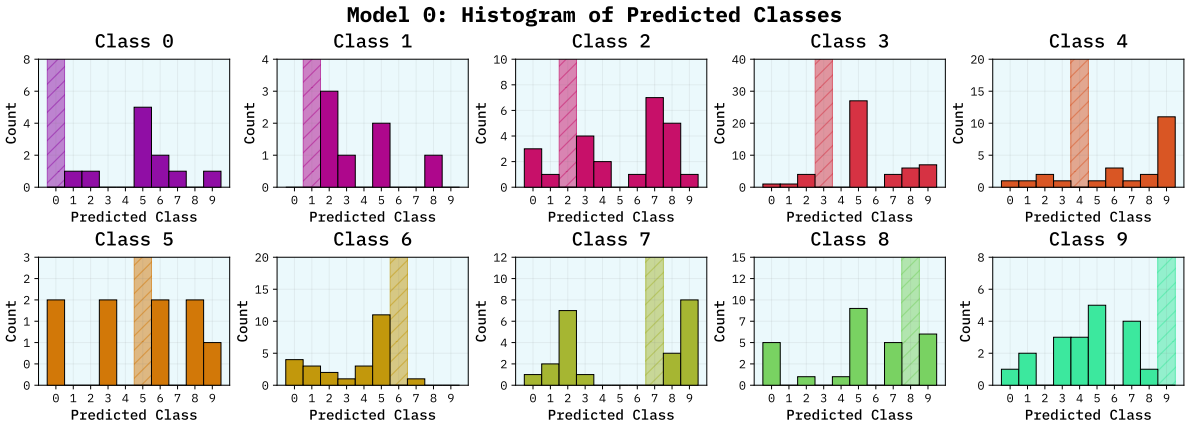

Figure 4: Histogram of the predicted labels for each true label for model 0.

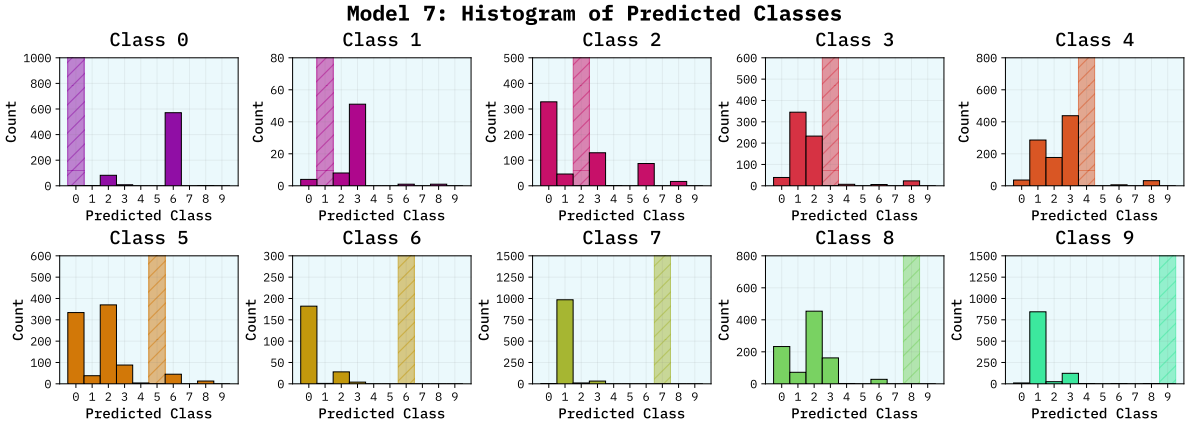

Figure 5: Histogram of the predicted labels for each true label for model 7.

We can see that there is a drastic difference between the two models. Model 0 has an overall very low number of misclassifications. It mostly misclassifies the digit 3 as 5 and 4 as 9. These are quite acceptable misclassifications as some of the images in the dataset are truly very vague even to the human eye. Model 7 on the other hand shows that it is horrible at classification. As seen in Figure 16, the ROC curves show that the model guesses randomly for most digits with the exceptions being the digits 1, 4 and 7. The histogram is consistent with these results. We can see that the digit 1 is the least misclassified digit. But that is not really saying much as the model also likes to misclassify other digits as the digit 1.

I feed the trained model with images that are completely different to see how the model responds. The results are presented in Figure 6. The images are some mouse-drawn shapes that could maybe be interpreted as digits and an image of my cat Happy and a picture of my friend getting roasted at General Relativity class.

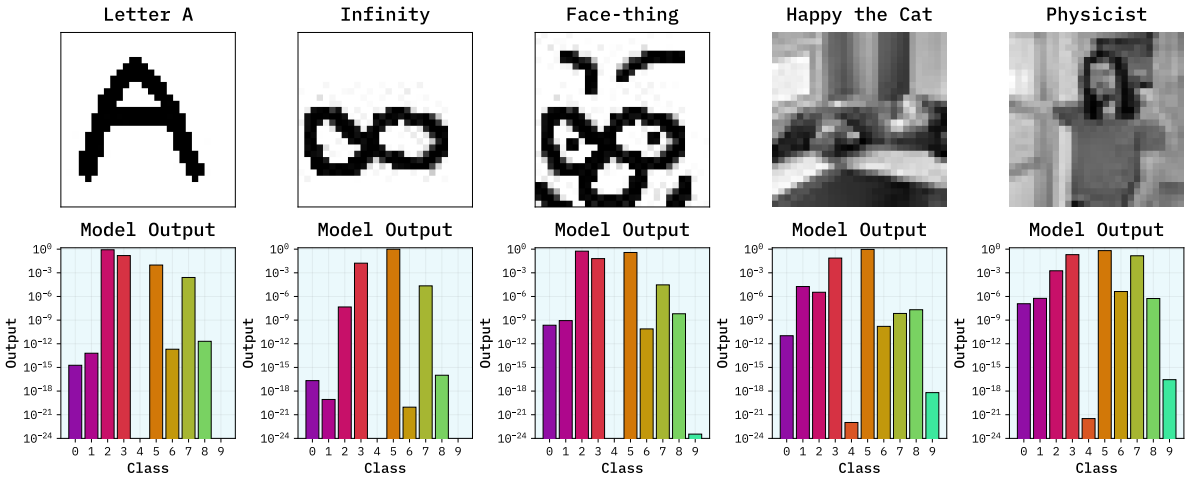

Figure 6: Images fed to the trained model and the corresponding predicted class probabilities.

The model has a strange tendency to call more or less everything a 5. In two cases it called the image a 2, in case of the image of the letter A and the image of a mouse-drawn face. The cat is a definite 5 per my model while the image of my friend is a bit more difficult to interpret as the model gives high probabilities to the digits $3, 5$ and 7. The model certainly got one thing right and that is that he's an odd fellow.

4.2 Deep Dream and Class Maximization

Since model 0 has been our go-to due to it's stellar performance let's allow it to dream up the ideal representation of each digit according to it's knowledge. This is what we called *class maximization* in the introduction. The results are presented in Figure 7.

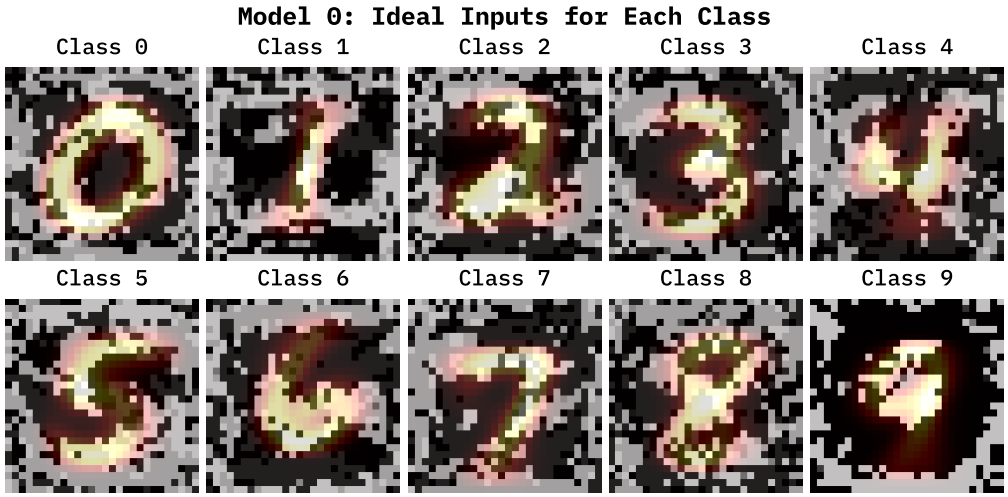

Figure 7: Class maximization for each digit according to model 0.

I plotted the average member of each class as a red to yellow overlay on top of what the model thinks is the ideal representation of each digit. We can definitely see that the model has a clear idea of the core features of each digit. $0, 1$ and 5 stand out to me in particular, while 8 and 9 perhaps require more imagination to see. What I find very odd is the seeming importance of the border of the image for all the digits. I'm not sure what causes this and I tried different inputs to see if the model would still focus

on the border but it seems that it is a consistent feature of the model. Finally lets have a look at what input will minimize the loss function of the model as much as possible. This is known as the *deep dream* from which we can see which parts of the image the model is most sensitive to. The results are presented in Figure 8.

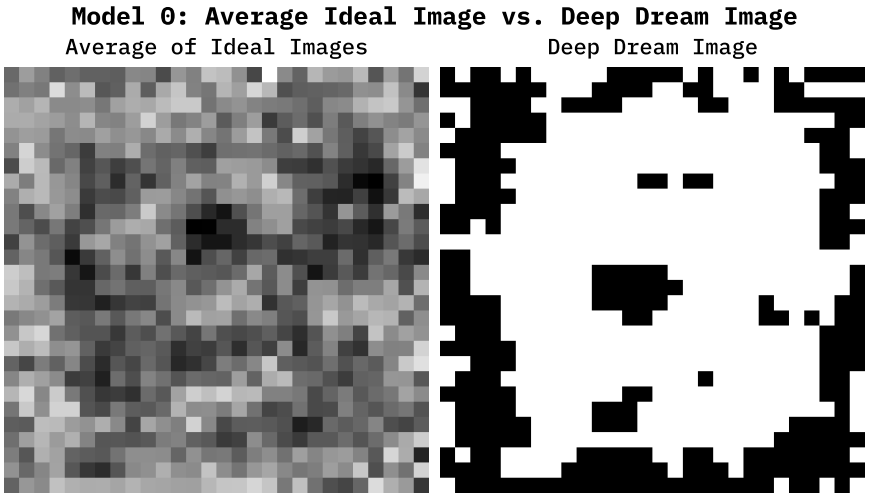

Figure 8: Average ideal image for each class and the deep dream for model 0.

The image is no doubt very strange and honestly quite disappointing. I was expecting to see some clear features of the digits but the image just displays a vague blob in the center with some strange features around the border. It mostly looked like an average of all the digits which would somewhat make sense. I plotted that on the left-hand subplot to compare the two images. The deep dream image is strange in that it only contains values of 0 and 1. I tried multiple inputs, learning rates and numbers of iterations but the result was consistently the same. I suppose the two images are somewhat similar in the most basic shapes. The border is slightly more pronounced and the rest of the focus is on the center of the image.

5 Conclusion and Comments

I had many more ideas for this project but it is unfortunately time to move on, due to upcoming deadlines. I wanted to get a working model using the MSE loss function but I had quite a few issues with formatting my data since MSE is not really suited for classification tasks. This is something I might have explored further if I had more time. I also wanted to explore the use of convolutional neural networks (CNNs) for this task but I didn't have the time to implement them. I think that CNNs would be a good choice for this task since they are good at recognizing patterns in images which is exactly what we're doing. I wish I could have explored the deep dreams for some other models. I think it would have been interesting to see how the model's performance affects the deep dream. Overall though I'm happy with the overall results and impressed at the performance of the models.

References

[1] Sudarshan Lamkhede and Christoph Kofler. Recommendations and Results Organization in Netflix Search. *arXiv e-prints*, page arXiv:2105.14134, 2021.

[2] Harald Steck and Linas et al. Baltrunas. Deep learning for recommender systems: A netflix case study. *AI Magazine*, 42(3):7–18, Nov. 2021.

[3] Chao-Yuan Wu and Christopher V. et al. Alvino. Using navigation to improve recommendations in real-time. *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016.

[4] Soheil Esmaeilzadeh, Negin Salajegheh, Amir Ziai, and Jeff Boote. Abuse and fraud detection in streaming services using heuristic-aware machine learning. *arXiv e-prints*, page arXiv:2203.02124, 2022.

[5] Pandu Nayak. How AI powers great search results. *Google*, Feb 2022. https://blog.google/products/search/how-ai-powers-great-search-results/.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv e-prints*, page arXiv:1810.04805, 2018.

[7] Amnesty International. Caught in TikTok's Surveillance Web, Jan 2024. https://www.amnesty.org/en/documents/POL40/7349/2023/en/.

[8] Dan Milmo. TikTok says it will fight US ban or forced sale after Bill passes. *The Guardian*, Apr 2024. https://www.theguardian.com/technology/2024/apr/22/tiktok-us-ban-or-forced-sale-bill-bytedance.

[9] Amnesty International. Driven into Darkness: How TikTok's 'For You' Feed Encourages Self-Harm and Suicidal Ideation, Jan 2024. https://www.amnesty.org/en/documents/POL40/7350/2023/en/.

[10] Daniel Klug, Ella Steen, and Kathryn Yurechko. How Algorithm Awareness Impacts Algospeak Use on TikTok. In *Companion Proceedings of the ACM Web Conference 2023*, WWW '23 Companion, page 234–237, New York, NY, USA, 2023. Association for Computing Machinery.

[11] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv e-prints*, page arXiv:1803.08375, March 2018.

6 Large Figures

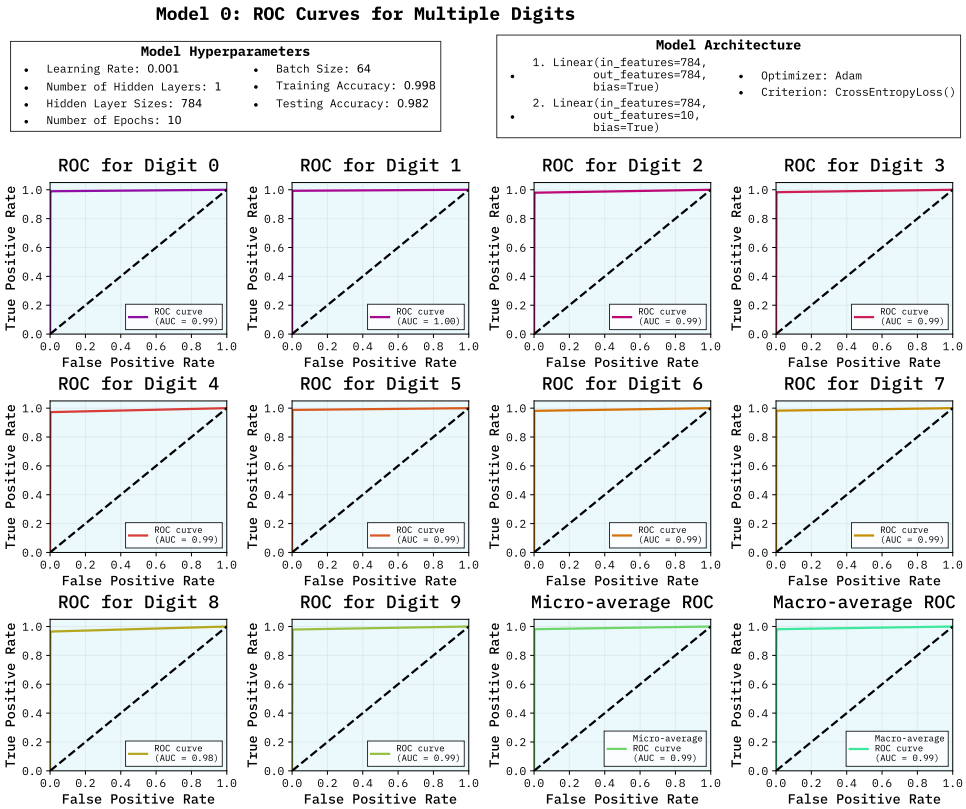

Figure 9: Testing ROC Curves for Model 0.

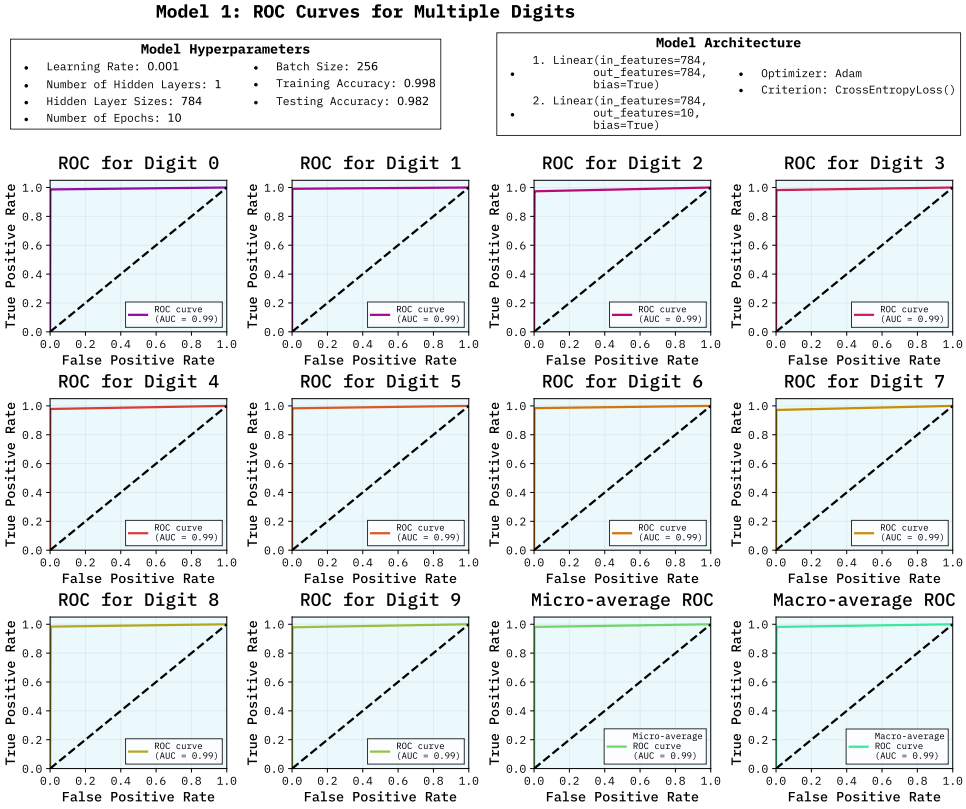

Figure 10: Testing ROC Curves for Model 1.

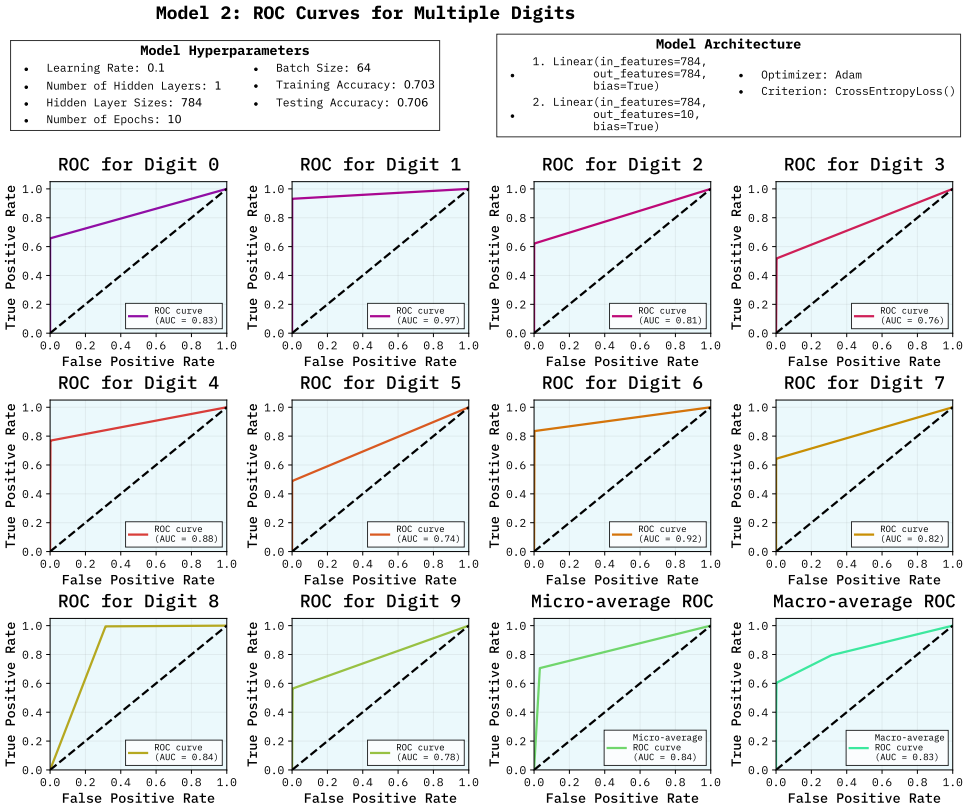

Figure 11: Testing ROC Curves for Model 2.

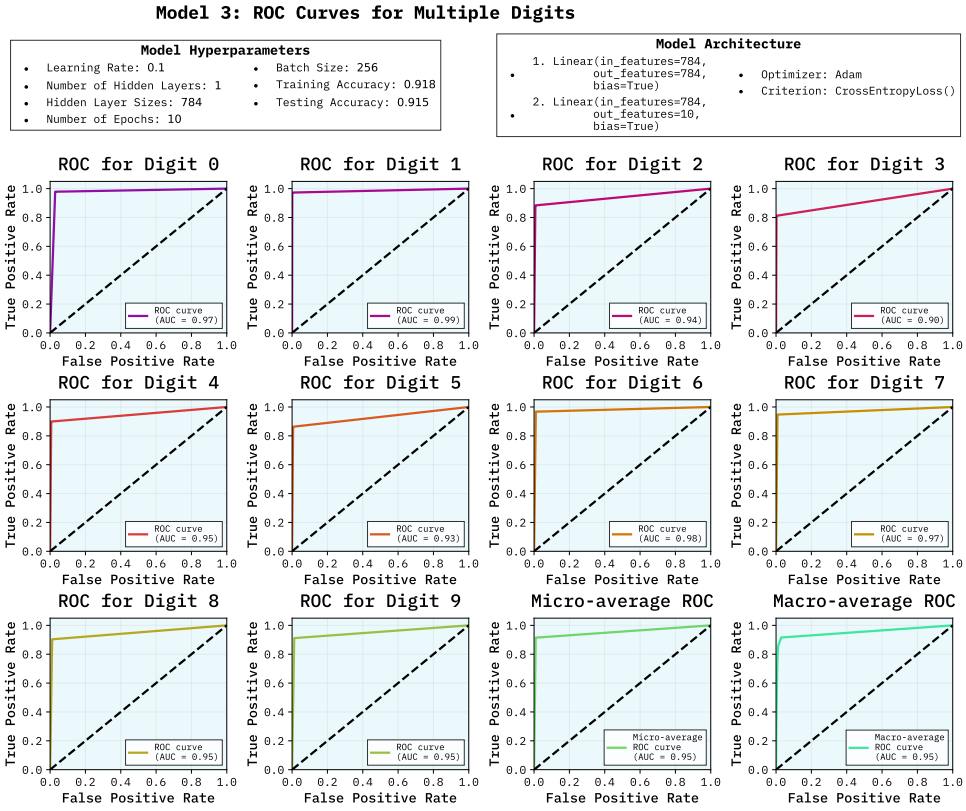

Figure 12: Testing ROC Curves for Model 3.

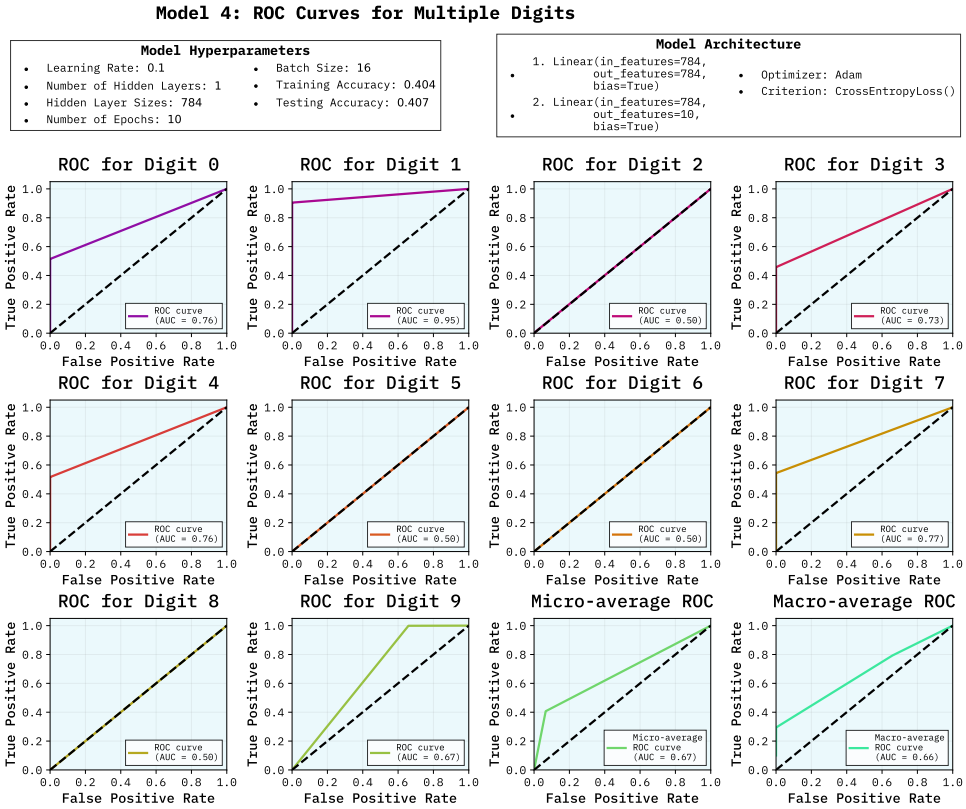

Figure 13: Testing ROC Curves for Model 4.

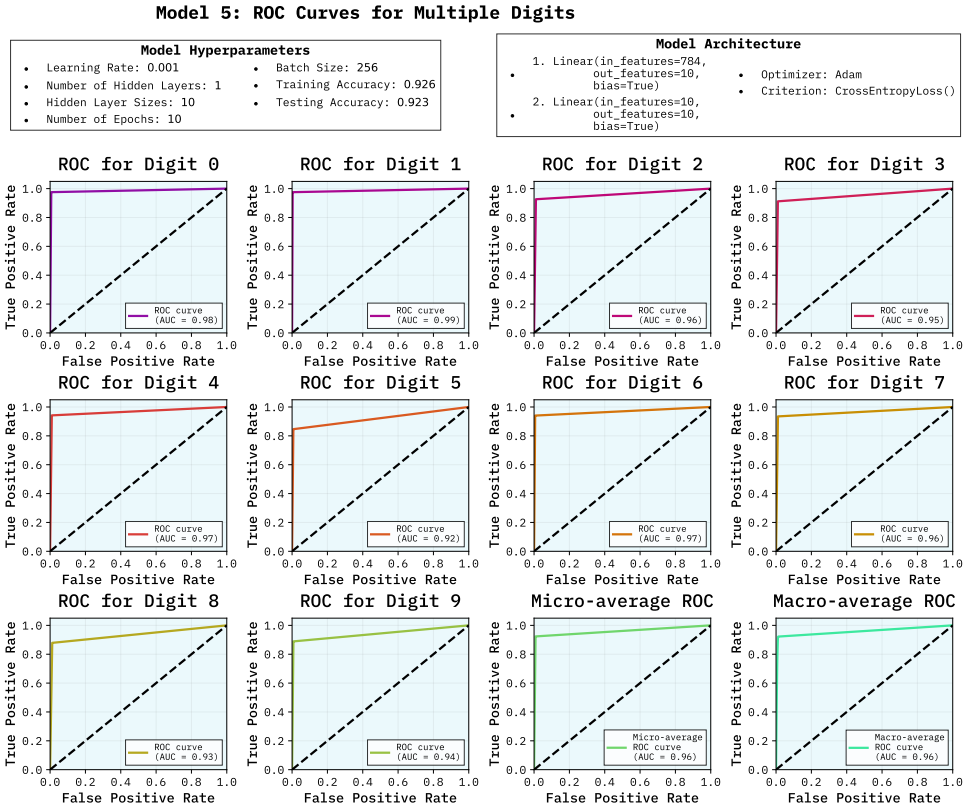

Figure 14: Testing ROC Curves for Model 5.

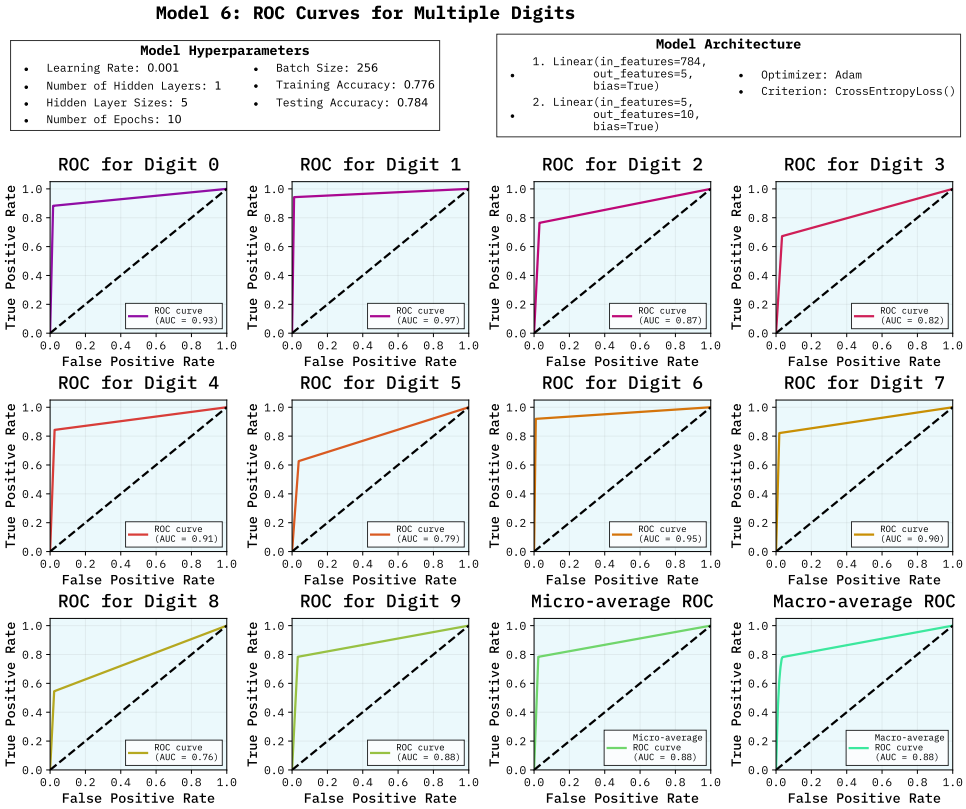

Figure 15: Testing ROC Curves for Model 6.

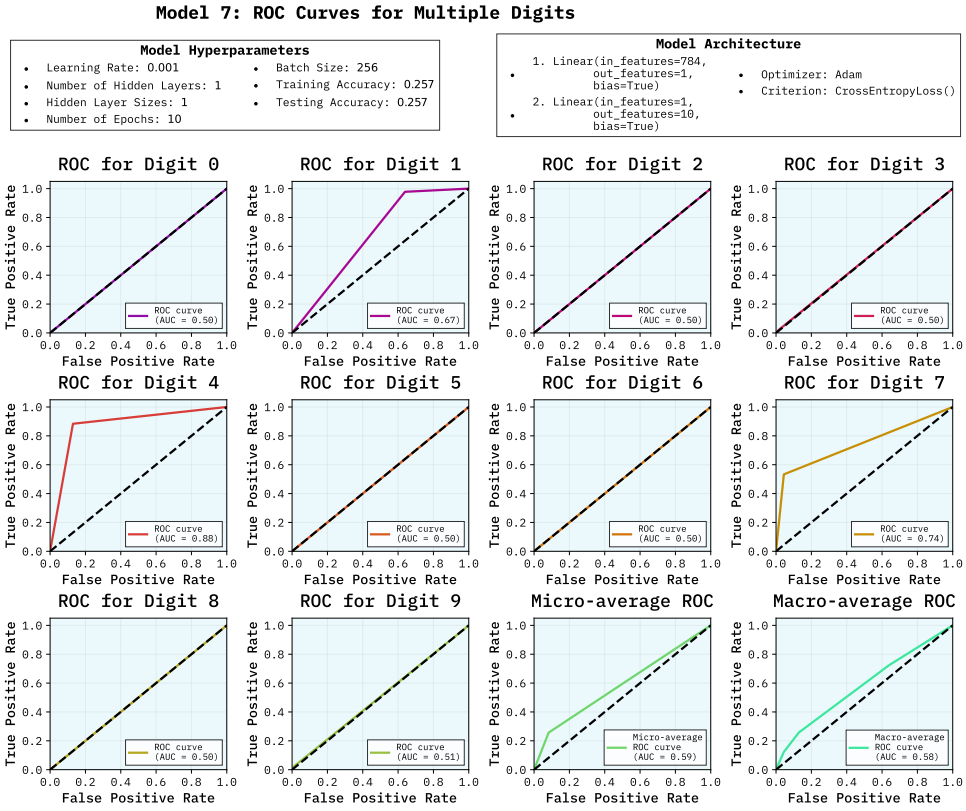

Figure 16: Testing ROC Curves for Model 7.

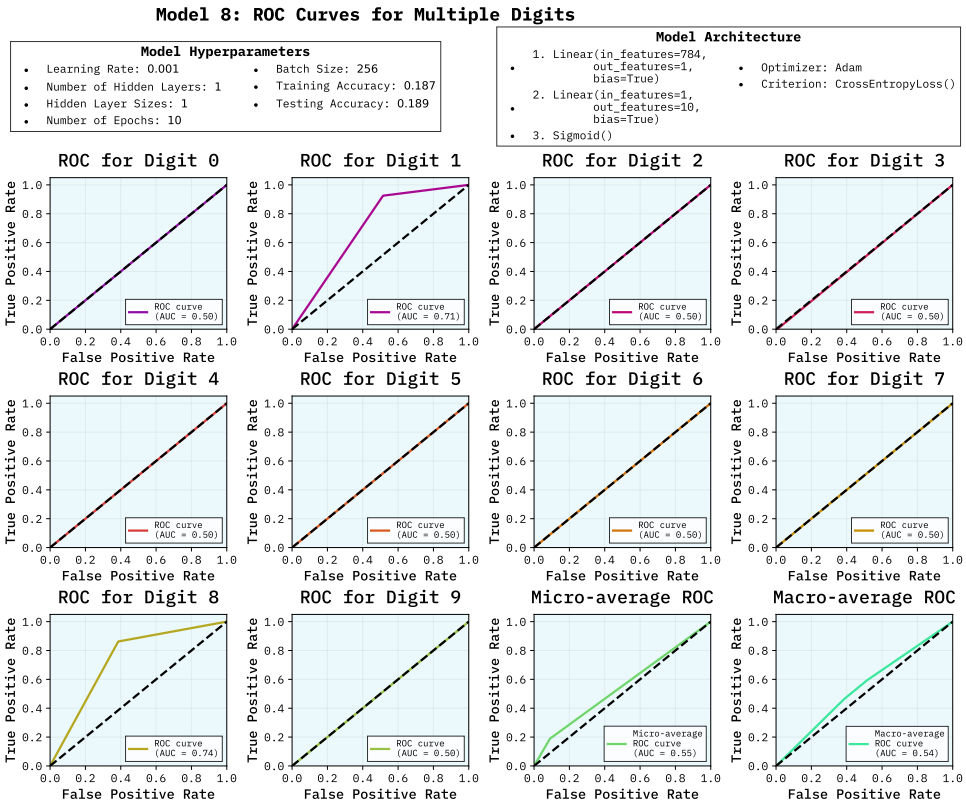

Figure 17: Testing ROC Curves for Model 8.

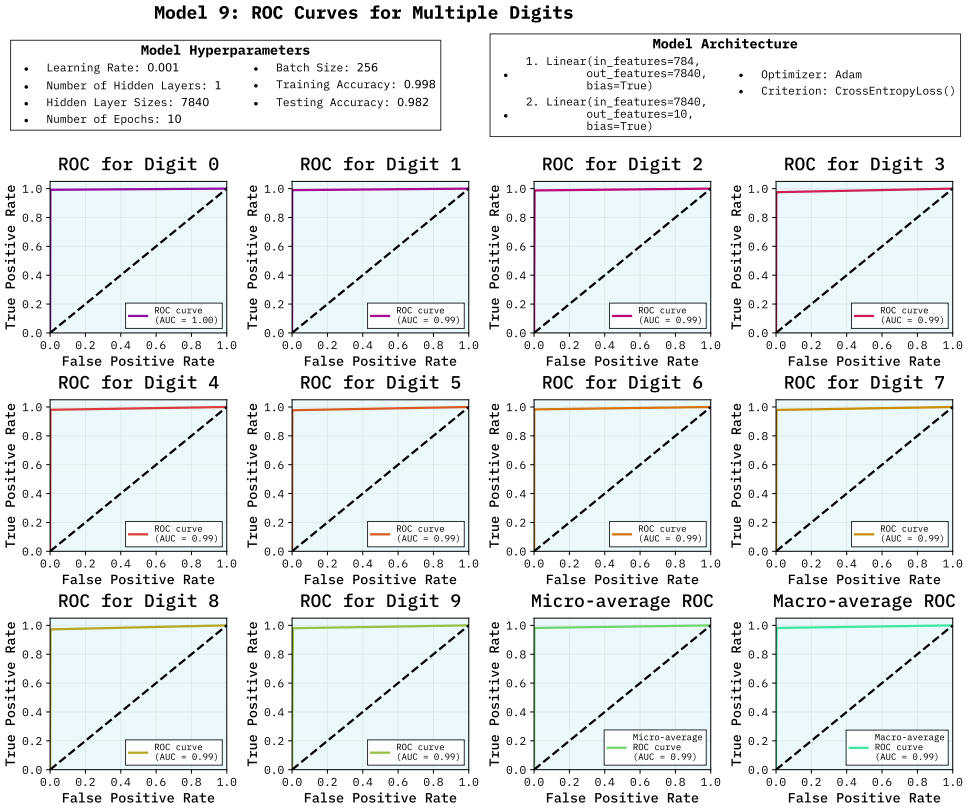

Figure 18: Testing ROC Curves for Model 9.

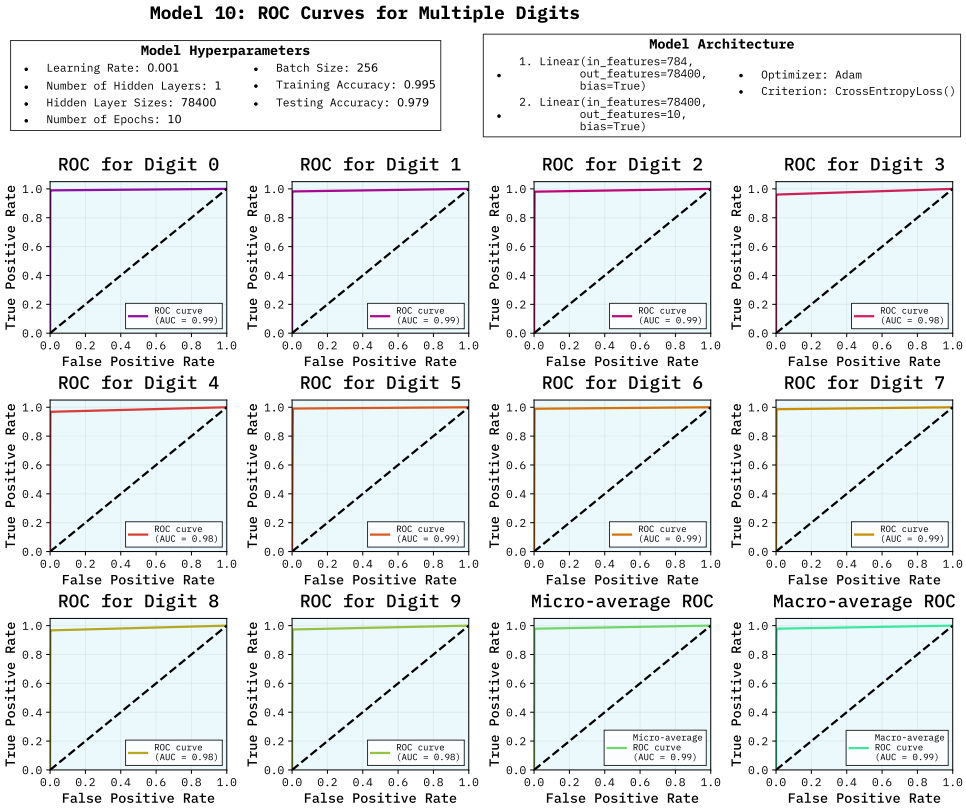

Figure 19: Testing ROC Curves for Model 10.

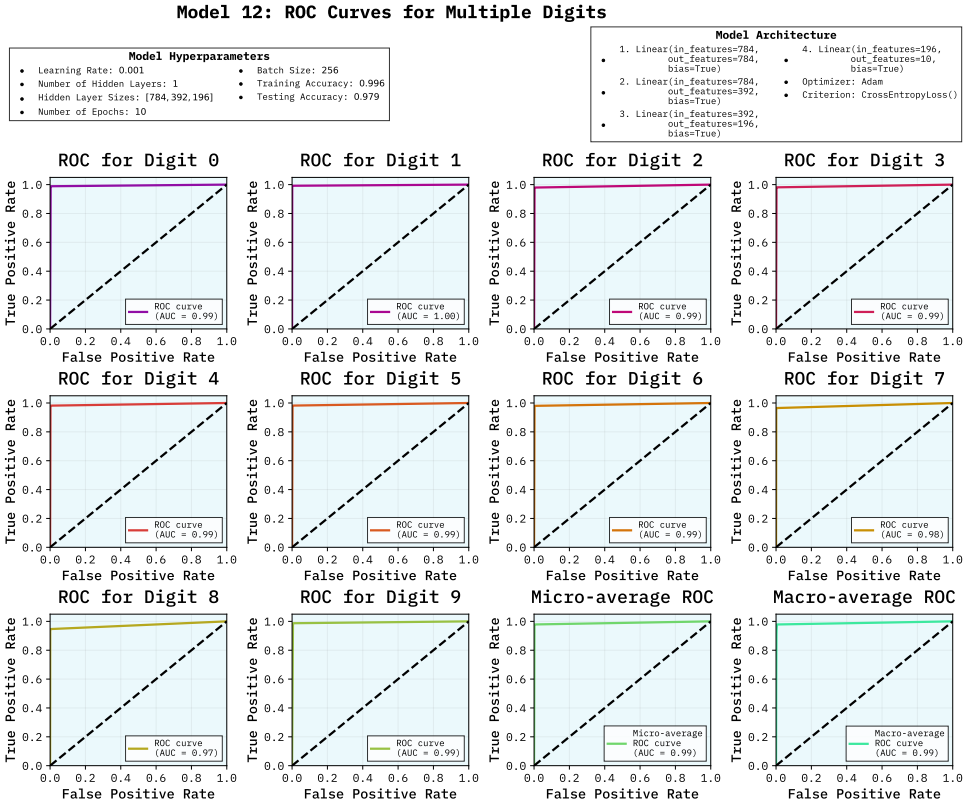

Figure 20: Testing ROC Curves for Model 12.

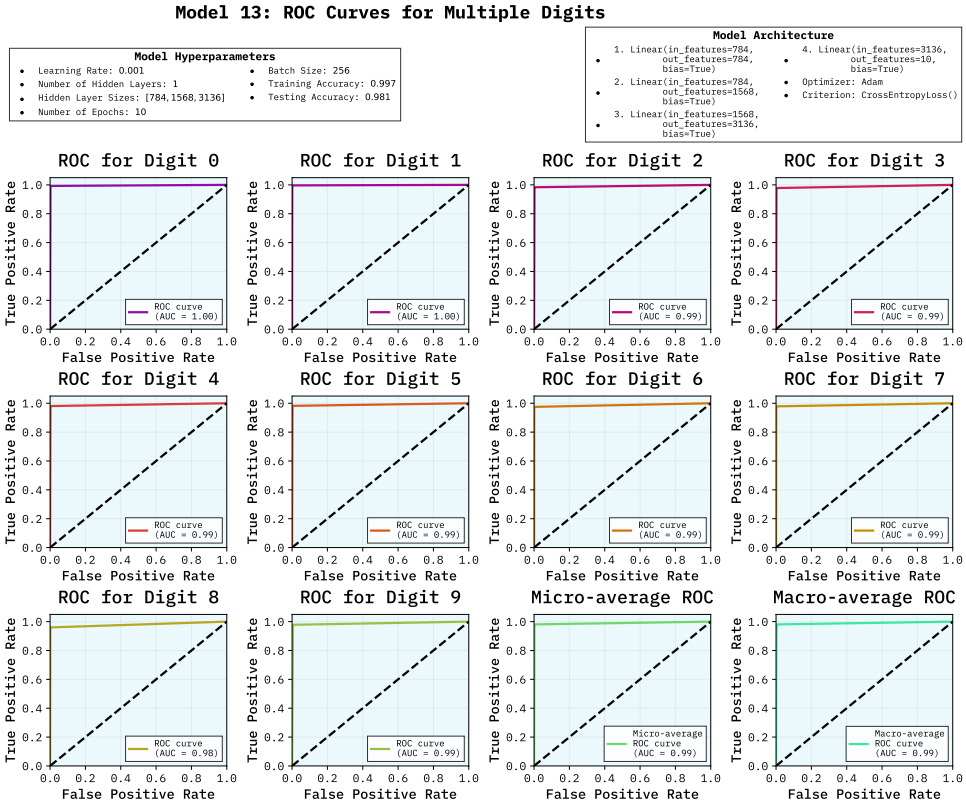

Figure 21: Testing ROC Curves for Model 13.

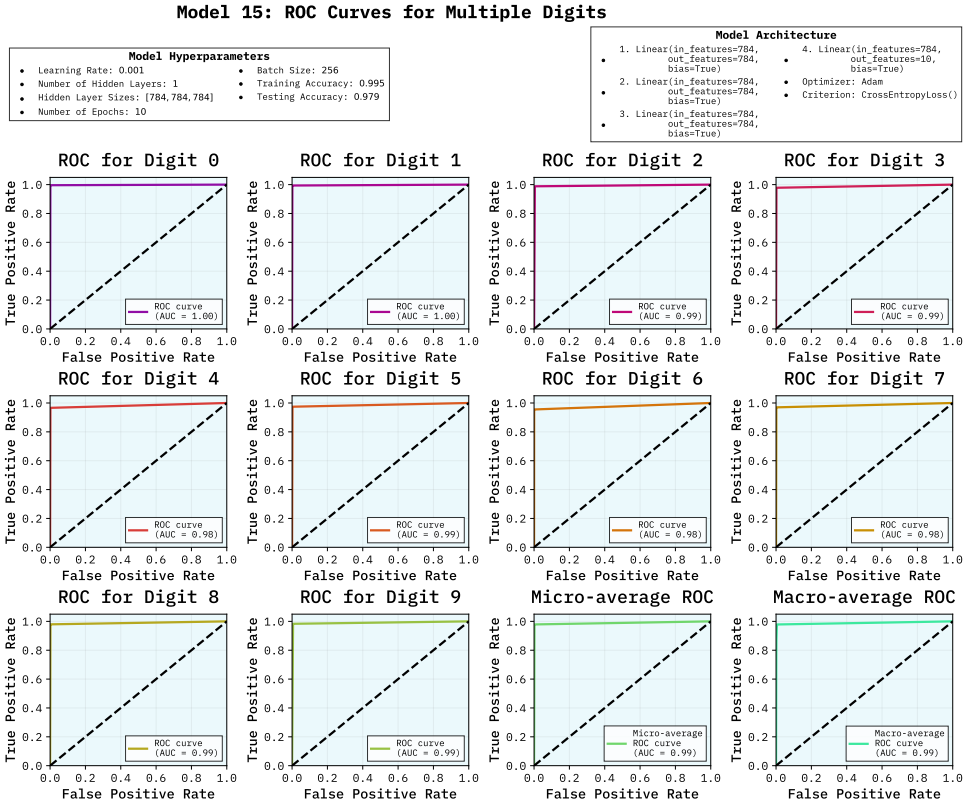

Figure 22: Testing ROC Curves for Model 15.