Convolutional Neural Networks

EE6132 - Lab 2

Anirudh R

November 1, 2021

Contents

1 Introduction

In this assignment, we look to analyse Convolutional Neural Networks(CNN). We will be using the MNIST dataset and will be building a CNN classifier. We will use the PyTorch library to build, train and test the models. We will also perform occlusion experiments to see how the classifier performs. We will also perform adverserial attacks on the trained CNN.

The following table shows the configuration of CNN we will look at.

Parameter	**Value**
Dataset	MNIST
Loss Function	Cross Entropy Loss
Optimizer	Adam
Mini-Batch Size	100
No. of Epochs	6
No. of Convolutional Layers	2
Filter size	3x3 (both layers)
No. of filters	32 (both layers)
MaxPooling	2x2 with stride 2 (both layers)
No. of MLP Layers	2
FC layer sizes	500, 10
Hidden Layer Activation	ReLU
Output Layer Activation	Softmax

2 Concepts

2.1 Dataset

We make use of the **MNIST**(Modified National Institute of Standards and Technology) Dataset here. This is a dataset of hand-written digits and the corresponding labels. This contains **60,000** training data points and **10,000** test data points. The digits have been size-normalized and centered in a fixed-size image (28×28 grayscale image).

2.2 Convolutional Neural Network(CNN)

In the last assignment, we saw that feature extraction from raw image data can be very advantageous in classification. We saw that a MLP trained with inputs being the HOG features of the images performed much better.

The CNN architecture uses the same principle of first performing feature extraction from images and then passing the obtained features through a MLP to do the classification. Here, the feature extraction is done by convolutional layers instead of HOG. This model that has the **convolutional feature extraction module** followed by the **MLP classifier module** has proved to be very good at image classification tasks.

Also, apart from the increased performance, convolutional layers also have lesser parameters compared to FC layers. A single neuron in a convolutional layer only takes input from a limited number of neurons in the previous layer(only F^2 neurons where F is the filter size) unlike FC neurons which take all outputs of previous layer as inputs.

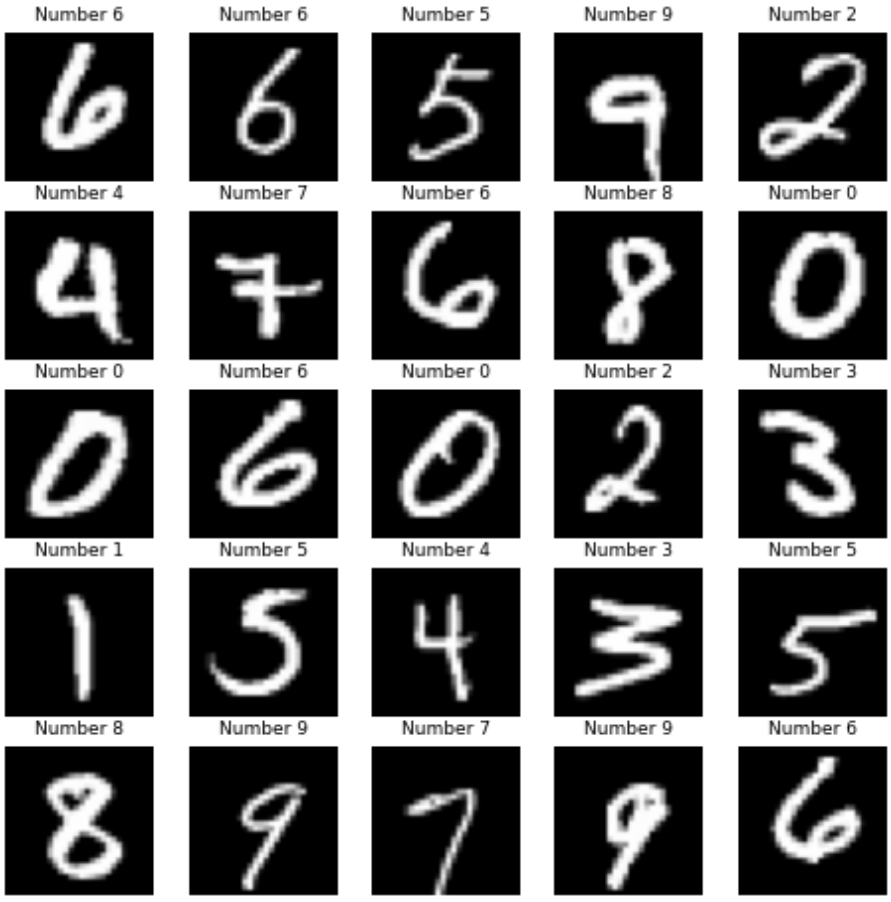


Figure 1: Samples from the MNIST Dataset

This **local connectivity** and reduced receptive field substantially decreases the number of parameters in a convolutional layer.

Apart from this, convolutional layers also have **parameter sharing** which means the weights and biases of all neurons corresponding to a filter in a convolutional layer is the same. This again massively reduces the number of parameters.

Due to the parameter sharing and local connectivity of neurons, a convolutional layer of neurons can be simply represented as the 2D convolution of various filters with the input image.

2.3 Structure

The baseline model involves,

- 32 3x3 convolutional filters with stride 1 and padding 1 in both convolutional layers.
- ReLU activation is used in both convolutional layers.
- Each of the convolutional layers is followed by a MaxPool layer(2x2 filter with stride 2).
- The outputs of the 2nd MaxPool layer is fed into the MLP classifier.
- The MLP classifier consists of 1 hidden layer of size 500 and an output layer of size 10(as there are 10 classes in MNIST).

- The hidden MLP layer uses ReLU activation while the output MLP layer uses Softmax activation as this is a classification task.
- Cross Entropy Loss is utilized as this is a classification task.
- Adam optimizer is used for the training.
- We also train the same network, but with Batch Normalization modules in each of the 3 hidden layers(baseline with batch normalization) to compare with the baseline model.

2.4 Occlusion

We see that the CNN architecture performs very well on the MNIST dataset. But, we want to know if the network classifies an image correctly for the right reasons. We want the network to classify an image correctly because of the number being present in the image and not based on something in the image other than the digit written itself.

To test this, we perform these occlusion experiments where we occlude/cover partches of the image and see how the network performs on this patched image. We will know that the network has learnt the right things if covering background parts and relevant parts of the image doesn't and does decrease class probability respectively.

2.5 Adverserial Attacks

CNNs have proven to be very good models for image classification tasks. But, they do have their weaknesses. In this section of the assignment, we look to understand how CNNs can mis-classify images that may seem very obvious to humans. We do not look to modify model parameters at all. All we look to find are inputs that makes the CNN classify into a particular class.

In **Non-Targeted Adverserial Attacks**, we start with Gaussian noise images and modify the input by performing **Gradient Ascent** on the cost. We will end up with something that may seem noisy but the network classifies it into the target class confidently.

$$Cost \ = \ logit[target\_class] \tag{1}$$

In **Targeted Adverserial Attacks**, we look to generate an input image that is very similar to the target image but gets classsified *wrongly* by the CNN as the target class. We do Gradient Ascent here as well to iteratively modify the input. We use a slightly modified cost function to take into account the required similarity between the generated and target images.

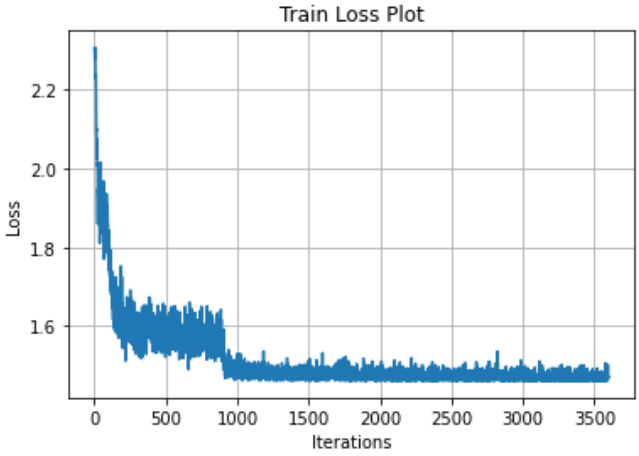
$$Cost \ = \ logit[target\_class] - \beta MSE(generated\_image, target\_image) \tag{2}$$

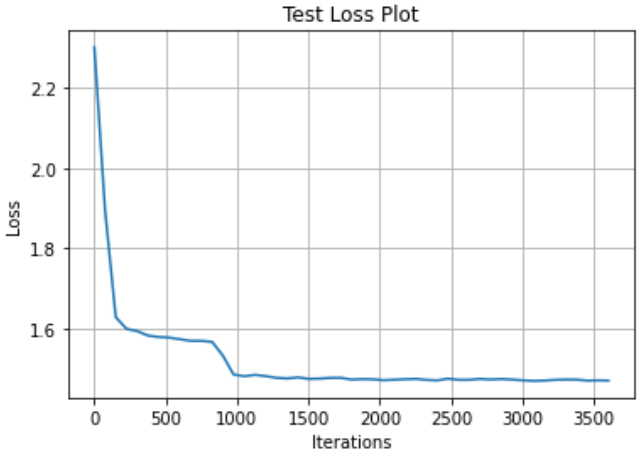
The results of this experiment are rather shocking because often, the resultant images obtained haven't changed much at all to the human eye but the CNN ends up misclassifying. This shows a very glaring weakness in CNNs where an adversary can just slightly modify the input to get the CNN to classify it into whatever class they want.

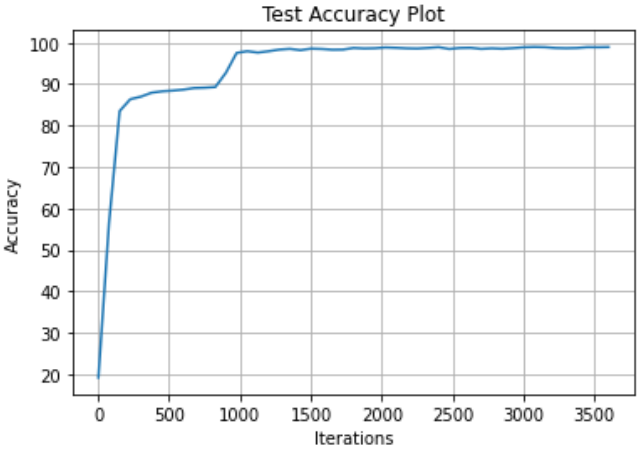
3 Experiments & Observations

3.1 MNIST classification using CNN

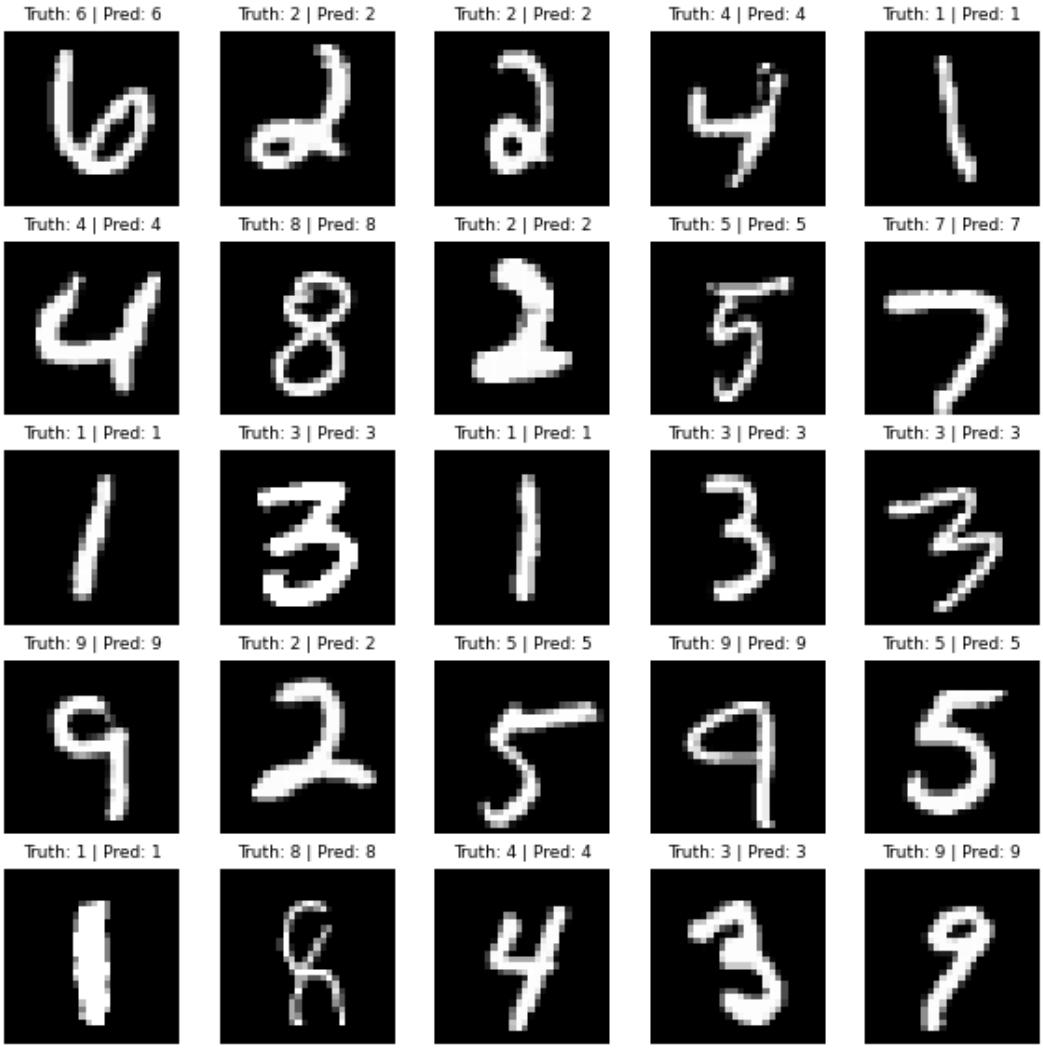
The baseline model already defined was trained for a total of 6 epochs. The following plots show the progression of train and test metrics through the training.







The **average test prediction accuracy** after training was **98.94%**. The following plot shows randomly selected test images with the caption showing their true class and predicted class(by the CNN).



Dimensions of input and output of each layer in the network:

- Conv1
  - Input: $1 \times 28 \times 28$
  - Output: $32 \times 28 \times 28$
- MaxPool1
  - Input: $32 \times 28 \times 28$
  - Output: $32 \times 14 \times 14$
- Conv2
  - Input: $32 \times 14 \times 14$
  - Output: $32 \times 14 \times 14$

- MaxPool2
  - Input: $32 \times 14 \times 14$
  - Output: $32 \times 7 \times 7$
- FC1
  - Input: $32 * 7 * 7 = 1568$
  - Output: 500
- FC2
  - Input: 500
  - Output: 10
- Softmax Activation
  - Input: 10
  - Output: 10

Number of learnable parameters:

- The MaxPool layers have 0 learnable parameters.
- The Conv1 layer has 32 biases(1 each for the 32 filters) and 3x3x32=288 weight parameters(3x3 for each filter).
- The Conv2 layer has 32 biases(1 each for the 32 filters) and (32x3x3)x32=9216 weight parameters(32x3x3 for each filter).
- The FC1 layer has 500 biases and 1568x500=784K weight parameters.
- The FC2 layer has 10 biases and 500x10=5K weight parameters.
- Softmax activation has no learnable parameters.

The **Total number of Learnable Parameters** in the network is **799,078**. The FC layers account for **789,510(98.80%)** and the convolutional layers account for **9,568(1.20%)**. We see that the learnable parameters are mostly found in the FC layers. This shows how convolutional layers have much fewer parameters compared to FC layers.

Number of neurons: The number of neurons can be easily found by looking at the output sizes of the each layer.

- The Conv1 layer contains 32x28x28=25,088 neurons.
- The Conv2 layer contains 32x14x14=6,272 neurons.
- The FC1 layer contains 500 neurons.
- The FC2 layer contains 10 neurons.

The **Total number of Neurons** is **31,870**. The convolutional layers account for **31,360(98.40%)** while the FC layers account for **510(1.60%)**. We see that with respect to neurons, the majority are in the convolutional layers with very few neurons in the FC layers. This trend is opposite to the trend of parameters.

So, FC layers account for the large majority of the parameters even though they have very few neurons and the convolutional layers account for huge number of neurons with very few parameters(*parameter sharing* and *local connectivity* lead to a drastically lower **Number of Parameters per Neuron** in convolutional layers compared to FC layers).

Batch Normalization:

While using batch normalization, the test accuracy is **99.18%** which is a bit higher compared to the original baseline model(**Test accuracy improves with batch normalization**). The original model took about 786 seconds to train while the model with batch normalization took about 1391 seconds(about 77% increase) to train(**Batch normalization affects the training time**). The test accuracy is a bit higher(about 0.24% higher) but batch normalization severely affects the training time and leads to a 77% increase. So, overall, batch normalization doesn't seem to be that useful here. Training takes much longer to give a very small increase in test accuracy.

3.2 Visualizing Convolutional Neural Network

3.2.1 Filters

- All the filters seem to be different from one another. Therefore, each of them is extracting something different from the image.
- Usually, in the first convolutional layer, we tend to get filters that look very similar to Gabor filters that help recognizing edges, orientation etc.
- The 2nd layer filters also seems very similar to the those of layer 1. There doesn't seem to be any recognizable pattern in layer 2 filters as well.
- The filters of deeper layers help recognizing more and more complex features(much more complex than just edges).
- There doesn't seem to be any recognizable pattern in the 3x3 filters otherwise. Perhaps, if we had filters of larger size, the patterns may be more visible and apparent.

3.2.2 Activations

- We see that the activation of layer 1 seems to extract very basic features from the image such as the edges.
- Also, the activations of layer 1 seem to even have varying background colors. Even the digit itself is represented in varying colors compared to the background. Perhaps this provides better adaptability to varying illumination in the input image.
- Some of the channels have produced crisp recreations of the actual digit(Ch 31) while there are also activations that have produced blurred versions of the input(Ch 19). Again, this is just some basic manipulations that layer 1 is applying on the input.

- Compared to the layer 1 activations, layer 2 activations seem much more abstract. It is already hard to interpret what this layer has extracted.
- So, overall, the activations produced by convolutional layers tend to get more and more complex and abstract as we go deeper into the network. As we go deeper, the convolutional layers put together features extracted by the previous layer to represent much more complex features in the input.
- Some layer 2 activations seem to still show the outline of digit 1 but the other activations have no such easy interpretation.
- Some layer 2 activations are even completely blacked out(fully zeros).

3.2.3 Occlusion Experiments

The learnings from occlusion experiments are definitely meaningful. Refer 2.4 for more explanation.

- From the examples shown, it is easy to see that even for the same number, with different images, the occlusion regions that maximally reduce class probability differ due to variation in the way the digits are written.
- Some images seem to be completely resistant to occlusion. If you consider 2, it is obvious that this resistance to occlusion is a very image specific thing and not dependent on the number written.
- For the 9, the top purple is due to occlusion leading to the digit becoming 7 while the bottom purple is due to formation of digit 0. Similarly, we may be able to interpret the heatmaps.

3.3 Adverserial Examples

3.3.1 Non-Targeted Attack

- **Yes**, the generated images predict the target classes with 100% probability(high confidence). In fact, the stopping condition used to generate these images was for the target class probability to reach 1. So, it is no surprise that this happens.
- **No**, the generated images do not look like numbers. We only trained the network to behave a particular way on inputs with handwritten digits on them. We never trained the network on how it should perform for inputs that don't look like the handwritten digits. Also, the CNN is just a mathematical model to compute probabilities given an input. So, irrespective of what the input contains, CNN will output some probabilities always. The input space is $\mathbb{R}^{784}$(input size is 28x28) which is a very large space and we trained the network only for a very small subset of inputs. It is no wonder that we managed to find an input which seems random that produces high target class logit. So, without putting constraints on how the input should look, it is completely understandable that we get an input that doesn't look like a hand-written digit at all while maximizing logit. This also shows a vulnerability of the CNN perhaps. The fact that the network is able to confidently classify this noisy input into a class is a bit scary.

The **cost**, irrespective of target, **increases** with iterations as we are performing a Gradient Ascent to maximize the cost.

3.3.2 Targeted Attack

Yes, the generated images look like numbers of the target image's class now.

3.3.3 Adding Noise

4 Code

The implementation of all the experiments used in this report can be found in the following colab notebook: Link

5 Conclusion

We have thus built CNN models that perform very well on the MNIST dataset and further analyzed them using various methods.

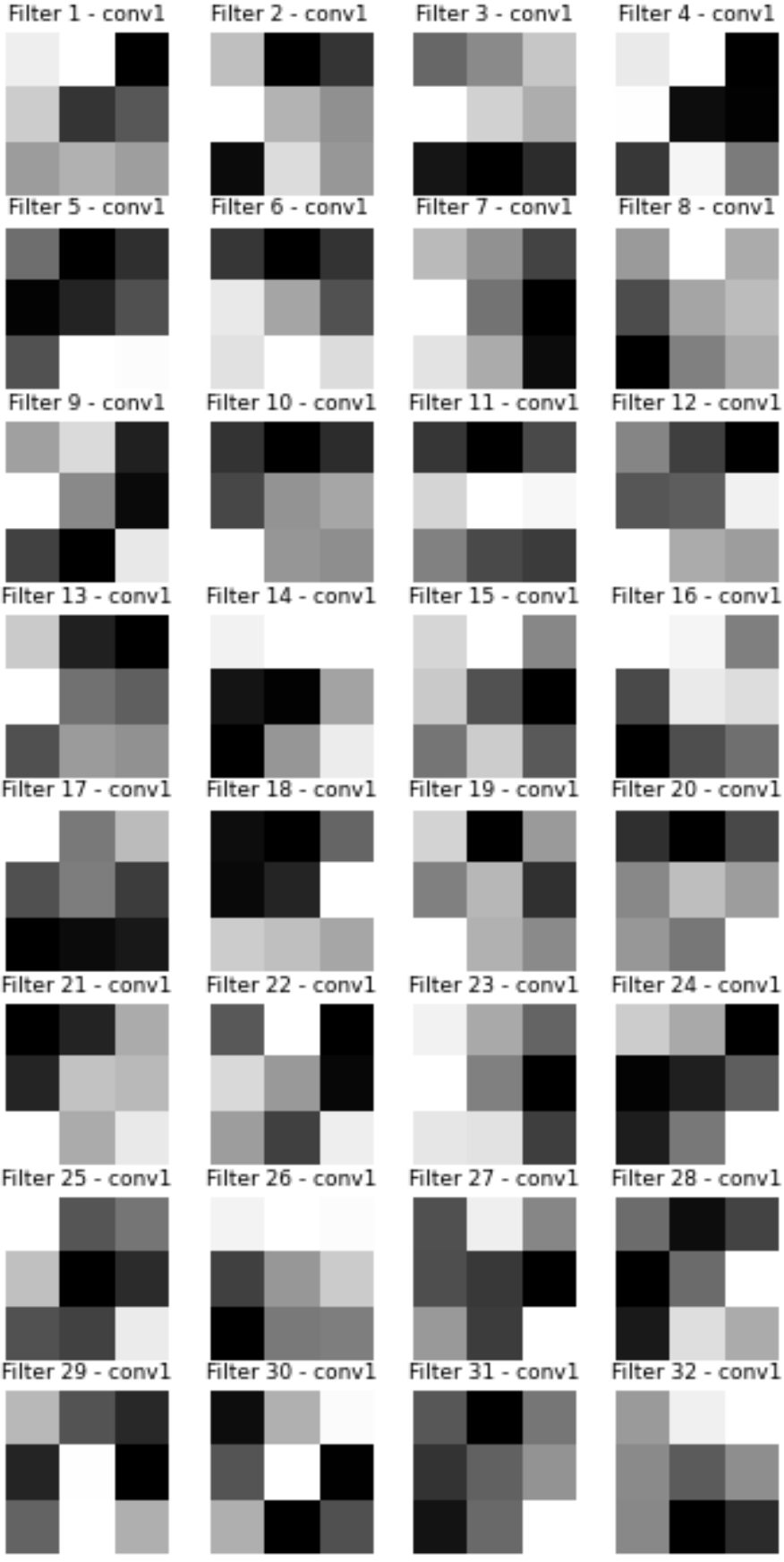


Figure 2: Filters in Conv1

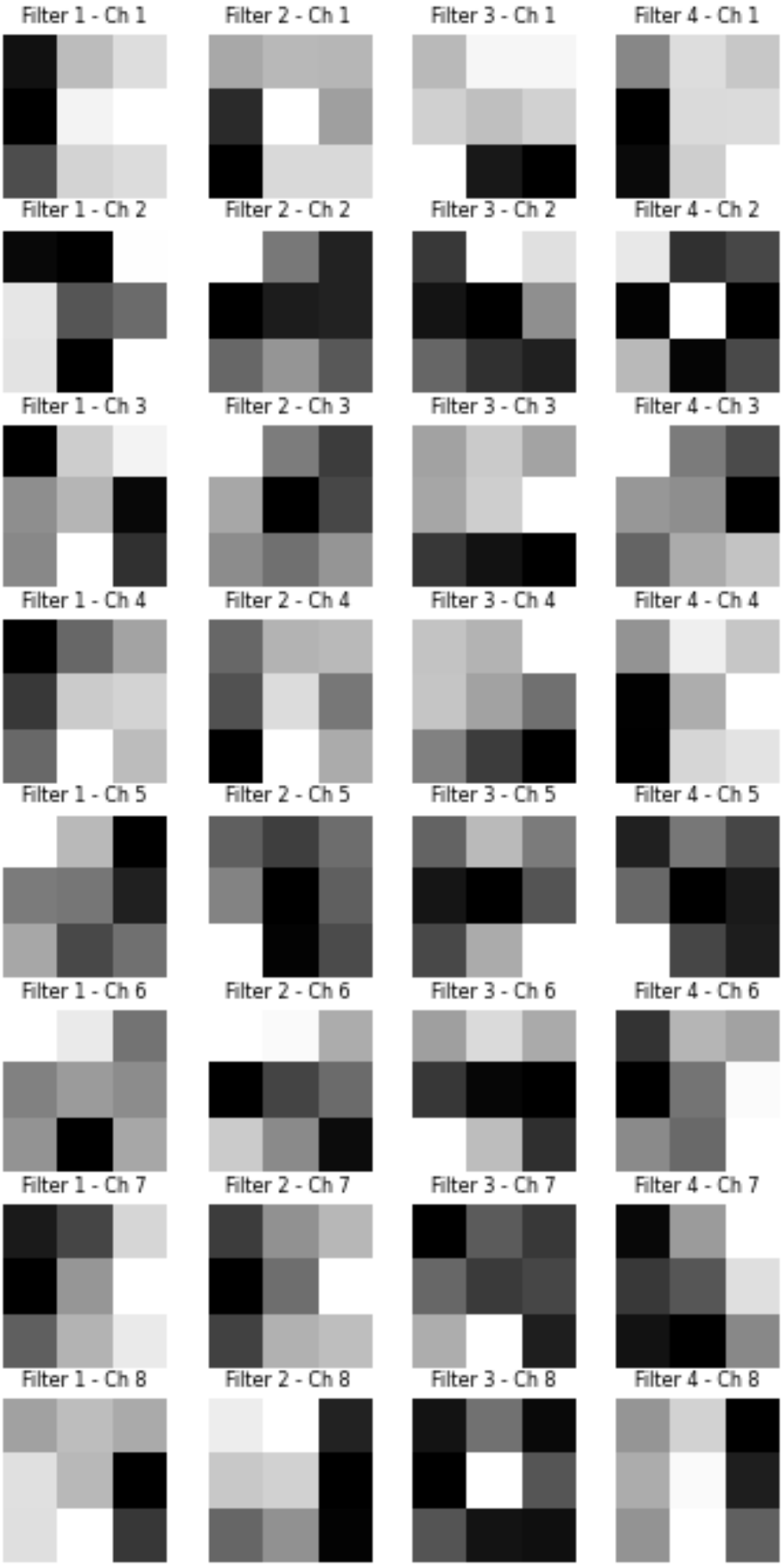


Figure 3: 8 channels each of 4 filters in Conv2

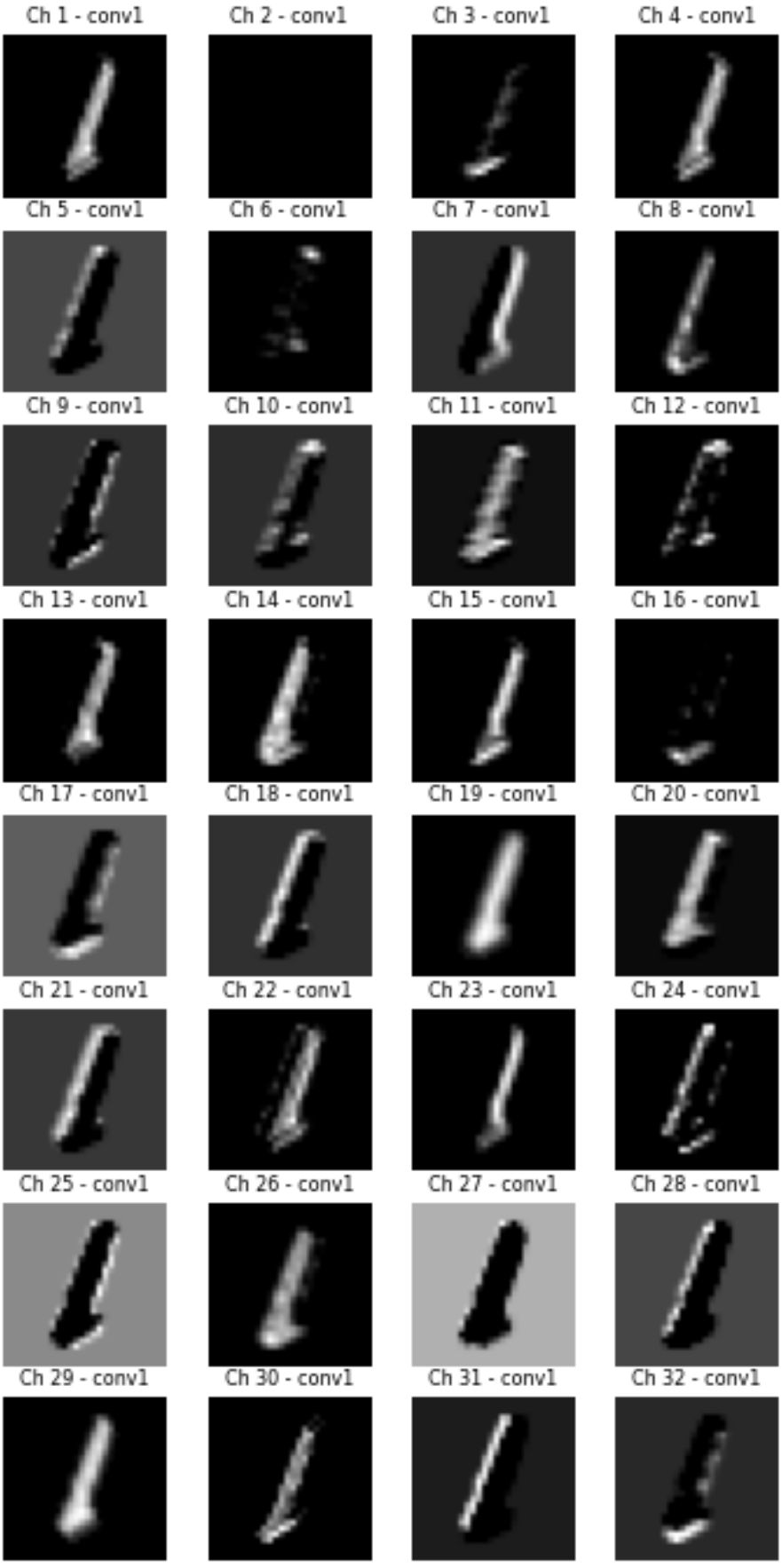


Figure 4: Activations of Conv1

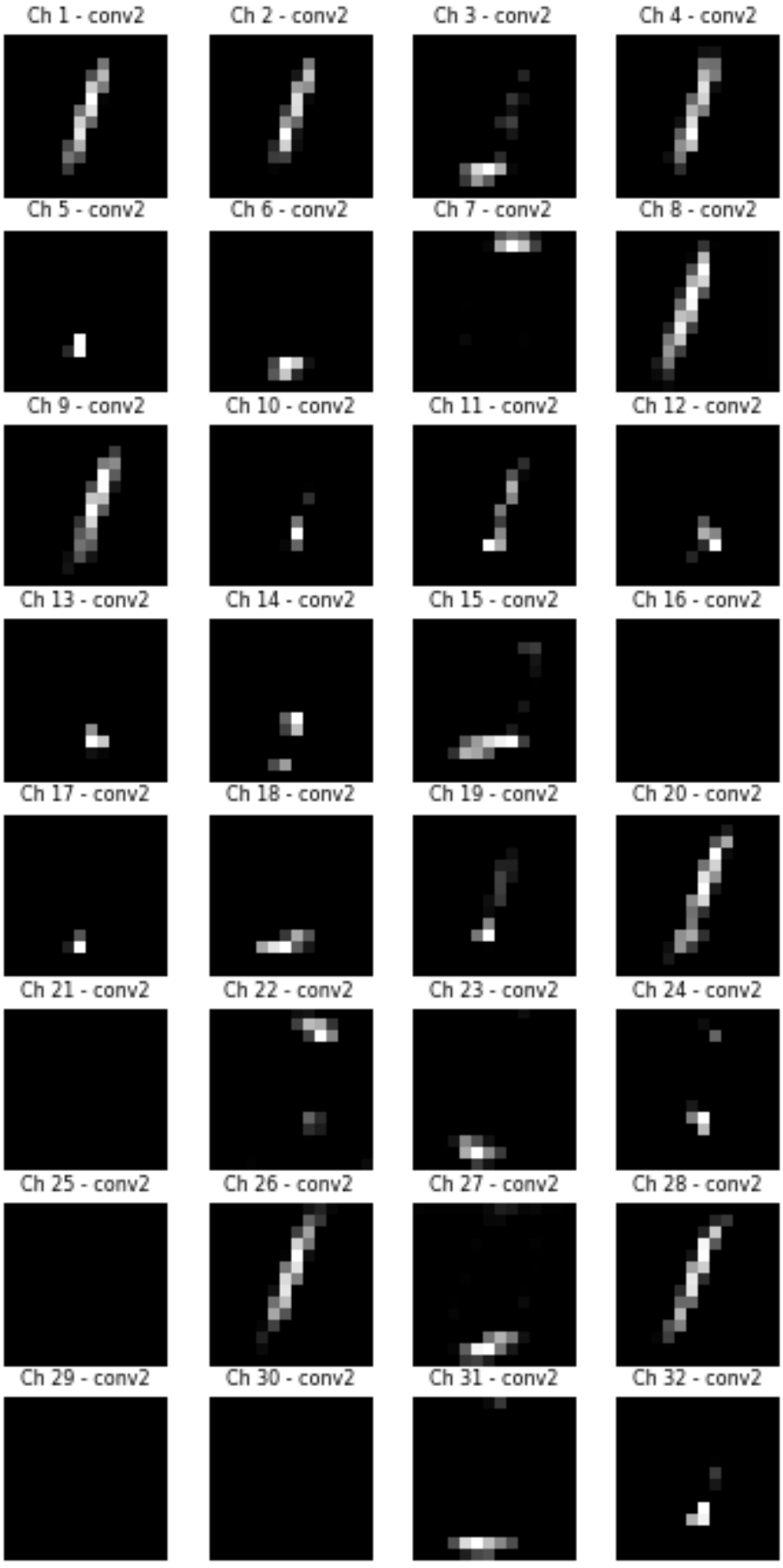


Figure 5: Activations of Conv2

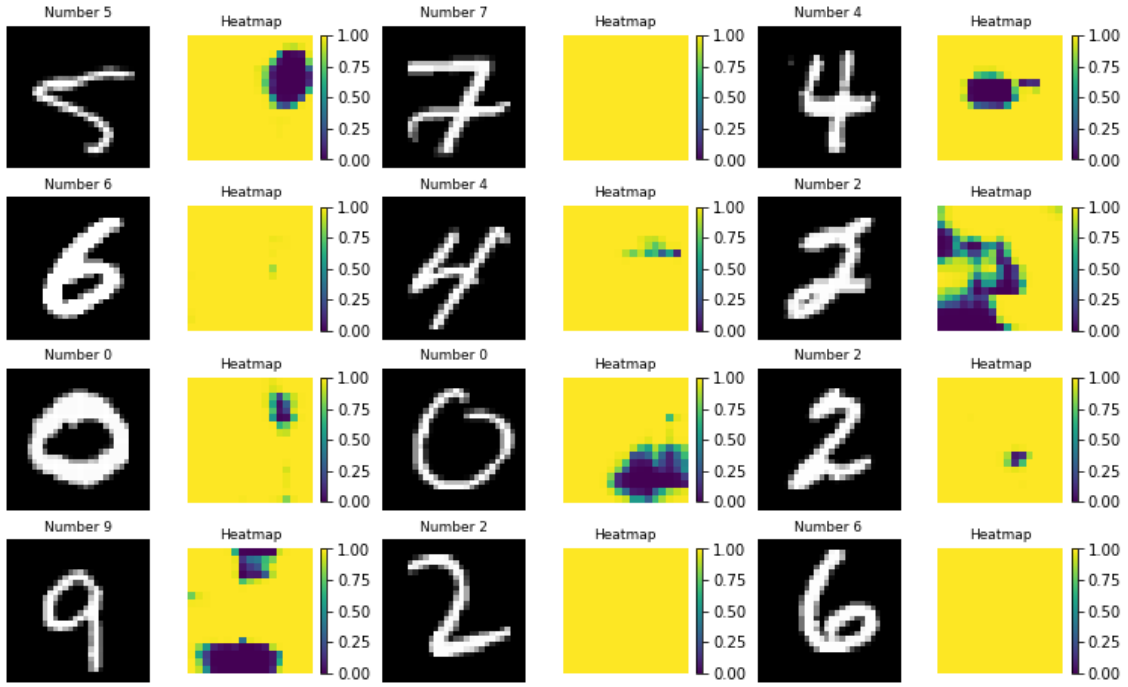


Figure 6: Occlusion Heatmaps

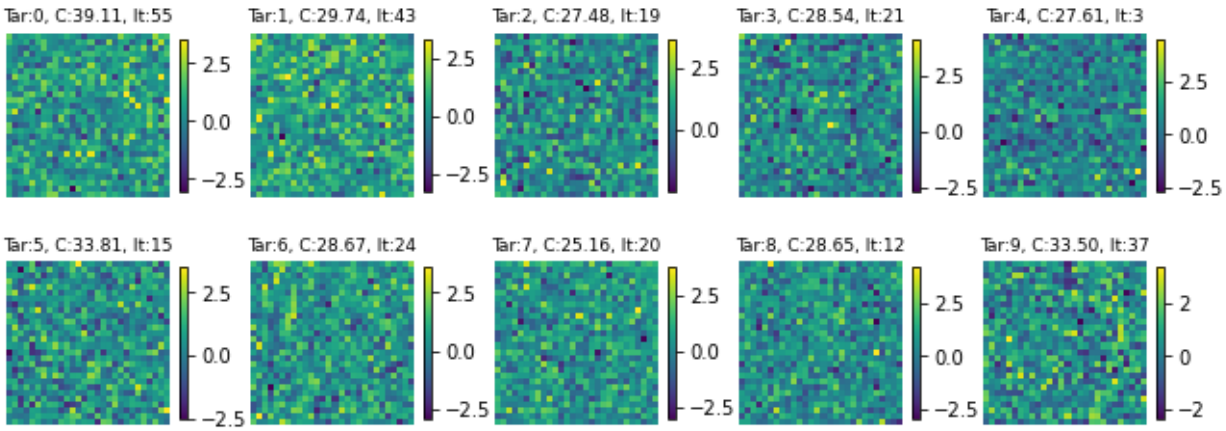


Figure 7: Non-Targeted Attack

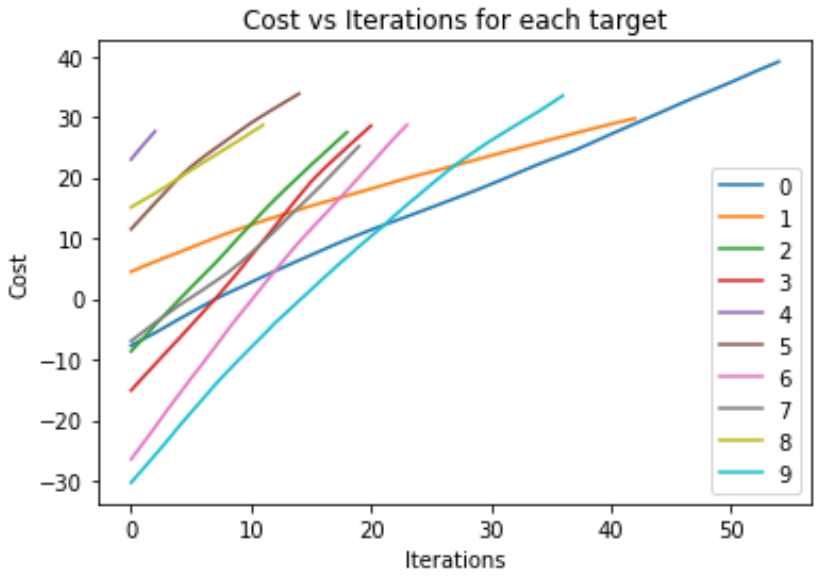


Figure 8: Cost vs Iterations (Non-Targeted Attack)

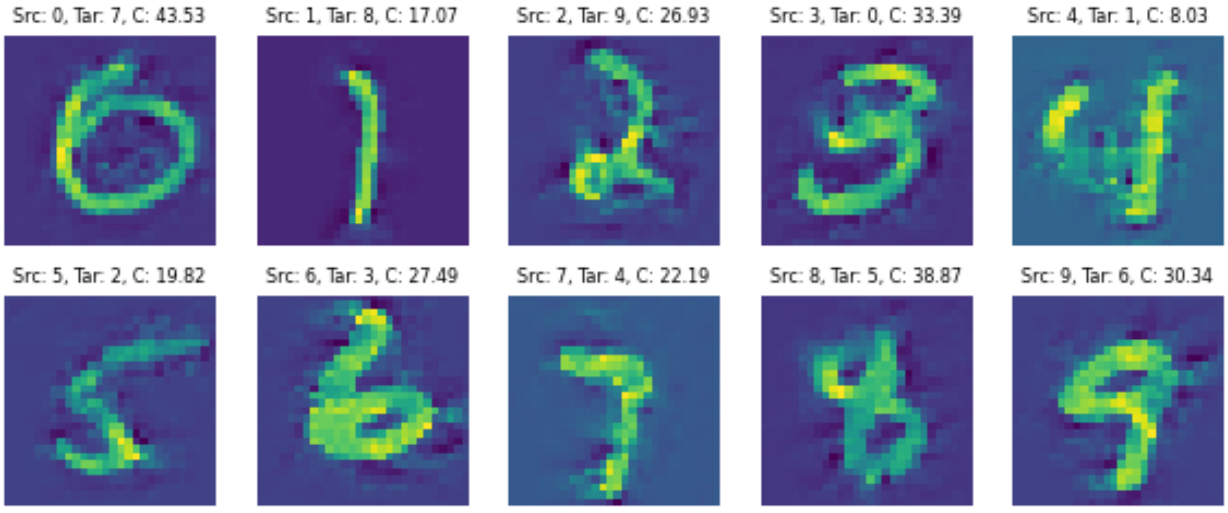


Figure 9: Targeted Attack

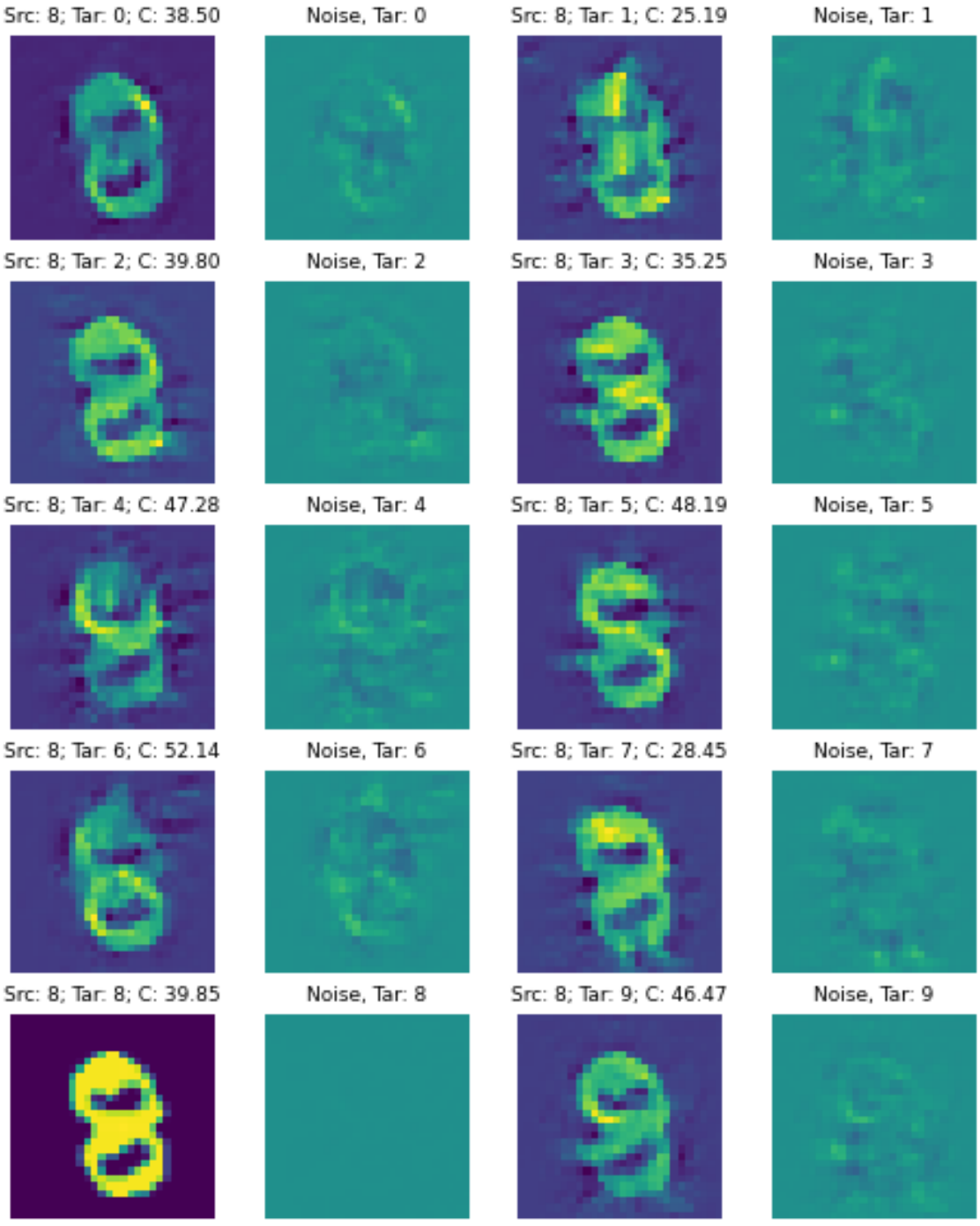


Figure 10: Adding Noise

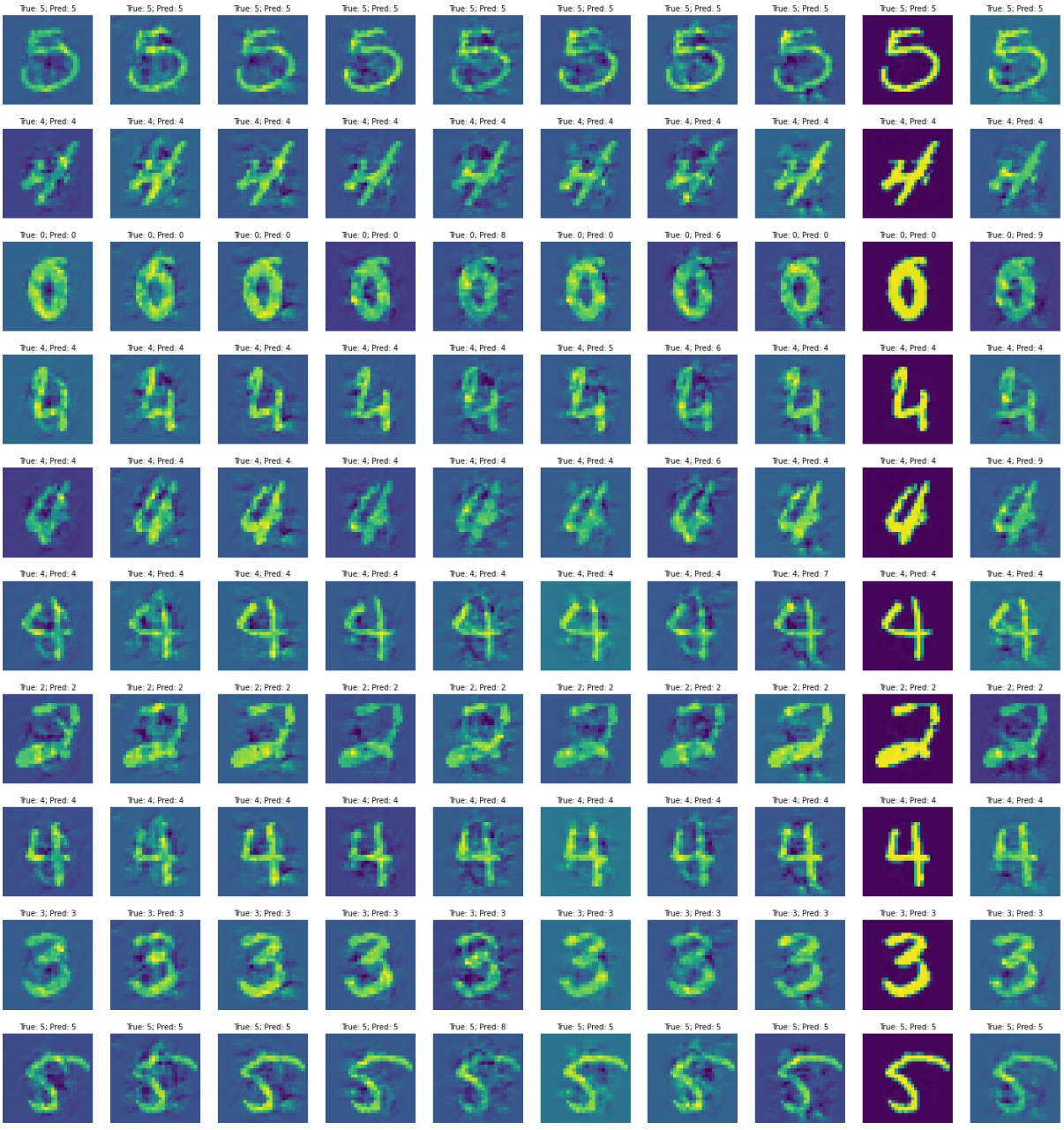


Figure 11: Adding Noise - 10 test examples