Curiosity: Models & Applications - Project

Submitted by:

Lior Sukman 319124244
Olga Soldatenko 342480308
Yana Dervin 310684386
Yuval Goldshtein 311552368

Part 1

Defining the problem: Choosing the Learning Task

The goal of this project is to learn what noise can fail classifiers, specifically in the form of convolutional neural networks (CNN), and cause them to mispredict. This field of study, also referred to as the attacking of neural networks, focuses on the creation of adversarial examples (AE) by the addition of carefully constructed “noise” to the input also called perturbations. These perturbations sometimes can be imperceptible to humans. Specifically we will use the MNIST dataset of handwritten digits for this task as it is a common dataset and was used in previous studies.

1. ### Definition of the input-output

   As an input we use a set of pictures of handwritten digits of the size 28x28 from the MNIST dataset.
   The output of the models should be the perturbation or noise we want to add to the sample (input). The output (or action) is defined by the location in which a specific alteration occurs and its direction (i.e. increases or decreases the pixel’s value - color).

2. ### Definition of the Problem’s Space

   Each sample of the input is a matrix of the size 28x28, where every element represents the color (or more precisely brightness) of the pixel; these values are continuous between the range 0 and 1.
   As mentioned earlier, the output is an action (or combination of actions). Each action corresponds to a pixel (i.e. in the 28 by 28 range) and direction by ∓ 1 (i.e. either increase or decrease pixel’s brightness).

   The alterations are done by either adding or subtracting ε from the pixel’s value (maintaining the value in the 0-1 range). It is important to limit the value of ε as otherwise by blackaning the input it is possible to get wrong predictions, yet humans could not recognize the altered images. Choosing the value of ε is not trivial, hence we implemented[1] the fast gradient sign method (FGSM; see Goodfellow, Shlens & Szegedy, 2015 for further explanation[2]). Using this method we checked different ε values and compared the accuracy rate of a CNN model which we created[3] after applying the method using these values, the results are described in **figure 1.1**. See **figure 1.2** for corresponding AE. Based on these results we decided to use ε=0.25, this value yielded a 16.2% accuracy rate (or 83.8% error rate). Note that the CNN’s accuracy rate is 98.48% (on the test set) and exhibited by using ε=0.

---

[1] This implementation is based on the code you can find on https://pytorch.org/tutorials/beginner/fgsm_tutorial.html

[2] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." *arXiv preprint arXiv:1412.6572* (2014).

[3] Our CNN model is based on the code you can find on https://github.com/pytorch/examples/tree/master/mnist

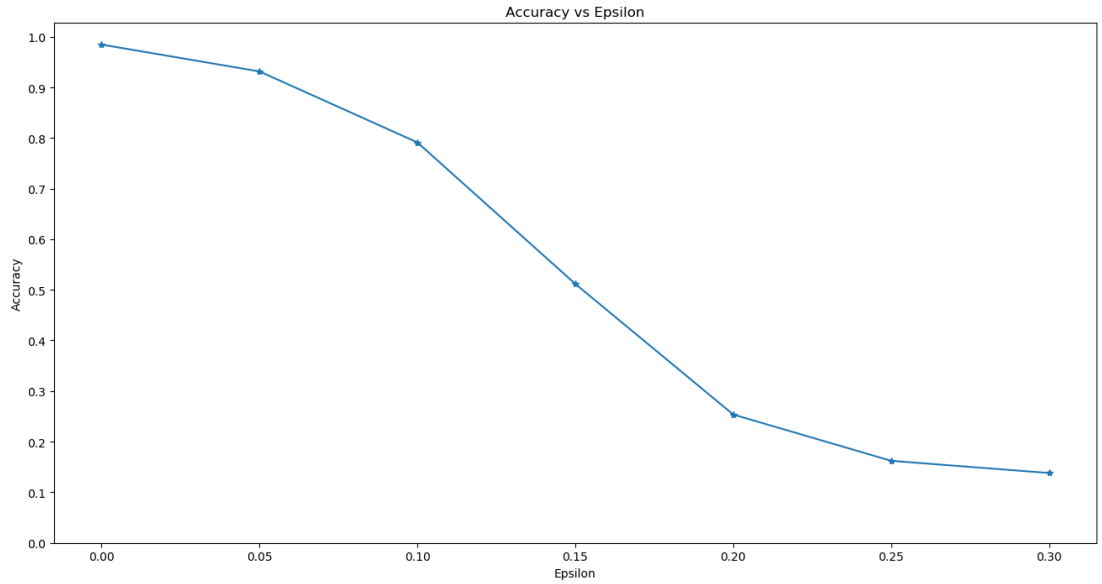

Figure 1.1: Model accuracy on the test set versus ε value using the FGSM.

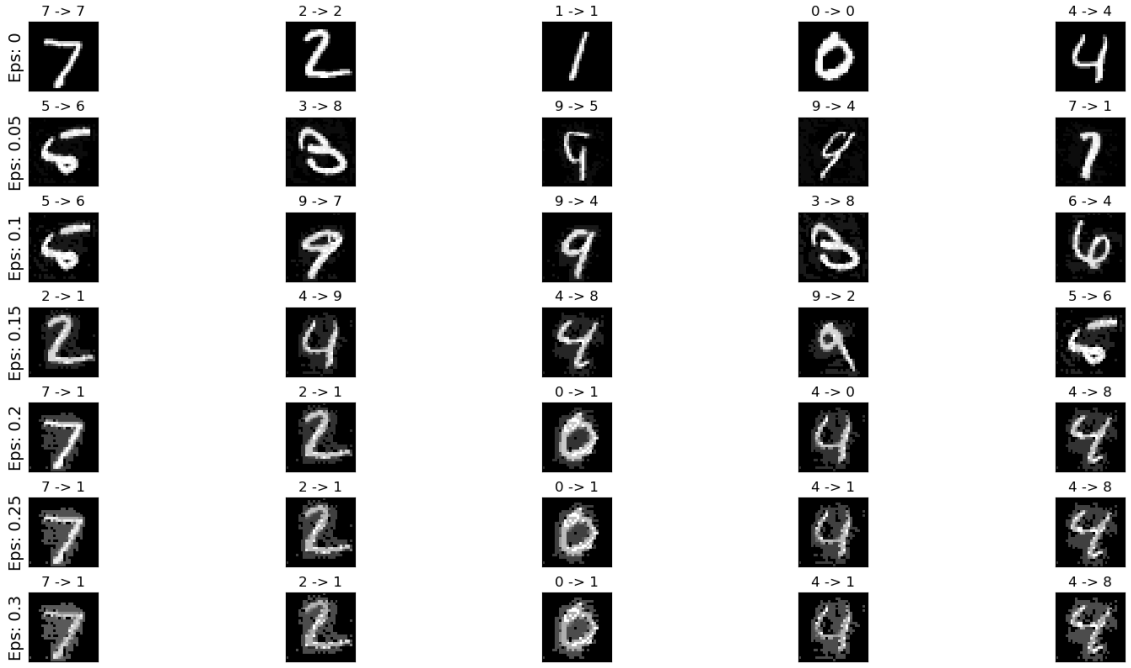

Figure 1.2: Examples generated by the FGSM using different ε values. Note that for all but ε=0, the examples shown were classified correctly before the perturbation but misclassified after.

3. Obtaining the data

We are using the MNIST dataset of handwritten digits (achievable through the torchvision package), in addition to that, we use the aforementioned CNN to make predictions for the pictures in the dataset after applying perturbations.

Formalization of the Problem Using Bayesian Inference

Formulation of the problem

Parameters: The model receives triplets of s, a and x where s is the original class of the image (note that here we use only the class rather than the complete 28x28 representation in order to make the space manageable, this will change in later parts).
Priors: The priors were initialized uniformly, although it is worth mentioning that the way the noise was added (see next section) can only rarely change the prediction of the model and hence it is very unlikely to be represented by a uniform distribution (it is not trivial to find another distribution which represents the reality better). Given our immense data (see programming section), we believe that the initialization of the priors has a relatively small effect.

Programming the Problem[4]

1. Learning Programming

To generate the data, for each sample, a corresponding matrix N of size 28 by 28 was generated with values of $\mp\varepsilon$ or 0, all in equal probability. The matrix was then added to the original sample and the new sample was re-predicted using our CNN. Multiple actions were assigned to each sample according to the following rule: If N(i,j)=ε then action i+28*j+1 is attributed to the sample and if N(i,j)=-ε the action i+28*j+2 is attributed to the sample, this is an enumeration of the actions required for the Bayesian model. In total there are 28*28*2=1,568 different possible actions. Each sample s is then saved multiple times to a csv according to the actions attributed to it (a1, a2, a3, …) and the predicted class x. It is reasonable to use this formalization as although we do not assign a single action to each sample in practice, the other actions can be regarded as noise. It was predictable that this method of generation will only rarely change the CNN's predictions, and that generally a single action (i.e. the limited change of only one pixel) will have a small if any effect.
We used 5,000 of the 10,000 samples in the MNIST test set (partition of this set will be more relevant in later parts, here it was used to reduce the complexity and check each sample with more actions). For each sample this was repeated 6 times, the expected number of actions attributed to each sample by a single matrix N is 28*28*⅔=522.66, this results in an expected number of 522.66*6*5000=15,680,000 (s, a, x) triplets for the Bayesian model.
To reduce the complexity in this problem we have examined another possibility in which the same value will be added to blocks of pixels instead of to single pixels. Using this method with blocks of 2 by 2[5], the total number of actions decreases to 392 (75% less than the previous method) and the expected number of actions per sample is 14*14*⅔=130.66 (similarly to previous calculation, this results in 3,920,000 triplets).

[4] The entire code that was used is submitted as well.
[5] Other block sizes were examined as well, yet they performed poorly.

Using this approach, at ε=0.25 the CNN achieves 31.83% accuracy rate (see **figures 1.3-1.4** for complete results and examples).

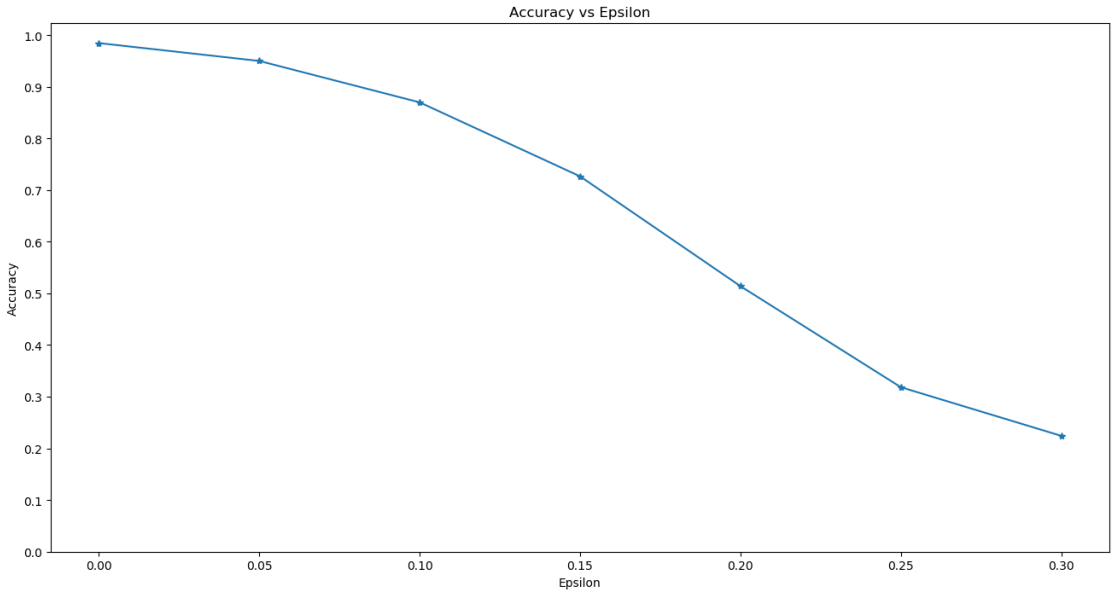

Figure 1.3: Model accuracy on the test set versus ε value using the FGSM with 2 by 2 blocks.

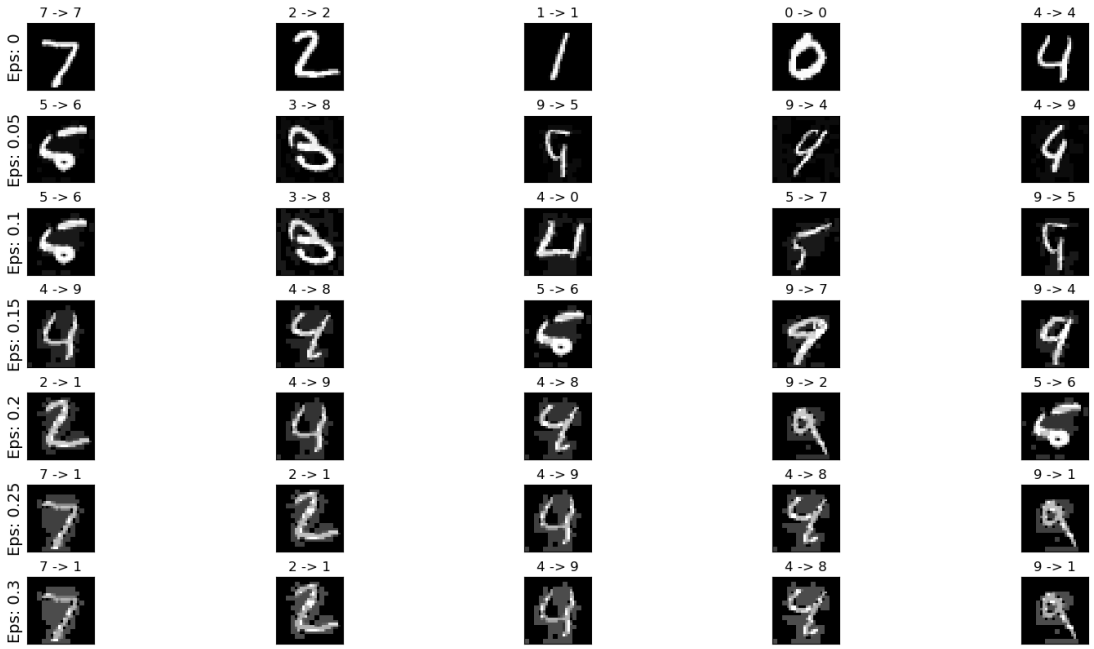

Figure 1.4: Examples generated by the FGSM using different ε values with 2 by 2 blocks. Note that for all but ε=0, the examples shown were classified correctly before the perturbation but misclassified after.

Using the generated data we learned a Bayesian model. It is important to mention that the model needs the probability of witnessing label x while the ground truth is t for the updates. We have chosen to take advantage of the information we have from the performance of the CNN and set this probability to be 0.985 when x=t and 0.015/9 otherwise (9 is the number of remaining classes). This surely is not the only approach that should be considered, yet for our purposes it was sufficient (see Papernot, McDaniel, Goodfellow, Jha, Celik & Swami, 2017 for some exploration of mistakes using different adversarial generation methods[6]). The information gain graph representing the updating process is shown in **figure 1.5**. In all cases the highest probability of a specific x given s and a was attributed to the x corresponding to s (i.e. predicting that the label will not change). We will note that the accuracy rate of the perturbed images was 97.2%-97.9% (depending on the block size) while on the original pictures it reached 98.3%. As this difference is minor, we worked using the entire data and not only the pictures that changed their predicted label. An alternative in which only samples that were classified correctly before manipulation but wrongly after it, is discussed in the **appendix**. Using the data generated using blocks, yielded similar results in this section, and hence its results are not specifically mentioned.

[6] Papernot, Nicolas, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. "Practical black-box attacks against machine learning." In Proceedings of the 2017 ACM on Asia conference on computer and communications security, pp. 506-519. 2017.

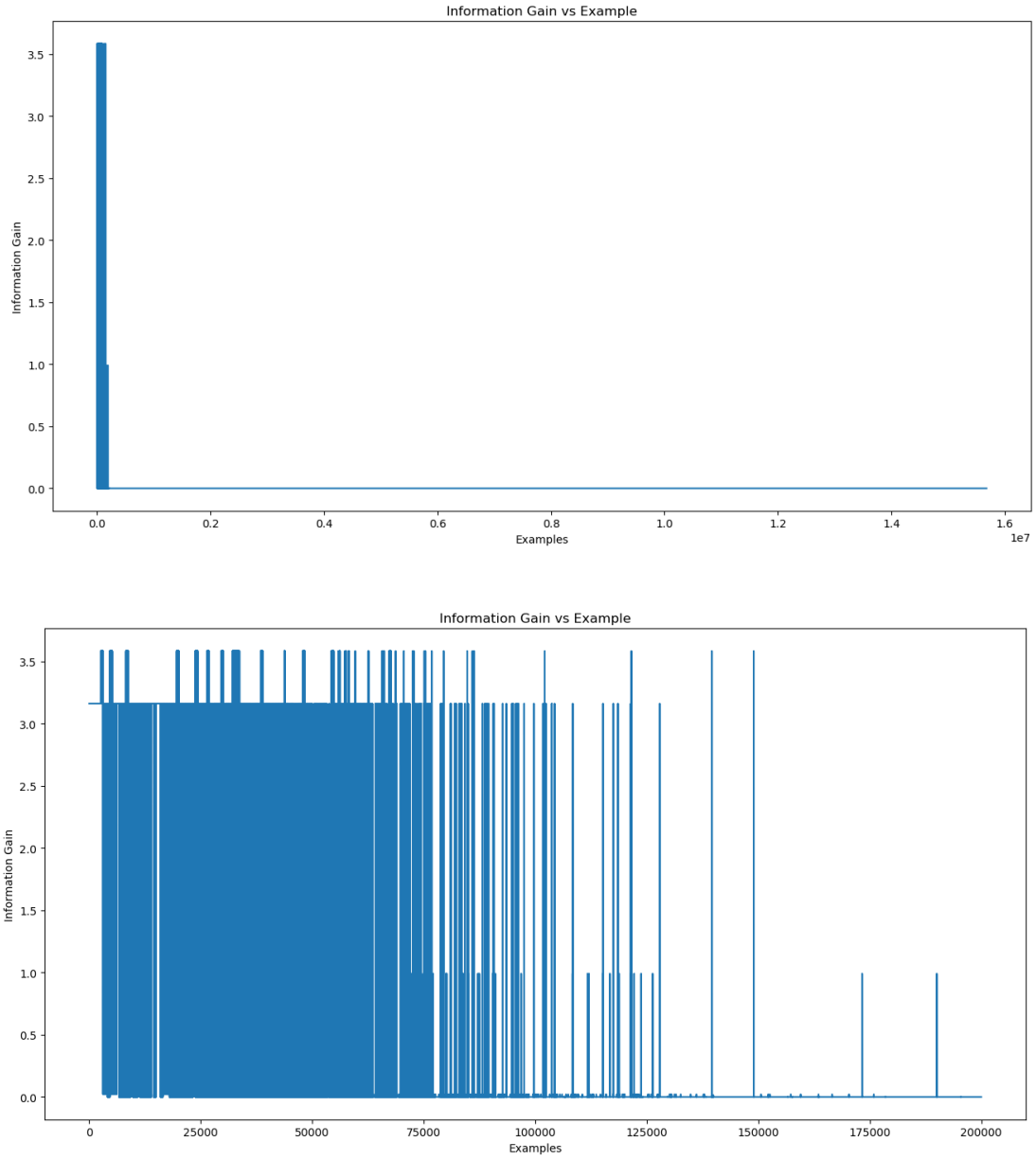

Figure 1.5: Information gain by example in the Bayesian updates procedure. Top shows full training while bottom shows only the first 200,000 examples.

2. Bonus: Maximal Information Data

In an attempt to find data that gives maximal information we have run the Bayesian updates procedure for a number of random samples of our data. After that, we have put the data points with highest average information gain on the top of the list to be processed first. See **figure 1.6** for the results.

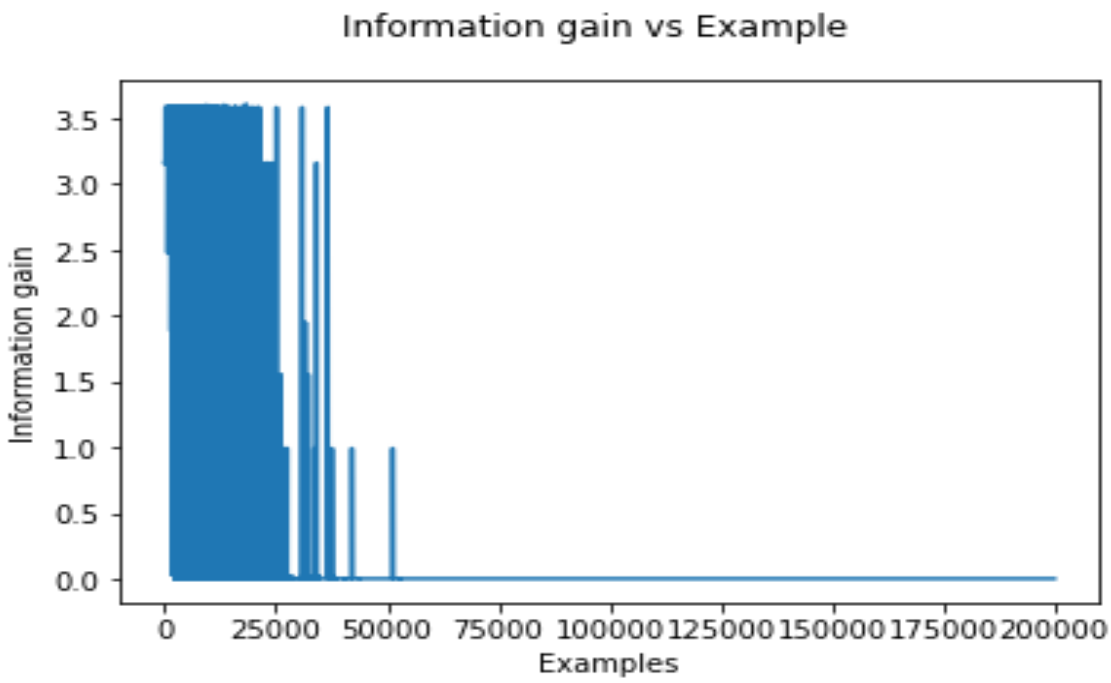

Figure 1.6: Information gain by example in the Bayesian updates procedure over the ordered data. The plot shows only the first 200,000 examples.

It is clear from the plot that after about 50,000 examples there is no information gain whereas for the original set the information gain continues to spike all the way till 150,000-175,000 examples.
This means that we have managed to find data with high information. Although it is not clear what characterizes this data, except for the fact that all of the data points were misclassified ($s \neq x$).
This matches the observation during the first run of the Bayesian updates: on the original dataset we have noticed that non fitting examples ($s \neq x$) on average had higher information gain than the fitting ones.

Part 2

Reminder

The goal of this project is to learn what noise can fail classifiers, specifically in the form of convolutional neural networks (CNN), and cause them to mispredict. We use the MNIST dataset of handwritten digits, and aim at finding noise to be added to a specific image which will cause some pre-trained network on this problem to mispredict. Our learner's goal is to create this noise and to minimize as much as possible the accuracy rate of the pre-trained model[7]. We have limited the change in each pixel of the input image to be in the range of (-ε, ε), more specifically, we constrained the change to be either -ε or ε (or none) with ε=0.25 (the choice of this ε value was done in the previous part). Limiting the change is crucial, as otherwise the model will learn to blacken the input, creating an image with no informative value. Although we did not show that the informative value holds after the addition of noise in this range (as a complete manual scanning of the perturbed images is not possible, nor methodologically valid), we stand on the shoulders of previous studies in this field claiming that these perturbations still allow a human to classify the images. Moreover, these studies show that training using the perturbed images as well reaches high accuracy rates and reduces the vulnerability of the network to similar attacks.

Formalization of the Problem Using a Neural Network

1. Architecture of the Neural Network

Input - An image of a handwritten digit in the form of a matrix of 28 by 28[8].

Output - The input image with the addition of noise (perturbed image)[9].
Datasets - As the MNIST dataset is commonly divided to train set of 60,000 examples and test set of 10,000 examples, in order to create the dev set we randomly partitioned the MNIST train set to a new train set of 50,000 examples and dev set of 10,000 examples.

The structure of the network was based on the structure of the CNN we used in the previous part. In both cases, the network started with a convolutional layer transforming the input image into a 16 channel 26 by 26 image (using a kernel of 3 by 3), followed by a ReLU activation function and max-pooling (with kernel of size 2 with stride equals 2 as

[7] See the loss definition in the "Learning Algorithm" subsection for further discussion regarding this metric.
[8] As defined in part 1 of the work, the values of the cells are in the contentious range between 0 and 1.
[9] The noise added to each cell was limited between ε and -ε. See next section for explanation regarding the transformation to a supervised setting.

well). After this part, a dropout layer with dropout probability of 0.25 was applied and after flattening the current network's state the process continued through fully connected layers (FCLs). The FCLs differed between the original CNN aiming at classifying the images and this network aiming at creating a perturbed image.
The original CNN that was used for the classification had two FCLs, The first, taking the current network's state of size 2704 and outputs a vector of size 128, then a ReLU activation function followed by a dropout layer (dropout probability of 0.5) are applied. The second FCLs (which is the output layer) input is this vector and the output is of size 10 on which log-softmax function is applied.
The second network (designed in this part) also has two FCLs, the first receives the current network's state and outputs a vector of size 1024. The second FCL (the output layer) outputs a vector of size 784 (equivalent to a 28 by 28 image). The process between and after these layers is similar, except that instead of using log-softmax, tanh is used. These values are multiplied by ε, then added to the input image and returned[10]. The latter network is presented in **figure 2.1**.

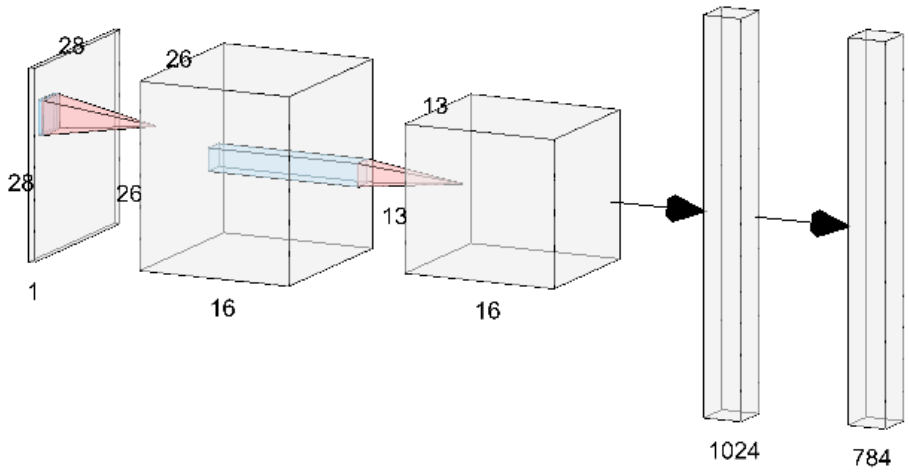

Figure 2.1: Architecture of the network creating the noise. Activation functions and dropout layers are not present in the illustration. Note that after the convolutional layer, the max-pooling process is illustrated.

2. Learning Algorithm

For the convenience of notation we will refer to the network generating the perturbed images as PGEN_NN and the classifying CNN simply as CNN.

[10] Clipping of the returned image to the 0-1 range is also applied.
Please see the next section for the process done on this image in order to transform the problem to a supervised setting and calculate a metric for success.

As mentioned in the previous part, the CNN had been pre-trained and evaluated, this network's accuracy rate on the test set reached 98.48%. See **appendix for part 2 A** for an illustration of the learning process of this network.

In order to transform a problem into a supervised setting the perturbed image that has been produced by the PGEN_NN is fed into the pre-trained frozen CNN for classification. Then, the loss is calculated based on the predicted labels according to the CNN and the true labels. As the weights of the CNN are frozen, the optimizer updates only the weights of the first network (i.e. the PGEN_NN). This process is also presented in **figure 2.2**.

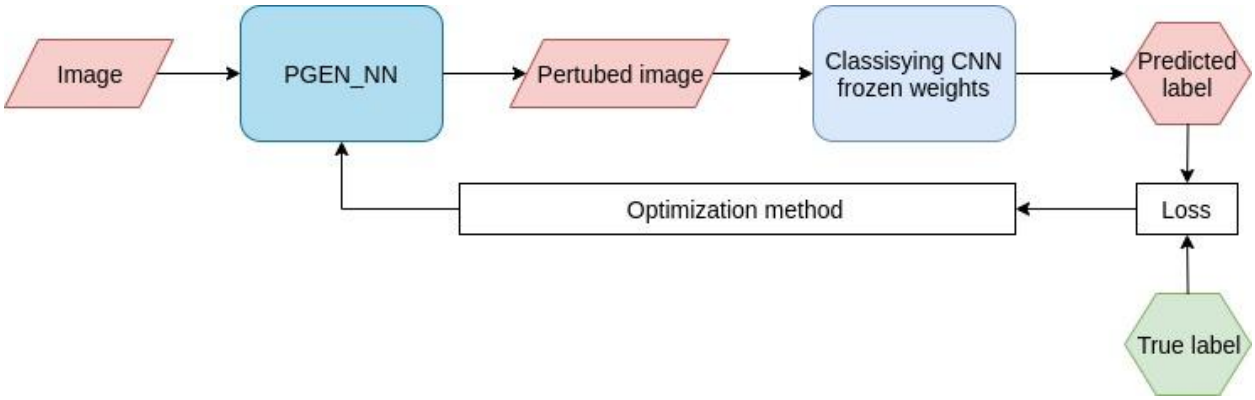

Figure 2.2: The scheme of the learning algorithm.

The loss function was chosen in order to "punish" high probabilities of the true label. We have chosen to use the inverse negative log-likelihood (NLL) loss. This loss function gives higher values the higher the CNN's value for the true label in its output layer on the perturbed image (compared with the other values in that layer). Although other options are possible such as using exponent or -NLL (can be called positive log likelihood), this loss yielded relatively good results.

The optimizer used for both networks was Adadelta. This optimizer was chosen as it generally shows good performance compared with other optimizers and it also eliminates the need for manual selection of the learning rate (Zeiler 2012[11]). The model was trained for 120 epochs.

Two separate functions were used for training and prediction: the forward function was used for training and the generate function was used for the prediction. The difference between these functions is in the functions applied on the output of the second FCL - forward uses tanh, whereas generate uses sign. The reason for this is that we wished to create a model fitting our discrete action space described in the previous part (i.e. each cell can change its value only by $\mp\varepsilon$) which will later be useful for the comparison of different models on the problem. The sign function was not used for the training as sign is not a differentiable function, harming the learning process. Note that there are different

[11] Zeiler, Matthew D. "Adadelta: an adaptive learning rate method." *arXiv preprint arXiv:1212.5701* (2012).

options for the generate function, such as changing only cells in which the absolute value of the noise in the corresponding cell exceeds a certain value, this is discussed in **appendix for part 2 C** as part of the discussion regarding regularization.

Programming of the problem and solution

1. Programming the Neural Network

Please refer to the code files submitted with this report.

2. Running the Algorithm

The trained model reached an average loss of 0.0756 and accuracy (of the CNN on the perturbed images) of 976/10,000 which is **9.76%** (as we aim at "attacking" the CNN, the lower the accuracy, the lower the loss) on the test set. The process of learning is described in **figure 2.3**. This accuracy rate is lower than the chance-level accuracy, thus we can argue that the model lost all its predictive power. Interestingly both the train and dev sets results are similar, and we see improvement (although minor all along the process). Another interesting observation is that at approximately the 40th epoch, a major decrease in the loss occurred which reduced the accuracy by roughly 8%.

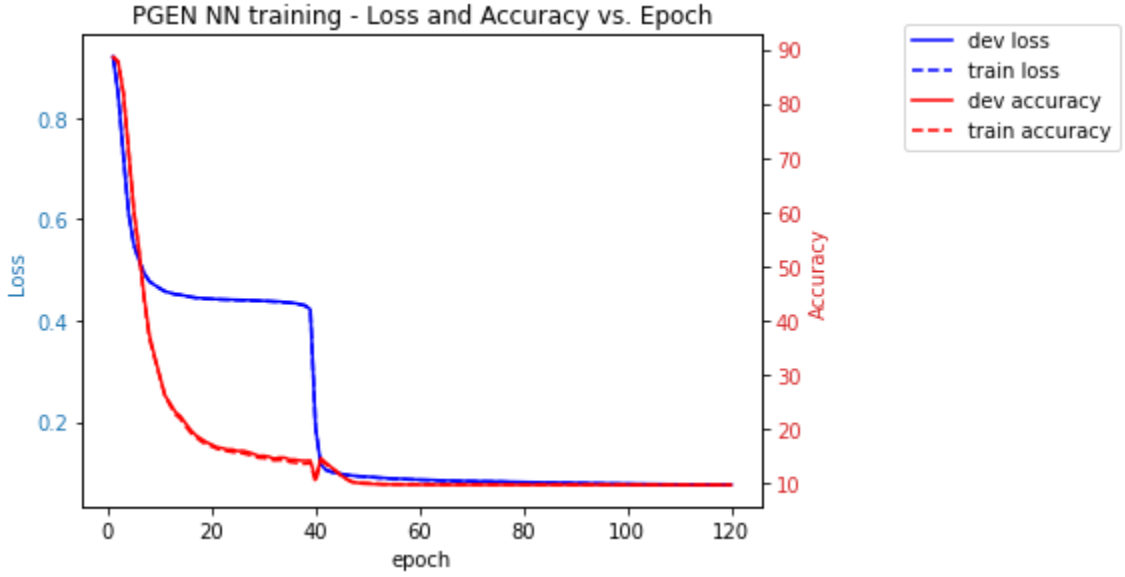

Figure 2.3: Learning process of the PGEN_NN for 120 epochs on the train and dev sets. Accuracy is presented as the percentage of correctly labeled perturbed images by the CNN.

For comparison, the FGSM described in the previous part reached a success rate of the CNN on the perturbed images of 16.2% - higher than that was reached by the PGEN_NN model. For more discussion regarding the performance of the network please refer to **appendix for part 2 B**.

Network Parameters Analysis

Changing the Network's Architecture[12]

For this section seven new models were evaluated.
In the first model another convolutional layer was added directly after the application of ReLU on the first convolution, receiving 16 channels and outputting 32 channels using a 3 by 3 kernel. Hence, the input of the first FCL was of size 4608. This model will be referred to as PLS_CONV (stands for plus convolution).
In the second new model, one FCL was removed and so the convolutional layers were directly connected to the output layer which now receives 2704 values. Note that for this model the second dropout layer was removed as well. This model will be referred to as MNS_LIN (stands for minus linear).

In addition to these models in which the addition or subtraction of layers was evaluated, we wanted to check how changing the sizes of the existing layers would affect the model's performance. For this, the number of output channels from the convolutional layer and the size of the hidden linear layer (i.e. the layer before the output layer) were manipulated; these sizes will be denoted as c and h respectively. The c values that were checked were 8, 16 and 32, and the h values that were checked were 1024 and 2048.

Analyzing the Learning Sensitivity to the Architecture[13]

The results of running the PLS_CONV, MNS_LIN and PGEN_NN (will also be referred to as the original model/network) networks are presented in **figure 2.4**[14].

[12] In addition to what is presented in the following section, please refer to **appendix C** for comparison of models with regard to regularization.
[13] Note that each network was only trained once for computational reasons. Hence, while examining the results we show in this and the next section, one should be aware of the effect of chance.
[14] For simplicity the numerical results are not presented, the submitted notebook contains all of the unmentioned results.

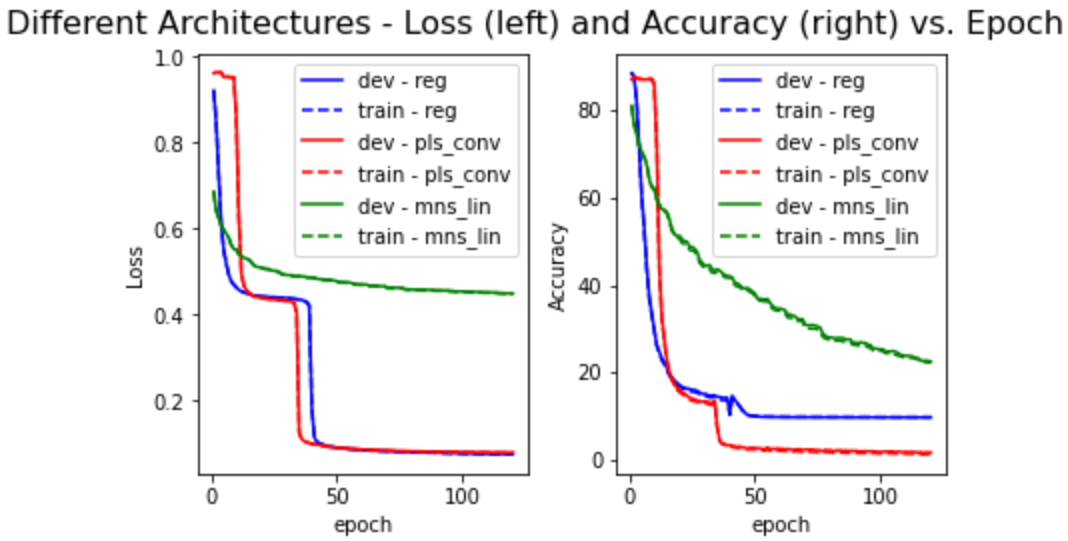

Figure 2.4: Learning process of the networks for 120 epochs on the train and dev sets. Accuracy is presented as the percentage of correctly labeled perturbed images by the CNN. reg represents the original network (PGEN_NN), PLS_CONV and MNS_LIN represents the new networks.

Following these results we checked the performance of the PLS_CONV network on the test set. The average loss was 0.0809 and accuracy was 142/10,000 which is **1.42%.** Although the loss is not smaller than the one achieved using the original model, the accuracy decreased even further. This indicates that while the likelihood of the true labels according to the CNN on the perturbed images remained somewhat high in most cases, the wrong labels received higher likelihood.

The MNS_LIN model on the other hand, showed inferior results compared with both other models. Although it seems to keep improving for more than 50 epochs after the other two models reached platto, it did not provide the better result. From this we can conclude that this model is inferior in terms of training time and accuracy.

The PLS_CONV model on the other hand showed faster learning with better accuracy (i.e. lower), indicating that further analysis of the input image using an additional convolutional layer was beneficial.

Next, all possible combinations of c (8, 16 or 32) and h (1204 or 2048) were assessed for 60 epochs[15] and the results are presented in **figure 2.5**[16].

[15] Following the results in the previous analysis, we decided to reduce the number of epochs as only little improvement happened after this point.

[16] For simplicity the numerical results are not presented, the submitted notebook contains all relevant results.

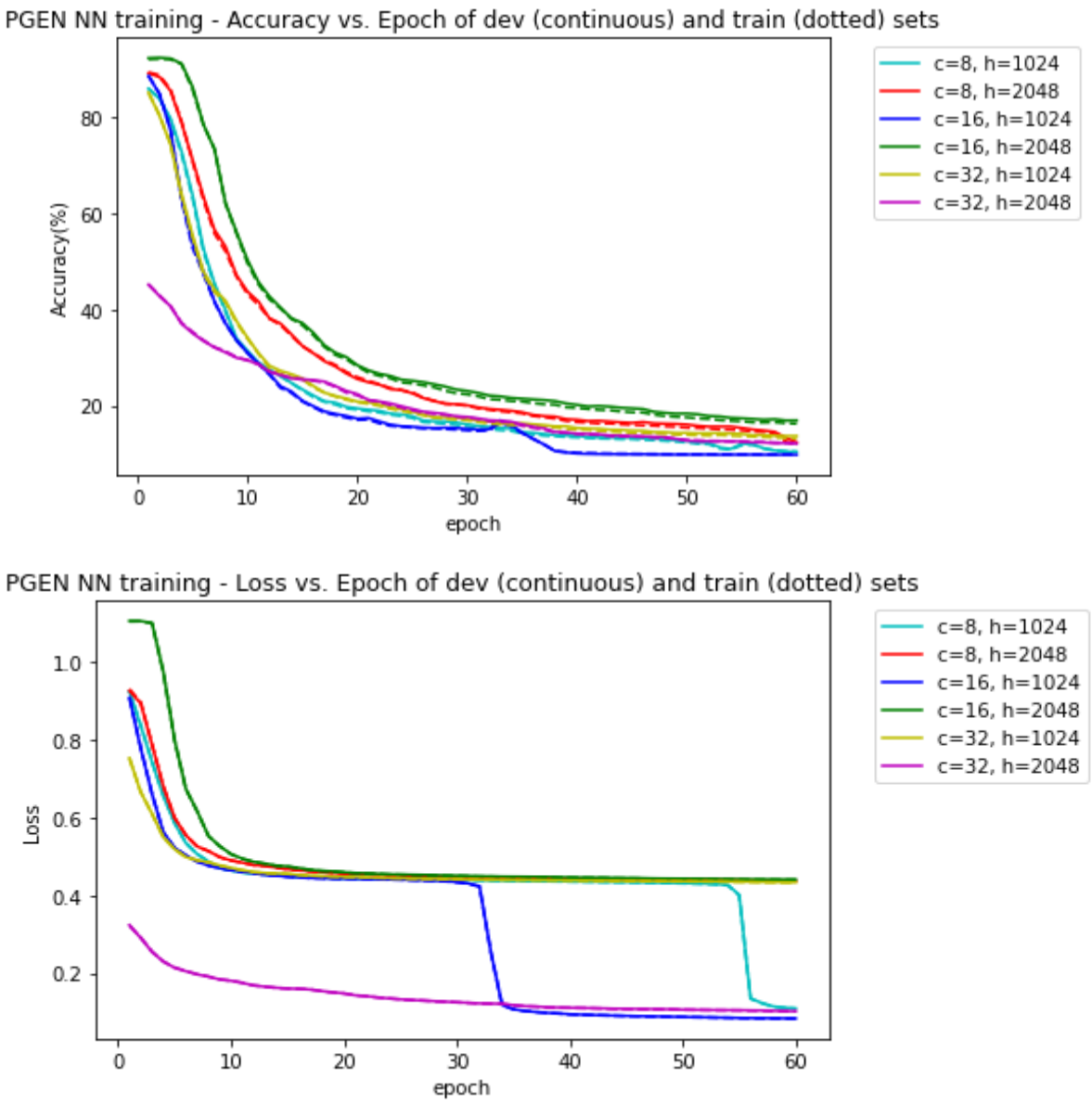

Figure 2.5: Learning process of the networks for 60 epochs on the train and dev sets in terms of accuracy (top) and loss (bottom). Accuracy is presented as the percentage of correctly labeled perturbed images by the CNN. c and h represent the output channels of the convolutional layer and the size of the hidden linear layer respectively.

From these results we can learn that our original model using c=16 and h=1024 exhibited the best accuracy and loss and even showed the fastest learning. Importantly the much more compact network[17] using c=8 and h=1024 reached comparable loss and accuracy to the original network, showing that 8 output channels for the convolutional layer are sufficient. The conclusion from this comparison is not clear as neither a specific c value nor a specific h value are related to superior/inferior results. Moreover, using c=16 yielded the best (with h=1024) and

[17] See the next section for number of parameters in each network

worst (with h=2048) results in terms of accuracy and loss, which suggests an important interaction between the different sizes of layers in the network.

The optimal network

In this section the models from the previous section were evaluated using the AIC and BIC. The evaluation was done separately on the train and dev sets. As these criteria require a likelihood estimation, we used the CNN to receive the probability for each perturbed image to be the correct label (more precisely we took the exponent on the output layer of the CNN to reverse the log of the log-softmax function) this value will be denoted as p. We then looked at 1-p, calculated its log and summed it across all examples, this value corresponds to the log-likelihood of the examples[18]. Other options are possible, such as using the positive log-likelihood or our own loss based on the inverse of the NLL, yet, to our understanding, none of these has an advantage over the aforementioned approach. First the number of parameters was calculated for each model, this is described in **table 2.1**.

Model	Number of parameters
c=8, h=1024	2,189,152
c=8, h=2048	4,377,440
c=16, h=1024 (original model)	3,573,680
c=16, h=2048	7,146,416
c=32, h=1024	6,342,736
c=32, h=2048	12,684,368
PLS_CONV	5,528,016
MNS_LIN	2,120,880

Table 2.1: Number of parameters for each model evaluated in the previous section.

Then, using the number of parameters and the log-likelihood for each model, AIC and BIC scores were calculated, the results are described in **table 2.2**.

[18] To handle low probabilities in some cases and to avoid log(0) when 1-p=0, we added a small $\delta=10^{-20}$. We are aware that this alters the results, yet it is a necessity.

Model	AIC score - dev	AIC score - train	BIC score - dev	BIC score - train
c=8, h=1024	4.38061e+06	4.38975e+06	2.01651e+07	2.36976e+07
c=8, h=2048	8.84248e+06	9.19495e+06	4.04053e+07	4.7803e+07
c=16, h=1024 (original model)	7.23496e+06	7.58743e+06	3.30024e+07	3.91065e+07
c=16, h=2048	1.42955e+07	1.43034e+07	6.58236e+07	7.73332e+07
c=32, h=1024	1.26878e+07	1.26969e+07	5.84211e+07	6.86384e+07
c=32, h=2048	2.53714e+07	2.53793e+07	1.1683e+08	1.37253e+08
PLS_CONV	1.10587e+07	1.10666e+07	5.09176e+07	5.98225e+07
MNS_LIN	**4.24406e+06**	**4.2532e+06**	**1.95363e+07**	**2.29589e+07**

Table 2.2: AIC and BIC scores of the networks based on the train and dev sets. In bold are the best (lowest) scores.

From these results, the MNS_LIN model seems to be the best. It is worth noting that this model also has the least amount of parameters, thus the results suggest that differences in accuracy do not have a significant influence on the scores.

Part 3

Formalization of the Problem as a Curiosity Problem

The purpose of the curiosity algorithm is to learn which parts of the data will be the most beneficial for the learning process of the noise-generating neural network (NN) which is the learner. In other words, the goal is to learn a policy determining how to choose the pictures for the train set for the NN.
We have decided to check whether pictures of some specific labels contribute more to the learning than of other labels, and in which order these labels are best to be revealed to the learner.

1. States and Actions

This problem is analogous to the curious feature selection (Moran & Gordon, 2019), in which the curious algorithm learns which features are the most effective for the learning, only in our case the "features" to be selected are the labels.
The state space is $s \in S$: $\{s_0, l_0, l_1, \ldots, l_9\}$, where s_0 is the initial state (a state before any label is selected) and l_i corresponds to the label i being previously chosen.
The actions are the labels that we are adding to the train set $a \in A$: $\{a_0, a_1, \ldots, a_9\}$where a_i corresponds to adding the data of label i to the train set.

2. Reward

The reward of the network is based on the change of the error of the learner. The initial error is 0.985 based on the success rate of the pre-trained classifying convolutional neural network (will simply be referred to as the classifying CNN). The error is the fraction of the data correctly classified by the classifying CNN after the perturbations made by the learner were applied.
Thus, given that the error at time t is denoted as e_t, and the initial error being $e_0 = 0.985$, the reward at time $t \geq 1$ will be $r_t = e_{t-1} - e_t$ corresponding to the improvement.

3. Closing the curiosity loop

The algorithm starts with the initialization of the Q matrix we wish to learn with values of 1, this creates a greedy policy with respect to the Q-values for the first few episodes. In addition, as described in the previous subsection, $e_0$is initialized as $e_0 = 0.985$. During

each episode a set of constant size is sampled and divided into training and evaluation sets (80% and 20% of the episode respectively)[19].
On each time step a new action is chosen based on the previous state. According to the chosen action, the data from the training part of the episode sample with the corresponding label is added to the data the model is trained on. After training, the model is evaluated on the entire evaluation set from the episode data. Based on this evaluation an error e_t and the reward r_t are calculated. Then, using the r_t the Q matrix is updated according to Bellman's rule of optimality.
Higher reward is given when the learner reduces the error more dramatically, and lower (negative) reward is given when the error of the learner increases. This reward system yields a model that is able to choose the most informative data to train the learner on and benefits the most when a drastic improvement follows an action.

Programming

1. Running the algorithm

The code files are submitted with this report.

2. Results

The curiosity loop was run for 1000 episodes (where each episode contained 600 examples which is 1% of the entire train set) in order to go over all of the states several times.
During the learning process both decrease in error and in loss were achieved (see **figure 3.1** and **figure 3.2**). Error was calculated as the accuracy of the classifying CNN on the evaluation set, and similarly, loss was calculated as the inverse negative log-likelihood (NLL) of the classifying CNN's output on the perturbed images. In both cases, these values were averaged across all steps of the episode.

[19] Any sampling and splitting of the data in the training was balanced, meaning that the number of examples of each class was the same for all the classes. In addition, the ratio of each class was maintained when dividing the episode data to train and evaluation sets.

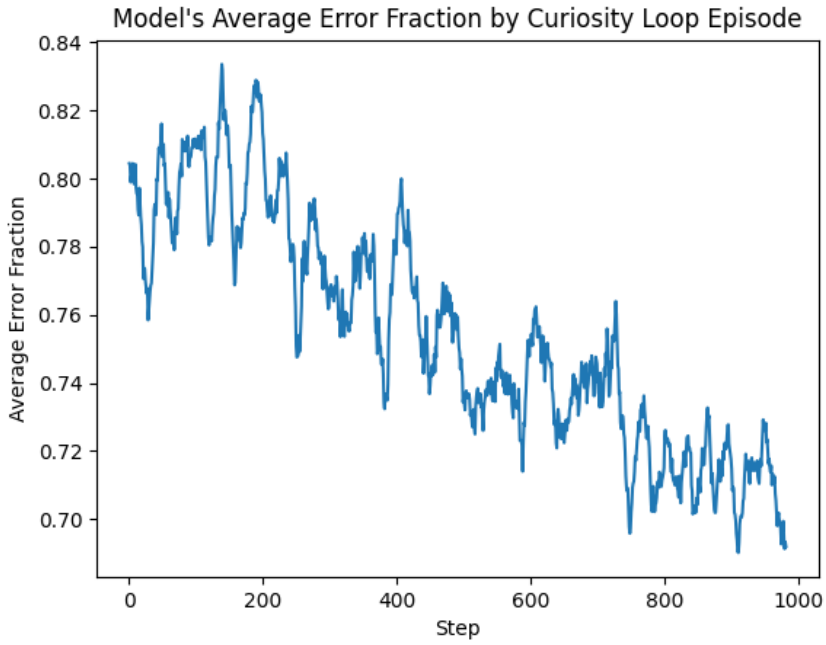

Figure 3.1: Average error over each episode. Average error is measured as the accuracy of the classifying CNN averaged over all steps within an episode. This graph shows a moving average of the average error with a window size of 20.

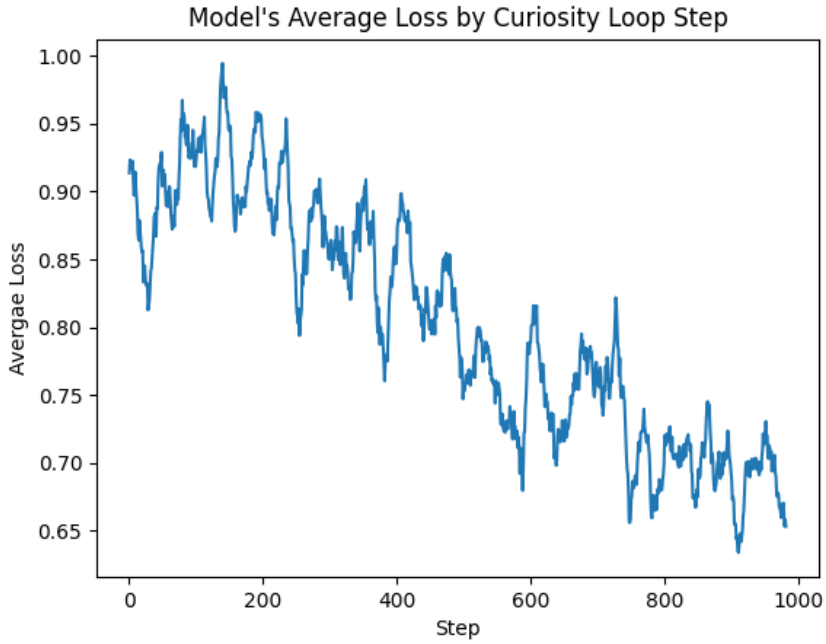

Figure 3.2: Average loss over each episode. Average loss is measured as the inverse NLL of the classifying CNN averaged over all steps within an episode. This graph shows a moving average of the average loss with a window size of 20.

The Q-table that was obtained during the learning process is presented in **table 3.1**. We can see that all of the values on the s_0 are larger than the initial value of 1 , which means that all of the labels can contribute to the learning of the noise-generating network. Yet, some values for states other than s_0 are lower than the initial value of 1, which indicates low to negative decrease in error[20]. As expected, the diagonal contains the initial value of 1 as each action can only be performed once.
According to the results, the optimal policy is $l_4 \rightarrow l_7 \rightarrow l_2 \rightarrow l_0 \rightarrow l_6 \rightarrow l_1 \rightarrow l_3 \rightarrow l_8 \rightarrow l_5 \rightarrow l_9$.

	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9
s0	1.101	1.024	1.205	1.153	1.25	1.197	1.153	1.147	1.047	1.079
l0	1	0.828	0.918	0.876	0.993	0.868	0.887	0.938	0.849	0.826
l1	0.888	1	0.91	0.863	1.00012	0.854	0.901	0.92	0.854	0.852
l2	0.913	0.853	1	0.87	0.973	0.868	0.868	0.935	0.853	0.809
l3	0.841	0.864	0.877	1	0.931	0.866	0.882	0.921	0.875	0.87
l4	0.895	0.784	0.903	0.844	1	0.873	0.88	0.907	0.794	0.843
l5	0.867	0.826	0.847	0.849	0.976	1	0.868	0.838	0.854	0.882
l6	0.888	0.839	0.86	0.836	1.014	0.836	1	0.877	0.836	0.836
l7	0.899	0.831	0.934	0.846	0.961	0.866	0.894	1	0.87	0.803
l8	0.875	0.887	0.871	0.861	0.947	0.886	0.922	0.901	1	0.884
l9	0.91	0.873	0.85	0.881	0.93	0.947	0.882	0.905	0.867	1

Table 3.1: The Q-table, marked yellow the actions chosen at each state (i.e. maximal Q-value).

We have compared the acquired policy to the random one (each action is chosen randomly from the available actions). In this section we partitioned the training data to a train set of size 50,000 and an evaluation set of size 10,000. At each iteration we trained the learner on the training set and chose the best model out of 50 epochs using the evaluation set[21]. Following this part, the model was evaluated on a test set of 10,000 examples.
Figure 3.3 shows the graph of error on the test set at every policy step, averaged across 10 trials (note that the errors were calculated as discussed before). As expected, after the last step the learners reach similar error rates. In addition, we can see that for the first two steps the computed policy is preferable, although further on, the policy does not have an advantage over the random choice and even performs worse.
This may be due to the fact that the model does not take into consideration the history, but only the previous state. It is possible that using a more complex model with recurrent layers (e.g. LSTM or GRUs) could solve this problem and show superior results, although such a model will require more computing power and additional hyperparameters and so is out of scope for this project.

[20] note that the initialization results in a somewhat arbitrary scale and as will later be shown, some actions with values lower than 1 still result in improvement
[21] This process was done to assure best generalizability.

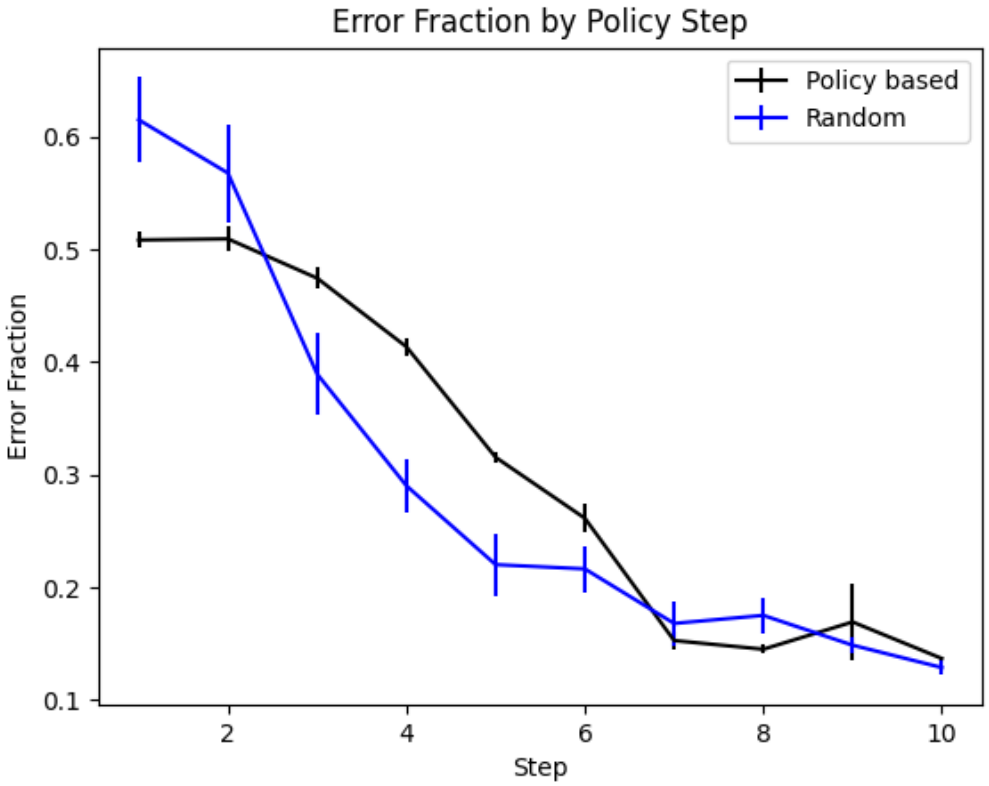

Figure 3.3:The error fraction against step (at each step the new label is added to the training set). Each point represents the average over 10 trials and is shown with standard error bars.

It is interesting to see that in both cases (policy based and random) after only a single step (i.e. training only on one class) the error decreased significantly (and much more than that class presence in the data). This suggests a common mechanism for the creation of perturbations for all classes. This is in line with previous research in this field (e.g. the fast gradient sign method that was presented in the first part of the project). Nevertheless, according to the Q-table presented in **table 3.1**, we do see variability between different classes that indicates that some classes are more beneficial for the learning process. It is reasonable to assume that this is a result of the average value of pixels (more signal may allow better learning) but this stands at odds with the fact that for s_0 both 1 and 8 have similar Q value (1.024 and 1.047 respectively) yet they are represented by different signal intensity[22]. Hence, we assume the differences in the informative value of different examples are more complicated and will require further research. It is worth emphasizing that according to this graph most classes had some informative value as the graph is approximately monotonic[23] indicating that complete training is better when possible, yet it is not clear whether additional examples of a specific class will be proven better than the addition of an unseen class.

[22] Note that this observation is intuitive (i.e. higher signal for 8 compared with 1) yet it was checked and confirmed.

[23] note that some increase in error is present after specific actions and according to the Q-table some actions when followed by others do not reduce error rate or only reduce it minorly.

Bonus - Mapping the Problem to the Brain and to Psychology[24]

First, we will discuss the mapping of the agent's role to the brain and psychology. From the neuroscience perspective it is reasonable to assume that the dopaminergic system will take part in the learning process as the problem contains some sort of a reward in the form of reduced error and improved performance over time. This means that ventral tegmental area (VTA) and substantia nigra (SN) complex will be active. These areas were shown to be active during question presentation (under assumption of a curious state), and thus we expect them to be active prior to the action selection. We also expect before an action is taken higher activity in the nucleus accumbens which is activated by the dopaminergic signals from the VTA/SN complex. Another expected observation is some activity of the hippocampus (HC) while encoding the new information (around the time of action selection and receival of reward), yet we note that the HC is commonly related to long term memory and looking at the learning as a short time process, consolidation of the memory is not required and hence the role of the HC in the context of encoding is minor. That said, the HC is also related to novelty detection and was shown to have effect in exploratory behavior (Voss et al., 2011).
In addition, after reward receival, we expect some sort of prediction error to be calculated, this was shown to be related to an activation in the ventral and dorsal striatum (Delgado et al., 2000; Knutson et al., 2000; Pagnoni et al., 2002; O'Doherty et al., 2003, 2004; McClure et al., 2003, 2004; Rodriguez et al., 2006), and was shown not only to be mediated by dopaminergic activity, but also is correlated with successful learning (Schonberg et al. 2007). Calculation of prediction error was also shown to cause activation in more nuclei of the basal ganglia including the globus pallidus (Joel, Niv & Rupin, 2002). Based on the review by Rushworth et al. (2011), the frontal cortex is engaged in reward-related learning and decision making in this context, specifically, the areas with an important role are the lateral orbitofrontal cortex, the ventromedial prefrontal cortex and adjacent medial orbitofrontal cortex, anterior cingulate cortex, and the anterior lateral prefrontal cortex. The functions of these areas are diverse and include expectations of loss and gain, value evaluation, comparison between the values of different options, creating action-reward association and exploratory behavior.

From a psychological point of view, the problem can be described with operant learning terminology. In operant learning, the learner faces a stimulus (or state) S, with possible response (or action) R and some reward O. According to Thorndike's law of effect (1911), if following a response R in the presence of some stimulus S there is a satisfying event, the association between the stimulus and the response would increase (i.e. the S-R association). According to this logic, given a state S the learner will respond with response R that has the highest S-R association of all possible responses (action selection). In addition, R-O associations are also created, indicating some expectation for specific reward following a response, similar to the aforementioned information from the neuroscience perspective.

[24] Any unreferenced information that is presented here is based on the lecture of Matthias Gruber.

Apart from that, it is possible to look at the problem under the scope of problem solving as a probabilistic problem in order to understand the possible strategies[25]. In probabilistic learning the learner faces various options (in our case possible actions), each one will yield some reward in a probabilistic manner unknown to her (in our case this is the error rate, which is probabilistic due to random initialization of the networks).
In probabilistic learning, there are three noteworthy strategies. The first is maximization, in this approach, after the first few responses, the learner stabilizes on a specific response for the remainder of the session. Another strategy is alteration, in which the response is changed frequently in order to cover as many possibilities as possible. The third hypothesis is testing, in which the probabilistic principle is denied by the learner and she aims at finding the rules behind the task.
Each of these strategies can be applied by a human learner in the process (balance between the maximization and alteration strategies are implemented in many reinforcement learning algorithms as the exploration-exploitation tradeoff), yet it is clear that if each of the first two strategies would be used exclusively the results are likely to be relatively inferior. A curious human learner may lean towards the hypothesis testing strategy, aiming at finding the rules behind the problem. In this case both responses that were found beneficial will be repeated but also new ones and new combinations of responses.

Another learning process to discuss is the learner's task (i.e. the noise generation). In this case we will explain why a human learner will most likely fail the task. In this case, the learner's input is an image of a handwritten digit. Visual input reaches the primary visual cortex (V1) of the brain in the occipital lobe. From V1 the information progresses and undergoes further processing, it then continues through two pathways (Goodale & Milner, 1992), the ventral (what) pathway and the dorsal (where/how) pathway. The processing in the brain still surpasses in many fields the performance of state of the art artificial neural networks, thanks to its complex structure and holistic processing[26]. While we have shown throughout our project that it is possible to reduce a trained network's accuracy to chance level by adding generated noise, we have also shown multiple examples demonstrating the fact that these perturbations minorly affect human perception. In light of these arguments, we assume this problem can not be solved efficiently by a human. It is worth emphasizing that there could be different formalizations of the problem and/or parallel problems which may be more suitable given the human's brain processing, yet they might be less intuitive.

[25] The information in this paragraph is based on the course: "Cognitive Psychology: Problem Solving and Creativity" which is passed in the psychology department of Tel Aviv University.
[26] You may refer to gestalt theory for some rules (although not complete in the context of holistic processing).

References

Delgado MR, Nystrom LE, Fissell C, Noll DC, Fiez JA (2000) Tracking the hemodynamic responses to reward and punishment in the striatum. *J Neurophysiol* 84:3072–3077.

Goodale, M. A., & Milner, A. D. (1992). Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1), 20-25.

Joel, D., Niv, Y., & Ruppin, E. (2002). Actor–critic models of the basal ganglia: New anatomical and computational perspectives. *Neural networks*, *15*(4-6), 535-547.

Knutson B, Westdorp A, Kaiser E, Hommer D (2000) FMRI visualization of brain activity during a monetary incentive delay task. Neuroimage 12:20 –27

McClure SM, Berns GS, Montague PR (2003) Temporal prediction errors in a passive learning task activate human striatum. *Neuron* 38:339 –346.

McClure SM, York MK, Montague PR (2004) The neural substrates of reward processing in humans: the modern role of FMRI. *Neuroscientist* 10:260 –268.

Moran M, Gordon G. “Curious Feature Selection.” Information Sciences, vol. 485, 2019, pp. 42-54.

O’Doherty J, Dayan P, Schultz J, Deichmann R, Friston K, Dolan RJ (2004) Dissociable roles of ventral and dorsal striatum in instrumental conditioning. *Science* 304:452–454.

O’Doherty JP, Dayan P, Friston K, Critchley H, Dolan RJ (2003) Temporal difference models and reward-related learning in the human brain. *Neuron* 38:329 –337.

Pagnoni G, Zink CF, Montague PR, Berns GS (2002) Activity in human ventral striatum locked to errors of reward prediction. *Nat Neurosci* 5:97–98.

Rodriguez PF, Aron AR, Poldrack RA (2006) Ventral-striatum/nucleus accumbens sensitivity to prediction errors during classification learning. *Hum Brain Mapp* 27:306 –313.

Rushworth, M. F., Noonan, M. P., Boorman, E. D., Walton, M. E., & Behrens, T. E. (2011). Frontal cortex and reward-guided learning and decision-making. *Neuron*, *70*(6), 1054-1069.

Schönberg, T., Daw, N. D., Joel, D., & O'Doherty, J. P. (2007). Reinforcement learning signals in the human striatum distinguish learners from nonlearners during reward-based decision making. *Journal of Neuroscience*, *27*(47), 12860-12867.

Voss, J. L., Gonsalves, B. D., Federmeier, K. D., Tranel, D., & Cohen, N. J. (2011). Hippocampal brain-network coordination during volitional exploratory behavior enhances learning. *Nature neuroscience*, 14(1), 115-120.

Appendix to part 1

Alternative Data:

An alternative in which only samples that were classified correctly before manipulation but wrongly after it were examined as well. This approach as mentioned results in much less examples, thus we repeated the aforementioned process of noise generation 500 times. The results of the Bayesian procedure are presented in **figure a1**. We can see that the information gain remains high throughout the entire process, indicating that no meaningful learning took place as the model was constantly “surprised” by the data. In addition, a large fraction of the possible combination of state and action did not occur in this data, increasing the difficulty of reaching conclusions from this process.

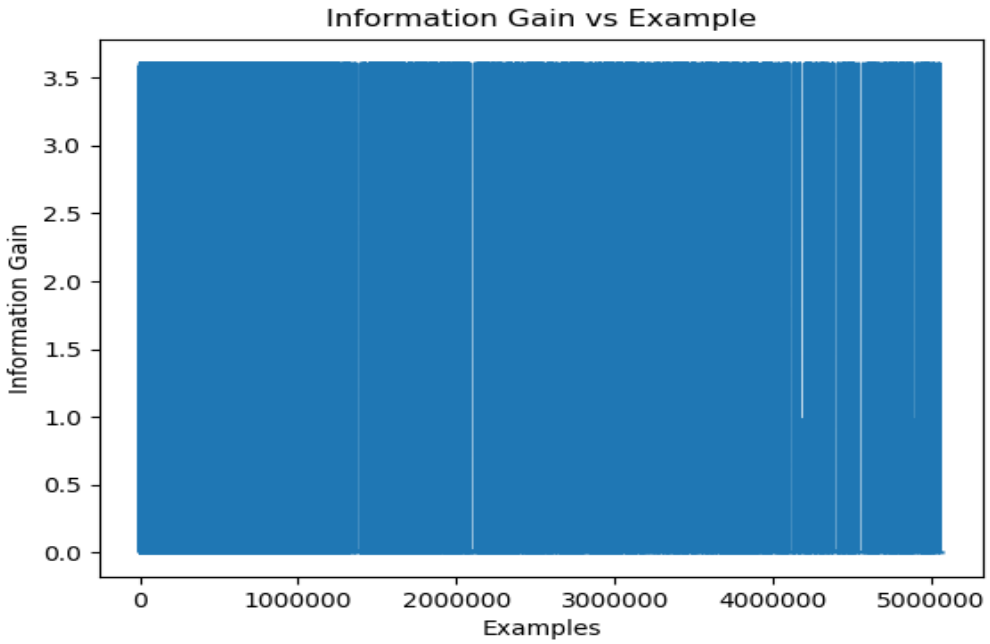

Figure a1: Information gain by example in the Bayesian updates procedure using only examples mispredicted after alteration but not before and block size of 2 by 2.

Appendix to part 2

A - The Classifying CNN

For illustration we trained again the CNN (this was not the network used as we wanted to use the same network as in the previous part for comparison). Note that for this we used the patience principle (i.e. we stopped the training when for 20 epochs we have seen no improvement in terms of the dev loss). The average loss of this model on the test set was 0.0475 and the accuracy was 98.73%, the training procedure is described in **figure a2**.

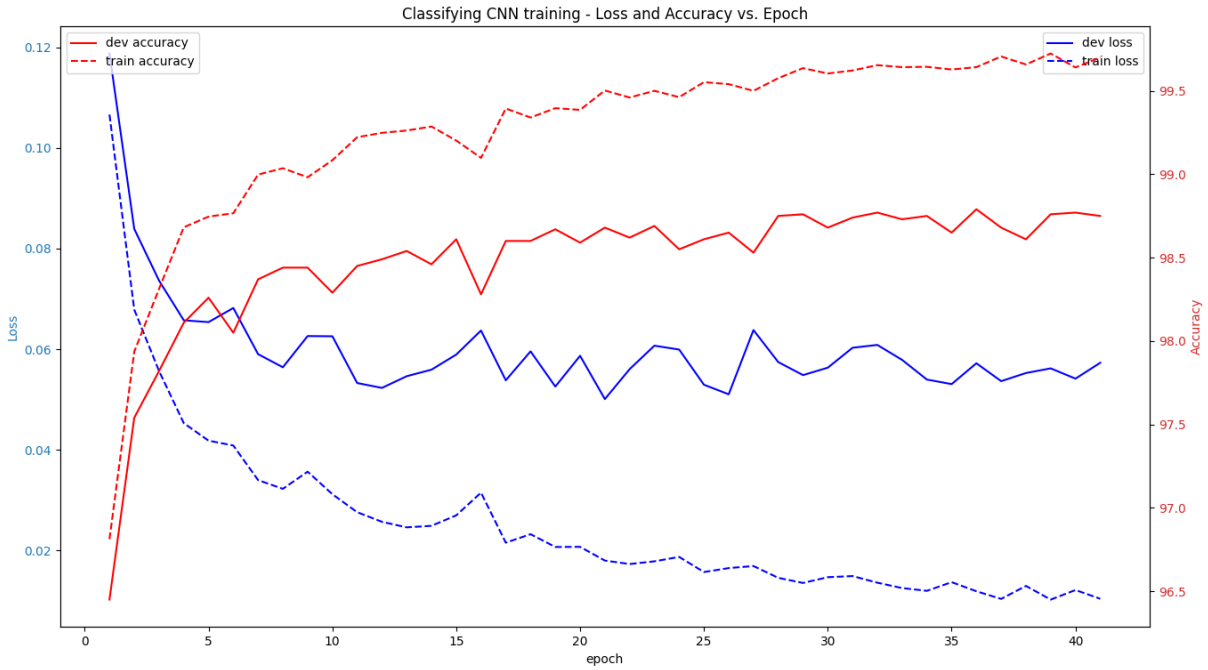

Figure a2: Learning process of the classifying CNN showing the change in accuracy and average loss on the train and dev sets. In blue are the losses, in red are the accuracies; in continuous line the dev set and in dotted line train set.

B - Perturbed Images Examples

This appendix's goal is to demonstrate the performance of the PGEN_NN model by creating adversarial examples, these examples are shown in **figure a3**.

Figure a3: Adversarial examples created using the original model based on random images from the test set. Above each image is the original prediction followed by the prediction on the perturbed image. It is possible that the original prediction was wrong, yet as it is a rare case with the CNN's accuracy reaching 98.48% we did not give this a special consideration.

We can see that although the network achieves around 10% accuracy, it predicts “8” for most of the examples, and thus loses its predictive power entirely. To address the question regarding the relations between the predicted label and the true label of the adversarial examples we ran a full comparison on the test set, the results indeed support the argument that the CNN had lost its predictive power. The results are shown in **figure a4**[27].

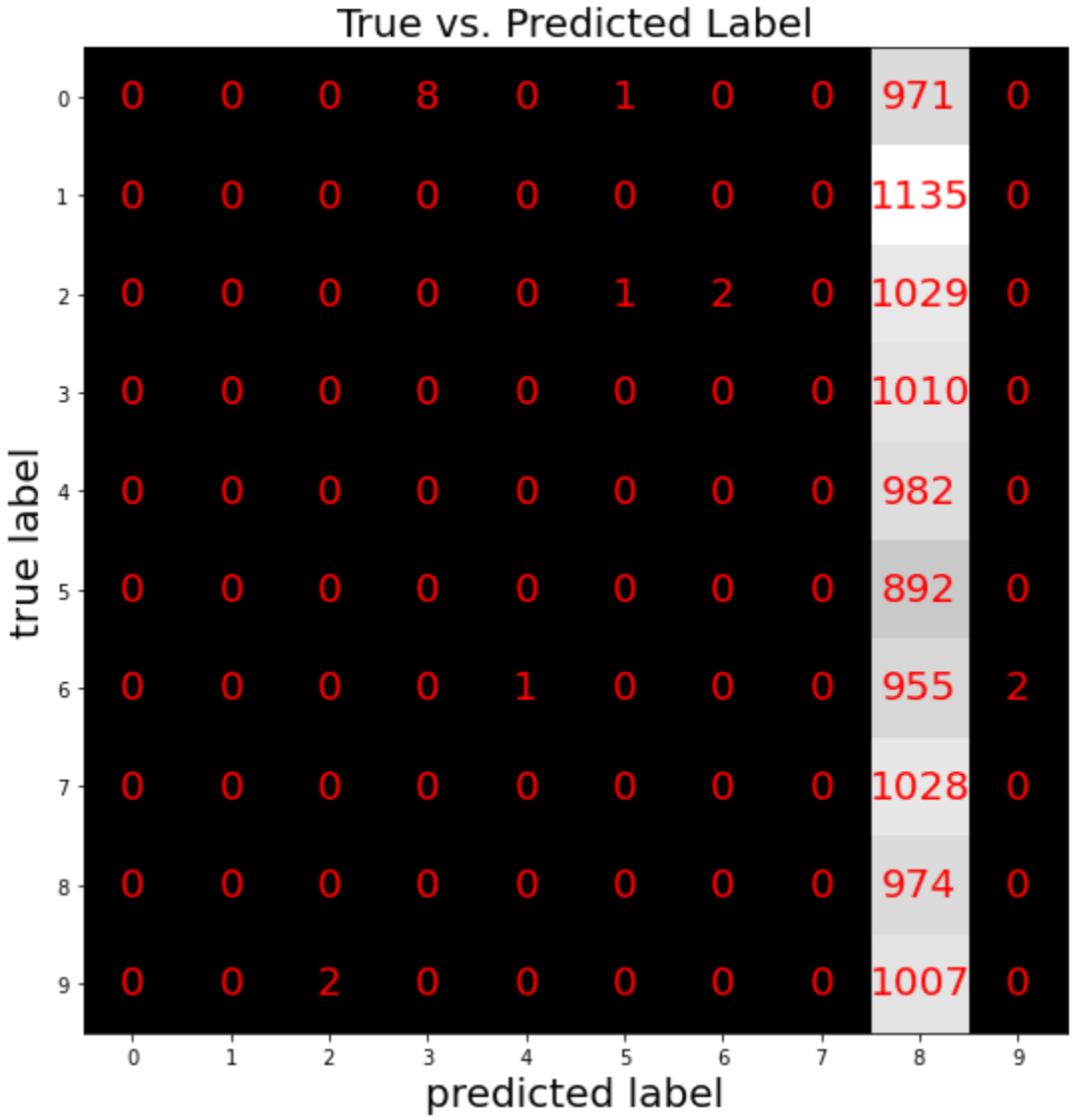

Figure a4: True vs. predicted label of the adversarial examples on the test set using PGEN_NN. In each cell is the count of the cases.

C - Regularization

In addition to the comparisons described in the work, we wanted to evaluate and analyze the amount of cells that were changed. For this section, a regularization term was added to the loss function. This regularization term is: $\gamma|N|$(N is the noise matrix produced by the last layer of

[27] For a similar plot testing PLS_CONV please refer to the submitted notebook.

PGEN_NN after the application of tanh activation function), i.e. the L1 norm of N (other options such as L2 norm are also possible) multiplied by some constant γ. This requires further processing of the noise before its addition to the original image by changing only cells in which the corresponding cell's absolute value of N reaches some threshold value. Note that this does not mean the corresponding cell in the image had actually been changed (values in the image are limited to the range between 0 and 1). We believe that it is reasonable to assume the model will be able to learn this and for simplicity we will refer to it when mentioning the number of altered cells.
The threshold value was set to be 0.9[28] for all models while the γ value changed. The γ values that were evaluated were 0, 0.01, 0.001 and 0.0001 and the results are described in **figure a5**[29] [30].

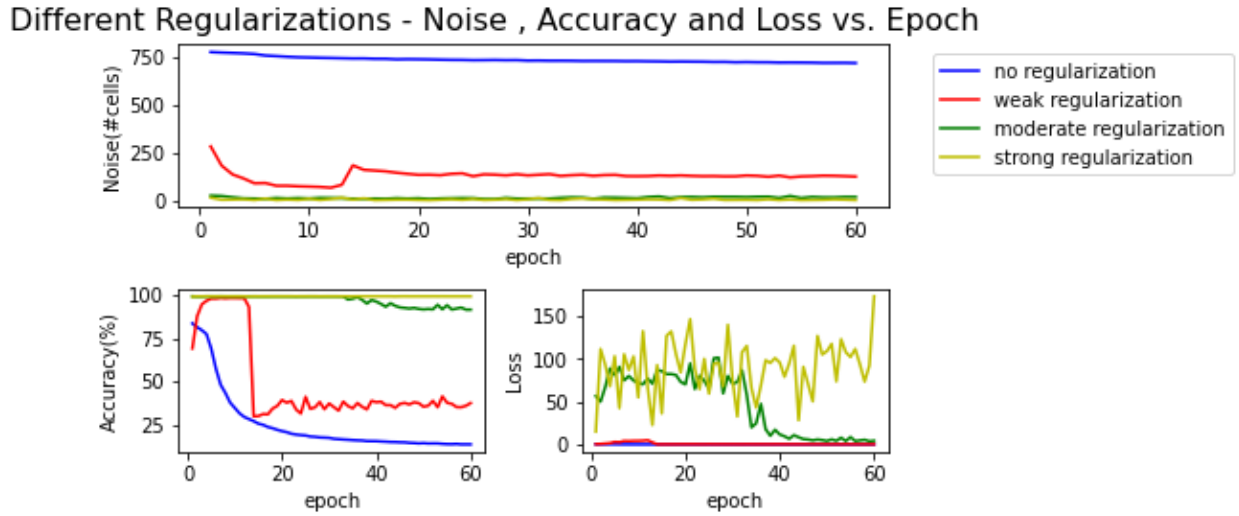

Figure a5: Learning process of the network showing the change in noise, accuracy and average loss on the dev sets. Top - noise; bottom left - accuracy; bottom right - loss.

As expected, we can see that the lower the regularization coefficient γ is, the accuracy is lower (better) and the number of altered cells is higher (worse).
The most important observation in this analysis is that while with no regularization most cells change, adding the weak regularization (i.e. $\gamma = 0.0001$) dramatically reduced the number of altered cells, while still creating a considerable amount of good adversarial examples.

[28] This value was chosen arbitrarily. For computational reasons more options were not evaluated, yet we believe the model should be able to handle this.
[29] Only the dev set's results were shown as previous inspection showed similar results for both train set and dev set.
[30] For simplicity the numerical results are not presented, the submitted notebook contains all unmentioned results.