Experiment Log 10.11~

1 Overall plan

1.1 Current stage

- Choose a well studied downstream task (mortality prediction, length-of-stay prediction), select features, form a sub-dataset by joining tables and filtering (refer to MIMIC docs)
- Build an NN for it (better easy to perform DD on, e.g. temporal convolutional network)
- Get a distilled dataset **that has the same structure as the original selected sub-dataset**
- Evaluate the DD on traditional classifiers as well as NN on the same objective

1.2 Future work

- Try different DD strategies
- Explore how to perform DD with traditional classifiers

2 Preliminary verification

2.1 Problem setup

- **Objective**: In-hospital mortality prediction based on the first 48hr of an ICU stay
- **Data**: ~20 selected features (variables), all in tabular format, from MIMIC (III or IV)
- **Motivation**:
    - Mainly inspired by the foundamental benchmark study on MIMIC-III: H. Harutyunyan et al. - Multitask learning and benchmarking with clinical time series data (2019)
    - Mortality is a primary outcome of interest in acute care: ICU mortality rates are the highest among hospital units (10% to 29% depending on age and illness), and early detection of at-risk patients is key to improving outcomes
    - The study selected out only **17** variables for all the 4 tasks, including mortality prediction, which is a relatively simple selected sub-dataset

Variable	MIMIC-III table	Impute value	Modeled as
Capillary refill rate	chartevents	0.0	categorical
Diastolic blood pressure	chartevents	59.0	continuous
Fraction inspired oxygen	chartevents	0.21	continuous
Glascow coma scale eye opening	chartevents	4 spontaneously	categorical
Glascow coma scale motor response	chartevents	6 obeys commands	categorical
Glascow coma scale total	chartevents	15	categorical
Glascow coma scale verbal response	chartevents	5 oriented	categorical
Glucose	chartevents, labevents	128.0	continuous
Heart Rate	chartevents	86	continuous
Height	chartevents	170.0	continuous
Mean blood pressure	chartevents	77.0	continuous
Oxygen saturation	chartevents, labevents	98.0	continuous
Respiratory rate	chartevents	19	continuous
Systolic blood pressure	chartevents	118.0	continuous
Temperature	chartevents	36.6	continuous
Weight	chartevents	81.0	continuous
pH	chartevents, labevents	7.4	continuous

    - For MIMIC-III, H. Harutyunyan et al. provided the code base; doing the similar thing on MIMIC-IV should not be too hard

2.2 Data processing

2.2.1 Feature selection

Useing the exact same pipeline of H. Harutyunyan et al., we have:

- **Size**
    - ~18k training subjects / stays
    - ~3k evaluating subjects / stays
- **Format**
    - Episodes (ICU stays) of **time series** of 48hr events, without a fixed sample rate (new timestamp is added each time a new lab/chart event happens)

Hours	Capillary refi	Diastolic bloo	Fraction insp	Glascow con	Glascow con	Glascow con	Glascow con	Glucose	Heart Rate	Height	Mean blood	Oxygen satu	Respiratory r	Systolic bloo	Temperature	Weight	pH
0.9013888888888889													18				
1.3180555555555555		126	0.5								138		0	177			
1.3347222222222221												100					
1.3513888888888888									107				19				
1.4513888888888888			1										0				
1.484722222222222								196									6
1.784722222222222									113				16		35.77777777777778		
1.9013888888888888		89							108		102	98	14	154			
2.084722222222222				To Pain	Localizes Pain		No Response-ETT										
2.201388888888889																	7.31
2.901388888888889		94							98		108	98	20	147			
3.151388888888889									98			97	19		36.111111111111114		
3.2680555555555557		108									118			152			
3.368055555555556											127						
3.3847222222222224		108												154			
3.401388888888889		110							98		127	94	18	155	36.111111111111114		
3.701388888888889																	7.33
3.901388888888889		119		To Pain	Flex-withdraws		No Response-ETT		99		135		18	160	36.22222222222222		
3.9180555555555556												94					
4.118055555555555			0.5										22				
4.651388888888889		104							88		118	100	22	141	36.111111111111114		
4.901388888888889		100							87		89		22	137	36.333333333333336		
5.451388888888888												96					
5.734722222222222																	7.36
5.901388888888889		109							86		98	100	22	153	36.166666666666664		
6.901388888888889		111							78		127	100	22	158			
7.901388888888889		120	0.5	To Pain	Flex-withdraws		No Response	135	85		133	100	22	163	36.44444444444444		
8.90138889		106		Spontaneous	Obeys Commands		No Response-ETT		78		129	100	22	169			
9.634722222222223																	7.48
9.90138889		101							67		123	100	22	159			
10.90138888888889		101							70		123	100	18	161	36.55555555555556		
11.90138888888889		92		To Pain	Localizes Pain		No Response-ETT		66		115	100	18	158	36.5		
12.084722222222222																	7.43
12.90138888888889		98							67		124	100	18	165			
13.40138888888889			0.5										16				
13.90138888888889		103						175	78		130	100	16	170	37.5		
14.484722222222222																	7.37
14.90138888888889		89							74		108	100	16	150			
15.90138888888889		92	0.5	To Speech	Obeys Commands		No Response-ETT		73		111	100	16	151			

    - Episode-level information (patient age, gender, ethnicity, height, weight) and outcomes (mortality, length of stay, diagnoses) are also available

- **Balance**
    - ~86% negative (safe)
    - ~14$ positive (mortality)

2.2.2 Preprocess

1. Resample: just like in the original paper, **resample** the timeseries to a fixed sample rate (1h), so that the length is unified
2. Recover missing variables: recover by **imputation** (forward filling), add mask columns for each feature column, representing whether the datapoint is imputed or real
3. Normilize each column using **Z-score normalization**
4. Each tensor is sized 48 (time steps) * 59 (num features, mask columns included)

2.3 Model

Mainly 2 types models to do the binary classification:

- `1DCNN`: 1-D CNN, with 2 conv layers and 2 fc layers (given that the temporal data has 1-D translational invariance)
- `MLP`: 3 fc layers

2.4 Experiments

2.4.1 Model capacity verification

This stage is to verify whether the dataset is good, and whether the model trained on train set can generalize onto test set.

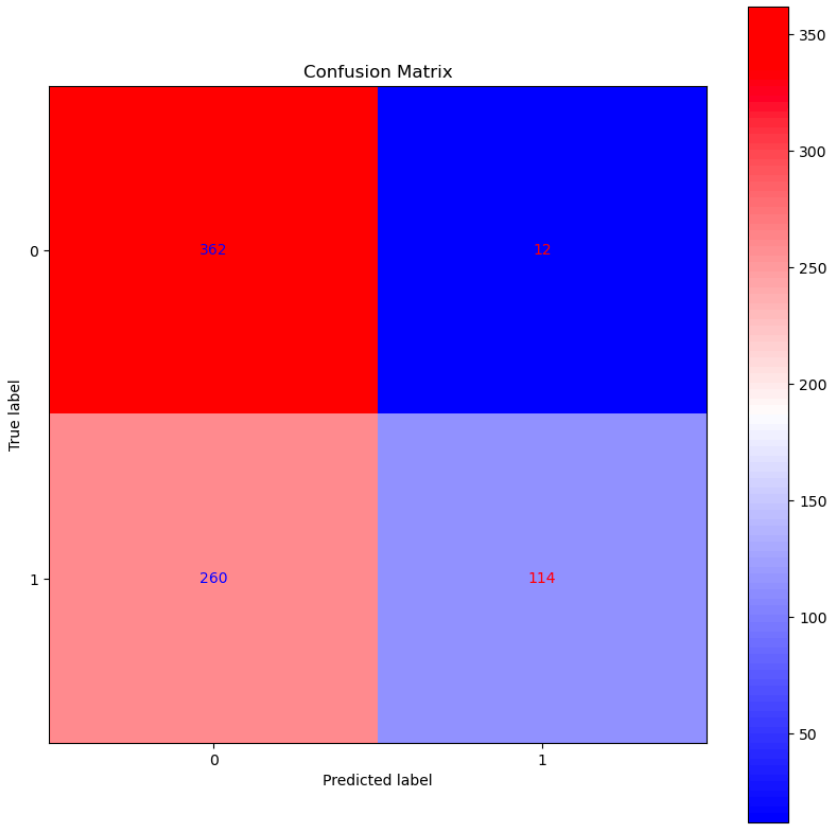
Training setup:

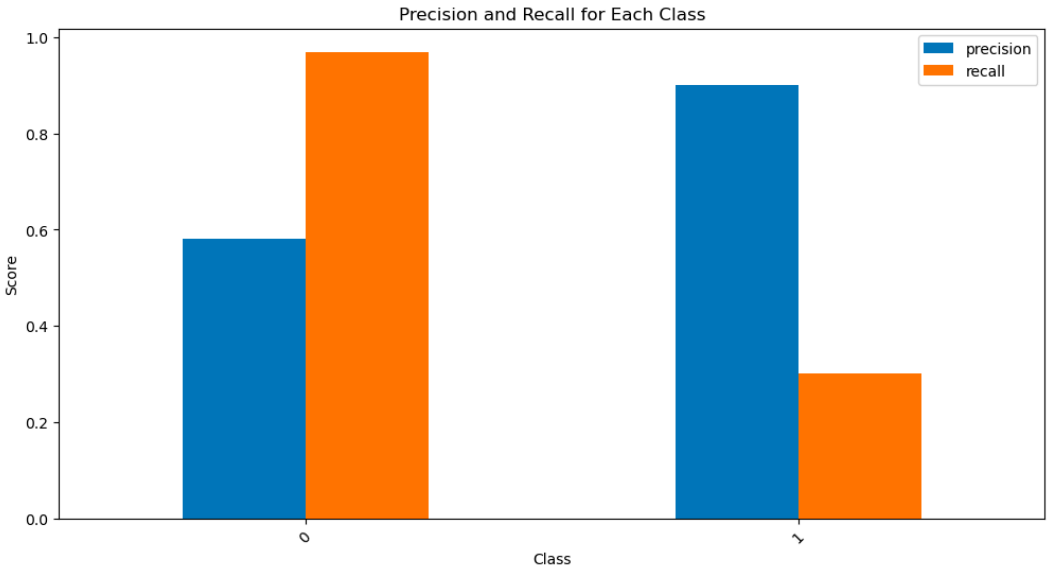
- lr = 0.001
- Optimizer = Adam
- Epoch = 100
- Data = unbalanced

On both models, test loss stops to decrease within 3 epochs, and then rise all the way up, which points to **severe overfitting**.

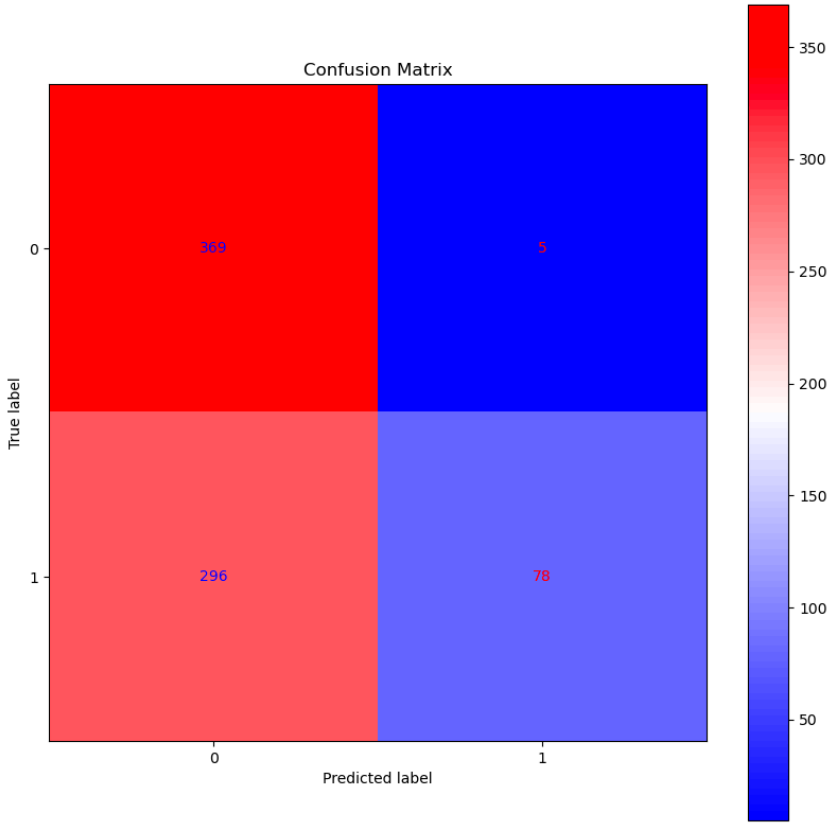
Pick the best performing epoch (overall acc ~90%), generate a classfication report, on a **balanced test set**:

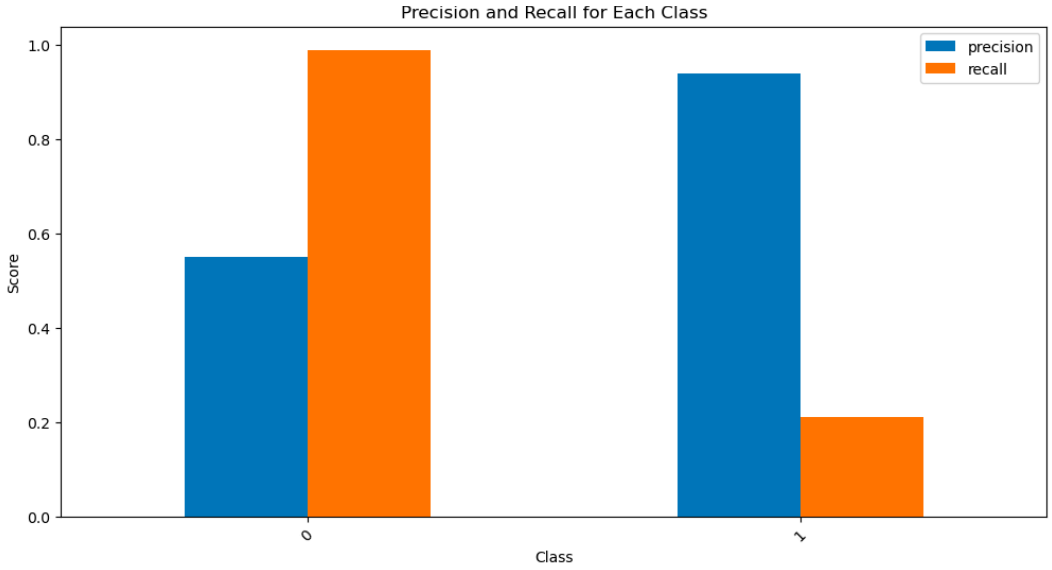
1DCNN:




Precision and Recall for Each Class
precision
recall
Score
Class
0
1
0.0
0.2
0.4
0.6
0.8
1.0

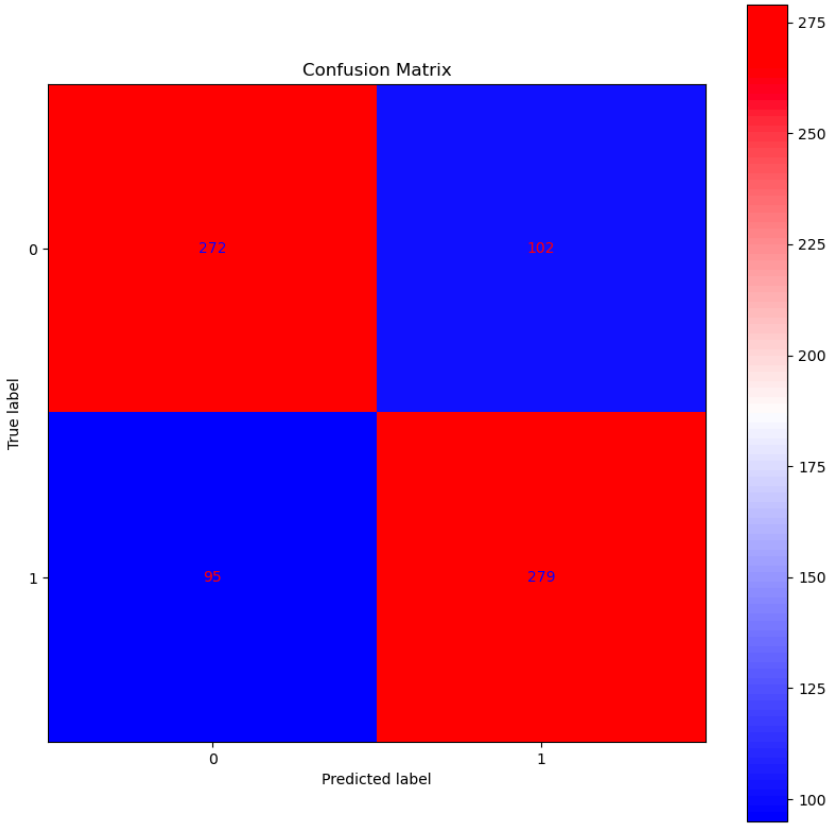
MLP:

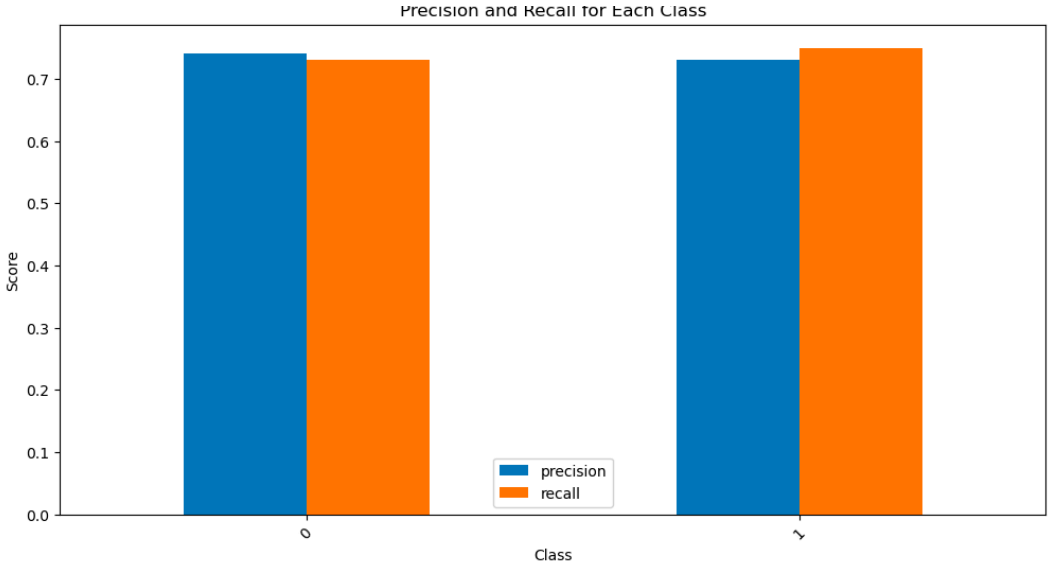

Confusion Matrix
True label
Predicted label
369
5
296
78
0
1
0
1
350
300
250
200
150
100
50



Further moves

- After configuring `weight_decay` to Adam (which allows L2 regularaztion), the overfitting is postponed, but not improving the best performance on test set (loss ~0.27)
- After taking out mask columns from training data, performance is slightly better (loss ~0.26)
- Also tried **training on balanced training set** and **evaluating on balanced test set** (by under-sampling)
  - Test acc is up to ~72%, which better than random guess for binary classification, but not impressive
  - Still suffer from overfitting: test loss starts to rise at around epoch 3





- Turn on mask again (with balance + mask), test acc is improved to ~75%
- AUC-ROC for metric

Observation summary

- Models generally suffer from overfitting
- Maybe the data itself just isn't good enough

2.4.2 Synthetic dataset distillation

Distilled dataset using Matching Gradients, 100 iterations.

Evaluate by:

- Train 2 models simultaneously, syn model trained on synthetic dataset, and real model trained on real dataset (balanced)
- Both models are evaluated (computing loss and accuracy) on real dataset after each epoch
- Compare both models' performance
- Result: syn model isn't learning anything, acc near 0.5 (random guess)

Vanilla method

Algorithm 1 Dataset Distillation

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data
Input: α: step size; n: batch size; T: the number of optimization iterations; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$
Initialize $\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly, $\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step $t = 1$ to T **do**
  Get a minibatch of real training data $\mathbf{x}_t = \{x_{t,j}\}_{j=1}^{n}$
  Sample a batch of initial weights $\theta_0^{(j)} \sim p(\theta_0)$
  for each sampled $\theta_0^{(j)}$ **do**
    Compute updated parameter with GD: $\theta_1^{(j)} = \theta_0^{(j)} - \tilde{\eta}\nabla_{\theta_0^{(j)}}\ell(\tilde{\mathbf{x}}, \theta_0^{(j)})$
    Evaluate the objective function on real training data: $\mathcal{L}^{(j)} = \ell(\mathbf{x}_t, \theta_1^{(j)})$
  end for
  Update $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha\nabla_{\tilde{\mathbf{x}}}\sum_j \mathcal{L}^{(j)}$, and $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha\nabla_{\tilde{\eta}}\sum_j \mathcal{L}^{(j)}$
end for
Output: distilled data $\tilde{\mathbf{x}}$ and optimized learning rate $\tilde{\eta}$

All latest experiment results

Model	Train set size	Distillation method	Traini settings	Eval on	AUROC	Comment
1DCNN	Original (15480+2424)	-	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.8340	Best performance (lowest eval loss) occurs in the first 5 epochs
MLP	Original (15480+2424)	-	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.8296	Best performance occurs in the first 5 epochs
1DCNN	20 (10+10)	Random sample	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.6980 (avg4)	Best performance occurs in the first 5 epochs
MLP	20 (10+10)	Random sample	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.7191 (avg4)	Best performance occurs in the first 5 epochs
1DCNN	100(50+50)	Random sample	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.7539 (avg4)	Best performance occurs in the first 10 epochs
MLP	100(50+50)	Random sample	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.7646 (avg4)	Best performance occurs in the first 10 epochs
1DCNN	500(250+250)	Random sample	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.7693 (avg4)	Best performance occurs in the first 20 epochs
MLP	500(250+250)	Random sample	Optim=Adam, lr=1e-3, wd=1e-3	Original test set (2862+375)	0.7817 (avg4)	Best performance occurs in the first 20 epochs
1DCNN	20(10+10)	Vanilla		Original test set (2862+375)	0.5421	
1DCNN	20(10+10)	Matching gradient	ol=10, il=50, lr_data=1e-3, lr_net=1e-3	Original test set (2862+375)	0.5177	
1DCNN	100 (50 + 50)	Matching gradient	ol=10, il=50, lr_data=1e-3, lr_net=1e-3	Original test set (2862+375)	0.5353	

2.4.3 Methodology verification on image distillation (11.15~)

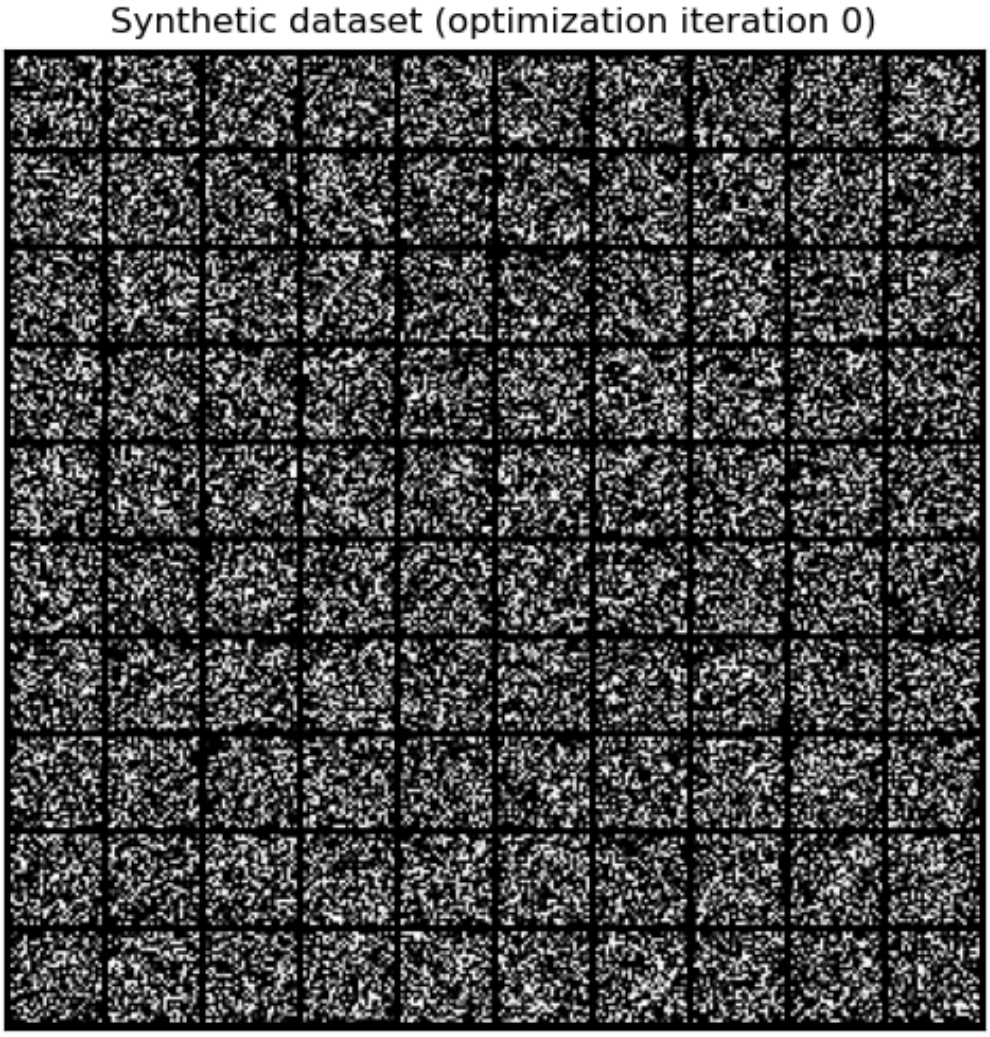
To verify the distillation methods as well as to grasp a basic idea of what a successful distilliation process will look like, adjust the codes for image distillation and test on MNIST / CIFAR10.

Exp. 1 Vanilla method on MNIST (fixed init)

Settings:

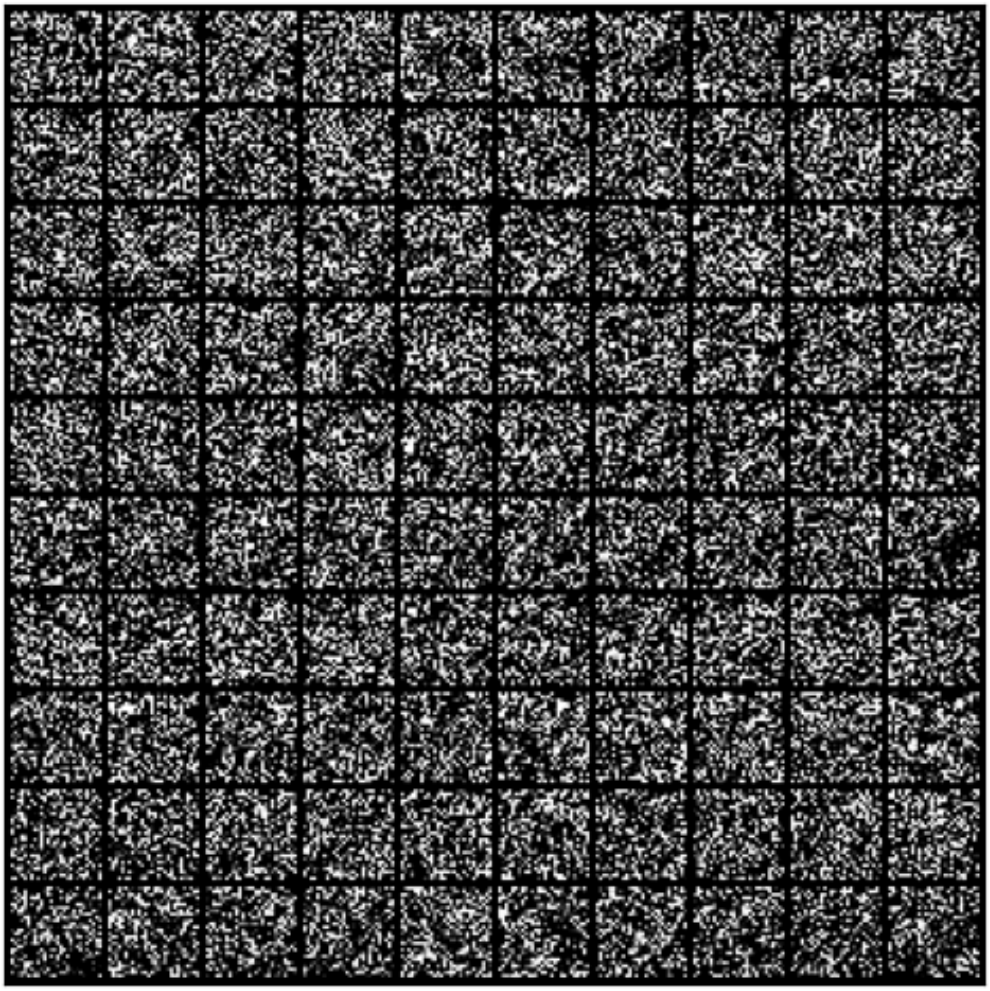
- **NUM_SAMPLES_PER_CLS = 10**
- **NUM_OPTIM_IT = 10000**
- EVAL_INTERVAL = 100
- INIT_LR = 0.001
- MINIMUM_LR = 1e-5
- STEP_SIZE = 0.001
- INIT_WEIGHTS_DISTR = "kaiming"
- **FIX_INIT_WEIGHTS = True**
- NUM_SAMPLED_NETS_TRAIN = 16
- NUM_SAMPLED_NETS_EVAL = 4
- NUM_SAMPLES_PER_CLS = 10

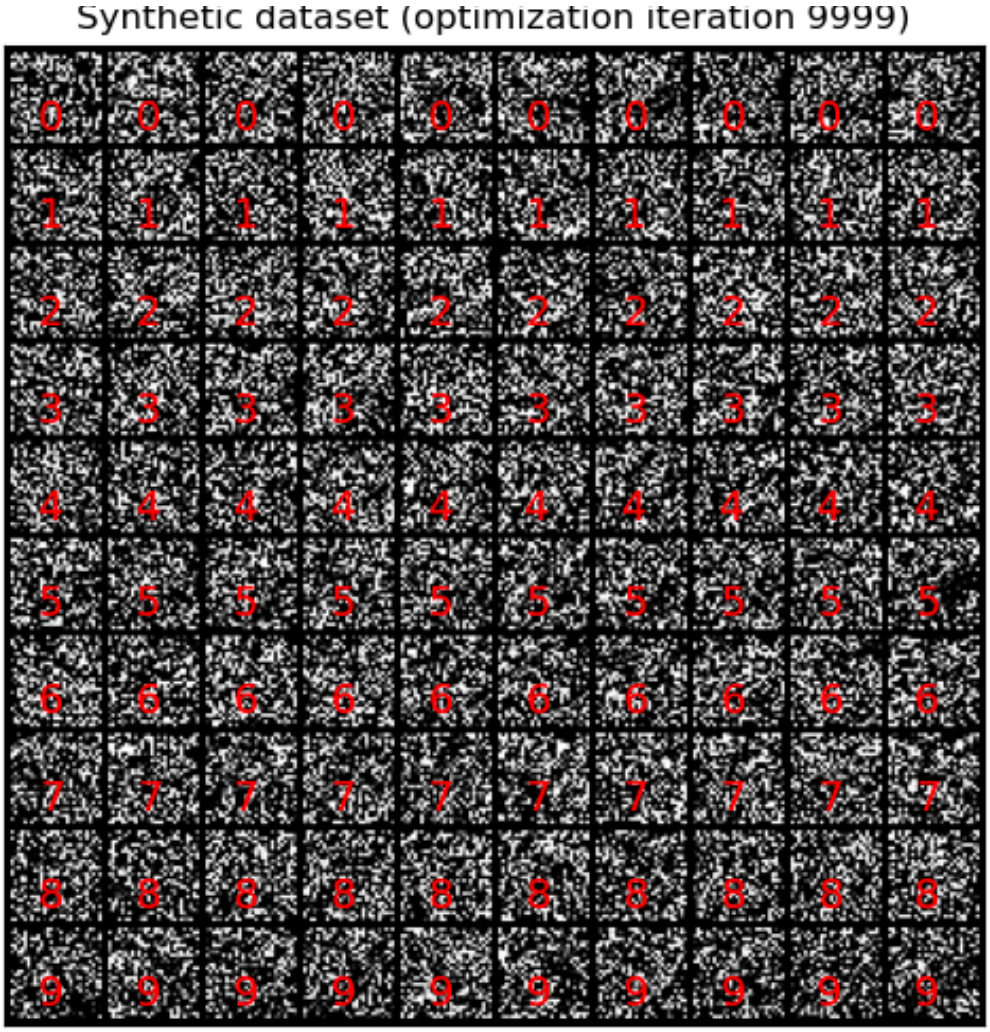
Original randomly initialized synthetic dataset:



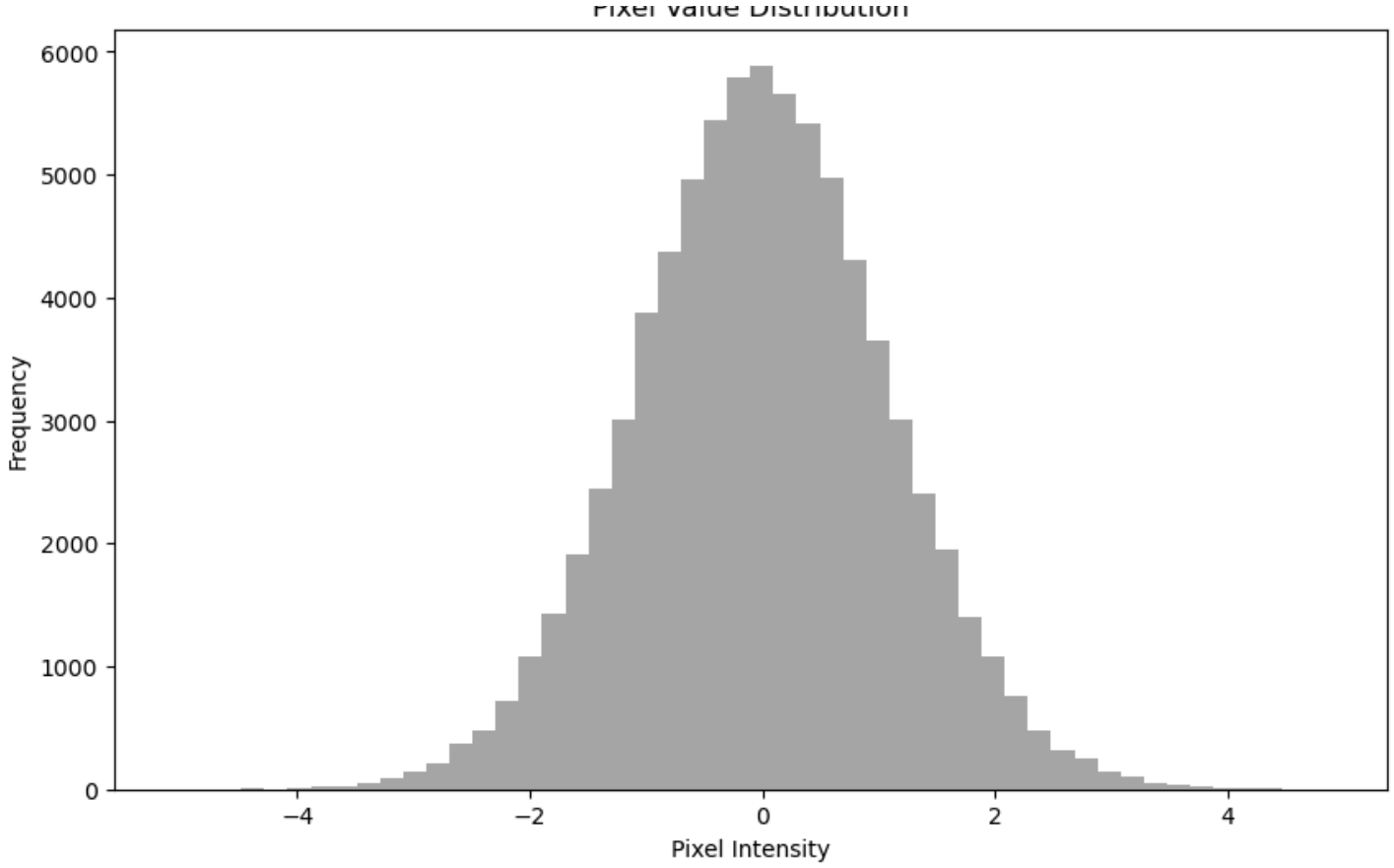
Synthetic dataset when training terminates:

Synthetic dataset (optimization iteration 9900)





Pixel Value Distribution



Training curves:

Evaluating on training set and test sets (train the model for one epoch using synthetic data and learnt LR):

	Train set	Test set
Accuracy (before)	0.0273	0.0225
Accuracy (after)	**0.9371**	**0.9424**

Conclusions:

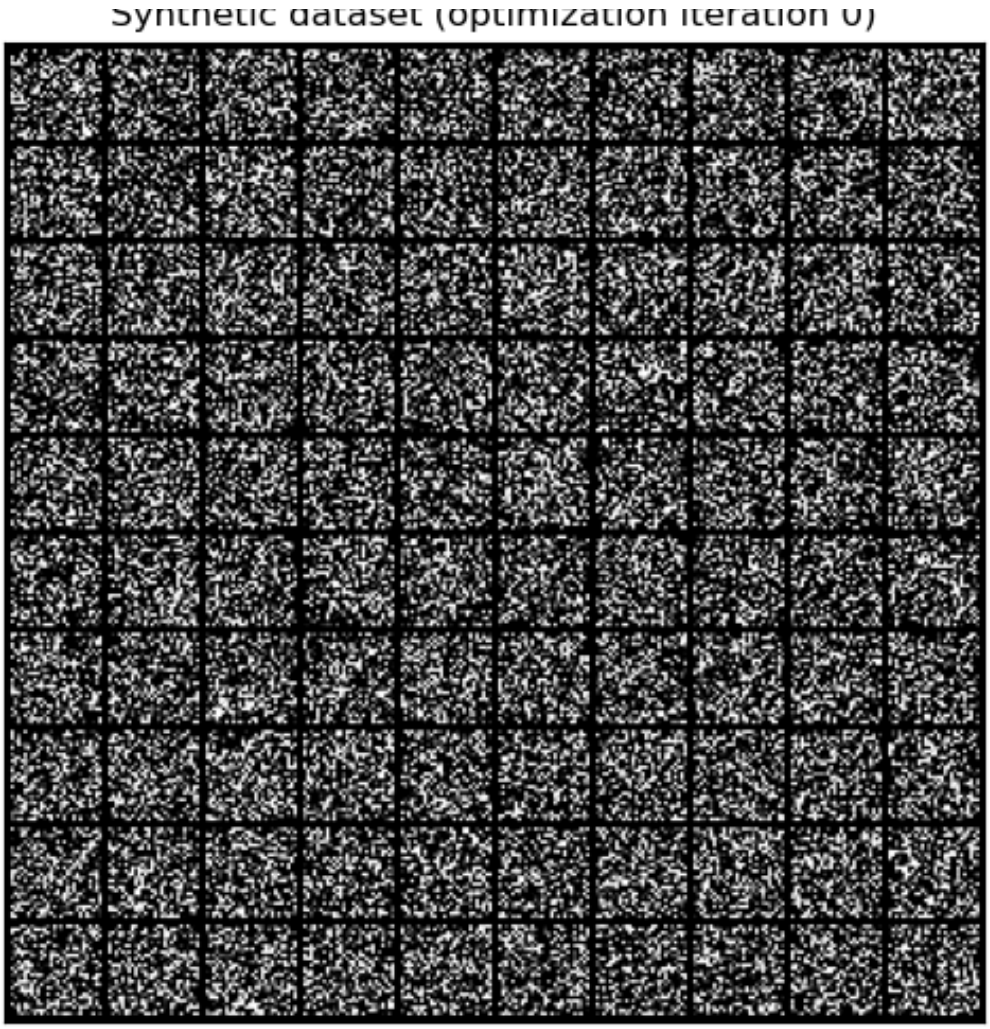
- Possible reasons why previous experiments (on EHR) was not working:
  - Distilled data was not fully trained
- Fixed initialization results in noisy, unrecognizable distilled images
- Distilled images' pixels are not bounded within [-1, 1]

Exp. 2 Vanilla method on MNIST (random kaiming init)

Settings:

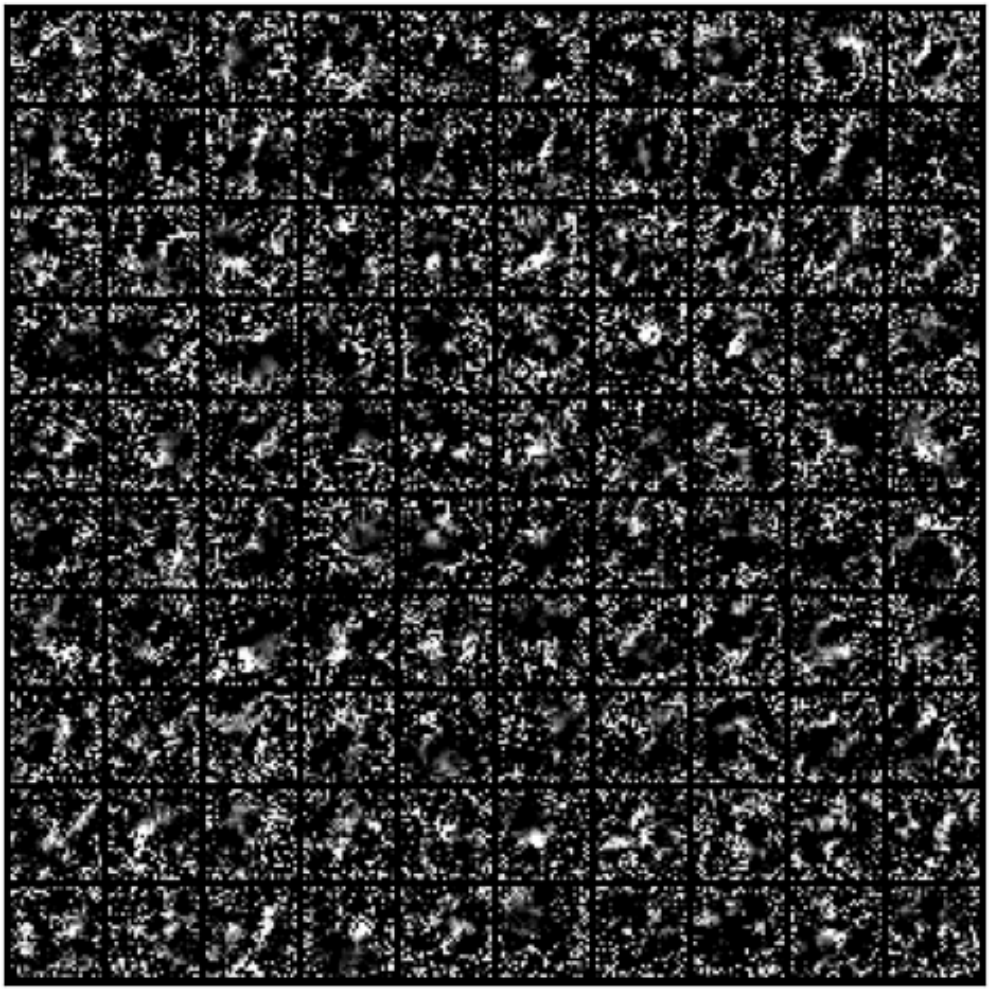
- **NUM_SAMPLES_PER_CLS = 10**
- NUM_OPTIM_IT = 10000
- EVAL_INTERVAL = 100
- INIT_LR = 0.001
- MINIMUM_LR = 1e-5
- STEP_SIZE = 0.001
- **INIT_WEIGHTS_DISTR = "kaiming"**
- **FIX_INIT_WEIGHTS = False**
- NUM_SAMPLED_NETS_TRAIN = 16
- NUM_SAMPLED_NETS_EVAL = 4
- NUM_SAMPLES_PER_CLS = 10

Original randomly initialized synthetic dataset:

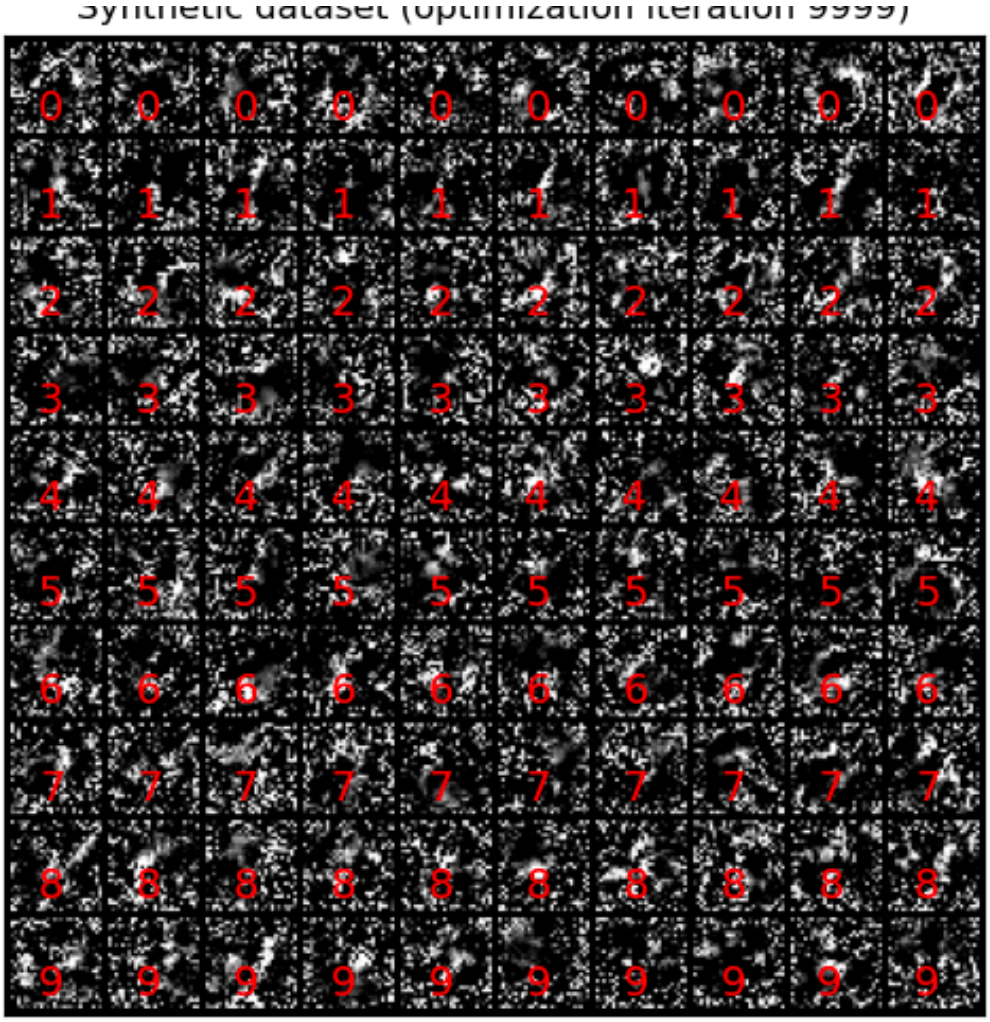


Synthetic dataset when training terminates:

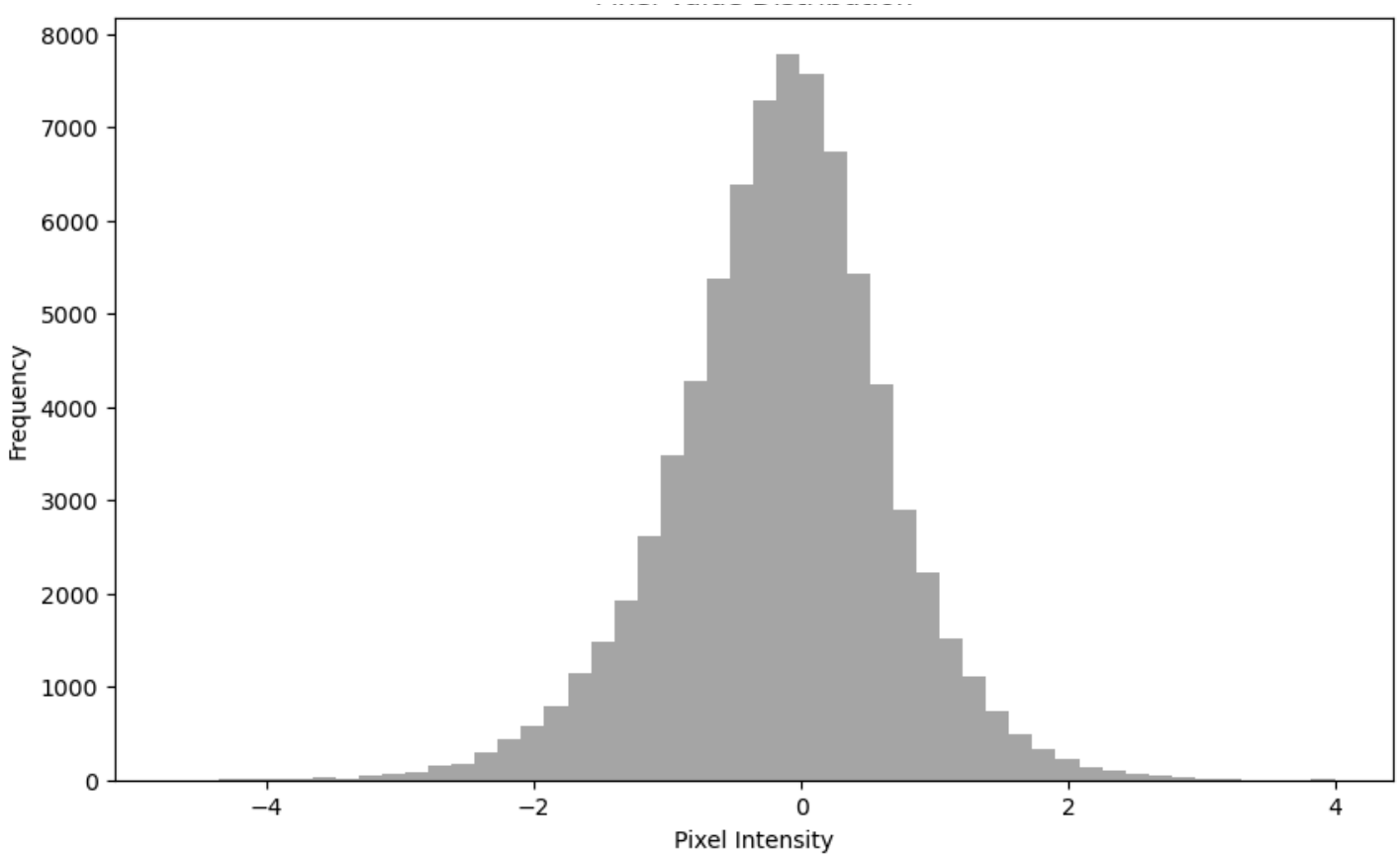
Synthetic dataset (optimization iteration 9900)



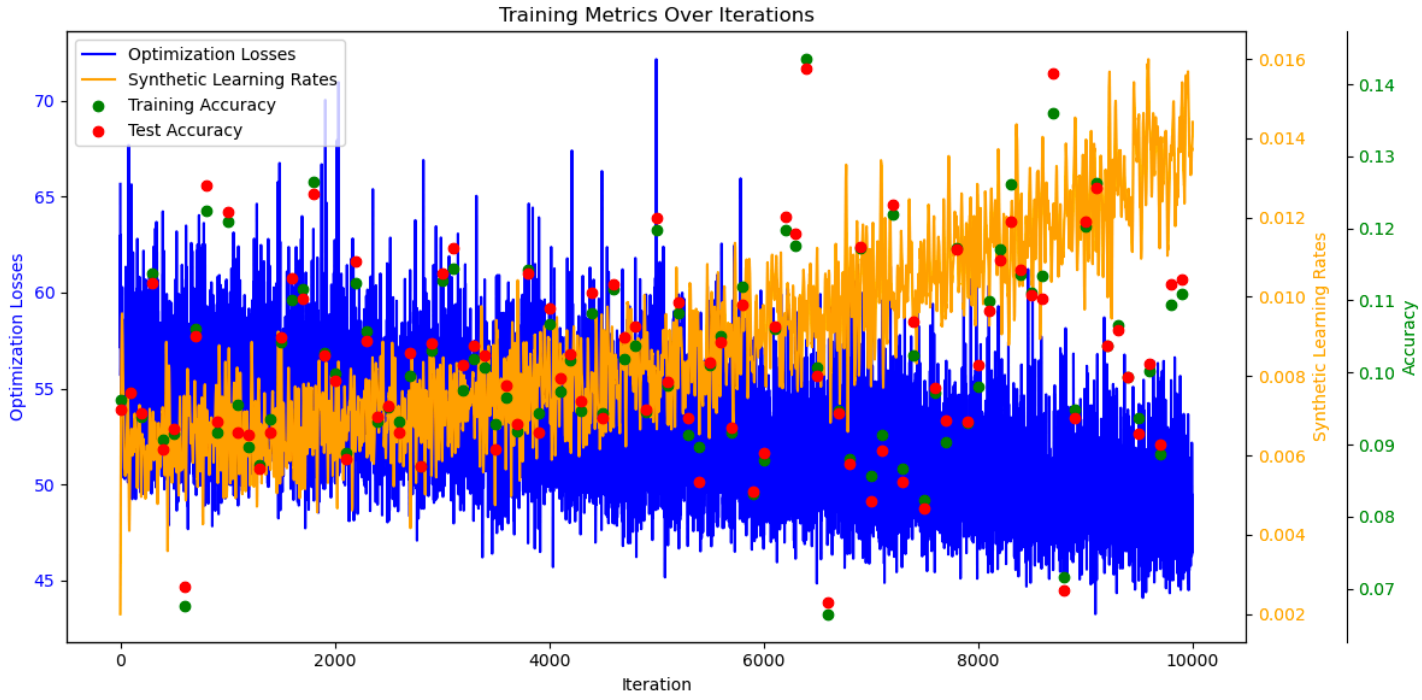
Synthetic dataset (optimization iteration 9999)



Pixel Value Distribution



Training curves:



Evaluating on training set and test sets (train the model for one epoch using synthetic data and learnt LR):

	Train set	Test set

	Train set	Test set
Accuracy (before)	0.0989	0.0979
Accuracy (after)	**0.1145**	**0.1136**

Conclusions:

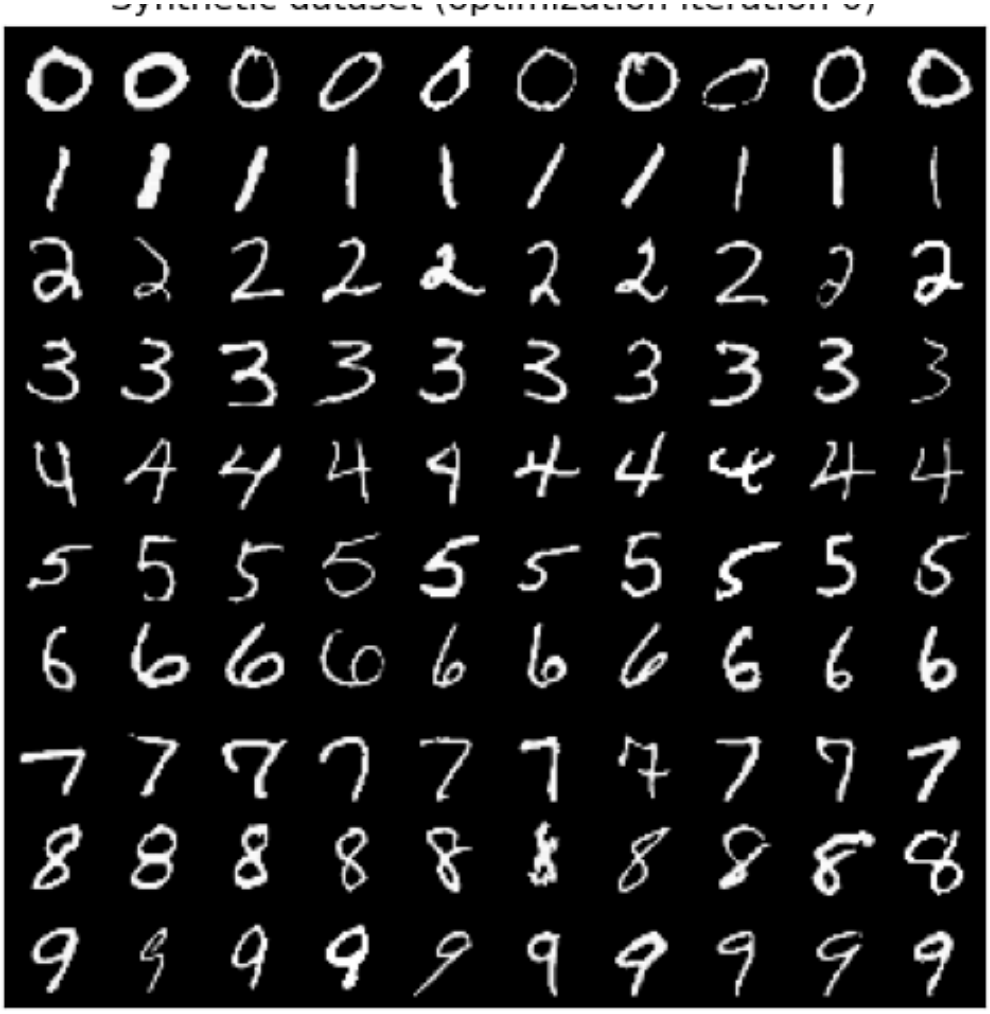
- We are seeing recognizable patterns in the distilled images! (Look at the '1's)
- Training isn't (computational) efficient and full, but it is working! Perhaps more iterations is needed
- Will it benefit if we start from random real samples?

Exp. 3 Vanilla method on MNIST (random kaiming init, init syn img from real samples, more it)

Settings:

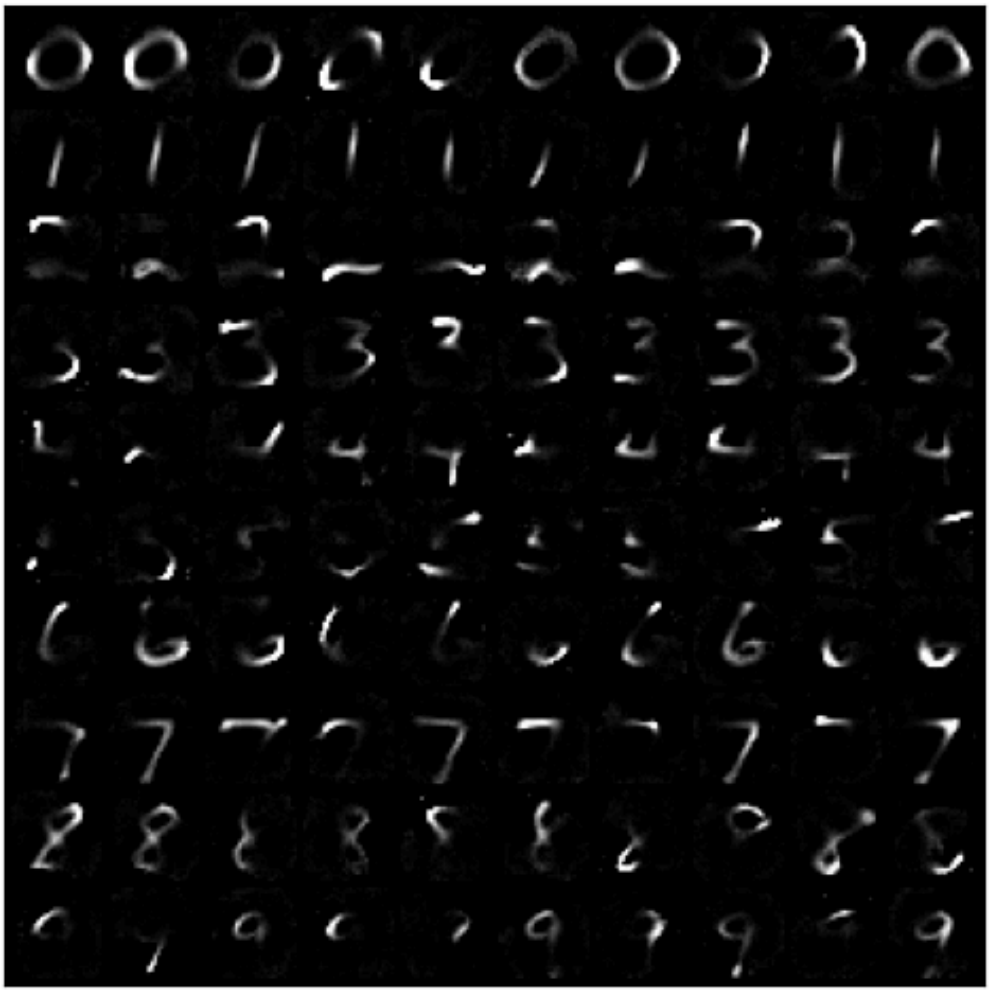
- **NUM_SAMPLES_PER_CLS = 10**
- **FROM_REAL_SAMPLES = True**
- **NUM_OPTIM_IT = 100000**
- **EVAL_INTERVAL = 1000**
- INIT_LR = 0.001
- MINIMUM_LR = 1e-5
- STEP_SIZE = 0.001
- **INIT_WEIGHTS_DISTR = "kaiming"**
- **FIX_INIT_WEIGHTS = False**
- NUM_SAMPLED_NETS_TRAIN = 16
- NUM_SAMPLED_NETS_EVAL = 4
- NUM_SAMPLES_PER_CLS = 10

Original randomly initialized synthetic dataset:

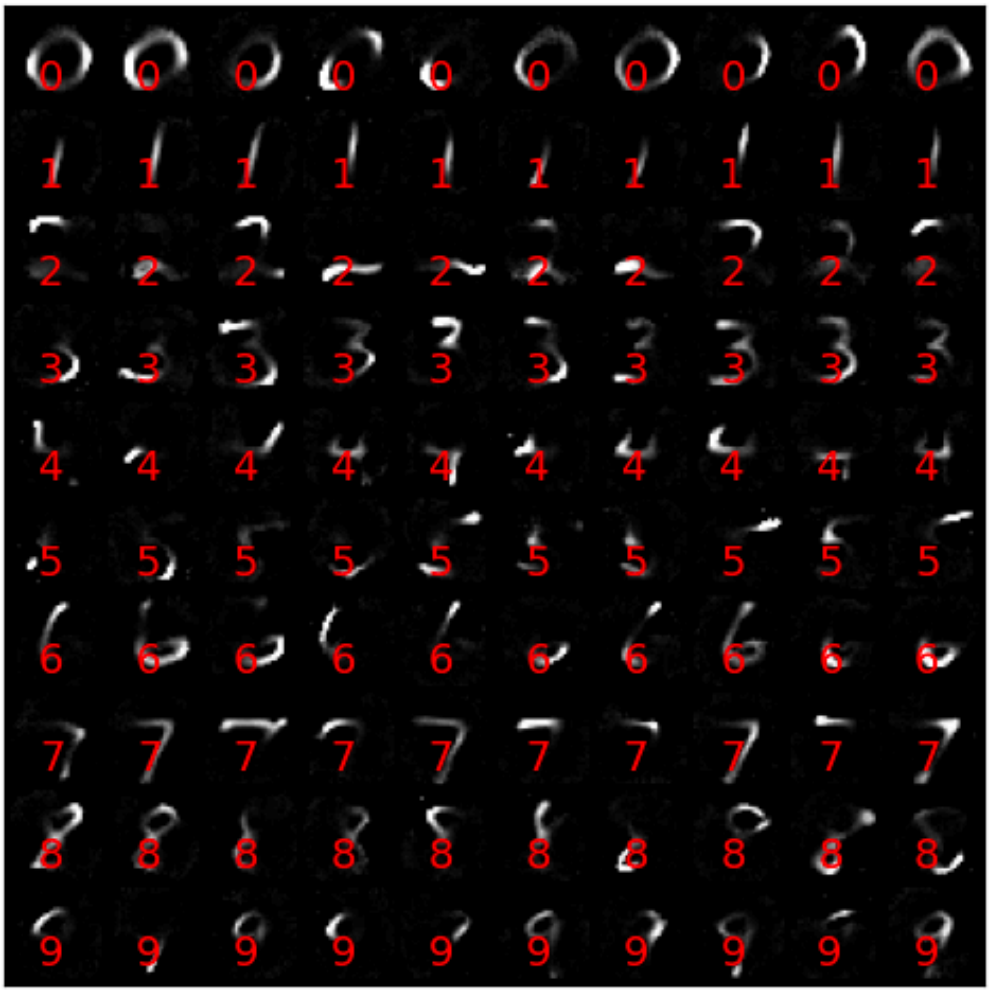


Synthetic dataset when training terminates (remote cluster session timed-out when it=31000...):

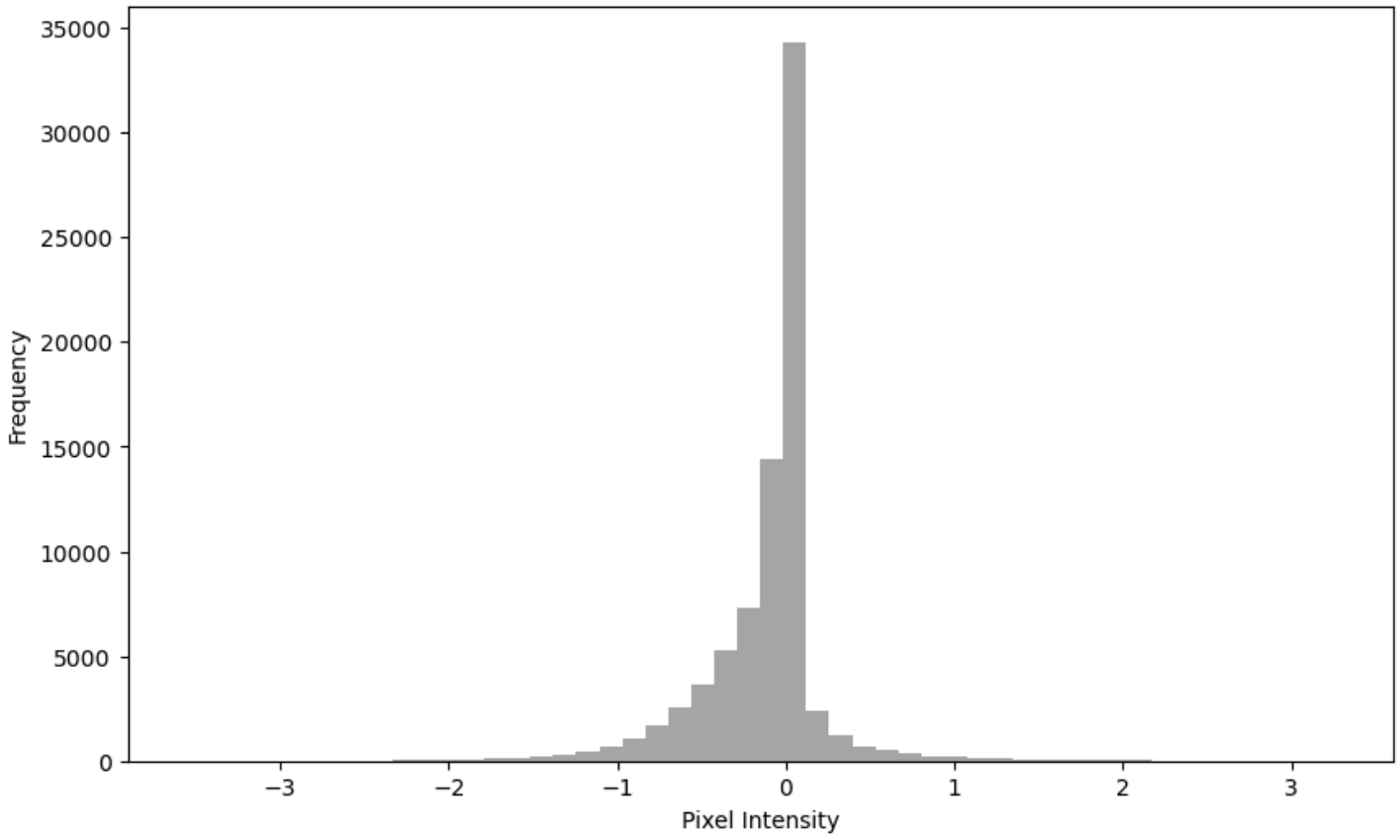
Synthetic dataset (optimization iteration 31000)



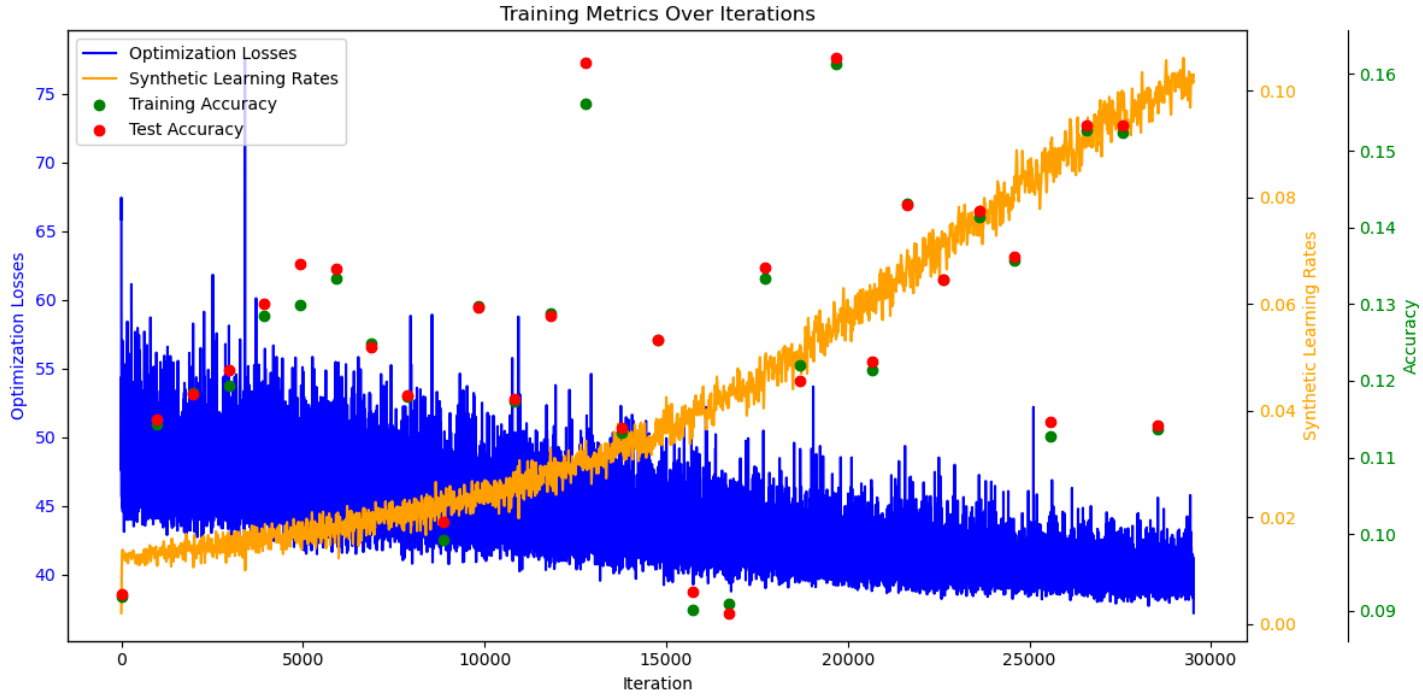
Synthetic dataset (optimization iteration 29520)



Pixel Value Distribution



Training curves:



Evaluating on training set and test sets (train the model for one epoch using synthetic data and learnt LR):

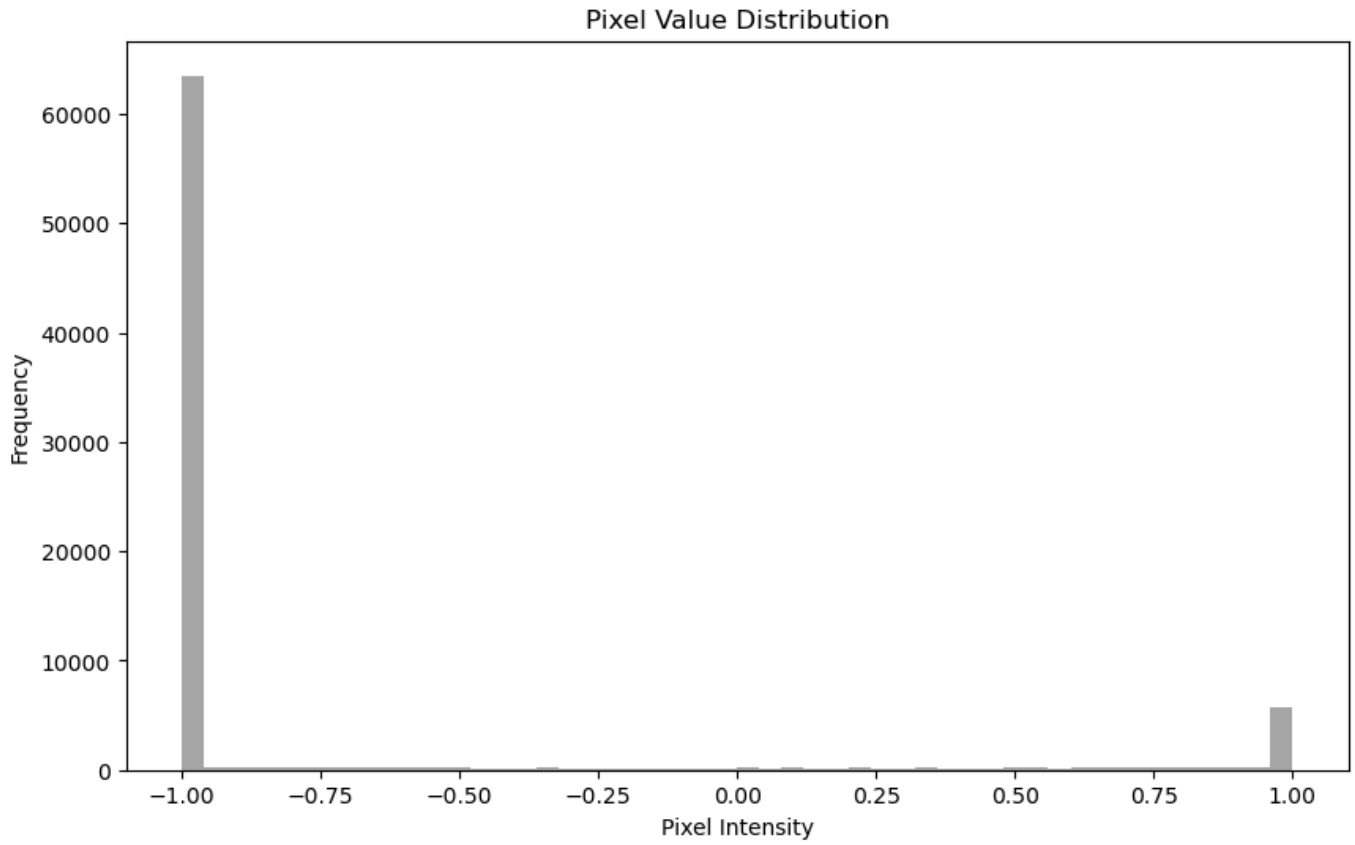
	Train set	Test set

Accuracy (before)	0.0910	0.0910
Accuracy (after)	**0.1550**	**0.1559**

Conclusions:

- Patterns are fading away (look at the pixel distribution, values very close to 0 are dominant). Why is that?

  This is what a initial synthetic dataset randomly sampled from real images will look like:



  Does that mean when pixels are 0, it won't be updated any more? **NO: the synthetic data will only be "frozen" if one of the images is all 0, making the optimization step nilpotent**

- Training isn't efficient

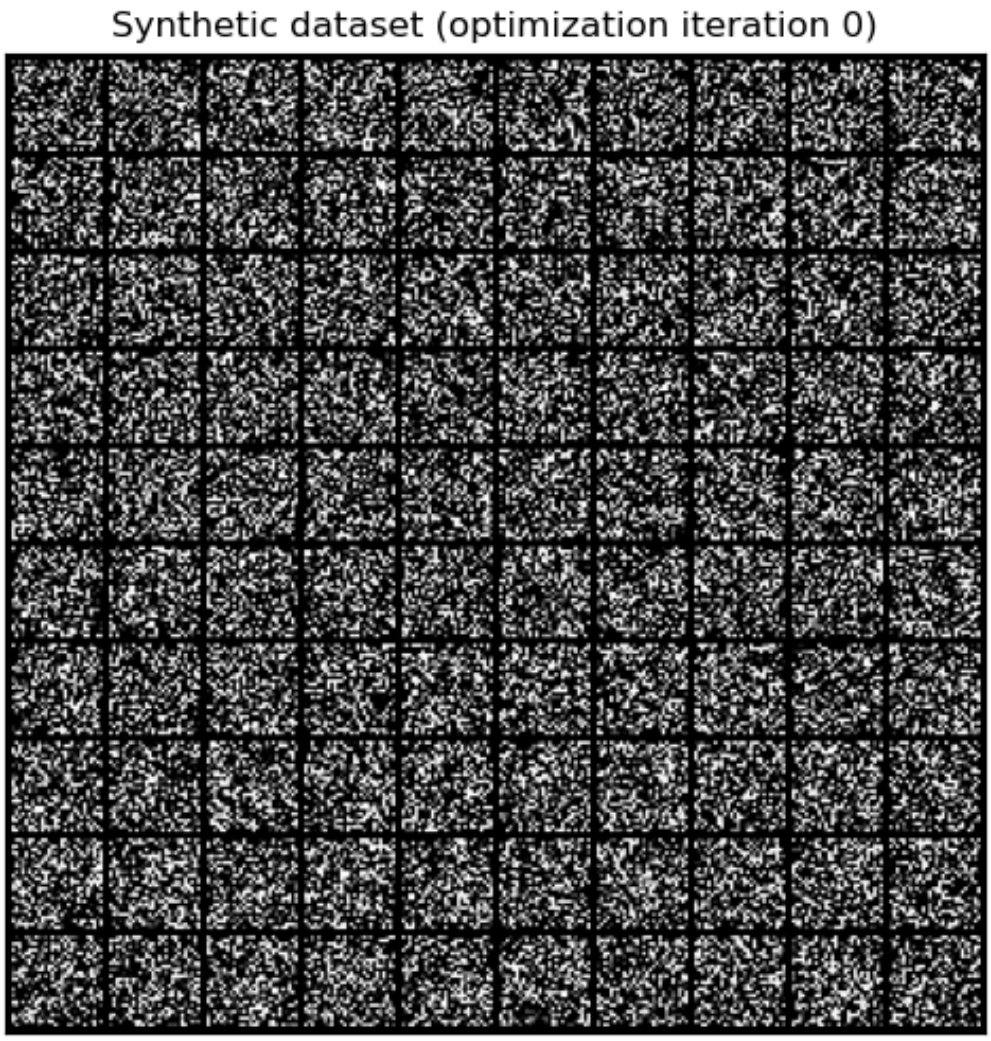
Exp. 4 Gradient matching on MNIST

Settings:

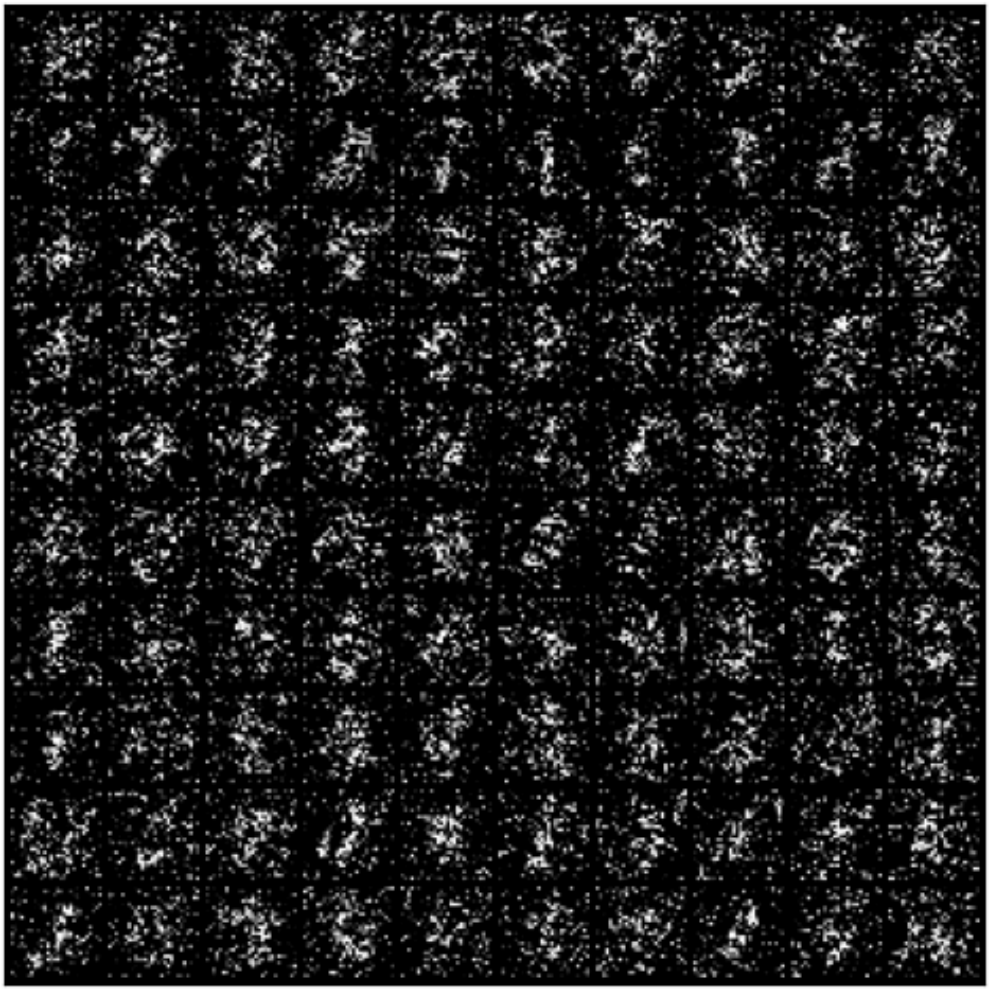
- model: LeNet (originally ResNet)
- NUM_OUTER_LOOPS = 1000
- EVAL_INTERVAL = 20
- EVAL_NUM_EPOCHS = 50
- NUM_SAMPLED_NETS_EVAL = 4
- **LR_DATA = 0.01** # original: 0.1

    - Note that in original paper (and code) there's no clamping on synthetic image pixels, setting this too large will make the pixel values explode really quickly to NaN
    - To avoid such explosion, I clamp the synthetic image pixels to range [-1, 1] after every iteration
- LR_NET = 0.01 # original: 0.01
- NUM_INNER_LOOPS = 10
- NUM_UPDT_STEPS_DATA = 1 # s_S
- NUM_UPDT_STEPS_NET = 50 # s_theta
- BATCH_SIZE_REAL = 256
- BATCH_SIZE_SYN = 256
- INIT_WEIGHTS_DISTR = "kaiming"
- FIX_INIT_WEIGHTS = False

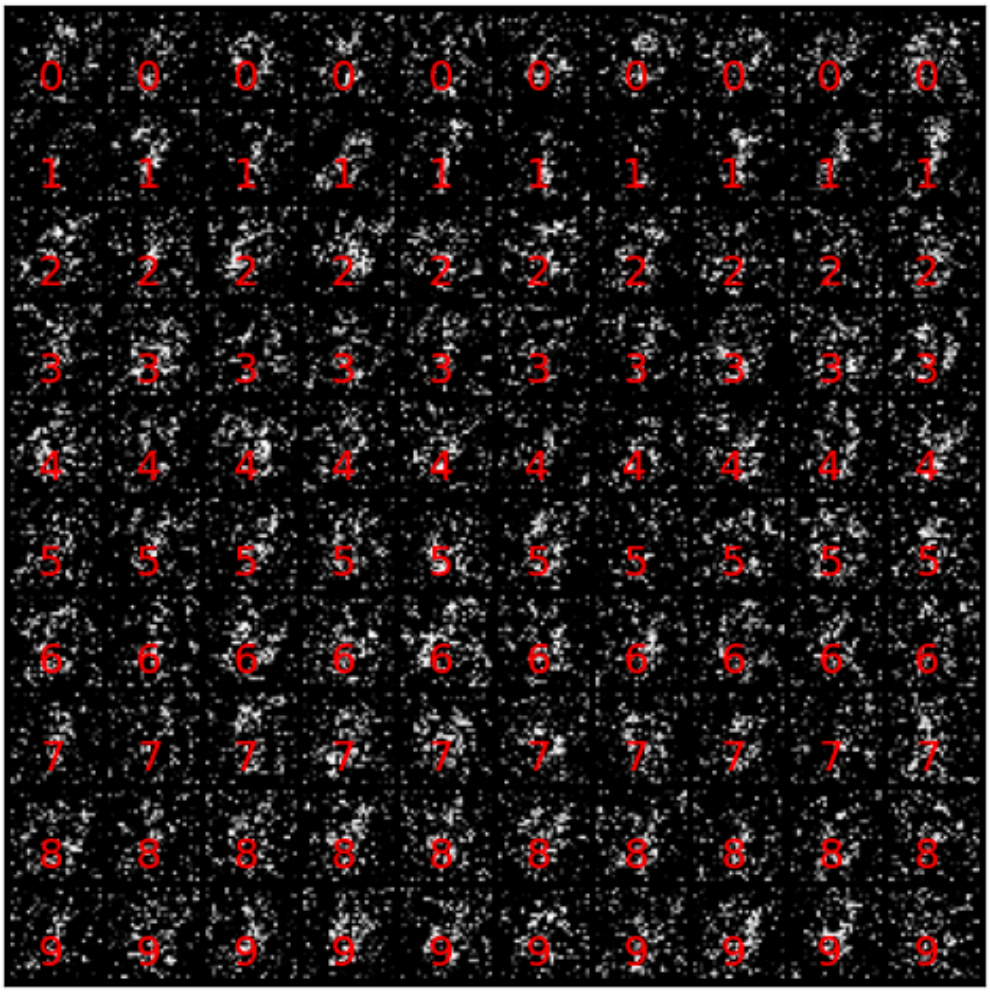
Original randomly initialized synthetic dataset:



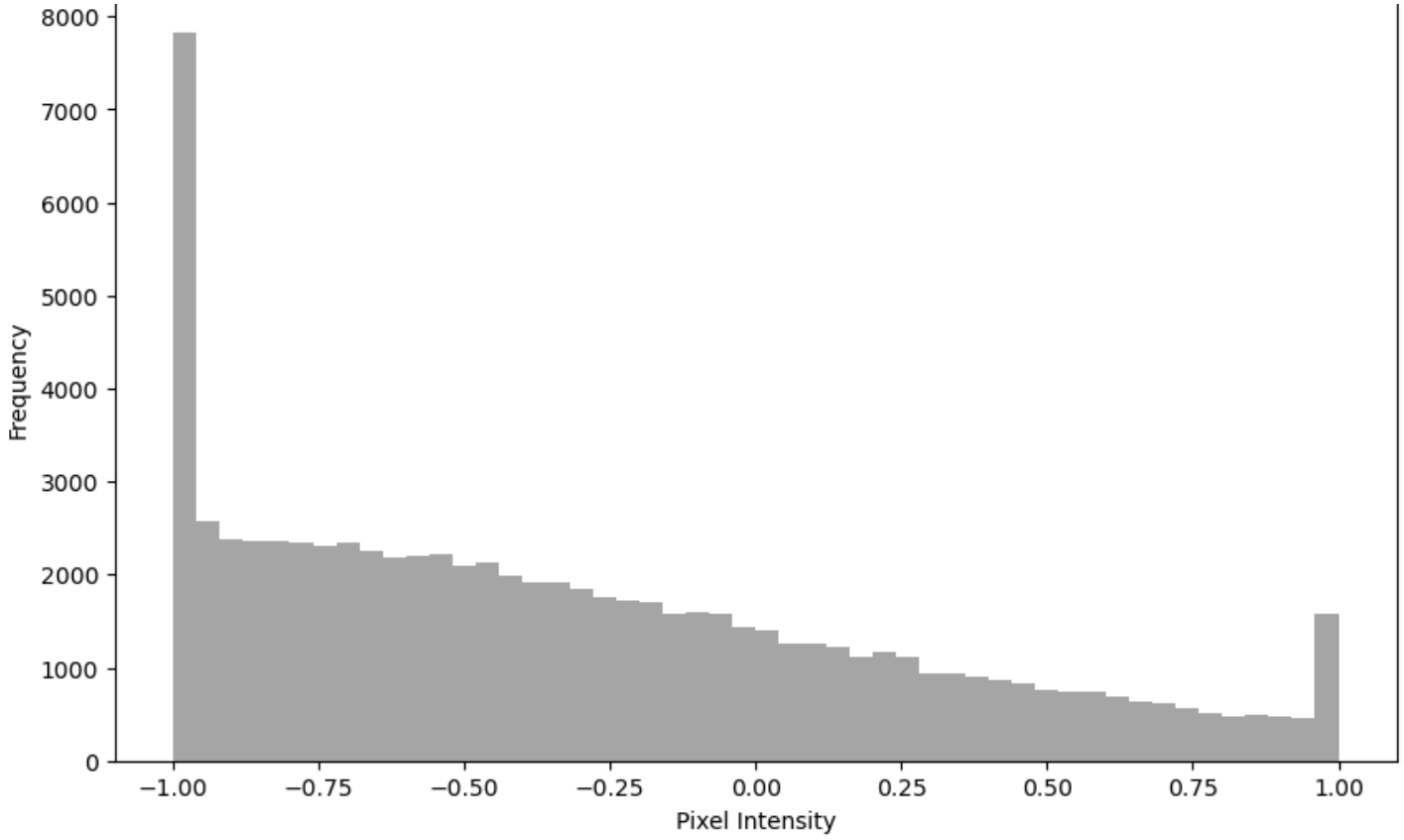
Synthetic dataset when training terminates:



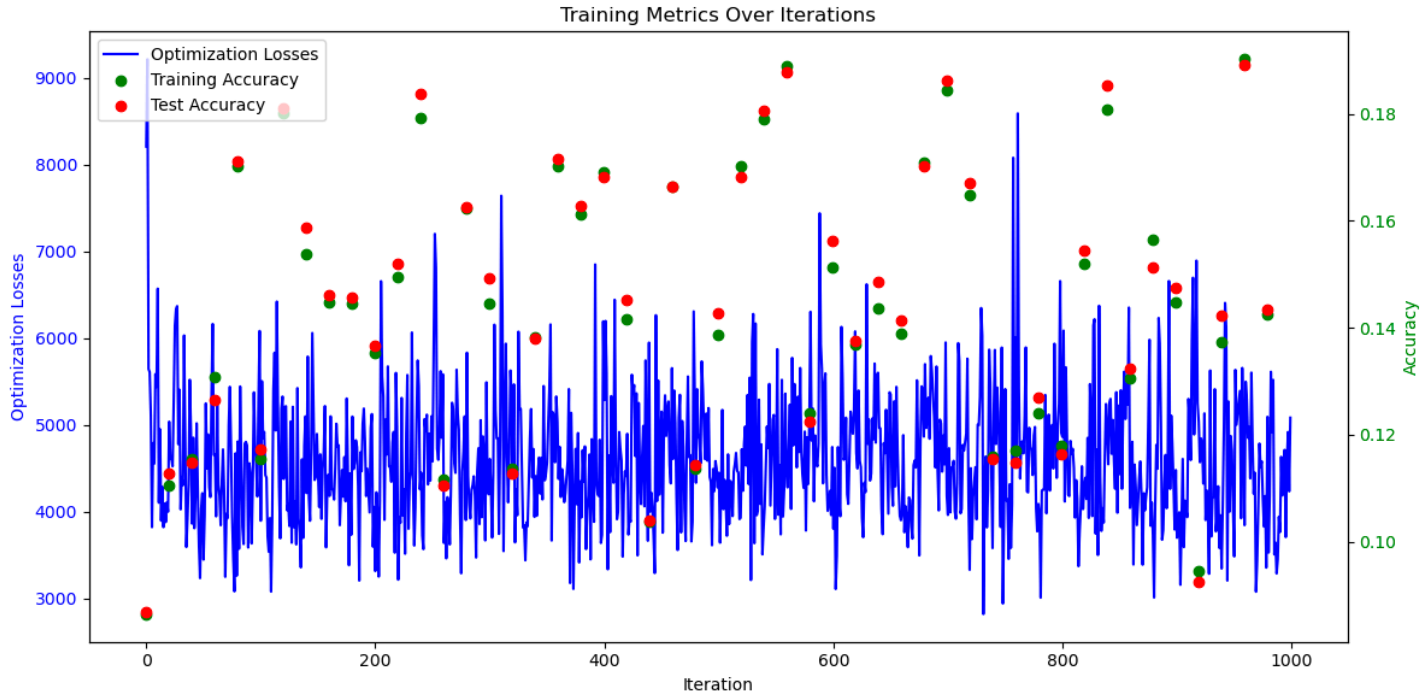
Synthetic dataset (optimization iteration 999)



Pixel Value Distribution



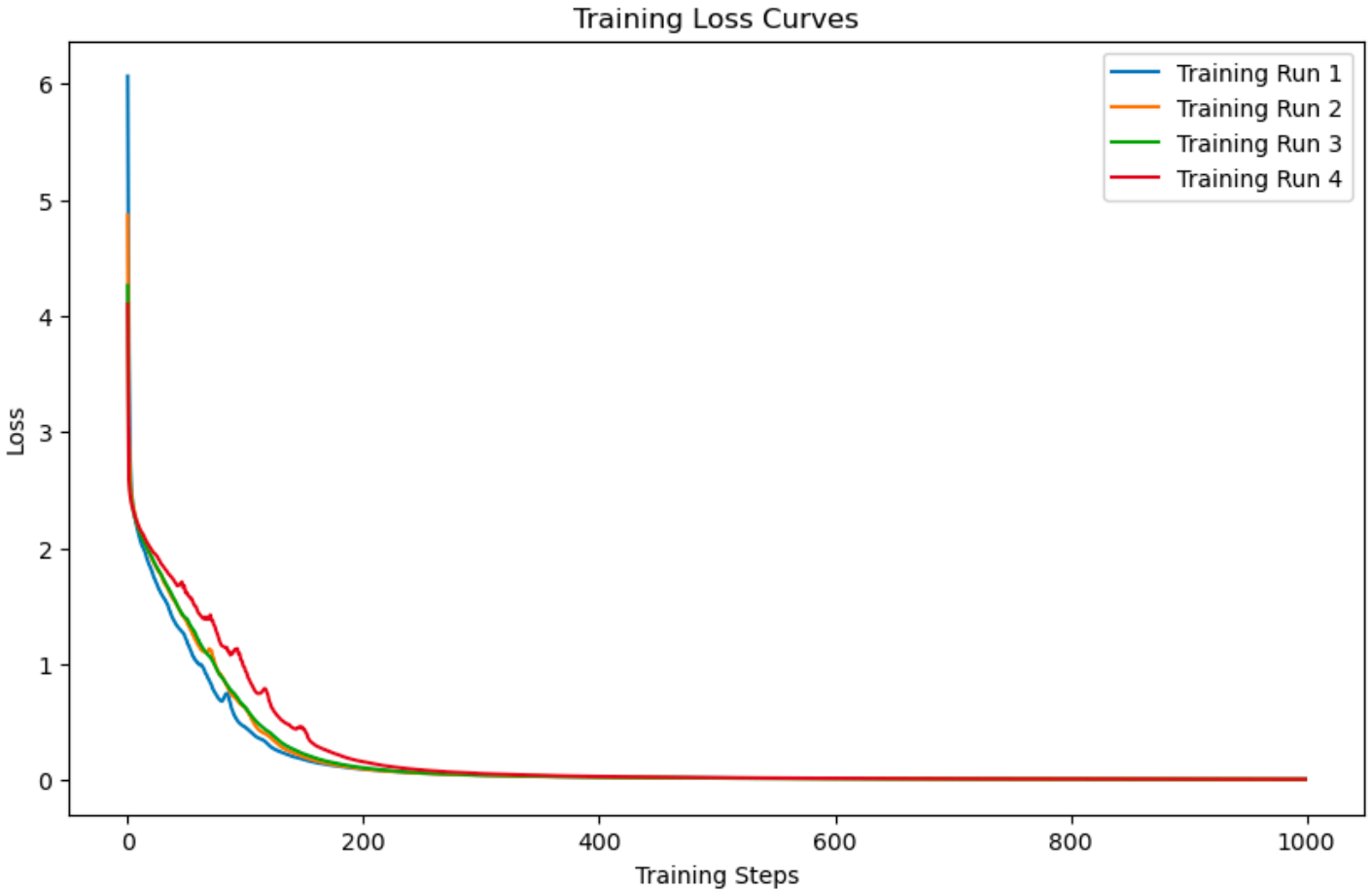
Training curves:



Evaluating on training set and test sets (train the model for as many as possible epochs using synthetic data and a good optimizer, see the training curves):

	Train set	Test set

Accuracy (before)	0.1016	0.1010
Accuracy (after)	**0.2342**	**0.2387**



Conclusions:

- This seems to be a promising method, with not bad training complexity, and visible / recognizable patterns in results
- However, how the original authors dealt with "pixel value explosion" is not clear
- I observed very rapid pixel value explosion, and from the very beginning stage of the training, the optimization seems to stop
    - Maybe it's because I used a relatively simple network (LeNet instead of ResNet)
- Need more experiments to get this working!