Adversarial Regularization for Convolution Filters

Dmitrii Gavrilev [1] **Evgeniy Garsiya** [1] **Lina Bashaeva** [1] **Farid Davletshin** [1] **Dmitriy Gilyov** [1]

Abstract

In this project we study the effect of adversarial regularization on a CNN's weights. Discriminators in GAN help to distinguish real images from generated with convolution filters, which are 2D matrices. If we train the model on the large amount of data, we will see that filters have some patterns. However, filters that are trained on a small dataset look random even after training. We collected good filters and trained a model on small data with adversarial techniques by applying a discriminator to regularize new filters. As a result we imposed patterns on kernels via discriminators.

Github repo: project link
Video presentation: video presentation link

1. Introduction

Nowadays, one of the most important and widespread problems in the deep learning area is image classification. Usually, this problem is solved by application of Convolutional Neural Networks (CNN). This networks consist of convolutional layers, which implement convolution operation on the image using kernels (2D weight matrices). When we consider filters of more or less big size (5x5, 7x7 and so on), we can observe some patterns in kernels of these filters. However, if we train our model on some small dataset, we can note that these kernels become random and we can't observe patterns anymore. This leads to the aspiration of saving kernel patterns while working with small datasets. Thereby, our motivation is to make kernels more meaningful. In order to solve this problem, we are going to implement Generative Adversarial Nets (GANs) functionality. As a fundamental functionality for this approach, we used GAN implementation from (Goodfellow et al., 2014). GAN models can help us to generate good filters with necessary patterns with adversarial approach.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Alexander Korotin <a.korotin@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

The main contributions of this report are as follows:

- Good (with patterns) filters were collected by training the CIFAR100 classification dataset with filters of size 7x7, 5x5 and random seeds of values 0 to 4.
- New dataset with small training set based on CIFAR10 was prepared
- Classifier was retrained on a new reduced dataset.
- Discriminator architecture was developed and implemented.
- Full model (classifier with discriminator for convolutional layer) was trained on a reduced dataset.
- Results of filter improvements were provided.

2. Related work

Nowadays, CNNs are state-of-the-art considering classification and other types of image processing problems. As described in (Alzubaidi et al., 2021) a commonly used type of CNN is similar to the multi-layer perceptron (MLP). It consists of numerous convolution layers preceding subsampling (pooling) layers, while the ending layers are Fully connected (FC) layers. There are three key benefits of the CNN: equivalent representations, sparse interactions, and parameter sharing. Unlike FC networks, shared weights and local connections in the CNN are employed to make full use of 2D input-data structures like image signals. This operation utilizes an extremely small number of parameters, which both simplifies the training process and speeds up the network. As mentioned in the article, for CNN models, over-fitting represents the central issue associated with obtaining well-behaved generalization. This can be solved by adding several layers like Dropout, Drop-Weights, Data Augmentation and Batch Normalization. Moreover, Batch Normalization helps to solve other problems besides overfitting. It prevents the problem of vanishing gradients from arising and significantly reduces the time required for network convergence.

CNN models are a fast growing segment of Deep Learning. There are a lot of new CNN models and publications every year. For example, one of the recent state-of-the-art

articles is (Agrawal et al., 2022) where authors proposed a novel technique to constrain parameters in CNN based on symmetric filters. They investigated the impact on accuracy for CIFAR-10 and CIFAR-100 datasets when varying the combinations and levels of symmetricity in the diverse basic blocks of NN models. Their models offer effective generalization and a structured elimination of redundancy in parameters.

In order to become familiar with GANs structures, we explored several articles about this topic. In (Goodfellow et al., 2014) authors provided us with detailed description of a framework for the work of GANs algorithm, where both generator (G) and discriminator (D) models are implemented as multilayer perceptron. GANs consist of two separated models Generator (G) and Discriminator (D), which have a structure of multilayer perceptrons. The adversarial modeling framework is most straightforward to apply when the models are both multilayer perceptrons. The training process takes place simultaneously for both models. The aim of G is to generate a sample based on captured data distribution, while D is trying to find the probability that a sample came from the training data rather than G. GANs training process works in such way, that G is trying to generate such a sample that the probability for D to make a mistake will be maximal. As a result, we are facing a two-player minimax game. As a result, authors provide an algorithm for minibatch stochastic gradient descent training of generative adversarial nets and a set of experiments on MNIST, TFD and CIFAR-10 datasets. They chose the following architecture for their experiments: the generator nets used a mixture of rectifier linear activations and sigmoid activations, while the discriminator net used maxout activations; dropout was applied in training the discriminator net. Also, the noise as the input was used to only the bottommost layer of the generator network. Probability of the test set data was estimated by fitting a Gaussian Parzen window to the samples generated with G and reporting the log-likelihood under this distribution. As a main disadvantage of this approach, they mentioned that G must not be trained too much without updating D, in order to avoid "the Helvetica scenario". On the other hand, there are several advantages. The advantages are, firstly, that Markov chains are never needed, only backprop is used to obtain gradients, no inference is needed during learning, and a wide variety of functions can be incorporated into the model. Secondly, they can represent very sharp, even degenerate distributions, while methods based on Markov chains require that the distribution be somewhat blurry in order for the chains to be able to mix between modes.

Considering our particular problem, we are going to use GANs with limited data. This approach is fully described in (Tseng et al., 2021), where authors propose a regularization approach for training robust GAN models on limited data. They theoretically show a connection between the regularized loss and an f-divergence called LeCam-divergence, which was found to be more robust under limited training data. Moreover, based on extensive experiments on several benchmark datasets it was demonstrated that the proposed regularization scheme improves the generalization performance and stabilizes the learning dynamics of GAN models under limited training data, and complements the recent data augmentation methods.

Moreover, we used an approach from (Radford et al., 2015) where authors provide a type of CNN for unsupervised learning called deep convolutional generative adversarial networks (DCGANs). Generally, in this network perceptron was replaced by discriminator and the whole architecture pipeline can be described as follows: any pooling layers were replaced with strided convolutions (discriminator) and fractional-strided convolutions (generator); batchnorm in both the generator and the discriminator was used; fully connected hidden layers for deeper architectures was removed; ReLU activation was used in generator for all layers except for the output, which uses Tanh and LeakyReLU activation was used in the discriminator for all layers.

As far as we work with filters we need to regularize them. Here is an interesting article considering this topic: (Dang et al., 2022). The authors considered Bayesian inference, a tool which allows the transformation of a prior distribution over parameters of a machine learning model to a posterior distribution after observing training data. They propose a special method that estimates the source kernel distribution in an implicit form and allows to perform variational inference with the specific type of implicit priors, based on assumption, that within a specific domain the source kernel distribution can be efficiently approximated with convolutional kernels of models, that were trained on a small subset of problems from this domain. A content of the paper can be summarized in the following way: authors proposed deep weight prior - a framework that approximated the source kernel distribution and incorporated prior knowledge about the structure of convolutional filters into the prior distribution. They also proposed to use an implicit form of this prior. Then, researchers developed a method for variational inference with the proposed type of implicit priors. In experiments, they showed that variational inference with deep weight prior significantly improved classification performance upon a number of popular prior distributions in the case of limited training data. They also found that initialization of conventional convolution networks with samples from a deep weight prior led to faster convergence and better feature extraction without training i.e., using random weights.

3. Models and Algorithms

3.1. Models

We use a convolutional neural network with 5 hidden layers and 3 different sizes of kernels (7x7, 5x5, 3x3) for both classifying images and generating 'fake' filters for the discriminator, which architecture is depicted in Figure 2

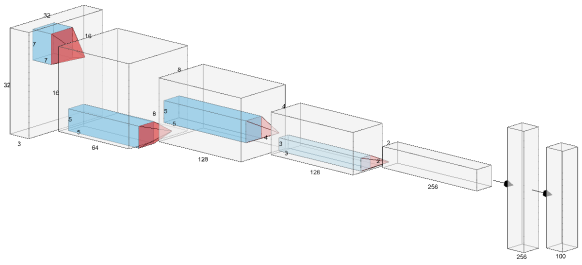

Figure 1. Classifier (Generator)

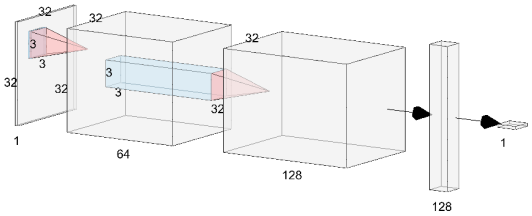

Figure 2. Discriminator

3.2. Algorithms

Algorithm 1 describes one epoch of training with adversarial regularization. The process of training is almost the same as the usual procedure of training classifiers. The only difference is that we add the regularization term, which is the generative loss, to train classifier. Besides, we perform one step of optimization for discriminators after updating classifier's weights.

More specifically, we iterate over the batches of images. First, we calculate the classification loss on the output of classifier and real image labels. Then, we form both the real and fake batches of kernels (the input to discriminator is denoted as x). As the real batches we take the kernels of

Algorithm 1 Epoch of training with regularization

for $imageBatch$ in $trainSet$ **do**
  $imagePred = classifier(imageBatch)$
  $loss =$ classifier loss on $(imageBatch, imagePred)$
  $realBatch =$ random batch of good kernels
  $fakeBatch =$ random batch of classifier kernels
  $loss+ = \lambda \cdot$ (generator loss on $fakeBatch$)
  Update weights of $classifier$ based on $loss$
  Train $discriminators$ on $realBatch$, $fakeBatch$
end for

models pretrained on large dataset. As for the fake batches, we form them from the current weights of classifier (weights are denoted as z). As convention, generative function G is also introduced and it is equal to the idenitity (the generator outputs its own weights):

$$G(z) = z \tag{1}$$

Next, we calculate the adversarial regularization loss as the generative loss on fake batches. Instead of minimizing $\log(1 - D(G(z)))$, we aim to maximize $\log D(G(z))$. This loss is accumulated with the global loss of classifier. Coefficient λ corresponds to the strength of regularization. The weights of classifier get updated based on the gradients from this global loss. And finally we pass both real and fake batches to discriminators and train them by maximizing $\log D(x)$. We additionally scale the discriminative loss by λ to match the scales of regularization and discriminator gradients.

4. Experiments and Results

4.1. Experiments

Datasets For our task, we used the CIFAR-10 and CIFAR-100 datasets, which are labeled subsets of the 80 million tiny images dataset.

The CIFAR-100 consists of 100 classes containing 600 images each, from which 500 images are train ones and the rest are test images. We took the whole train and test sets as our train/test, respectively. Our data preprocessing of CIFAR-100 involved only normalization with mean 0.5 and std 0.5 per each channel.

The CIFAR-10 dataset is similar to the CIFAR-100, but it contains 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. We took 100 random samples for the train set with uniform distribution between classes and for our test set we took the whole test batch, which has 1000 images from each class, selected randomly. For augmentation of the train set we used composition of random crops

and horizontal flips and finally normalized the images with the same mean and s.t.d. as for CIFAR-100, but for the test set we only applied normalization.

Main experiments All experiments have been conducted using Google Colab's computational resources. We used PyTorch as a framework for our project. If not stated otherwise, the optimizers were given the default hyperparameters.

First, we have trained five models with different random weight initialization on CIFAR-100 for 100 epochs. The optimizer was set as Adam with weight decay equal to 0.001. The batch size was equal to 128. The accuracy on test set was 0.55. We used these models to collect kernels with patterns and to form a dataset of 'real' kernels, some of which are shown in Figure 3. As can be seen, training on a large dataset results in kernels with patterns.

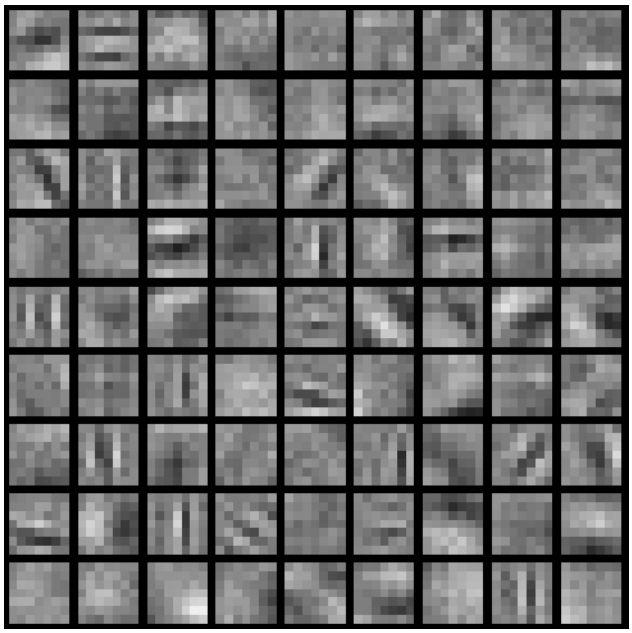

Figure 3. Visualization of the 7×7 kernel collection obtained by training on the full train set of CIFAR-100. The shape of kernels is 7×7

On the other hand, if we train a model with the same architecture on a small dataset (100 samples from CIFAR-10), then kernels might look random (See Figure 6). The number of epochs and batch size were fixed as 1000 and 16 respectively. The learning rate, weight decay and betas were set to 0.002, 0.001 and (0.5, 0.999) respectively.

Next, we trained the same model (and with the same optimizer hyperparameters) from scratch but with adversarial regularization. For this purpose, we used Algorithm 1. A discriminator was attached to the first convolutional layer (7×7). The plots for regularization and discriminative losses are presented in Figure 4. Besides, we show the dynamics of the discriminator's output $D(x)$ over the course of training, which is shown in Figure 5. The blue line is the probability conditioned on a kernel with pattern and the orange line corresponds to the probability conditioned on a generator's kernels. As can be seen from this plot, this dynamic looks quite natural. Both probabilities converge to 0.5. It means that by the end of training, the discriminator fails to distinguish between the pre-collected and new kernels.

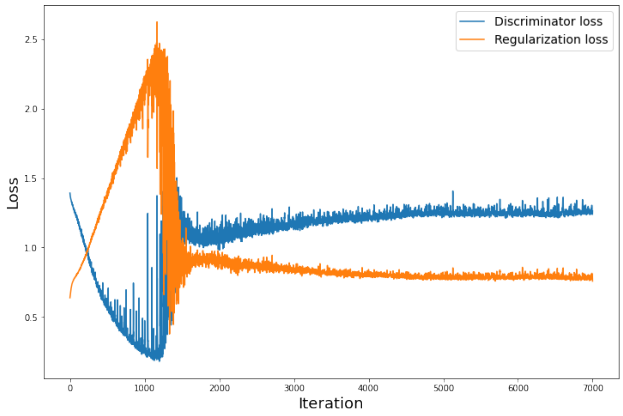

Figure 4. Regularization (orange) and discriminative (blue) losses with respect to iteration. The total number of iterations is the number of passes through batches of images dataset.

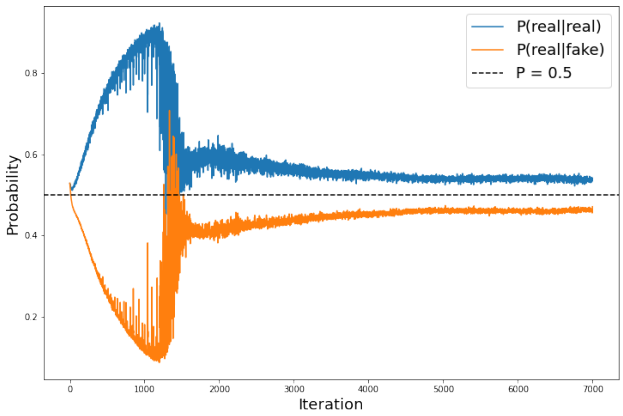

Figure 5. The dynamics of conditioned probabilities.

4.2. Results

For our model with adversarial regularization kernels we can see that patterns became structured (See Figure 10). This means that generator created kernels from the distribution where discriminator makes mistakes.

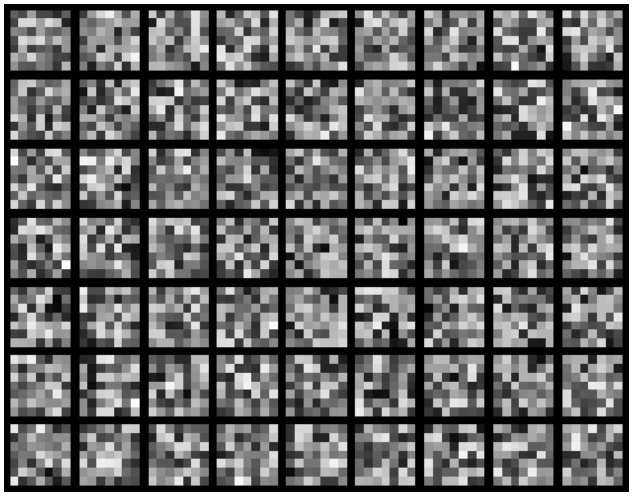

Figure 6. Visualization of the 7×7 kernel collection obtained by training on 100 samples from CIFAR-10. The shape of kernels is 7×7

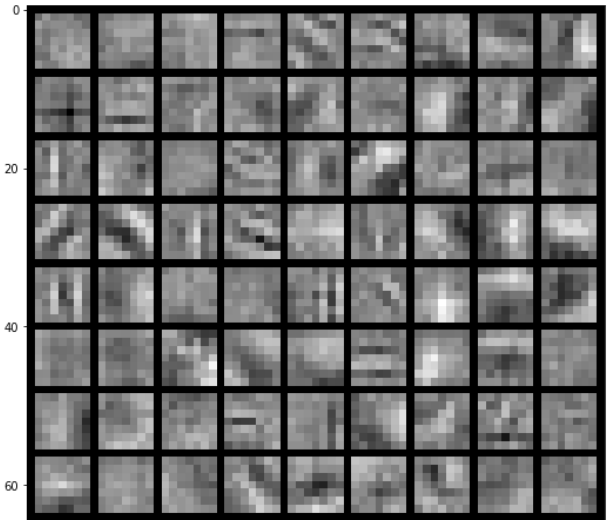

Figure 7. Visualization of the 7×7 kernel collection obtained by model using adversarial regularization technique. The shape of kernels is 7×7

Next, we calculated scores on test sample both for without (Figure 8) and with (Figure 9) regularization. As we can see accuracy behavior remained the same: fluctuating near 0.2 value.

5. Conclusion

We have succeeded in demonstrating the adversarial technique on model weights. We regularized a model trained on a small dataset such that its kernels have patterns. It has

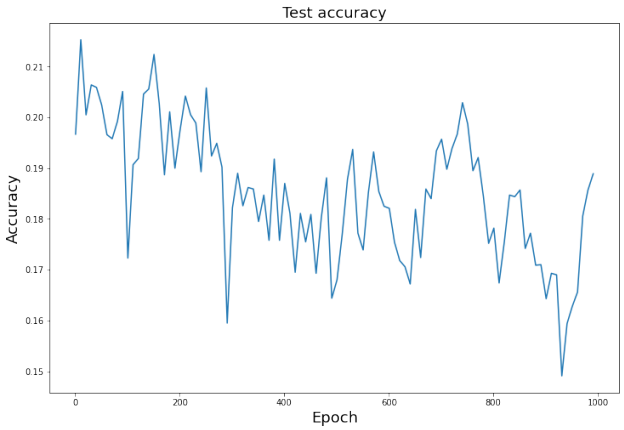

Figure 8. Test score for small dataset model

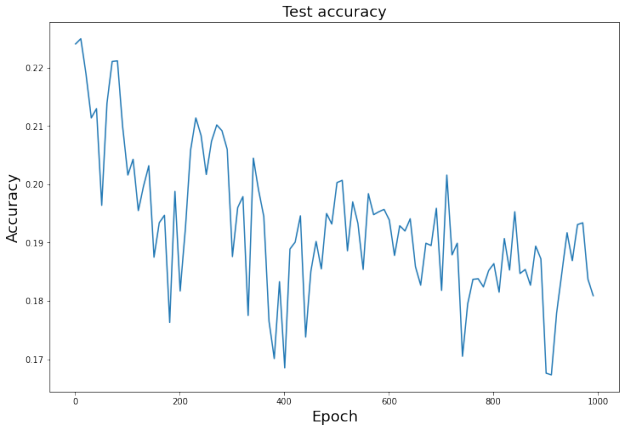

Figure 9. Test score for model with adversarial regularization technique

advantage of being more interpretable while it does not drop the accuracy. However, the accuracy is still rather bad and it generalizes poorly. Still we think the direction of adversarial regularization is promising.

References

Agrawal, H., T., S., and Nandy, S. K. Symmetric convolutional filters: A novel way to constrain parameters in cnn, 2022. URL https://arxiv.org/abs/2202.13099.

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., and Farhan, L. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1), March 2021.

doi: 10.1186/s40537-021-00444-8. URL https://doi.org/10.1186/s40537-021-00444-8.

Dang, H., Mecke, L., and Buschek, D. Ganslider: How users control generative models for images using multiple sliders with and without feedforward information. 2022. doi: 10.48550/ARXIV.2202.00965. URL https://arxiv.org/abs/2202.00965.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015. URL https://arxiv.org/abs/1511.06434.

Tseng, H.-Y., Jiang, L., Liu, C., Yang, M.-H., and Yang, W. Regularizing generative adversarial networks under limited data, 2021. URL https://arxiv.org/abs/2104.03310.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Dmitrii Gavrilev

- Coding the main algorithm
- Training models
- Preparing Experiments and Algorithm sections
- Designing architectures
- Preparing datasets

Farid Davletshin

- Training models
- Reviewing papers
- Writing report
- Making experiments
- Working with datasets

Evgeniy Garsiya

- Reviewing literate on the topic (3 papers)
- Experimenting with regularization coefficients
- Preparing the Presentation and Video
- Preparing the Section 1,2 of this report

Dmitriy Gilyov

- Training the large model to prepare the dataset of filters
- Implementing architecture extension ideas
- Conducting experiments with the new architecture of the algorithm
- Conducting experiments with the regularization coefficients
- Preparing the models section of the report

Lina Bashaeva

- Training model to prepare the dataset of filters
- Reviewing literate on the topic (2 papers)
- Experimenting with regularization coefficients
- Writing a review on readen papers and dataset description

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: Some bits of PyTorch's tutorial on GANs helped a lot.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: The repository contains code for downloading the dataset.

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   - [x] Yes.
   - [ ] No.
   - [x] Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   - [x] Yes.
   - [ ] No.
   - [ ] Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    - [x] Yes.
    - [ ] No.
    - [ ] Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    - [ ] Yes.
    - [x] No.
    - [ ] Not applicable.

    Students' comment: The only metric we used is accuracy which is quite common and does not neet to be reintroduced.

12. Clearly defined error bars are included in the report.

    - [ ] Yes.

- ☐ No.
- ☑ Not applicable.

Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - ☑ Yes.
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: Pytorch Framework and Google Colab VMs are mentioned.

C. Appendix

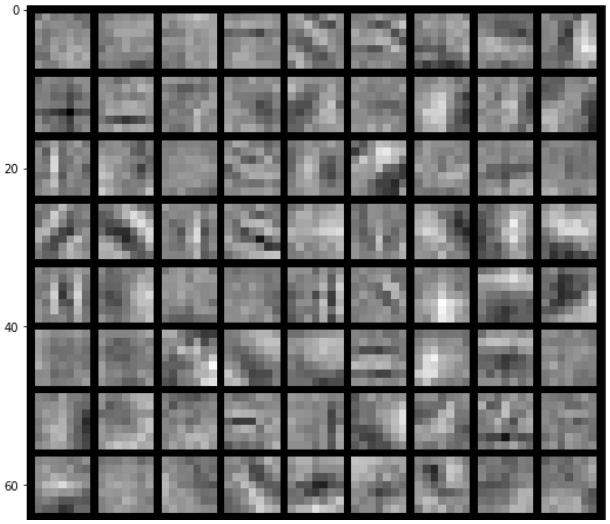

Figure 10. Visualization of the 7×7 kernel collection obtained by model using adversarial regularization technique. The shape of kernels is 7×7