École Polytechnique - Kyung Hee University
Dual Master Degree Program

Inner Workings of a denoising network

M1 Stage

Hangyeol Kang

Host organization: IDS (Telecom Paris)
Supervisor: Saïd Ladjal
Duration: 2021-05-01   2021-08-31

Acknowledgement

First of all, I would like to thank Professor Saïd Ladjal for giving me the opportunity to conduct my research in IDS, guiding me the right direction in research and all the support during the internship. Thank you for always lending your time for discussions and helping me for my experimental works. I really appreciate that you have always made time when I asked for a meeting even though you were very busy. It was my first time research in computer science lab and I was anxious that I would have many inadequacies. But thanks to your support, I could finish my internship well. From this internship, I could learn so many things, and I could think about what I lacked and what I need in the future. I really appreciate again for giving me the opportunity to experience this precious experience and your support.

Abstract

How can we interpret and explain the decision of deep neural networks? This is not fully answered question to this day. In this report, we tried to understand the inner workings of denoising neural network by tracing training points which are most influential on the model's decision. To track the points we applied two method; attribution method and influential function method. We executed some experiments of these methods on mnist classifier rather than directly applying them on FFDNet. From the experiments, we could know that the explainable points from the methods are stable and explainable. However, when we applied the methods on FFDNet, we couldn't achieve a satisfactory results. It seems that further experiments on applying the methods on FFDNet are needed.

Contents

Chapter 1
Introduction

Over the past decade, AI and especially Deep Neural Networks (DNNs) has received enormous attention across many areas of society and this AI techniques have been successfully applied in many fields; they run in our pockets on our cell phones[5], in cars to help avoid car accidents[8], in banks to manage our investments[3] and evaluate loans[13], in hospitals to help doctors diagnoses disease symptoms[11], at law enforcement agencies to help recover evidence from videos and images to help law enforcement[6].

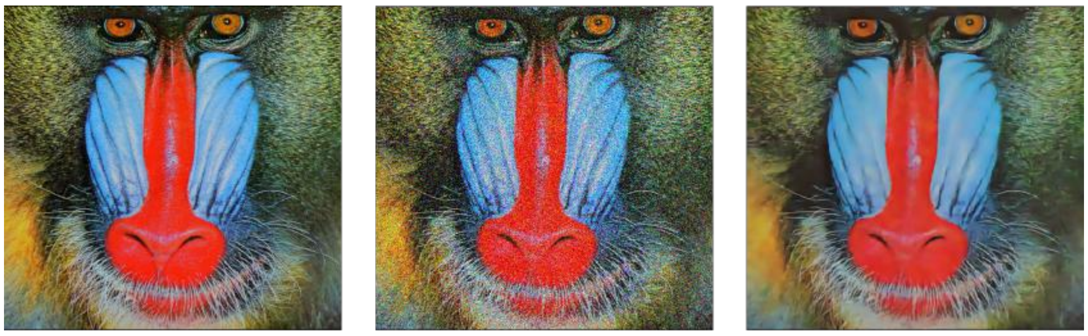
However, the techniques are complex and the results are not easy to understand, interpret and explain. This poses a serious problem in a number of application areas, especially in regulated or health-related industries like banking or hospital; when a major financial decision must be made, when a medical treatment is to be assigned, it is understandable that we want AI to suggest or recommend a course of action with reasonable evidence, rather than to merely prescribe one. The use of present-day DNNs leaves an important question unanswered: how can one who will be held accountable for a decision trust a DNN's recommendation[2, 16]?

There were also attempts to explicitly build in explainability into the architecture of ML algorithms[2]: a fast algorithm based on special tree structures (GAMs)[10, 15, 17], neural interaction transparency to disentangle shared-learning across different interactions via a special neural network structure[14].

The aim of this project is to try to understand the innerworkings of a denoising neural network. Image denoising being a low-level task, we believe that applying explainability methods to a denoising network can shed light on the underlying distribution of images and may pave the way to a fully understandable and non-deep denoising (or restoration) method.

While having a complete human-understandable explanation of how a deep network works is out of reach, for this particular project we will restrict ourselves to some attainable goals or even a consequent subset of them in the context of image denoising. Could we trace the behavior of the network to a particular part of the training set? And if so, we would be able to answer questions as 'Here the network denoised based on its experience with these images'. Such those answers start to look like an explanation.

We tried to track explainable training points of FFDNet[18], which is a denoising neural network with succesful performance in recent years. We used two methods to track those points: A method using influence functions introduced in Koh et



(a) HR image (b) Noisy image (c) Output image

Figure 1.1: Result of FFDNet what I used for our experiment (a) High resolution image from Set14. (b) Noisy image made from HR image and noise level, $\sigma = 50$. This image is the input of FFDNet. (c) The output of FFDNet.

al.[9] and our method using attribution techniques. In this report, we will call these two methods 'influence function method' and 'attribution method'. They[9] used hessian matrix to track influence points on the model's decision. Meanwhile, we used integrated gradients to track what parts of the database participated the most to the weights of network. Since then, we combined the activations of the network with the weight's attribution to point out what points in the database served as examples to achieve the denoising for each part of the denoised image.

I executed several experiments applying two methods to mnist classifiier first, a network much smaller than FFDNet, rather than applying the methods directly to FFDNet:

1. Whether the integrated gradients can point out what training points participated the most to the decision of particular neurons in network.
2. Whether we can track explainable images of test images by using both methods?
3. Stability and explainability test for both methods. Here, we measured the stability by counting how many identical explainable images we obtained when we ran same code several times, and we measured explainability by the change of loss function of each test image and the change of model's decision when we manipulated (changing labels or removing explainable points) training data and trained the network with that database.

We then applied those two methods to FFDNet and tried to track the expandable images of FFDNet. All the code for our experiments is available at https://github.com/hangyeol013/Stage_Telecom.

Chapter 2
Fundamental

In this chapter, we briefly review and discuss the major relevant concepts to this work: Gradient descent, denoising neural network, FFDNet (Fast and flexible denoising convolutional neural network) and influence functions.

2.1 Gradient Descent

In recent years, deep learning has attracted much attention as it has shown high performance in many fields like computer vision, speech recognition, natural language processing and medical image analysis. Deep learning is a type of machine learning algorithm that extract high level features from raw input data using many layers. Most deep learning is based on artificial neural network (ANN), which is literally a neural network created artificially. This ANN becomes a network that outputs more sophisticated features from input data through 'learning'.

Then, how is ANNs learned? First, the network receives input data and desired output data as input of the network and compares the actual output obtained by passing the input data through the network with the desired output. The difference between these two values is reduced by an algorithm called 'back propagation', in which the neurons of the network are updated. In addition, the method or algorithm for updating parameters in the network is called optimizer.

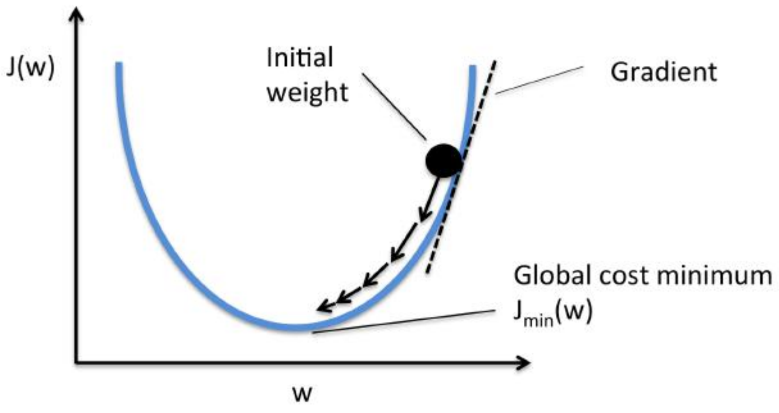


Figure 2.1: Gradient descent. x-axis means weight in network and y-axis means cost function of the network with respect to the weight. This shows how the gradient is updated.

There are many different optimizer algorithms in modern times, but the most basic optimizer is the gradient descent. This gradient descent updates parameters using the gradient of parameters for errors (the difference between desired output and network output, also called loss function). It calculates the gradient at the current point, updates the parameters by descending along the direction of the gradient, and finds the local minimum of the loss function. Through this process, the network is trained as a network that produces optimal output.

2.2 FFDNet

In general, image denoising methods can be grouped into two major categories, model-based methods and discriminative learning based ones. Model-based methods such as BM3D[4] and WNNM[7] are flexible in handling denoising problems with various noise levels, but they suffer from several drawbacks. Moreover, model-based methods usually employ hand-crafted image priors, which may not be strong enough to characterize complex image structures.

As an alternative, discriminative denoising methods aim to learn the underlying image prior and fast inference from a training set of degraded and ground-truth image pairs. However, existing discriminative denoising methods are limited in flexibility, and the learned model is usually tailored to a specific noise level. Besides, all the existing discriminative learning based methods lack flexibility to deal with spatially variant noise.

To overcome the drawbacks of existing CNN based denoising methods, a fast and flexible denoising convolutional neural network (FFDNet) was presented by Kai et al.[18]. By taking a tunable noise level map as input, a single FFDNet is able to deal with noise on different levels, as well as spatially variant noise.

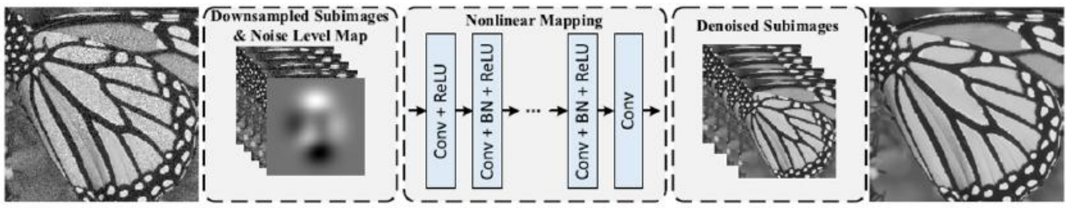


Figure 2.2: The architecture of FFDNet for image denoising. The network receives noisy input image, passes through the layers in network and outputs a denoised image.

The architecture of FFDNet is illustrated in figure 2.2. The pre-processing layer takes the input image and downsamples the input image into 4 sub-images. Further a tunable noise level map M is concatenated with the downsampled sub-images. This data is fed into 15-layers (for grayscale image, 12 layers for color image) that contains 64 filters (for grayscale image, 96 filters for color image) with a 3x3 kernel followed by a ReLU. The final output is reconstructed by the four denoised images. Since FFDNet operates on downsampled sub-images, the receptive field is larger as compared to a network with the same depth and operating at full at full resolution.

2.3 Influence Functions

We want models that are not just high-performing but also explanatory. By understanding why a model does what it does, we can hope to improve the model and provide end-users with explanations of actions that impact them.

How can we explain where the model came from? Koh et al.[9] tackled this question by applying the method called Influence Functions (IF). The idea of this method is to trace a model's predictions through its learning algorithm and back to the training data, where the model parameters ultimately derive from.

They begin by studying the change in model parameters due to removing a training point z from the training set. However, retraining the model for each removed z is too expensive. Instead, influence functions can give us an efficient approximation. The idea is to compute the parameter change if z were upweighted by some small ϵ, giving us new parameters $\widehat{\theta}_{\varepsilon,z} \overset{\text{def}}{=} argmin_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n} L(z_i,\theta)+\epsilon L(z,\theta)$. Since removing a point z is the same as upweighting it by $\epsilon = -\frac{1}{n}$, we can linearly approximate the parameter change due to removing z without retraining the model by computing $\widehat{\theta}_{-z} - \widehat{\theta} \approx -\frac{1}{n}\pounds_{up,params}(z)$

To get the influence of upweighting z on the loss at a test point z_{test} we compute the following expression:

$$\begin{aligned}\pounds_{up,params}(z) &\overset{\text{def}}{=} \frac{dL(z_{test},\widehat{\theta}_{\epsilon,z})}{d\epsilon}\bigg|_{\epsilon=0} \\ &= \nabla_\theta L(z_{test},\widehat{\theta})^\top \frac{d\widehat{\theta}_{\epsilon,z}}{d\epsilon}\bigg|_{\epsilon=0} \\ &= -\nabla_\theta\, L(z_{test},\widehat{\theta})^\top H_{\widehat{\theta}}^{-1}\,\nabla_\theta\, L(z,\widehat{\theta})\end{aligned}$$

where $H_{\widehat{\theta}} \overset{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\widehat{\theta})$ is the Hessian and is positive definite (PD) by assumption.

Influence function reveals insights about how models rely on and extrapolate from the training data. We can see that when the values of influence functions of some training points are positively big, that means those training points are helpful because removing each point of those training points increases test loss. On the other hand, when the values are negatively big, that means those training points are harmful because removing each point of them decreases test loss. It means that the training set without those training points has lower loss. Lastly, when the values are close to zero, it means those training images have almost no influence to the test loss.

Chapter 3

Experiment

Our goal is to understand the inner working of denoising network (FFDNet) by tracking explanatory points of test images. We used two methods to track the explanatory points, influence function method introduced in Koh et al.[9] and attribution method we propose. Before applying these two methods on FFDNet, which is wide and deep network, we tested those methods on a MNIST classifier which is much smaller network than FFDNet.

3.1 MNIST

In this section, we will describe the mnist model, data set and the two methods we used for tracking explanatory points of test images.

3.1.1 Model

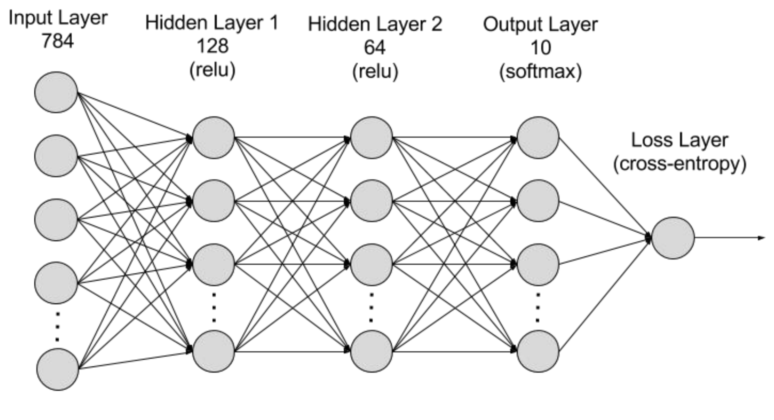


Figure 3.1: MNIST classifier network. This network receives input of size 784 and outputs the prediction on the class of input image.

We utilized a general MNIST classifier for our experiment. This model has two fully-connected hidden layers, each of them has 128 and 64 nodes. Input layer's size is 784, which is made in a row from a 28x28 size image. Between each layer, we have ReLU activation layers and at the end of the network, we have softmax (nn.LogSoftmax) layer.

3.1.2 Data set

We used built-in MNIST dataset in Pytorch (torchvision.datasets.MNIST). There are 60,000 images for training and 10,000 images for test. In our method, in order to track which images from the training data set were dominantly served as examples to achieve a result, We needed the index for each image in the data set. I edited the built-in module to output a dictionary for each image including image index, image values, and image label.

3.1.3 Experiment

In this section, we explain two methods we used to track explanatory images of test image. First, we will explain attribution method we propose, and will explain influence function method introduced in Koh et al.[9]. We will execute the methods and compare the results of both methods in the results and discussion section.

Method 1. Attribution method

Our goal is to track points which are most influential to the decision of the model on specific test image. We will achieve this goal by using the norm of gradients of nodes in the network with respect to training images and activations of layers obtained when we pass the specific test image to the network.

The gradient of node means the degree to which the loss changes with respect to parameters given training images. Therefore, we can reason from the gradients the extent to which training images affect on a particular node in the network. Meanwhile, the activation value of a particular node when we pass a test image through the layers in the trained network represents the response of the network to the given test image in that node. Therefore, those two values allow us to know the extent to which training images affect on a particular test image

The gradients norm

Before we use those two values (gradient norm and activations) to track the explanatory points of test images, we examined whether we can get enough explanatory images on each node in the network from the gradient norm.

We can check this from the norm of gradients of parameters between second hidden layer and output (softmax) layer of mnist network. This norm vector allows us to know the effect of given training images on each node in softmax layer. Softmax layer outputs the probability that an input image belongs to certain class (in mnist, it corresponds with classes from 0 to 9) from the features received from the previous layer. Therefore, first node in the softmax layer gives the possibility value that the network output is zero, second node has a possibility value that the network to output 1, and same for other nodes. As a result, we can examine if we can get enough explanatory images on each node of softmax layer in the network from the gradient of parameters.

Specific process to compute the norm of gradients is as follows:

Algorithm 1 Attribution method: computing gradients norm

```
Input:
∘ training data $z_{i=1}^{N} = (x, y)_{i=1}^{N}$
∘ epoch: E
∘ batch size: m
Output:
∘ Gradient norm vectors of training images

Algorithm:
1: Make an array (len(training images), len(norm of gradient of $l$th layer)), v
2: for $e = 1, ..., E$ do
3:     for $b = 1, ..., n$ do
4:         Sample mini-batch of $m$ images $(z^{(1)}, ..., z^{(m)})$
5:         Compute the loss function for the mini-batch images:
               $$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad (3.1)$$
6:         Compute the gradient of $l$th layer with respect to the loss:
               $$grad(\theta_l) = \bigtriangledown_{\theta_l} L(\theta) \quad (3.2)$$
7:         Compute the L2 norm of the gradient computed in the previous step:
               $$v = norm(\theta_l) = \|grad(\theta_l)\| \quad (3.3)$$
8:         Add the L2 norm vector into the place of array corresponding the index of
           training images from the mini-batch
9:     end for
10: end for
11: Save the v values into json file
```

Gradient norm is computed for every epoch and added to each position of an array corresponding the index of training images from mini-batch. We set batch size to 8 and epoch number to 20. This batch size and the number of epoch can affect on the result. Because if we set the batch size too big, the same gradient values will be spread over many images in the mini-batch, and this leads to inaccurate results. On the other hand, if the batch size is too small, the training process can be too slow. In the same way, the number of epoch has effect on the results. If we set the number of epoch big, the gradient is added more times and the results will be more accurate but the model can be overfitted and the training time increases. Setting batch size into 8 and epoch into 20 was enough to get stable results.

We will look at the results in results and discussion section.

The activation values

From the gradient norm vector, we could track explanatory points on the network. Furthermore, we used activation values to find out explanatory points on the decision of model with respect to a particular test image. Specific process is as follows:

Algorithm 2 Attribution method: computing influence values

```
Input:
∘ test samples $z_{i=1}^{m} = (x, y)_{i=1}^{m}$
∘ trained model, $h_\theta$
∘ gradient norm vector, v (see Algorithm 1
Output:
∘ influence values (s) on each test image

Algorithm:
1: Load trained model
2: Load gradient norm vector, v
3: for i=1, ..., m do
4:     Compute the values passed the $l$th activation layer:
           $act_{\theta_l^i} = h_{\theta_l}(x^{(i)})$    (3.4)
5:     Multiply the values of $l$th activation function and $l$th gradient norm vector:
           $s(\theta_l^i) = v_{\theta_l} \cdot act_{\theta_l^i}$    (3.5)
6:     Repeat steps 4,5 on every hidden layer and sum $s(\theta_l^i)$ for every hidden layers
7: end for
```

Depending on which layer is selected in the algorithm above, the influence values obtained can vary. When we compute influence values and track explanatory points using the gradient norm and activation values from all layers, the explanatory images were closer to test image than using the gradient norm and activation values from a particular layer. Therefore, we used gradients and activations from all hidden layers.

By listing the influence values in order of biggest to smallest, we can track the training images in order of most influencial on the prediction of specific test image. We will discuss this results in result section.

Method 2. influence function method

This method is proposed by Koh et al.[9]. This method uses hessian matrix to track the model's prediction from training data. For the computational efficiency of hessian matrix, they used stochastic estimation method[1] using hessian-vector products (HVPs) proposed by Pearlmutter et al.[12]. This method uniformly sample t points from the training data and recursively compute hessian matrix. They set t to be large enough such that hessian estimator stabilizes, and to reduce variance they repeat this procedure r times and average results. Since then, They obtained influence functions of each training points by multiplying the final hessian estimator and gradient vector of each training points.

The specific process is below: see Koh et al.[9] for more details.

Algorithm 3 influence function method

```
    Input:
    ∘ training images $z^N_{i=1} = (x, y)^N_{i=1}$
    ∘ test samples $z^M_{t,i=1} = (x_t, y_t)^M_{i=1}$
    ∘ trained model, $h_\theta$
    ∘ recursion numbers, R
    ∘ training points, P
    Output:
    ∘ influence functions of training images with respect to test images

    Algorithm:
 1: for i = 1, ..., M do
 2:     Compute loss function for the test image
 3:     Compute the gradient of loss given a test sample: $v = \bigtriangledown_\theta L(z^i_t, \hat{\theta})$
 4:     Set $s^0_{test} = \tilde{H}^{-1}_0 v = v$
 5:     for r = 1, ..., R do
 6:         for p = 1, ..., P do
 7:             Sample randomly a training point, $z^{(p)}$
 8:             Compute loss function for the training point:
 9:             Compute HVP (Hessian vector products):
                $$\begin{aligned} hvp &= \bigtriangledown^2_\theta L(z_\theta, \hat{\theta})\tilde{H}^{-1}_{j-1} v \qquad (\tilde{H}^{-1}_0 v = v) \\ &= \bigtriangledown^2_\theta L(\theta) v' \\ &= \frac{\bigtriangledown_\theta f(\theta + \tau v') - \bigtriangledown_\theta(\theta)}{\tau} \end{aligned}$$
10:             Calculate Hessian estimator($s_{test}$):
                $$\begin{aligned} s_{test} &= \tilde{H}^{-1}_j v \\ &= v + (I - \bigtriangledown^2_\theta L(z_{t_j}, \hat{}))\tilde{H}^{-1}_{j-1} v \\ &= v + \tilde{H}^{-1}_{j-1} v - \bigtriangledown^2_\theta L(z_\theta, \hat{\theta})\tilde{H}^{-1}_{j-1} v \\ \rightarrow \quad s^p_{test} &= v + s^{p-1}_{test} - hvp \end{aligned}$$
11:         end for
12:     end for
13:     for j = 1, ..., N do
14:         Sample a training image, $z^{(i)} = (x^{(i)}, y^{(i)})$
15:         Compute gradient vector, $\bigtriangledown_\theta L(z_i, \hat{\theta})$
16:         Compute influence functions, $\pounds_{up,loss}(z, z_{test}) = -s_{test} \cdot \bigtriangledown_\theta L(z_i, \hat{\theta})$
17:     end for
18: end for
```

By listing the influence function values obtained from above algorithm in order of biggest to smallest, we can list them in the order of helpful images to harmful images to the test image. We can know the meaning of 'helpful' and harmful' from the influence function equation, where the loss of the entire training image is subtracted from the loss of the training images without 'z' image. Big positive IF (influence function) value means that when the training set doesn't have 'z' image, they had bigger loss, which means 'z' image is a helpful image to predict that test image. On the other hand, big negative IF value means that when the training set doesn't contain 'z' image, they had smaller loss, which means 'z' image is a harmful image to predict that test image.

Like the previous method, from this method we could track explanatory points of test images. We will discuss the results in the results and discussion section.

3.2 FFDNet

In this section, we will describe the FFDNet model, data set and the two methods we used for tracking explanatory points of test images in FFDNet.

3.2.1 Model

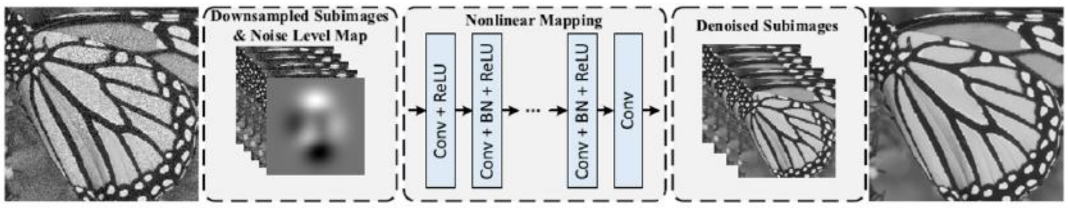


Figure 3.2: The architecture of FFDNet for image denoising. The network receives noisy input image, passes through the layers in network and outputs denoised image.

We used FFDNet[18] as a denoising neural network for our experiment. This network has a structure that passes downsampled input through the layers and then upsample features to get the output image at the last layer of the network. This network has different number of layers and channels depending on which image is passed through the network. If color image (3 channels) is passed through the network, the network has 12 layers and 96 channels each. On the other hand, if the image is gray scale image (1 channel), the network has 15 layers and 64 channels. We experimented with color image, the network we used has 12 layers and 96 channels. The kernel size is 3x3 and zero-padding is added on all layers so that the size of image is maintained. Between layers, batch normalization layers were added for regularization and ReLU layer was used as the activation layer.

3.2.2 Data set

We trained FFDNet with about 4000 training images, which contain natural images. We cropped the training image into patches for training instead of using the full images. Therefore, we trained FFDNet with about 200,000 patches. To train denoiser, we needed to add noise into high resolution image to make input image. We added random noise, sigma range from 0 to 75. To test the trained model, we used popular data sets (set5 and set14). When I applied the methods to track the explanatory points, I set the noise level , $\sigma = 25$.

3.2.3 Experiment

Here, we explain the two methods that we applied to FFDNet to track explanatory points. In the process of changing the model from mnist to FFDNet, it was difficult to apply those methods directly on FFDNet as we did on mnist. There were three major changes in the application of the methods.

1. Changed from classification model to regression model.
2. Changed from fully connected layers to convolutional layers.
3. The number of layers and channels has been increased.

First, with the first change, we tried to find the explanatory points for a certain area of test image instead for the entire area of test image. Therefore, we needed additional parameter which contain the information of point where we want to track. In the case of influence function method, we couldn't track explanatory points for a certain area of test image due to the nature of the method, but instead, we tried to track explanatory points for the entire area of the test image.

Due to the second change, we modified the process of computing the influence value. In mnist, we could multiply two vectors (gradient norm, activation), it was because mnist model contains only linear layers. However, in FFDNet, the network contains convolutional layers, we couldn't directly multiply gradient norm and activation value. If a point where we want to track the explanatory points is given, the shape of activation becomes (96,1,1). We could make the shape of gradient norm into (image numbers, 96) by summing 9 gradients in 3x3 kernel. This allows us to compute the influence value in FFDNEt in the same way we did in mnist.

As the third change, we needed to choose a layer to calculate the influence value. In the case of mnist model, because the network was small, we could compute influence value with values from all layers. However, in the case of FFDNet, because the network is deep and wide, it was difficult to compute the values from all layers. There were a computation time issue as well as a computational load issue. Therefore, we compared the results obtained from other layers and decided which layer to use.

Chapter 4
Results and Discussion

In this chapter, we will discuss about the results obtained by using the two methods introduced in the previous chapter. Before applying those methods on FFDNet, we applied those methods on a smaller network, mnist classifier. We will look at the images obtained by applying those methods to mnist classifier and some experiments first, followed by the image results obtained by applying those methods to FFDNet.

4.1 MNIST

In this section, we will discuss about the results obtained by applying two methods on mnist classifier; attribution method we proposed and influence function method introduced in Koh et al.[9]. We will look at three major results as below:

1. First, we will examine what training points participated the most to each node in last (softmax) layer of mnist network. In this process, we used gradient norm and from that value, we could track most influential training points on each node.
2. Second, we will examine what training points participated the most to certain test image. In the process of finding those explanatory points, we used two methods introduced before. In the case of influence function method, we could obtain helpful training points and harmful training points on certain test image.
3. Third, we will execute stability experiment and explainability experiment of those two methods. We measured the stability by counting how many identical explanatory images we could obtain when we run code several times. We also measured explainability by the change of loss of each test image when we manipulated training data and trained the network with that data set.

4.1.1 Explanatory images on Network

Here we will look for the explanatory images for each node in the last (softmax) layer of mnist network before we find the explanatory images for test image. We tracked the explanatory images on network nodes by computing the gradient norm of last layer. Later, this gradient norm will be also used to track the explanatory points for test image in our method. The results of explanatory images on each node are below:

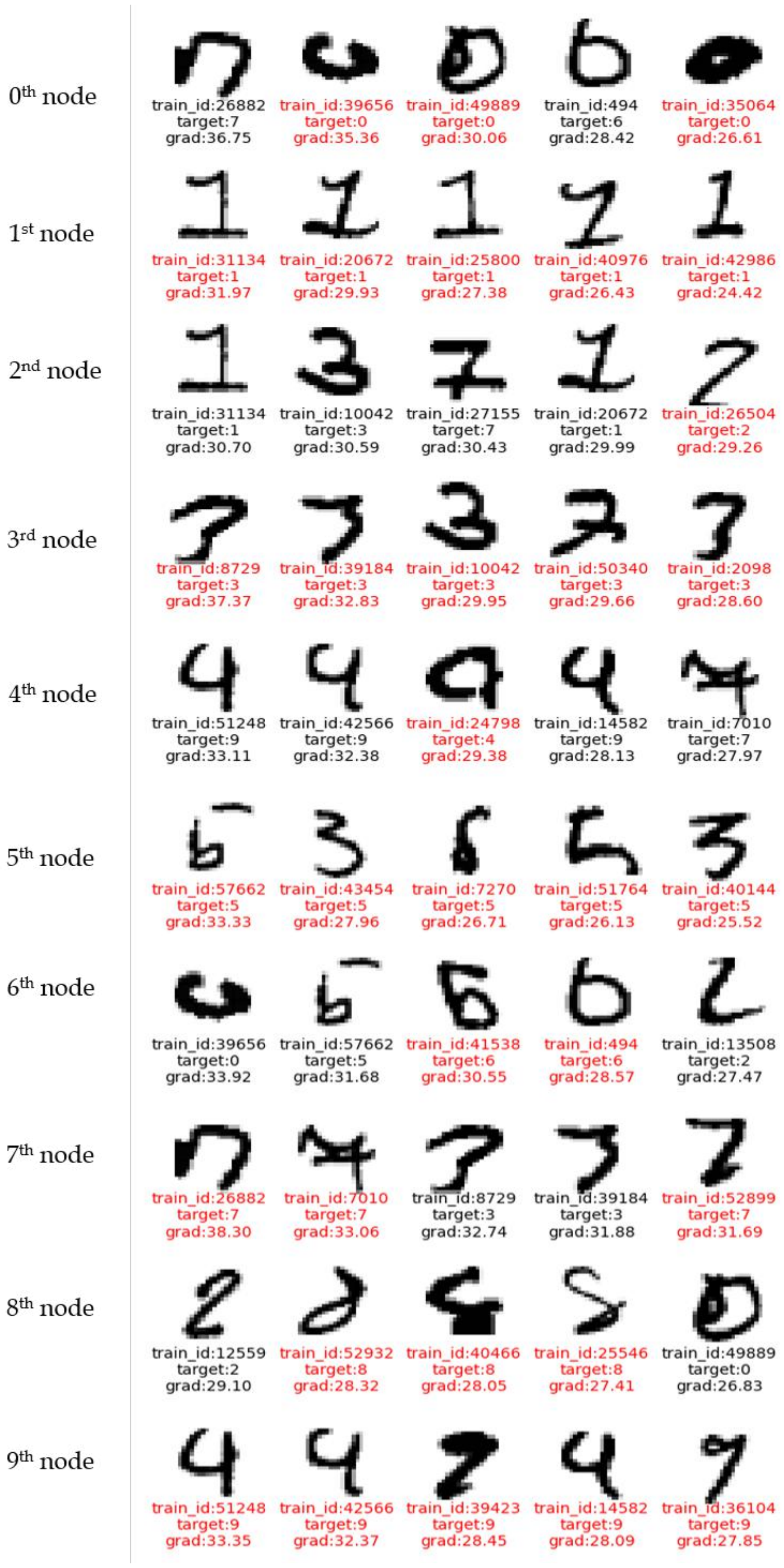

0th node
train_id:26882 target:7 grad:36.75
train_id:39656 target:0 grad:35.36
train_id:49889 target:0 grad:30.06
train_id:494 target:6 grad:28.42
train_id:35064 target:0 grad:26.61
1st node
train_id:31134 target:1 grad:31.97
train_id:20672 target:1 grad:29.93
train_id:25800 target:1 grad:27.38
train_id:40976 target:1 grad:26.43
train_id:42986 target:1 grad:24.42
2nd node
train_id:31134 target:1 grad:30.70
train_id:10042 target:3 grad:30.59
train_id:27155 target:7 grad:30.43
train_id:20672 target:1 grad:29.99
train_id:26504 target:2 grad:29.26
3rd node
train_id:8729 target:3 grad:37.37
train_id:39184 target:3 grad:32.83
train_id:10042 target:3 grad:29.95
train_id:50340 target:3 grad:29.66
train_id:2098 target:3 grad:28.60
4th node
train_id:51248 target:9 grad:33.11
train_id:42566 target:9 grad:32.38
train_id:24798 target:4 grad:29.38
train_id:14582 target:9 grad:28.13
train_id:7010 target:7 grad:27.97
5th node
train_id:57662 target:5 grad:33.33
train_id:43454 target:5 grad:27.96
train_id:7270 target:5 grad:26.71
train_id:51764 target:5 grad:26.13
train_id:40144 target:5 grad:25.52
6th node
train_id:39656 target:0 grad:33.92
train_id:57662 target:5 grad:31.68
train_id:41538 target:6 grad:30.55
train_id:494 target:6 grad:28.57
train_id:13508 target:2 grad:27.47
7th node
train_id:26882 target:7 grad:38.30
train_id:7010 target:7 grad:33.06
train_id:8729 target:3 grad:32.74
train_id:39184 target:3 grad:31.88
train_id:52899 target:7 grad:31.69
8th node
train_id:12559 target:2 grad:29.10
train_id:52932 target:8 grad:28.32
train_id:40466 target:8 grad:28.05
train_id:25546 target:8 grad:27.41
train_id:49889 target:0 grad:26.83
9th node
train_id:51248 target:9 grad:33.35
train_id:42566 target:9 grad:32.37
train_id:39423 target:9 grad:28.45
train_id:14582 target:9 grad:28.09
train_id:36104 target:9 grad:27.85

Figure 4.1: Top 5 explanatory images on each node in softmax layer of mnist classifier. Red letters indicates training points having same label with each node.

Softmax layer outputs the probability that the label of input image belongs to a certain class (from 0 to 9). From figure 4.1, we could see that the images matched each node. From the results of the first node, we could also see that all images are visually close to 1 and the actual targets of those images are 1. On the other hand, there was a result showing visually different and having different targets between node and resulting images (2nd node). In other cases, the targets of explanatory images were different from the node, but it was visually close to the number of node (4th node). Some results weren't visually close to their number of node and also had different targets, but overall, we were able to obtain training points that were visually close to each node and had the target same with the node.

4.1.2 Explanatory images on test image

Here we will look for the explanatory images for test images. We tracked the explanatory images on test images obtained by using two methods introduced previous chapter. For the test images, we chose 10 images, each of them has different class from 0 to 9 and we showed the most explanatory 5 training points. First, we will examine the explanatory images obtained by using attribution method, and later we will examine the explanatory images obtained by using influence function method. In particular, influence function method could track helpful training points and harmful training points, we will look at both of them.

Method 1. Attribution method

We tracked the explanatory images by computing influence values calculated from gradient norm and activation. Figure 4.2 shows little consistency between targets of the explanatory images and the prediction of the test image. Six out of 10 test images did not match up with any of the corresponding five most explanatory images. Although most targets of the points didn't match up with that of test image, some explanatory images were visually close to the corresponding test image (test image: 9489, 7362). However, we couldn't tell whether these results are really explanatory images. To see if they are really explanatory images, we will execute an explpainability test later.

Method 2. Influence function method

Here, we will look at explanatory images obtained by influence function method. Using this method, we could track helpful points and harmful points on test images.

First if we look at figure 4.3, we could see almost every explanatory images were visually close to the test image and at the same time, the targets of them were same with the prediction of test image. On the other hand, when we look at figure 4.4, only one test image's prediction (test id: 7362) matched up with targets of explanatory points. Some of other points were visually close to their test image while having different targets, but overall, the explanatory images were not visually close to the test images. Likewise, with this result, we couldn't tell if these results are really explanatory images. We will also execute the explainability test of this method later.

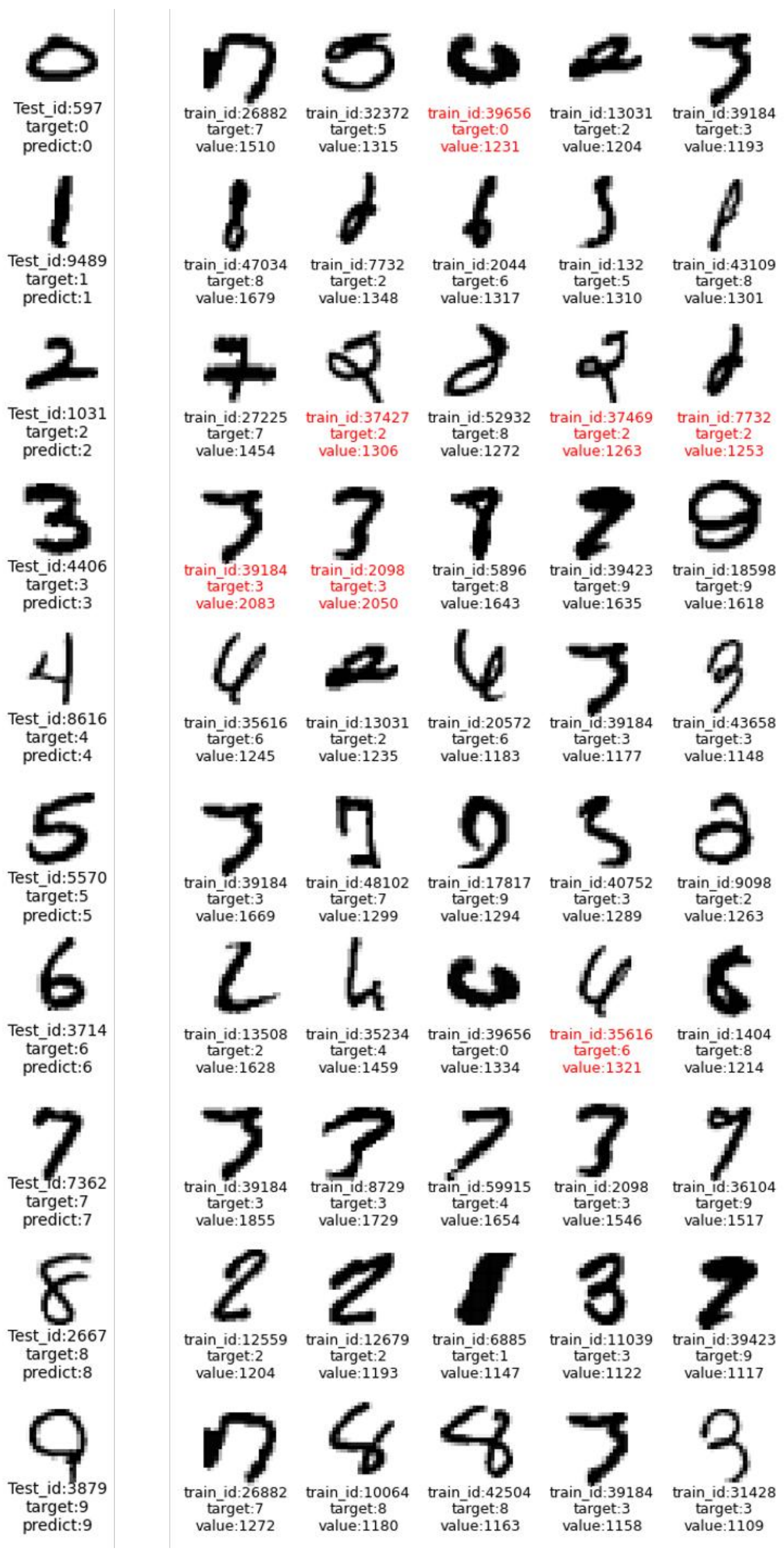

Test_id:597 target:0 predict:0
train_id:26882 target:7 value:1510
train_id:32372 target:5 value:1315
train_id:39656 target:0 value:1231
train_id:13031 target:2 value:1204
train_id:39184 target:3 value:1193
Test_id:9489 target:1 predict:1
train_id:47034 target:8 value:1679
train_id:7732 target:2 value:1348
train_id:2044 target:6 value:1317
train_id:132 target:5 value:1310
train_id:43109 target:8 value:1301
Test_id:1031 target:2 predict:2
train_id:27225 target:7 value:1454
train_id:37427 target:2 value:1306
train_id:52932 target:8 value:1272
train_id:37469 target:2 value:1263
train_id:7732 target:2 value:1253
Test_id:4406 target:3 predict:3
train_id:39184 target:3 value:2083
train_id:2098 target:3 value:2050
train_id:5896 target:8 value:1643
train_id:39423 target:9 value:1635
train_id:18598 target:9 value:1618
Test_id:8616 target:4 predict:4
train_id:35616 target:6 value:1245
train_id:13031 target:2 value:1235
train_id:20572 target:6 value:1183
train_id:39184 target:3 value:1177
train_id:43658 target:3 value:1148
Test_id:5570 target:5 predict:5
train_id:39184 target:3 value:1669
train_id:48102 target:7 value:1299
train_id:17817 target:9 value:1294
train_id:40752 target:3 value:1289
train_id:9098 target:2 value:1263
Test_id:3714 target:6 predict:6
train_id:13508 target:2 value:1628
train_id:35234 target:4 value:1459
train_id:39656 target:0 value:1334
train_id:35616 target:6 value:1321
train_id:1404 target:8 value:1214
Test_id:7362 target:7 predict:7
train_id:39184 target:3 value:1855
train_id:8729 target:3 value:1729
train_id:59915 target:4 value:1654
train_id:2098 target:3 value:1546
train_id:36104 target:9 value:1517
Test_id:2667 target:8 predict:8
train_id:12559 target:2 value:1204
train_id:12679 target:2 value:1193
train_id:6885 target:1 value:1147
train_id:11039 target:3 value:1122
train_id:39423 target:9 value:1117
Test_id:3879 target:9 predict:9
train_id:26882 target:7 value:1272
train_id:10064 target:8 value:1180
train_id:42504 target:8 value:1163
train_id:39184 target:3 value:1158
train_id:31428 target:3 value:1109

Figure 4.2: Top 5 explanatory points from attribution method on test images. Red letters indicates the points having same label with the label of the test image.

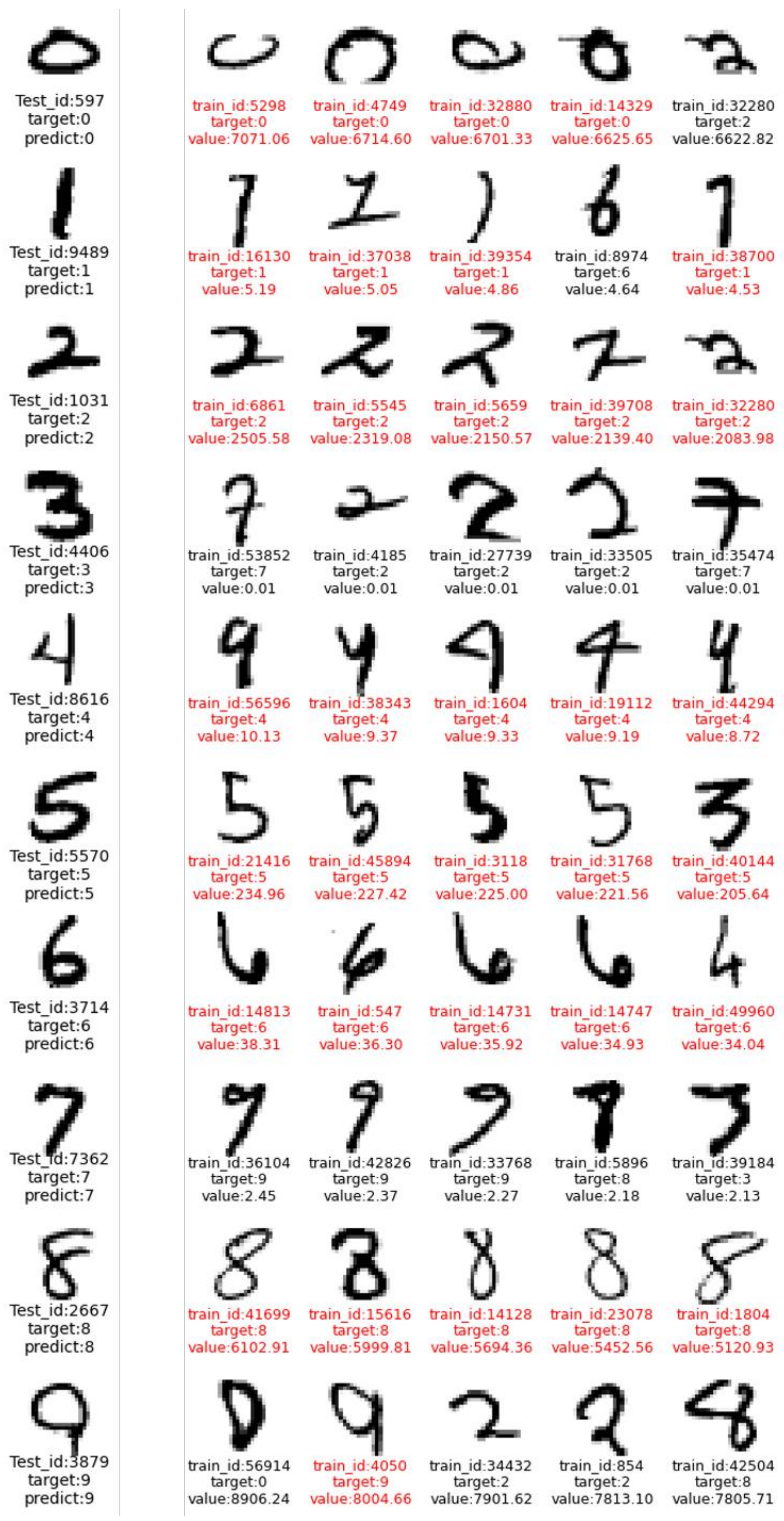


Figure 4.3: Top 5 helpful explanatory points from influence function method[9] on test images. Red letters indicates the points having same label with the label of the test image.

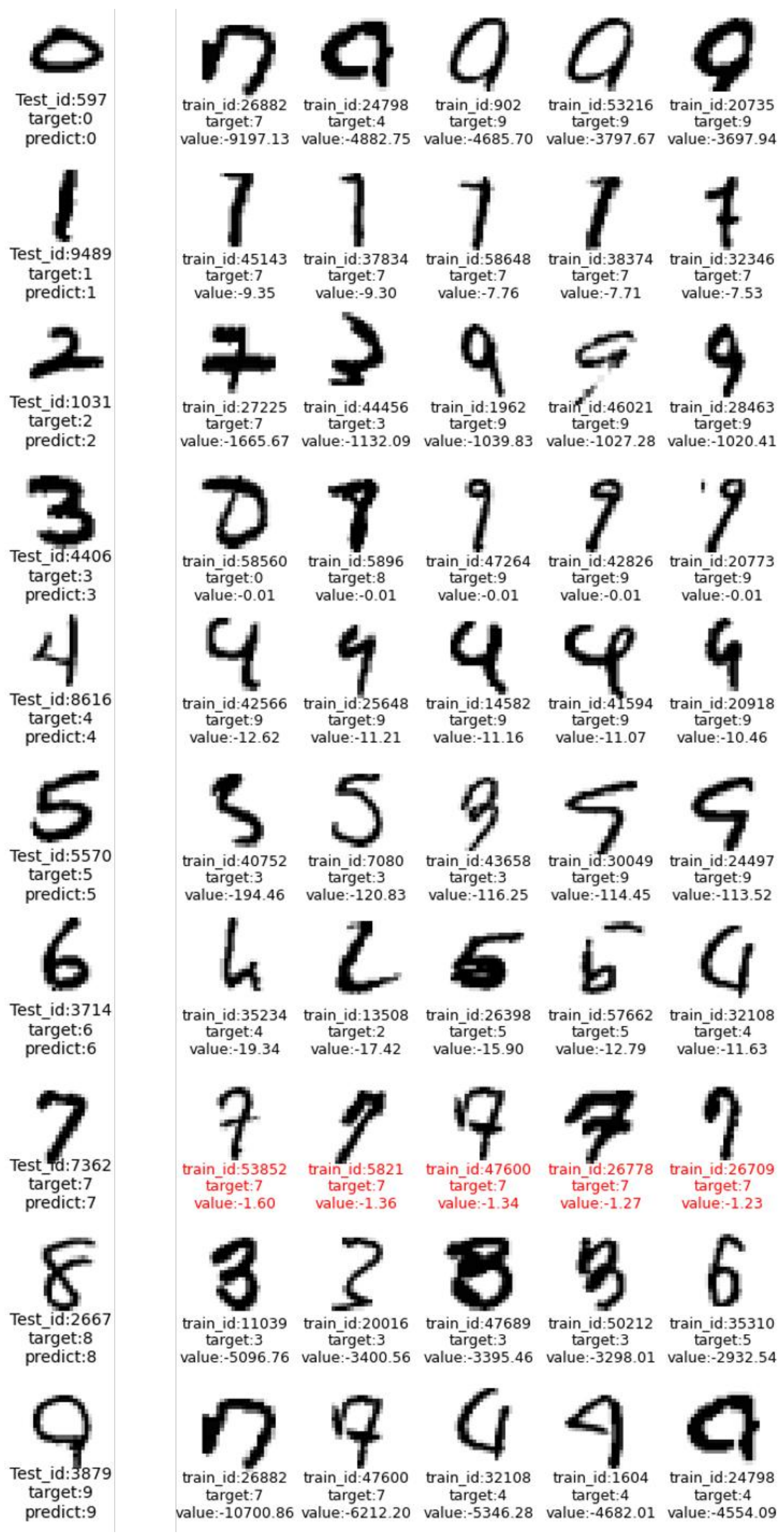

Test_id:597 target:0 predict:0
train_id:26882 target:7 value:-9197.13
train_id:24798 target:4 value:-4882.75
train_id:902 target:9 value:-4685.70
train_id:53216 target:9 value:-3797.67
train_id:20735 target:9 value:-3697.94
Test_id:9489 target:1 predict:1
train_id:45143 target:7 value:-9.35
train_id:37834 target:7 value:-9.30
train_id:58648 target:7 value:-7.76
train_id:38374 target:7 value:-7.71
train_id:32346 target:7 value:-7.53
Test_id:1031 target:2 predict:2
train_id:27225 target:7 value:-1665.67
train_id:44456 target:3 value:-1132.09
train_id:1962 target:9 value:-1039.83
train_id:46021 target:9 value:-1027.28
train_id:28463 target:9 value:-1020.41
Test_id:4406 target:3 predict:3
train_id:58560 target:0 value:-0.01
train_id:5896 target:8 value:-0.01
train_id:47264 target:9 value:-0.01
train_id:42826 target:9 value:-0.01
train_id:20773 target:9 value:-0.01
Test_id:8616 target:4 predict:4
train_id:42566 target:9 value:-12.62
train_id:25648 target:9 value:-11.21
train_id:14582 target:9 value:-11.16
train_id:41594 target:9 value:-11.07
train_id:20918 target:9 value:-10.46
Test_id:5570 target:5 predict:5
train_id:40752 target:3 value:-194.46
train_id:7080 target:3 value:-120.83
train_id:43658 target:3 value:-116.25
train_id:30049 target:9 value:-114.45
train_id:24497 target:9 value:-113.52
Test_id:3714 target:6 predict:6
train_id:35234 target:4 value:-19.34
train_id:13508 target:2 value:-17.42
train_id:26398 target:5 value:-15.90
train_id:57662 target:5 value:-12.79
train_id:32108 target:4 value:-11.63
Test_id:7362 target:7 predict:7
train_id:53852 target:7 value:-1.60
train_id:5821 target:7 value:-1.36
train_id:47600 target:7 value:-1.34
train_id:26778 target:7 value:-1.27
train_id:26709 target:7 value:-1.23
Test_id:2667 target:8 predict:8
train_id:11039 target:3 value:-5096.76
train_id:20016 target:3 value:-3400.56
train_id:47689 target:3 value:-3395.46
train_id:50212 target:3 value:-3298.01
train_id:35310 target:5 value:-2932.54
Test_id:3879 target:9 predict:9
train_id:26882 target:7 value:-10700.86
train_id:47600 target:7 value:-6212.20
train_id:32108 target:4 value:-5346.28
train_id:1604 target:4 value:-4682.01
train_id:24798 target:4 value:-4554.09

Figure 4.4: Top 5 harmful explanatory points from influence function method[9] on test images. Red letters indicates the points having same label with the label of the test image.

4.1.3 Stability test

Here we will look at the results of stability test of both methods. We measured stability by counting how many same explanatory points we obtained from trained model when we retrain mnist model from scratch 5 times. We counted how many same points are in 100 of images for each method. We trained and tracked the explanatory points 5 times and compared the results with the results obtained from a model trained with seed '0'. In this experiment, we sampled 10 test images with various loss values based on the seed '0' model.

Method 1. Attribution method

We will look at the results of stability test of attribution method with a table showing how many explanatory points obtained from each model are same with the points obtained from seed '0' model.

test_id	3520	3879	597	2667	5673	8616	5570	9489	3714	7362
1st	61%	65%	55%	48%	53%	65%	45%	63%	61%	66%
2nd	60%	67%	57%	54%	57%	55%	53%	64%	64%	62%
3rd	62%	65%	60%	52%	57%	54%	54%	67%	64%	67%
4th	63%	64%	55%	56%	62%	60%	54%	63%	58%	65%
5th	65%	61%	48%	52%	55%	54%	48%	65%	61%	68%
Avg	62%	64%	55%	52%	57%	58%	51%	64%	62%	66%

Table 4.1: We compared the explanatory points obtained from each model with explanatory points obtained from seed '0' model. Here we used attribution method to get the explanatory points

From table 4.1, we could see that attribution method is considerably stable. By test image, the average stability was between 51% and 66%. There were totally 60,000 training data, 60% means when we choose 100 points from 60,000 points using attribution method, we can obtain 60% identical points. This has a very small chance to happen if the images were chosen randomly.

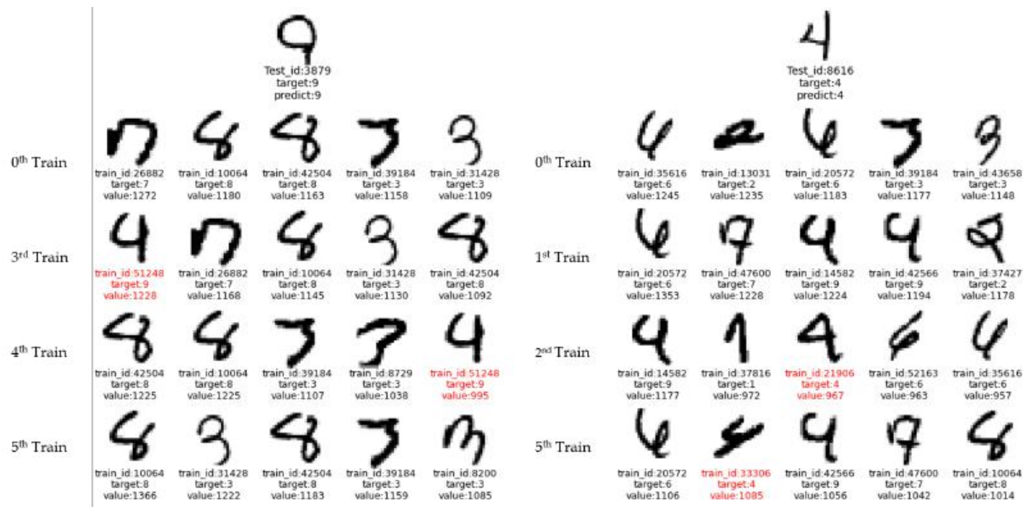

Test_id:3879 target:9 predict:9
0th Train
train_id:26882 target:7 value:1272
train_id:10064 target:8 value:1180
train_id:42504 target:8 value:1163
train_id:39184 target:3 value:1158
train_id:31428 target:3 value:1109
3rd Train
train_id:51248 target:9 value:1228
train_id:26882 target:7 value:1168
train_id:10064 target:8 value:1145
train_id:31428 target:3 value:1130
train_id:42504 target:8 value:1092
4th Train
train_id:42504 target:8 value:1225
train_id:10064 target:8 value:1225
train_id:39184 target:3 value:1107
train_id:8729 target:3 value:1038
train_id:51248 target:9 value:995
5th Train
train_id:10064 target:8 value:1366
train_id:31428 target:3 value:1222
train_id:42504 target:8 value:1183
train_id:39184 target:3 value:1159
train_id:8200 target:3 value:1085
Test_id:8616 target:4 predict:4
0th Train
train_id:35616 target:6 value:1245
train_id:13031 target:2 value:1235
train_id:20572 target:6 value:1183
train_id:39184 target:3 value:1177
train_id:43658 target:3 value:1148
1st Train
train_id:20572 target:6 value:1353
train_id:47600 target:7 value:1228
train_id:14582 target:9 value:1224
train_id:42566 target:9 value:1194
train_id:37427 target:2 value:1178
2nd Train
train_id:14582 target:9 value:1177
train_id:37816 target:1 value:972
train_id:21906 target:4 value:967
train_id:52163 target:6 value:963
train_id:35616 target:6 value:957
5th Train
train_id:20572 target:6 value:1106
train_id:33306 target:4 value:1085
train_id:42566 target:9 value:1056
train_id:47600 target:7 value:1042
train_id:10064 target:8 value:1014

Figure 4.5: Test image and the explanatory points obtained from different trained model. Red letters indicates the points having same label with that of test image.

We also looked at the explanatory images obtained from some models. From figure 4.5, we could see that the target of most explanatory images are different than the prediction of the test image. This is consistent with the results we saw in the previous section. Here, we could also see that most explanatory images appear more than once across different models, which means this method is stable.

From the results above, we could see that attribution method is stable enough.

Method 2. Influence function method (helpful)

Here, we will look at the results of stability test of influence function method with a table showing how many helpful points obtained from each model are same with the points obtained from seed '0' model.

test_id	3520	3879	597	2667	5673	8616	5570	9489	3714	7362
1st	7%	8%	21%	31%	25%	14%	21%	15%	28%	44%
2nd	16%	1%	11%	0%	0%	3%	4%	1%	1%	1%
3rd	18%	8%	0%	4%	0%	0%	3%	3%	0%	28%
4th	15%	12%	40%	27%	1%	16%	1%	5%	34%	44%
5th	14%	7%	1%	0%	0%	9%	2%	23%	1%	3%
Avg	14%	7%	15%	12%	5%	8%	6%	9%	13%	24%

Table 4.2: We compared the helpful points obtained from each model with helpful points obtained from seed '0' model. Here we used influence function method to get the explanatory points.

From table 4.2, we could see that helpful points of influence function method is relatively less stable. By test image, the average stability was between 5% and 24%. We can see that this method is fairly stable but it was less stable compared to attribution method.

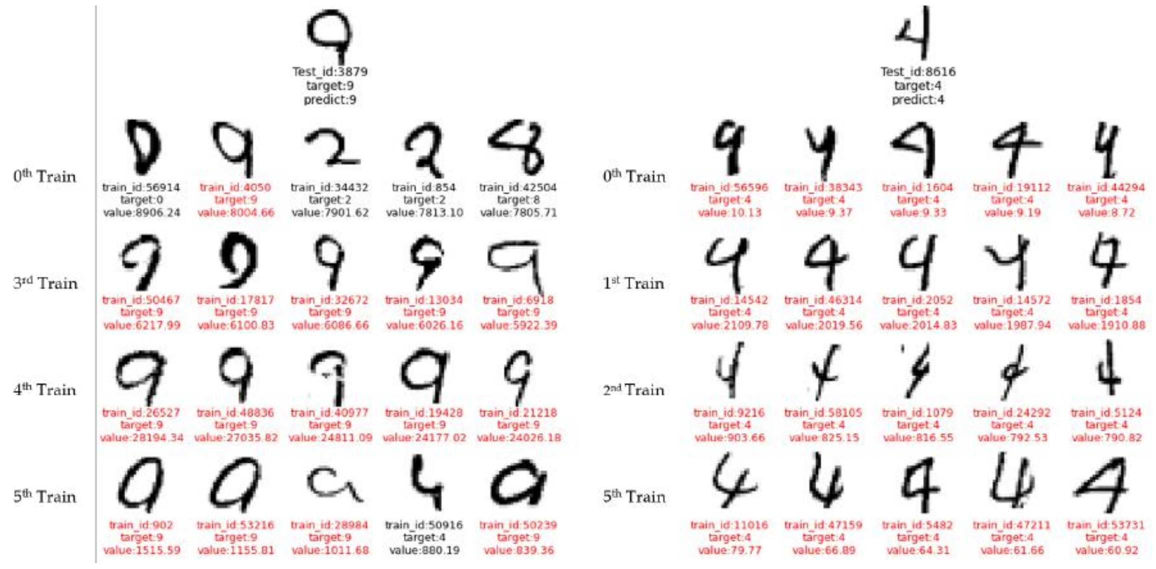

Test_id:3879
target:9
predict:9
Test_id:8616
target:4
predict:4
0th Train
3rd Train
4th Train
5th Train
1st Train
2nd Train
train_id:56914 target:0 value:8906.24
train_id:4050 target:9 value:8004.66
train_id:34432 target:2 value:7901.62
train_id:854 target:2 value:7813.10
train_id:42504 target:8 value:7805.71
train_id:50467 target:9 value:6217.99
train_id:17817 target:9 value:6100.83
train_id:32672 target:9 value:6086.66
train_id:13034 target:9 value:6026.16
train_id:6918 target:9 value:5922.39
train_id:26527 target:9 value:28194.34
train_id:48836 target:9 value:27035.82
train_id:40977 target:9 value:24811.09
train_id:19428 target:9 value:24177.02
train_id:21218 target:9 value:24026.18
train_id:902 target:9 value:1515.59
train_id:53216 target:9 value:1155.81
train_id:28984 target:9 value:1011.68
train_id:50916 target:4 value:880.19
train_id:50239 target:9 value:839.36
train_id:56596 target:4 value:10.13
train_id:38343 target:4 value:9.37
train_id:1604 target:4 value:9.33
train_id:19112 target:4 value:9.19
train_id:44294 target:4 value:8.72
train_id:14542 target:4 value:2109.78
train_id:46314 target:4 value:2019.56
train_id:2052 target:4 value:2014.83
train_id:14572 target:4 value:1987.94
train_id:1854 target:4 value:1910.88
train_id:9216 target:4 value:903.66
train_id:58105 target:4 value:825.15
train_id:1079 target:4 value:816.55
train_id:24292 target:4 value:792.53
train_id:5124 target:4 value:790.82
train_id:11016 target:4 value:79.77
train_id:47159 target:4 value:66.89
train_id:5482 target:4 value:64.31
train_id:47211 target:4 value:61.66
train_id:53731 target:4 value:60.92

Figure 4.6: Test image and the helpful points obtained from different trained model. Red letters indicates the points having same label with that of test image.

From figure 4.6, we could see that the target of most explanatory images are same with the prediction of the test image. This is consistent with the results we saw in the previous section. Here, we could also see that most explanatory images appear only once across different models, which means this method isn't that stable

From the results above, we could see that influence function method (helpful points) is relatively less stable.

Method 2. Influence function method (harmful)

Here, We will look at the results of stability test of influence function method with a table showing how many harmful points obtained from each model are same with the points obtained from seed '0' model.

test_id	3520	3879	597	2667	5673	8616	5570	9489	3714	7362
1st	24%	7%	21%	26%	23%	14%	27%	32%	10%	29%
2nd	26%	8%	12%	0%	0%	12%	8%	3%	3%	1%
3rd	17%	10%	1%	21%	0%	0%	15%	3%	4%	23%
4th	47%	17%	36%	21%	0%	19%	20%	12%	20%	18%
5th	10%	7%	4%	1%	0%	19%	12%	38%	1%	3%
Avg	25%	10%	15%	14%	5%	13%	16%	18%	8%	15%

Table 4.3: We compared the harmful points obtained from each model with harmful points obtained from seed '0' model. Here we used influence function method to get the explanatory points.

From table 4.3, we can see that harmful points of influence function algorithm is relatively less stable. By test image, the average stability was between 5% and 25%. This results were similar with helpful points. We can see that this method is fairly stable but it was less stable compared to attribution method.

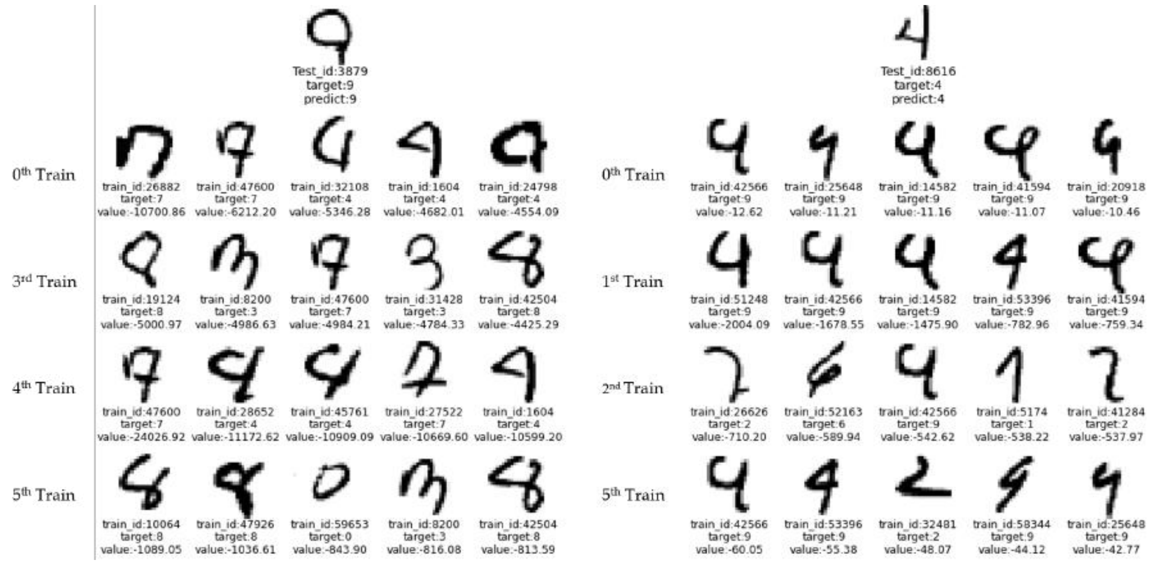


Figure 4.7: Test image and the helpful points obtained from different trained model.

From figure 4.7, we can see that the target of most explanatory images are different with the prediction of corresponding test image. This is consistent with the results we saw in the previous section. Here, we can also know that some explanatory images only appear once across different models but some explanatory images appear more than once across different models

From the results above, we could see that influence function method (harmful points) is relatively less stable.

4.1.4 Explainability test

Here we executed explainability test of both methods. We measured explainability by the change of loss of test images passed through models that are trained with database containing manipulated (change labels or remove) data.

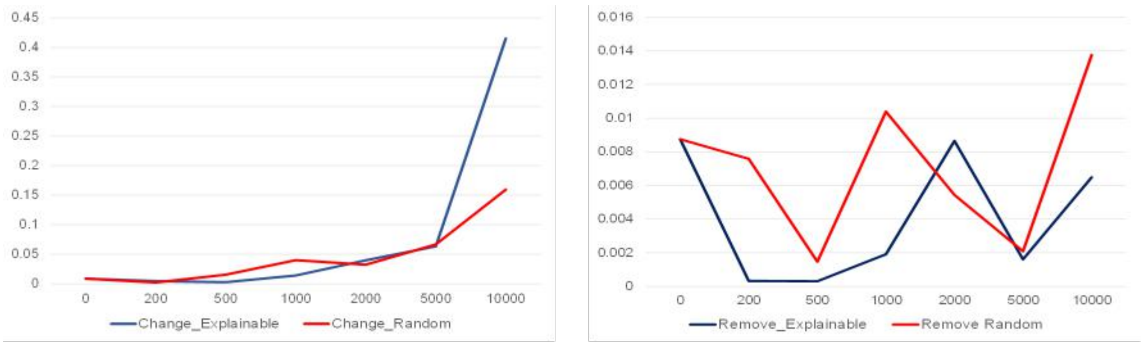
We manipulated the data in the following two ways:

1. We changed labels of explanatory points or labels of random points. At this time, we changed all labels of explanatory points into '0', but when the target label of test image is '0', we changed the labels of explanatory points into '1'.
2. We removed explanatory points or random points.

We tested with 10 test images sampled in the same way we did in stability test. We looked at the change of loss when we manipulated 0, 200, 500, 1000, 2000, 5000, 10,000 points, and we plotted the average of the change of loss of test images.

Method 1. Attribution method

Here, we will look at the change of loss of test images when we pass those images through models trained with training data set containing manipulated random points or explanatory points obtained from attribution method.



(a) Change labels (b) Remove points

Figure 4.8: The change of loss of test images. We plotted the average loss of test images (a) when we changed labels of explanatory points (blue) or random points (red) (b) when we removed explanatory points (blue) or random points (red).

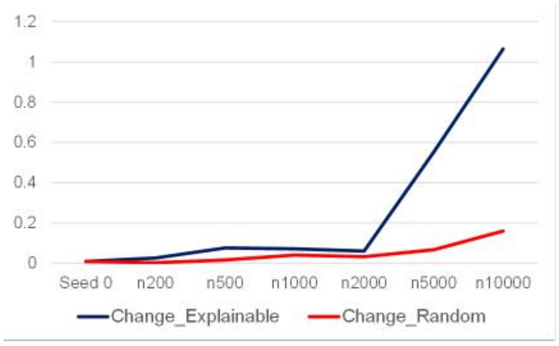
From figure 4.8 (a), we could see that the loss increases when we change labels of more training points. Especially, it shows changing labels of explanatory points more than doubled the loss compared to changing labels of random points. On the other hand, From figure 4.8 (b), it was difficult to find any tendency whether to remove explanatory points or random points.

When we changed labels of the points, there was a change in the model's decision when we changed 10,000 explanatory points (the decision of test image (id: 597) changed from 0 to 1). On the other hand, when we changed labels of random points, there was no change in the model's decision.

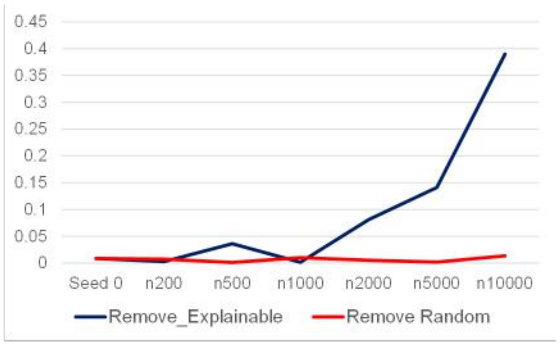
This shows that the explanatory points obtained from attribution method were not powerful but could affect the model's decision to some extent.

Method 2. Influence function method (helpful)

Here, we will look at the change of loss of test images when we pass those images through models trained with training data set containing manipulated random points or helpful points obtained from influence function method.



(a) Change labels



(b) Remove points

Figure 4.9: The change of loss of test images. We plotted the average loss of test images (a) when we changed labels of helpful points (blue) or random points (red) (b) when we removed helpful points (blue) or random points (red).

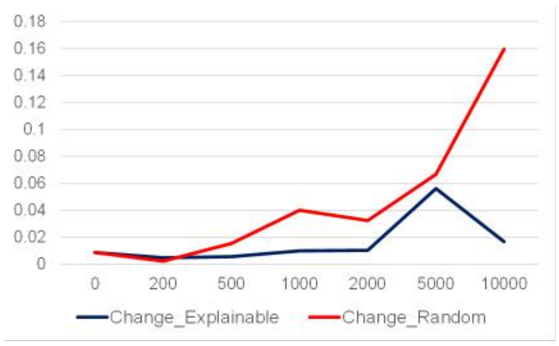
From figure 4.9 (a), we could see that the loss increases when we change labels of more training points. Especially, when we change labels of helpful points, the loss was more than five times bigger than when we change labels of random points. This trend can also be seen in figure 4.9 (b), where the loss is steeply increased when helpful points are removed compared to when random points are removed.

In fact, when we changed labels of the points, from when we changed 5000 helpful points, the model decision on some test images changed. When we changed 5,000 helpful points, model decisions of two test points out of a total of 10 test points were changed (id:3520 from 6 to 0 and id:597 from 0 to 1). Even when labels of 10,000 helpful points were changed, the decision of five test points out of a total of 10 test points were changed (id:3520 from 6 to 0, id:3879 from 9 to 0, id:597 from 0 to 1, id:2667 from 8 to 0 and id:5570 from 5 to 0). When we removed helpful points, some decisions were also changed (when 5,000 helpful points were removed, id:597 from 0 to 9 and when 10,000 helpful points were removed, id:597 from 0 to 4). On the other hand, when we change labels of random points or remove random points, there was no change in model decision.

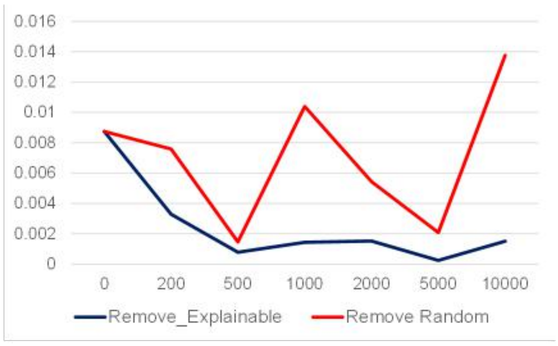
This showed that the helpful explanatory points obtained through influence function method were powerful to affect the model's decision.

Method 2. Influence function method (harmful)

Here, we will look at the change of loss of test images when we pass those images through models trained with training data set containing manipulated random points or harmful points obtained from influence function method.



(a) Change labels



(b) Remove points

Figure 4.10: The change of loss of test images. We plotted the average loss of test images (a) when we changed labels of harmful points (blue) or random points (red) (b) when we removed harmful points (blue) or random points (red).

From figure 4.10 (a), we could see that changing labels of harmful points has no significant effect on the loss change. On the other hand, from figure 4.10 (b), loss tends to decrease when the harmful points are removed.

There was no significant loss change in test image as a way to change the label or remove harmful points. In fact, even when we manipulated more harmful points, the model decision didn't change.

4.2 FFDNet

In this section, we will discuss about the results obtained by applying the methods experimented in previous section to FFDNet; attribution method we proposed and influence function method.

4.2.1 Explanatory images of on test images

We applied the methods that were previously applied to mnist model to FFD-Net to track explanatory points for test images. This time, unlike mnist, we tried to track the points for a particular point in test images, but due to the nature of the method, tracking the points for a particular point in test image was possible only with attribution method. In the case of influence function method, because the explanatory points for the entire area of test image is computed, it is difficult to obtain explanatory points for a particular point. Therefore, we could track explanatory points for a particular point in test images only with attribution method. Although it is difficult to track the points for a specific area of test image with influence function method, we tried to track the points for the entire area of test image with influence function method.

Method 1. Attribution method

Unlike the mnist model, FFDNet has a wide and deep network, making it difficult to utilize gradient norm and activations of all layers. Therefore, we have to choose a specific layer to compute the influence value, we selected the last layer of FFDNet for computing them and tracked the explanatory points using the values of last layer. We sampled two test images from Set14 data set and tracked the points for particular points of those images.

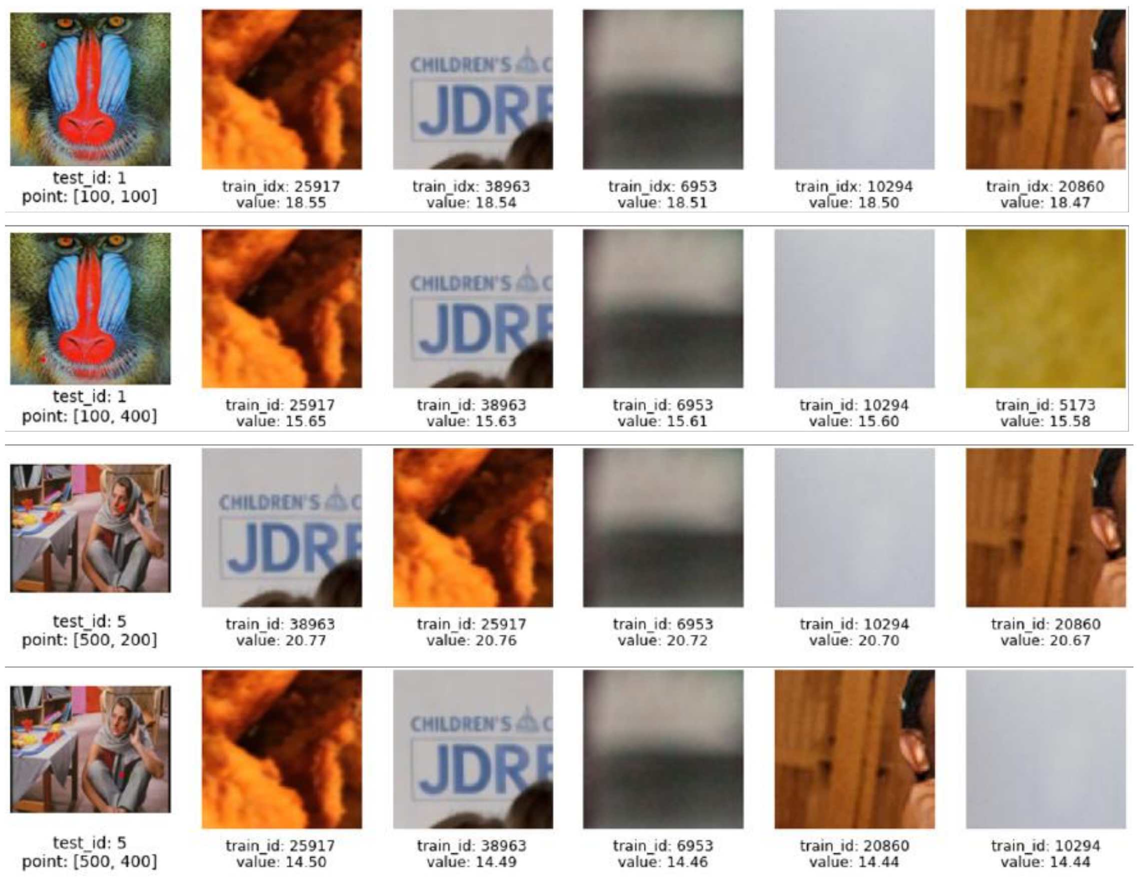


Figure 4.11: The explanatory points for a particular point of two test images. Red point on test image means a particular point where we want to track.

From figure 4.11, even though we tried to track explanatory points for different points of two different test images, we obtained same explanatory images. images that have nothing to do with the particular point in the test images were shown. Further modifications and experiments on the method are needed. We will discuss about this in the next subsection.

Method 2. Influence function method

Here, we tried to track explanatory points obtained from influence function method. Due to the nature of this method, we couldn't find the points for particular points of test image, instead we tried to find the points for the entire area of test image.

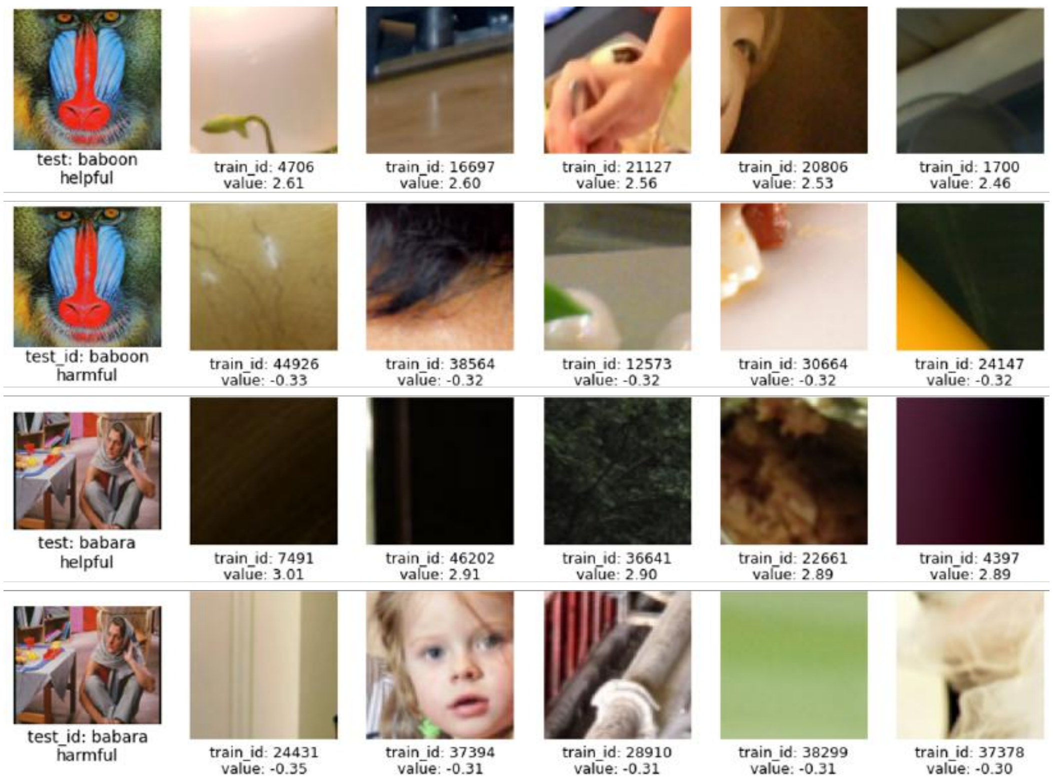


Figure 4.12: The explanatory points for the entire area of two test images. We could get two results (helpful points and harmful points) from influence function method.

From figure 4.12, we could see that unlike the attribution method, the resulting images were different across different test image. However, the resulting points didn't seem to be related to the test image, which requires an further experiments on this method.

4.2.2 Explanatory images - Modifications

Eliminating Outliers

Here, we will explore why images unrelated to the test image appear repeatedly across multiple points of multiple images. We first thought that the same images appear repeatedly across different points of images because the gradient norm of those images are too large comparing to other points. Therefore, we plotted a histogram of gradient norm values to see if there are outliers. The graph is shown below.

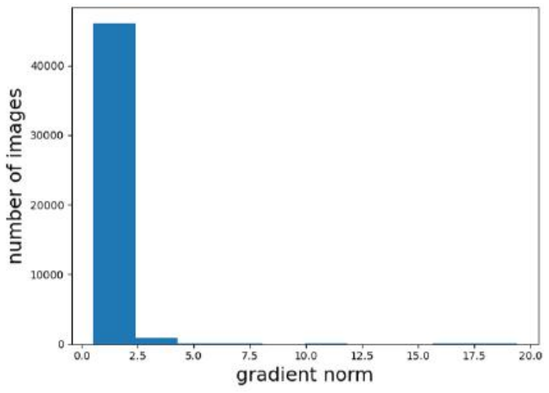


Figure 4.13: Histogram of gradient norm of training points.

Figure 4.13 shows that the gradient norm of most images range from 0 to 5, but some images were bigger than 10 and even some of them bigger than 17.5. There were 122 images having gradient norm bigger than 5. When we look at the gradient norm of the points shown in figure 4.11, all of them had a gradient norm greater than 12. This can be the reason that same images appear repeatedly across different points. Therefore, we tried to track the points excluding these outliers. The results are below:

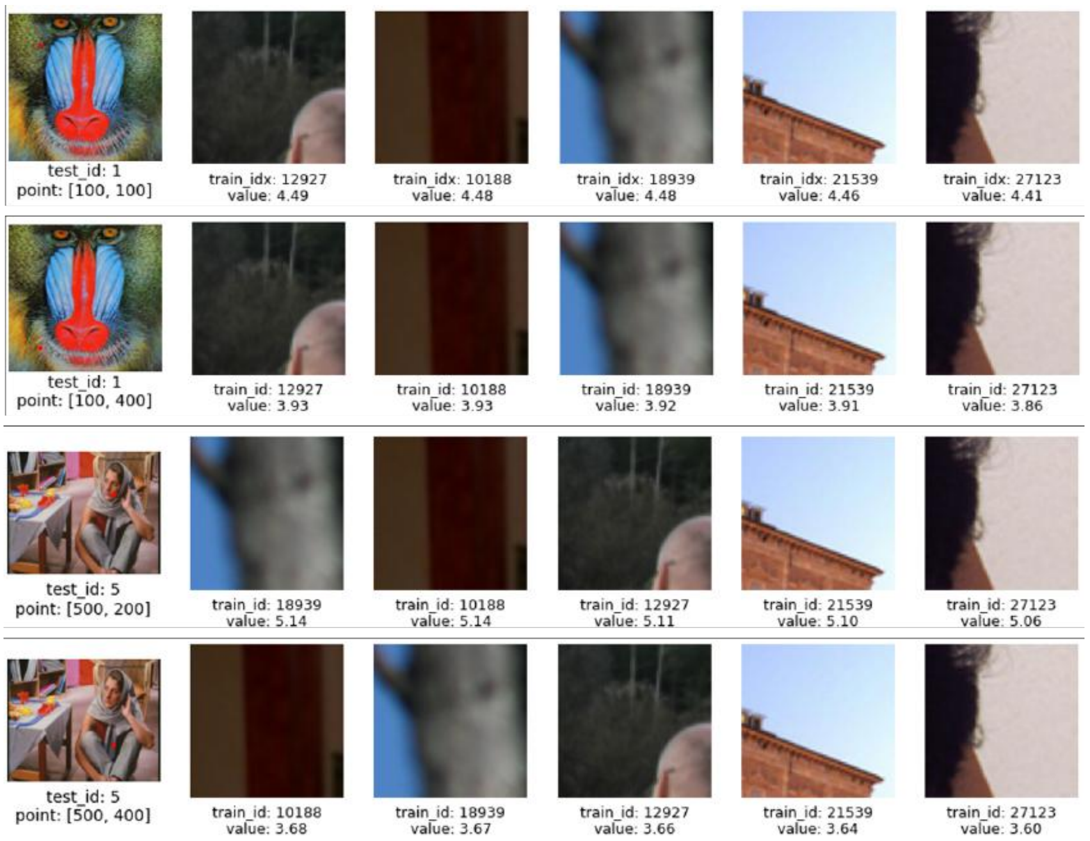


Figure 4.14: explanatory points excluding points having gradient norm greater than 5.

From figure 4.14, although we tried to find explanatory points excluding outliers having gradient norm bigger than 5, we obtained same points across different points like previous results. The images still don't seem to have much to do with the test image's point. It appears that further modifications to this method are needed.

Normalized gradient

Instead of looking for an explanatory points excluding outliers, we tried to track explanatory images with normalized the gradient norm. By normalizing the gradient norm, we expected the computation of influence value to be more stable. The results with normalized gradient norm are below:

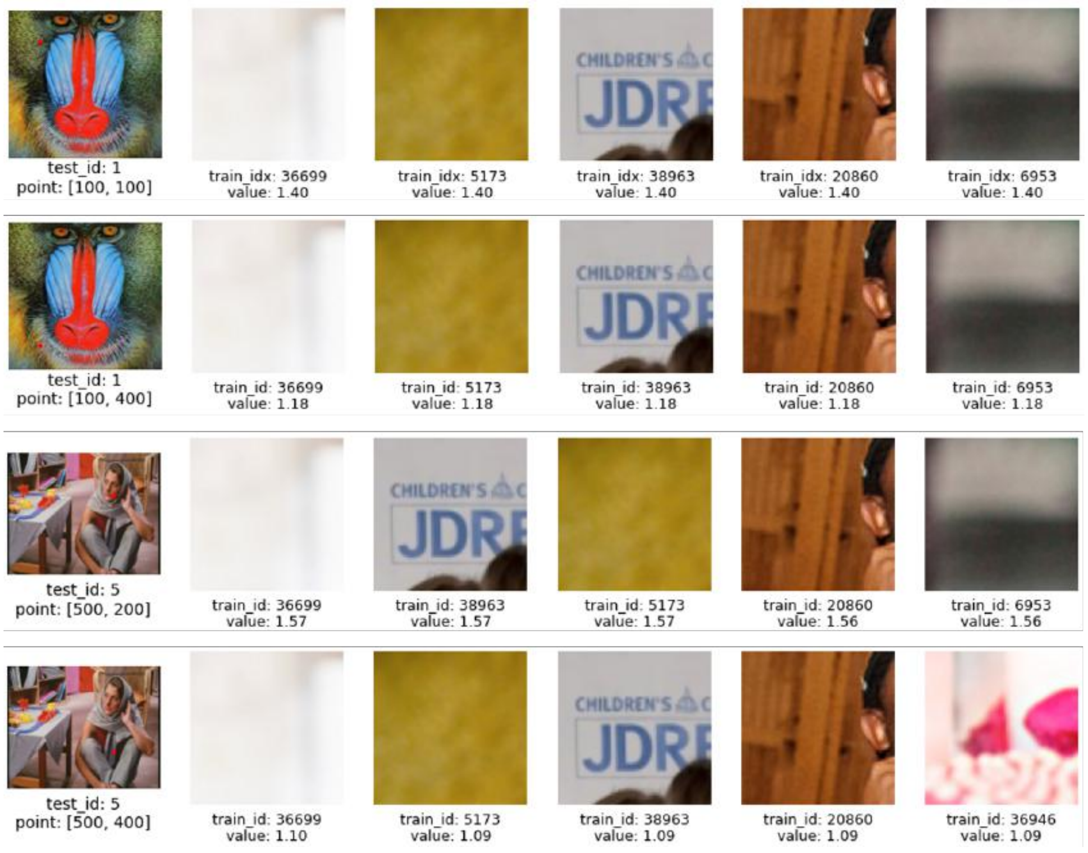


Figure 4.15: explanatory points derived with normalized gradient norm

From figure 4.15, although we tried to find explanatory points with normalized gradient norm, we obtained same points across different points. And they still didn't seem to be related to the test image's point.

Which layer?

So far, we tried to find the points with the values from 11th layer in FFDNet. Here we tried to find the explanatory points with the values from various layers in FFDNet and compared the results.

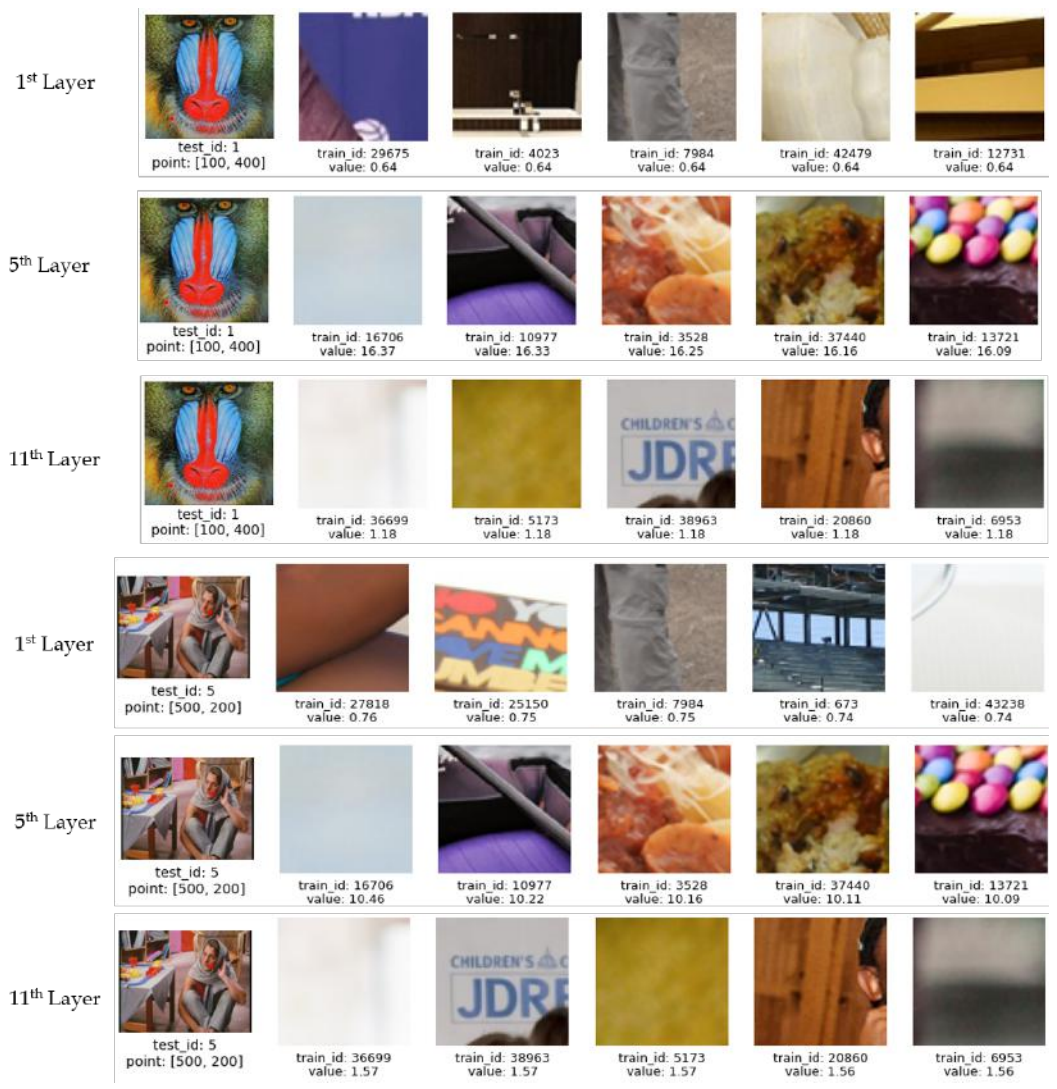

1st Layer
test_id: 1 point: [100, 400]
train_id: 29675 value: 0.64
train_id: 4023 value: 0.64
train_id: 7984 value: 0.64
train_id: 42479 value: 0.64
train_id: 12731 value: 0.64
5th Layer
test_id: 1 point: [100, 400]
train_id: 16706 value: 16.37
train_id: 10977 value: 16.33
train_id: 3528 value: 16.25
train_id: 37440 value: 16.16
train_id: 13721 value: 16.09
11th Layer
test_id: 1 point: [100, 400]
train_id: 36699 value: 1.18
train_id: 5173 value: 1.18
train_id: 38963 value: 1.18
train_id: 20860 value: 1.18
train_id: 6953 value: 1.18
1st Layer
test_id: 5 point: [500, 200]
train_id: 27818 value: 0.76
train_id: 25150 value: 0.75
train_id: 7984 value: 0.75
train_id: 673 value: 0.74
train_id: 43238 value: 0.74
5th Layer
test_id: 5 point: [500, 200]
train_id: 16706 value: 10.46
train_id: 10977 value: 10.22
train_id: 3528 value: 10.16
train_id: 37440 value: 10.11
train_id: 13721 value: 10.09
11th Layer
test_id: 5 point: [500, 200]
train_id: 36699 value: 1.57
train_id: 38963 value: 1.57
train_id: 5173 value: 1.57
train_id: 20860 value: 1.56
train_id: 6953 value: 1.56

Figure 4.16: Comparison between explanatory points derived from various layers in FFD-Net. We normalized the gradient norm.

From figure 4.16, we couldn't get images related to the point of test image from any layer. It seems that additional experiments and modifications are needed to apply the attribution method to FFDNet, rather than the question of which layer was selected.

Chapter 5
Conclusion

Today, understanding and explaining the decision of DNNs model is still an unanswered question. In this report, we tried to understand the inner-working of a denoising neural network. Because having a complete human-understandable explanation of DNN works is out of reach, we tried to track what training points were most influential for the model to denoise certain area of test image.

We tried to track explanatory images for test images by using attribution method we introduced and influence function method introduced in Koh et al.[9].

Before we apply the methods directly on FFDNet, we executed some experiments by applying them on mnist classifier.

From the experiments, we could know:

1. We can track explaining points for the network with gradient norm.
2. We can track explanatory points for test images by using the methods. With attribution method, we could get the points having different targets with that of test image but visually close to the test image. With influence function method, we could get the points having same targets with that of test image and visually close to the test image.
3. From stability test and explainability test, we could know that attribution method is considerably stable and the explanatory points weren't powerful but can affect on the decision of model to some extent. Also, influence function method wasn't stable as much as attribution method but fairly stable, and the explanatory points could effectively affect on the decision of model.

After the experiments on mnist classifier, we tried to track the explanatory points of denoising neural network by applying the methods to FFDNet. As a result, in both methods, we couldn't achieve a satisfactory results, which most images didn't seem to be related to the point of test image we want to track. Further experiments on applying the methods on FFDNet seem to be needed.

Bibliography

[1] Naman Agarwal, Brian Bullins, and Elad Hazan. “Second-order stochastic optimization in linear time”. In: *stat* 1050 (2016), p. 15.

[2] Jie Chen et al. “Adaptive explainable neural networks (axnns)”. In: *arXiv preprint arXiv:2004.02353* (2020).

[3] Eunsuk Chong, Chulwoo Han, and Frank C Park. “Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies”. In: *Expert Systems with Applications* 83 (2017), pp. 187–205.

[4] Kostadin Dabov et al. “Image denoising by sparse 3-D transform-domain collaborative filtering”. In: *IEEE Transactions on image processing* 16.8 (2007), pp. 2080–2095.

[5] Petko Georgiev et al. “Low-resource multi-task audio sensing for mobile and embedded devices via shared deep neural network representations”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1.3 (2017), pp. 1–19.

[6] Gaurav Goswami et al. “MDLFace: Memorability augmented deep learning for video face recognition”. In: *IEEE International Joint Conference on Biometrics*. IEEE. 2014, pp. 1–7.

[7] Shuhang Gu et al. “Weighted nuclear norm minimization with application to image denoising”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 2862–2869.

[8] Ashesh Jain et al. “Car that knows before you do: Anticipating maneuvers via learning temporal driving models”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 3182–3190.

[9] Pang Wei Koh and Percy Liang. “Understanding black-box predictions via influence functions”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1885–1894.

[10] Yin Lou et al. “Accurate intelligible models with pairwise interactions”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013, pp. 623–631.

[11] Liqiang Nie et al. “Disease inference from health-related questions via sparse deep learning”. In: *IEEE Transactions on knowledge and Data Engineering* 27.8 (2015), pp. 2107–2119.

[12] Barak A Pearlmutter. “Fast exact multiplication by the Hessian”. In: *Neural computation* 6.1 (1994), pp. 147–160.

[13] Thai T Pham and Yuanyuan Shen. “A deep causal inference approach to measuring the effects of forming group loans in online non-profit microfinance platform”. In: *arXiv preprint arXiv:1706.02795* (2017).

[14] Michael Tsang et al. “Neural interaction transparency (nit): Disentangling learned interactions for improved interpretability”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 5804–5813.

[15] Joel Vaughan et al. “Explainable neural networks based on additive index models”. In: *arXiv preprint arXiv:1806.01933* (2018).

[16] Ning Xie et al. “Explainable deep learning: A field guide for the uninitiated”. In: *arXiv preprint arXiv:2004.14545* (2020).

[17] Zebin Yang, Aijun Zhang, and Agus Sudjianto. “Enhancing explainability of neural networks through architecture constraints”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[18] Kai Zhang, Wangmeng Zuo, and Lei Zhang. “FFDNet: Toward a fast and flexible solution for CNN-based image denoising”. In: *IEEE Transactions on Image Processing* 27.9 (2018), pp. 4608–4622.