The landscape of local minimas for image classification

Milad Bakhshizadeh*, Saravanan Govindarajan*, Sairam Satwik Kondamudi*

Abstract

Despite the astonishing success of deep convolutional neural networks, the mechanism by which their inner layers act has not been understood completely yet. We develop multiple tools to study the evolution of parameters and performances of such networks during and after training. Moreover, we study the relation of extracted features by different models using statistical tools such as correlation and mutual information. We train 126 models and make over 3000 plots which visualize the information we extracted by this study. Nevertheless, tools we develop here are not restricted to the data we generate and can be used in a quite general setup. We believe the tools we develop here and the data we generate by using them can be insightful for future studies and can help us to understand the inner mechanism of widely used neural network models.

1 Introduction

Image classification has been studied extensively in the literature of Machine learning. Several labeled datasets have been made for the purpose of learning and many neural network architectures have been proposed to solve the image classification problem.

Despite the practical success of such models, the inner mechanism of neural networks have remained mostly unknown. As a matter of fact, the unclear performance of hidden layers in deep network models is one of the most important drawback of such models and has raised several objections. Treating neural network as a successful black-box for computationally intensive problems prevents us to extend their applications to sensitive areas such as medical diagnoses and self driving cars. Recently, there has been some efforts to shed a light on the dynamics of inner parameters of a deep network and explain the rationale of its success.

We study the dynamic of all the parameters of a network during the training as well as the relation between features it extracts to classify an image. While some of the phenomenon we observed experimentally may have been known theoretically, developing a general framework to investigate any network experimentally and visualizing their data can be insightful for future studies and may reveal new facts about deep neural networks and their performances. Given the extensive literature of network architectures and datasets we narrow down our study to three well-known architectures, namely

Lenet, Resnet18, and VGG, and train them to classify images given by three datasets, namely Cifar10, Fashion MNIST, and MNIST. Nevertheless, our methodology is quite general and results can be extended to other models and datasets with minimal extra effort.

In order to study the landscape of the loss for each model we use a couple of initialization methods, namely kaiming-uniform, and kaiming-normal, introduced in [4]. Moreover, we load available pre-trained models, and perturb the parameters of each trained model to initialize a new model and train it. With this method, we obtain bunch of local minimas for each model which gives us the ingredients of studying the landscape of local minimas for all the models we investigate in this paper.

Furthermore, we study the features each trained model extracts from an image. We look at the correlation and mutual information of the extracted features and associate features obtained by different models with a greedy algorithm. Moreover, using a simple update on [10], we visualize characteristics of an image that the network sees in giving out certain feature. We change only one thing from their approach, i.e., the localization maps will be w.r.t a particular unit in the feature but not the classification score.

1.1 Related works

Image Recognition. LeNet-5([8]) is one of the first works on image recognition in the space of Deep Learning. Being one of the earliest models, this model had a very simple 5-layer deep architecture. It is believed that deeper architectures have ability to learn better features. AlexNet([7]) with billions of trainable parameters and a deeper architecture paved the way in this direction. VGG-Net([11]) is further improvement on AlexNet([7]) with smaller receptive fields and lesser number of parameters. The original paper proposed 2 models with 16 and 19 layers(VGG16, VGG19) deep networks respectively. But this method of training reached a plateau in the learning ability of deep network. With increasing depth, contradictory to the regular belief the performance started dipping with further depth. This is called the degradation problem: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and further depth increased even the training error. Resnet([5]) introduced skip connections to solve this problem and introduced multiple models with skip connections and depths of 18, 34, 50, 101, 152 layers respectively. All these models have excellent performance on the Imagenet Image recognition task. With minimal change in performance for different initializations, all these models consistently produced significant performance. We intend to study the properties of the features extracted for different initializations([4]) and features of models obtained by finetuning the Imagenet-pretrained models. Studying cross correlation and Mutual information between features of different models is an interesting study. Works like MINE([1]) maximize the lower bound on the mutual information between two variables using a neural network. Works like this have made the Mutual information estimation simpler. Furthermore, in relating to our analysis on the parameters of the trained

networks for different initialization methods, we slightly relate to the "Lazy training" phenomenon. Without having to do any explicit scaling like it was done by [2] we noticed something similar in our experiments. We also experiment on deeper networks like Resnet([5]) unlike [2]. Our analysis is based on the widely accepted assumption that there are many local minima for the Neural network optimization problem. There is an ongoing debate about the which minima are better, sharp or flat?([6],[3]). We are not going to discuss this debate, but consider this debate's premise that there are many minimas(possibly infinite) all of which yield a descent solution.

2 Main result

2.1 Background

In this section we briefly review basic concepts of deep neural networks which are important to present our results.

A deep neural network consists of several layer of transformations. Each layer is made of a linear transformation, i.e. $\boldsymbol{x} \mapsto \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ followed by a nonlinear transformation $\boldsymbol{y} \mapsto \sigma(\boldsymbol{y})$. The most commonly used non-linearity is called Rectified Linear Unit (ReLU) which is defined as following:

$$\sigma(\boldsymbol{y}) = \big(\max(y_1, 0), \max(y_2, 0), ..., \max(y_n, 0)\big)\,,$$

where $\boldsymbol{y} \in \mathbb{R}^n$. $\boldsymbol{W}, \boldsymbol{b}$ in the linear transformations are called weights and biases, respectively. The output of each layer is fed as an input to nodes of the following layer. We will call the value of each node at the last convectional layer a feature. Extracted features of an image will be fed to a linear classifier, which is usually called the fully connected layer (FC), to determine the class to which the image belongs. The number of nodes in the FC layer matches with the number of classes we have, and the value of i^{th} node in the FC layer is interpreted as the probability that the image belongs to the i^{th} class. The following diagram shows the structure of a network. The number of the layers and connection between the inner nodes are determined by the architecture of the network.

To train a model, we define a loss function between models predictions and true label of the images and update the value of tunable parameters, weights and biases, to minimize the loss. In this work, we use Cross Entropy between predictions and truth as the loss function.

For a deep network with n features, we will denote its features by $f_{\boldsymbol{\theta}} = \big(f_{\boldsymbol{\theta},1}, ..., f_{\boldsymbol{\theta},n}\big)$, where the vector $\boldsymbol{\theta}$ denotes the value of ALL weights and biases except for the FC layer. Each $f_{\boldsymbol{\theta},i}$ will be a mapping from images to real numbers. The FC layer will be denoted by $fc_{\boldsymbol{\theta}_{fc}} : \mathbb{R}^n \to \{1, 2, ..., c\}$, where $\boldsymbol{\theta}_{fc}$ consists of weights and biases of the last layer, and c is the number of classes. Hence a network can be seen as a map from images to set $\{1, 2, ..., c\}$ and will be denoted by $fc_{\boldsymbol{\theta}_{fc}} \circ f_{\boldsymbol{\theta}}$. With this notation, training is finding a local minima $(\boldsymbol{\theta}, \boldsymbol{\theta}_{fc})$ that locally minimizes the chance of wrong prediction by the model. When it is clear from the context, we absorb $\boldsymbol{\theta}_{fc}$ to $\boldsymbol{\theta}$ and write $\boldsymbol{\theta}$ for $(\boldsymbol{\theta}, \boldsymbol{\theta}_{fc})$.

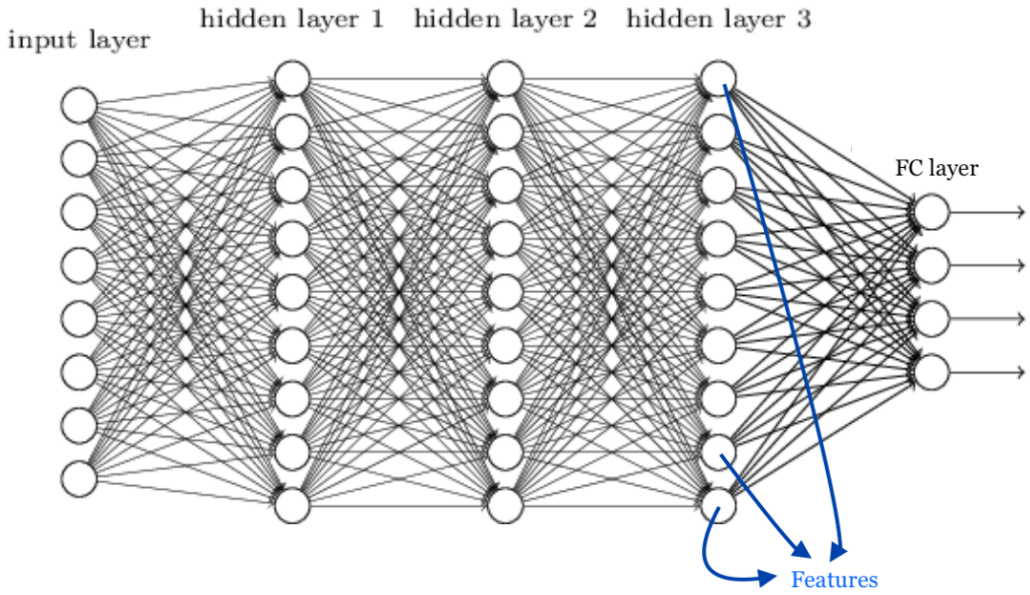

Figure 1: Network architecture from [9]

2.2 Networks and datasets

We trained three different neural architectures, namely Lenet, Resnet18, and VGG19, which have 3246, 11×10^6, and 138×10^6 trainable parameters in 5, 16, and 19 layers, respectively. In order to make all models compatible with our datasets, we changed the output size of FC layer to match the number of classes in our datasets, i.e. 10.

We train each architecture over three datasets, CIFAR10, MNIST, and Fashion MNIST. Each dataset contains 60000 samples which is partitioned to subsets of size 50000 and 10000 samples for training and testing. All three datasets we used have 10 different classes and the goal of each trained network is to classify images to one of the 10 classes. Below table denotes more details about these datasets.

Name	Image size	Description
CIFAR10	$3 * 32 * 32$	Objects including airplane, automobile, ..., truck.
MNIST	$1 * 28 * 28$	Handwritten digits $(0 - 9)$
Fashion MNIST	$1 * 28 * 28$	Clothing including T-shirt/top, Trouser, ..., Ankle boot.

Table 1: Details of datasets we have used

Note that the CIFAR10 dataset comes in color image format and has 3 channels, while MNIST and Fashion MNIST come in grayscale style with only one channel. To unify the dimension of our datasets we repeated the single channel to the last two datasets a couple of times and consider all images as a 3-channel image. Figure 2 denotes a few samples from each dataset.

Figure 2: Sample images of each dataset. First row: MNIST, second row: Fashion MNIST, third row: CIFAR10.

2.3 Initialization

In order to obtain different local minimas for $\boldsymbol{\theta}$, we use variety of initialization methods to start the training with. We use kaiming-uniform and kaiming-normal for each of our models. In addition, for Resnet18 and VGG19 we load the available pre-trained model on ImageNet dataset as an initial point as well. Furthermore, after training each model with all of above initializations, we also disturb the obtained local minima $\boldsymbol{\theta}$ by adding a constant of value 10^{-6} to each parameter, i.e. $\boldsymbol{\theta} \mapsto \boldsymbol{\theta} + 10^{-6} \times \mathbf{1}$, and by flipping the sign of each parameter, i.e. $\boldsymbol{\theta} \mapsto -\boldsymbol{\theta}$. This way, we obtain two new initial point to retrain the network again starting from them.

There are a few cases that some of above initializations are not available for a particular model. For instance, there is no pre-trained data for Lenet model, and flipping the sign of $\boldsymbol{\theta}$ yields some error for a pre-trained VGG19 model. In these cases we simply skip such initialization for that particular model.

2.4 Optimizer

We have used two commonly used optimizers, namely SGD and ADAM, to train our models from each of the initial point explained above. For SGD, we use *learning rate = 0.005* , and *momentum = 0.9* . For optimizing with ADAM, we again use *learning rate = 0.005* and *epsilon = 0.1*. We train each model for 20 epochs.

2.5 Methodology

Our study contains four main parts. At the first part, we study the dynamic of a network parameters during the training. In the second part, we study the relation between features extracted by different models with different parameters. Next, we obtain the mutual information between some extracted features, and finally visualize each feature using Grad-CAM. Details of each part is explained in the following.

2.5.1 Part 1: dynamic of $\boldsymbol{\theta}$

In this part we track the evolution of the parameter $\boldsymbol{\theta}$. For a model with fixed initialization and optimizer let $\boldsymbol{\theta}_i$ denotes the vector of trainable parameters for $i = 0, 1, ..., 20$. At epochs $0, 5, 10, 15, 20$ we plot the distance of $\boldsymbol{\theta}_i$ with the previous checkpoints, i.e. $\|\boldsymbol{\theta}_5 - \boldsymbol{\theta}_0\|, \|\boldsymbol{\theta}_{10} - \boldsymbol{\theta}_5\|, \|\boldsymbol{\theta}_{10} - \boldsymbol{\theta}_{15}\|, \|\boldsymbol{\theta}_{20} - \boldsymbol{\theta}_{15}\|$. This is illustrated in the top right subplot of Figure 3.

Moreover, we tracked the portion of parameters that remain fixed and those who change during the training and plotted their relative norm and frequencies. We also plot the histogram of both fixed and changing parameters at the initial and final step. These are all shown in the top middle and all the second row subplots of Figure 3.

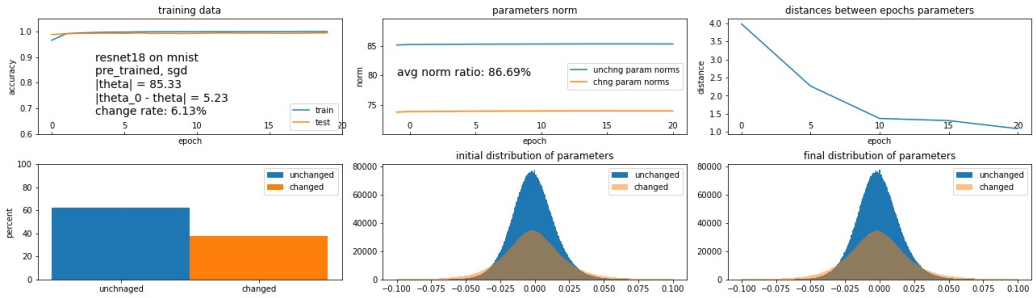

Figure 3: A sample visualization of part 1 analysis

The plot of part 1 analysis for all of our trained models can be found at this link (https://drive.google.com/drive/folders/16JFL6wOcv0unfHW5djpDowfmcgKX7DxH?usp=sharing).

2.6 Part 2: relation of extracted features

As we discussed previously, the features $f_{\boldsymbol{\theta},1}, ..., f_{\boldsymbol{\theta},n}$ can be seen as functions that map an image to a real number. If we sample an image, denoted by img, from the same distribution that our dataset is coming from, we can see $f_{\boldsymbol{\theta},i}(img)$ as a random variable. We study the correlation of features extracted by different trained models. We do so for all features extracted by one model, as well as pairwise comparison of features extracted by different trained parameter $\boldsymbol{\theta}$ and the pairwise comparison with features extracted by different architectures.

Note that since we do not have access to the exact distribution of images, we need to approximate the correlation of the features. Since we have a large sample from each distribution we can obtain the empirical correlation each two different features $f_{\boldsymbol{\theta}_1}, g_{\boldsymbol{\theta}_2}$ by the following formula as long as they are extracted from the same dataset.

$$Cov(f_{\boldsymbol{\theta}_1}, g_{\boldsymbol{\theta}_2}) \simeq \frac{1}{m} \sum_{i=1}^{m} (f_{\boldsymbol{\theta}_1}(img_i) - \bar{f_{\boldsymbol{\theta}_1}})(g_{\boldsymbol{\theta}_2}(img_i) - \bar{g_{\boldsymbol{\theta}_2}})^T,$$

where m is the number of our sample size and $\bar{f_{\boldsymbol{\theta}_1}} = \frac{1}{m} \sum_{i=1}^{m} f_{\boldsymbol{\theta}_1}(img_i)$, and $\bar{g_{\boldsymbol{\theta}_2}}$ is defined similarly.

The heat map of correlations between pairs of features in plotted in top left subplots of Figure 4.

In addition, we use the calculated correlations to associate features of each pair of models we have. The association map is plotted in the top middle subplots of Figure 4. We also rearrange the columns of correlation matrix so that associated features show up in the same row and column. This is shown in the top left subplots. In case one model has less features than the other, we leave the rest of columns (or rows) in the same order. We also plot the correlations between associated features in a descending order, and the boxplots of all correlations and associate correlations in the bottom subplots as shown in the same Figures as above.

Plots of part 2 analysis for all pairs of models we investigate can be found at this link.

2.7 Part 3: Mutual information study

We conducted an experiment for analysing Mutual information between features of different models. Like the correlation study we did earlier, we wanted to find the MI between features of one model with another, where both the models were trained on same data and share similar architecture but only differ by the way they're initialized. We decided to use the approach used for calculating mutual information in [1] as it is known to have provided the best estimates for Mutual information among the available estimators. But the problem with this is that, for getting the possible MI between features of 2 models which have feature size of 512 alone, we needed to train a network for MI around 130,086 times and expanding this study to all the combinations of models is very time consuming and also doesn't make sense. So we did a restricted study the following way. Amongst the different models we got by varying the initialization methods that we defined earlier, we found 3 types of models for every architecture type: model with highest accuracy, model with second highest accuracy and worst performing model on the test set. Among these 3 models, we get 3 pairs of combinations. In each pair, we pick the better performing model's feature which corresponds to the weight with highest l2 norm. We calculate the MI between this feature and all the other features of the weaker model. Through this, we focus our attention to some features of interest. Intuitively this makes sense because, we wanted to see how the features are related to the most important feature in the better performing model. We have plotted bar plots for the results obtained and a sample of this simple experiment is shown in figure 5. The other results can be found in the link. The plots can also be found in 28-33. In all these images, the topmost image contains the MI evaluated between the feature corresponding to the longest weight of the most accurate model and the features of second most accurate model. Similarly, the second image contains the MI evaluated between the feature corresponding to the longest weights of the second most accurate network and the all the features of the least accurate models. Lastly, the bottom image contains the MI between the feature corresponding to the most accurate model and all the features of the least accurate model. All the models that we considered are taken from the models trained on a particular dataset(details of which are mentioned in caption of each image).

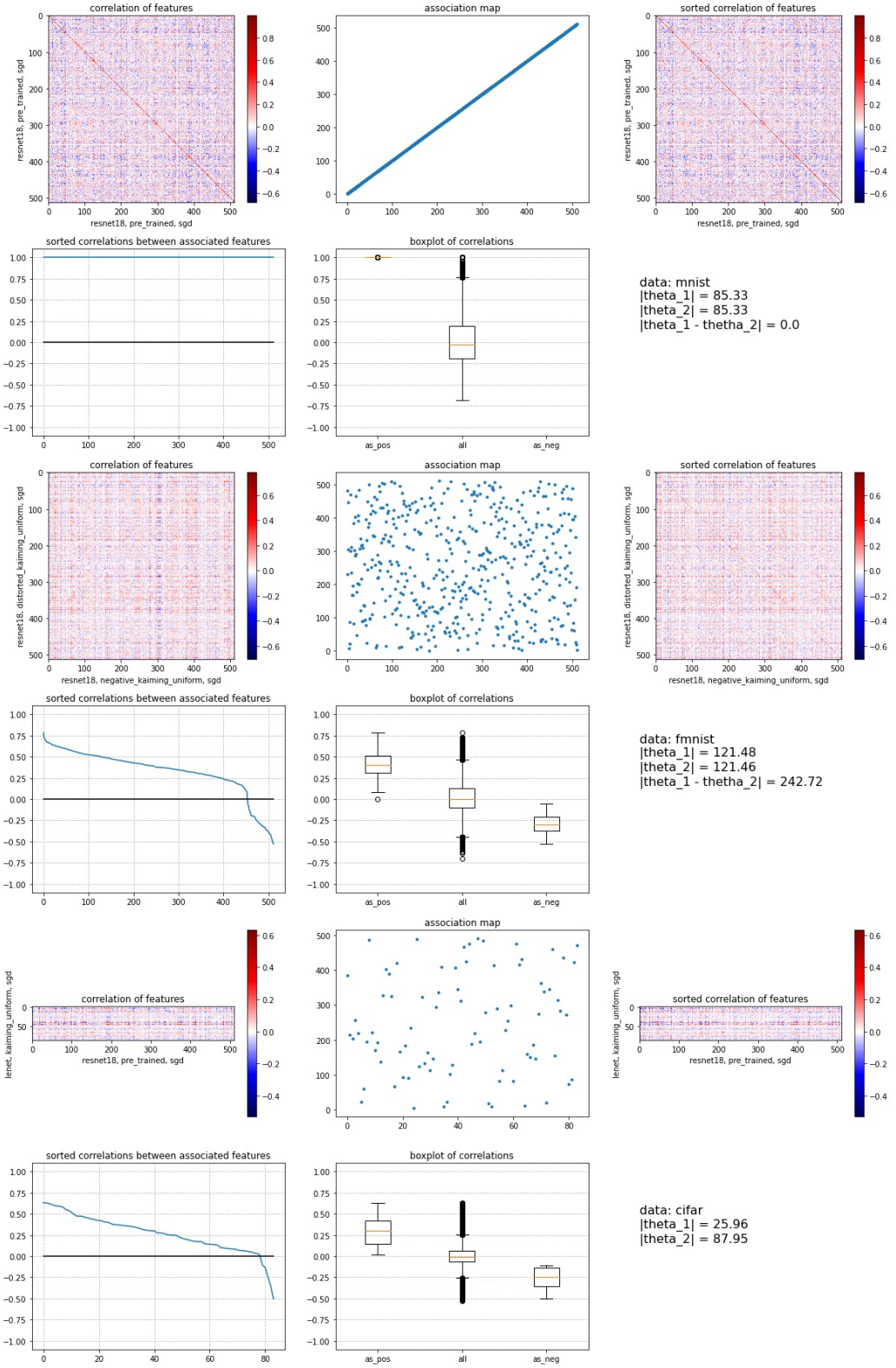

Figure 4: A sample visualization of part 2 analysis

2.8 Part 4: Visualizing features- Modified Grad-CAM

For a given unit in the feature representation of the bottle-neck layer, it is interesting to see what the network is seeing in the image to activate that

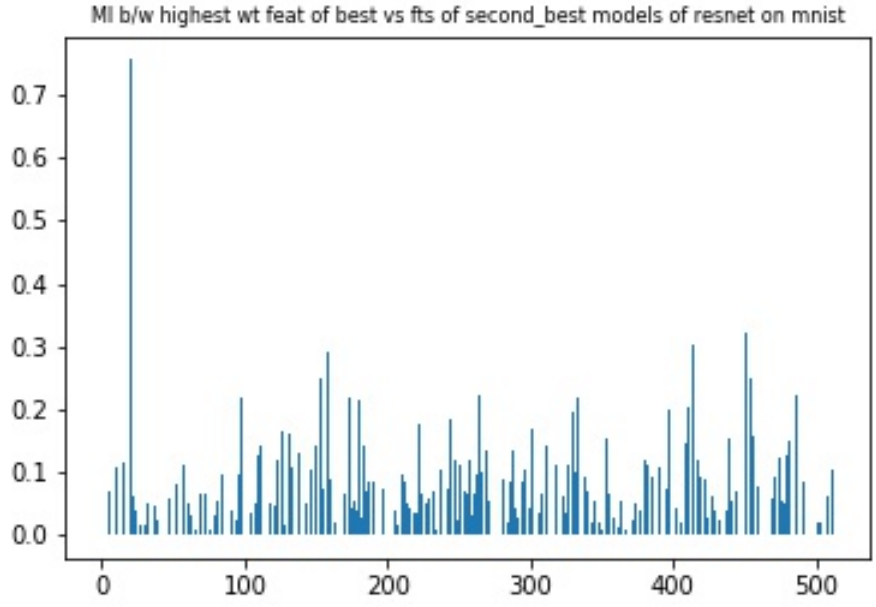

Figure 5: MI between highest l2 norm-ed feature of resnet model with the highest accuracy and features of the second most accurate resnet model

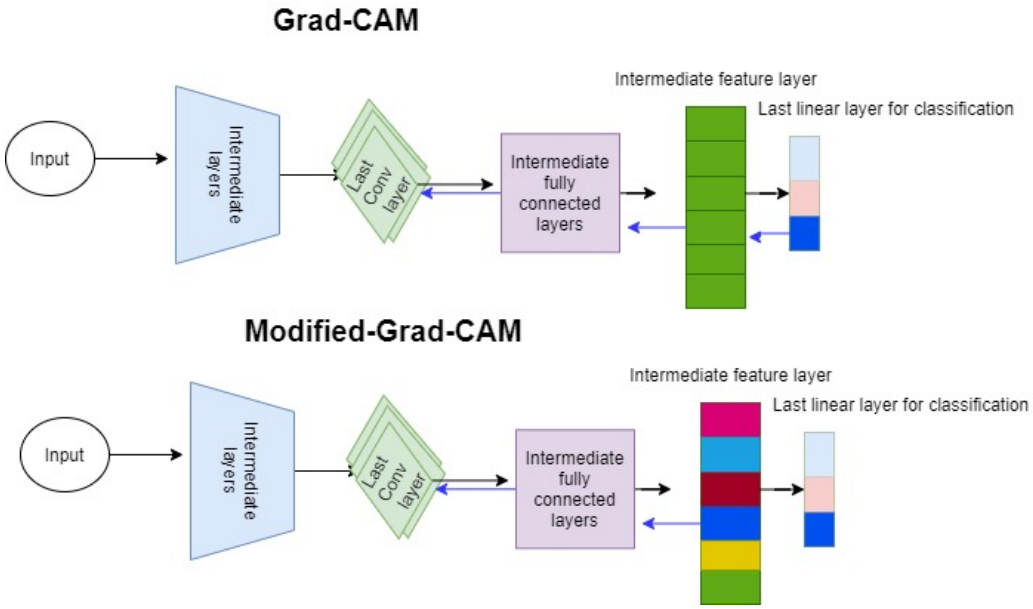

Figure 6: Modified gradcam flow chart. The black lines represent the forward propagation in the network. The Upper diagram is the grad-CAM[10] where the gradient with respect to the class-score represented by the unit in deep blue color in the last linear layer in the above diagram, flows through the last convolution layer. In our modified approach, the gradient is taken w.r.t to unit represented by the deep blue color in the intermediate feature layer. The gradients are represented by the arrows in blue.

particular neuron in the bottle-neck layer. It is believed that the final convolution layer has a greater information about the perspective of the whole

network [10]. To solve this problem, we slightly modified the Grad-CAM's[10] approach. We observe the gradients that flow from the feature unit of interest into the last convolution layer of the deep network to understand the importance of each neuron in modeling a particular feature unit. This is how our method is different from Grad-CAM[10]: Grad-CAM observes the gradients from the class-score flowing into the last convolution layer. With this minor modification, we'll get a localization map similar to that in Grad-CAM but we get a localization map for every neuron in the bottleneck layer. We wanted to visualize what regions in the inputs are focused by the network for a given unit of the feature representation. Similar to grad-CAM, in a single shot, we backpropagate from the feature unit till the last convolution layer. The localization map: $L_{Modified-gradcam} \in \mathbb{R}^{u \times v}$ is obtained by taking the ReLU over the linear combination of activation maps of the convolution layer. The weights used for each activation map A_k in the linear combination is the mean of all the gradients of that channel. To write formally, for a conv layer with K activation maps with A_k being the k^{th} activation map, the localization map $L_{Modified-gradcam}$ with respect to l^{th} feature unit in the bottleneck feature f (denoted by f_l)is:

$$L_{Modified-gradcam} = ReLU(\sum_{i=1}^{K} \alpha_k^l A_k) \text{ where,}$$

$$\alpha_k^l = \frac{1}{uv} \sum_{i=1}^{u} \sum_{j=1}^{v} \frac{\partial A_{ij}}{\partial f_l}$$

Just like in Grad-CAM, we super impose this localization map onto the image to understand how the network is interpreting the image. We performed this experiment on LeNet successfully. [8]). A sample of the visualization can be seen in 7 With deeper networks like ResNet[5], we observed zero gradients and hence the weights of the linear combination turned out to be NaN values. Some models of LeNet we trained also exhibited this trend. The images in 7 with no heat map are such images.

2.9 Results

Using Google Colab, We train 126 models and compare the correlation and mutual information between each pairs of extracted features. This results in 2917 pairwise correlation plots which all can be found at this link. The developed tool and massive generated data give us useful information that can be used for theoretical analysis and for gaining insight about the dynamic of training a neural network. We admit that the possibility of analysis on the data we generated is much beyond what we have done so far. We summarize our observations and analysis in this section and leave further analysis to future works. Due to the limit in space, for each point we present a few supporting plots. Nevertheless, all plots we made can be found at here (https://drive.google.com/drive/folders/1dcxBQ74iXRHCq5cV0-w-nLjLUncd04c5?usp=sharing).

1. Challenging level of datasets: MNIST < Fashion MNIST < CIFAR10

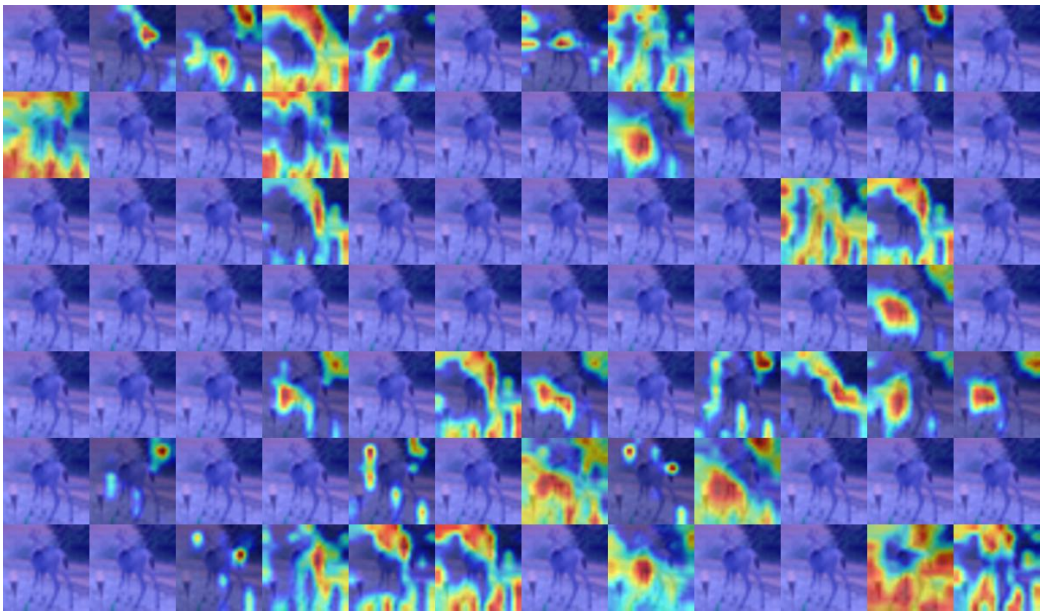

Figure 7: visualization of the localization maps of the last convolutional layer obtained with respect to the 84 feature units of LeNet. The sub-images with no heat map are the ones obtained with zero gradients through the last convolutional layer.

In general, models could handle classifying the MNIST data easier and much faster. Even at the initial point with completely random weights most model could achieve over 80% accuracy in classification. At the next level, Fashion MNIST was quite easy dataset for models to classify. CIFAR10 was the most challening dataset, and some models with some of our initializations could not get accuracy above 75% after 20 epochs of training. Figures 8, 9, and 10 present some plots that support this observation.

Given the simplicity of single channel images in MNIST and Fashion MNIST datasets, this observation is quite natural.

2. We observed that in training Resnet models, almost 60% of parameters remain unchanged, regardless of the initialization, the optimizer, and the data on which the model is trained. See Figure 11. Moreover, more than 87% of the norm of the vector of parameter $\boldsymbol{\theta}$ corresponds to those parameters that were changing, while they are less than half of all parameters. That is if we partition $\boldsymbol{\theta} = \left(\boldsymbol{\theta}_{changing}, \boldsymbol{\theta}_{fixed}\right)$ we have:

$$\frac{dim(\boldsymbol{\theta}_{changing})}{dim(\boldsymbol{\theta})} \leq 0.4, \quad \frac{\left\|\boldsymbol{\theta}_{changing}\right\|_2}{\|\boldsymbol{\theta}\|_2} \geq 0.87,$$

in most of the cases.

While this can be explained by lazy training phenomenon discussed in [2], we so not see the same thing about VGG model which is both deeper and wider than Resnet. In fact, the rate of unchanged parameters in VGG is varying notably depending on the initial point. Figure 12 denotes this fact.

Lenet as a shallow model do not show any tendency to keep some parameter fixed. The same behavior of changing parameters observed in Figure 8 can be seen in all other settings, regardless of the initial point, optimizer, and dataset.

3. For huge architectures such as Resnet and VGG, SGD usually finds a local mimina very close to any initial point $\boldsymbol{\theta}_0$. ADAM on the other hand has a tendency to find further local minimas. Interestingly, this does not have a notable effect on the performance of the trained model and models trained with SGD and ADAM in the same setting perform more or less the same. The closeness of the local minima is measured based on the norm of the parameter $\boldsymbol{\theta}$ and will be reported in the top left subplot of all of our plots. The distance that parameter $\boldsymbol{\theta}$ moves at selected epochs is also plotted at the top right subplots. Note that the scale of the $y-$axis for this subplot will automatically change with the the values of distance. While the relative distance of the local minima found by SGD is usually less than 10% of the norm of the $\boldsymbol{\theta}$, this relative distance is notably higher for the minima found by ADAM. This is illustrated in Figures 13 and 14.

   This is in fact an experimental illustration of the existence of exponentially many local minimas in terms of number of parameters ($dim(\boldsymbol{\theta})$), because for each random initial point, SGD can find a local minima in a small neighborhood of it.

4. We noticed that when our huge architectures are trained by ADAM optimizer, the extracted features have stronger positive correlations among each other as compare to trained models with SGD. This phenomenon is not does not happen for the small network Lenet. Figures 15, 16, and 17 display this observation by plotting the correlation matrix of features when we trained a model with same initial setting and by the two optimizers.

5. As expected, the small distortion that we made to a train model does not usually change the model and features to much. In fact, when retrained with SGD, all the models more or less come back to the initial trained model. This can be seen both with the distance of vector of parameters and very high correlation (almost 1) of features with an identity association map. However, we observed on exception for this normal behavior. Retraining a distorted a Resnet model with ADAM resulted in a very far parameter vector! Moreover, we observed many nan values in our correlation matrix for this case. The reason of this unpleasant phenomenon is still unclear to us. Figures 18 - 22 denote this observation.

6. By flipping the sign of a trained parameter ($\boldsymbol{\theta} \mapsto -\boldsymbol{\theta}$) and retrain it we forced the network to find a local minima far away from a trained one. Naturally, we have seen irregular association map for the features of one model and its negative trained version. However, there are two surprising similarity for the correlations of associated maps:

- While features are associated with absolute value of correlations, for Resnet and Lenet architectures most associated features show a strong positive correlation
- For VGG architecture the curve of correlations between associated features seem symmetric with respect to 0
- Regardless of the dataset on which the network is trained, we see a similar curve for correlation of associated features

This is illustrated in Figures 23 - 27.

7. Though not fully robust, our approach on visualizing the conv-layer's activation for a given feature unit can be considered a decent starting point to understand the convolutional layer's reasoning for assigning a particular value for a given feature unit. We would be keen on analysing why we are getting zero gradients. One possible fix to this problem is to adjust the gradient values by adding a small value.

8. Our experiments on MI made apparent the supremacy of SGD over ADAM in relation to convergence. On all the 3 datasets, the most accurate models(of both LeNet and ResNet) are all found to have been trained using SGD.

9. One more interesting pattern we found was that whenever there is a MI calculation involved between a model and the model obtained by starting the training by initializing with parameters obtained by distorting the original model, the scale of MI is higher(Observe top-most images in 28, 31, 32, 33) and bottom images in 32, 33. All these images have MI values of order $1E-2$. This observation corroborates with our finding in point 5.

References

[1] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *International Conference on Machine Learning*, pages 531–540. PMLR, 2018.

[2] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *arXiv preprint arXiv:1812.07956*, 2018.

[3] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[8] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[9] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, 2015.

[10] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

A Links to our data

One may access to all the data we generated and source codes by clicking on the following links while he/she is logged in to LionMail.

1. Codes
2. Trained models
3. Images

Figure 8: Lenet with kaiming-uniform initialization

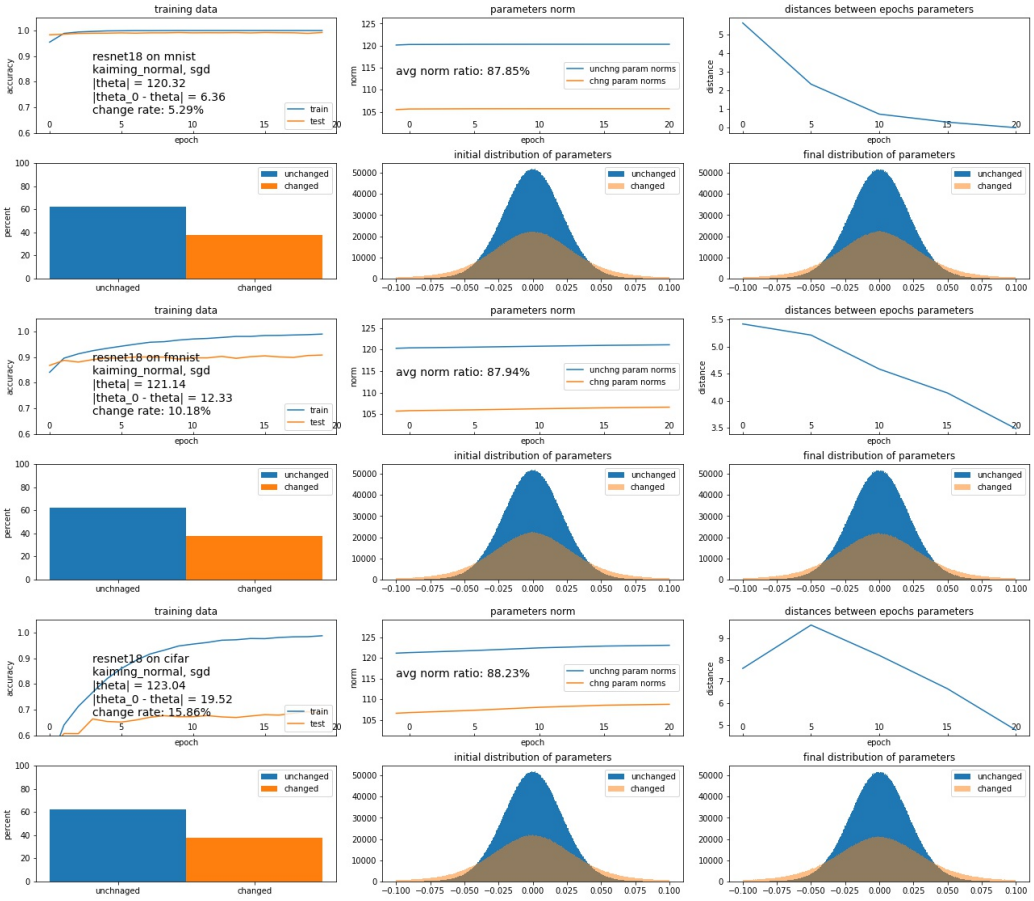

Figure 9: Resnet with kaiming-normal initialization and SGD optimizer

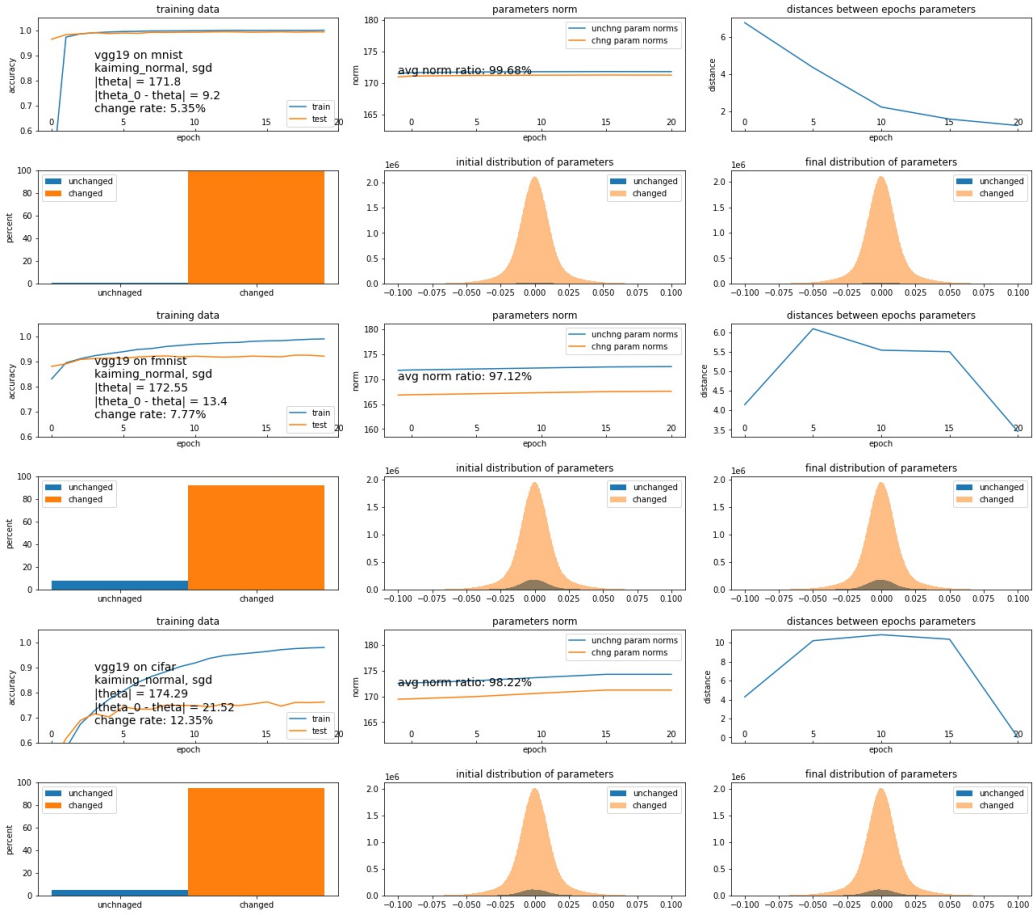

Figure 10: VGGnet with kaiming-normal initialization and SGD optimizer

Figure 11: Resnet models in different settings: 60% of parameters remain unchanged

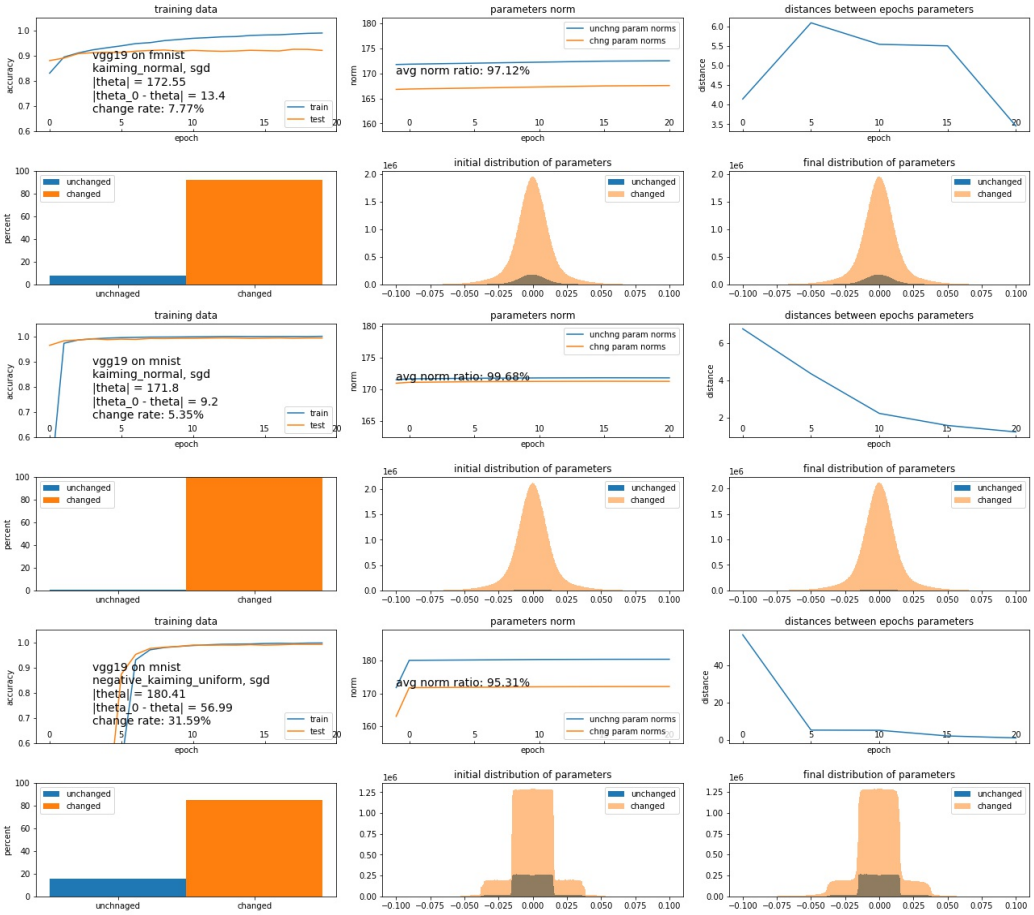

Figure 12: VGG with three initializations: the rate of unchanged parameters varies from 0 to 20%

Figure 13: Comparison of the test accuracy and the movement of parameter under SGD and ADAM for Resnet model

Figure 14: Comparison of the test accuracy and the movement of parameter under SGD and ADAM for Lenet and VGG models

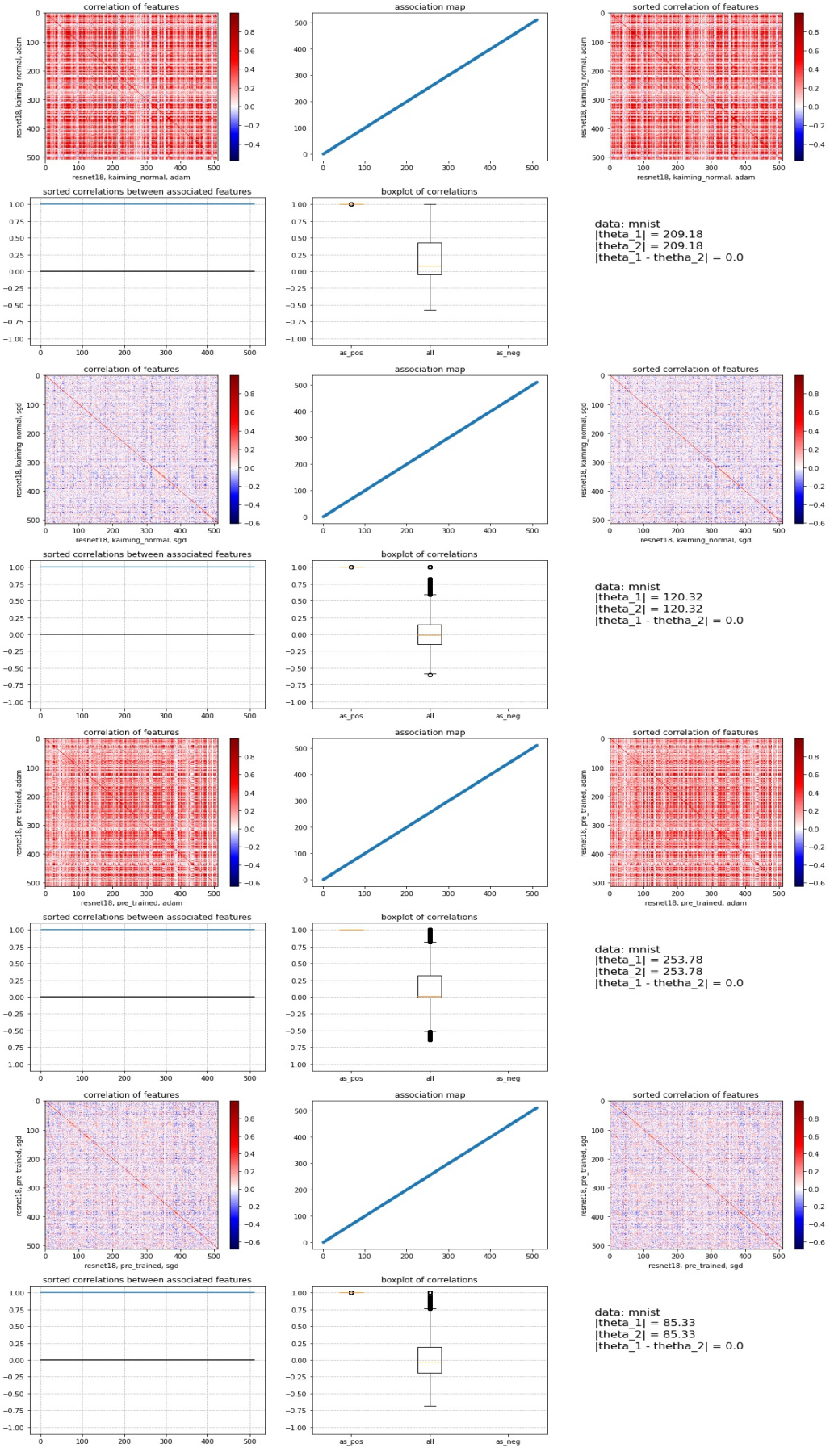

Figure 15: Correlation of Resnet features trained with ADAM and SGD

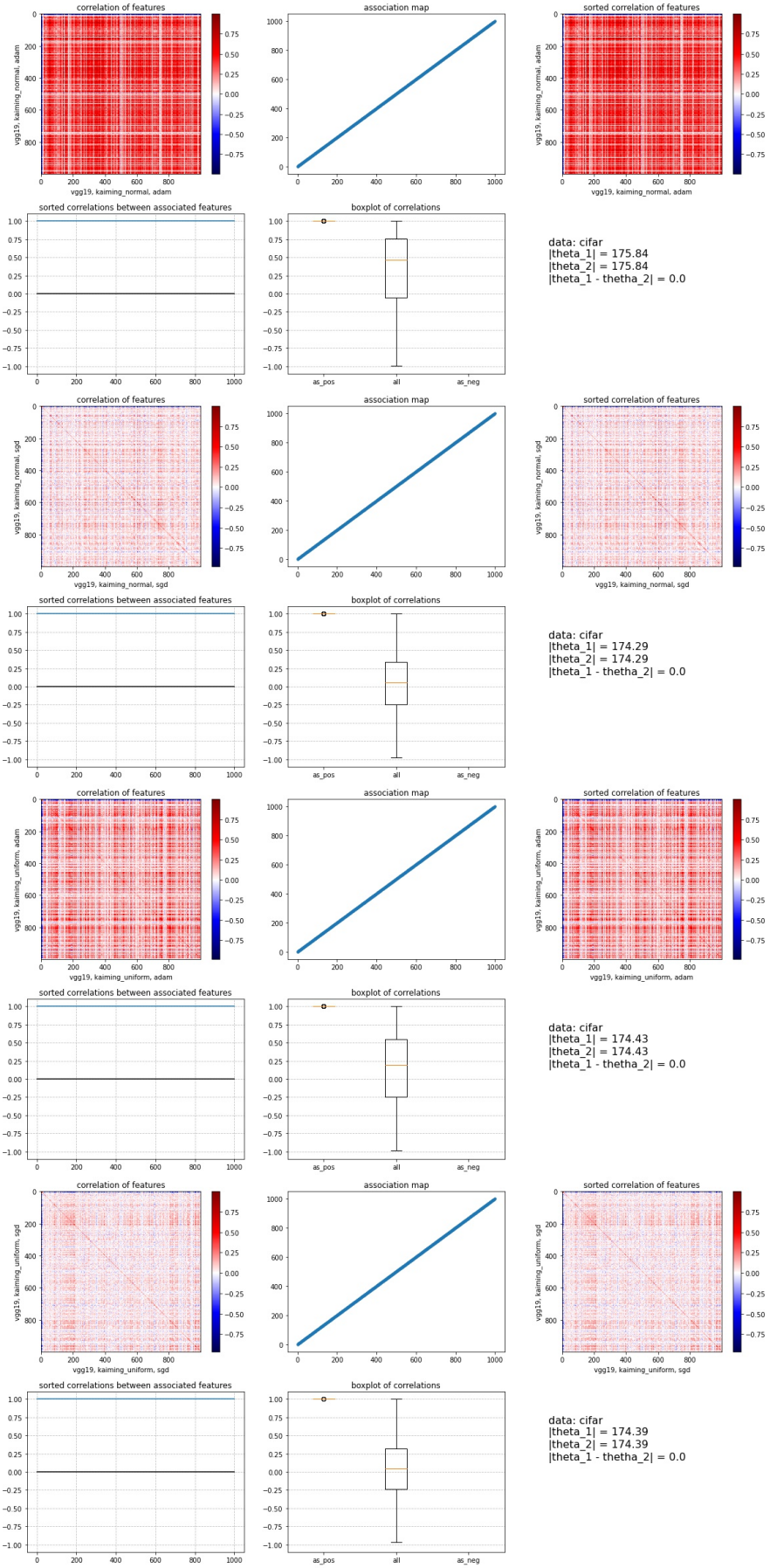

Figure 16: Correlation of VGG features trained with ADAM and SGD

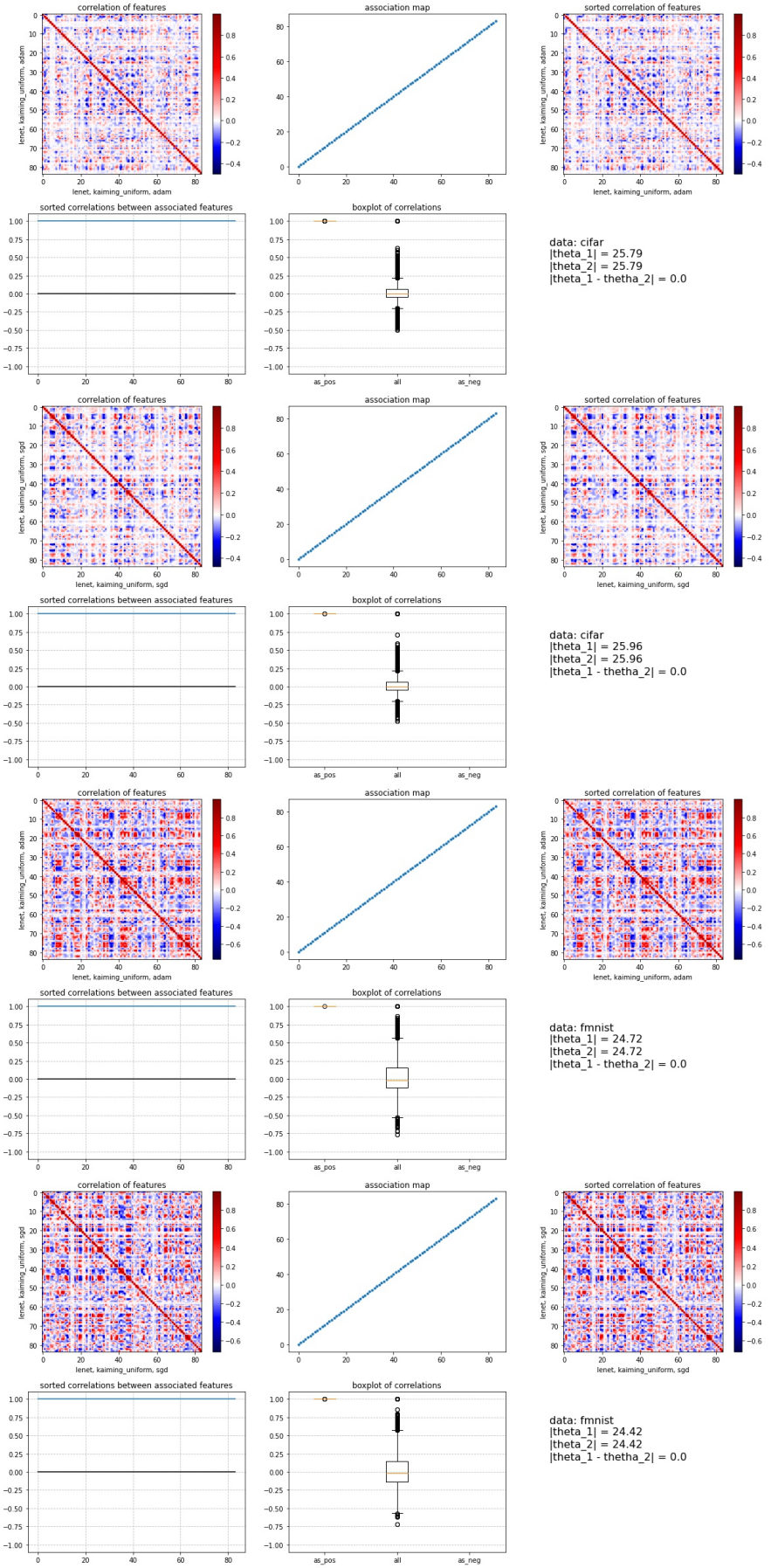

Figure 17: Correlation of Lenet features trained with ADAM and SGD

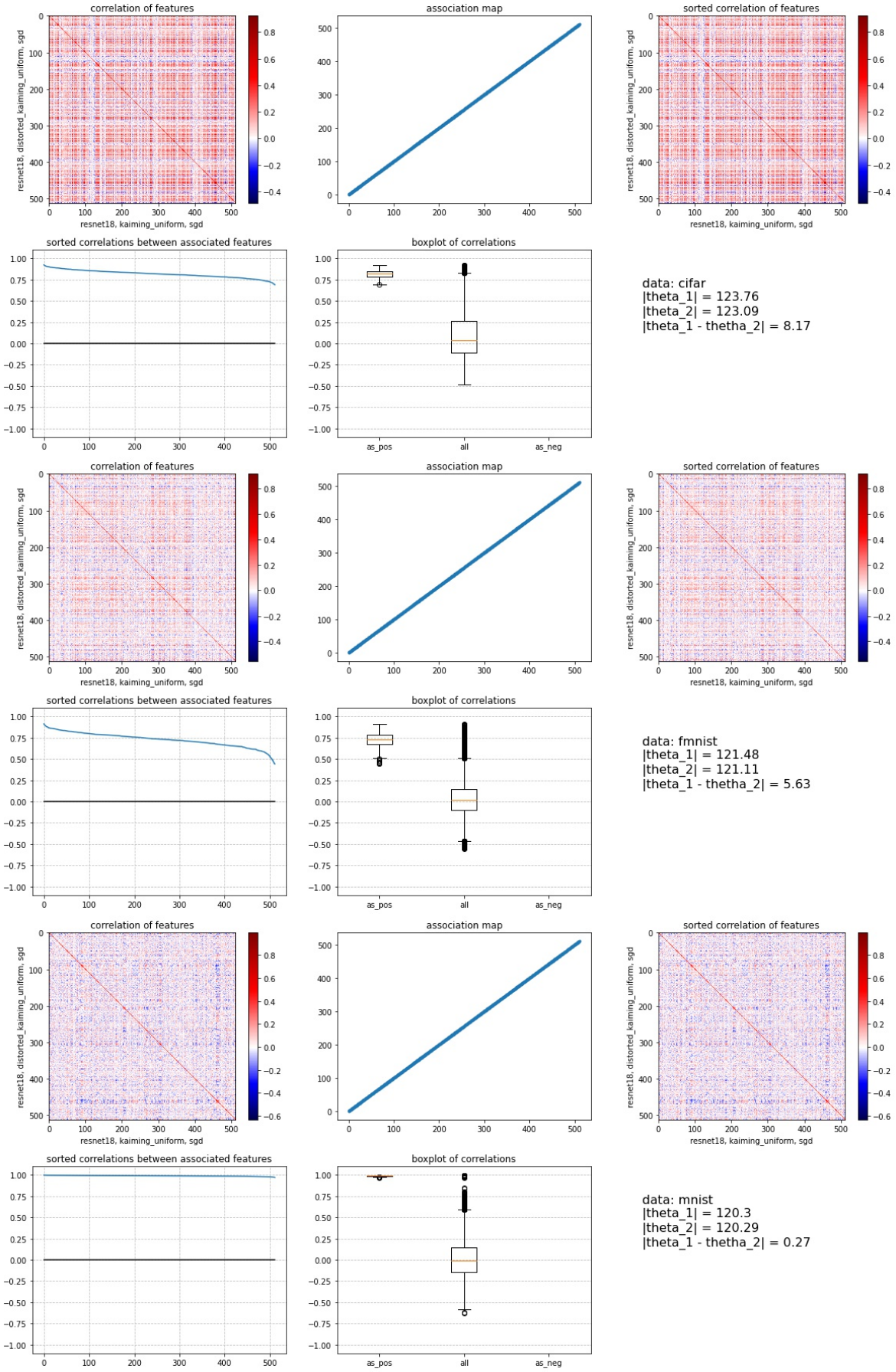

Figure 18: Effect of distortion on Resnet while retrain with SGD

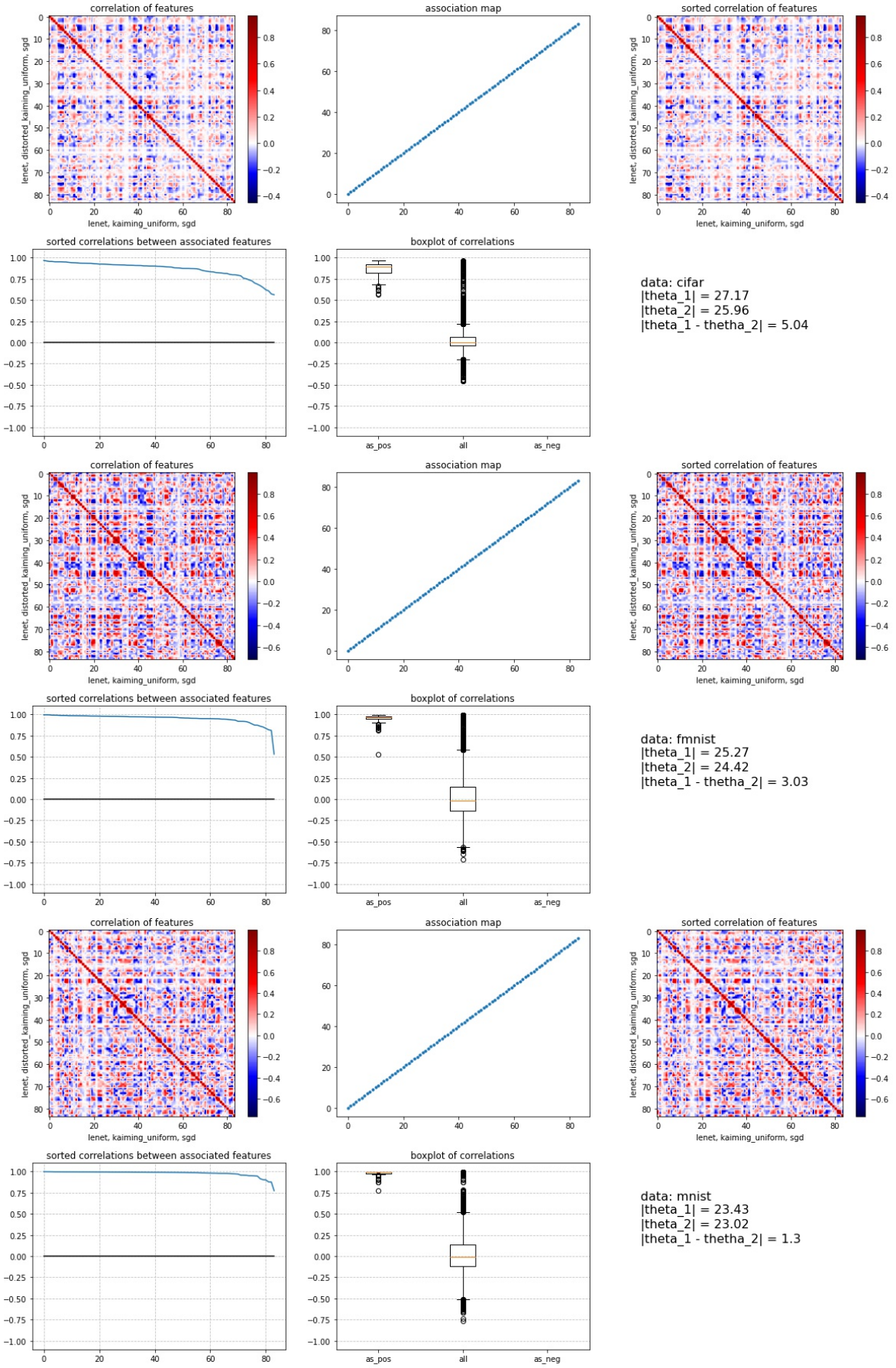

Figure 19: Effect of distortion on Lenet while retrain with SGD

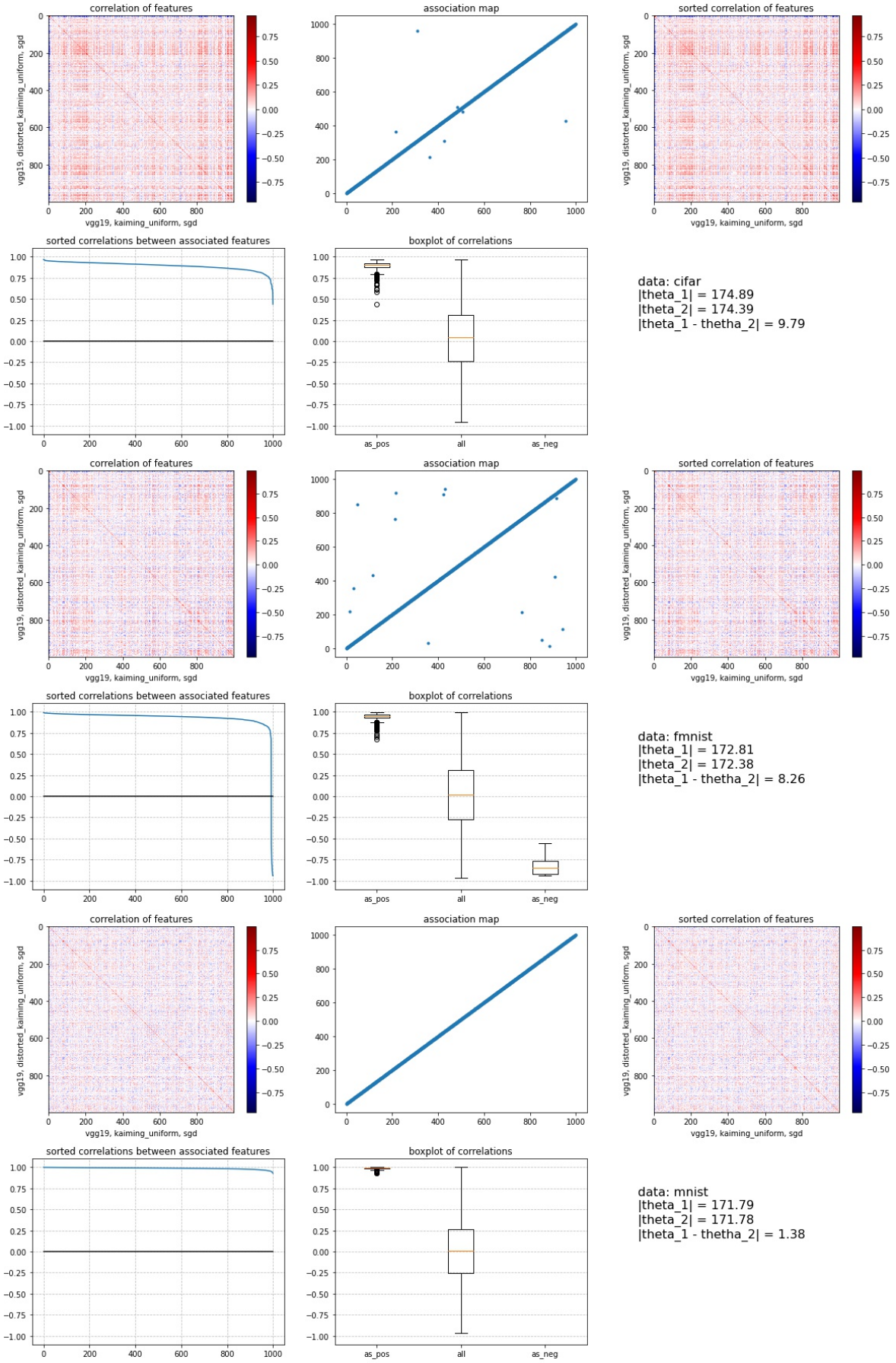

Figure 20: Effect of distortion on VGG while retrain with SGD

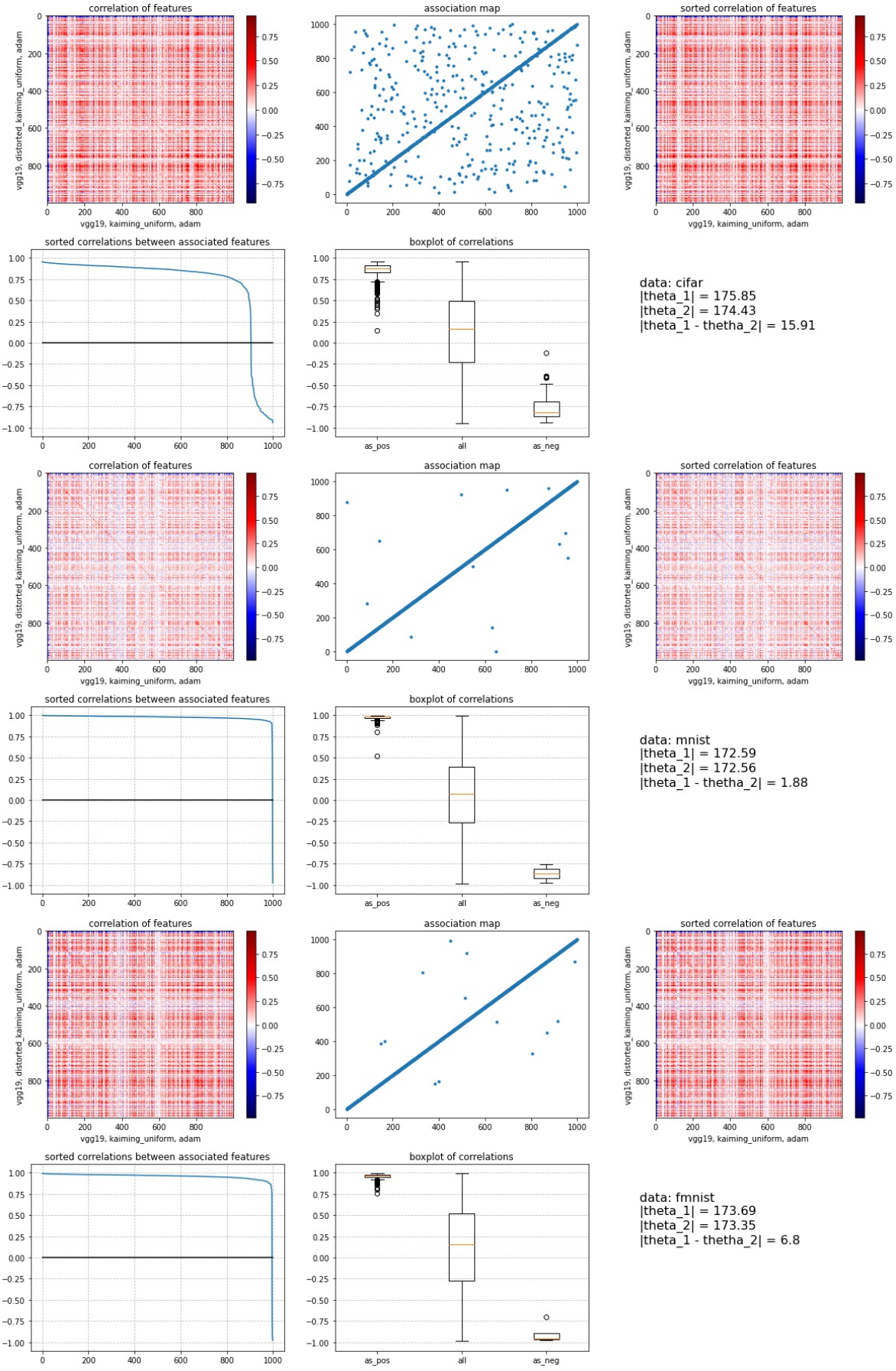

Figure 21: Effect of distortion on VGG while retrain with ADAM

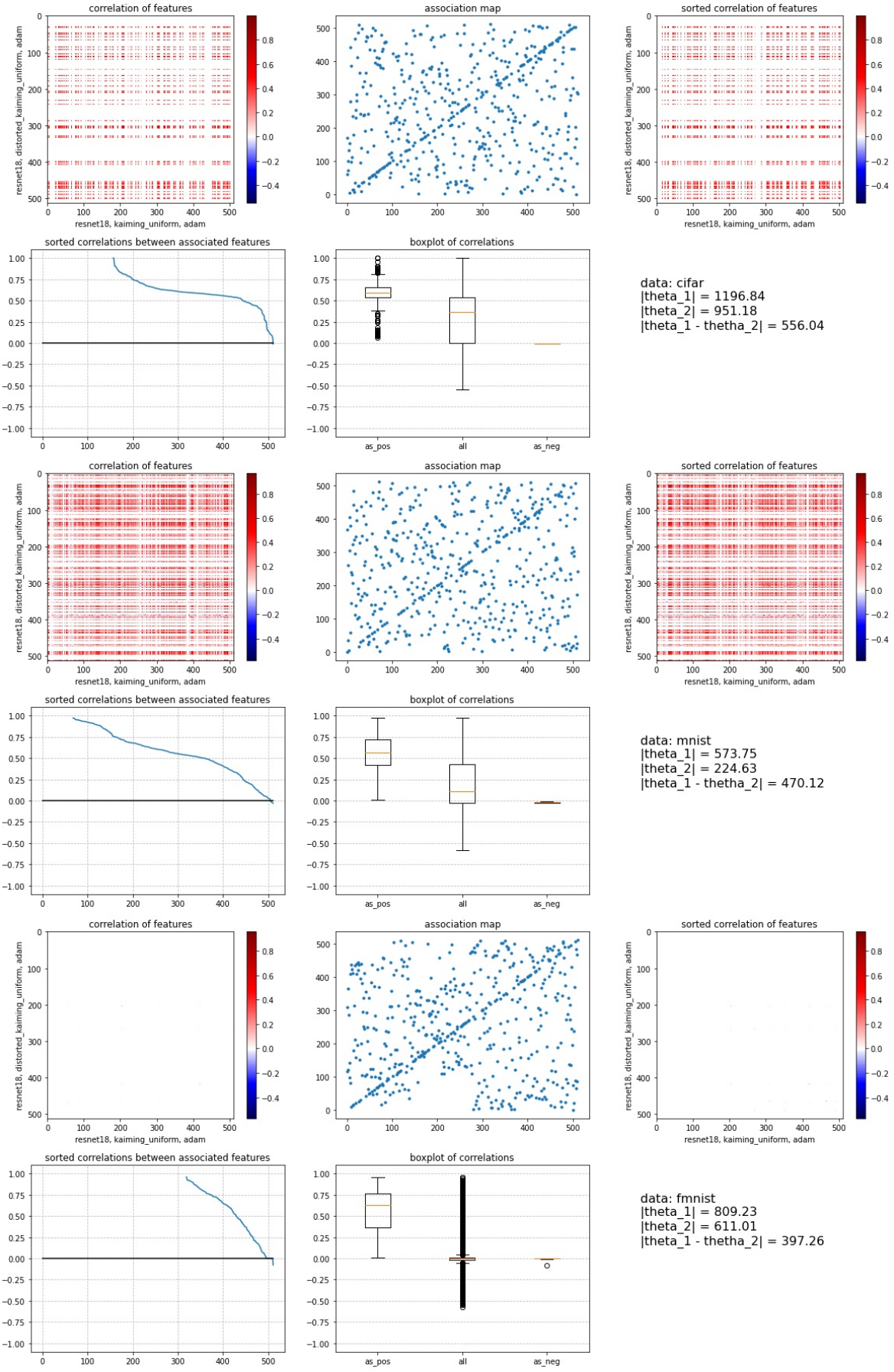

Figure 22: Unusual effect of distortion on Resnet while retrain with ADAM

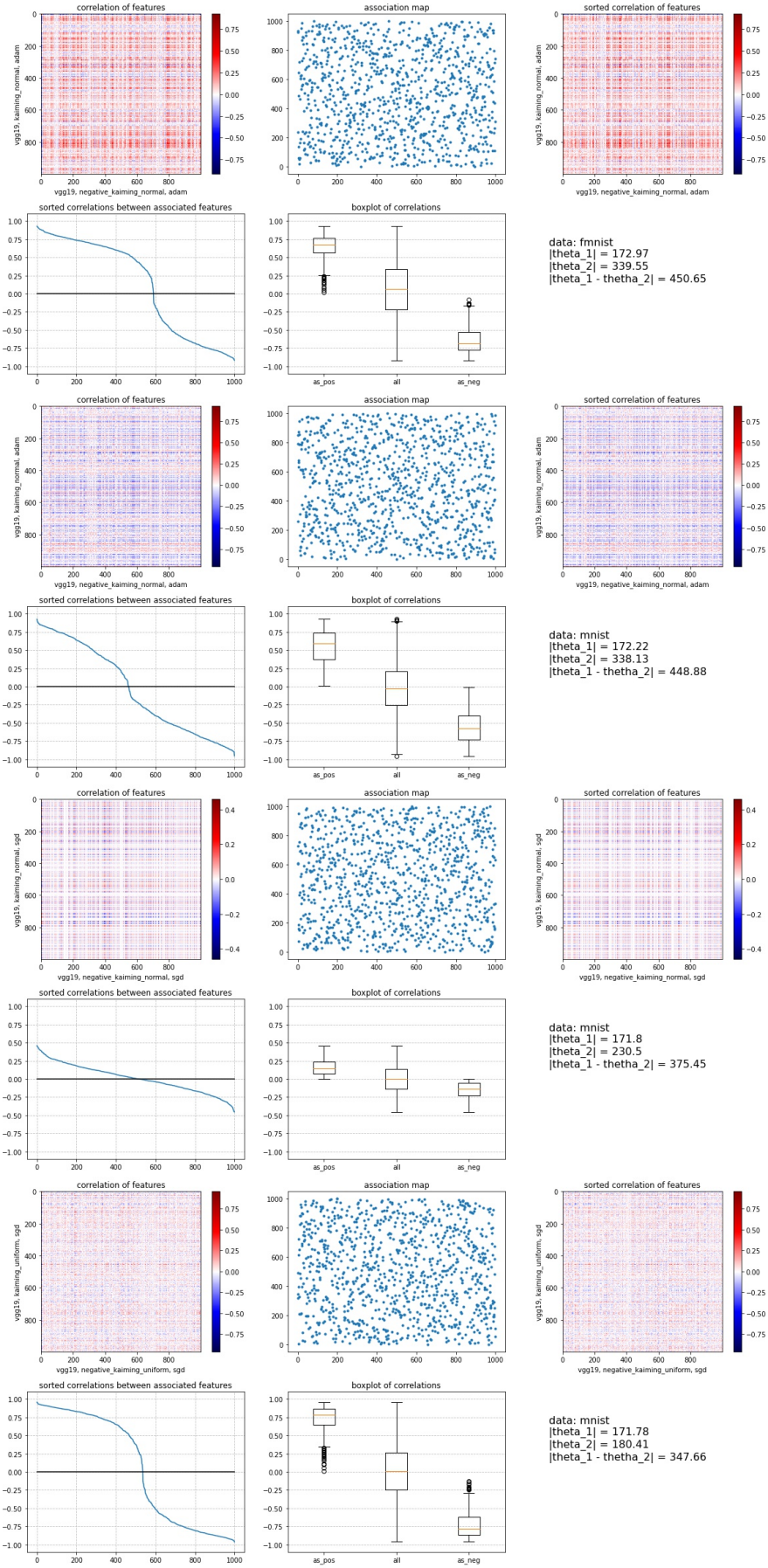

Figure 23: Correlation of associated features for negative VGG

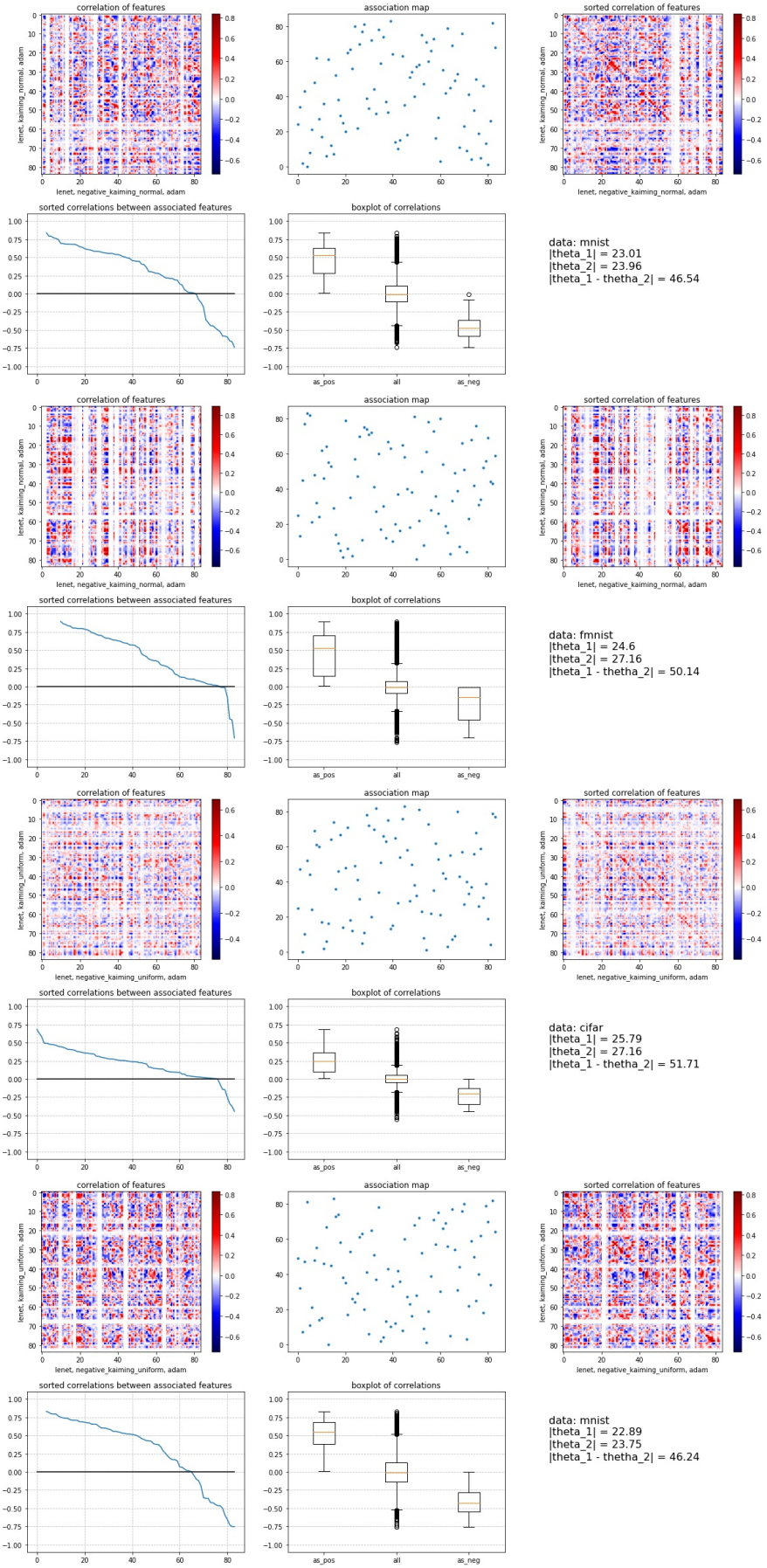

Figure 24: Correlation of associated features for negative Lenet

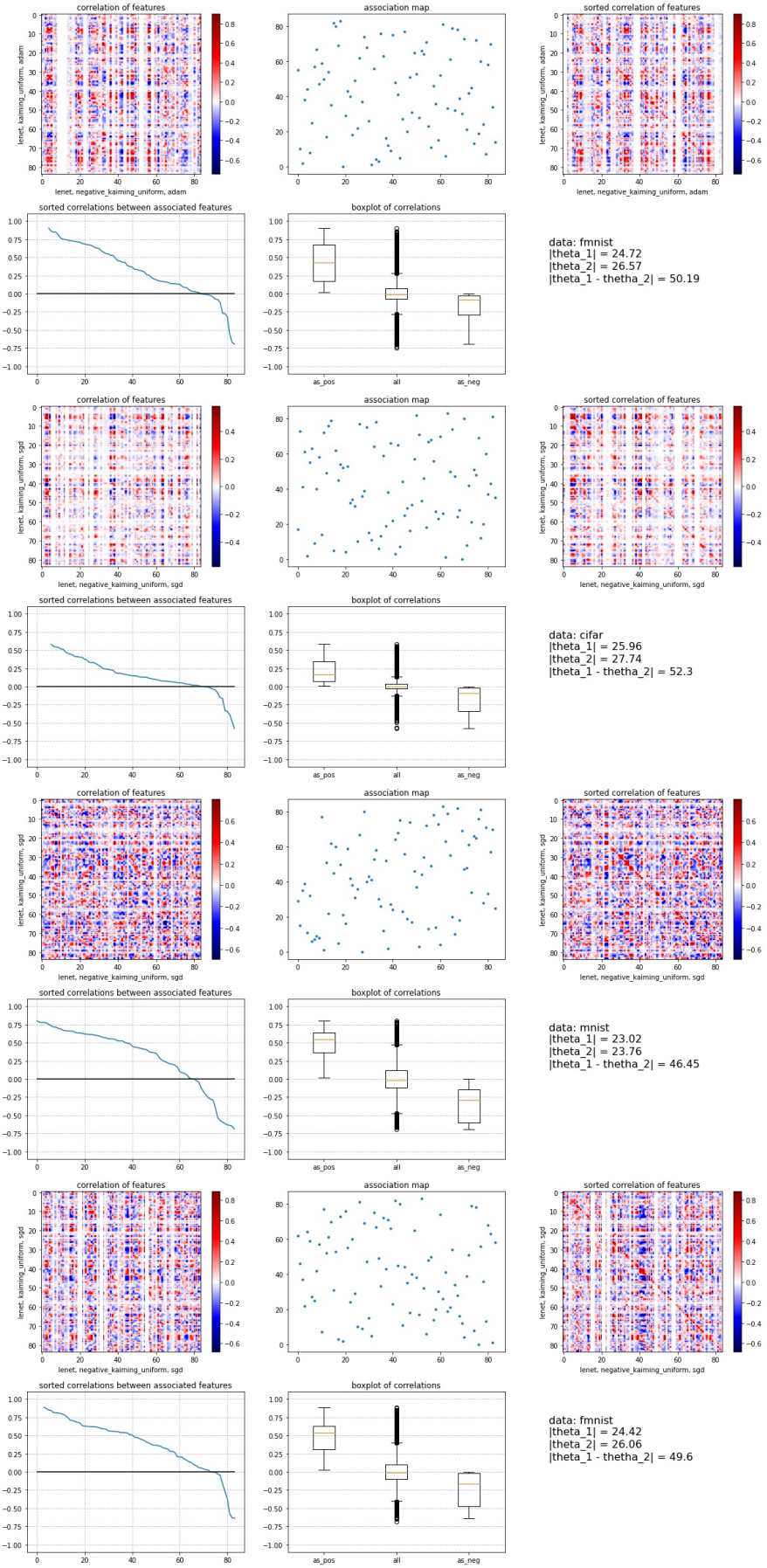

Figure 25: Correlation of associated features for negative Lenet

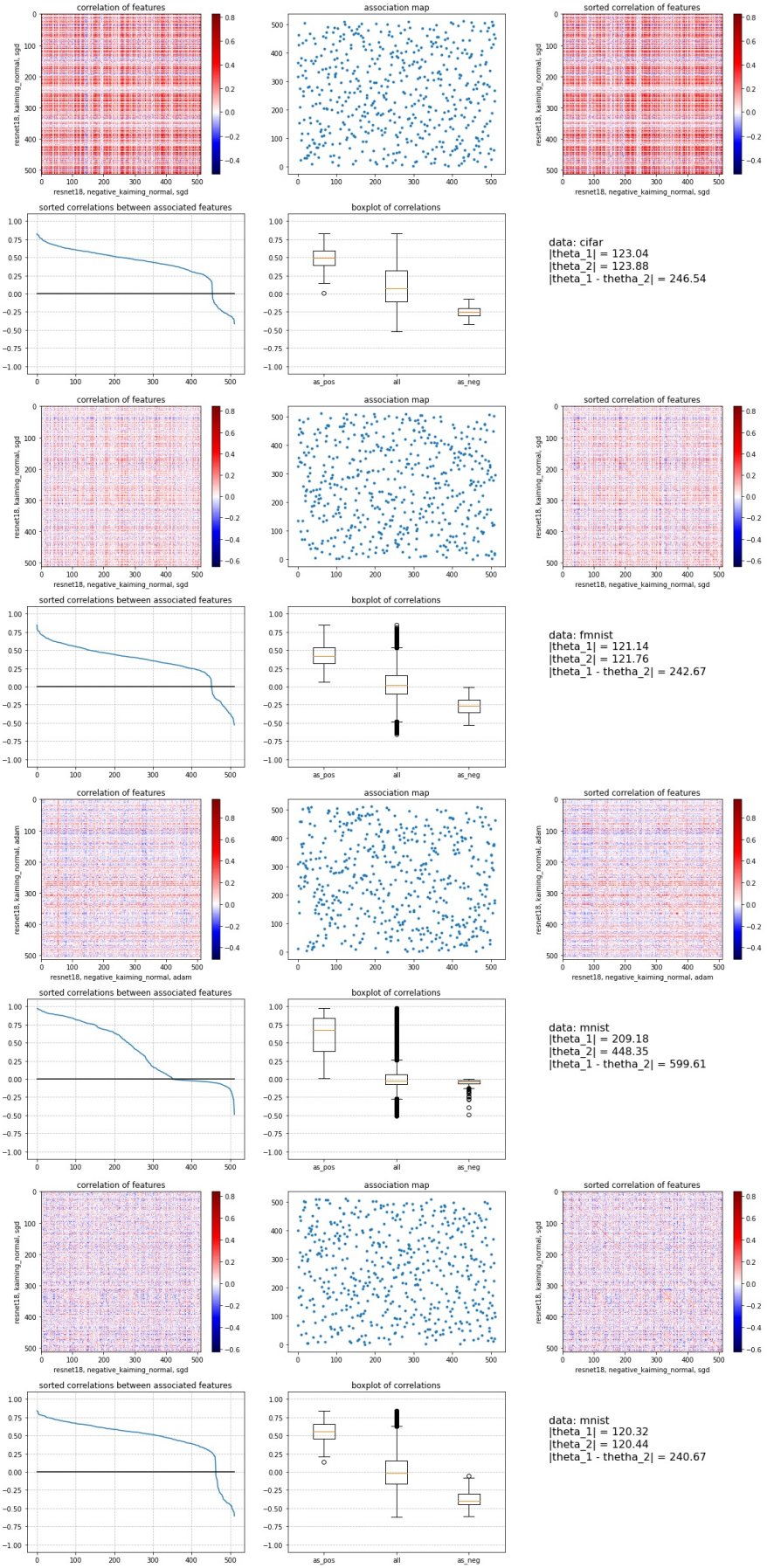

Figure 26: Correlation of associated features for negative Resnet

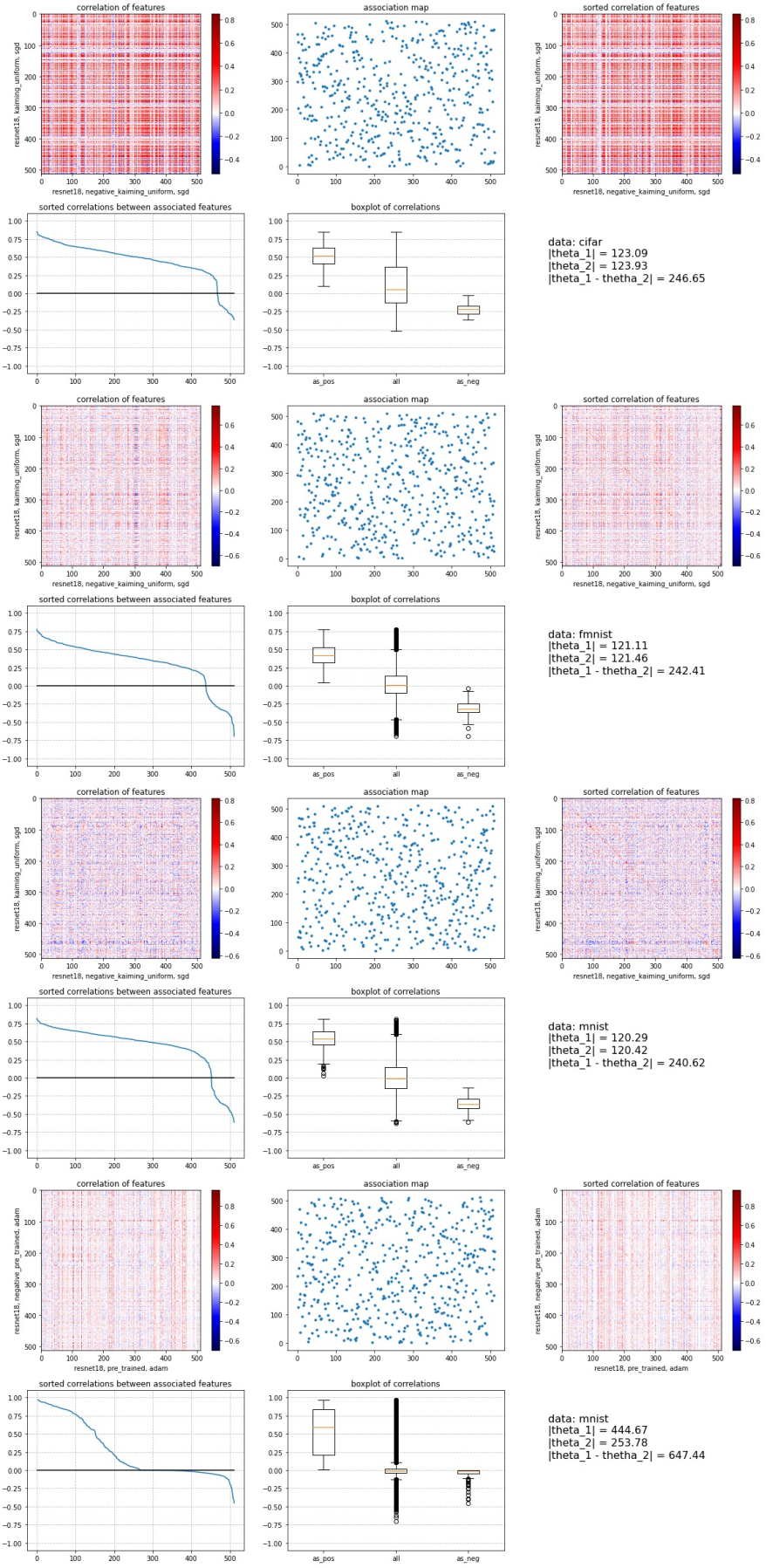

Figure 27: Correlation of associated features for negative Resnet

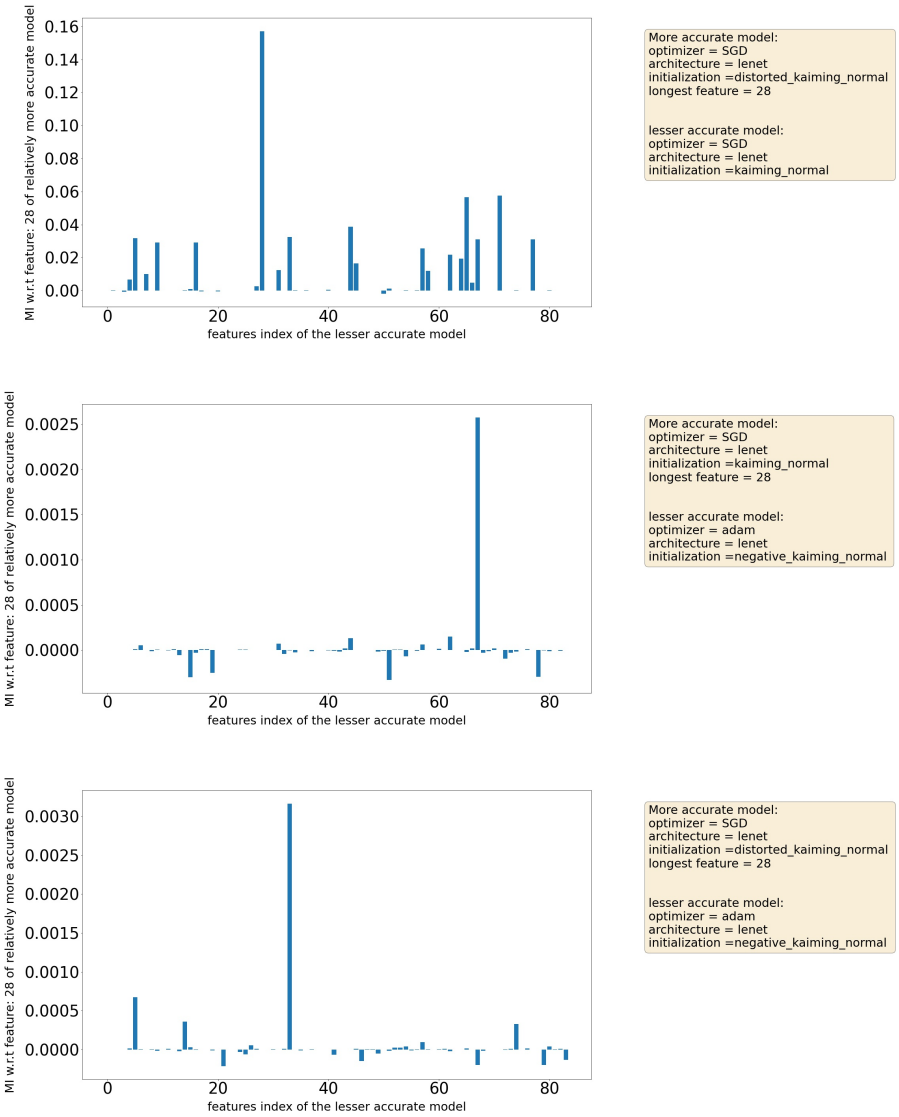

Figure 28: visualization of Mutual information experiment on a LeNet model on MNIST data. To the right of each plot are the details of the models used.

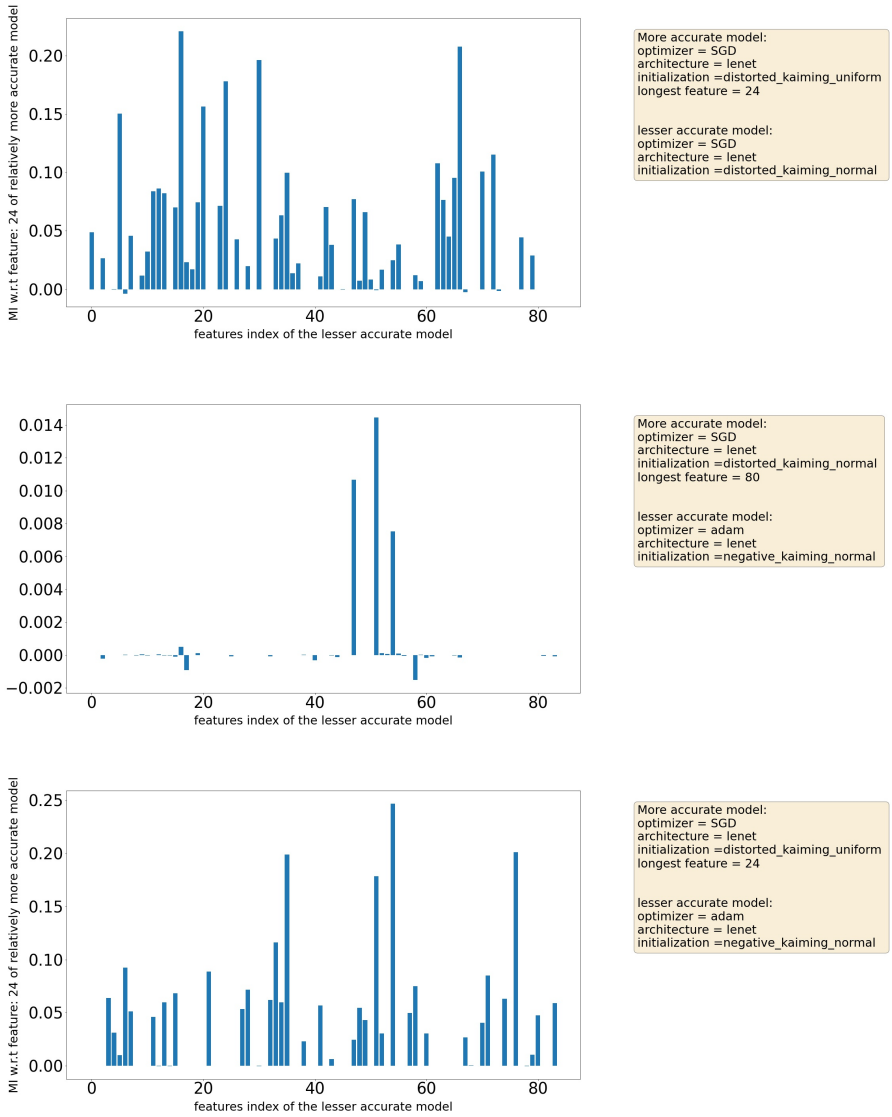

Figure 29: visualization of Mutual information experiment on a LeNet model on FMNIST data. To the right of each plot are the details of the models used.

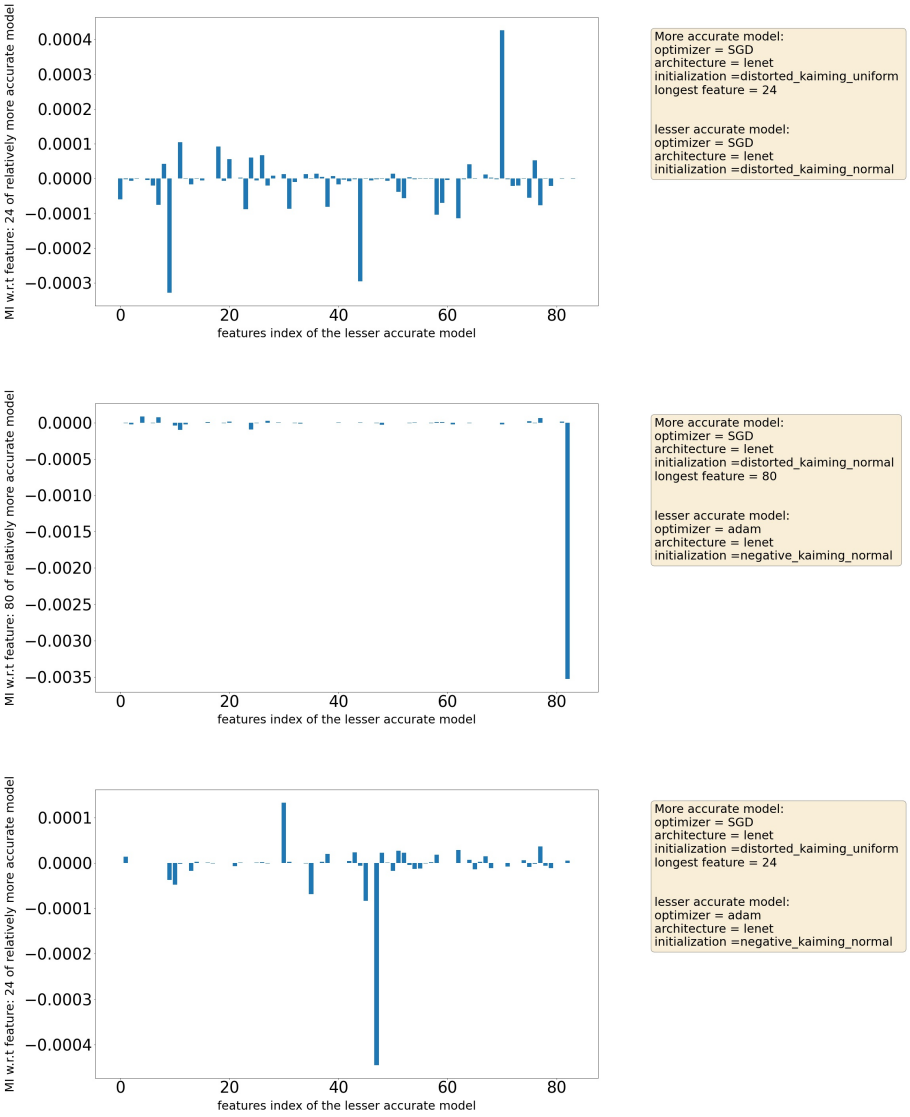

Figure 30: visualization of Mutual information experiment on a LeNet model on CIFAR10 data. To the right of each plot are the details of the models used.

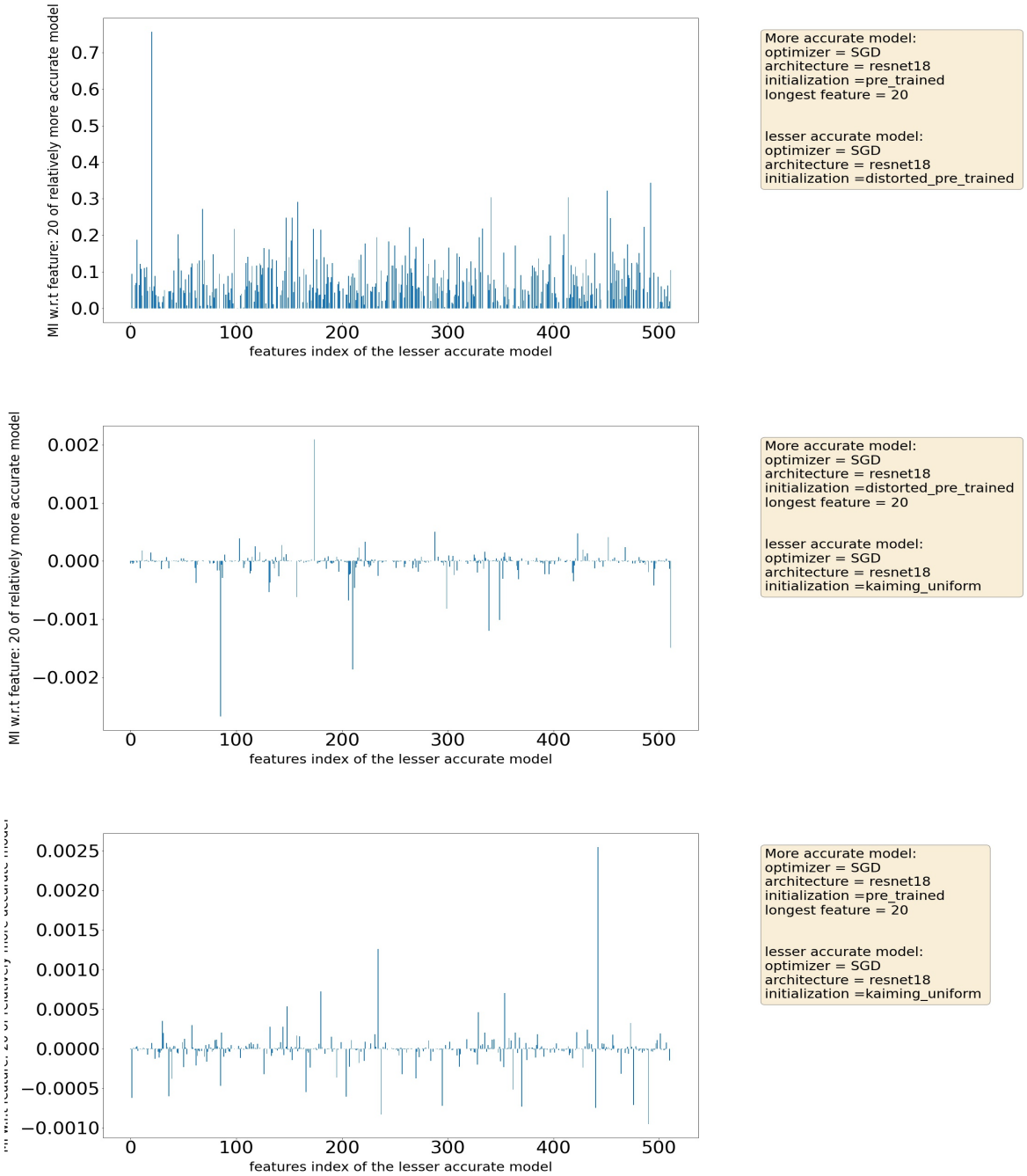

Figure 31: visualization of Mutual information experiment on a ResNet model on MNIST data. To the right of each plot are the details of the models used.

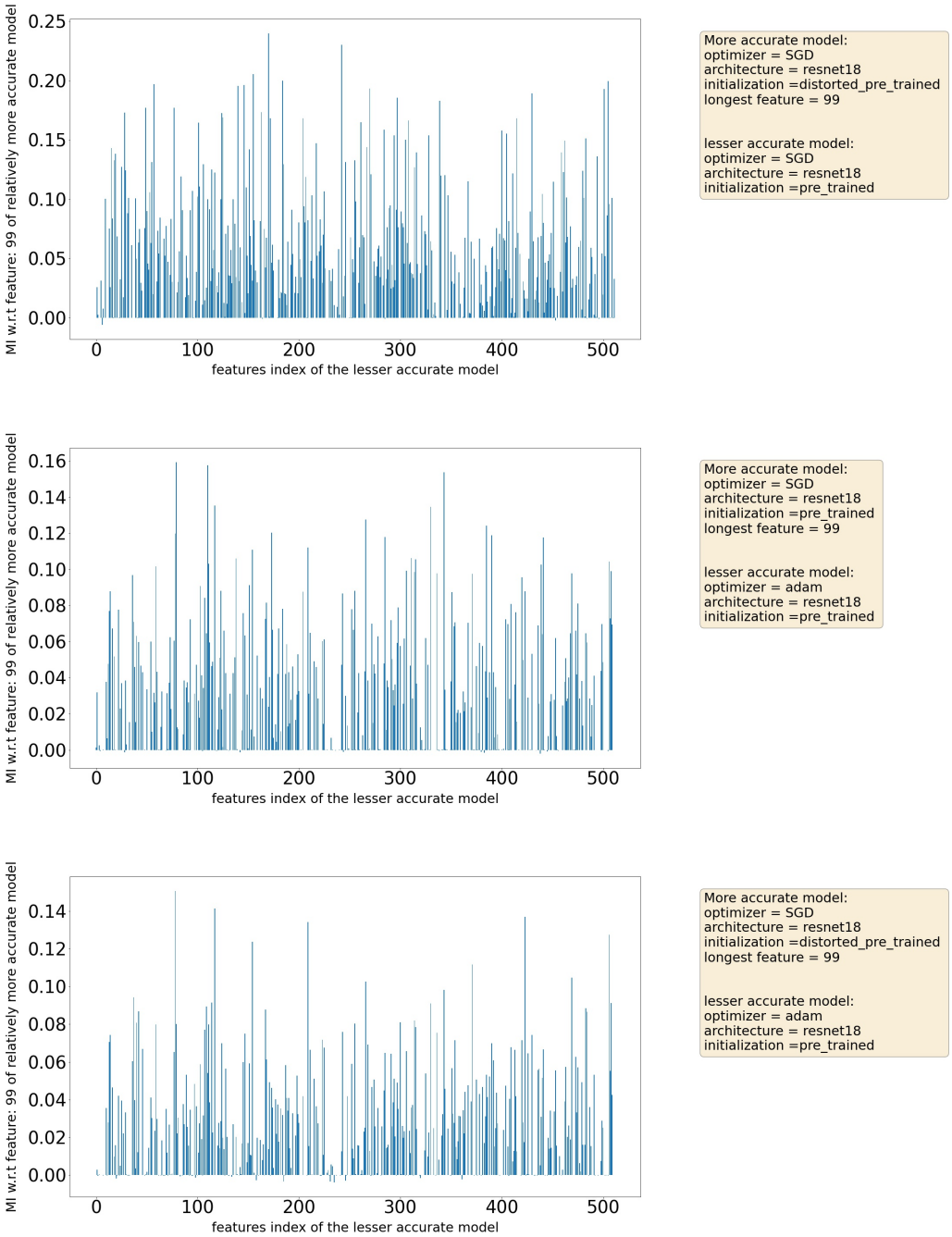

Figure 32: visualization of Mutual information experiment on a ResNet model on FMNIST data. To the right of each plot are the details of the models used.

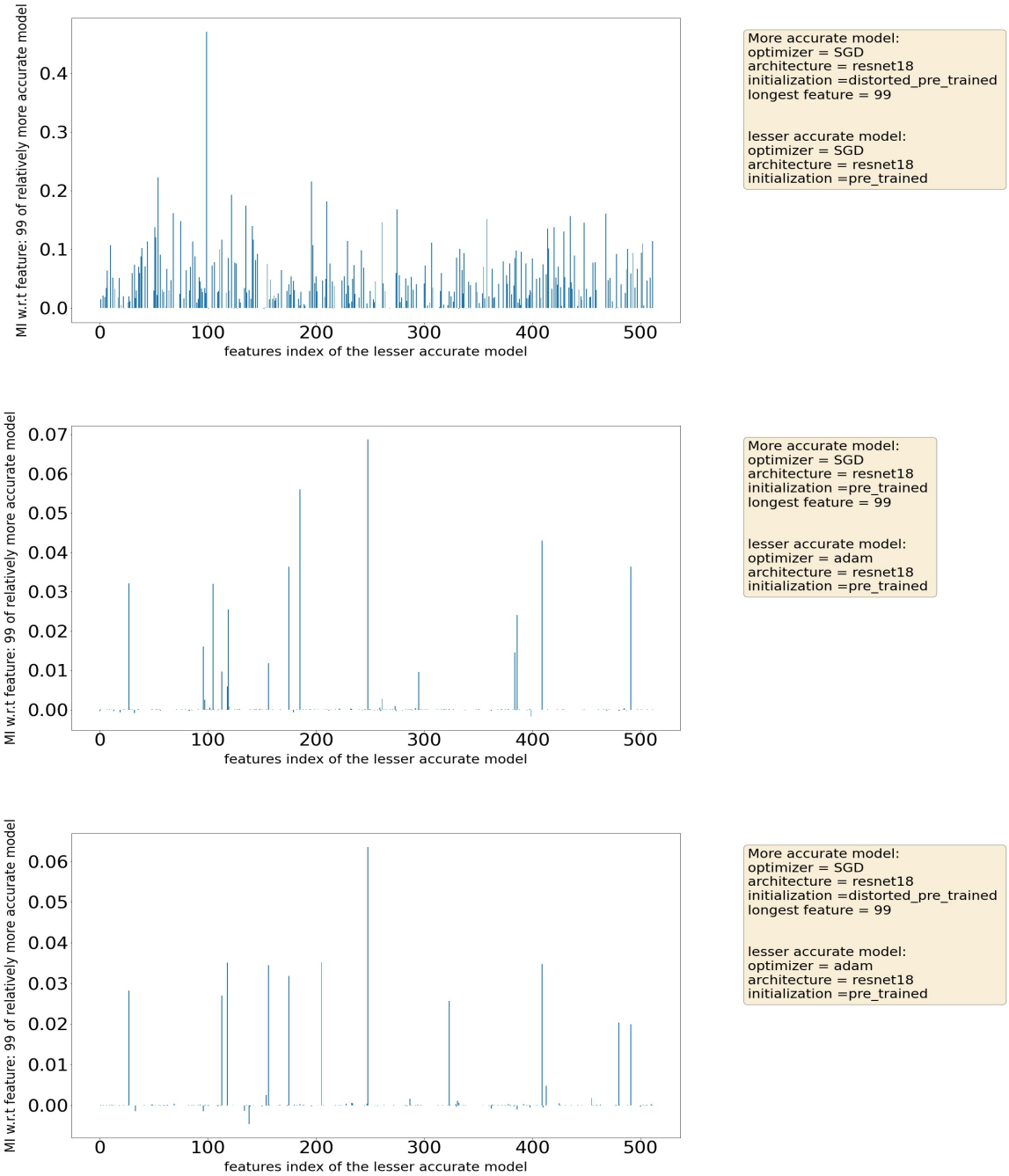

Figure 33: visualization of Mutual information experiment on a ResNet model on CIFAR10 data. To the right of each plot are the details of the models used.