Privacy-Aware Image Classification

Alexandros Manolakos
Electrical Engineering Department
Stanford University
amanolak@stanford.edu

Nima Soltani
Electrical Engineering Department
Stanford University
nsoltani@stanford.edu

Abstract

Training neural networks is a computationally-intensive task that is best suited for massively parallel machines like GPUs or server farms, and as such users realistically would have to give their data to the cloud for building the classifier. When giving this data away to be processed, it is at risk of being taken by non-intended parties. In this work, we propose modifying the data being sent, in this case images, such that if it were intercepted, it would be difficult to reconstruct the original image. We propose multiple methods for achieving this privacy strategy, and show the tradeoffs encountered in these scenarios.

1. Introduction

Privacy is an important concern in society today, especially when data is voluntarily sent to the cloud for storage or processing. There are plenty of news stories about data being compromised such as the hacked celebrity iCloud accounts, or the cyber attack on Anthem BlueCross, in which patient health records were accessed. In many scenarios, it is necessary to store personal information in the cloud to be able to provide their respective services.

For this project, we consider a scenario where a user submits a repository of his/her own images to the cloud for training a neural network-based classifer. For example, for the next big scientific breakthrough, neural networks could be used to classify objects in experiments (a task that would take many man-hours to complete manually), but a lab would not like to have it leaked to competitors; or it could be anyone who wants to build a neural network to detect their own face in images, and as a result needs to train on a vast dataset of their own images. As the training process is a long one, these people have their information trained on a neural network trainer in the cloud, and as a result submit their images to the cloud. It goes without saying, that in either of these example scenarios, if cloud machines are compromised, users would be legitimately concerned about hackers distributing their private data.

In this project, we look at different ways in which the user can take an active role in maintaining the privacy of his/her data, and compare their respective performances. We try to find ways of reducing the damages of this kind of privacy leak by not uploading the raw data to the cloud, but rather performing some pre-processing on it as a perturbation and uploading that so the dataset would then be stored on the cloud for training the neural networks. This way, if a leak were to occur, it would be a much more arduous task to recover the actual dataset. This will come at a cost of additional complexity, and a decrease in the accuracy of the resulting classifier. This is what we wish to explore in this project.

2. Background

While the problem we propose has not been looked at, a number of related problems have been. In [2], the premise is similar, in that we want to train a neural network but jointly across machines, and each one wants to keep its data private from the other. As a result they calculate they only communicate their changes to the model parameters to one another after each iteration. We are interested in the learning being solely done on the cloud so as to fully offload the work, and not require a network connection be established throughout this lengthy process.

Our question is more similiar to [3], in which the tradeoff between accuracy and privacy is explored. They, like us, implement local privacy, meaning that instead of giving the learner the real samples X_i, perturbed samples Z_i are given instead. They assume a very generic framework and establish bounds on mutual information between the X_i and Z_i. This work captures much of the spirit of what we wish to explore, but the loss functions they use are assumed to be convex in the parameters of the model, typical of SVMs and linear regression-type estimators. Unfortunately, that cannot be assumed in our model, as we are dealing with neural networks, which by the existence of blocks like ReLUs and MaxPools, are non-linear and non-convex.

Figure 1. Basic neural network architecture used. CR is CONV-RELU, CRP is CONV-RELU-POOL, FC is fully connected and SM is a softmax classifier.

3. Approach

We use the CIFAR-10 dataset, as the images are small enough for us to try a variety of privacy preserving techniques in the time we have this quarter. Due to time constraints, our goals are not necessarily to find the best possible classifier given data, as it may take a long time. Instead we plan to optimize over around ten epochs, and find the best fitting models for the different strategies. We feel that this would allow for a relatively fair comparison.

We use the Python code from Assignment 2 to build a baseline classifier. The architecture we use is shown in Figure 1. Using this dataset we are able to get a classification accuracy of 74%. We also trained a three layer network, which had a CR-CRP-FC-SM architecture, but it obtained a validation accuracy of 67%. As result, for the remainder of the paper we refer to the setup of as a baseline for comparison.

We develop trade-off curves between privacy and classification accuracy. Enumerating privacy will depend on the particular technique used to perturb the image. For example, one simple technique is blurring, in which case the privacy metric is the radius of the blur - the wider the radius, the more it blurs and the less recognizable the image becomes, which while is potentially good for privacy, is bad for classification. We also look at scrambling the input images, and in particular scrambling within non-overlapping sub-blocks within the image. As the size of these sub-blocks increase, the more random the image will appear.

For this project we implement a few different techniques, which all apply some sort of linear transformation to the images. The approaches we try are:

3.1. Gaussian Blur Filter

First we try to apply a blurring filter, which is an intuitive privacy preserving transformation that is often used for censoring images. In this strategy, we convolve the input images with Gaussian blur filter of different sizes. Increasing the size of the filter makes it more difficult to recognize. The blur is implemented by OpenCV [1], and it is a $K \times K$ filter, characterized by a Gaussian function centered at $\mu = \frac{1}{2}(K-1)$ with standard deviation

$$\sigma = 0.3\left(\frac{1}{2}(K-1)-1\right)+0.8.$$

Figure 2. For block scrambling, we first divide the image into blocks as pictured above, and then permute the pixels within the block.

3.2. Random Filter

Our next approach generalizes the Gaussian blur with a filter with random weights, sampled i.i.d. from a standard Gaussian distribution. Whereas in Gaussian blurring, there is a sense of down-sampling the image by low pass filtering, in this case we apply a random filter which could low pass filter, but could also high pass filter and bring out edges.

3.3. I.I.D. Random Filter

In the previous section we assumed that one filter is generated by sampling coefficient i.i.d. from a Guassian distribution. Another approach is that for every picture, we use a *different* filter, generated in a similar manner. This should pick up much more higher level qualities about the image, as the minute details will be lost through the different filters applied to the image.c

3.4. Block Scrambling

Another method that is commonly used to preserve privacy is to scramble images. By scrambling, we specifically mean spatial rearrangement of pixels within the image. In order to be able to build convolutional neural networks, however, there needs to be some spatial structure. In order to be able to do so we divide the images into blocks like in Figure 2, then within each block, we apply a random permutation matrix to move the pixels around within the block.

3.5. Block Scrambling with Descrambler

In order to build deep networks within a reasonable number of samples, we hypothesize that the classifier needs to somehow find out how to descramble the data before passing on to the convolutional layers. As such we modify the baseline architecture to that of Figure 3, where we added a block fully connected (BFC) layer. Like it sounds, this layer of neurons breaks the image down into blocks and takes as input all the elements within the block.

Figure 3. Architecture of the block scrambling technique with a descrambling filter.

This saves on parameters compared to a regular fully connected layer because each output pixel only depends on the pixels in the previous layer that lie in the same block, not the whole layer.

As the scrambling/descrambling operation within the sub-block can be thought of as a multiplication by a permutation matrix, which consists only of ones and zeros, we use l_1 regularization for when building the loss function.

4. Experiment

In this section we will go over the effects of the perturbations of the image on the accuracy of the classifier. We will apply the different perturbations to the car example image, so as to maintain consistency.

4.1. Baseline

We performed a hyper-parameter search and found that with a learning rate of 0.001 and a regularization strength of 0.001 we obtained our baseline classification accuracy of 74%. For the remainder of the experiments, unless stated otherwise, we used these same parameters. Also, it is important to keep in mind there are 10 labels and thus the accuracy of getting the classification by chance is 10%.

4.2. Gaussian Blur Filter

We show an example filtering of the car with a Gaussian blur filter in Figure 4. As we increase the size of the blur filter, we see that the image becomes more and more blurry. Finer resolution details may be lost, but the general features still remain visible. It is essentially a low pass filtering that one would apply before downsampling the image.

The results of the random filtered images classified using the architecture of Figure 1 can be seen in Figure 5. We see that while there is a definite drop in the performance of the Gaussian blur filter, it is less than 5% for all settings. This result is reasonable and expected, as we are essentially trying to classify the images by a smaller version of them, not necessarily distorted - a task that humans should be able to do easily.

4.3. Random Filter

We show an example random filtering of the car in Figure 6. It is interesting that the results of filtering with random weights still maintains some structure of the original image. The colors are distorted and as the size gets larger, it does become more difficult to tell what the original image was, if you didn't see the original image first, but it still maintains much of the details.

The results of the random filtered images classified using the architecture of Figure 1 can be seen in Figure 7. For each filter size, we try multiple runs (indicated with dots) with different random filters so as to get an idea of the vari-

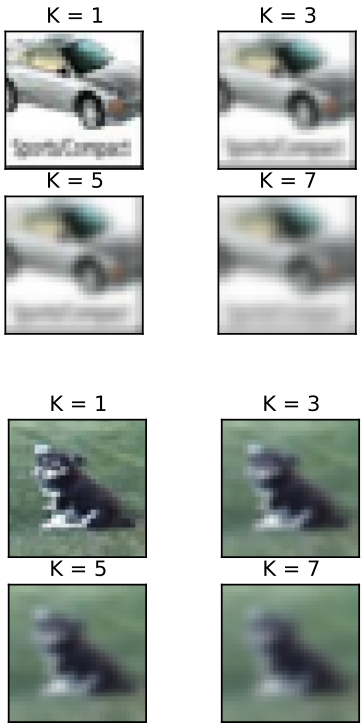

Figure 4. Example results of filtering the image with random 3×3, 5×5 and 7×7 filters.

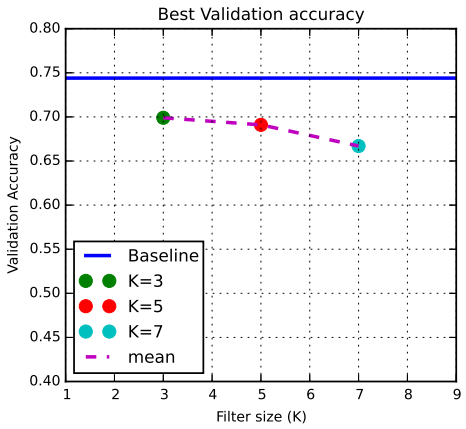

Figure 5. Validation accuracy of images with random filters of size 3, 5 and 7.

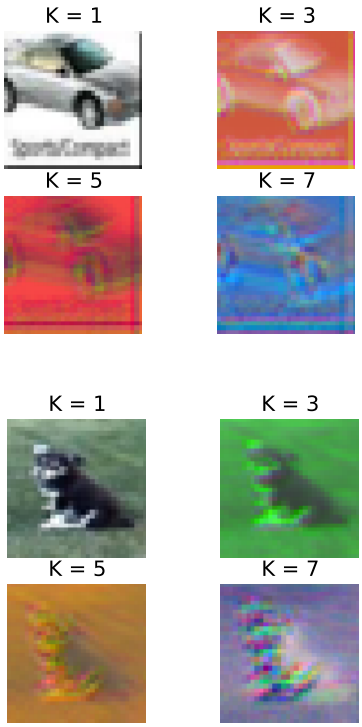

Figure 6. Example results of filtering the image with random 3×3, 5×5 and 7×7 filters.

ability of the method. We see that for some random filters the validation accuracy is very good, whereas for others it is worse, but on average the performance is roughly the same as the Gaussian blur filter. This may be a better option than the blur filter as it is not as easy a task to recognize the orignal image.

4.4. I.I.D. Random Filter

We show an example random filtering of the car in Figure 8. In this method, we randomly generate different filters to apply to each image, and in this figure we show the results of applying three different randomly generated 5×5 filters. The goal is that with the differently filtered images, the classifier will be able to learn more general features.

The results of the i.i.d. random filtered images classified using the architecture of Figure 1 can be seen in Figure 9. We see that the performance is significantly worse than the single random filter, that is a drop of almost 20% in accuracy. Having said that, we are still performing significantly better than the 10% accuracy of random guessing. It says a lot that we can capture information from this set of randomly filtered images, and the privacy of this is better than the previous case, as we cannot leverage the color information of one image to help us in identifying the others.

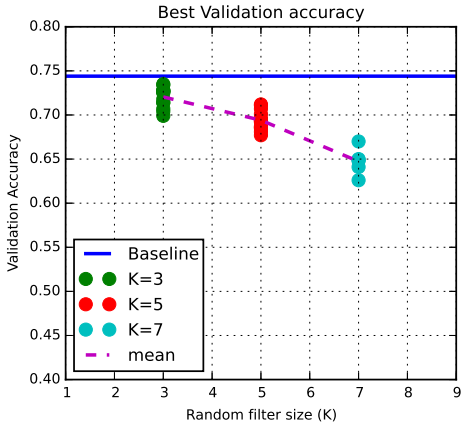

Figure 7. Validation accuracy of images with random filters of size 3, 5 and 7.

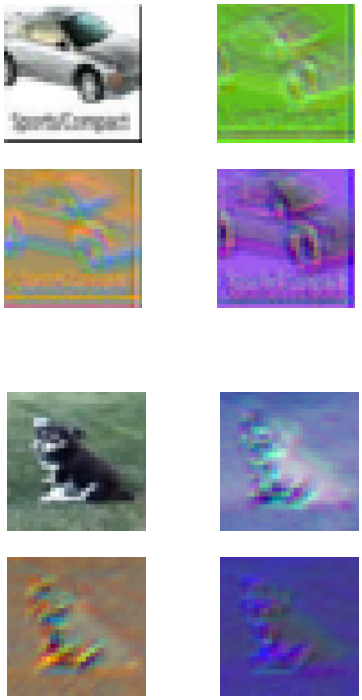

Figure 8. Example results of filtering the image with different random 5×5 filters.

4.5. Block Scrambling

We show an example block scrambling of the car in Figure 10. When the block size is 2×2, the image is still discernable as a car. When the block size is 4×4, the fact that it is a car becomes very unclear, and when it is 8×8, it

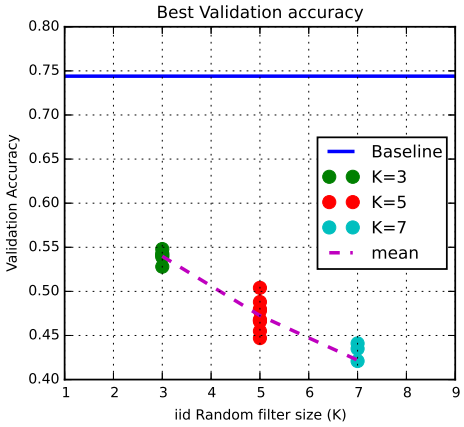

Figure 9. Validation accuracy of images with i.i.d. random filters of size 3, 5 and 7.

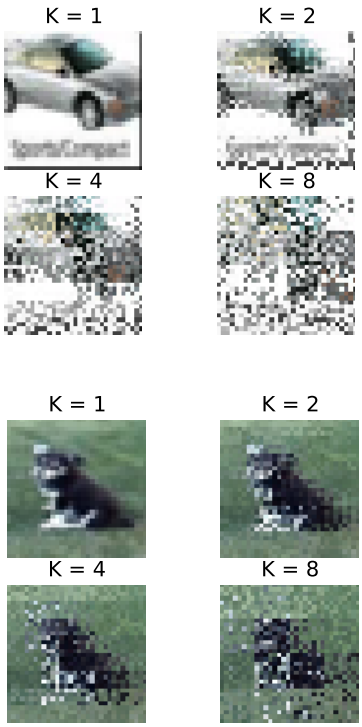

Figure 10. Example results of scrambling the image with blocks of size 2, 4 and 8.

is impossible to guess that this was originally a car.

The results of the block scrambling feature classified using the architecture of Figure 1 can be seen in Figure 11. It is surprising how well this technique works at classifying the images, despite the scrambling. The performance is better than the i.i.d. random filters, but slightly poorer

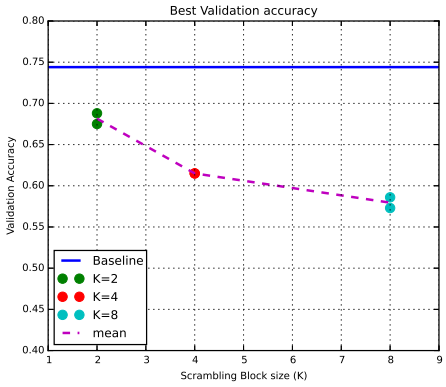

Figure 11. Example results of scrambling the image with blocks of size 2, 4 and 8.

than when a single random filter is applied. The accuracy is 58% for an 8x8 scrambling which is impossible to decode visually.

4.6. Block Scrambling with Inferred Descrambler

Finally we look again at the block scrambling technique, but instead try to descramble the image first. The perturbed data looks the same as Figure 10, but the difference is in the classifier, which has the architecture of Figure 3. The results can be seen in Figure 12. Comparing this to the previous set up without the descrambler, we see that the performance is always roughly 10% worse. What this indicates is that the descrambler has not learned the descrambling well, because otherwise the performance should increase. In this case we did have more parameters than the rest of the examples, so it could be because regularization and learning parameters were not set optimally, or that more epochs were required to train the classifier. We are still performing much higher than the 10% accuracy of random guessing, but we did not gain anything by including this descrambler block.

To analyze these results further, for each block in the image, we find the closest permutation matrix of the inferred descrambler in terms of l_2 distance. We refer to this permutation matrix as the l_2 *inferred descrambler* and then compare it with the true permutation matrix that should be used to descramble the block. Figure 13 plots the percentage of blocks in which the l_2 inferred descrambler successfully descrambles some of the pixels inside the block. We observe that even for the $K = 2$ case, the percentage of blocks which had an l_2 inferred descrambler exactly equal to the correct descrambler is less than 5%, and that approximately 70% of the l_2 inferred descramblers do not descramble correctly not even one pixel. The situation is even worse for the $K = 4$ case. These result verify that the current procedure is not able to descramble the scrambled images, which

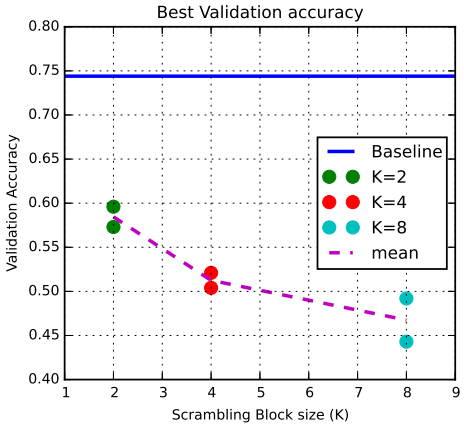

Figure 12. Example results of scrambling the image with blocks of size 2, 4 and 8, and descrambling with a block fully connected layer.

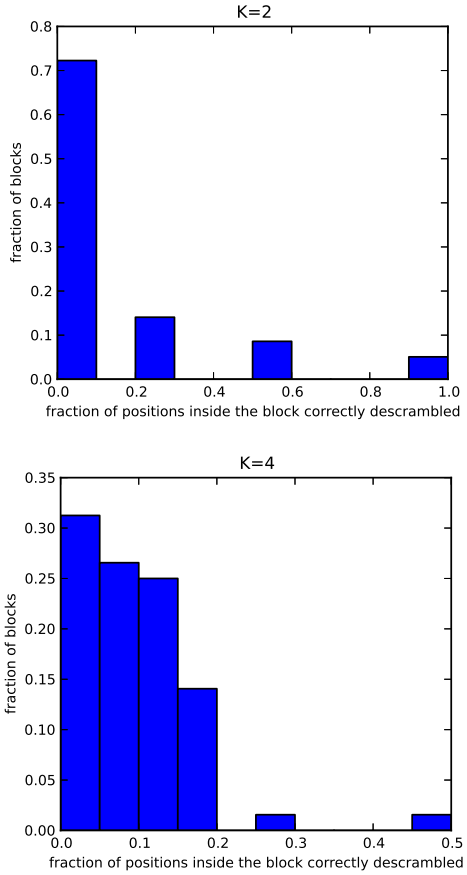

Figure 13. Histogram of the accuracy of the Inferred Descramblers.

corroborates the significantly worse accuracy performance.

5. Conclusion

In this project we covered different pertubations we could make to the original images to make them difficult to recognize visually, but still allow for classification. We saw that Gaussian blurring essentially low pass filtered the image losing the finer features but still maintaining the general structure of the image, and it classifies very well. With random filtering, it distorted colors, and sometimes blurred or brought out edges, making it more difficult to recognize than the Gaussian blurred image, but still performed the classification almost as well as the Gaussian blurred image. Next we looked at i.i.d. random filtering where we applied a different random filter to the images, and our performance suffered a lot, but we would not be able to visually use what we know from one image to help us decode others, as they are filtered differently. We also looked at a different, non-linear transformation, block scrambling, and we saw that the images became very difficult to identify visually, but the performance was quite good, only slightly poorer than Gaussian blurring. We tried to improve this technique by including a descrambling filter that the classifier would try to learn, but we were not able to achieve the results we were looking for and got much poorer performance, which is most likely due to an incorrect selection of tuning parameters.

Another technique to look into in the future is to look at the performance when we add noise to the transformed data. In this case, the more noise we add, the more difficult it should be to classify the images, but also the more difficult to tell what the original images were. Adding i.i.d. noise may not be the ideal case, as the eye may be able to do some local averaging to remove the noise, but some sort of spatially correlated noise may make it difficult to tell what the original image was.

It is also possible to generalize the sub-block scrambling, by making each pixel a random linear combination of the sub-block inputs. If we used a block fully connected layer, the layer would need to be optimized with l_2 regularization as the weights are no longer sparse, but random weights.

The issue of privacy in classification is important for the future of machine learning in general, and in this project we covered a few ways of achieving it. While these techniques might not be the ones used in the future, we feel that scrambling should be involved in some way in the future due to its great disruption in the spatial structure of the image, and how neural networks can somehow still learn underlying features.

References

[1] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*.

[2] T. Chen and S. Zhong. Privacy preserving back-propagation neural network learning. *IEEE Transactions on Neural Networks*, 20(10):1554–1564, 2009.

[3] J. Duchi, M. Jordan, and M. Wainwright. Privacy aware learning. *Journal of the ACM*, 61(6), 2014.