Sound-to-Image

Thom Miano
Department of Computer Science, University of Virginia, Charlottesville, VA 22904
[tnm4s]@virginia.edu

Abstract

Convolutional Neural Networks (CNNs) are widely used for machine learning tasks involving digital images. When digital images are sampled, light intensity is captured at some range along the electromagnetic spectrum, with multiple ranges (i.e., "channels") possibly being captured. CNNs are effective at finding patterns in images as they relate to some outcome of interest (e.g., the identification of an object) without explicit indication from humans through manual feature engineering. Digital audio is a representation of waveforms in discrete samples, where each sample value has some amplitude and the frequency of the sound is defined by the number of samples that fall within a cycle. CNNs have been used on audio far less frequently than they have been used on images. The goal of this project is to build a real-time environmental sound classifier that can be plugged into a visualization generator for real-time visualizations of predicted sounds. The visualization generator could take on many forms, and in this paper we investigate generating spectrograms as well as fixed class images that have Deep Dream applied.

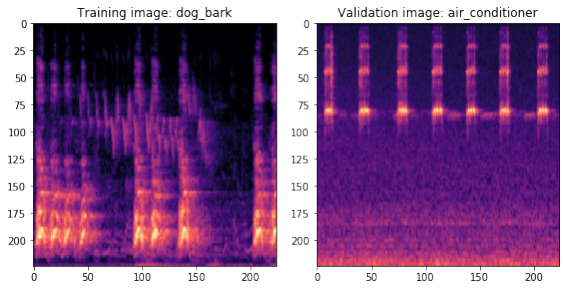

Figure 1. Spectrograms are scaled to 224x224 pixels before being passed into the sound classification model.

1. Introduction

While neural network (NN) architectures are widely used to classify images, they are less commonly used for sound. Advances in computer vision in the past few years have made image classification and segmentation much stronger. The corresponding tasks for sound, on the other hand, remain behind. A strength of CNNs are their capacity for self-identifying features relevant for distinguishing labeled categories of input data. Several researchers have investigated CNN transfer learning for sound classification, and overall they have had good results on a range of tasks [2] [1].

2. Related Work

Testing multiple basic NN architectures and using a very clean dataset of musical instruments, [4] demonstrates that networks trained using sound data represented in the frequency-space perform significantly better than those trained on the same data represented in the pressure-space. [4] also found that a CNN architecture (similar to LeNet) performed the best at 97% accuracy on their test dataset. [7] lays groundwork for use of CNNs in noisier sound data, like natural environments. [1] demonstrates that the strength of CNN-derived features for sound classification comes from the network structure itself and that even non-domain CNN features (e.g., from a vision task) can be useful if there is not a suitable source task for transfer learning. [1] also show that low level features (especially coming from the first and second layers of a CNN) can be important in predicting a class by extracting features at each layer, performing average pooling for each, and then concatenating the output into a single feature vector. Additionally, [3] uses AlexNet pretrained on ILSVRC-2012 to classify GTZAN, a music genre dataset, at a 78%. These findings are consistent with existing approaches that use manually derived features of low-level information like rhythmic patterns and tempo [6] [8].

3. Data

We use the UrbanSound8k dataset [9] to train our environmental sound classification network. UrbanSound8k contains 8732 labeled sound excerpts (less than 4s) of 10 classes. We do not perform any preprocessing on the data. The files as provided are pre-sorted into 10 folds, where

each fold subdirectory has a random sample of the different sound classes. For ease of use with PyTorch, we restructure the data so that each file belonging to a particular class occupies its respective class directory. We then end up with a dataset directory containing class subdirectories that each contain their respective sound samples.

Once our data were structured, we then converted all of the sound samples to melspectrograms, which took about 30 minutes. In our implementation, we use Librosas melspectrogram function, passing the sample rate for each of our files and otherwise using the default parameter values. We experimented with some of the parameters (e.g., n_fft, n_mels, and fmax), but we found this could lead to poorly rendered spectrograms when the value of the parameter actually turned out to be inappropriate. In comparison, we found that producing the spectrograms using the default parameters led to outputs without any obvious visual artifacts. Once all of our data were converted to spectrograms, we then randomly split the data into a training set, validation set, and final test set comprising 80%, 10%, and 10% of the data for each class, respectively.

Reviewing the data in Figure 4 of the Appendix, we notice a couple of concerning characteristics. First, the classes car horn, gun shot, and siren have fewer observations than the other classes, which have 1000 observations. In particular, car horn and gun shot stand out because they have fewer than 500 observations. Second, while most of the classes observations are at around 4 seconds in length, car horn and gun shot have a high percentage of observations that are less than 4 seconds. Most of the gun shot observations are around 1 second, but they are actually fairly distributed from 0 to 4 seconds. These concerns could pose problems in our ability to accurately classify real environmental sounds in that the time duration of a given sample could bias the model during training. For example, when we convert our sounds to spectrograms we will not take the length of our sample into account for determining the dimensions of the output spectrogram, and when our CNN sees a spectrogram it will see it a 224x224 pixel image (rescaled from 440x600 pixels). Because of this, longer sound samples will produce higher fidelity spectrograms. It is likely that the CNN will find these associations. In future research, we will investigate the effects of taking length into account. Another concern about the time duration of training observations is that it will likely affect the classifiers performance at different duration rates at test time. For example, if most of the training observations represent a 4-second window, then the model will likely perform better at test time using 4-second windows. It is our goal to eventually create a model that can perform well at high frame rates (e.g., 0.5 to 1-second time windows).

Before training any models, we listened to samples from each class. We noticed several audio-content characteristics that could pose problems in classification. First, most of these recordings come from "the field", which means that they often have background noise. If the noise is strong enough, it could drown out the signal thus preventing the classifier from being able to reliably identify it. Sec-

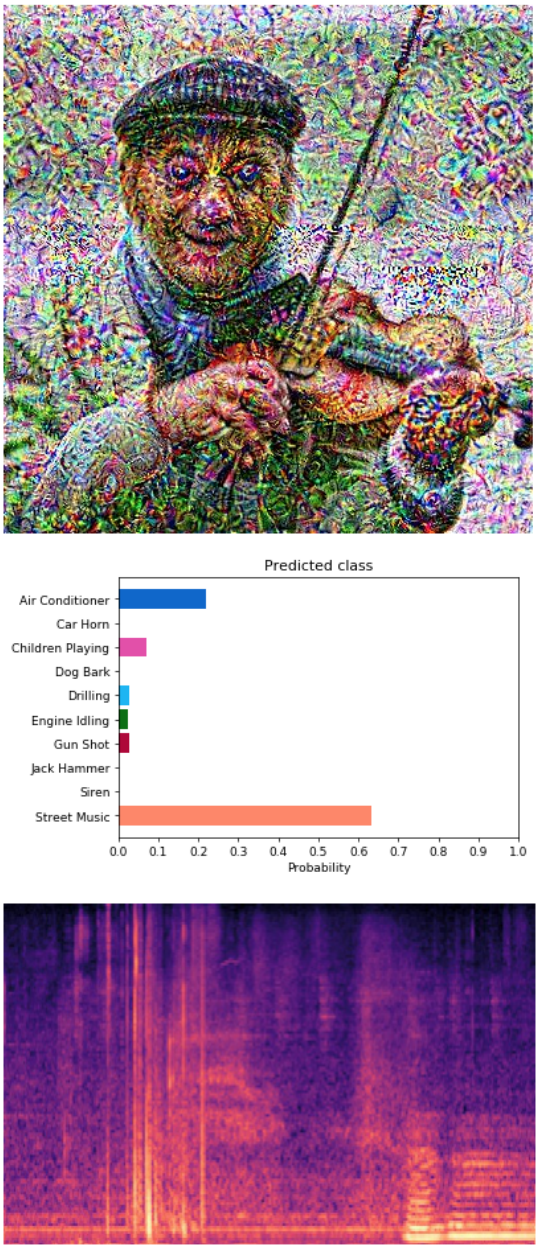

Figure 2. Sound classifier running in real-time on sounds recorded from a computer microphone. This 4-second recording was taken in a tea shop where there were several background sounds including music, a refrigerator hum, people talking, and clacks and bangs. Top: image corresponding to top predicted class run through deep dream; Middle: class probabilities; Bottom: Spectrogram of 4-second sample.

ond, we noticed several instances of ”class contamination”, wherein sounds that belong one class (e.g., ”dog bark”) can be heard in the sound space of another class (e.g., ”children playing”). Third, related to the previous two, there is an ”air conditioning” class that can sound like the background noise that is in other classes. As a result, this class could pose difficult to identify.

4. Model

To build our environmental sound classifier we fine-tune the PyTorch implementation of ResNet-50, using a single Pascal Titan-X GPU. The only change we make to the model is to adjust the final fully-connected layer so that it outputs a probability vector with 10 elements. We rescale our input images to make them 224x224 pixels, and we use a batch size of 16. We trained several models, but we found the best model used the SGD optimizer, a learning rate of 0.001, momentum of 0.9, weight decay of 0.0005 and 15 epochs. See Appendix Figure 6.

5. Experiments and Results

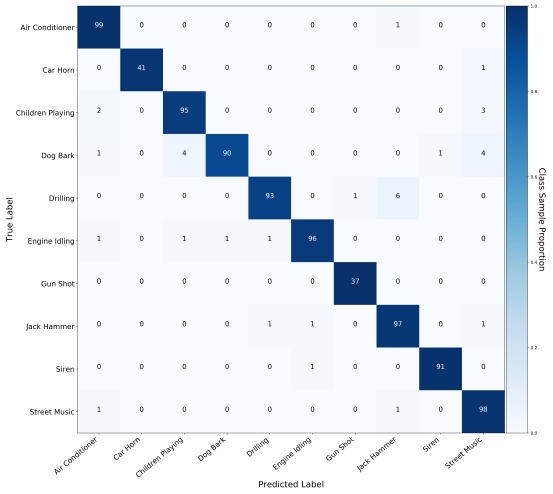

Figure 3. Normalized confusion matrix showing classification results on the testing dataset.

The key goal of this project was to classify environmental sounds in real time so that corresponding images could be generated in as close to real time as possible. As such, we wrote a simple Python program that captures input sound from a microphone, generates a spectrogram, and then runs it through the trained sound classifier. We found that at 4-second time windows we could get the classifier to guess the sound we were vocally trying to make. Figure 2 shows what this looks like live.

Beyond creating a reliable sound classifier, we also wanted to build a mechanism for creatively visualizing the predicted class. We explored several approaches for doing this. First, we rendered the spectrogram that was actually classified and its corresponding sound classification probability vector. Next, we pre-loaded 10 images that corresponded to the classes and displayed the appropriate class image instead of or with the spectrogram and probability vector. Finally, we explored applying Deep Dream to the class image being served.

We used Xavier Lins implementation [5] of Deep Dream in PyTorch, customizing some of the code for our purposes. The goal in using Deep Dream was to create self-recursive fractals of the predicted image, such that if the classifier predicted dog bark then an image of a dog being made up of dogs itself would be shown. Another goal was to use the weights of the sound prediction vector as well as other sonic properties of the input sample like amplitude as functions of the Deep Dream hyper-parameters, like dream iterations and learning rate. Our preliminary experiments with this demonstrated success.

However, in order to accomplish this in a high-fidelity way, we have several practical problems we need to solve. In experiments, we found that it takes about 4.5 seconds to capture a live 4-second windowed time segment, convert it to a spectrogram, and generate a class probability vector. At the fastest, it takes about .2 seconds to generate a Deep Dream image on a 200x200 pixel image and about .8 seconds to generate a Deep Dream image on a 400x400 pixel image. The render time varies greatly depending on the number of dream iterations, the number of octaves (sub-iterations), the number of deep dream layers the image passes through, and the number of pixels in the image resolution. In all of these cases, fewer means a faster render time. Table 1 shows the render times for several scenarios. Beyond simply taking longer, another problem with a longer Deep Dream job is that it holds up the sound classification pipeline, meaning there is a time gap between samples taken from the environment.

There are a couple of ways we could deal with these time constraints while still being able to adjust the time-affecting parameters. First, in our implementation we called Deep Dream code after we generated our sound prediction. Instead, we could insert the sound classifier inside of the Deep Dream code so that if the highest-predicted class at the current time step matches that of the previous time step, then the Deep Dream will continue where it left off for another iteration. If the highest-predicted class for the current time step is different than that of the previous time step, then the Deep Dream will start a new job with a new image, dropping the previous image. Second, in our implementation we have a single, discrete-sampling sound-to-image pipeline, which means that we will always have a minimum 4-second time gap between samples taken from the environment at each time step. There are two ways this could be addressed. One method would be to run parallel pipelines after each

other, at some time interval so that the image being displayed to the screen is rendered with a higher frame rate. Another method could be to continuously stream the input sound rather than record discrete 4-second chunks. That said, even with these methods there will always be at least a 4-second time gap between the beginning of a sample segment and the visualization of it. Under the sound classification architecture, in order to reduce the fundamental time gap between each sample we will have to reduce the sample window size for the generation of the spectrogram.

Seconds	Resolution	Iterations	Octaves	Layers
4.7	200x200	1	1	1
4.8	200x200	1	1	2
5.0	200x200	1	1	3
4.9	200x200	2	1	1
5.0	200x200	3	1	1
5.9	200x200	2	2	3
6.5	200x200	3	2	3
5.3	400x400	1	1	1
5.7	400x400	1	1	2
6.0	400x400	1	1	3
6.3	400x400	2	1	1
6.8	400x400	2	1	2
7.6	400x400	2	1	3
6.0	400x400	1	2	1
6.3	400x400	1	3	1

Table 1. Impact of Deep Dream parameters on run-time speed.

6. Future Work

While there are some methodological concerns that need to be addressed, this project demonstrates that environmental sounds can be reliably classified by converting them to spectrograms and using a performant CNN pre-trained on a non-domain task (e.g., image classification). This project could easily expand into other areas of research like voice-based person identification, car-engine trouble classification, appliance and power-consuming device identification (using energy signals), and animal language translation.

Continuing directly with this project, we have a long list of next steps. First, we will improve the order of the sound classification and the Deep Dream image generation so that previous sound predictions can be used to continue iterating a class image in Deep Dream or to start a new Deep Dream sequence. Second, we will incorporate the use of a Generative Adversarial Network (GAN) to generatively create class images instead of using a pre-determined images, which will then be passed into the Deep Dream assuming that the Deep Dream can be made more efficient. The incorporation of the GAN was part of the original vision of this project, but it was set aside to target the Deep Dream component first. Third, we will retrain our model, figuring out a way to take into account time segments so that classes are not biased by the duration of their observations. One way to address this could be to make every observation short (e.g., 1 second) by sub-dividing longer samples, which in turn would generate more training data. Another approach could be to create a mix of durations for each class, the idea being that the model would learn features for the classes at different time windows. Fourth, we will expand our dataset adding additional classes. Fifth, we will add interactive controllers control parameters in the pipeline at run time. For example, using flex sensors in a glove, one could tie the response of a glove flex sensor to the learning rate of the Deep Dream model such that a fist could create a more intense Deep Dream image whereas an open hand could create a Deep Dream image with little to no Deep Dream effect.

Finally, perhaps concurrently with the objectives above, we will wrap up the code so that it can be shared and run easily. Beyond the underlying scientific interest, the whole point of this project was to use deep learning to build an interactive and fun tool that could be enjoyed by anyone. Instead of a Fisher Price See n Say, children could play with something like this as a Say and See. Additionally, musicians and performing artists could use this to visualize their environments based on the sounds they produce. There are three methods we are considering for sharing this. First, we could build a Flask app and run everything in Python, keeping our sound classification and Deep Dream models in PyTorch. Second, we could move our models to Keras, in which case we could use Keras.js to easily share this. This would make implementing a drawn image-to-sound project much easier, but we may still need to use Python for some of the back-end processing. That said, during this project we were able to get a streaming spectrogram generating using p5.js.

References

[1] K. Choi. Transfer learning for music classification and regression tasks.

[2] W. et al. Scaling up to large vocabulary image.

[3] G. Gwardys. Deep image features in music information retrieval.

[4] A. Juliani. Recognizing sounds (a deep learning case study).

[5] X. Lin. https://github.com/xavierlinnow/deepdream_pytorch.

[6] U. Marchand. The extended ballroom dataset.

[7] K. Piczak. Environmental sound classification with convolutinal neural networks.

[8] B. Sturm. Revisiting priorities: Improving mir evaluation practices.

[9] UrbanSound8k. https://serv.cusp.nyu.edu/projects/ urbansounddataset/urbansound8k.html.

7. Appendix

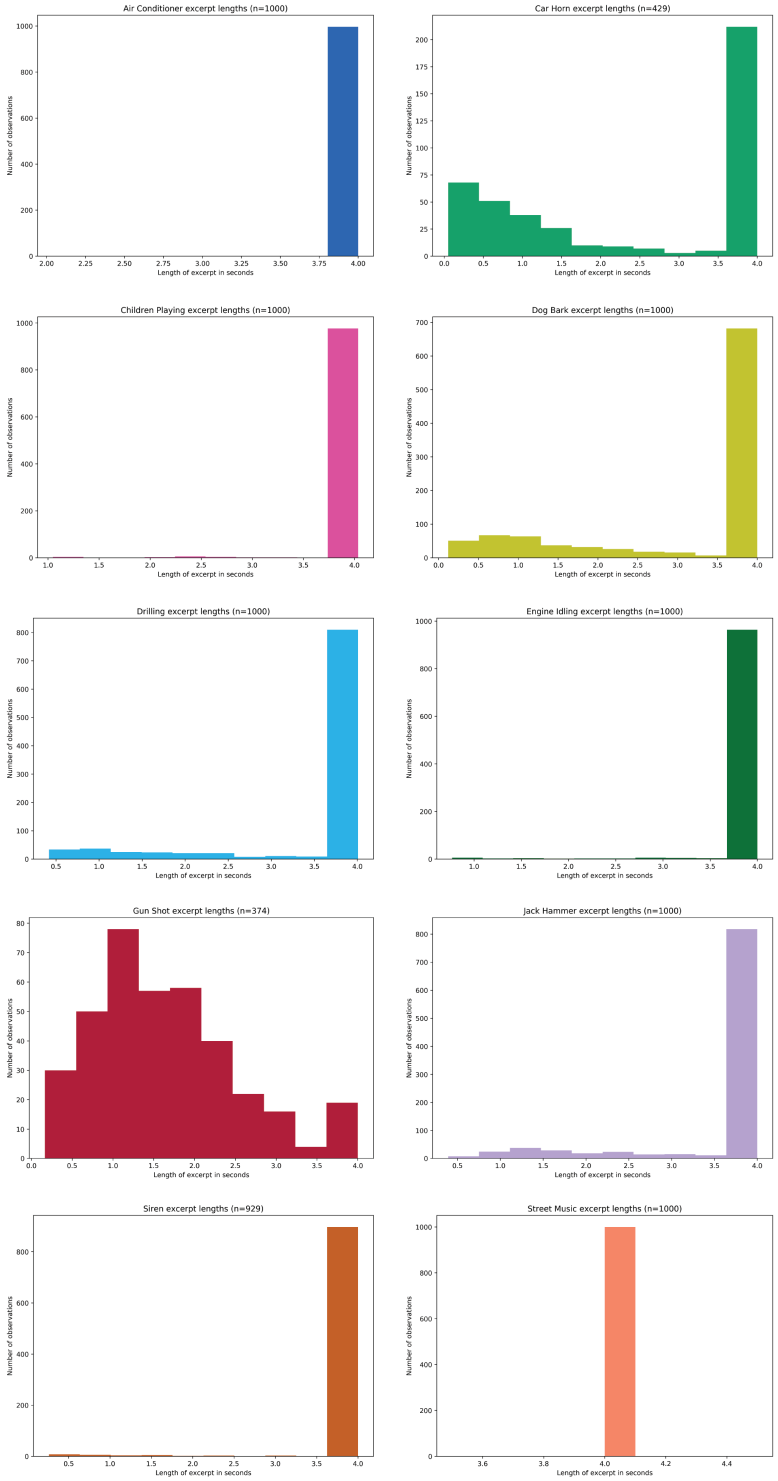

Figure 4. Histograms of UrbanSound8k classes. We see sounds range from 0 to 4 seconds, and we see that some classes have a high proportion of observations that are less than 4 seconds.

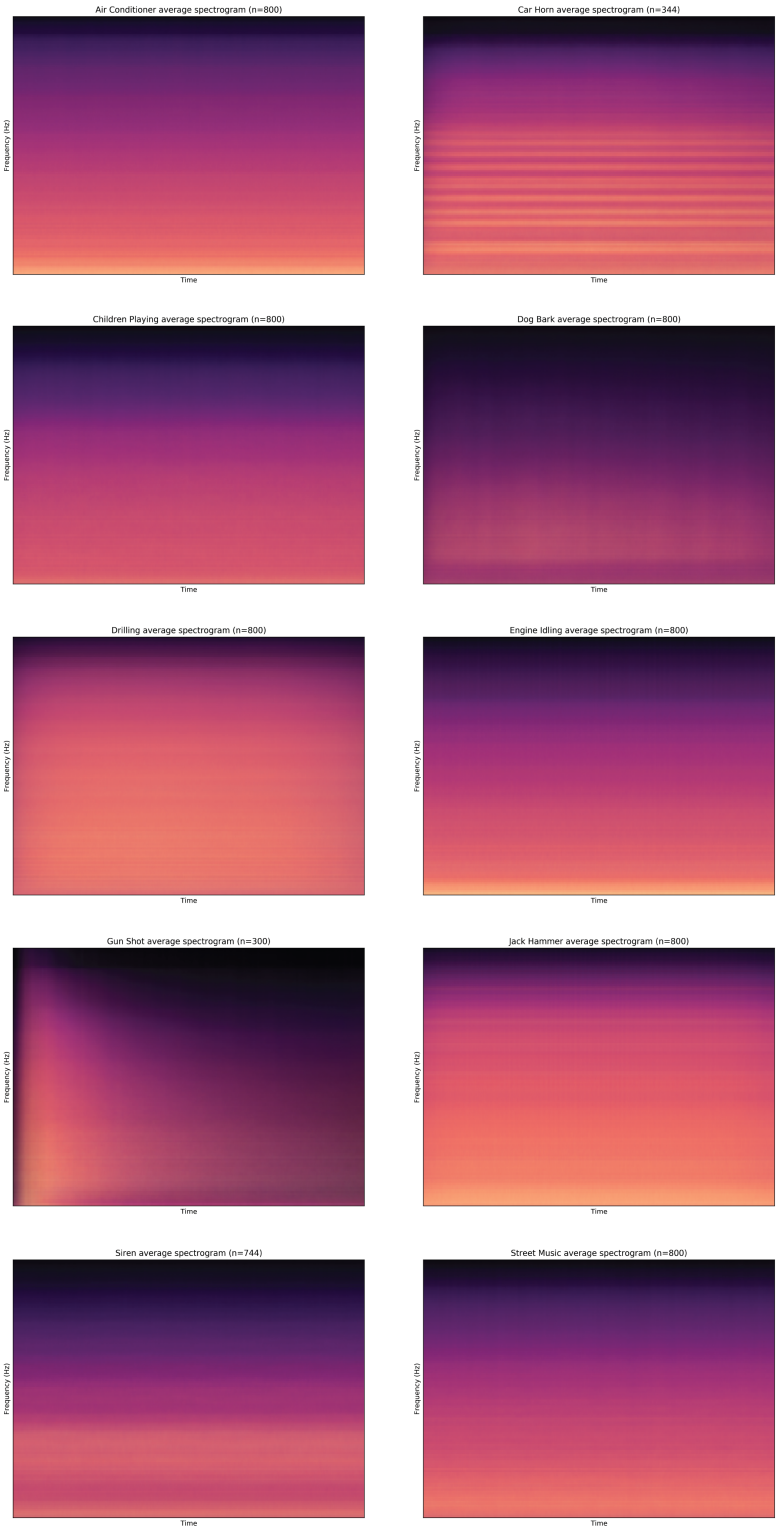

Figure 5. Class image averages for spectrograms in the training dataset.

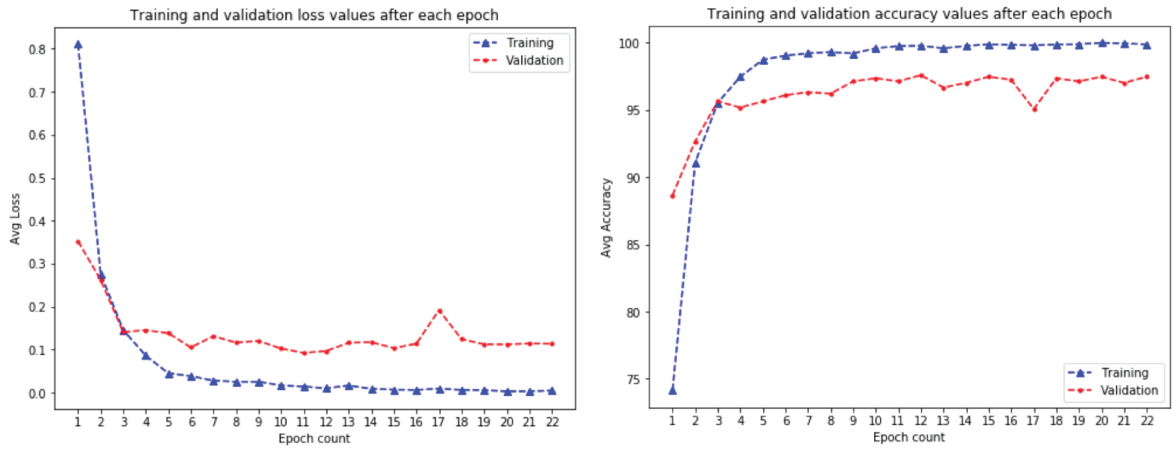

Figure 6. Training losses and accuracy at 22 epochs. We trained several models, but we found that under the parameters identified in the paper our model reached its best at around 10-15 epochs.