TINE-CNN Augmentation: Automatic data augmentation for any image classification task

Robbie Ostrow
Stanford University
ostrowr@stanford.edu

Abstract

This paper introduces a new data augmentation technique for image classification problems. The method, TINE-CNN augmentation[1]*, requires no human knowledge about the classification task and can effectively augment a dataset without supervision.*

We evaluate our method on CIFAR-10 and TinyImageNet. TINE-CNN augmentation is an effective regularization technique for both datasets, but our technique is especially helpful for TinyImageNet, which has more classes but fewer samples per class. We trained a VGG-9 model on each dataset augmented with TINE-CNN augmentation and achieved 84.2% and 42.1% top-1 accuracy for CIFAR-10 and TinyImageNet respectively. Training an identical model using hand-tuned augmentation achieved 87.1% and 37.9% on the same datasets. Training without augmentation or with naive "kitchen-sink" augmentation performed significantly worse. TINE-CNN augmentation can be combined with other augmentation techniques to improve the overall quality of the training data without requiring an expert to provide any specialized knowledge about the classes.

1. Introduction

Neural networks need a great deal of data on which to train. On many tasks, this problem is partially mitigated by data augmentation and transfer learning. However, it's not always clear what type of data augmentation is reasonable for a specialized dataset – for example, adding rotation into MNIST will result in confusing 6 and 9. In addition, deciding how to effectively augment a training dataset adds yet more hyperparameters to an already large model-design decision space. Augmentation is typically an ad-hoc process, and although there is recent work attempting to integrate it more naturally into the training process, there is no generic framework that researchers tend to use. Even so, data augmentation has long been in the toolbox of any neural-net designer [13] and frameworks such as Keras often provide easy augmentation APIs [1]. Unfortunately, these frameworks run into all of the above problems. Modern regularization techniques such as dropout can also help train CNNs on comparatively small datasets, but nothing replaces simply having more data with which to train.

[1]Pronounced Tiny-CNN augmentation

This paper introduces and evaluates a mechanism to automatically generate artificial data on a per-class basis. We test our approach by evaluating the performance of a simplified VGG architecture [14] on CIFAR-10 [9] and TinyImageNet [2]. We compare this to the performance of the same model trained on the same datasets with naive data augmentation, with hand-tuned augmentation, and with augmentation using our new approach. We find that our approach for augmentation significantly outperforms no augmentation and naive augmentation, and can outperform hand-tuned augmentation. However, this is no free lunch, as there are preprocessing costs associated with our approach.

2. Related Work

Dempster *et al*. published a seminal paper in 1977 that introduces an approach to the expectation-maximization (EM) algorithm that works with incomplete data [4]. Tanner and Wong expand on their work to formalize data augmentation as a technique to help calculate more accurate statistics about datasets that are known to be incomplete [15]. While neither of these papers had neural networks in mind, they are nonetheless extremely helpful to understand the theoretical justification for data augmentation in a generic classification task.

The EM-algorithm can be described in an elegant closed-form that allowed Dempster and Tanner to produce useful theoretical results. Unfortunately, convolutional neural networks tend to be a great deal messier, so formal frameworks for data augmentation for CNNs remain lacking. Models that perform well on the ImageNet challenge [10, 14] almost always use ad-hoc data augmentation that happens to work well on the particular problem, but may not be extendable to arbitrary classification tasks. These models tend to use the same set of augmentation techniques (random rotations, flips, crops, shifts, etc.) on every class in their dataset, simply testing many parameter choices to find out which are the most effective [6].

There are two major problems with the above approach. First, deciding which data augmentation techniques to use is a form of feature engineering, which makes the model harder to create and less likely to generalize. Second, and more importantly, these augmentations are applied to the entire dataset at once, rather than on a per-class basis. This makes sense for a dataset that has many similar classes that tend to be deformed in similar ways, but classes for a large image classification task are unlikely to all be helped by a constant set of data augmentation techniques. For example, an image of a bathtub will almost always be right-side up, so vertical flips will probably not be a useful transform for the "bathtub" class. On the other hand, an image of a can opener should be recognized as a can opener from any direction. It's clear that ImageNet's "can opener" and "bathtub" classes [5] shouldn't be augmented in the exactly the same way.

As the number of classes increases, it becomes more and more difficult for a human to choose good augmentation techniques on a per-class basis; and it's not clear that a human would know how to choose good parameters even if given the time. Hauberg *et al.* introduce a mechanism to automatically learn distributions for each class from which they can generate their own augmentation data [7]. This paper is closest in spirit to our approach. However, that paper attempts to learn exactly the transformations that each class label is invariant to, which may be an extremely complicated function. It's hard to know exactly the effectiveness of this approach as they only evaluated their augmentation technique on variants of MNIST, which is generally considered an "easy" dataset for modern classification techniques [12].

There are many other techniques such as using encoder-decoder networks to learn interpretable transformations [16] or using a generative network to automatically blend images within classes [11], but these approaches are very new and it's not yet clear how effective they are in practice[2].

3. Datasets

We evaluate our data augmentation technique on two separate datasets. First, we use CIFAR-10 [9], which has 60,000 images, each of size 32x32x3. These images are divided into 10 classes of 60,000 images each. For training and evaluation, we chose a subset of 49,000 training images, 1000 validation images, and 10,000 test images.

We also use TinyImageNet [2], which has similar types of images but many more classes and many fewer images per class. TinyImageNet has 110,000 images, each of size 64x64x3. These images are divided into 200 classes, with 450 training images, 50 validation images, and 50 test images per class.

We normalized both datasets by subtracting from each image the mean pixel value of the training dataset and dividing by the standard deviation.

We chose to evaluate on both of these datasets to determine how different sizes of training data (4,900 training images per class vs. 450) would impact the performance of our data augmentation technique.

4. Methods

To evaluate our technique without model hyperparameter choices confounding our analysis, we use a very simple model based on a truncated version of VGG-16 to train all networks in this paper (we call it VGG-9; see Table 1 for its specification). This architecture is powerful enough to perform moderately well on either TinyImageNet or CIFAR-10, but simple enough that changes in performance from run to run are likely due only to the quality of the training data and not to some nuance of the underlying architecture.

For both datasets, we train the VGG-9 network using the Adam optimization method [8] with learning rate 0.0005. Let C be the set of classes and let $p(d, c)$ be the probability that the classifier assigns image d to class c. Our network attempts to minimize the crossentropy loss:

$$\text{minimize}\left(-\frac{1}{|data|}\sum_{c\in C}\sum_{d\in data}\mathbb{1}(\text{d's true label is i})\log(p(d,i))\right)$$

We compare four different approaches to generating data to feed into this VGG-9 network:

1. **No augmentation** – as a baseline, we fit the VGG-9 architecture to each dataset without any data augmentation. Even with aggressive dropout, we see significant overfitting.

2. **Kitchen-sink augmentation** – uniformly randomly apply some subset of the following transforms to each image:

   - Flip horizontally
   - Flip vertically
   - Rotate somewhere between -90 and 90 degrees
   - Shift up to 33% horizontally
   - Shift up to 33% vertically

[2]A few days ago, a paper was posted on Arxiv that claims state-of-the-art performance on CIFAR-10 using a technique similar to mine [3]. Sadly, I only noticed that paper a few minutes ago, so I wasn't able to incorporate its techniques into this project!

Input (32x32x3 or 64x64x3)
3x3 Convolution (64 filters)
3x3 Convolution (64 filters)
2x2 Max Pool
3x3 Convolution (128 filters)
3x3 Convolution (128 filters)
2x2 Max Pool
3x3 Convolution (256 filters)
3x3 Convolution (256 filters)
2x2 Max Pool
Dropout (in training, drop 90%)
FC 1024
Dropout (drop 90%)
FC 1024
Dropout (drop 90%)
FC (10 or 200 classes)

Table 1. VGG-9 architecture. Convolutional and fully-connected layers use leaky RELU activations.

- Zoom from 70% to 130%
- Shift color channels up to 20%
- Shear image up to .2 degrees.

3. **Hand-tuned augmentation** – choose a good set of universal augmentations based on human knowledge about the dataset.

4. **TINE-CNN augmentation** – filter the data generated by kitchen-sink augmentation based on so-called "TINE-CNNs" as described in Section 4.1.

4.1. TINE-CNN augmentation

The fundamental goal of data augmentation is to draw new samples from a partially-known distribution. More formally, we want to find the set T of transformations over all possible input images I for which the class label is maintained:

$$T = \{t : I \to I | \forall .i \in I : label(i) = label(t(i))\}$$

.

This is an extremely hard problem, in part because the vast majority of possible images don't have a reasonable label, and partially because finding a universal transformation function t is likely to be very difficult. As discussed in Section 1, a simple transformation that maintains the class label "can opener" may not also maintain the class label "bathtub."

Instead, we attempt to approximate the solution to a simpler problem: given a class c and an image i, what is the probability that i belongs to class c? This is a bit of a chicken-or-egg problem because we could use any oracle that could answer this question in place of our image classifier. However, in order to find good augmentation data, we only need to determine whether it would be reasonable to guess that i belongs to class c.

We can construct a naive representation of a class c by training a very simple binary classifier on elements of that class versus random samples from other classes. If we have a black-box that generates augmentation data, we can determine which samples from that generator are "reasonable" samples by checking whether the binary classifier thinks sample s belongs to class c with probability above some threshold p_t.

Hopefully, the latent representation of the class c inside the binary classifier will learn some fundamental properties of the samples it's seen. If every single element of the "tree frog" class is green, it will assign very low probability to a non-green image being a tree frog. On the other hand, another classifier learning about parrots might be much more flexible about the color of the image.

This strategy depends on the training data being a representative sample of all possible class images. This is, of course, a simplifying assumption, but it's one that's impossible to avoid; a learning algorithm shouldn't be able to make inferences about training data that doesn't exist.

Notice that we already have a way to generate potential class members – kitchen-sink augmentation! We can also build a simple set of binary classifiers using very shallow CNNs. Finally, we're ready to introduce the Training Images Naively Embedded in a CNN, or "TINE-CNN" augmentation algorithm.

A high-level overview of Algorithm 1: The TINE-CNN augmentation algorithm trains three TINE-CNNs (see Table 2) for each class[3]. The input to train a TINE-CNN for class c is every element of class c, along with an equal number of randomly sampled elements from the rest of the dataset, with binary labels 1 and 0 respectively. We train several classifiers so a poor sample doesn't skew the results. Train for 10 epochs, holding out 10% for validation and using the best validation model to prevent overfitting.

To generate augmentation data, we iterate through the training dataset and perform a random transformation on each sample. If the median TINE-CNN classifies the transformed image as the original class with probability greater than some threshold p_t, then we accept this transformation and pass it on to the global model. Otherwise, we simply return the original image without any transforms. We can generate arbitrarily many samples in this way, and generate them in parallel while we train the global model. Table 3 shows three transforms that TINE-CNN accepts, and three

[3]For TinyImageNet, this ends up being 600 classifiers, which was too much for my GPUs, so we only trained one per class for TinyImageNet.

Input (32x32x3 or 64x64x3)
3x3 Convolution (16 filters)
3x3 Convolution (16 filters)
2x2 Max Pool Dropout (in training, drop 80%)
FC 256
Dropout (drop 80%)
FC 2 (binary classification)

Table 2. TINE-CNN architecture. Convolutional and fully-connected layers use RELU activations.

that it doesn't. TINE-CNN tends to accept about 75% of kitchen-sink transforms with $p_t = 0.1$, though the exact value varies from class to class, as some classes are more robust to transforms than others.

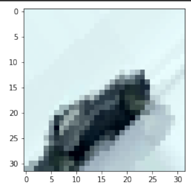
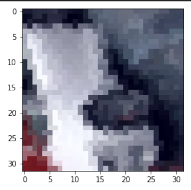
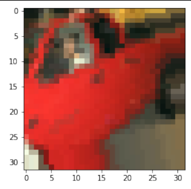
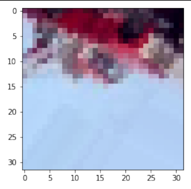
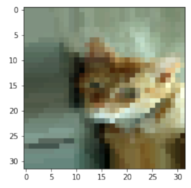
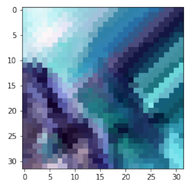
	Allowed	**Rejected**
Automobile?		
Truck?		
Cat?		

Table 3. Randomly chosen transforms and whether they pass the TINE-CNN filter.

5. Experiments & Results

5.1. Setup

We used the same setup for all experiments to fix every variable except the data augmentation technique. We trained a VGG-9 classifier using an Adam optimizer for 50 epochs with learning rate 0.0005. For some of the experiments, it was clear that the model overfits in far fewer than 50 epochs, or was still training with this learning rate after 50 epochs, but we determined that it was more important to have a consistent set of evaluation metrics than a perfectly

Algorithm 1 TINE-CNN Augmentation.

```
procedure AUGMENT(data, classes, threshold)
    $classifiers \leftarrow \{\}$
    for $cl \in classes$ do
        $members \leftarrow \{d \in data | label(d) = cl\}$
        $nonmembers \leftarrow \{d \in data | label(d) != cl\}$
        $[sample] \leftarrow$ take $|members|$ random items from $nonmembers$, 3 times
        $classifiers[cl] \leftarrow$ train 3 TINE-CNNs on members, [sample]
    end for
    while True do                                ▷ generate data indefinitely
        for $d \in data$ do
            $td \leftarrow randomTransform(d)$       ▷ using kitchen-sink transformations
            $pc \leftarrow median(classifiers[label(d)].predict(td))$
            if pc > threshold then
                Yield td
            else
                Yield d
            end if
        end for
    end while
end procedure
```

optimal model. We used a batch size of 256 and generated all augmentation data on demand.

We primarily examine top-1 and top-2 accuracy for CIFAR-10, and top-1 and top-5 accuracy for TinyImageNet, though examining the loss curves and confusion matrices are also instructive to help understand how the models are training.

We trained 3 TINE-CNNs per class for CIFAR-10. Unfortunately, due to GPU constraints[4], we could only train 1 TINE-CNN per class for TinyImageNet. We trained each TINE-CNN for 10 epochs and chose the epoch with the best validation accuracy. Since each TINE-CNN is so small and only trains on a few thousand datapoints, training all 200 for TinyImageNet took only about an hour; training VGG-9 on TinyImageNet for 50 epochs takes approximately three times as long, so preprocessing is expensive but not prohibitively so. Once all of the TINE-CNNs were trained, we generated the augmentation data in parallel to the training – which required two GPUs, because we had to constantly run TINE-CNN predictions alongside the VGG-9 training – but with the extra GPU, this wasn't a training bottleneck.

For the "hand-tuned" augmentation, we found that good choices for augmentation for both of these datasets were

- Rotate between -30 and 30 degrees

[4]All training was done on Google Cloud on a machine with 2 NVIDIA Tesla K80 GPUs, 16 vCPUs, and 104 GB memory.

- Shift horizontally up to 20%
- Shift vertically up to 20%
- Zoom from 90% to 110%
- Flip the image horizontally
- Shear counterclockwise up to .2 degrees

These are certainly not the optimal parameter choices, but they work well for both CIFAR-10 and TinyImageNet.

5.2. Results

Table 4 and Table 5 show basic statistics for VGG-9 trained using each of the data augmentation techniques described above.

We found that TINE-CNN augmentation consistently outperforms no augmentation and naive "kitchen-sink" augmentation. After 50 epochs, the model trained with hand-tuned augmentation had slightly better performance on the CIFAR-10 test set, though the model training with TINE-CNN augmented data was still underfit and had not yet finished training. Figure 1 shows the top-1 validation/training accuracies for each approach on CIFAR-10.

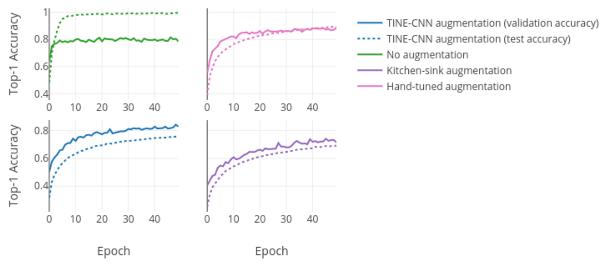


Figure 1. Training and validation accuracies for CIFAR-10. It's easy to see that without any data augmentation, we have significant overfit, while we're we're still training after 50 epochs with TINE-CNN augmentation. Validation accuracy in solid lines, training accuracy in dashed lines.

For TinyImageNet, TINE-CNN augmentation was the most effective approach by far. However, both hand-tuned augmentation and TINE-CNN augmentation both somewhat overfit TinyImageNet. Figure 2 shows the top-5 training and validation accuracies for each augmentation technique. The top-1 graphs have similar shapes. The TINE-CNN model can be tuned to prevent overfit by decreasing the threshold p_t, but we report results for $p_t = 0.1$ for all experiments in this paper. While TINE-CNN augmentation with $p_t = 0.1$ was not enough to prevent a slight overfit of TinyImageNet, it was still training on CIFAR-10 after 50 epochs, while the model trained with hand-tuned augmentation overfit both.

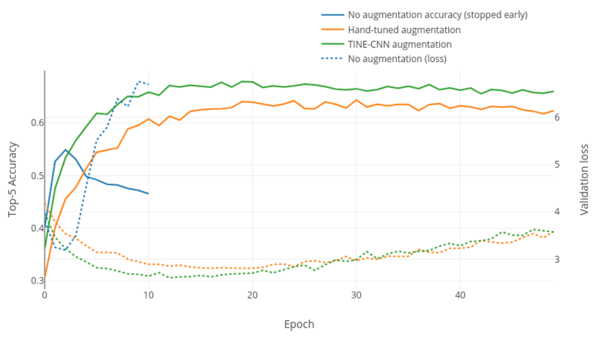


Figure 2. Top-5 validation accuracies for TinyImageNet, and the corresponding validation losses per epoch. Kitchen-sink augmentation is not included here since it performed much worse.

Figure 3 shows the confusion matrix for TinyImageNet trained using TINE-CNN augmentation. It's a bit too large to see, but the largest non-diagonal entry is between labels n02814533 and n03100240, or "pickup truck" and "station wagon." This is to be expected because these two classes are fundamentally similar, and it's likely that a TINE-CNN trained on a "pickup truck" class will accept a transform that makes it look more similar to a station wagon.

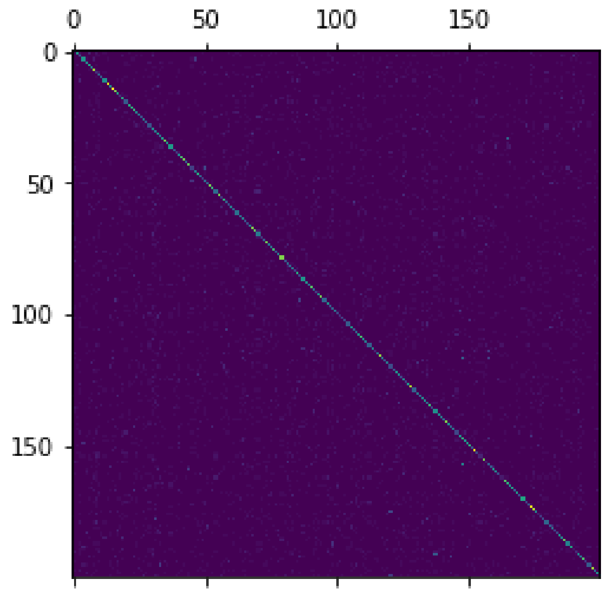


Figure 3. Confusion matrix on TinyImageNet's test set for a VGG-9 model trained with the TinyImageNet training set with TINE-CNN augmentation

It's a bit easier to understand the confusion matrices for a 10-class problem; Figure 4 shows the confusion matrices for our model trained with hand-tuned augmentation and trained with TINE-CNN augmentation. The third image

Augmentation type	**None**	**Kitchen-sink**	**Hand-tuned**	**TINE-CNN**
Top-1 test accuracy	0.785	0.741	**0.871**	0.842
Top-2 test accuracy	0.898	0.886	**0.957**	0.951
Minimum training loss	0.016	0.874	0.304	0.680
Minimum validation loss	0.664	0.753	0.412	0.639
Maximum training accuracy	0.995	0.690	0.892	0.757

Table 4. CIFAR-10 statistics

Augmentation type	**None**	**Kitchen-sink**	**Hand-tuned**	**TINE-CNN**
Top-1 test accuracy	0.285	0.410	0.379	**0.421**
Top-5 test accuracy	0.550	0.100	0.644	**0.679**
Minimum training loss	0.167	5.044	1.436	1.003
Minimum validation loss	3.191	4.917	2.807	2.604
Maximum training accuracy	0.950	0.027	0.608	0.716

Table 5. TinyImageNet statistics

shows the differences between the two matrices – a value below zero means that the TINE-CNN-augmented model had more entries in that slot. Interestingly, the model trained with TINE-CNN augmentation does a much better job of differentiating between the dog and cat classes, which was the biggest problem for the hand-tuned model. It's hard to say why this is, but some visual examination of the training set suggests that photos of cats are often rotated more than 30% from vertical – which TINE-CNN can account for but the hand-tuning cannot.

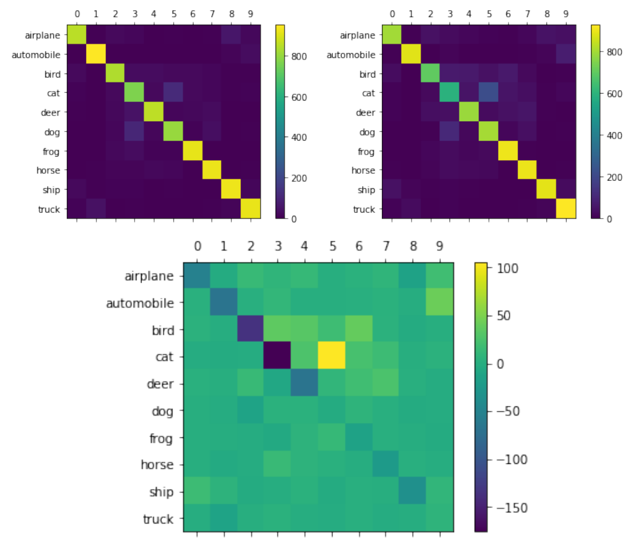


Figure 4. Left: confusion matrix for the CIFAR-10's test set trained with hand-tuned data augmentation; Right: confusion matrix when trained with TINE-CNN augmentation; Center: Difference between the confusion matrices (Left - Right).

In general, kitchen-sink augmentation consistently underfit the datasets because it was constantly generating training data that didn't quite fit the class labels. Training without augmentation overfit each dataset within just a few epochs. Hand-tuned augmentation and TINE-CNN augmentation performed comparably, though TINE-CNN augmentation worked marginally better on TinyImageNet. We believe that TINE-CNN augmentation's good performance on TinyImageNet is due in part to the smaller class sizes, but did not thoroughly test this hypothesis.

TINE-CNN augmentation performs around the same level or better than hand-tuned augmentation for both CIFAR-10 and TinyImageNet. It has the disadvantage of requiring a bit of extra preprocessing time, but the significant advantage of not requiring any ad-hoc augmentation that requires human knowledge. On an extremely specialized image classification technique, I expect that per-class hand tuning will still work marginally better than TINE-CNN augmentation. However, for any problem with many classes or when an expert is unavailable, TINE-CNN augmentation works – and can be dropped into any image classification task without configuration.

6. Conclusion & Future Work

Our automatic data augmentation algorithm performed better than raw data or naive data augmentation on both the CIFAR-10 and TinyImageNet datasets. While the hand-tuned augmentation performed slightly better on CIFAR-10, our model trained with TINE-CNN augmentation was the most effective for classifying samples from TinyImageNet.

These results are very encouraging because they provide convincing evidence that automatic data augmentation can be as effective as current ad-hoc techniques. However, due to computation constraints, we performed all of our analyses on very a very simple model architecture that was unable to get near-state-of-the-art results. With more computational resources, we would like to repeat this analysis on a more sophisticated model architecture. Additionally, the

TINE-CNNs themselves do not, in principle, have to actually be tiny. They could be sophisticated binary classifiers in their own right. This wasn't feasible with our experimental setup, but I'm curious to find out whether training a better set of TINE-CNNs will improve augmentation performance.

TINE-CNN augmentation is fundamentally a method to filter augmentation images generated in some other way. In this paper, we use "kitchen-sink augmentation," which is simply a combination of simple image transforms. We could apply the same idea to filter the augmentation strategies proposed by any of [7, 3, 16] to see if a combination of these strategies is more effective than a single one.

Finally, we analyzed TINE-CNN augmentation on two traditional datasets, but would like to find out how it performs in a more specialized space where expert knowledge is much more important, such as a medical imaging classification task.

7. Contributions and Acknowledgements

All work for this project was my own, but I leaned heavily on Andrej Karpathy's course notes and on the wonderful slides and lectures from this year's iteration of CS231N.

References

[1] F. Chollet. Building powerful image classification models using very little data. *Retrieved December*, 13:2016, 2016.

[2] CS231N. Tiny imagenet visual recognition challenge.

[3] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data, 2018.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[7] S. Hauberg, O. Freifeld, A. B. L. Larsen, J. Fisher, and L. Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In *Artificial Intelligence and Statistics*, pages 342–350, 2016.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] J. Lemley, S. Bazrafkan, and P. Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869, 2017.

[12] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, pages 301–320, 2007.

[13] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[15] M. A. Tanner and W. H. Wong. The calculation of posterior distributions by data augmentation. *Journal of the American statistical Association*, 82(398):528–540, 1987.

[16] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Interpretable transformations with encoder-decoder networks. In *The IEEE International Conference on Computer Vision (ICCV)*, volume 4, 2017.