Training a Robust Image Classifier with Domain Randomization and Data Augmentation Techniques for 2D Vision

Harry Zhang, Jenny Song, Zhitong Guan, and Shichao Han[1]

***Abstract*— Image classification is a crucial task in the field of computer vision. Starting in 2012, well-known ConvNets architectures have pushed the boundary of what a neural classifier could achieve to a great extent. Deep learning-based methods are becoming overwhelmingly popular. However, we as human still outperform machines in a variety of vision tasks. For example, human are extremely good at classifying a perturbed image, but it is difficult for machines to recognize useful patterns in the perturbed images. Moreover, datasets such as ImageNet do not contain real-world perturbations such as random occlusions and noise, so solely training on such off-the-shelf datasets does not improve the performance of neural networks when classifying noisy images. Therefore, in this paper, we research techniques for data augmentation and domain randomization that can increase the robustness of the trained classifier. We evaluate the performance of the techniques and our model on the provided Tiny ImageNet dataset, which contains 200 classes. We show that our trained classifier based on ResNet-50 achieves 70% training accuracy and 57.3 % validation accuracy, which is on par with the current state-of-the-art classifiers. We have made our project open-source and our code is available here.**

I. Introduction

Deep Convolutional Neural Networks (CNNs) have led to a series of breakthroughs for image classification [1], [2], [3], [4]. Furthermore, He et al. [5] leveraged residual connections in deep neural networks to tackle the vanishing gradients problem, which in turn tremendously improved the performance of deep networks. The emergence of ResNet encouraged researchers to use deeper networks more frequently than ever before without the fear of vanishing/exploding gradients. Apparently, deeper networks are a crucial part in the task of image classification and pattern recognition. However, the public datasets that are used to train the deep networks are not sufficient for the networks to be robust to real world perturbations. This is because datasets such as [6] do not contain real-world perturbations. As a result, networks trained on such data may not be generalizable to real-world data. In order to boost the robustness of the trained networks, we need to expose the networks to perturbed data so that the networks are "prepared" for real-world perturbations.

In this paper, we present a series of data augmentation and domain randomization techniques that would generate perturbations to the images in the Tiny ImageNet dataset that mimic real-world perturbations, which could enable the networks to robustly learn from the perturbations. The data augmentation operation is fully randomized and automated and can be run on GPU-accelerated devices. We improve a ResNet-based classifier and train it on the augmented dataset. Meanwhile, we attempt to make our results more approachable, explainable, and intuitive by visualizing the filters of the first convolutional layer, the class activation maps [7], and t-SNE feature projections [8].

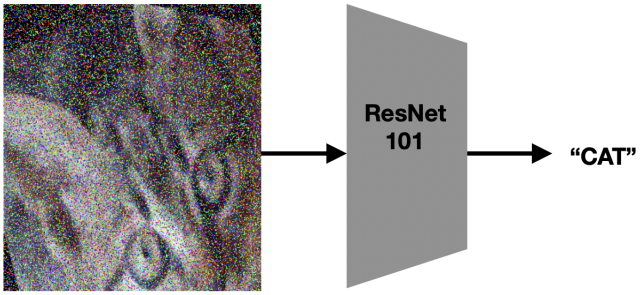


Fig. 1. Training a classifier that is robust to noises and perturbations

This paper contributes:

1) A comparison between ResNet 50 and ResNet 101
2) A robust neural image classifier based on ResNet 50 architecture.
3) An automated data augmentation process that generates perturbations on the original data to mimic real-world images.
4) A series of approaches that facilitate the interpretability of the classification results.

II. Related Work

a) Deep Neural Networks: LeCun et al. [2] is the first work that applied backpropagation in training a neural network. In [1], Krizhevsky et al. first used GPUs to accelerate the training process and applied ReLU, a nonlinear activation function, to boost the gradient flow in the deep networks. In [4], the authors came up with a very deep convolutional neural network that improved the classification accuracy. Iandola et al. introduced a SqueezeNet architecture [3] that substantially reduces the number of parameters in the network to save computational cost without harming the accuracy level. He et al. leveraged residual connections in deep networks to alleviate the vanishing/exploding gradients issues in [5]. With the use of residual connections and BatchNorm, ResNet architectures have become the state-of-the-art models for a plethora of vision tasks.

[1]The authors are with the University of California at Berkeley, Berkeley, CA, USA 94720 {harryhzhang, jingyi_song, guanzhitong,schan21} @berkeley.edu

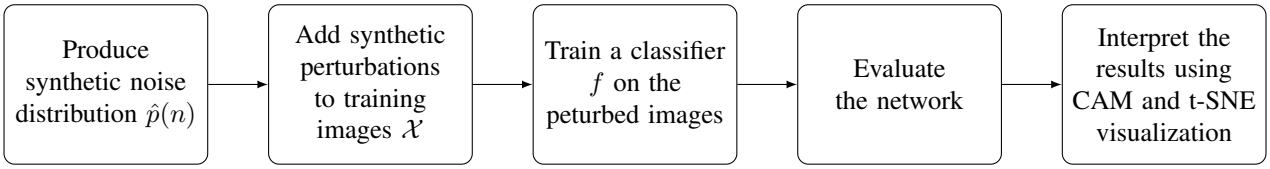


Fig. 2. Training and evaluation pipeline

b) Data Augmentation and Domain Randomization: In classical computer vision, affine transformations are a general approach to transform and deform the images [9], which is one of the methods we applied to the Tiny ImageNet dataset. Furthermore, affine transformations are also a common type of perturbation in real-world images. Perez et al. introduced a deep-learning-based approach to augment the data in [10]. By adding synthetic noise and perturbations to the training data, the learned classifier can be better generalized to novel data, as discussed in [11].

c) Classification Interpretability: Visualization is arguably the most effective way to increase the interpretability of the networks. Maaten and Hinton introduced t-SNE that projects the learned features onto a lower-dimensional space for visualization [8]. In [7], the authors introduced class activation mapping that can be applied to neural nets to see the regions in a image that a network is most "interested" in when classifying an image. Moreover, DrLIM [12] is another way to cluster similar data which uses unsupervised metric learning to learn a metric function that is invariant to dimensions.

III. Problem Definition

In training time, we wish to train a neural network image classifier f, that takes in an RGB image X, where $X \in \mathbb{R}^{H\times W\times 3}$, and outputs a vector y, where $y \in \mathbb{R}^C$ and C is the total number of classes in the dataset, that represents the probabilities of X belonging to each of the provided classes. We assume that each image in the test set is perturbed by some random noise n, where n is from an unknown distribution $p(n)$, and $n \sim p(n)$. Therefore, each testing image X_i becomes a perturbed image, which can be parameterized by

$$X_i \leftarrow X_i \oplus n_i$$

where $\oplus$ indicates that the perturbation is imposed on the image. Our goal is to correctly classify the test images under unknown perturbations. To achieve this, we add synthetic perturbations and noises to the training data during training time. We make a synthetic noise distribution $\hat{p}(n)$ and for each training image X_i, we infer the corresponding noise $\hat{n_i} \sim \hat{p}(n)$ and impose the noise onto the image X_i by

$$X_i \leftarrow X_i \oplus \hat{n}_i$$

We wish that under the perturbation generated from the synthetic noise distribution $\hat{p}(n)$, the network can learn enough information that will be able to have some robustness under the unknown noise distribution $p(n)$.

The evaluation metric we use in training, validation, and testing dataset is accuracy, which is defined as:

$$acc = \frac{N(f(X) = y)}{N(X)}$$

IV. Method

Our training and evaluation process is divided into five steps. Fig 2 shows the pipeline of our method.

A. Classifier Architecture

The classifier we use is a ResNet architecture. Specifically, we compare ResNet 50 and ResNet 101. Both networks have 4 building blocks. In a ResNet 101, the first building block contains 3 Bottleneck blocks, the second building block contains 4 Bottleneck blocks, the third building block contains 23 Bottleneck blocks, and the fourth building block contains 3 Bottleneck blocks, as introduced in [5]. In a ResNet 50, the first building block contains 3 Bottleneck blocks, the second building block contains 4 Bottleneck blocks, the third building block contains 6 Bottleneck blocks, and the fourth building block contains 3 Bottleneck blocks. The illustration of a Bottleneck block is shown in Fig. 3.

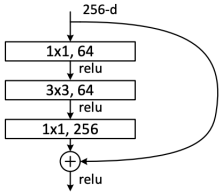
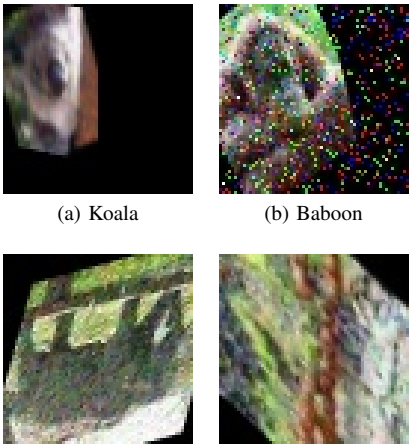


Fig. 3. An illustration of the bottleneck block used in ResNet 50 and ResNet 101.

B. Data Augmentation and Domain Randomization

To add randomization and synthetic perturbations to the images, we randomly apply the following actions to images in each batch with a probability p:

1) Random horizontal flips
2) Random vertical flips
3) Random affine transformations with shearing, translation, and scaling
4) Random crop
5) Gaussian noise
6) Gaussian filter blurring
7) Random rectangle occlusion
8) Salt and pepper noise



(a) Koala (b) Baboon

(c) Elephant (d) Chain

Fig. 4. Four examples of synthetic perturbed images. Specifically, Object a is perturbed by random affine transformation and random Gaussian noise, Object b is perturbed by random affine transformation, occlusion, and salt and pepper noise, Object c is perturbed by random horizontal flip, affine transformation, and Gaussian blurring, and Object d is perturbed by random vertical flip and shearing.

Here are some examples of the generated noisy images, as shown in Fig. 4. In each iteration, the random perturbations are applied to a random subset of the images. By applying the noise and perturbations to the training images, we augment the size of the training set and add randomness to the training set.

C. Overfitting Reduction

Deep networks are likely to overfit. To reduce overfitting, we used two techniques. The first technique is Dropout, as introduced by Srivastava el al. in [13]. Dropout mimics model ensemble in deep networks by randomly dropping some nodes' connections. In a convolutional neural network, this can be conveniently done before the fully connected layer.

The second overfitting reduction technique we used is weight decaying, which is also called L2 regularization of the weights. Mathematically, this can be expressed as:

$$L^* = L + \lambda ||w||_2^2$$

, where L is the original loss function, and L^* is the new loss function. By adding extra penalty on the magnitude of weights, we prevent the weights from getting too large, thus reducing overfitting.

D. Interpretability

To increase the interpretability of the network and better explain the classification results, we implement several methods to visualize the output of the classifier.

First, we apply class activation mapping (CAM) to the network [7]. The idea of CAM is similar to saliency maps [14], but CAM is able to leverage a global activation pooling to localize the discriminative image regions despite just being trained for solving classification task. ResNet makes CAM easy and straightforward to implement because in ResNet architecture, there is a global activation pooling (GAP) after the last convolutional layer, which is then fed into the fully connected layer. Therefore, we do not need any extra operations or layers to implement CAM in ResNets. CAMs take the weighted sum of the GAP output to extract the important features for sake of visualizing the discriminative regions, which indicates where in the input image the network is most "interested in".

Second, we also visualize the results using t-SNE [8]. t-SNE is a dimensionality-reduction technique that takes high-dimensional feature vectors in the feature space and projects them into a low-dimensional space that is suitable for visualization. Using t-SNE, points that are close in high-dimensional spaces should also be close in low-dimensional spaces. Therefore, t-SNE visualization can cluster images with similar features.

Third, we also visualize the convolutional filters as RGB images. Neural networks with non-linearity activation functions can be used to approximate any arbitrary function [15], [16]. Therefore, by visualizing the filters that the network learns, we are able to see how the network extracts information from the images. In most cases, just by visualizing the filters, we can see the shapes, edges, and colors that the filters are trying to extract.

V. Results

A. Training Details

The training dataset is comprised of 100k 64x64x3 RGB images from 200 classes. The validation dataset is comprised of 10k 64x64x3 different RGB images from the 200 classes. We apply data augmentation on the training images only, and evaluate the performance of the learned model on the unperturbed validation dataset. We used a batch size of 256. The probability of perturbation is $p = 0.7$ for each batch.

The difference between our ResNet and its counterpart proposed in [5] is that we apply a Dropout layer before the last fully connected layer. The use of Dropout layer is able to decrease the extent of overfitting [13].

We train the first network, ResNet 101, for 600 epochs. The loss function we use is Cross Entropy Loss, which is suitable for classification tasks. We start with a pretrained model that was trained on a normal ImageNet dataset. For the first 200 epochs, we start with a learning rate of 0.1, train the network upon the pretrained model, and decay the learning rate by 50% every 10 epochs. In the first 200 epochs, we set weight decay (λ in the L2 regularization) to 5e-4 and momentum to 0.9. The Dropout keep rate we use for the first 200 epochs is 0.8. After the 200 epochs, we save the weights of the current network and the corresponding training, validation loss and accuracy of each epoch for future use. Then in epochs 201-400, we start with the saved weights from epoch 200, and start with a learning rate of 0.01, decaying it by 50% every 20 epochs. In the second 200 epochs, we increase the weight decay parameter to 5e-3, by a factor of 10, and then we decrease the momentum to 0.85. The Dropout keep rate we use in the second 200 epochs is decreased to 0.5. Similarly, we save the weights, losses,

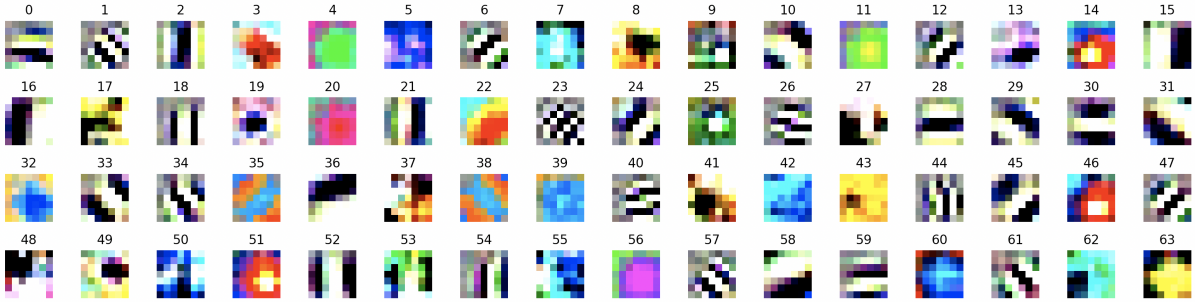


Fig. 5. Filters of the first convolutional layer in ResNet 101

and accuracies regularly. Lastly, in epochs 401-600, we start by restoring the weights from epoch 400 and begin with a lower learning rate of 0.005, and decay it by 50% every 20 epochs. In the third 200 epochs, we use 5e-3 weight decay parameter and 0.85 momentum.

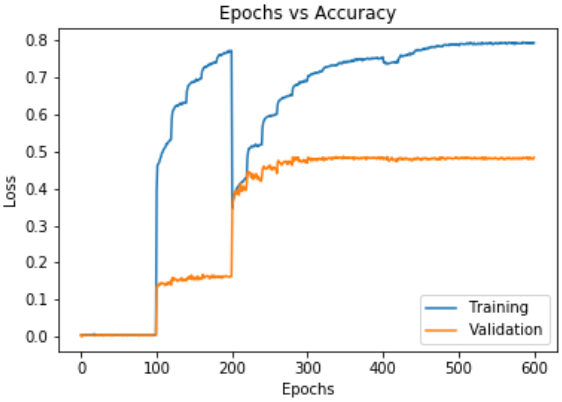
As a comparison, we train the second network, ResNet 50, for 300 epochs. We use the same loss function in ResNet 50. In the first 200 epochs, we start with a batch size of 256, a learning rate of 0.1, train the network upon the pretrained model that was trained on regular ImageNet, and decay the learning rate by 50% every 10 epochs. We use a weight decay of 1e-4 in the first 200 epochs, and we set the momentum to 0.9. In the last 100 epochs, we use a batch size of 128, and start with a learning rate of 0.01 and decay it by 50% every 15 epochs. During the last 100 epochs, we use a batch size of 256. The use of Dropout in our ResNet 50 is similar to our ResNet 101, where we start with a keep rate of 0.8 and decrease it to 0.5. Unlike ResNet 101, we also apply Nesterov momentum in ResNet 50.

The training process is done on Google Cloud platform, which is running a Nvidia Tesla V100 GPU. The code is written in PyTorch. In total, the time spent on training the ResNet 101 is about 10 hours. In comparison, the time spent on training the ResNet 50 is about 4 hours.

B. Models Comparison and Accuracies

For ResNet 101, we achieve a 79% training accuracy and a 49% validation accuracy. The loss and accuracy curves are shown in Fig. 6. From the plots, we can see that at first, the network that was pretrained on regular ImageNet does a bad job classifying the perturbed images. At epoch 95, there is a sharp increase in the training accuracy. At epoch 201, when we restore the weights from epoch 200 with a relatively high learning rate (0.01), the training accuracy drops since the step size is too big. Then we see a gradual increase of the accuracy and the loss in both training and validation. The curves resemble the learning curves pattern in the original ResNet paper [5]: due to the residual connections, we would expect to see a sharp decrease of the loss and a sharp increase of the accuracy at the beginning of the training process. We can also see that every time we decrease the learning rate, there is a decrease in training loss and an increase in training accuracy, especially in epoch 100-300. The increasing use of Dropout and weight decay, as expected, helps with reducing

(a) ResNet 101 Loss curve

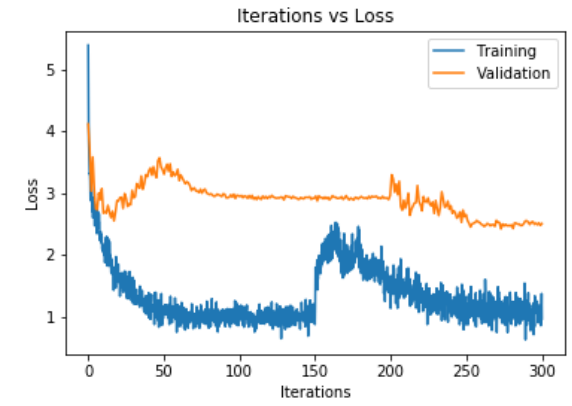


(b) ResNet 101 Accuracy curve

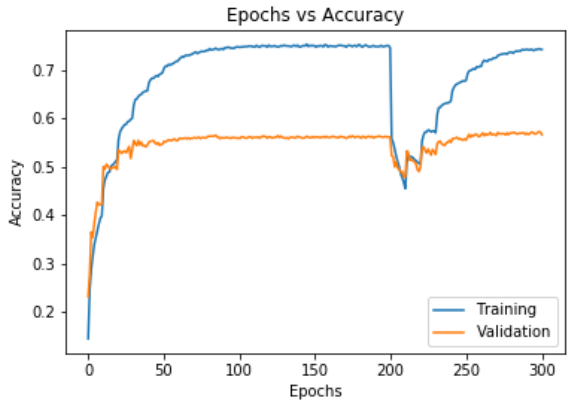
Fig. 6. **Top row:** training loss and validation loss plotted against the number of iterations in ResNet 101. **Bottom row:** training accuracy and validation accuracy plotted against the number of epochs in ResNet 101.

overfitting in the network, which can be seen in the increase of validation accuracy and the smaller gap between training and validation accuracies after 200 epochs.

For ResNet 50, we achieve a 70% training accuracy and a 57.3% valiation accuracy. The loss and accuracy curves are shown in Fig. 7. With fewer epochs, ResNet 50 is able to achieve a higher validation accuracy and the gap between the training and validation accuracies is smaller. From [5],



(a) ResNet 50 Loss curve



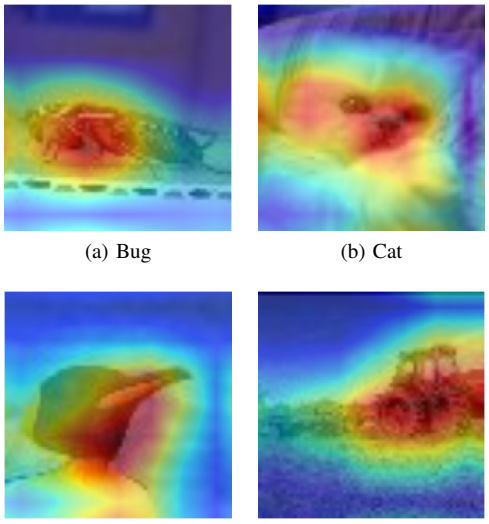
(b) ResNet 50 Accuracy curve

Fig. 7. **Top row:** training loss and validation loss plotted against the number of iterations in ResNet 50. **Bottom row:** training accuracy and validation accuracy plotted against the number of epochs in ResNet 50.

we know that ResNet 101 should achieve a better result than ResNet 50. The result we have here might seem counter-intuitive, but the result actually suggests a potential problem with ResNet 101, or deep neural networks in general: deep networks are sensitive to hyperparameters and are likely to overfit. Using ResNet 50, we are able to achieve a better result in overfitting reduction, thus a higher validation accuracy. Therefore, the results suggest that ResNet 50 is smaller than ResNet 101, but it is easier to tweak the hyperparameters in ResNet 50 to boost performance.

C. Interpretation

We first show a visualization of the filters from the first convolutional layer in ResNet 101 in Fig. 5. We choose to visualize the convolutional layer because it can extract the higher level information in the images. Meanwhile, we choose the convolutional layer from such an early stage in that the extracted information at this stage is still interpretable by human. For example, filter 18 and 30 in Fig. 5 correspond to a vertical edge detector and a horizontal edge detector, respectively. Filter 46 and 51 are likely circle detectors. Thus, we can extract some high-level information



(a) Bug (b) Cat

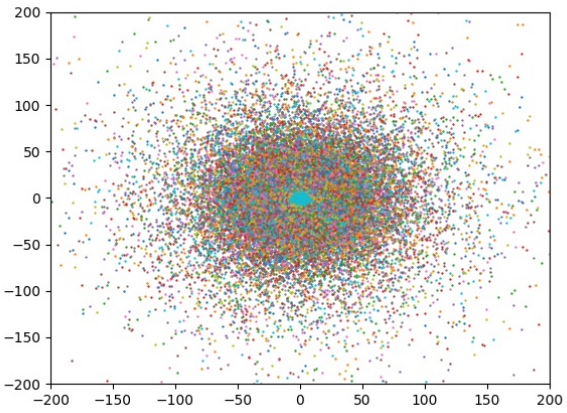
(c) Penguin (d) Tractor

Fig. 8. Four examples of class activation mapping (CAM) visualization on training images.

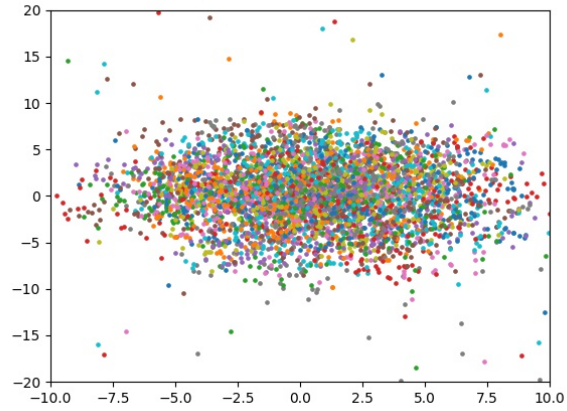
about the topology of the objects in the images using this visualization.

Next, we show some visualizations of class activation mapping (CAM) using the learned weights from ResNet 50. CAM is capable of visualizing the discriminative regions in the images. Our implementation is similar to the implementation from the original paper [7], where we overlay a heatmap of class scores onto the original image. The center of the class scores heatmap should correspond to the region in the image that the network shows the most interest to, and it should intuitively be the part of the object that is most representative of the object itself. We show 4 examples of CAM visualization on training images in Fig. 8. From the visualization, we have seen some interesting results. For the bug image (a), the class score is highest at the torso and feet area of the bug. For the cat image (b), the class score is highest at the face of the cat. For the penguin image (c), the class score is highest at the beak and neck area of the penguin. For the tractor image (d), the class score is high near the wheel and cockpit area of the tractor. All four results are closely tied to human perception in that the 4 areas with high class scores in the 4 images should also stimulate signals in human's receptive fields when classifying an image. To some extent, we can say that our neural network mimics human vision system when classifying the images, which was first shown by Hubel and Wiesel in [17]. Therefore, we can say that CAM does a fairly good job explaining the results of classification.

We also show visualization results using t-SNE. We run t-SNE on both training and validation images. The t-SNE implementation we use is from Sci-kit Learn, which does not support parallelism and GPU, so the running time of t-SNE is fairly long on the dataset. Specifically, it takes about 1.2 hours to make a full pass of training data to t-SNE and 6 minutes for validation data. The results of t-SNE are shown



(a) t-SNE on training data



(b) t-SNE on validation data

Fig. 9. t-SNE visualization on both training and validation datasets.

in Fig. 9. The results of t-SNE are not as satisfactory as CAM due to the sheer size of outliers population. Also, due to the large number of classes, and the likely cross-class similarity, the t-SNE projects are not well clustered. In the validation data visualization, we can see some clusters but they are all tangled up. We suspect that this is caused by the fact that 2D representation is not sufficient to represent the features of Tiny ImageNet dataset.

D. Challenges

There are a number of challenges we have encountered during training. First, the network is prone to overfitting due to the depth. Therefore, we modify the original ResNet architecture by adding Dropout. We also find that adding more penalty to the weights helps reduce overfitting too. Thus, we increase the strength of Dropout and weight penalty in the second 200 epochs.

Second, it is hard to find a good learning rate. A high learning rate is good for faster learning but may cause the network to fail to converge and a low learning rate is good for convergence but it might be too slow to learn. Therefore, realizing this tradeoff, we write a custom step learning rate decay scheduler, implemented in PyTorch. We also slow down the decay rate of learning rate towards the end of training because we would like to keep the results consistent. Starting with large learning rate explores the level sets in the cost function well and slowing down the gradient steps prevents the network from getting stuck at a plateau or local minimum.

Third, ImageNet/Tiny ImageNet itself is a difficult dataset to learn from. There is consistently a gap between the network's performance on the training set and the performance on the validation set. To be specific, there is a difference of 15% between training and validation accuracies in both ResNet 50 and ResNet 101. In the first 200 epochs, the difference is even larger due to overfitting, and if we reduce overfitting via L2 penalty and Dropout, the gap between training and validation accuracies becomes smaller but still exists.

VI. Conclusion

We present a trained image classifier based on ResNet that is robust to perturbations and noises in images. We compare ResNet 50 with ResNet 101 and conclude that ResNet 50 is easier to train. With an automated preprocessing procedure to add random noises and peturbations to Tiny ImageNet images, we make the classifier more robust to the perturbed images. We receive a validation accuracy that is on par with the current state-of-the-art classifier. Meanwhile, we try to make the learned neural network more interpretable and explainable by using various visualization techniques such as t-SNE and CAM. Specifically, CAM results demonstrate that the neural network focuses on the regions in the image that would stimulate most signals in human vision receptive fields when classifying the image.

However, a 57.3% validation accuracy is still too low for real-world applications, and the time spent on training ResNet 101 is too long. Therefore, in terms of future work, we would like to try some ensemble methods such as Boosting and Bagging, and we would also use networks with fewer parameters such as [3], [18] in order to reduce the computational cost.

Team Contributions: Harry first implemented a ResNet 50 and did a baseline, and finally decided to modify PyTorch's official ResNet 101. Harry and Zhitong wrote the training script and a helper script that formats the validation dataset in the correct way. Jenny and Shichao did the first half of data augmentation, and Harry and Zhitong wrote the second half of data augmentation. Harry implemented the t-SNE, CAM, and other postprocessing, visualization stuff. Harry trained the first 400 epochs, and Jenny and Zhitong trained the last 200. Jenny, Zhitong, and Shichao fine-tuned some hyperparameters to boost the performance of the model in later epochs. Harry and Jenny also did the literature review on data augmentation. Zhitong and Shichao did the literature review on deep neural networks.

REFERENCES

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

[2] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[3] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[7] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2921–2929.

[8] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[9] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[10] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[11] J. Tremblay, A. Prakash, D. Acuna, M. Brophy, V. Jampani, C. Anil, T. To, E. Cameracci, S. Boochoon, and S. Birchfield, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 969–977.

[12] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1735–1742.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[15] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[17] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[18] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.