Implementing The Newest Model And Optimizer For Image Classification

Ting-Liang Huang
Oregon State University
huangtin@oregonstate.edu

Kai-Hung Huang
Oregon State University
huangkai@oregonstate.edu

Abstract

Recent papers have shown that convolutional neural networks(CNNs) [9] perform remarkable results in image classification, such as VGG [10], Inception [11], ResNet [8], DenseNet [1], and EfficientNet [2]. Moreover, in order to speed up the time performance of the training process, optimizer acts as an important character as well such as Adam [12], diffGrad [5], and Ranger [6, 4, 3]. Last but not the least, data augmentation techniques [13] help increasing the robustness of CNNs which obtains higher test accuracy. Therefore, this paper will concentrate on finding the best scenario to make a breakthrough in image classification. The source code is made publicly available at https://github.com/LeohuangLeo/CIfar10_GTSRB-Pytorch.

1. Introduction

In this paper, we first explore the problem of original datasets in image classification and demonstrate the effectiveness of data augmentation techniques such as rotating, cropping, flipping, and contrasting which enhance the robustness of datasets to against noise. However, some drawback of image augmentation such as long-time training processes and inappropriate choosing transformers may cause worse results as well. The datasets we examine are The German Traffic Sign Recognition Benchmark (GTSRB) and Cifar-10 [14, 15]. GTSRB comprise of 40k training and 12k test images of dimensions for 32x32x3. There are total 43 distinct classes. Cifar-10 consists of 50k training set and 10k in the test set of dimensions 32x32x3. There are total 10 distinct classes. Since self driving cars are widely been investigated and implemented not only in academia but in industry, image detection technique acts an important character to recognize traffic sign, thus, it can largely decrease traffic accidents. Cifar-10 is another commonly used datasets. Even though the datasets only divide into ten classes, the complex of images is an extreme challenge for CNNs to extract useful features.

In addition, CNNs has become the dominant deep learning model for image and object detection. The original LeNet5 [7] consisted of 5 layers which started development of deep learning generation, Residual Networks (ResNets) [8] have surpassed 100 layers that reduce the problem of vanishing gradient, Densely Connected Convolutional Networks (DenseNets) [1] provided stronger connection between layers and layers which enhance feature reuse and increase time performance, and Rethinking Model Scaling for Convolutional Neural Networks (EfficientNets) [2] used compound scaling that achieved state-of-the-art accuracy.

Moreover, optimizers are another significant element to improve traininig and testing accuracies. Although SGD algorithm [16] outperform Adam algorithm finally considering accurate perspective, this will only happened only when the number of epoch close to one thousand which is time consuming. In this paper, we focus on Ranger and diffGrad optimizers which build on the top of Adam algorithm. Both of them show surprising results on thesis and there's rarely paper make a comparison of these two optimizers. Therefore, it's worth to evaluate which optimizer can take advantage in the field of image classification.

Despite some novel models and optimizers that researchers proposed in recent years largely solved some issues such as gradient vanishment and slow time convergence, or even can outperform human's knowledge, most papers are only focussing on specific part of deep learning architecture. In order to optimize the performance of deep learning architecture, we are looking for the best combination of model and optimizer to achieve a new level of milestone in image classification.

2. Related Work

Data Augmentation. has been shown as an effective way to improve model generalization and reduce overfitting on deep learning [17]. In this paper, we use the general data augmentation method includes: random rotation, horizontal flip, color jitter, random size crop for the original dataset. After data augmentation, we generalize about 10 times more data from the original datasets.

Data Normalization. can enhance the performance of the training model and make the whole training process becomes easier [18]. In our paper, we normalize each image we generate by using pytorch library to make a better result.

To select an appropriate model, we mainly focus on DenseNet and EfficientNet for comparison. DenseNet is CNN that adds a dense block to connect each layer densely, so each layer can take the preceding feature map as input. Owing to the dense structure that each layer can get the gradient of loss function and original input directly, so it can lead to an implicit deep supervision. Also, because the size of feature maps are small, the number of parameters for DenseNet is much smaller than CNN models like LesNet and ResNet. EfficientNet is another CNN that uses MobileNet as baseline with a new scaling technique called compound scaling method to scale the width, depth and resolution appropriately. Therefore, these two training model both can get great performance on image classification with many fewer parameters than other SOTA model like ResNet and LesNet [1] [2]. In our paper, we use these two models on GTSRB dataset to compare the performance based on the accuracy of testing data.

Moreover, we also use different optimizer for comparison that is intended to get a best performance on image classification. To get the best performance with less number of epoch, we focus on two optimizer which are Diffgrad and Ranger [5][6]. The optimizer Diffgrad will remember both previous change of gradient and current change of gradient, and it uses an adjustable friction clamping that can let parameter update slower to avoid getting stuck in a local minimum. The optimizer Ranger that combines Rectified Adam and Lookahead, so it can have a better start compare to other optimizer and also ensure to achieve the global minimum. We compare the performance of these two optimizer by using them on the two training model DenseNet and EfficientNet.

3. Methodology

The following subsections explain the principle of deep learning techniques that we're going to implement in this paper. The methodology will be divided by three parts: data preprocessing, model architecture, and optimizer description.

3.1. Data Preprocessing

Data preprocessing is extremely important before starting to train datasets. The most efficient way to train neural networks is to increase the number of training datasets. This is the main reason why we use image augmentation to do data preprocessing. Moreover, normalization technique reduces the influence of redundant data. Torchvision [19] will be the tool that we implement image augmentation.

GTSRB. GTSRB is relatively easier dataset comparing to Cifar-10 because of its simplicity. Most of augmentation techniques in torchvision such as rotating, cropping, and flipping.

Cifar-10. Comparing GTSRB, Cifar-10 is relatively complex dataset even the number of its classification is weight more smaller. However, we carefully selected appropriate transformers and finally concatenate those features together. Therefore, the quantity of datasets will be several times larger than the original one.

3.2. Model Architectures

DenseNet and EfficientNet are two state-of-the-art models that are trying to solve long running time that ResNet has and dimension problem that CNNs are always looking for the best shape of the model.

3.2.1 DenseNet.

Architecture of DenseNet can roughly divide into six parts as below:

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Figure 1. DenseNet architectures for ImageNet. The growth rate for all the networks is k = 32. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.[1]

3.2.1.1 Dense connectivity.

Huang, Gao, et al.[1] introduce direct connection from any layer to all subsequent layers. Figure 1 illustrates the layout of the resulting DenseNet schematically.

3.2.1.2 Composite function[1].

Composite function consisted of three consecutive operations: batch normalization (BN) [20], followed by a rectified linear unit (ReLU) [21] and a 3x3 convolution (Conv).

3.2.1.3 Pooling layers.

Huang, Gao, et al.[1] refer to layers between blocks as transition layers. In addition, the transition layers used in the experiments consist of a batch normalization layer and a 1x1 convolutional layer followed by a 2x2 average pooling layer.

3.2.1.4 Growth rate.

Growth rate k, the number of channels, regulates how much new information each layer contributes to the global state. [1]

3.2.1.5 Bottleneck layers.

Huang, Gao, et al.[1] implement 1x1 convolution to improve computational efficiency which reduces the number of input feature-maps.

3.2.1.6 Compression.

To further improve model compactness, Huang, Gao, et al.[1] reduce the number of feature-maps at transition layers

3.2.2 EfficientNet.

3.2.2.1 Compound Model Scaling.

MingXing, Tan, et al.[2] develop a new compound scaling method to scale the model by a compound coefficient ϕ, and the formula shows below[2]:

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

where α, β, γ are constant can be calculated by grid search.

3.2.2.2 Architecture.

The baseline of EfficientNet is based on MnasNet[22], but the model is modified by optimizing FLOPS rather than latency. The way is to use compound scaling method on the model can be divided in two steps.
First, initializing the value of ϕ, and doing small grid search to obtain α, β, γ by the equation above.
Second, scaling up the baseline network by the equation mentioned above to get EfficientNet-B1 to EfficientNet-B7.

3.3. Optimizer Description

The following optimizers are proposed in recent months and both of them address core problem that current optimizers are facing nowadays.

Diffgrad. According to Dubey, Shiv Ram, et al.[5], the step size of diffGrad optimization is adjusted for each parameter in such a way that it should have a larger step size for faster gradient changing parameters and lower step size for lower gradient changing parameters.

Ranger. Ranger [6] consisted of RAdam [4] + LookAhead [3]. According to Liyuan Liu, et al. [4], RAdam algorithm (Figure 2) not only explicitly rectifies the variance and is theoretically sound, but also compares favorably with the heuristic warm up. On the other hand, LookAhead algorithm (Figure 3), slow and fast weight updates, help smooth out the oscillations through the parameter interpolation and make rapid progress along the low curvature direction [3]. It further reduces variance and then finally enables to converge rapidly in practice.

```
Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\beta_1, \beta_2\}$: decay rate to calculate moving average and moving 2nd
       moment, $\theta_0$: initial parameter, $f_t(\theta)$: stochastic objective function.
Output: $\theta_t$: resulting parameters
1  $m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)
2  $\rho_\infty \leftarrow 2/(1-\beta_2) - 1$ (Compute the maximum length of the approximated SMA)
3  while $t = \{1, \cdots, T\}$ do
4      $g_t \leftarrow \Delta_\theta f_t(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)
5      $v_t \leftarrow 1/\beta_2 v_{t-1} + (1-\beta_2) g_t^2$ (Update exponential moving 2nd moment)
6      $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ (Update exponential moving 1st moment)
7      $\widehat{m_t} \leftarrow m_t/(1-\beta_1^t)$ (Compute bias-corrected moving average)
8      $\rho_t \leftarrow \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$(Compute the length of the approximated SMA)
9      if the variance is tractable, i.e., $\rho_t > 4$ then
10         $l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$ (Compute adaptive learning rate)
11         $r_t \leftarrow \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$ (Compute the variance rectification term)
12         $\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \widehat{m_t} l_t$ (Update parameters with adaptive momentum)
13     else
14         $\theta_t \leftarrow \theta_{t-1} - \alpha_t \widehat{m_t}$ (Update parameters with un-adapted momentum)
15 return $\theta_T$
```

Figure 2. **Algorithm:** Rectified Adam. All operations are element-wise

```
Require: Initial parameters $\phi_0$, objective function $L$
Require: Synchronization period $k$, slow weights step
  size $\alpha$, optimizer $A$
  for $t = 1, 2, \ldots$ do
    Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
    for $i = 1, 2, \ldots, k$ do
      sample minibatch of data $d \sim \mathcal{D}$
      $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
    end for
    Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
  end for
  return parameters $\phi$
```

Figure 3. **Algorithm:** Lookahead Optimizer

4. Results

In the simulation results, we use pelican server with GTX1080 hardware environment. Pytorch and Tensorboard are used for training datasets and visualization work. We first evaluate image augmentation, models, and optimizers separately. Lastly, we will choose the best architecture based on the results from above to train GTSRB and Cifar-10.

4.1. Image Augmentation

Figure 4 shows the image of GTSRB. This is the results from image augmentation. It's obvious that some traffic sign rotated and even changes the brightness. These small noise successfully simulate some traffic image may have in the real world.

Figure 5 shows the image of Cifar-10. This is the results from image augmentation as well. Considering the complexity of image, some augmentation techniques such as brightness and contrasting not only help test accuracy but also increase training time.

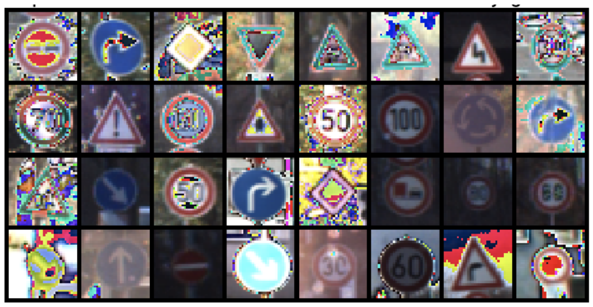
Figure 4. GTSRB dataset visualization after image augmentation

Figure 5. Cifar-10 dataset visualization after image augmentation

4.2. Model Comparison

Figure 6 shows the result of testing accuracy on GTSRB that makes a comparison of DenseNet121 and EfficientNet-b5. Both models trained in epoch=20, batch_size=32, optimizer=Adam. DenseNet121 can be easier to attach local minimum comparing to Efficient-b5. However, both models can attach roughly same local minimum in 20 epochs. The highest test accuracy in DenseNet121=97.12% and EfficientNet-b5=97.32%

4.3. Optimizer Comparison

Figure 7 shows the evaluation of training results comparing current the-state-of-art optimizers Adam, diffGrad, and Ranger. In view of running time, Adam and Ranger outperform diffGrad. In view of test accuracy, Ranger outperform the rest optimizers. Therefore, we will use ranger to be our best optimier to train the model in the next section.

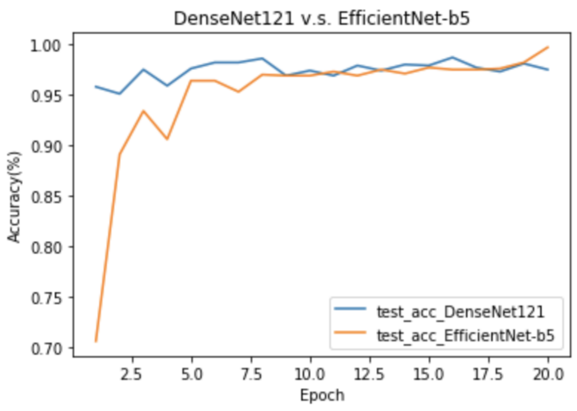

Figure 6. Model comparison between DenseNet121 and EfficientNet-b5 on GTSRB

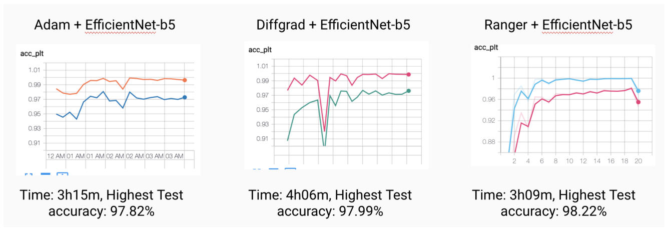

Figure 7. Optimizer comparison among Adam, diffGrad, and Ranger on GTSRB

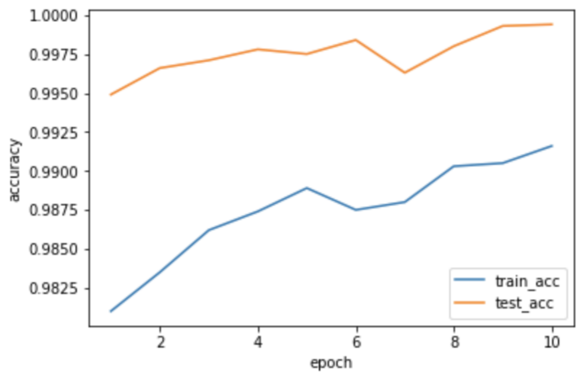

Figure 8. Result from EfficientNet-b7 model + Ranger optimizer + Image Augmentation on GTSRB

4.4. Evaluation of the Best Scenario

GTSRB. From the evaluation results above, we decided to use DenseNet either DenseNet or efficientNet as our model. Since both of them have the same performance in image classification. In turns of running performance, we will prefer to use DenseNet. Moveover, it is necessary to use pre-trained model to train efficientNet. However, we wouldn't confirm that if this pre-trained model is perfectly match the training data we're going to use. Ranger will be optimizer and add some image augmentation of data preprocessing. The result in figure 8 shows amazing performance.

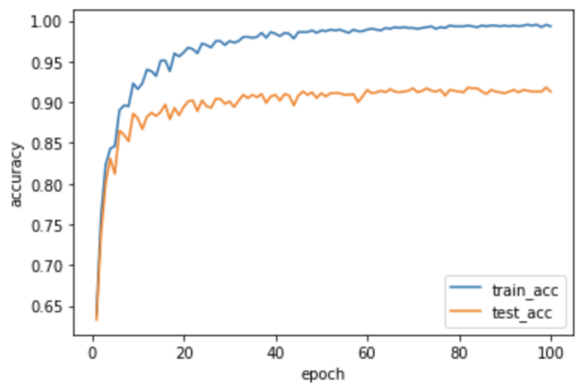

Figure 9. Result from DenseNet model + Ranger optimizer + Image Augmentation on Cifar-10

The highest test accuracy is 99.98% which beat the current test accuracy 99.71% from [23].

Cifar-10. Figure 9 shows the training process of DenseNet for cifar-10. The highest test accuracy we can achieve is 92.57% which is lower than we expected. Starting from epoch 25, test accuracy stuck in the certain points. Multiple reasons can explain this problem such as the initialization setting cause the whole model stop in certain local minimum. However, the results we achieved still outperform the general CNNs.

5. Conclusions

This is the first paper make a comparison of Ranger and diffGrad optimizers. In dataset GTSRB, we obtained Unprecedented 99.98% test accuracy. In dataset cifar-10, we obtained 92.57% test accuracy that general CNNs cannot achieve. Overall, this paper summarizes the newest models and optimizer addressing on image classification.

6. Future Works

Future work will focus on the structure of models, even though efficientNet and DenseNet have better performance than ResNet in most of the cases. Recent papers such as [24] shows that BiT-L (ResNet) can obtain 99.3% test accuracy in cifar-10 with extra training data which is way higher than our result. The method ”PyramidNet + ShakeDrop + Fast AA + FMix” [25] obtains 98.64% test accuracy without extra training data in cifar-10. These give us more direction to improve the model that we implemented.

References

[1] Huang, Gao, et al. “Densely Connected Convolutional Networks.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, doi:10.1109/cvpr.2017.243.

[2] M. Tan and Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. 36th Int. Conf. Mach. Learn., 2019, pp. 6105–6114.

[3] Zhang, Michael R., James Lucas, Geoffrey Hinton, and Jimmy Ba. ”Lookahead Optimizer: k steps forward, 1 step back.” arXiv preprint arXiv:1907.08610 (2019).

[4] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han, “On the variance of the adaptive learning rate and beyond,” arXiv:1908.03265 (2019).

[5] Dubey, Shiv Ram, et al. “DiffGrad: An Optimization Method for Convolutional Neural Networks.” IEEE Transactions on Neural Networks and Learning Systems, 2019, pp. 1–12., doi:10.1109/tnnls.2019.2955777.

[6] Wright. “New Deep Learning Optimizer, Ranger: Synergistic Combination of RAdam LookAhead for the Best of...” Medium, Medium, 4 Sept. 2019, medium.com/@lessw/new-deep-learning-optimizer-ranger-synergistic-combination-of-radam-lookahead-for-the-best-of-2dc83f79a48d.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradientbased learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In CVPR, 2016.

[9] Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” Communications of the ACM, vol. 60, no. 6, 2017, pp. 84–90., doi:10.1145/3065386.

[10] Liu, Shuying, and Weihong Deng. “Very Deep Convolutional Neural Network Based Image Classification Using Small Training Sample Size.” 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), 2015, doi:10.1109/acpr.2015.7486599.

[11] Szegedy, Christian, et al. “Rethinking the Inception Architecture for Computer Vision.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, doi:10.1109/cvpr.2016.308.

[12] Kolkiewicz, Adam. “Stochastic Mesh Method.” Encyclopedia of Quantitative Finance, 2010, doi:10.1002/9780470061602.eqf13013.

[13] Mikolajczyk, Agnieszka, and Michal Grochowski. “Data Augmentation for Improving Deep Learning in Image Classification Problem.” 2018 International

Interdisciplinary PhD Workshop (IIPhDW), 2018, doi:10.1109/iiphdw.2018.8388338.

[14] “The German Traffic Sign Recognition Benchmark.” benchmark.ini.rub.de/?section=gtsrbsubsection=news

[15] CIFAR-10, www.cs.toronto.edu/ kriz/cifar.html.

[16] Ruder, and Sebastian. “An Overview of Gradient Descent Optimization Algorithms.” ArXiv.org, 15 June 2017, arxiv.org/abs/1609.04747.

[17] Shorten, C., Khoshgoftaar, T.M. A survey on Image Data Augmentation for Deep Learning. J Big Data 6, 60 (2019). https://doi.org/10.1186/s40537-019-0197-0

[18] J. Sola and J. Sevilla, ”Importance of input data normalization for the application of neural networks to complex industrial problems,” in IEEE Transactions on Nuclear Science, vol. 44, no. 3, pp. 1464-1468, June 1997.

[19] Pytorch,https://pytorch.org/

[20] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

[21] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In AISTATS, 2011.

[22] xM. Tan et al., ”MnasNet: Platform-Aware Neural Architecture Search for Mobile,” 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019, pp. 2815-2823.

[23] CNN with 3 Spatial Transformers, DeepKnowledge Seville, Álvaro Arcos-García and Juan A. Álvarez-García and Luis M. Soria-Morillo, Neural Networks

[24] Kolesnikov, et al. “Large Scale Learning of General Visual Representations for Transfer.” ArXiv.org, 24 Dec. 2019, arxiv.org/abs/1912.11370.

[25] Harris, et al. ”Understanding and Enhancing Mixed Sample Data Augmentation” ArXiv:2002.12047