Deep Networks for Image Classification

Konstantinos Skoularikis

May 18, 2021

Abstract

Deep neural networks have been been a matter of great interest to researchers around the globe for the past 15 years. Novice architectures and learning strategies have been proposed, enabling the training of such networks. Focusing on three major implementations [13, 15, 6], this paper describes their workings, presents state of the art models based upon their concepts and proposes alternative architectures and training strategies for each one individually, inspired by latest developments on the field. The evaluation of the vanilla and improved models is performed on the MNIST and CIFAR10 datasets. Code is available through the accompanying notebooks.

1 Introduction

Over the course of the years, convolution based neural networks have dominated the computer vision field due to their ability to identity hierarchical and spatial data patterns in an image through the use of filters while requiring little pre-processing compared to other methods. Particularly, for image classification where the goal is to output a class or more accurately, a probability that the image belongs to a certain class, a plethora of research papers have been published, proposing alternative convolutional architectures and training methods for achieving better results on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [12].

Among the numerous improvements, recent findings suggest that the network's depth is crucial for enriching the "levels" of low/mid/high features [6], which can positively impact performance. Creating very deep networks, however, is not an easy task since in practice the accuracy gets saturated and degrades rapidly when increasing the number of layers, regardless of whether the model is overfitting or not.

In this context, three models were introduced, each one with its unique characteristics, focusing on increasing the convolutional network depth while minimizing the aforementioned degradation problem. Their principles can be used in other areas of computer vision apart from image recognition and in fact many computer vision models today owe their success to these breakthroughs.

The rest of the paper is organized as follows. The unique traits of each model are described in Section 2. In Section 3, recent workings based on the basic models' ideas are discussed and critically analyzed while alternative designs and training methods are proposed for each model, in Section 4. The original and improved models are deployed in Section 5 on the MNIST and CIFAR10 datasets to acquire tangible experimental results. Finally, Section 6 concludes the paper. All relative images and plots will be placed in the appendix.

2 Model Description

VGG: The authors of the paper[13] introduced the concept of using stacked 3x3 convolution layers, which can replace the layers with large receptive fields, as used in [9, 18]. These convolution layers were not picked at random. Their kernel size of 3x3 has the minimum receptive field, needed to capture the notion of left/right, up/down and center, which is why a stack of three of them will have the same receptive field as a 7x7 convolution layer. Their difference, the main idea of this paper, lies upon that fact that non-linearity is applied three times now instead one, which results in a more discriminative desicion function, while the number of parameters used by the network is dramatically decreased. Furthermore, in some variations of the initial model, they also incorporated 1x1 convolution layers to linearly project the input channels, which further increased the non-linearity of the decision function while keeping a space of the same dimensionality. The same idea

has been recently utilized in [10]. Regarding the intricacies of the model, all hidden layers use the rectification (ReLU [9]) non-linearity, while spatial pooling is done by five max-pooling layers with 2x2 window size and stride 2, which position depends upon the network's configuration. An adaptive average pooling with window size 7x7 pools the features of the last convolution to feed their compressed representation to a stack of three fully connected layers, which are responsible for the classification of the images. The detailed architecture of each model is shown in Table 1. It should be mentioned that the authors used the same architecture for other computer vision tasks, such as localization, with minimal changes.

Training Process: ConvNet, alternative name to VGG, optimizes a rather common objective, namely a multinomial logistic regression using mini-batch (with size, 256) gradient descent with momentum. Weight decay and dropout layers with ratio 0.5 were used to regularize the training. In some cases, network's weights were initialized from a pre-trained model or a probability distribution like [4], which proved to be essential for accelerating and avoiding instability during the learning process. A decaying learning rate was used to push the gradient to an optimal point in the space, while various data augmentation techniques were implemented (i.e. random cropping, horizontal flipping, random RGB colour shift) which improved model's final accuracy and generalization.

ResNet: ResNet's [6] contribution to alleviating the degradatation problem was the incorporation of residual learning for the training of very deep learning models. As the name suggests, the new objective is to learn an alternative underlying residual mapping. This residual mapping reformulation suggests that the training of the added layers should produce a training error no greater than a shallower version of the network, providing they approximate the identity mappings. Eq. (1) displays the new mapping the stacked nonlinear layers will fit upon.

$$F(x) := H(x) - x \tag{1}$$

Here $F(x)$ represents our residual function, $H(x)$ is the underlying mapping for a few stacked layers and x denotes the input to the first layer of the stack. Eq. (1) can also be written as:

$$y = F(x, W_i) + x \tag{2}$$

The form of the residual F can be different, depending on the number of layers it is entailed with. No apparent advantages are observed in the case where F contains only a single layer because Eq. (2) behaves as a linear operator, hence the authors of the paper use a function F that encapsulates two or three layers. In this context, the operation $F + x$ is performed by a shortcut connection and element-wise addition, which does not increase the model's computation complexity. To account for the dimensionality difference during training (the number of the input and output channels are not the same), Eq. (2) incorporates a weight matrix W_s when needed, which linearly projects the input x.

Shortcut connections[2], in general, are used to skip one or more layers of the model. In this case, they simply perform identity mapping and their output is added to the output of the stacked layers. Fig. 1 provides a visual insight on such a residual block with convolution layers.

Architectural design: The majority of network's convolution layers use 3x3 filters, while following two simple design rules, depending on the output feature map size. The network's higher layers(deeper layers) output is averaged by a global average pooling layer, which is fed into a 1000-way fully connected layer with softmax. This depends on the model's purpose which in this case, is classification, hence the choice of last dense layer. Nevertheless, the model's base architecture is general enough to be utilized for other computer vision tasks as well. Based on the above incomplete architecture, shortcut connections are introduced, transforming it into a residual model. There are two options to be considered again depending on the difference in dimensions between the input and the output. For a detailed description of the model's design, refer to the paper's [6] *section 3.3*.

Training Process: Being very similar to VGG's training process with regard to techniques used (i.e penalty for L2, objective function, weights initialization and data augmentation methodologies) and not to the actual values used by each one of them, a few key points will be mentioned here, providing the reader with a basic understanding of ResNet's implementation. BN(batch normalization [8]) layers were adopted right after each convolution layer and before the application of non-linearity (ReLU [9]). Their weight and bias were initialized to 1 and 0, respectively. All layers were trained from scratch. This includes both plain and residual units. Dropout layers were not used in this case as suggested from [8], since batch normalization acts as a regularization fac-

tor on the model's forward propagated signals.

GoogLeNet: The famous model, GoogLeNet[15], introduced the new "Inception module", named after the alternative model's codename "Inception". In an attempt to mimic biological systems and overcome the drawbacks of traditional wide and deep networks, the Inception architecture tries to find the optimal local sparse structure of convolutional building blocks while making use of dense components, suitable for today's software and hardware computations. Driven by this idea, the proposed design entails repetitions of single output vectors, as the direct result of the concatenation of individual outputs from 1x1, 3x3 and 5x5 convolution layers on the same input features, respectively. In addition, to avoid the dramatic increase in computational requirements and aid in the process of finding a more discriminative decision function, dimensionality reductions and projections, performed by 1x1 convolutions, are incorporated into the aforementioned pipeline before the expensive 3x3 and 5x5 convolutions. Aside from utilizing these types of modules stacked upon each other, Inception uses occasionally max-pooling layers with stride 2 between them to handle the overhead of the network, as well as traditional convolution layers in the beginning of the network (lower layers) to account for memory inefficiencies. Fig. 2 provides a visual illustration of the final Inception module. Worth noting is that, all convolution layers, regardless of their place and filter size, have a ReLU [9] activation function to accompany them while the classifier consists of a global average pooling layer followed by a dropout layer, which is essential according to the authors. Due to the task being multi-class classification, a softmax function is used. Of course, this can vary, depending on the nature of problem in need of solving. Finally, optional but not trivial is the use of auxiliary classifiers. They are connected to the intermediate layers of the overall Inception architecture and act as a regularization aspect, increasing the back propagated gradient signal. Their loss is added partially (discount weight factor - 0.3) to the total network's loss during training while they are ignored during inference.

The benefits that the above design comes with, is the ability to increase the number of layers in the network without increasing the computational requirements exponentially. We can also easily understand that the nature of the "Inception module" promotes the concept of processing visual information on different scales simultaneously.

Training process: Briefly referring to the training methodology of GoogLeNet, we can say that it resembles both the ones used in VGG and ResNet. In fact, the only instrumental difference between them except for the actual learning rate, learning decay percentage values and data augmentation techniques used, lies in the training data. Various sized crops (some smaller, some larger) were used for training while the crop's sample for each patch, contained different images both on size and aspect ratio. For the avid reader, the complete training methodology can be found in section 6 of the paper [15].

3 Critical Analysis

As mentioned before in Section 1 VGG, ResNet and Inception models are used extensively nowdays for a number of computer vision applications, hence the attention of the research community is turned towards, improving their structures and training strategies. This section will briefly summarize some of the noteworthy efforts while accessing the reasons behind these enhancements.

Going through the relative literature review for the first model, namely VGG, someone will find out that there are no explicit improvements or major changes the model's architecture. In fact, small adjustments, which are usually task-dependent, are prevalent up to this date. To elaborate, structural changes revolve around removing convolution layers in accordance with available dataset's complexity as well as introducing batch normalization [8] layers, between the convolution or after max-pooling layers, to stabilize training process, counter the vanishing/exploding gradient problem and combat the internal covariance shift due to the simultaneous layer's weights update. Particularly for classification tasks, researchers were interested in replacing the last part of model, its classifier. The rationality behind these attempts is the extremely large number of parameters that these layers handle, which may cause the model prone to overfitting. Another possible reason is, reducing the computational burden placed on the network and discarding the dropout layers which aggravate performance when used in conjunction with batch normalization [8]. Therefore, they suggest using a GAP layer instead, which retains feature information by combining the latest input feature map into a whole. It worth noting that the principles of VGG (i.e stacking 3x3 convolution layers to replace larger kernel sizes convolutions) and their variations have been employed from other state-of-art models' construction. [16, 17]

Unlike VGG's case, there is an abundance of models based on ResNet's philosophy. Focusing on the residual block itself, ResNetXt[17] introduces a new dimension/hyperparameter called cardinality, denoting the number of paths(from [6] the internal dimension for each path is d=4), which allows different paths to be internally "splitted-transformed-merged" by pointwise grouped convolution for each block, ultimately sharing the same topology unlike the similar-looking Inception [15]. As pointed out by the authors of the paper, the new model maintains the same complexity and number of parameters as [6] but achieves higher accuracy by a large margin. Other works, such as Densely connected cnn [7] further exploit the effects of shortcut connections. The input of each layer consists of the feature maps (depth-concatenated) of all earlier layer, and its output is passed to each subsequent layer. This idea makes the network parameter-efficient and promotes feature reuse since the output of identity mapping acts as a delaying factor for information flow on layers with dissimilar distributions, which evidently increase block's output variance. Recent papers (e.g [1, 3]) centre their attention around the use of recently-proposed regularization and network's/images' scaling strategies as well as the incorporation of self-attention while performing minor changes on the baseline model. The former [1], ResNet-RS, not only achieves stronger performance and significantly faster training time(2.1x - 3.3x on GPUs) on ImageNet than previous models but also generalizes well on a diverse set of computer vision tasks both on 2D(images) and 3D(video) data where labels may be partially present. The latter[3], can slightly outperform some "classic" ResNet based models due to its ability to capture the most important global features from the sequence of convolutional outputs, providing there is a large dataset available. The authors splitted the image into patches and provided the sequence of linear embeddings as the input to the Transformer.

The last model's, Inception [15], improvements can be mainly attributed to Google researchers. [16] premise (Inception v2) was the reduction of representational bottleneck produced from drastically changing the input's dimension, while retaining computational complexity by smart factorizing methods. This was achieved by changing the model's "expensive" (i.e. with 5x5 receptive field) convolution layers to a "cheaper" stack of smaller 3x3 convolutions, factorizing existing convolutions(nxn) to a combination of 1xn and nx1 and finally widening model's filter banks. In the same paper, Inception v3, incorporates all of the above, but also adds batch normalization layers to auxiliary units, factorizes initial 7x7 convolution, changes optimizer to RMSProp and finally implements label smoothing. Most of these techniques were discussed before in previous sections, hence their advantages will not be analyzed. Label smoothing will be further discussed in Section 4 due to its usage in the proposed models. Since some modules are more complicated than necessary, Inception v4[14] introduced specialized reduction blocks, which were responsible for adjusting the width and the height of the grid. Finally, Inception-Resnet v1 and v2[14] incorporated the previously mentioned residual connections which adds the output of the convolution operation of the inception module, to the input, gaining the extensive advantages that residual units offer.

4 Improved Models

The three "improved" models (one for each baseline model) have common training strategies. We choose to describe them first to give the reader a basic understanding of the reasons behind their use.

Weight decay: A regularization technique where a small penalty is added to the L2 norm of the weights that our loss function uses. It is used to prevent the network from overfitting, because it retains small values for weights during training, avoiding problems such as exploding gradient. Pytorch adds this penalty to both the weights and the bias.

Label Smoothing[11]: Another regularization method that tackles overfitting and overconfidence. An overconfident model has predicted probabilities for each class which do not resemble the real per class accuracy values, but are instead higher. It introduces noise to the "hard" classifications targets(labels) replacing their values with smooth labels. For instance, for the one-hot encoded labels(e.g [1,0,0]), each 0 will become $alpha/k$ while 1 will be $1-alpha$, where $alpha$ is a hyperparameter that determines the amount of smoothing and k is the number of classes. In our case, where our loss function is cross entropy, label smoothing is calculated as $alpha * loss + (1 - alpha) * y$, where $loss$ is the mean of the sum of negative log probabilities produced by the our softmax divided by batch size and y is the negative log likelihood loss.

Improved VGG: The proposed VGG16 based model, uses batch normalization layers after each convolution layer and before the activation func-

tion(ReLU) to fix layer's input mean and variance to 0 and 1. Another change is the replacement of the 7x7 Adaptive Average Pooling layer of the default pre-built Pytorch implementation with an 5x5 Adaptive Max Pooling one. This was primarily done to reduce the number parameters passed into the classifier and subsequently model's parameters as well as acquire the maximum value for each patch on a smaller area (more local information) of the final feature map due to the initial reduced image size. To take it a step further, convolution layers were initialized by the method proposed in [5] using a normal distribution while linear layers' weights values were randomly sampled from a Gaussian distribution(mean=0,std=0.01). Batch normalization layers' weights and biases were initialized with fixed values of unit and zero, respectively.

Improved ResNet: Influenced by VGG's principles, the new network, based on ResNet18, uses a stack of three 3x3 convolution layers instead of a large 7x7 convolution layer. Batch normalization layers are added between each convolution layer and before their activation. The pooling and stride for each convolution layer were picked appropriately to maintain the same dimensions as the original model. Max pooling layer remained the same as before. Finally, a dropout layer with ratio 0.1 was placed before the linear classifier, which can have a positive impact on performance when combined with the aforementioned regularization techniques [1]. The weights and biases of the network's layers were initialized from the same process as the improved VGG model.

Improved GoogLeNet: Minor were the changes for the Inception model. There is no specific modification to original Inception module. Instead, the initial 7x7 convolution layer is replaced by a stack of three 3x3 convolutions, inspired by VGG's principles.

5 Experiments

5.1 Training process:

Vanilla Models: To simplify the training process of the vanilla models[13, 15, 6], a fixed learning rate(0.001) was selected. The models were trained for 50 epochs due to limited resources. The training is carried out by optimizing a stochastic gradient descent with momentum(0.9) on a batch size equal to 128.

Improved Models: The learning process for the proposed models is more complicated. The same batch size and number of epochs is selected. SGD with the same momentum but with a weight decay factor of 4e-5 is used. The cross-entropy loss function is replaced by its label-smoothed counterpart. The base learning rate is set on 0.01 but it decays by gamma=0.1 when reaching the milestones of 5,27,40,47 (epochs) respectively. Fig. 12 and Fig. 11 display the training logs for baselines models on MNIST and CIFAR10.

Data augmentation: The data augmentation techniques used are dataset-dependent and they were only applied to the proposed models. For MNIST, the images are resized to 64x64 pixels via bilinear interpolation. They were also rotated by 15 degrees, translated(shifted) by random sampling both on the horizontal and vertical axes with a shift value of 0.1 and scaled randomly with a scaling factor of 0.8 to 1.2 of the image's original size. For CIFAR10, the inputs were also resized to a size of 64 with the same operation, horizontal flipped, random rotated(10 degrees), sheared(10 degrees) and translated and scaled similarly with MNIST. Color jittering was also performed to adjust the brightness(0.2), contrast(0.2) and saturation(0.2). Finally, regardless of the dataset, training datapoints(inputs) were normalized by taking the mean of their per channel mean and standard deviation of their pixels. During inference, the only augmentation performed is a bilinear interpolation to resize the input to necessary size(64). Fig. 14 and Fig. 13 illustrate the training logs for the improved models on MNIST and CIFAR10.

5.2 Datasets

MNIST: Contains a training dataset of 60000 grayscale images with individual size 28x28. Its testing dataset is 1/6 the size of the training dataset, having images with the same attributes. The chosen validation set, used for evaluation, has the same size as well. The displayed images are random numbers(integers) between 0-9. The number is almost certainly at its center, constructed by white pixels. The rest of the image is black. After all, the image has only one channel. Specifically for our models, the grayscale images translate to a single channel network's input, hence every model, described before, must change the input to its first convolution layer to accept one channel. Furthermore, since MNIST classes are 10 and not 1000 like Image Net, the last dense layer needs to output 10 features. Fig. 3 displays a sample of MNIST images.

CIFAR10: Contains the same number of images on both training, testing, validation(our choice) as MNIST. The classes are also 10, which means that

our last layer should output 10 features, as before. The differences with MNIST, is that images are 32x32 and colourful(RGB spectrum). This means that our fist convolution layer can remain the same, accepting a three channel input. Fig. 4 displays a sample of CIFAR10 images(distorted due to interpolation).

5.3 Testing Results

Since the purpose of this paper is not to explicitly find networks that rival or outperform the baseline models[13, 6, 15], but rather explore alternative network designs and training methodologies, the results, adduced below, focus on providing useful insights to the reader.

Loss on the MNIST dataset: Due to the very low dataset's complexity, models' final loss value is expected to be close to zero. Fig. 5 verifies our assumption. In fact, regardless of the model's implementation and training methodology, loss values approximate zero, which makes MNIST, a dataset unsuitable for acquiring representative results to compare our models. Nevertheless, for the purpose of this paper, we will try to analyze these results. We observe that the baseline GoogLeNet model outperforms the other two models. Loss is not strictly correlated(inversely) to performance or accuracy but there certainly a connection between them. Usually low loss value means better accuracy, which is the case here(Fig. 7). Despite its better performance, the former is more unstable and converges slower than VGG and ResNet. The plots of the proposed models(right column), only validate our belief that loss, which measures the difference between raw prediction and the class, is not exactly inversely related to accuracy, since vanilla GoogLeNet's loss is lower than that of its counterpart, while the latter perform slightly better than the former. Of course, all values are so close to zero that these findings may not be entirely accurate, which only makes further evaluation necessary.

Loss on the CIFAR10 dataset: In contrast with MNIST, CIFAR10 can serve as a testbed for our findings. Fig. 6 displays all models' loss plots. We notice that baseline models' loss values is increasing for almost every epoch, which may be a direct result of the fixed learning rate that was set but is unlikely due to the fact that loss is increasing from a very early stage (<10 epochs). On the contrary, the loss for all of the improved models is deminishing, which also justifies their increased performance and accuracy. We cannot exactly pinpoint, which architectural alteration or training strategy is the reason behind this behavior, since this would require to retrain the models for every change individually as well as in a group of two or more techniques. Naturally, this task is infeasible with our current hardware resources even on a dataset of resized images.

Accuracy on the MNIST dataset: Similarly with the loss on this dataset, accuracy values are not examplary for evaluating our models. We notice, however, that all proposed models have higher accuracy than all vanilla ones. In fact, their values are so similar that we cannot say, which model is better. Based entirely on validation and testing sets performance, we suggest that our GoogLeNet based model is the best performing model, in accordance with our findings before. Fig. 7 provides a visual illustration of the network's accuracy on the MNIST dataset.

Accuracy on the CIFAR10 dataset: Fig. 8 presents the accuracy plots for each model. Their average accuracy approximates 85% on the inference set and 86% on the validation set. Compared with the average values, 75% and 76% respectively, of their vanilla version, we conclude that our models vastly outperform their simpler versions. Especially for the case of VGG, we see a difference of around 10%, which is astonishing considering the fact that the number of parameters is much lower. Finally, GoogleNet baseline implementation's accuracy only improves by +2.5%, which is surprising considering all the additional methods.

Further Evaluation

Table 2 and Table 3 illustrate the per class accuracy for each dataset accordingly. We notice that ResNet was the model that improved the most on both cases, which showcase the significance of decaying learning rate paired with regularization techniques on a residual network. For MNIST, the class with label 6 was the most "confusing" on average Fig. 9. For CIFAR, this target would be surprisingly the "cat" Fig. 10. For extra information, the reader can also take a look at the confusion matrices and the compressed representation of the dataset over 2 PCA components in the accompanying notebooks.

6 Conclusion

In this work, we demonstrated that the representational depth is beneficial for the classification accuracy and described well-known architectures, used extensively nowadays for various computer vision tasks. State-of-the-art models based on their implementations were also briefly analyzed as well as alternative implementations and training methods for the basic models [6, 13, 15] were proposed and evaluated on the MNIST and CIFAR10 datasets.

References

[1] Irwan Bello, William Fedus, Xianzhi Du, Ekin D Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. *arXiv preprint arXiv:2103.07579*, 2021.

[2] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[10] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[11] Rafael Müller, Simon Kornblith, and Geoffrey Hinton. When does label smoothing help?, 2020.

[12] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[17] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017.

[18] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Appendix

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 1: The configurations of VGG, as shown in the [13]. The ReLU activation function and the adaptive average pooling layer are not shown for brevity.

Class	GoogleNet	ResNet	VGG	Improved GoogLeNet	Improved ResNet	Improved VGG
0	99.79	99.38	99.59	99.79	99.89	100.0
1	99.82	99.55	99.64	99.91	100	100.0
2	99.90	99.51	99.41	99.61	99.61	99.70
3	99.80	99.60	99.50	99.90	99.70	99.70
4	99.08	98.98	99.08	99.38	99.49	99.89
5	99.32	98.54	99.10	99.43	99.55	99.10
6	99.37	98.95	99.06	99.37	99.06	99.58
7	99.31	98.54	98.92	99.61	99.70	99.41
8	99.17	98.54	99.17	99.79	99.58	99.58
9	99.40	97.81	98.31	99.60	99.60	99.40

Table 2: Per class accuracy on MNIST with 2 decimal points

Class	GoogLeNet	ResNet	VGG	Improved GoogLeNet	Improved ResNet	Improved VGG
airplane	86.2	70.19	82.5	90.8	85.9	86.8
automobile	93.8	76.0	84.3	95.5	93.4	93.4
bird	76.2	50.0	69.19	81.2	72.8	66.3
cat	69.19	48.19	62.4	69.1	58.69	50.7
deer	81.6	56.10	75.1	82.1	79.10	78.8
dog	77.2	55.60	63.6	77.5	75.1	61.6
frog	88.2	76.5	81.5	88.6	86.7	87.6
horse	82.69	68.7	77.5	91.0	86.8	84.89
ship	89.4	81.2	84.5	91.9	90.10	89.3
truck	87.0	72.8	79.2	90.8	90.10	90.8

Table 3: Per class accuracy on CIFAR10 with 2 decimal points

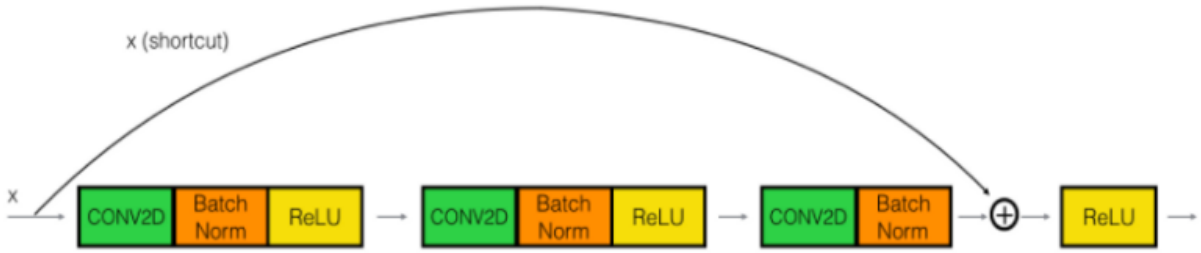


Figure 1: A residual build block with convolution, batch normalization and activation function layers

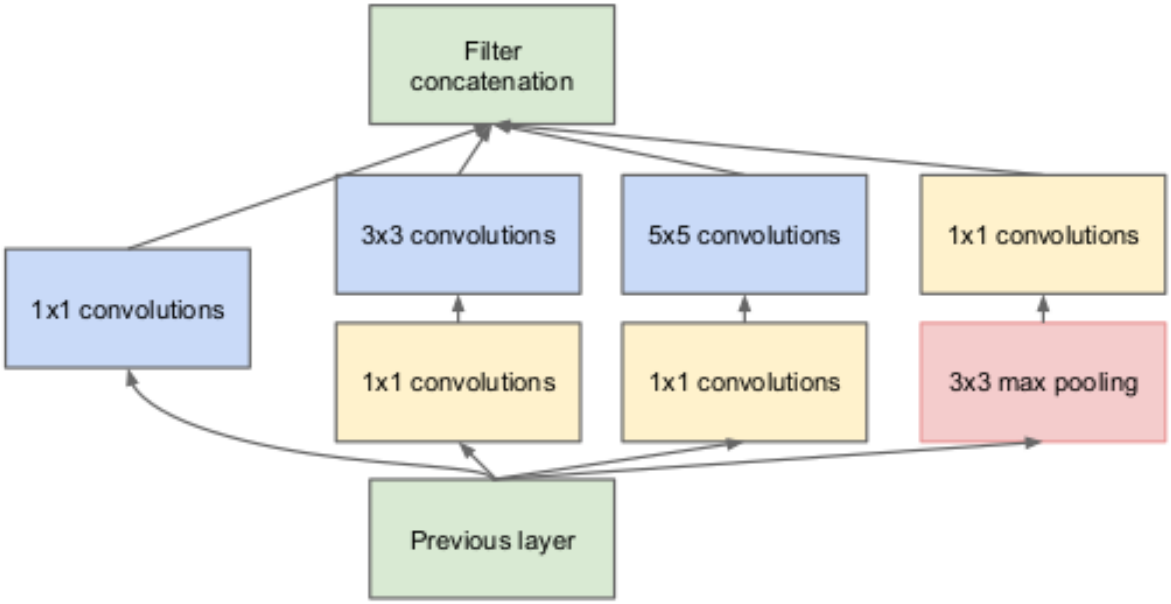
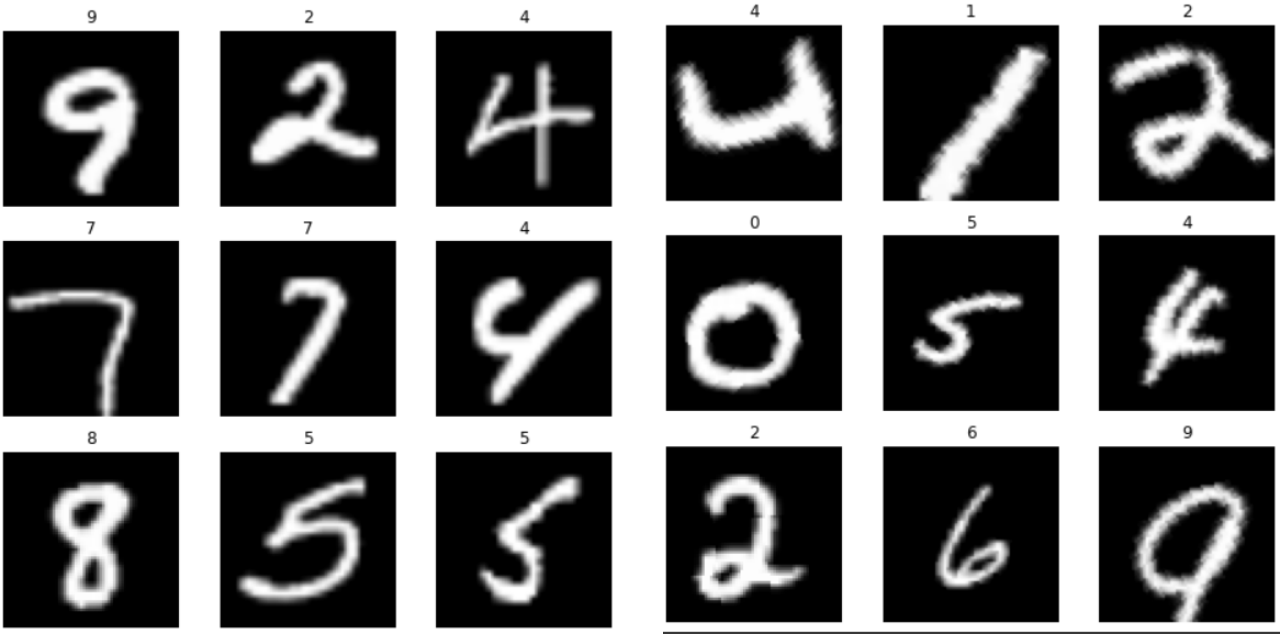


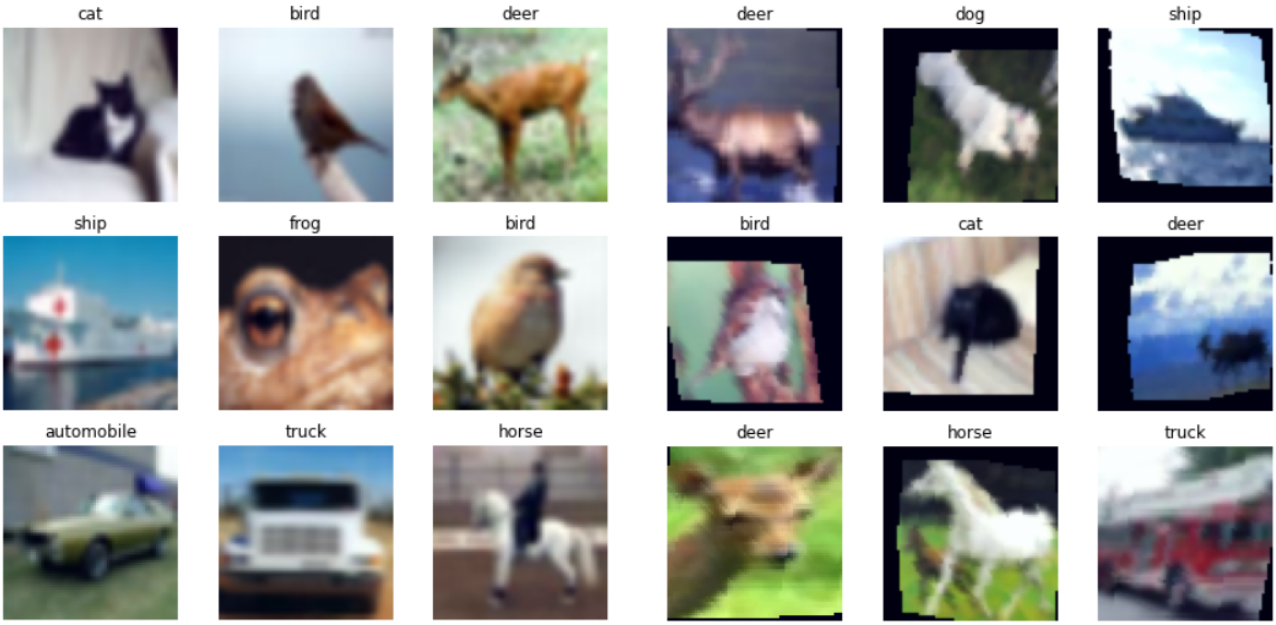
Figure 2: Inception module with dimensions reduction as shown in [15]



(a) Original sample

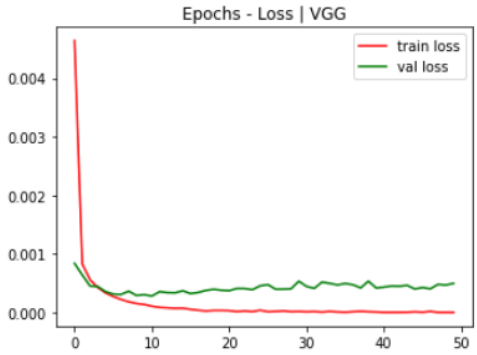
(b) Augmented sample

Figure 3: Images from MNIST

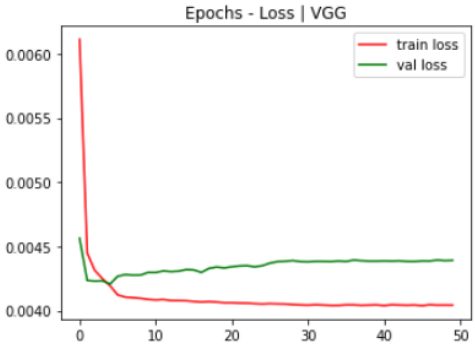


(a) Original sample (b) Augmented sample

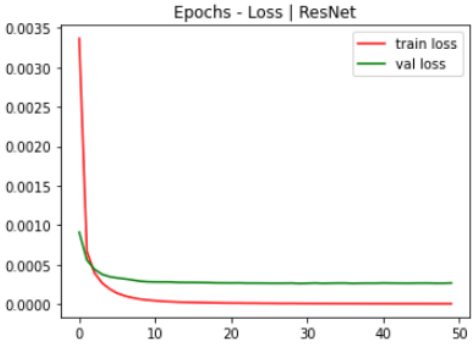
Figure 4: Images from CIFAR10



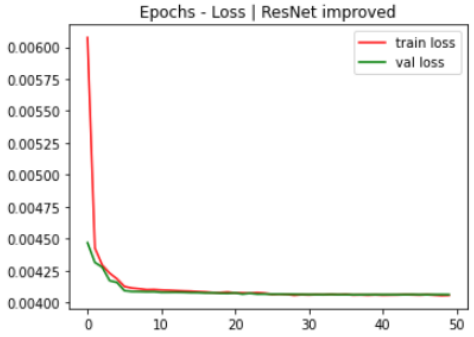
(1) VGG



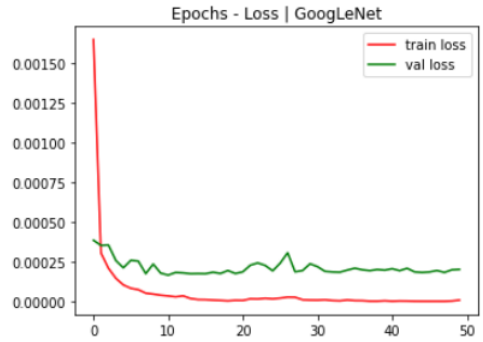
(2) Improved VGG



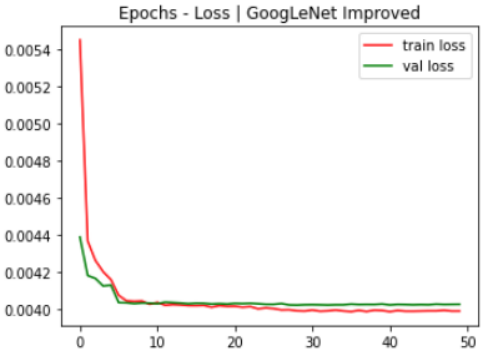
(3) ResNet



(4) Improved ResNet

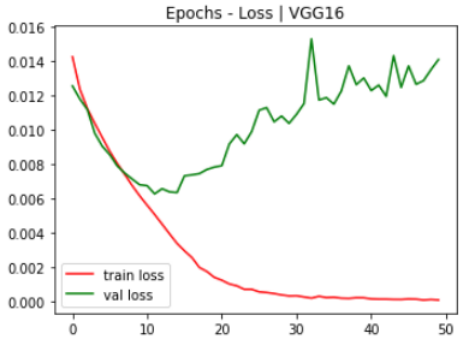


(5) GoogLeNet

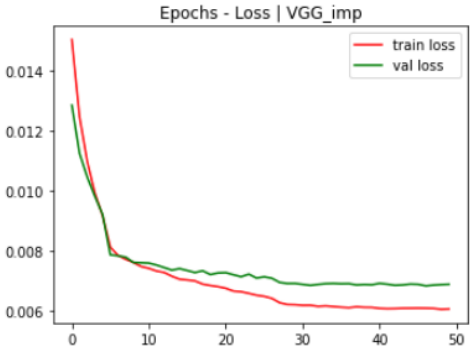


(6) Improved GoogLeNet

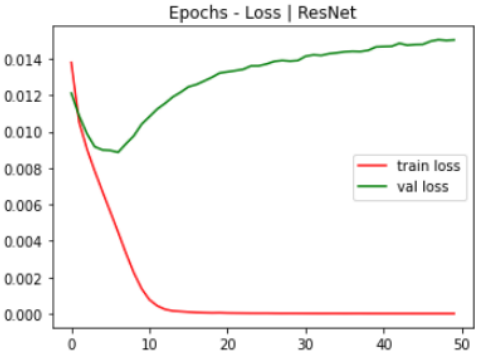
Figure 5: Loss plots on the MNIST dataset



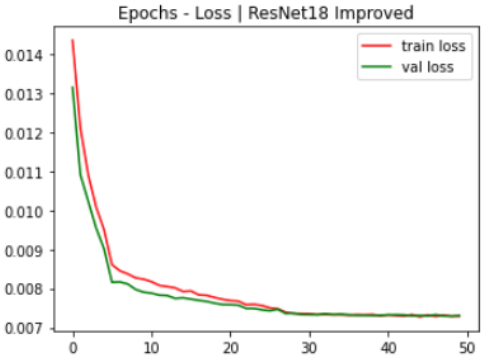
(1) VGG



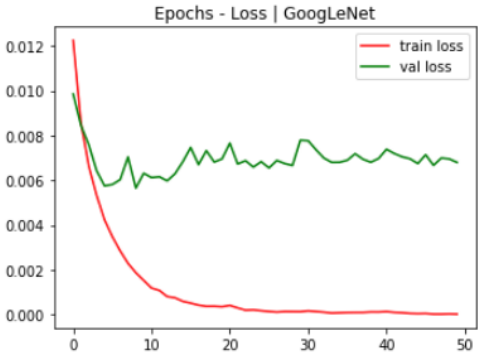
(2) Improved VGG



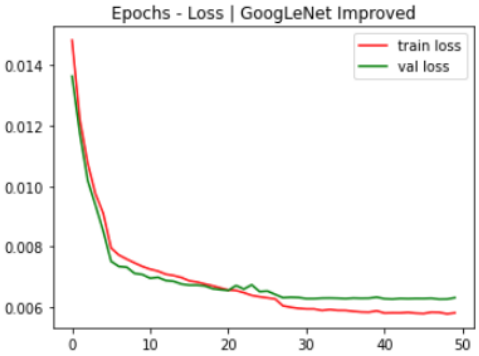
(3) ResNet



(4) Improved ResNet

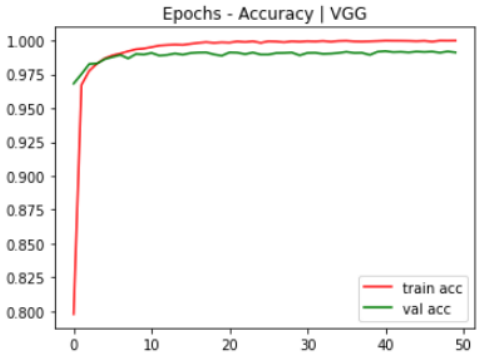


(5) GoogLeNet

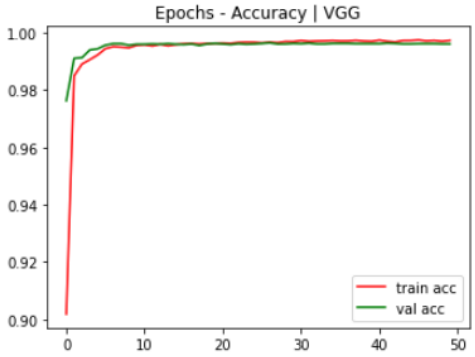


(6) Improved GoogLeNet

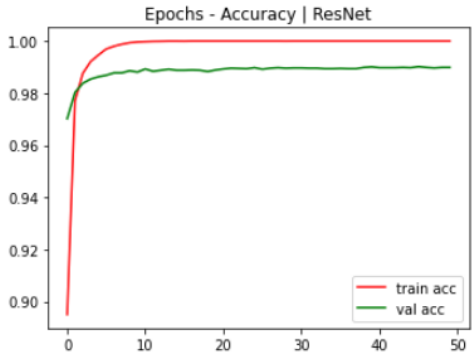
Figure 6: Loss plots on the CIFAR10 dataset



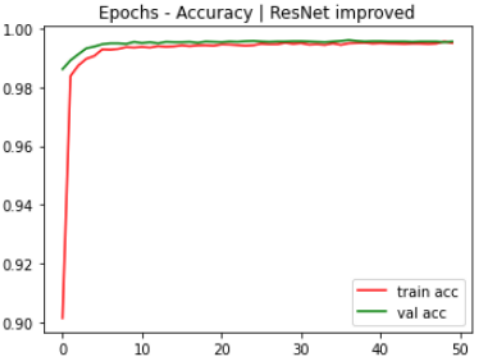
(1) VGG



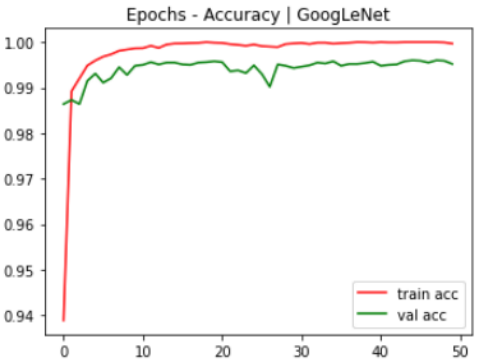
(2) Improved VGG



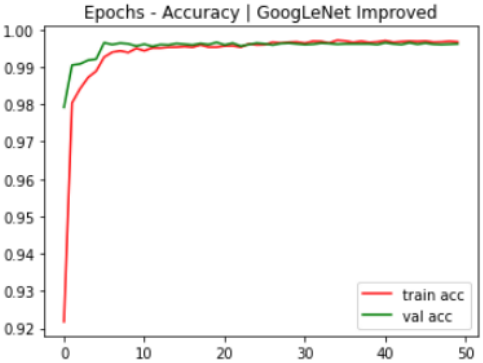
(3) ResNet



(4) Improved ResNet

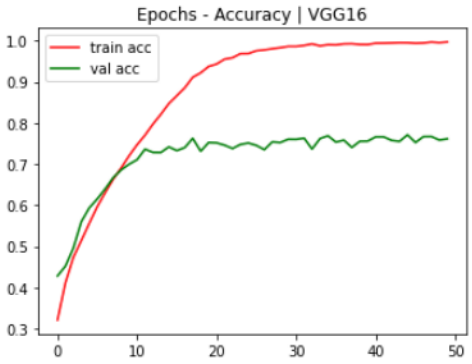


(5) GoogLeNet

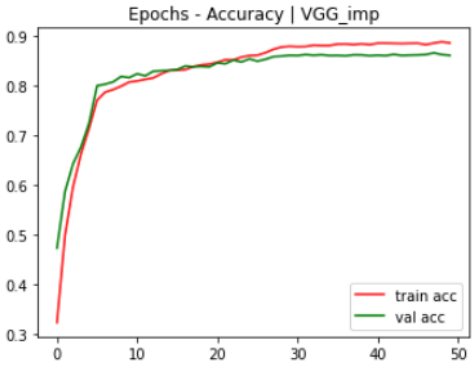


(6) Improved GoogLeNet

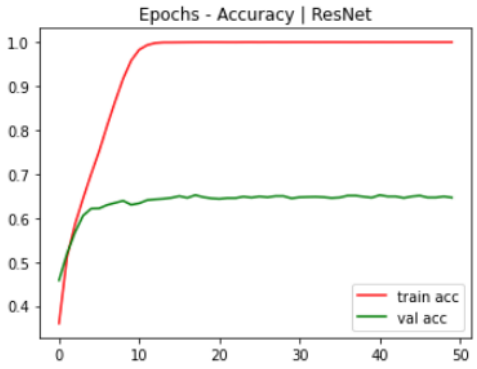
Figure 7: Accuracy plots on the MNIST dataset



(1) VGG

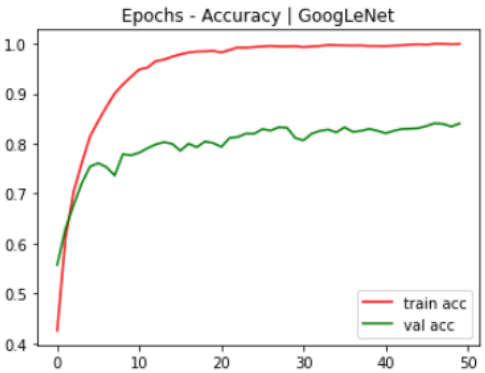


(2) Improved VGG

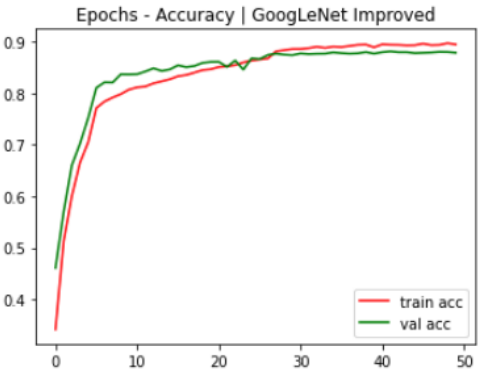


(3) ResNet

(4) Improved ResNet



(5) GoogLeNet



(6) Improved GoogLeNet

Figure 8: Accuracy plots on the CIFAR10 dataset

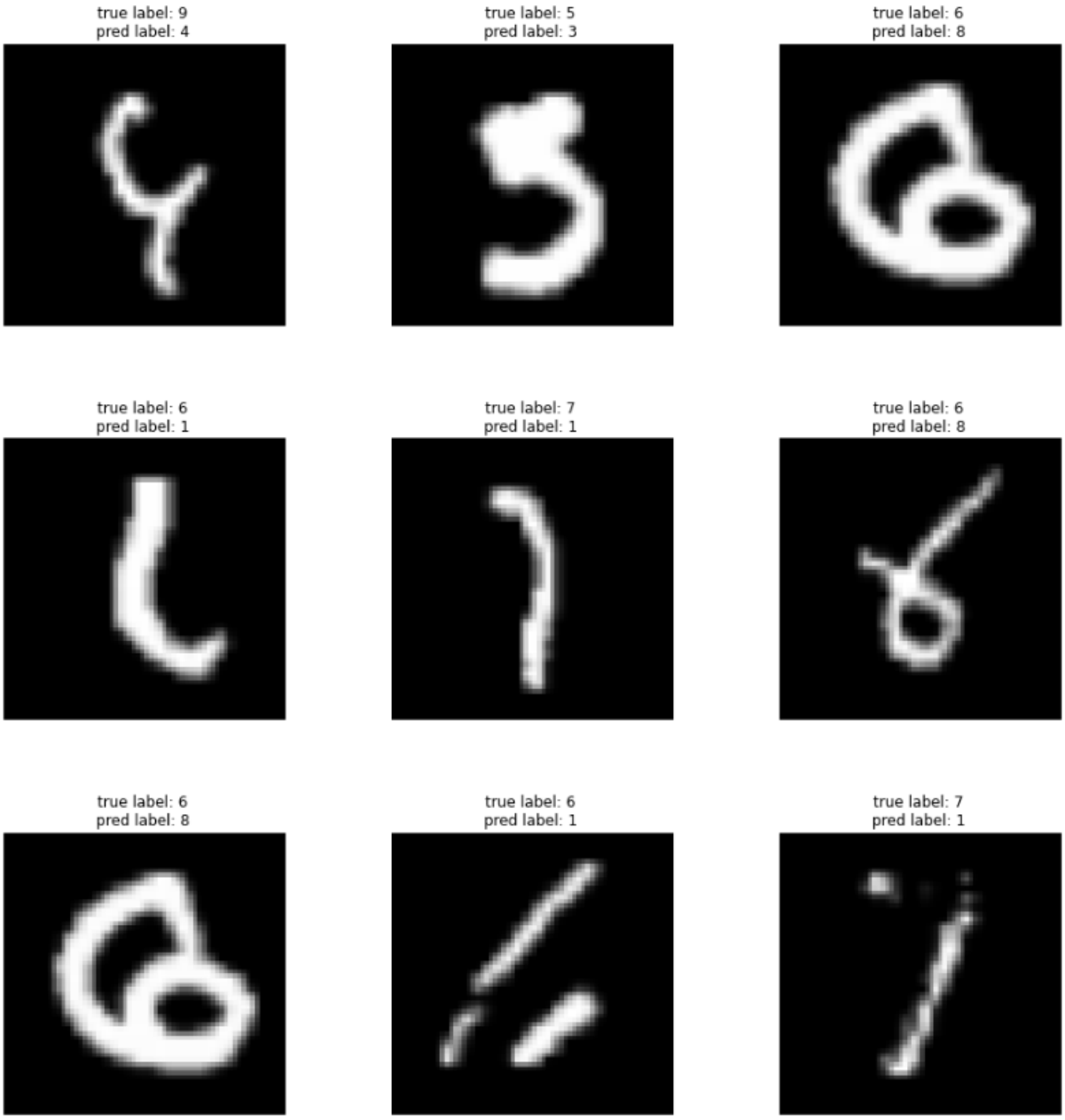


Figure 9: The class label 6 is the target with the lowest accuracy on average among the six models on MNIST

Figure 10: The class label "cat" is the target with the lowest accuracy on average among the six models on CIFAR10

```
Train Epoch 34:
Learning rate: 0.001
Train Epoch: 34 [12800/45000 (28%)]     Loss: 0.025848
Train Epoch: 34 [25600/45000 (57%)]     Loss: 0.044255
Train Epoch: 34 [38400/45000 (85%)]     Loss: 0.026173

On Training set Average loss: 0.0003, Accuracy: 44424/45000 (98.720%)
On Val set Average loss: 0.0117, Accuracy: 3809/5000 (76.180%)

Epoch 34 of 50 with 223.26 s

Train Epoch 35:
Learning rate: 0.001
Train Epoch: 35 [12800/45000 (28%)]     Loss: 0.033564
Train Epoch: 35 [25600/45000 (57%)]     Loss: 0.043707
Train Epoch: 35 [38400/45000 (85%)]     Loss: 0.014222

On Training set Average loss: 0.0002, Accuracy: 44553/45000 (99.007%)
On Val set Average loss: 0.0119, Accuracy: 3843/5000 (76.860%)

New best Validation accuracy (76.860%) achieved so far!

Epoch 35 of 50 with 223.33 s
```

(a) VGG

```
Train Epoch 4:
Learning rate: 0.001
Train Epoch: 4 [12800/45000 (28%)]      Loss: 0.940136
Train Epoch: 4 [25600/45000 (57%)]      Loss: 1.006058
Train Epoch: 4 [38400/45000 (85%)]      Loss: 1.037130

On Training set Average loss: 0.0078, Accuracy: 28999/45000 (64.442%)
On Val set Average loss: 0.0092, Accuracy: 3027/5000 (60.540%)

New best Validation accuracy (60.540%) achieved so far!

Epoch 4 of 50 with 37.65 s

Train Epoch 5:
Learning rate: 0.001
Train Epoch: 5 [12800/45000 (28%)]      Loss: 0.692529
Train Epoch: 5 [25600/45000 (57%)]      Loss: 0.971092
Train Epoch: 5 [38400/45000 (85%)]      Loss: 0.942895

On Training set Average loss: 0.0067, Accuracy: 31478/45000 (69.951%)
On Val set Average loss: 0.0090, Accuracy: 3106/5000 (62.120%)

New best Validation accuracy (62.120%) achieved so far!

Epoch 5 of 50 with 37.58 s
```

(b) ResNet

```
Train Epoch 13:
Learning rate: 0.001
Train Epoch: 13 [12800/45000 (28%)]     Loss: 0.114410
Train Epoch: 13 [25600/45000 (57%)]     Loss: 0.124072
Train Epoch: 13 [38400/45000 (85%)]     Loss: 0.065649

On Training set Average loss: 0.0008, Accuracy: 43424/45000 (96.498%)
On Val set Average loss: 0.0060, Accuracy: 3991/5000 (79.820%)

New best Validation accuracy (79.820%) achieved so far!

Epoch 13 of 50 with 32.42 s

Train Epoch 14:
Learning rate: 0.001
Train Epoch: 14 [12800/45000 (28%)]     Loss: 0.116522
Train Epoch: 14 [25600/45000 (57%)]     Loss: 0.080548
Train Epoch: 14 [38400/45000 (85%)]     Loss: 0.109496

On Training set Average loss: 0.0007, Accuracy: 43556/45000 (96.791%)
On Val set Average loss: 0.0063, Accuracy: 4014/5000 (80.280%)

New best Validation accuracy (80.280%) achieved so far!
```

(c) GoogleNet

Figure 11: Training logs of baseline models on CIFAR10

```
Train Epoch 49:
Learning rate: 0.001
Train Epoch: 49 [12800/50000 (26%)]     Loss: 0.000051
Train Epoch: 49 [25600/50000 (51%)]     Loss: 0.000007
Train Epoch: 49 [38400/50000 (77%)]     Loss: 0.000042

On Training set Average loss: 0.0000, Accuracy: 49996/50000 (99.992%)
On Val set Average loss: 0.0005, Accuracy: 9920/10000 (99.200%)

Epoch 49 of 50 with 64.43 s

Train Epoch 50:
Learning rate: 0.001
Train Epoch: 50 [12800/50000 (26%)]     Loss: 0.000034
Train Epoch: 50 [25600/50000 (51%)]     Loss: 0.000035
Train Epoch: 50 [38400/50000 (77%)]     Loss: 0.000026

On Training set Average loss: 0.0000, Accuracy: 49999/50000 (99.998%)
On Val set Average loss: 0.0005, Accuracy: 9913/10000 (99.130%)

Epoch 50 of 50 with 64.61 s

Model fit in : 3222.86s
```

(a) VGG

```
Epoch 36 of 50 with 35.87 s

Train Epoch 37:
Learning rate: 0.001
Train Epoch: 37 [12800/50000 (26%)]     Loss: 0.000359
Train Epoch: 37 [25600/50000 (51%)]     Loss: 0.000673
Train Epoch: 37 [38400/50000 (77%)]     Loss: 0.000460

On Training set Average loss: 0.0000, Accuracy: 50000/50000 (100.000%)
On Val set Average loss: 0.0003, Accuracy: 9894/10000 (98.940%)

Epoch 37 of 50 with 36.17 s

Train Epoch 38:
Learning rate: 0.001
Train Epoch: 38 [12800/50000 (26%)]     Loss: 0.000352
Train Epoch: 38 [25600/50000 (51%)]     Loss: 0.000321
Train Epoch: 38 [38400/50000 (77%)]     Loss: 0.000637

On Training set Average loss: 0.0000, Accuracy: 50000/50000 (100.000%)
On Val set Average loss: 0.0003, Accuracy: 9894/10000 (98.940%)

Epoch 38 of 50 with 35.93 s
```

(b) ResNet

```
Train Epoch 24:
Learning rate: 0.001
Train Epoch: 24 [12800/50000 (26%)]     Loss: 0.000677
Train Epoch: 24 [25600/50000 (51%)]     Loss: 0.000179
Train Epoch: 24 [38400/50000 (77%)]     Loss: 0.000244

On Training set Average loss: 0.0000, Accuracy: 49960/50000 (99.920%)
On Val set Average loss: 0.0002, Accuracy: 9932/10000 (99.320%)

Epoch 24 of 50 with 30.61 s

Train Epoch 25:
Learning rate: 0.001
Train Epoch: 25 [12800/50000 (26%)]     Loss: 0.000857
Train Epoch: 25 [25600/50000 (51%)]     Loss: 0.001163
Train Epoch: 25 [38400/50000 (77%)]     Loss: 0.005709

On Training set Average loss: 0.0000, Accuracy: 49974/50000 (99.948%)
On Val set Average loss: 0.0002, Accuracy: 9949/10000 (99.490%)

Epoch 25 of 50 with 30.86 s
```

(c) GoogleNet

Figure 12: Training logs of baseline models on MNIST

```
Train Epoch 44:
Current learning rate: [1e-05]
Train Epoch: 44 [12800/45000 (28%)]     Loss: 0.757117
Train Epoch: 44 [25600/45000 (57%)]     Loss: 0.759442
Train Epoch: 44 [38400/45000 (85%)]     Loss: 0.880652

On Training set Average loss: 0.0061, Accuracy: 39859/45000 (88.576%)
On Val set Average loss: 0.0069, Accuracy: 4309/5000 (86.180%)

Epoch 44 of 50 with 118.03 s

Train Epoch 45:
Current learning rate: [1e-05]
Train Epoch: 45 [12800/45000 (28%)]     Loss: 0.735547
Train Epoch: 45 [25600/45000 (57%)]     Loss: 0.822281
Train Epoch: 45 [38400/45000 (85%)]     Loss: 0.713938

On Training set Average loss: 0.0061, Accuracy: 39877/45000 (88.616%)
On Val set Average loss: 0.0069, Accuracy: 4312/5000 (86.240%)

Epoch 45 of 50 with 117.72 s
```

(a) VGG

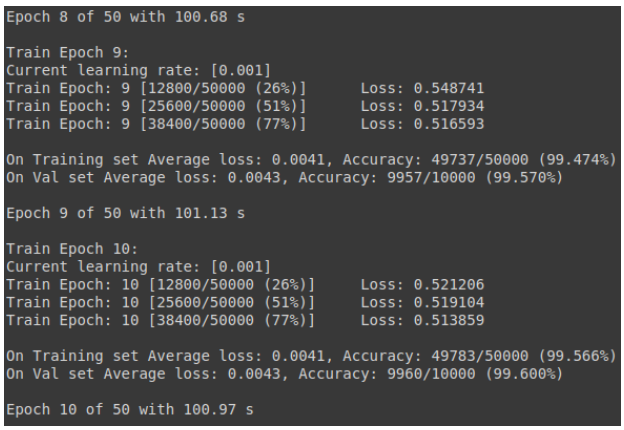
```
Train Epoch 49:
Current learning rate: [1.0000000000000002e-06]
Train Epoch: 49 [12800/45000 (28%)]     Loss: 0.889027
Train Epoch: 49 [25600/45000 (57%)]     Loss: 0.932063
Train Epoch: 49 [38400/45000 (85%)]     Loss: 0.940668

On Training set Average loss: 0.0073, Accuracy: 36585/45000 (81.300%)
On Val set Average loss: 0.0073, Accuracy: 4093/5000 (81.860%)

Epoch 49 of 50 with 73.49 s

Train Epoch 50:
Current learning rate: [1.0000000000000002e-06]
Train Epoch: 50 [12800/45000 (28%)]     Loss: 0.960140
Train Epoch: 50 [25600/45000 (57%)]     Loss: 0.838578
Train Epoch: 50 [38400/45000 (85%)]     Loss: 0.916030

On Training set Average loss: 0.0073, Accuracy: 36672/45000 (81.493%)
On Val set Average loss: 0.0073, Accuracy: 4099/5000 (81.980%)

Epoch 50 of 50 with 73.39 s

Model fit in : 3738.48s
```

(b) ResNet

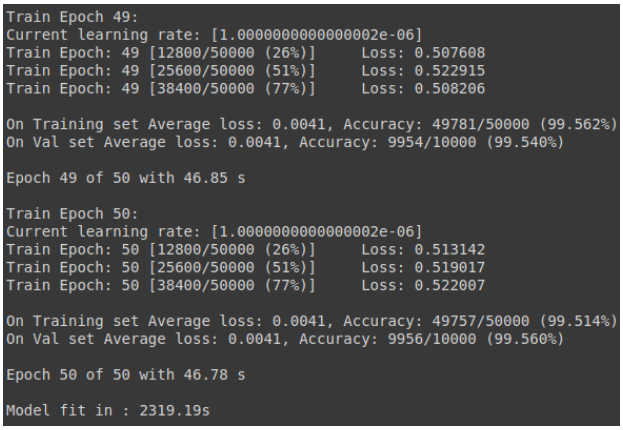
```
Train Epoch 49:
Current learning rate: [1.0000000000000002e-06]
Train Epoch: 49 [12800/45000 (28%)]     Loss: 0.733610
Train Epoch: 49 [25600/45000 (57%)]     Loss: 0.693295
Train Epoch: 49 [38400/45000 (85%)]     Loss: 0.768020

On Training set Average loss: 0.0058, Accuracy: 40360/45000 (89.689%)
On Val set Average loss: 0.0063, Accuracy: 4399/5000 (87.980%)

Epoch 49 of 50 with 134.87 s

Train Epoch 50:
Current learning rate: [1.0000000000000002e-06]
Train Epoch: 50 [12800/45000 (28%)]     Loss: 0.668559
Train Epoch: 50 [25600/45000 (57%)]     Loss: 0.690771
Train Epoch: 50 [38400/45000 (85%)]     Loss: 0.682710

On Training set Average loss: 0.0058, Accuracy: 40246/45000 (89.436%)
On Val set Average loss: 0.0063, Accuracy: 4392/5000 (87.840%)

Epoch 50 of 50 with 134.76 s

Model fit in : 6661.58s
```

(c) GoogleNet

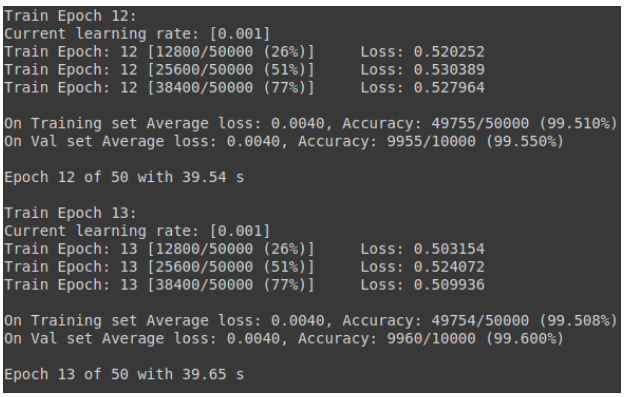
Figure 13: Training logs of improved models on CIFAR10



(a) VGG



(b) ResNet



(c) GoogleNet

Figure 14: Training logs of improved models on MNIST