CW3 - Deeper Networks for Image Classification

ECS795P Deep Learning and Computer Vision
Divya Hitesh Chhipani
200369536

1. Introduction

Image Classification is a complex task which involves segregation of images into categories by a model. This task relies on many factors and a large dataset is required to build a robust model to efficiently perform the classification. Deep Learning models are often employed to solve the image classification problem as they are capable of learning several features of different complexities from the input images without any manual feature-engineering. Deep Networks such as variants of convolution neural networks are widely researched upon for this purpose as CNNs are best suited for working with visual data. The aim of this paper is to study different architectures on Deep CNNs and perform image classification on MNIST and CIFAR10 datasets using these models.

2. Critical Analysis

Yann LeCun first introduced application of Convolution Networks to read zip codes, digits, with LeNet[1] which consisted of 5 layers - three sets of convolution layers with a combination of average pooling following which there are two fully connected layers. Softmax activation was used for classification of images into respective class. AlexNet[2] introduced the application of Convolutional Networks in the field of Computer Vision and had a very similar architecture to LeNet, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other instead of pooling layers after every convolution layer. An improvement on AlexNet was ZFNet[3] which was developed by making changes to the architecture hyperparameters, like reducing the stride and filter size of the first layer, and expanding the size of the convolutional layers in the middle of the network. Karen Simonyan and Andrew Zisserman introduced VGGNet[4], the runner-up in ILSVRC 2014. This network emphasized the importance of depth of the network for a good performance. Their final best network was VGG16 which has convolution/fully-connected layers and features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling throughout the network. One of the issues of the VGGNet is the large number of parameters(140M) most of them being in the FC layer, and needs a lot more memory and it is very expensive to evaluate. The winning architecture of ILSVRC 2014 was GoogLeNet[5] which was launched by Google in 2014. GoogLeNet introduced the concept of Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M) even though the depth and width of the network was increased. GoogLeNet overcame the limitations of VGGNet by using the Average Pooling layer instead of Fully Connected layers at the top of the ConvNet, thereby reducing the number of parameters without affecting the performance of the network. Many improved variants of GoogLeNet have been released so far, recent one being Inception-v4. Training of deep models is time-consuming and such models are also prone to overfitting due to lack of data. ResNet (Residual Network)[6] architecture was introduced in 2015 to improve the training process while making the networks deeper. As the networks get deeper, the error rates get higher which is known as the degradation problem. ResNet aimed to mitigate this by implementing a technique called ‘‘Residual mapping’’ so that the layers explicitly fit these mappings. The architecture features special skip connections to connect output from previous layer to the layer ahead, making the network deeper with these identity mappings while establishing that the error rate is not greater than the architecture’s shallower versions. It also employs a heavy use of batch normalization to mitigate vanishing gradients problem. Also, fully connected layers are not used at the end of the network. ResNets are currently considered the state-of-the-art Convolutional Neural Network models and are the default choice in practice. A recent improvement on ResNet is WResNet(Wide Residual Networks)[7] which mentions that instead of making the networks deeper, they can be made wider for achieving similar performance as that of their deeper counterparts. The architecture consists of a bunch of ResNet blocks stacked together, where each ResNet block follows the BatchNormalization-ReLU-Conv structure.

3. Method/Model Description

3.1. Model Architectures

3.1.1 VGGNet

VGGNet has many variants differing only in depth; from 11 weight layers (8 convolutional and 3 fully-connected layers), to 19 weight layers (16 convolutional and 3 fully-connected layers). The

number of channels of convolutional layers is 64 in the first layer and this number keeps increasing by a factor of 2 after each max-pooling layer, until it reaches 512. The architecture takes input of fixed image size, (244 × 244). In a pre-processing step, the mean RGB value is subtracted from each pixel in an image. The images are passed through a stack of convolutional layers with receptive-field filters of size (3×3). The filter size is set to (1 × 1) in some architecture variants, which can be identified as a linear transformation of the input channels (followed by non-linearity). The stride for the convolution operation is fixed to 1. Spatial pooling is carried out by five max-pooling layers over a (2 × 2) pixel window, with stride size set to 2, which follow several convolutional layers. The configuration for fully-connected layers is always the same; the first two layers have 4096 channels each, the third layer performs classification such that number of channels correspond to the number of classes in the classification task and the final layer is the Softmax layer. ReLu activation function is used after each of the hidden layers. The training is carried out by optimizing the multinomial logistic regression objective using mini-batch gradient descent based on backpropagation with momentum (0.9). The learning rate used initially is 0.001. Finally, the dropout ratio of 0.5 can be added for the first two fully-connected layers for training.

Illustration of the model's architecture can be found in the appendix.

3.1.2 GoogLeNet

The GoogleNet Architecture is 22 layers deep (27 layers including pooling layers). The input layer of the GoogLeNet architecture takes in an image of the dimension 224 x 224 in the RGB color space with zero mean. The first conv layer in figure 2 uses a filter size of 7x7 and stride of 2, which is relatively large compared to other patch sizes within the network. The input image size is reduced by a factor of four at the second convolution layer which has a depth of two and leverages the 1x1 conv block. This is followed by another max pooling layer before the first inception module which reduces the original image by a factor of 8 by this stage. There are nine inception modules in this network with a max-pooling layer after first two inception modules and the other one after next four inception modules; this is followed by the remaining inception modules. All the max-pooling layers in the network have a fixed filter size of 3x3 and stride of 2. Each inception module performs 1×1, 3×3, 5×5 convolution and 3×3 max pooling on the input parallelly. The output from all these four units is concatenated to get the final output. The output from the final inception module is then passed to the average pooling layer which takes a mean of all the feature maps received with a filter of size 7x7 and stride 1. This is followed by a dropout layer with dropout ratio of 0.4 following which is a linear layer consisting of channels corresponding to the number of classes and a final Softmax activation layer. During the time of training, auxiliary classifiers are added in between inception modules such that first one is placed after the third inception module and the second one is placed after the sixth inception module. These classifiers consist of a 5×5 average pooling layer with a stride of 3, a 1×1 convolution with 128 filters, two fully connected layers of 1024 outputs and number of outputs corresponding to number of classes, and a softmax classification layer; just like the final set of layers of the architecture. These classifiers help mitigate the vanishing gradients problem. The loss from these classifiers is added and multiplied by a weight of 0.3 before adding it to the loss from the last layer all of which is Cross Entropy Loss. For gradient calculation, asynchronous stochastic gradient descent is used with a momentum of 0.9. Learning rate of 0.01 is set initially and can be experimented with.

Illustration of the model's architecture can be found in the appendix.

3.1.3 ResNet

Similar to VGGNet, ResNet has variants differing in the number of layers (18, 34, 50, 101, and so on). Describing the architecture for ResNet18, the network takes in input of image size of 224x224. First layer is a convolution layer with 7x7 filter size and stride 2, followed by a 3x3 filter with stride 2 max-pooling layer. This is now followed by 4 consecutive residual blocks with skip connections from the input to the output of each block such that they are added together to get the input for the next set of layers. Each residual block consists of two 3×3 convolutional layers with the same number of output channels (64,128,256, 512 in respective blocks). Each convolutional layer is followed by a batch-normalization layer and a ReLU activation function such that the skip connection is added right before the ReLU activation. Output from the last residual block is passed to an average pooling layer and the resulting feature map is passed to the fully connected layers followed by softmax function to get the final output.

Illustration of the model's architecture can be found in the appendix.

3.2. Data Pre-processing

Data Augmentation

Computer vision tasks with Deep Learning models require large amount of data especially with the models with 5+ layers. Data augmentations helps in increasing the population of the image set while saving the model from overfitting and making it more

robust. Data Augmentation techniques involve transforming original images from the training set like random rotation, color jittering, and flipping to name a few and images with these transformations are added back to the training set. Data normalization is done to fix the input's mean and variance between 0 and 1.

Batch Normalization
Batch normalization is generally implemented to speed up training. Vanilla models described before have a batch-normalization layer tugged between convolution layer and ReLU activation layer. However, there is a debate as to whether the batch-normalization should be applied before or after the activation as experiments have shown that using batch-normalization after ReLU activation results in a better performance in practice.

3.3. Parameter Tuning

Learning Rate
All the model variants were initially trained with a learning rate of 0.001 and then trained with the value of 0.01 to reduce the training time.

Optimizers
VGGNet and ResNet are trained with different optimizers like RMSprop, Stochastic Gradient Descent, adding momentum and using ADAM optimizer. ADAM is usually seen as an improvement over RMSprop to improve the model performance.

Epochs and Batch Size
Increasing the batch size greatly improves the accuracy of the model at the cost of computation cost. Current resources are not compatible at handling batch sizes of more than 64 due to large number of weights. Reduction of batch size is done to train at the maximum batch size capacity for each model. We fix the epochs to 10 due to computation limitation.

3.4. Evaluation Metrics
During the training step, we get the loss from logits. We also get probability of predictions and total number of records in this which is used to create batch dictionary and passed it to the next function which gets executed after the epoch ends. After the training, we use these values to calculate the accuracy and then add those values into the tensorboard memory. We obtain final set of per-class and average accuracy, precision, recall and f1score during testing.

3.5. Training Methodology

For training, we first consider the vanilla models of the architectures described in the previous sections and run the models according to the settings mentioned in their introductory paper. Models such as VGG and ResNet were introduced with different variants in their original paper but, for ResNet, instead of using a large 7x7 convolution in the beginning, we use a stack of three 3x3 convolutional filters. We pick the vanilla version of the model variant which performs the best under original settings and make improvements to these models for training. Improvements such as changing learning rate, trying different optimizers, data augmentation, and different placement of batch-normalization is tried. Training is done for a fixed number of 10 epochs due to limited availability of computing resources however, for future trials, number of epochs such as 50 or 100 can be considered. Parallel processing on TPUs (mostly required for VGGNet) and GPU (for ResNet and GoogLeNet) is employed for training purposes as each model consists of more than 10 million features. Batch sizes are varied to avoid out-of-memory errors. However, we aim to increase the batch size and number of epochs to train the final set of best models achieved from initial experimentation.

4. Experiments

In this section, we train classifier models which architectures discussed above on the MNIST and CIFAR10 datasets.

4.1 Datasets

The models in this paper are trained on MNISTand CIFAR10 datasets.

MNIST
The MNIST (Modified National Institute of Standards and Technology)[8] database of handwritten digits between 0 and 9, has a training set of 60,000 examples and a test set of 10,000 examples. Each image is 28 x 28 pixels. There are 10 classes in total corresponding to the digits [0-9]. All the images are 1-channel grayscale images.

Source:

https://en.wikipedia.org/wiki/MNIST_database

CIFAR10
CIFAR10[9] consists of 60000 images (50,000 in training

set and 10,000 in test set) of size 32x32 which are coloured images (3-channels). There are 10 classes, with 6000 images per class. The class names are airplane, automobile, bird, cat, deer, dog frog, horse,ship, truck.

4.2 Test Results

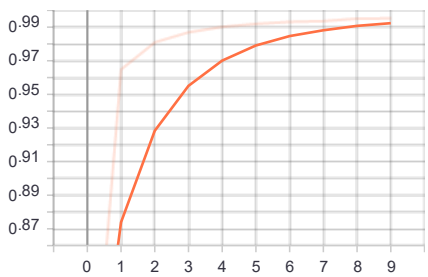
4.2.1 MNIST

VGGNet: We have different variants of VGGNet which were trained and evaluated on MNIST dataset. Results on test set are shown below –

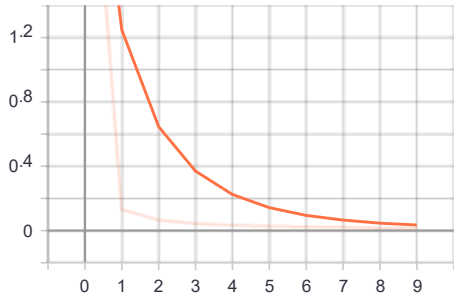
Model	Accuracy
VGG16	99.1
VGG11	99.1
VGG19	83
VGG16 with LR = 0.01	98.9
VGG16 with SGD optimizer	90.2
VGG16 with SGD optimizer and momentum	99.2
VGG16 with ADAM optimizer	98.2
VGG16 with Batch Normalization after ReLU activation	9.8
VGG16 with Data Augmentation	99.3

Training plots for best model (VGG16 with Data Augmentation) -

Training Accuracy vs Epochs Plot



Training Loss vs Epochs Plot

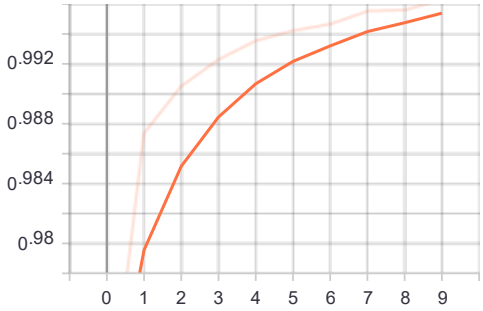


ResNet: We have different variants of ResNet which were trained and evaluated on MNIST dataset. Results on test set are shown below –

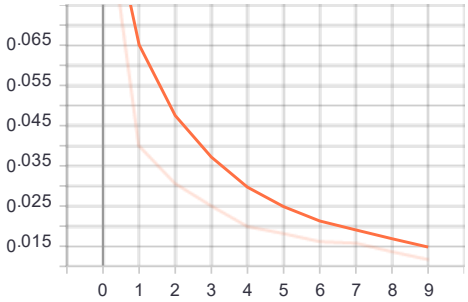
Model	Accuracy
ResNet18	99.3
ResNet101	99.3
ResNet18 with LR = 0.01	99
ResNet18 with SGD optimizer	98.4
ResNet18 with SGD optimizer and momentum	99.3
ResNet18 with ADAM optimizer	99.4
ResNet18 with Data Augmentation	99.3

Training plots for best model (ResNet18 with ADAM optimizer) -

Training Accuracy vs Epochs Plot



Training Loss vs Epochs Plot

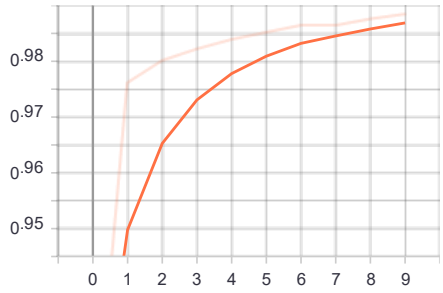


GoogLeNet: We have different variants of GoogLeNet which were trained and evaluated on MNIST dataset. Results on test set are shown below –

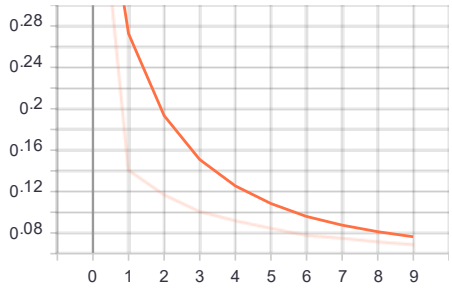
Model	Accuracy
GoogLeNet	11.4
GoogLeNet with Data Augmentation	11.4
GoogLeNet with ADAM optimizer	99
GoogLeNet with ADAM optimizer and Data Augmentation	99.1

Training plots for best model (GoogLeNet with ADAM optimizer and Data Augmentation) -

Training Accuracy vs Epochs Plot



Training Loss vs Epochs Plot



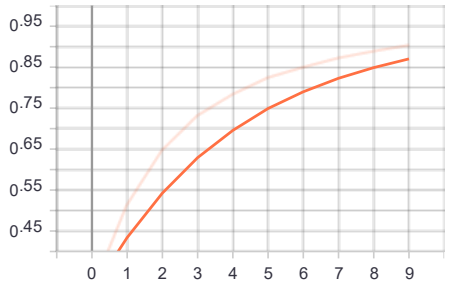
4.2.2 CIFAR10

VGGNet: We have different variants of VGGNet which were trained and evaluated on CIFAR10 dataset. Results on test set are shown below –

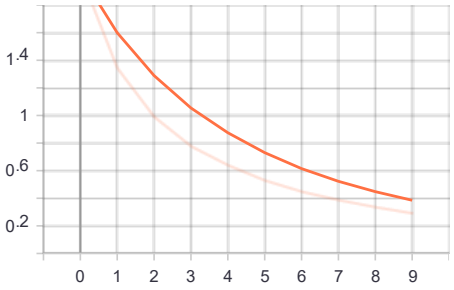
Model	Accuracy
VGG16	78.6
VGG11	73.6
VGG19	78.5
VGG16 with LR = 0.01	79.2
VGG16 with SGD optimizer	65.6
VGG16 with SGD optimizer and momentum	78.6
VGG16 with ADAM optimizer	82.7
VGG16 with Batch Normalization after ReLU activation	80.8
VGG16 with Data Augmentation	72.5
VGG16 with ADAM optimizer, LR 0.01, Batch Normalization after ReLU, epoch 20	84.8

Training plots for best model (VGG16 with ADAM optimizer, LR 0.01, Batch Normalization after ReLU, epoch 20) -

Training Accuracy vs Epochs Plot



Training Loss vs Epochs Plot

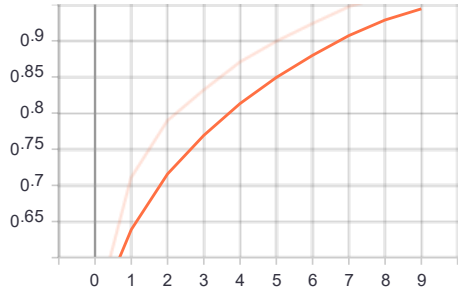


ResNet: We have different variants of ResNet which were trained and evaluated on CIFAR10 dataset. Results on test set are shown below –

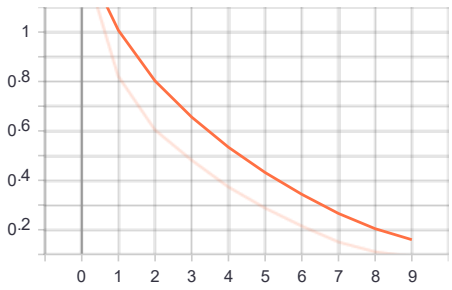
Model	Accuracy
ResNet18	78.9
ResNet101	77.8
ResNet with LR = 0.01	80.4
ResNet18 with SGD optimizer	64.6
ResNet18 with SGD optimizer and momentum	78.4
ResNet18 with ADAM optimizer	84.3
ResNet18 with Data Augmentation	81.2
ResNet18 with ADAM optimizer, LR 0.01, epoch 20	83.8

Training plots for best model (ResNet18 with ADAM optimizer, LR 0.01, epoch 20) –

Training Accuracy vs Epochs Plot



Training Loss vs Epochs Plot



GoogLeNet: We have different variants of GoogLeNet which were trained and evaluated on CIFAR10 dataset. Results on test set are shown below –

Model	Accuracy
GoogLeNet	10
GoogLeNet with Data Augmentation and ADAM optimizer	10

The model was not training above 2 epochs due to memory limitation and no relevant results could be observed as the model didn't learn anything both times.

4.3 Further evaluation

We use sk-learn's classification report to check the metrics for each class. For CIFAR10, per-class f1score for all the classes except cat, dog, and deer is above 85% and precision being generally higher than recall, while in MNIST, per-class metrics are almost equally good with a value of 98% and above.

Per-class metrics from the best models are shown in the Appendix.

5. Image Segmentation

As an additional task over image classification, we performed Image Segmentation task which aims to trace the boundary of the object in an image. This is done by classifying each pixel of the image to a particular class. We have implemented a FCN (fully-convolution network) for this task which uses VGG16 pre-trained on ImageNet and changes the final layer to convolution layers of 1x1 convolution. This produces a low-resolution heatmap of class presence which is later upsampled. High resolution feature maps from lower layers of VGG16 are used during upsampling. Skip connections are also used to extract abstract class features. Image Segmentation is performed on VOC2012 Semantic Segmentation dataset which consists of annotated images from Flickr. Evaluation metrics used were IOU score achieving 42% and DICE score which was 57% on the test set.

Source: https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html#sec_fcn

6. Conclusion

From our experimentation, we can see that deep model perform generally well on MNIST dataset over CIFAR10 for small number off epochs probably owing to the fact that MNIST images are single-channel and are rather simple than images in CIFAR10. Practically, ResNet is the simplest model to train given less computation required due to the presence of skip connections and results could be better if we train for more epochs in future. ADAM optimizer seems as easy choice to improve the model performance. Vanilla versions of GoogLeNet do not seem to learn given less epochs and physical limitation of resources do not help us in improving it except for using ADAM optimizer for MNIST dataset which gives the performance a massive boost. No cases of overfitting have been found so far for VGGNet and ResNet as experimentation was limited to 10 epochs and so, not much advantage of Batch Normalization and Data Augmentation was observed. Also, depth of the models did not seem to make much difference for performance on both the datasets. To conclude, ResNet seems to be the state-of-the-art model for image classification as also learned from the literature review.

7. References

[1] LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. 1995.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1.

[3] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in European conference on computer vision. Springer, 2014, pp. 818–833.

[4] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs] 2014.

[5] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2014. Going deeper with convolutions. CoRR abs/1409.4842.

[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, 2016.

[7] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. arXiv preprint arXiv:1605.07146, 2016.

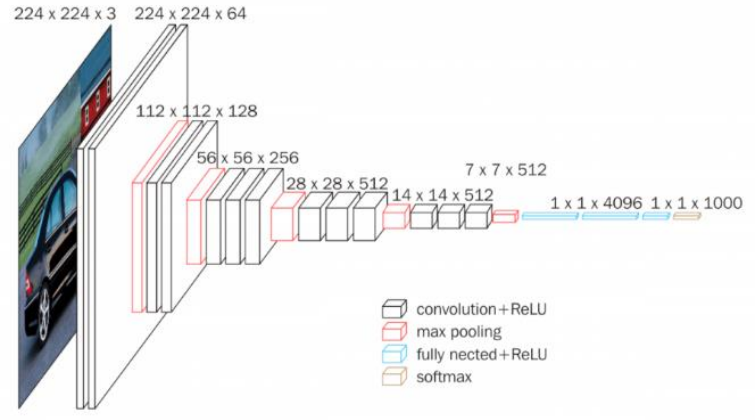
[8] Brownlee, J. (2020, August 24). How to Develop a CNN for MNIST Handwritten Digit Classification.

[9] Https://Www.Cs.Toronto.Edu/~kriz/Cifar.Html.

Appendix

Model Architectures

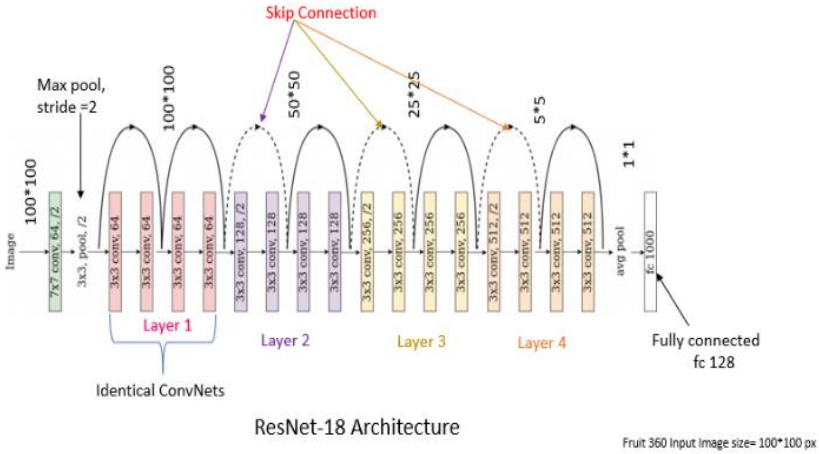
- **VGG16 Architecture**



Source: https://paperswithcode.com/method/vgg

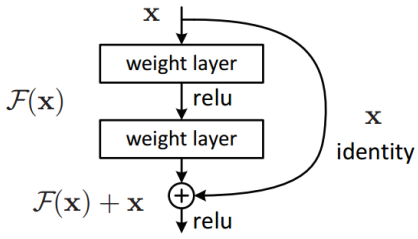
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	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					

- **ResNet18 Architecture**



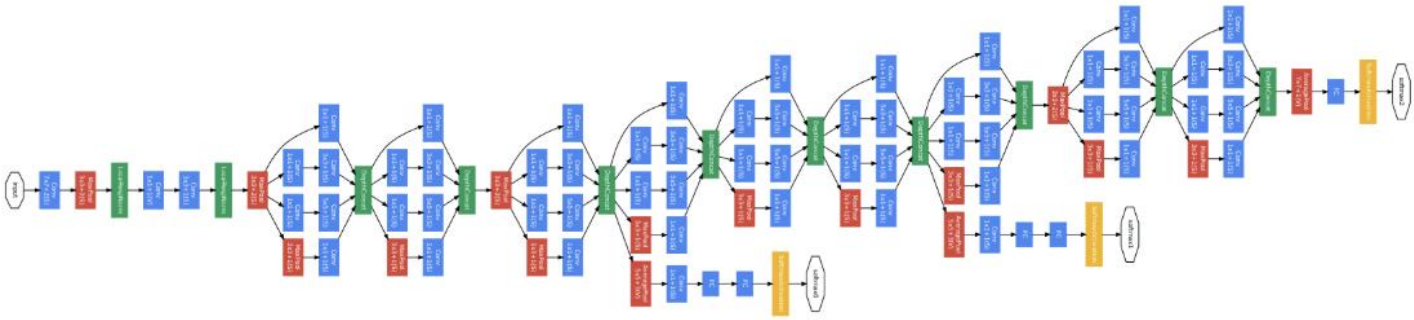
Source:https://www.pluralsight.com/guides/introduction-to-resnet

Single Residual Block



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

- **GoogLeNet Architecture**

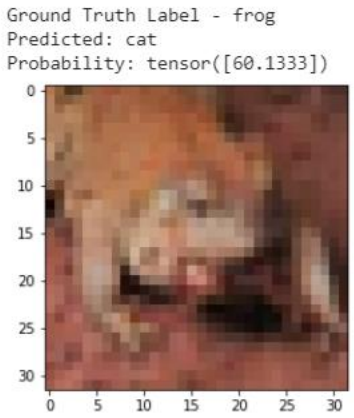


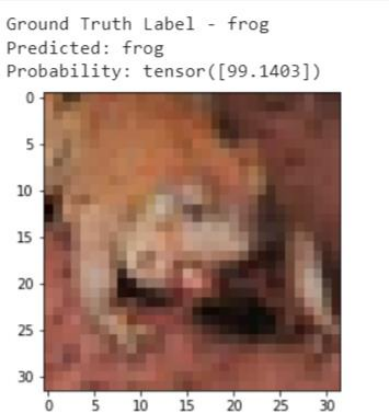
Source:https://paperswithcode.com/method/googlenet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Failure and Success Case

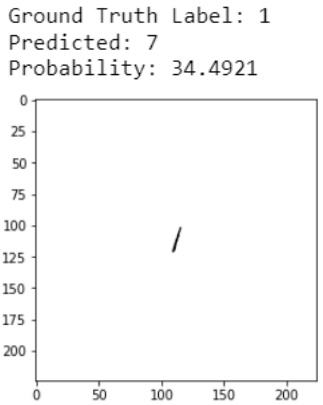
- **CIFAR10**

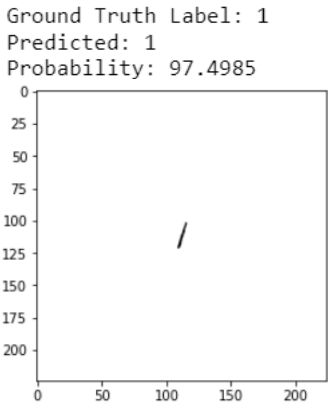




Failure case from VGG16 with Data Augmentation Success case from ResNet18 with Data Augmentation

- **MNIST**



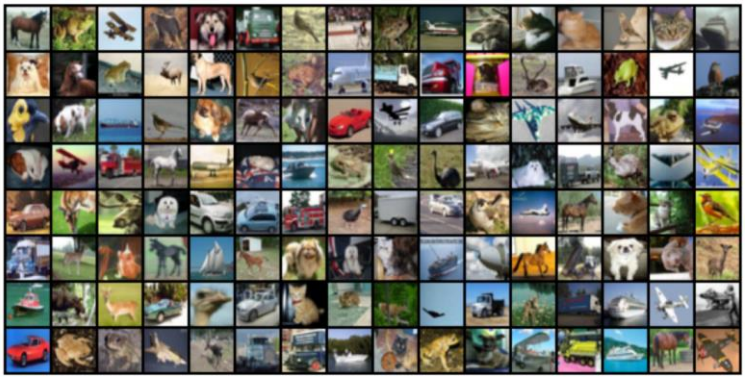


Failure case from GoogLeNet Success case from VGG16 with Data Augmentation

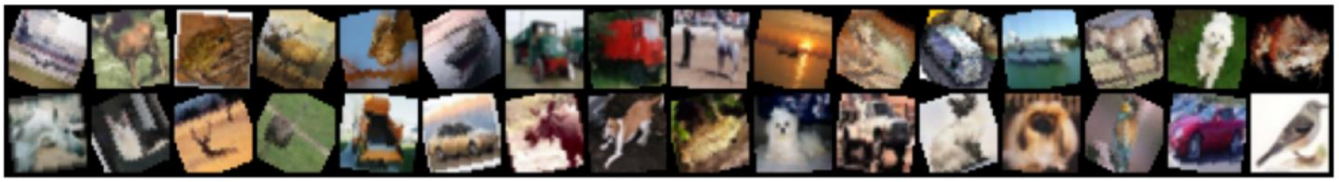
Data Augmentation

- **CIFAR10**

  Actual Image Data

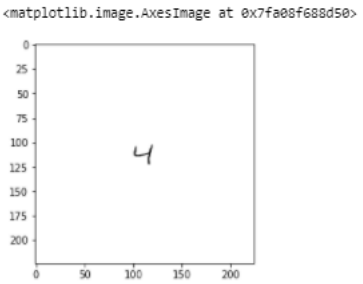


Augmented Image Data

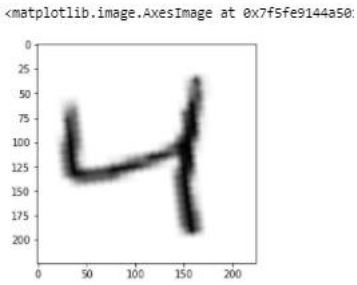


- **MNIST**

  Actual Image Data



  Augmented Image Data



Further Evaluation from Best Models

	precision	recall	f1-score	support
0	0.000	0.000	0.000	1000
1	0.000	0.000	0.000	1000
2	0.000	0.000	0.000	1000
3	0.000	0.000	0.000	1000
4	0.000	0.000	0.000	1000
5	0.100	1.000	0.182	1000
6	0.000	0.000	0.000	1000
7	0.000	0.000	0.000	1000
8	0.000	0.000	0.000	1000
9	0.000	0.000	0.000	1000
accuracy			0.100	10000
macro avg	0.010	0.100	0.018	10000
weighted avg	0.010	0.100	0.018	10000

GoogLeNet CIFAR10

	precision	recall	f1-score	support
0	0.995	0.987	0.991	980
1	0.988	0.998	0.993	1135
2	0.995	0.990	0.993	1032
3	0.998	0.988	0.993	1010
4	0.987	0.994	0.990	982
5	0.990	0.991	0.990	892
6	0.985	0.989	0.987	958
7	0.985	0.990	0.988	1028
8	0.997	0.995	0.996	974
9	0.991	0.988	0.990	1009
accuracy			0.991	10000
macro avg	0.991	0.991	0.991	10000
weighted avg	0.991	0.991	0.991	10000

GoogLeNet MNIST

	precision	recall	f1-score	support
0	0.880	0.829	0.854	1000
1	0.935	0.940	0.938	1000
2	0.786	0.786	0.786	1000
3	0.715	0.682	0.698	1000
4	0.769	0.893	0.826	1000
5	0.736	0.813	0.772	1000
6	0.909	0.840	0.873	1000
7	0.873	0.867	0.870	1000
8	0.909	0.914	0.912	1000
9	0.954	0.868	0.909	1000
accuracy			0.843	10000
macro avg	0.847	0.843	0.844	10000
weighted avg	0.847	0.843	0.844	10000

ResNet CIFAR10

	precision	recall	f1-score	support
0	0.996	0.996	0.996	980
1	0.996	0.996	0.996	1135
2	0.992	0.991	0.992	1032
3	0.993	0.997	0.995	1010
4	0.991	0.997	0.994	982
5	0.993	0.992	0.993	892
6	0.997	0.986	0.992	958
7	0.990	0.995	0.993	1028
8	0.995	0.995	0.995	974
9	0.993	0.990	0.992	1009
accuracy			0.994	10000
macro avg	0.994	0.994	0.994	10000
weighted avg	0.994	0.994	0.994	10000

ResNet MNIST

	precision	recall	f1-score	support
0	0.855	0.869	0.862	1000
1	0.920	0.937	0.928	1000
2	0.822	0.757	0.788	1000
3	0.708	0.736	0.722	1000
4	0.791	0.847	0.818	1000
5	0.761	0.782	0.771	1000
6	0.901	0.864	0.882	1000
7	0.900	0.863	0.881	1000
8	0.932	0.907	0.919	1000
9	0.901	0.914	0.908	1000
accuracy			0.848	10000
macro avg	0.849	0.848	0.848	10000
weighted avg	0.849	0.848	0.848	10000

VGGNet CIFAR

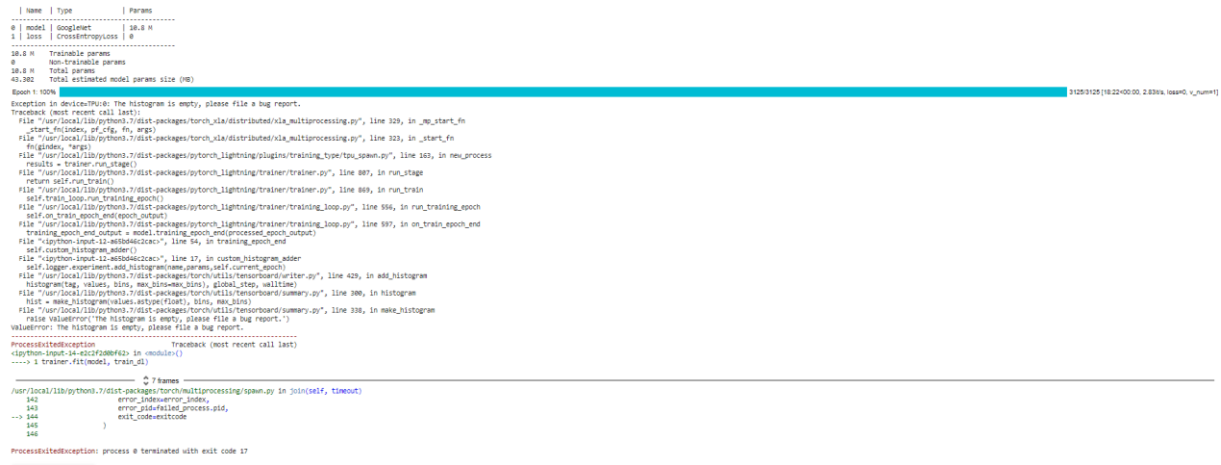
	precision	recall	f1-score	support
0	0.991	0.998	0.994	980
1	0.994	0.998	0.996	1135
2	0.989	0.998	0.994	1032
3	0.998	0.989	0.994	1010
4	0.988	0.996	0.992	982
5	0.986	0.994	0.990	892
6	0.994	0.984	0.989	958
7	0.996	0.990	0.993	1028
8	0.996	0.988	0.992	974
9	0.997	0.992	0.995	1009
accuracy			0.993	10000
macro avg	0.993	0.993	0.993	10000
weighted avg	0.993	0.993	0.993	10000

VGGNet MNIST

Runtime Screenshots

- Successful training (GoogLeNet on MNIST dataset)

- Training failure due to memory error (GoogLeNet on CIFAR10 dataset)



Logs corresponding to best models can be viewed within respective notebooks as output.

Tensor Logs

Training accuracy and training loss values for each of 10 epochs are present in the ‘Training logs’ folder for each of the combinations tried.

Note -

Due to submission memory limitation, graphs for training and further evaluation screenshots were added only for best models.