A Survey of Very Deep Neural Networks for Image Classification

Amir Sepehr Aminian
School of Electrical Engineering and Computer Science
Queen Mary University of London
London, United Kingdom
a.aminian@se21.qmul.ac.uk

***Abstract*—This survey will investigate different features of some of the most famous neural network architectures for the task of image classification. In this study, we used MNIST and CIFAR10 to test the capabilities of our models. Particularly we will inspect GoogLeNet, VGG16 and ResNet in this survey.**

***Index Terms*—Neural Networks, GoogLeNet, VGG, resNet**

I. 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. This survey will investigate different features of some of the most famous deep neural network architectures for the task of image classification. In this study, we used MNIST and CIFAR10 to test the capabilities of our models. Particularly we will inspect GoogLeNet, VGG16 and ResNet in this survey. This survey consists of 6 chapters. The first chapter is an introduction to the topic and an overview of the document. The second chapter is a critical analysis of the three models comparing them and the advancements since the introduction. The third chapter is a more detailed explanation of the models, data pre-processing and parameter tuning. The next chapter presents the results taken out with multiple settings and the fifth chapter is concluding the whole experiments.

II. 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 GoogLeNetb5 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.

III. METHODOLOGY

A. Model Architectures

*1) **VGG16**:* VGGNet has many variants differing only in depth; from 11 weight layers (8 convolutional and 3 fullyconnected 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 maxpooling 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 maxpooling 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.

*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 maxpooling 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 maxpooling 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) **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 batchnormalization 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.

B. 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.

C. 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 with added momentum and using ADAM optimizer. ADAM is normally giving better results but sometimes convergence is only achieved with stochastic gradient descent.

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.

IV. RESULTS

A. 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.

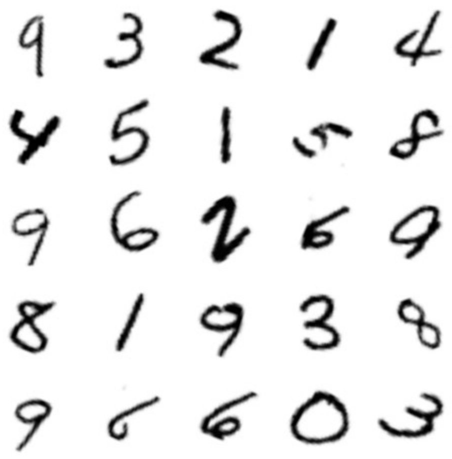


Fig. 1. MNIST sample.

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.

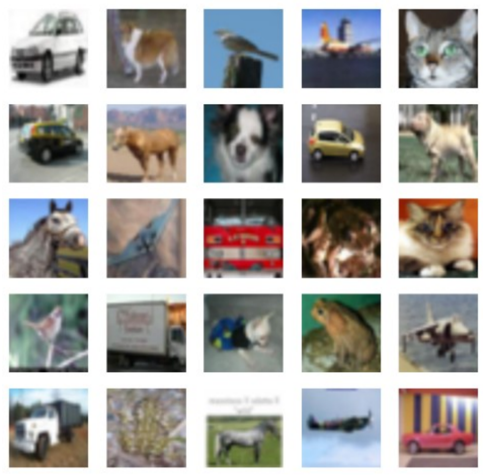


Fig. 2. CIFAR10 sample.

B. Test Results

1) ***MNIST:*** **VGG16:** We have different variants of VGGNet which were trained and evaluated on MNIST dataset.

TABLE I
ACCURACY RESULTS WITH DIFFERENT SETTINGS

Model	Accuracy
VGG16	99.1
VGG11	99.1
VGG19	83
VGG16 with lr=0.01	96.9
VGG16 with SGD	90.2
VGG16 with SGD and momentum	98
VGG16 with ADAM	98.2
VGG16 with Data Augmentation	98.98

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

TABLE II
ACCURACY RESULTS WITH DIFFERENT SETTINGS

Model	Accuracy
ResNet18	99.3
ResNet18 with lr=0.01	98.1
ResNet18 with SGD	97
ResNet18 with SGD and momentum	97.3
ResNet18 with Adam	98.29
ResNet18 with Data Augmentation	97.9

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

TABLE III
ACCURACY RESULTS WITH DIFFERENT SETTINGS

Model	Accuracy
GoogLeNet with adam	99.23
GoogLeNet with adam and data augmentation	99.23
GoogLeNet with data augmentation	99.1

*2) **CIFAR10:*** **VGG16:** We have different variants of VGGNet which were trained and evaluated on MNIST dataset.

TABLE IV
ACCURACY RESULTS WITH DIFFERENT SETTINGS

Model	Accuracy
VGG16	78.6
VGG11	73.6
VGG19	78.5
VGG16 with lr=0.01	79.2
VGG16 with SGD	65.6
VGG16 with SGD and momentum	78.6
VGG16 with ADAM	82.3
VGG16 with Data Augmentation	72.5

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

TABLE V
ACCURACY RESULTS WITH DIFFERENT SETTINGS

Model	Accuracy
ResNet18	78.9
ResNet18 with lr=0.01	80.4
ResNet18 with SGD	64.6
ResNet18 with SGD and momentum	78.4
ResNet18 with Adam	81.8
ResNet18 with Data Augmentation	81.2

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

TABLE VI
ACCURACY RESULTS WITH DIFFERENT SETTINGS

Model	Accuracy
GoogLeNet with SGD	76.48
GoogLeNet with SGD and data augmentation	76.1
GoogLeNet with data augmentation	99.1

C. 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 80% 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.

V. 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 though sometimes SGD turned better results. 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-ofthe-art model for image classification as also learned from the literature review.

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

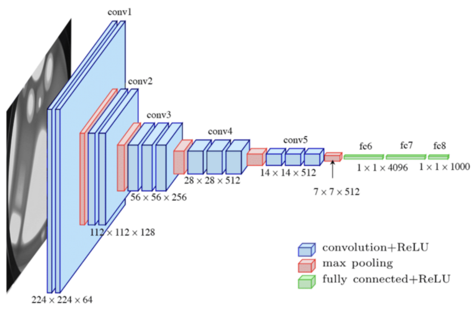


Fig. 3. VGG16 architecture.

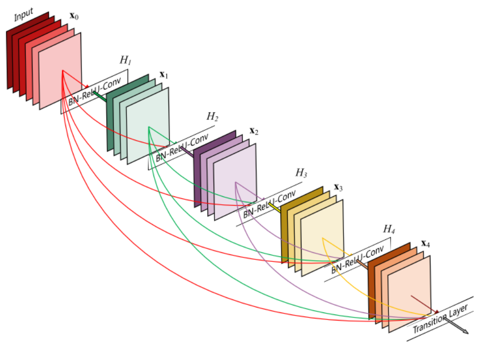


Fig. 4. ResNet architecture.

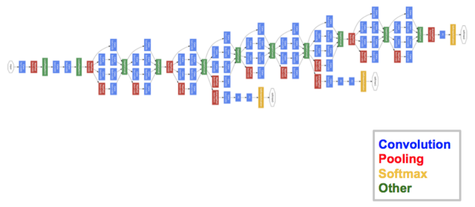
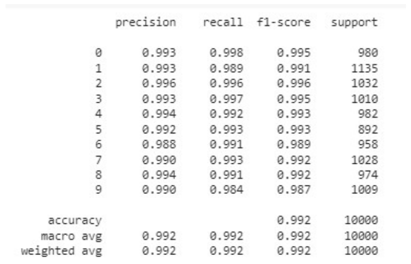
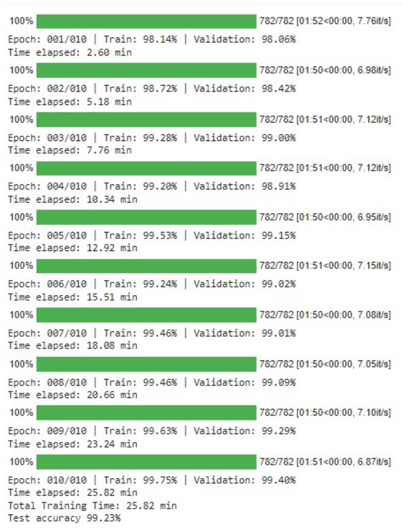


Fig. 5. GoogLeNet architecture.



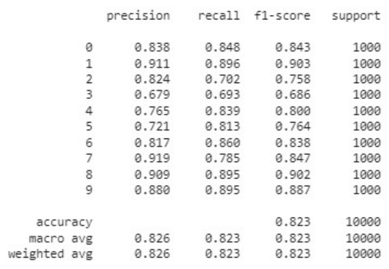
```
              precision    recall  f1-score   support

           0      0.993     0.998     0.995       980
           1      0.993     0.989     0.991      1135
           2      0.996     0.996     0.996      1032
           3      0.993     0.997     0.995      1010
           4      0.994     0.992     0.993       982
           5      0.992     0.993     0.993       892
           6      0.988     0.991     0.989       958
           7      0.990     0.993     0.992      1028
           8      0.994     0.991     0.992       974
           9      0.990     0.984     0.987      1009

    accuracy                          0.992     10000
   macro avg      0.992     0.992     0.992     10000
weighted avg      0.992     0.992     0.992     10000
```

Fig. 6. MNIST VGG classification report



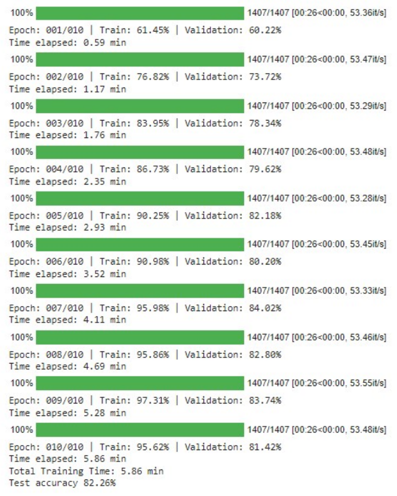
```
100% 782/782 [01:52<00:00, 7.76it/s]
Epoch: 001/010 | Train: 98.14% | Validation: 98.06%
Time elapsed: 2.60 min
100% 782/782 [01:50<00:00, 6.98it/s]
Epoch: 002/010 | Train: 98.72% | Validation: 98.42%
Time elapsed: 5.18 min
100% 782/782 [01:51<00:00, 7.12it/s]
Epoch: 003/010 | Train: 99.28% | Validation: 99.00%
Time elapsed: 7.76 min
100% 782/782 [01:51<00:00, 7.12it/s]
Epoch: 004/010 | Train: 99.20% | Validation: 98.91%
Time elapsed: 10.34 min
100% 782/782 [01:50<00:00, 6.95it/s]
Epoch: 005/010 | Train: 99.53% | Validation: 99.15%
Time elapsed: 12.92 min
100% 782/782 [01:51<00:00, 7.15it/s]
Epoch: 006/010 | Train: 99.24% | Validation: 99.02%
Time elapsed: 15.51 min
100% 782/782 [01:50<00:00, 7.08it/s]
Epoch: 007/010 | Train: 99.46% | Validation: 99.01%
Time elapsed: 18.08 min
100% 782/782 [01:50<00:00, 7.05it/s]
Epoch: 008/010 | Train: 99.46% | Validation: 99.09%
Time elapsed: 20.66 min
100% 782/782 [01:50<00:00, 7.10it/s]
Epoch: 009/010 | Train: 99.63% | Validation: 99.29%
Time elapsed: 23.24 min
100% 782/782 [01:51<00:00, 6.87it/s]
Epoch: 010/010 | Train: 99.75% | Validation: 99.40%
Time elapsed: 25.82 min
Total Training Time: 25.82 min
Test accuracy 99.23%
```

Fig. 7. MNIST VGG training log



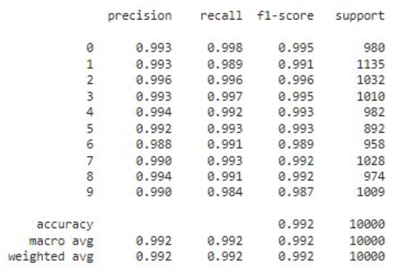
```
              precision    recall  f1-score   support

           0      0.838     0.848     0.843      1000
           1      0.911     0.896     0.903      1000
           2      0.824     0.702     0.758      1000
           3      0.679     0.693     0.686      1000
           4      0.765     0.839     0.800      1000
           5      0.721     0.813     0.764      1000
           6      0.817     0.860     0.838      1000
           7      0.919     0.785     0.847      1000
           8      0.909     0.895     0.902      1000
           9      0.880     0.895     0.887      1000

    accuracy                          0.823     10000
   macro avg      0.826     0.823     0.823     10000
weighted avg      0.826     0.823     0.823     10000
```

Fig. 8. CIFAR VGG classification report



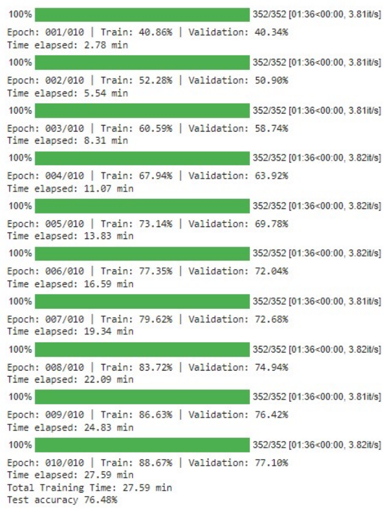
```
100% 1407/1407 [00:26<00:00, 53.36it/s]
Epoch: 001/010 | Train: 61.45% | Validation: 60.22%
Time elapsed: 0.59 min
100% 1407/1407 [00:26<00:00, 53.47it/s]
Epoch: 002/010 | Train: 76.82% | Validation: 73.72%
Time elapsed: 1.17 min
100% 1407/1407 [00:26<00:00, 53.29it/s]
Epoch: 003/010 | Train: 83.95% | Validation: 78.34%
Time elapsed: 1.76 min
100% 1407/1407 [00:26<00:00, 53.48it/s]
Epoch: 004/010 | Train: 86.73% | Validation: 79.62%
Time elapsed: 2.35 min
100% 1407/1407 [00:26<00:00, 53.28it/s]
Epoch: 005/010 | Train: 90.25% | Validation: 82.18%
Time elapsed: 2.93 min
100% 1407/1407 [00:26<00:00, 53.45it/s]
Epoch: 006/010 | Train: 90.98% | Validation: 80.20%
Time elapsed: 3.52 min
100% 1407/1407 [00:26<00:00, 53.33it/s]
Epoch: 007/010 | Train: 95.98% | Validation: 84.02%
Time elapsed: 4.11 min
100% 1407/1407 [00:26<00:00, 53.46it/s]
Epoch: 008/010 | Train: 95.86% | Validation: 82.80%
Time elapsed: 4.69 min
100% 1407/1407 [00:26<00:00, 53.55it/s]
Epoch: 009/010 | Train: 97.31% | Validation: 83.74%
Time elapsed: 5.28 min
100% 1407/1407 [00:26<00:00, 53.48it/s]
Epoch: 010/010 | Train: 95.62% | Validation: 81.42%
Time elapsed: 5.86 min
Total Training Time: 5.86 min
Test accuracy 82.26%
```

Fig. 9. CIFAR VGG training log



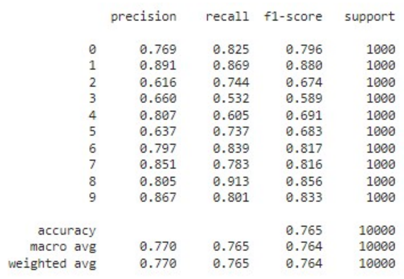
```
              precision    recall  f1-score   support

           0      0.993     0.998     0.995       980
           1      0.993     0.989     0.991      1135
           2      0.996     0.996     0.996      1032
           3      0.993     0.997     0.995      1010
           4      0.994     0.992     0.993       982
           5      0.992     0.993     0.993       892
           6      0.988     0.991     0.989       958
           7      0.990     0.993     0.992      1028
           8      0.994     0.991     0.992       974
           9      0.990     0.984     0.987      1009

    accuracy                          0.992     10000
   macro avg      0.992     0.992     0.992     10000
weighted avg      0.992     0.992     0.992     10000
```

Fig. 10. MNIST GoogLeNet classification report



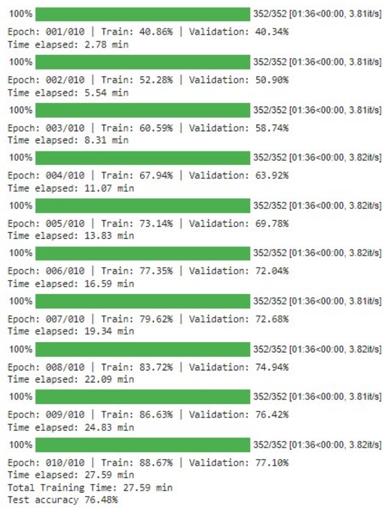
```
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 001/010 | Train: 40.86% | Validation: 40.34%
Time elapsed: 2.78 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 002/010 | Train: 52.28% | Validation: 50.90%
Time elapsed: 5.54 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 003/010 | Train: 60.59% | Validation: 58.74%
Time elapsed: 8.31 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 004/010 | Train: 67.94% | Validation: 63.92%
Time elapsed: 11.07 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 005/010 | Train: 73.14% | Validation: 69.78%
Time elapsed: 13.83 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 006/010 | Train: 77.35% | Validation: 72.04%
Time elapsed: 16.59 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 007/010 | Train: 79.62% | Validation: 72.68%
Time elapsed: 19.34 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 008/010 | Train: 83.72% | Validation: 74.94%
Time elapsed: 22.09 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 009/010 | Train: 86.63% | Validation: 76.42%
Time elapsed: 24.83 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 010/010 | Train: 88.67% | Validation: 77.10%
Time elapsed: 27.59 min
Total Training Time: 27.59 min
Test accuracy 76.48%
```

Fig. 11. MNIST GoogLeNet training log



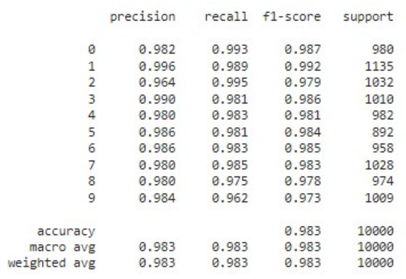
```
              precision    recall  f1-score   support

           0      0.769     0.825     0.796      1000
           1      0.891     0.869     0.880      1000
           2      0.616     0.744     0.674      1000
           3      0.660     0.532     0.589      1000
           4      0.807     0.605     0.691      1000
           5      0.637     0.737     0.683      1000
           6      0.797     0.839     0.817      1000
           7      0.851     0.783     0.816      1000
           8      0.805     0.913     0.856      1000
           9      0.867     0.801     0.833      1000

    accuracy                          0.765     10000
   macro avg      0.770     0.765     0.764     10000
weighted avg      0.770     0.765     0.764     10000
```

Fig. 12. CIFAR GoogLeNet classification report



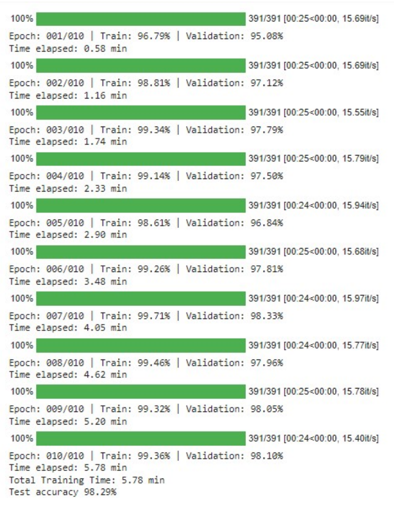
```
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 001/010 | Train: 40.86% | Validation: 40.34%
Time elapsed: 2.78 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 002/010 | Train: 52.28% | Validation: 50.90%
Time elapsed: 5.54 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 003/010 | Train: 60.59% | Validation: 58.74%
Time elapsed: 8.31 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 004/010 | Train: 67.94% | Validation: 63.92%
Time elapsed: 11.07 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 005/010 | Train: 73.14% | Validation: 69.78%
Time elapsed: 13.83 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 006/010 | Train: 77.35% | Validation: 72.04%
Time elapsed: 16.59 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 007/010 | Train: 79.62% | Validation: 72.68%
Time elapsed: 19.34 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 008/010 | Train: 83.72% | Validation: 74.94%
Time elapsed: 22.09 min
100% 352/352 [01:36<00:00, 3.81it/s]
Epoch: 009/010 | Train: 86.63% | Validation: 76.42%
Time elapsed: 24.83 min
100% 352/352 [01:36<00:00, 3.82it/s]
Epoch: 010/010 | Train: 88.67% | Validation: 77.10%
Time elapsed: 27.59 min
Total Training Time: 27.59 min
Test accuracy 76.48%
```

Fig. 13. CIFAR GoogLeNet training log



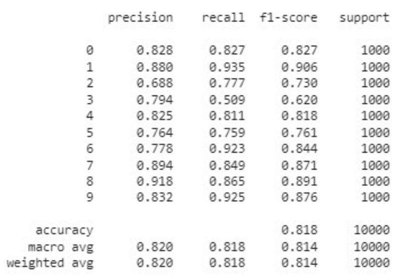
```
              precision    recall  f1-score   support

           0      0.982     0.993     0.987       980
           1      0.996     0.989     0.992      1135
           2      0.964     0.995     0.979      1032
           3      0.990     0.981     0.986      1010
           4      0.980     0.983     0.981       982
           5      0.986     0.981     0.984       892
           6      0.986     0.983     0.985       958
           7      0.980     0.985     0.983      1028
           8      0.980     0.975     0.978       974
           9      0.984     0.962     0.973      1009

    accuracy                          0.983     10000
   macro avg      0.983     0.983     0.983     10000
weighted avg      0.983     0.983     0.983     10000
```

Fig. 14. MNIST ResNet classification report



```
100% 391/391 [00:25<00:00, 15.69it/s]
Epoch: 001/010 | Train: 96.79% | Validation: 95.08%
Time elapsed: 0.58 min
100% 391/391 [00:25<00:00, 15.69it/s]
Epoch: 002/010 | Train: 98.81% | Validation: 97.12%
Time elapsed: 1.16 min
100% 391/391 [00:25<00:00, 15.55it/s]
Epoch: 003/010 | Train: 99.34% | Validation: 97.79%
Time elapsed: 1.74 min
100% 391/391 [00:25<00:00, 15.79it/s]
Epoch: 004/010 | Train: 99.14% | Validation: 97.50%
Time elapsed: 2.33 min
100% 391/391 [00:24<00:00, 15.94it/s]
Epoch: 005/010 | Train: 98.61% | Validation: 96.84%
Time elapsed: 2.90 min
100% 391/391 [00:25<00:00, 15.68it/s]
Epoch: 006/010 | Train: 99.26% | Validation: 97.81%
Time elapsed: 3.48 min
100% 391/391 [00:24<00:00, 15.97it/s]
Epoch: 007/010 | Train: 99.71% | Validation: 98.33%
Time elapsed: 4.05 min
100% 391/391 [00:24<00:00, 15.77it/s]
Epoch: 008/010 | Train: 99.46% | Validation: 97.96%
Time elapsed: 4.62 min
100% 391/391 [00:25<00:00, 15.78it/s]
Epoch: 009/010 | Train: 99.32% | Validation: 98.05%
Time elapsed: 5.20 min
100% 391/391 [00:24<00:00, 15.40it/s]
Epoch: 010/010 | Train: 99.36% | Validation: 98.10%
Time elapsed: 5.78 min
Total Training Time: 5.78 min
Test accuracy 98.29%
```

Fig. 15. MNIST ResNet training log



```
              precision    recall  f1-score   support

           0      0.828     0.827     0.827      1000
           1      0.880     0.935     0.906      1000
           2      0.688     0.777     0.730      1000
           3      0.794     0.509     0.620      1000
           4      0.825     0.811     0.818      1000
           5      0.764     0.759     0.761      1000
           6      0.778     0.923     0.844      1000
           7      0.894     0.849     0.871      1000
           8      0.918     0.865     0.891      1000
           9      0.832     0.925     0.876      1000

    accuracy                          0.818     10000
   macro avg      0.820     0.818     0.814     10000
weighted avg      0.820     0.818     0.814     10000
```

Fig. 16. CIFAR ResNet classification report

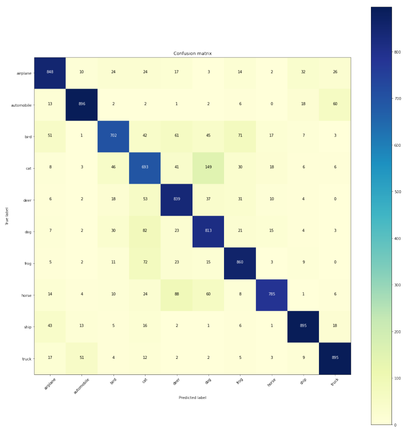
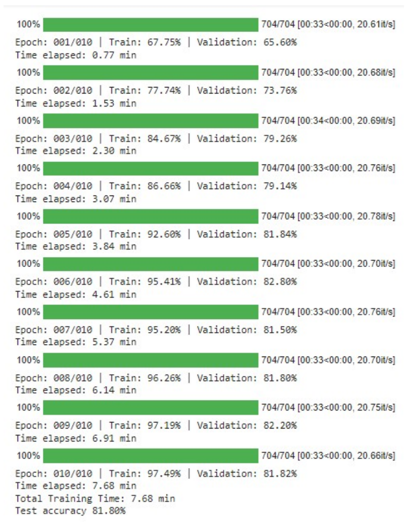


Fig. 19. CIFAR VGG confusion matrix



```
100% 704/704 [00:33<00:00, 20.61it/s]
Epoch: 001/010 | Train: 67.75% | Validation: 65.60%
Time elapsed: 0.77 min
100% 704/704 [00:33<00:00, 20.68it/s]
Epoch: 002/010 | Train: 77.74% | Validation: 73.76%
Time elapsed: 1.53 min
100% 704/704 [00:34<00:00, 20.69it/s]
Epoch: 003/010 | Train: 84.67% | Validation: 79.26%
Time elapsed: 2.30 min
100% 704/704 [00:33<00:00, 20.76it/s]
Epoch: 004/010 | Train: 86.66% | Validation: 79.14%
Time elapsed: 3.07 min
100% 704/704 [00:33<00:00, 20.78it/s]
Epoch: 005/010 | Train: 92.60% | Validation: 81.84%
Time elapsed: 3.84 min
100% 704/704 [00:33<00:00, 20.70it/s]
Epoch: 006/010 | Train: 95.41% | Validation: 82.80%
Time elapsed: 4.61 min
100% 704/704 [00:33<00:00, 20.76it/s]
Epoch: 007/010 | Train: 95.20% | Validation: 81.50%
Time elapsed: 5.37 min
100% 704/704 [00:33<00:00, 20.70it/s]
Epoch: 008/010 | Train: 96.26% | Validation: 81.80%
Time elapsed: 6.14 min
100% 704/704 [00:33<00:00, 20.75it/s]
Epoch: 009/010 | Train: 97.19% | Validation: 82.20%
Time elapsed: 6.91 min
100% 704/704 [00:33<00:00, 20.66it/s]
Epoch: 010/010 | Train: 97.49% | Validation: 81.82%
Time elapsed: 7.68 min
Total Training Time: 7.68 min
Test accuracy 81.80%
```

Fig. 17. CIFAR ResNet training log

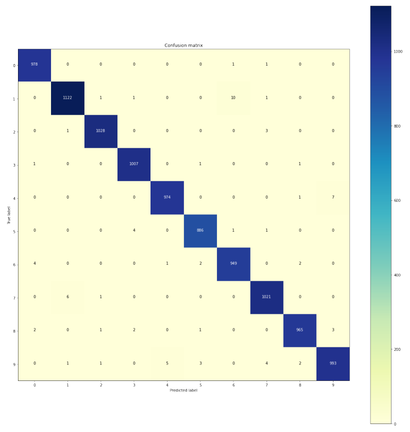


Fig. 20. MNIST GoogLeNet confusion matrix

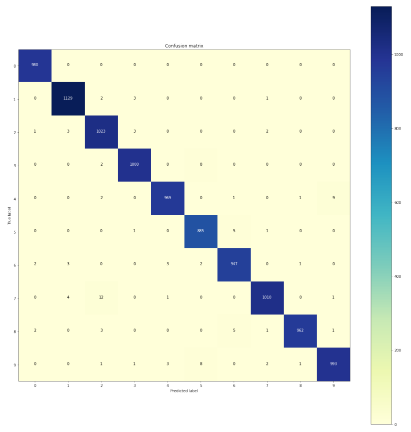


Fig. 18. MNIST VGG confusion matrix

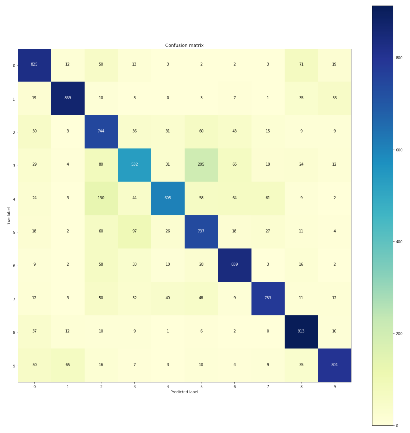


Fig. 21. CIFAR GoogLeNet confusion matrix

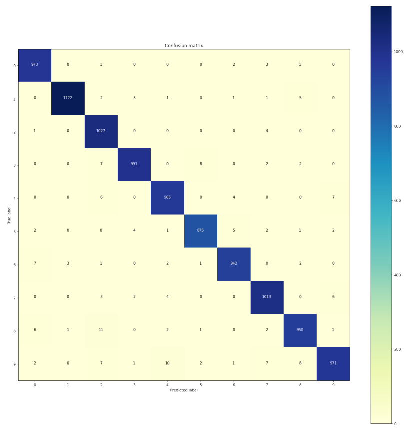


Fig. 22. MNIST ResNet confusion matrix

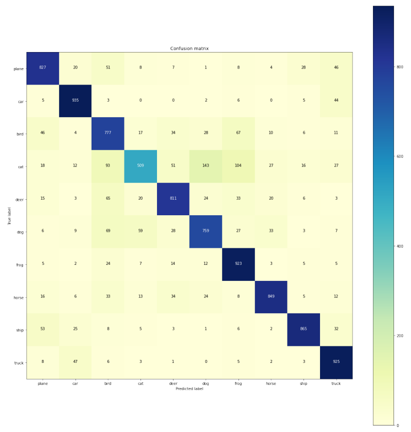


Fig. 23. CIFAR ResNet confusion matrix