# Deeper Networks for Image Classification

Iliass Elyaakoubi Benssaleh
200922847
ECS795P
Deep Learning and Computer Vision

## I. Introduction

Deeper neural networks have been shown to be essential for attaining ground-breaking performance gains in the quickly developing field of image classification. Researchers have gradually expanded the complexity and depth of neural architectures as computer resources have become more economical and available, which has resulted in notable improvements in accuracy and dependability. This work focuses on two well-known models that have been instrumental in pushing the limits of image classification job performance: VGG-16 and ResNet18. The purpose of this research is to investigate the effects of architectural depth and learning rate modifications on classification accuracy using these models on two popular datasets: MNIST and CIFAR-10. Our analysis offers comparative insights that could direct future improvements in the field, in addition to revealing each model's robustness and sensitivity to hyperparameters.

## II. Critical Analysis / Related work

Image classification has always been a challenging problem to tackle but with deeper networks developed, we have seen an astonishing improvement in this field. These networks are constantly getting more complex and increasing in depth. Over the years, model architecture has rapidly progressed, and researchers are developing unique ideas and innovative methods. This analysis will explore six crucial models that have shaped the evolution of deep learning in image classification.

### 1. LeNet (1998)

The evolution of deep learning models in image classification started with LeNet in 1998 [1]. LeNet was composed of two convolutional layers, average pooling layers, and three fully connected layers for the output. Tested on handwritten characters, LeNet achieved an impressive 0.6% error rate. This kick started much research on the field of computer vision and image classification.[2].

### 2. AlexNet (2012)

AlexNet was a truly remarkable achievement in the field of image classification back in 2012 [3]. It was first introduced in the ImageNet competition, Large Scale Visual Recognition Challenge (ILSVRC), which is a large visual database used for image recognition research with over 14 million images. The team behind this model was called Supervision and achieved an error rate of 15.315%, which outperformed the competition by a large margin. The model was composed of five convolutional layers with max pooling and three fully connected layers for the output [4]. The reason for this model's success was a combination of very innovative implementations. Firstly, they used ReLU activation layers to solve the vanishing gradient problem. In addition, they have also applied dropout in the fully connected layer to prevent overfitting. Finally, the implementation of data augmentation on the input images allows the increase and diversity of the training set which leads to a lower error rate.

### 3. Network in Network (2013)

In 2013, the following year, Min Lin et al. introduced Network in Network (NIN) [5]. NIN put forward a new design that substituted standard fully connected layers with 1x1 convolutional layers to boost the model's ability to represent information. This new development increased levels of nonlinearity, improving feature abstraction without significantly raising computational load. NIN performed well in the CIFAR-10 and CIFAR-100 dataset by reducing the error 1-5% compared to other methods such as Stochastic Pooling and DropConnect.

### 4. GoogleNet (Inception v1, 2014)

One year later, a model developed by Google was introduced. GoogleNet was a way deeper model of 22 layers that allowed data to be analysed from different perspectives simultaneously [6]. This was possible through the implementation of Inception blocks which are composed of many convolutional layers of different windows sizes, max pooling layers and by using the concept of NiN (1x1 convolutional layers) to change the number of channels. The whole architecture of GoogleNet is composed of three groups of nine inception blocks with max-pooling in between and a global average pooling to further reduce the parameters and the model less prone to overfitting. These series of ideas have allowed the model to run cheaper than the competition while achieving a higher accuracy. It was able to win the 2014 ImageNet competition, achieving a low error rate of 6.67% [7].

### 5. VGG (2014)

In the same year, VGG was the runner of ImageNet 2014 [7] just behind GoogleNet with an error rate of

7.3%. VGG was a very popular contender because of its focus on simplicity by experimenting with the effect of increasing the model’s depth and it proved that it was critical for achieving a good performance. For previous models, increasing the model's depth was more computationally expensive and it greatly led to overfitting. However, the VGG model allowed the ability to extend to more than 16 layers of convolutions. The depth is increased by implementing a series of blocks that are stacked on top of each other in series. A VGG block [8] is composed of several 3x3 convolutional layers and a max pool layer at the end to reduce spatial resolution of the feature maps. VGG has now grown popular in many feature extraction tasks and transfer learning.

6. ResNet (Residual Network, 2015)

Finally, ResNet is another winner of the ImageNet in 2015. It achieved an astonishing error rate of 3.57% [9]. The reason of ResNet success is the introduction of residual blocks and skip connections. Residual blocks [10] are composed of several layers where the main path has in addition a shortcut connection (skip connection) that allows the input/output to skip one or more layers. This connection adds the input of the previous layer into the output of the last layer. This allows the implementation of large models of many layers because skip connection tackles the issue of the vanishing gradient.

With the release of each model, we can see how much impact each model presents on future models and observe a trend towards more simplistic and deeper models. This allowed better feature representation and an increase in performance.

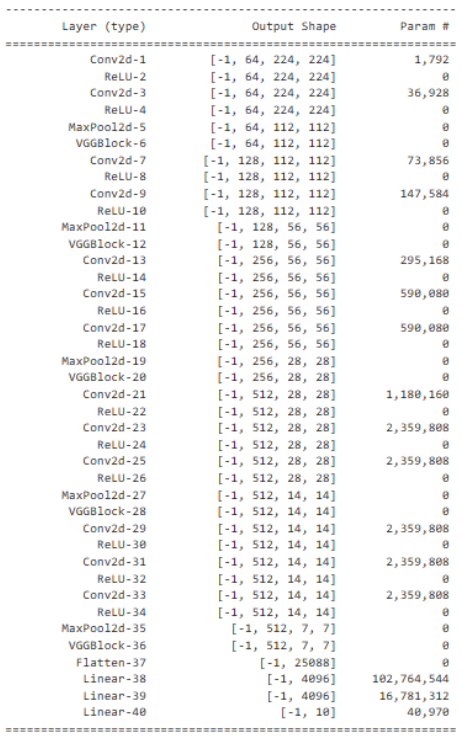
## III. Method / Model Description

In this study, two different models (VGG-16, and ResNet18) were employed to assess the impact deeper networks on image classification performance using the MNIST and CIFAR-10 dataset.

### A. Model Architecture

1. VGG-16

Out of the different versions of VGG models, I choose VGG-16 for its balance of accuracy and manageable complexity. It introduces five VGG blocks where each are composed of a 3x3 convolutional layer with 1 padding that’s activated by a ReLU layer. At the end of the block a 2x2 max pool layer is also added. In the VGG-16 models, the first two blocks have two convolutional layers with their respective ReLU layer while the next three blocks have three convolutional layer and their respective ReLU layer. The number of filters the first block is 64 and progressively doubles until the last two blocks remain at a filter count of 512, using a filter setting of (64,128,256,512,512). The input channel of the first block depends on the input of the image like when using a black and white dataset such as MNIST I used a single channel and when using a RGB dataset such as CIFAR-10 I used 3 channels. The input images will be fed from the first VGG block until the last block where the model captures increasingly complex features, block by block. After the final VGG block, the output is flattened, transforming the data from a multidimensional array into a single one-dimensional vector. This flattened vector is then fed into three fully connected layers where the first two have 4096 neurons each. These layers are responsible for integrating the high-level features extracted by the convolutional blocks. Finally, the last fully connected layer has a few neurons equal to the number of classes in the target dataset (e.g., 10 for MNIST, 10 for CIFAR-10). This final layer is typically followed by a SoftMax activation function, which outputs a probability distribution over all possible classes for the input image.

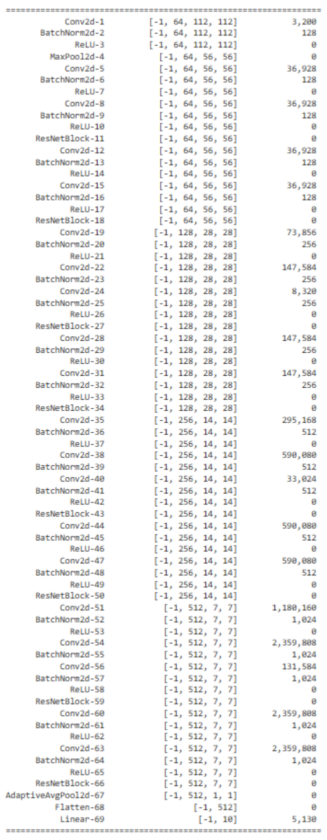


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 224, 224]           1,792
              ReLU-2         [-1, 64, 224, 224]               0
            Conv2d-3         [-1, 64, 224, 224]          36,928
              ReLU-4         [-1, 64, 224, 224]               0
         MaxPool2d-5         [-1, 64, 112, 112]               0
          VGGBlock-6         [-1, 64, 112, 112]               0
            Conv2d-7        [-1, 128, 112, 112]          73,856
              ReLU-8        [-1, 128, 112, 112]               0
            Conv2d-9        [-1, 128, 112, 112]         147,584
             ReLU-10        [-1, 128, 112, 112]               0
        MaxPool2d-11          [-1, 128, 56, 56]               0
         VGGBlock-12          [-1, 128, 56, 56]               0
           Conv2d-13          [-1, 256, 56, 56]         295,168
             ReLU-14          [-1, 256, 56, 56]               0
           Conv2d-15          [-1, 256, 56, 56]         590,080
             ReLU-16          [-1, 256, 56, 56]               0
           Conv2d-17          [-1, 256, 56, 56]         590,080
             ReLU-18          [-1, 256, 56, 56]               0
        MaxPool2d-19          [-1, 256, 28, 28]               0
         VGGBlock-20          [-1, 256, 28, 28]               0
           Conv2d-21          [-1, 512, 28, 28]       1,180,160
             ReLU-22          [-1, 512, 28, 28]               0
           Conv2d-23          [-1, 512, 28, 28]       2,359,808
             ReLU-24          [-1, 512, 28, 28]               0
           Conv2d-25          [-1, 512, 28, 28]       2,359,808
             ReLU-26          [-1, 512, 28, 28]               0
        MaxPool2d-27          [-1, 512, 14, 14]               0
         VGGBlock-28          [-1, 512, 14, 14]               0
           Conv2d-29          [-1, 512, 14, 14]       2,359,808
             ReLU-30          [-1, 512, 14, 14]               0
           Conv2d-31          [-1, 512, 14, 14]       2,359,808
             ReLU-32          [-1, 512, 14, 14]               0
           Conv2d-33          [-1, 512, 14, 14]       2,359,808
             ReLU-34          [-1, 512, 14, 14]               0
        MaxPool2d-35            [-1, 512, 7, 7]               0
         VGGBlock-36            [-1, 512, 7, 7]               0
          Flatten-37                [-1, 25088]               0
           Linear-38                 [-1, 4096]     102,764,544
           Linear-39                 [-1, 4096]      16,781,312
           Linear-40                   [-1, 10]          40,970
================================================================
```

*Figure 1 - VGG16 model summary.*

2. ResNet-18

For the ResNet module I picked ResNet-18 for its balance between depth and computational efficiency. The main feature for ResNet is the Residual block which includes skip connections/shortcuts to skip some layers. A residual block consists of two 3x3 convolutional layers with stride of1 1 and padding of 1. In addition, a 1x1 convolutional layer is added for dimensionality reduction. The ResNet-18 architecture starts with a 7x7 convolutional layer and a batch normalisation that is activated by a ReLU function and followed by 3x3 max pooling layer. It then uses 4 pair of residual blocks in the following configuration of filters (2x64, 2x128, 2x256, 2x512). In between each pair we add a stride of 2 for down-sampling. After the residual blocks, global average pooling reduces dimensionality before feeding into a fully connected layer (with a neuron count matching the target classes) and a final SoftMax activation for classification. ResNet-18's design allows it to capture deep feature hierarchies with less computational overhead and improved training dynamics compared to deeper models like ResNet-34 or ResNet-50.



```
================================================================
            Conv2d-1         [-1, 64, 112, 112]           3,200
       BatchNorm2d-2         [-1, 64, 112, 112]             128
              ReLU-3         [-1, 64, 112, 112]               0
         MaxPool2d-4           [-1, 64, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]          36,928
       BatchNorm2d-6           [-1, 64, 56, 56]             128
              ReLU-7           [-1, 64, 56, 56]               0
            Conv2d-8           [-1, 64, 56, 56]          36,928
       BatchNorm2d-9           [-1, 64, 56, 56]             128
             ReLU-10           [-1, 64, 56, 56]               0
      ResNetBlock-11           [-1, 64, 56, 56]               0
           Conv2d-12           [-1, 64, 56, 56]          36,928
      BatchNorm2d-13           [-1, 64, 56, 56]             128
             ReLU-14           [-1, 64, 56, 56]               0
           Conv2d-15           [-1, 64, 56, 56]          36,928
      BatchNorm2d-16           [-1, 64, 56, 56]             128
             ReLU-17           [-1, 64, 56, 56]               0
      ResNetBlock-18           [-1, 64, 56, 56]               0
           Conv2d-19          [-1, 128, 28, 28]          73,856
      BatchNorm2d-20          [-1, 128, 28, 28]             256
             ReLU-21          [-1, 128, 28, 28]               0
           Conv2d-22          [-1, 128, 28, 28]         147,584
      BatchNorm2d-23          [-1, 128, 28, 28]             256
           Conv2d-24          [-1, 128, 28, 28]           8,320
      BatchNorm2d-25          [-1, 128, 28, 28]             256
             ReLU-26          [-1, 128, 28, 28]               0
      ResNetBlock-27          [-1, 128, 28, 28]               0
           Conv2d-28          [-1, 128, 28, 28]         147,584
      BatchNorm2d-29          [-1, 128, 28, 28]             256
             ReLU-30          [-1, 128, 28, 28]               0
           Conv2d-31          [-1, 128, 28, 28]         147,584
      BatchNorm2d-32          [-1, 128, 28, 28]             256
             ReLU-33          [-1, 128, 28, 28]               0
      ResNetBlock-34          [-1, 128, 28, 28]               0
           Conv2d-35          [-1, 256, 14, 14]         295,168
      BatchNorm2d-36          [-1, 256, 14, 14]             512
             ReLU-37          [-1, 256, 14, 14]               0
           Conv2d-38          [-1, 256, 14, 14]         590,080
      BatchNorm2d-39          [-1, 256, 14, 14]             512
           Conv2d-40          [-1, 256, 14, 14]          33,024
      BatchNorm2d-41          [-1, 256, 14, 14]             512
             ReLU-42          [-1, 256, 14, 14]               0
      ResNetBlock-43          [-1, 256, 14, 14]               0
           Conv2d-44          [-1, 256, 14, 14]         590,080
      BatchNorm2d-45          [-1, 256, 14, 14]             512
             ReLU-46          [-1, 256, 14, 14]               0
           Conv2d-47          [-1, 256, 14, 14]         590,080
      BatchNorm2d-48          [-1, 256, 14, 14]             512
             ReLU-49          [-1, 256, 14, 14]               0
      ResNetBlock-50          [-1, 256, 14, 14]               0
           Conv2d-51            [-1, 512, 7, 7]       1,180,160
      BatchNorm2d-52            [-1, 512, 7, 7]           1,024
             ReLU-53            [-1, 512, 7, 7]               0
           Conv2d-54            [-1, 512, 7, 7]       2,359,808
      BatchNorm2d-55            [-1, 512, 7, 7]           1,024
           Conv2d-56            [-1, 512, 7, 7]         131,584
      BatchNorm2d-57            [-1, 512, 7, 7]           1,024
             ReLU-58            [-1, 512, 7, 7]               0
      ResNetBlock-59            [-1, 512, 7, 7]               0
           Conv2d-60            [-1, 512, 7, 7]       2,359,808
      BatchNorm2d-61            [-1, 512, 7, 7]           1,024
             ReLU-62            [-1, 512, 7, 7]               0
           Conv2d-63            [-1, 512, 7, 7]       2,359,808
      BatchNorm2d-64            [-1, 512, 7, 7]           1,024
             ReLU-65            [-1, 512, 7, 7]               0
      ResNetBlock-66            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
          Flatten-68                  [-1, 512]               0
           Linear-69                   [-1, 10]           5,130
================================================================
```

*Figure 2 - ResNet-18 model summary.*

B. Improvements

Firstly, because VGG-16 is very computationally hungry with over 100M parameters, resized the image to 32x32, added batch normalisation to speed up training times and some dropout for better accuracy. In addition, to increase the accuracy of the classification for the CIFAR-10 dataset, I added some data augmentation to increase the variety of data samples.

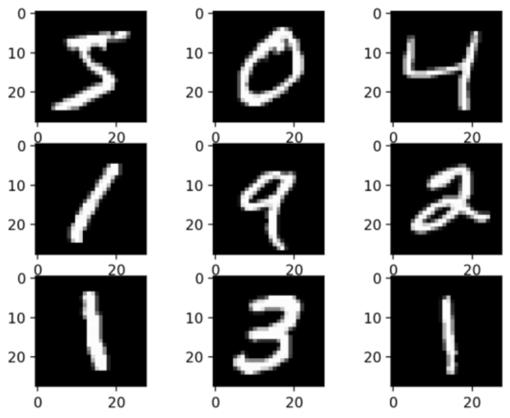
## IV. Experiments

### A. Datasets

For the three selected models and their improvements, they are trained using the MNIST character dataset and the CIIFAR-10 dataset.

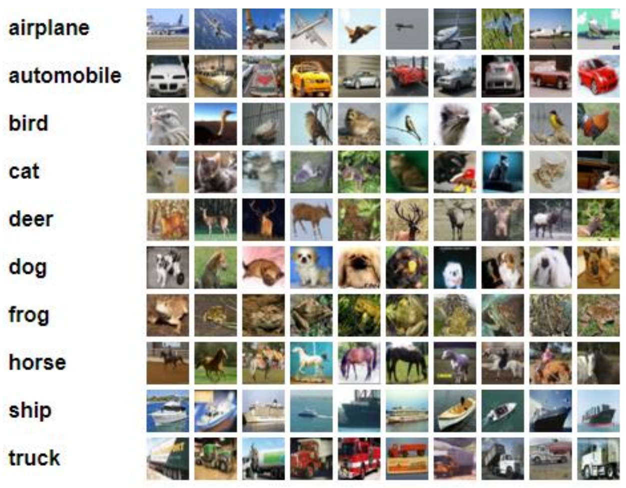
**-** MNIST

The Modified National Institute of Standards [11] and Technology dataset is a large database containing handwritten digits from 0 to 9 (Figure x). It is made up of 60,000 examples for the training set and 10,000 examples for the test set. The images are in black and white and have a size of 28x28 pixels. It is a commonly used dataset to compare different image classification models.



- CIFAR-10

The CIFAR [12] dataset are labelled subsets of the 80 million tiny images dataset collected by Alex Krizhevsky et al. The CIFAR-10 dataset is composed of 50000 training images and 10000 test images. Each image is a 32x32 colour image in 10 classes. The training set is divided into five training batches and one test batch, each with 10000 images. The classes are mutually exclusive where each image contains exactly one class.



B. Testing Results

Both VGG and ResNet have been trained on the MNIST and CIFAR10 dataset. I ran 10 epoch for the MNIST set and 20 epochs for the CIFAR10 set using the Stochastic gradient descent optimizer with many different learning rates to evaluate the performance of both models. For the loss , I used the cross-entropy loss. Every run is also stored as a tensorboard file. Following, I evaluated each model over three different learning rates for each dataset and analysed each performance individually.

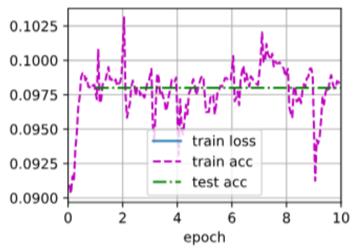
1. **MNIST Dataset**

Because the input photos are too small to perform any feature extractions with VGG and ResNet, I enlarged the images to 32 by 32 pixels and ran 10 epochs on this dataset to stop the model from overfitting any further.

- VGG16

**Lr = 0.1:**

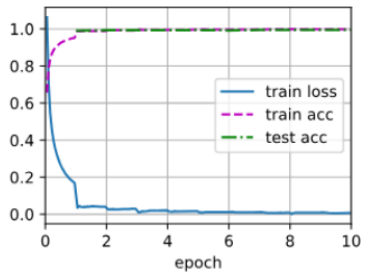
```
Epoch 10, Loss: nan, Train Acc: 0.099, Test Acc: 0.098
5024.8 examples/sec on cuda
```



Here we can see the accuracy fluctuating because the learning rate is large, and the model keep overshooting the minima. Achieving and testing accuracy of 9.8% which is basically guessing at random.

**Lr = 0.01**

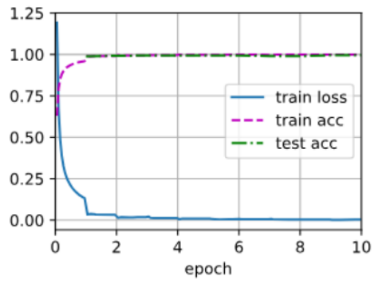
```
Epoch 10, Loss: 0.007, Train Acc: 0.998, Test Acc: 0.995
4999.1 examples/sec on cuda
```



With a lower learning rate, we can now see the loss gradually decreasing and accuracy increasing. Both the accuracy converges very quickly after the first epoch reaching an accuracy of 99.5%

**Lr = 0.001**

```
Epoch 10, Loss: 0.003, Train Acc: 0.999, Test Acc: 0.995
5059.9 examples/sec on cuda
```



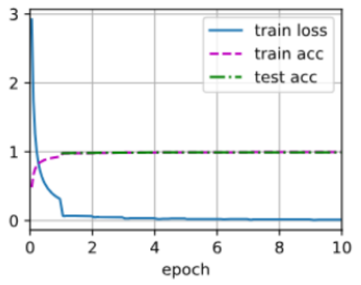
Similarly, with a lower learning rate we start from a higher lower loss but very quickly reaching a loss of around 0 and an accuracy of 99.5%.

VGG16 performed very well with the MNIST dataset, reaching the accuracy of 99% after only the first epoch for lower learning rates.

- ResNet18

**Lr = 0.1**

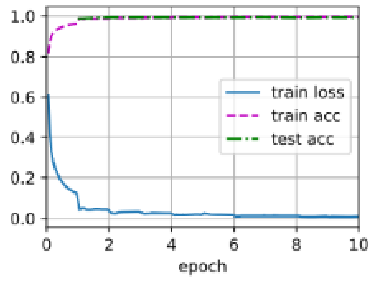
```
Epoch 10, Loss: 0.015, Train Acc: 0.995, Test Acc: 0.992
6473.0 examples/sec on cuda
```



The Resnet model can train accurately even using a higher leaning rate, converging also after the first epoch and reaching an accuracy of 99.2%.

**Lr = 0.01**

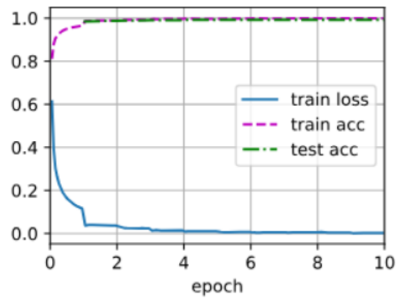
```
Epoch 10, Loss: 0.008, Train Acc: 0.997, Test Acc: 0.993
6949.0 examples/sec on cuda
```



Similarly, the model achieved similar results, however we did get a 0.1% increase in test accuracy.

**Lr = 0.001**

```
Epoch 10, Loss: 0.003, Train Acc: 0.999, Test Acc: 0.992
6427.6 examples/sec on cuda
```



With the smallest learning rate, we receive results are the like both learning rate and achieving an accuracy of 99.2%.

ResNet18 also performed very good on the MNIST dataset. It achieved results over 99% for every single learning rate and achieving the maximum accuracy of 99.3% using 0.01 as the learning rate.

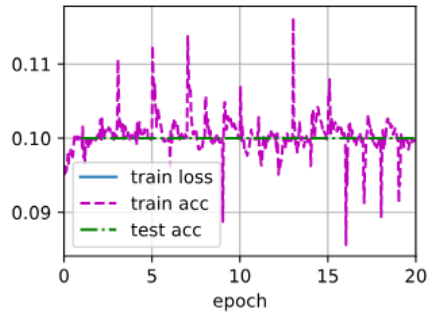
## 2. CIFAR10 Dataset

For this dataset, the input images are already set to 32x32, so we don't need to change its dimensions. In addition, because we are working with images with many features, data augmentation has been applied and each model will train the data for 20 epoch to captures as many features as possible.

- VGG16

**Lr = 0.1**

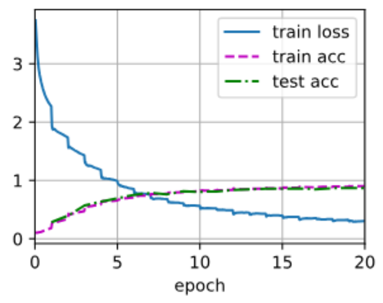
```
Epoch 20, Loss: nan, Train Acc: 0.100, Test Acc: 0.100
3245.7 examples/sec on cuda
```



Using a large learning rate, we can see that the accuracy is fluctuating and not converging toward a minimal. The model returns an accuracy of 10% which is the same as random guessing.

**Lr = 0.01**

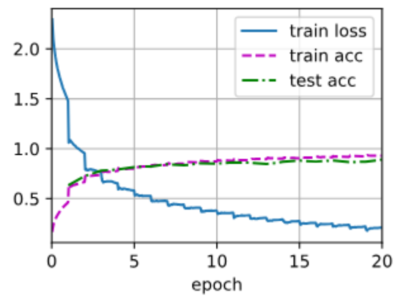
```
Epoch 20, Loss: 0.303, Train Acc: 0.900, Test Acc: 0.871
3241.8 examples/sec on cuda
```



Using 0.01 as the learning rate, the model's accuracy is increasing, and the loss is decreasing. The accuracy slowly increases for the first 5 epochs but starts to converge after reaching an accuracy of 87.1%

**Lr = 0.001**

```
Epoch 20, Loss: 0.208, Train Acc: 0.929, Test Acc: 0.892
3217.5 examples/sec on cuda
```



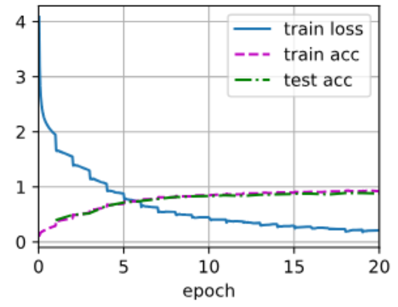
With a higher accuracy, the model rapidly increases to around 76% in 5 epochs and slowly converges to around 89.2%.

For the CIFAR10, VGG16 did very good to achieve an accuracy close to 90%. A way to improve this could be to resize to 224x224 and use a more powerful hardware to train using a model with over 100M parameters.

- ResNet18

**Lr = 0.1**

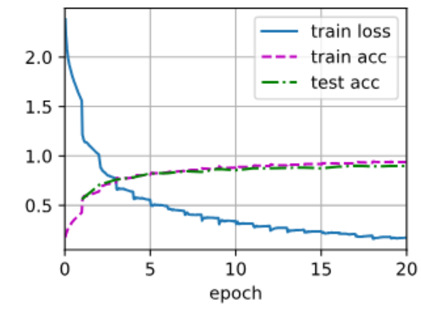
```
Epoch 20, Loss: 0.210, Train Acc: 0.926, Test Acc: 0.881
2861.2 examples/sec on cuda
```



Similarly, with the MNIST dataset, the model was able to train the dataset using a higher learning rate and achieving a higher accuracy of 88.1%.

**Lr = 0.01**

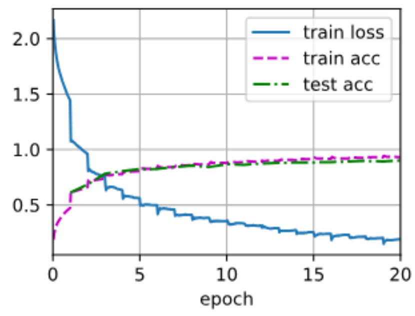
```
Epoch 20, Loss: 0.174, Train Acc: 0.938, Test Acc: 0.900
2660.3 examples/sec on cuda
```



Decreasing the results, has also decreased the loss and increased the accuracy to 90%.

**Lr = 0.001**

```
Epoch 20, Loss: 0.191, Train Acc: 0.932, Test Acc: 0.901
2683.1 examples/sec on cuda
```



Using the lowest learning rate, the model achieved an increase of 0.1%,

C. Model Comparison

- MNIST

| VGG16 | | | Resnet18 | | |
|---|---|---|---|---|---|
| 0.1 | 0.01 | 0.001 | 0.1 | 0.01 | 0.001 |
| 9.8% | 99.5% | 99.5% | 99.2% | 99.4% | 99.3% |

When comparing the performance between both models, we can see that VGG16 has performed the best across both learning of 0.01 and 0.001. It was able to achieve 0.1% more than ResNet18. However, for a higher learning rate of 0.1, ResNet18 was able to get a higher accuracy of 99.2% in comparison to 9.8%.

- CIFAR10

| VGG16 | | | Resnet18 | | |
|---|---|---|---|---|---|
| 0.1 | 0.01 | 0.001 | 0.1 | 0.01 | 0.001 |
| 10% | 87.1% | 89.2% | 88.7% | 90% | 90.1% |

For the CIFAR10 dataset, we can see that ResNet18 outperforms VGG16 across all learning rates. The best performance is seen with ResNet18 at the lowest learning rate of 0.001, it achieved an accuracy of 90.1%.

## V. Conclusion

Using the MNIST and CIFAR-10 datasets, this work has thoroughly examined how well the VGG-16 and ResNet18 models perform at various learning rates. Our results highlight the significance of selecting suitable learning rates and network topologies based on the features of the dataset. Although VGG-16 demonstrated remarkable efficacy at reduced learning rates, it showed notable declines in performance at increased rates, as demonstrated by the MNIST dataset. As evidenced by CIFAR-10, ResNet18, on the other hand, proved to be a more reliable option for tasks requiring higher dimensional and more complicated images due to its exceptional consistency and resilience across a range of learning rates. These findings support the idea that, even though deeper networks typically improve performance, particular architecture decisions and hyperparameter configurations are still very important for optimising image categorization systems.

## VI. References

[1] Lecun, Y., L Eon Bottou, Bengio, Y. and Patrick Haaner Abstract| (1998). Gradient-Based Learning Applied to Document Recognition. PROC. OF THE IEEE. [online] Available at: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

[2] Paperswithcode.com. (2022). Papers with Code - MNIST Benchmark (Image Classification). [online] Available at: https://paperswithcode.com/sota/image-classification-on-.

[3] Image-net.org. (2024). ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012). [online] Available at: https://image-net.org/challenges/LSVRC/2012/results.html.

[4] Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, [online] 60(6), pp.84–90. doi:https://doi.org/10.1145/3065386.

[5] Lin, M., Chen, Q. and Yan, S. (n.d.). Network In Network. [online] Available at: https://arxiv.org/pdf/1312.4400.

[6] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (n.d.). Going deeper with convolutions. [online] Available at: https://arxiv.org/pdf/1409.4842.

[7] Image-net.org. (2014). ILSVRC2014 Results. [online] Available at: https://image-net.org/challenges/LSVRC/2014/results.php.

[8] Simonyan, K. and Zisserman, A. (2015). Published as a conference paper at ICLR 2015 VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. [online] Available at: https://arxiv.org/pdf/1409.1556.

[9] Image-net.org. (2015). ILSVRC2015 Results. [online] Available at: https://image-net.org/challenges/LSVRC/2015/results.php.

[10] He, K., Zhang, X., Ren, S. and Sun, J. (2015). Deep Residual Learning for Image Recognition. [online] Available at: https://arxiv.org/pdf/1512.03385.

[11] Lecun.com. (2024). MNIST handwritten digit database, Yann LeCun, Corinna Cortes, and Chris Burges. [online] Available at: http://yann.lecun.com/exdb/mnist/].

[12] Toronto.edu. (2024). CIFAR-10 and CIFAR-100 datasets. [online] Available at: https://www.cs.toronto.edu/~kriz/cifar.html.