Comparing LeNet, VGG16, and a modified EfficientNet CNN architecture

Aaron Chauhan
AKC200003
The University of Texas School of Computer Science

Kofi Boadu
KOB200000
The University of Texas School of Computer Science

Abstract(A:55% B:45%)

This report presents an analysis of three convolutional neural network architectures: LeNet, VGG16, and a modified version of EfficientNet. We begin by outlining the fundamental concepts behind each architecture, showing features and applications in the field of image recognition. Our methodology involves implementing each model on a standardized dataset (Fashion-Mnist and CIFAR10) to evaluate their performance in loss, and training and testing and validation accuracy. The LeNet model is the simplest and demonstrates adequate performance with low-resolution images but falls short in handling more complex visuals. VGG16, known for its depth and significantly outperforms LeNet in accuracy but at the cost of increased resources of the CPU. The modified EfficientNet, designed for higher efficiency, shows a balanced performance with a focus on scalability and adaptability to various environments. Results suggest that VGG16 is capable for tasks requiring high accuracy in a quicker time period for small-medium size datasets, and the modified EfficientNet is more suitable for applications demanding complex datasets but not for small-medium datasets since it overfits. This analysis shows selecting models based on specific requirements and constraints in applications.

Introduction(A:50% B:50%)

There have been rapid advancements in machine learning and deep learning has revolutionized the field of image processing, which has been leading to the development of numerous convolution neural networks (CNN) architectures. CNN is inspired by our visual skills that every living organism has to show the 2D structure of images. Each architecture offers unique advantages and limitations, thus it makes the choice of the most suitable model an important decision depending on the dataset and practical application. This project is motivated by the need to evaluate and compare the effectiveness of three distinct CNN architectures which are LeNet,VGG16, and a modified version of EfficientNet (B0 version). The goal is to determine which of the architectures has the best balance of accuracy and computational time/efficiency for different image recognition tasks. The problem addressed by this study is first to perform a detailed comparative analysis of the architectures in terms of performance on 2 different datasets, and second, to evaluate the computation scalability of the project. The main points to the study are:

1. In-Depth performance comparison of the LeNet, VGG16, and modified EfficientNet models, and highlight their accuracy and efficiency when processing these high resolution images.
2. An assessment of computation resources of each model, providing insights into their practical applicability in resource- constrained environments.
3. Recommendations for selecting most appropriate CNN architecture best on Specific applications and the need of different data sets.

The report starts with comprehensive overviews of concepts necessary to understand CNNs, including the architecture of convolutional lays, the role of pooling and activation functions, and the significance of depth and breadth in the design itself. The features and the development of LeNet, VGG16, and EfficientNet models are also discussed to provide context for their comparisons.

To test the capabilities of the models, there were a series of experiments that were conducted using a standard dataset for images. These experiments were designed to measure the performance metrics of model accuracy, training time, and show the number of parameters, which directly relate to the model complexity and resources. The findings from these experiments show the strengths and weaknesses of each model but provide clear explanations for how they would be used, showcasing how limited processing power computing systems can have model accuracy.

This detailed analysis offers insights into the selection of CNN architectures, thus it tailored the specific demands of image recognition tasks, and optimizes the performance in different environments.

The project shown uses 2 datasets which are Fashion-Mnist and CIFAR10, which would evaluate the accuracy of the models in different environments. The EfficientNet CNN model would use the B0 version of the program that has been modified to meet the criteria of both datasets. It has been modified by fine tuning the model to fit the

What our group did in the experiment is first create the model and the forward for both the LeNet and the VGG16 and run it through the neural network to fine-tune the weights. The CNN would ingest all the data represented in the image as a matrix in pixel value. Then CNNs would use feature detections to apply several filters known as

kernels. Then we use Relu activation to introduce non-linearity and allow the model to learn more complex patterns. Then the pooling layers such as max pooling would be used to reduce spatial dimensions of the input volume for the next convolutional layer load. This operation helps to decrease the computational load of parameters. After several convolutional and pooling layers, all layers are connected to all neurons. These layers classify the features. The output layer would then identify the picture and category that consist of 10 classes. After that backpropagation would be used to optimize the weights of the filters using optimization algorithms to minimize the loss function. It then iterates over the dataset using the 25 epochs. With each epoch the CNN continues to learn and improve its accuracy on the training data, which also tunes its filters. For the modified EfficientNet the difference is that we are using a pretrained model that has been fine tuned for our datasets as well as using SiLU as activation functions. Finally we also have tensorboard graphs and exploratory data analysis consisting of PCA, LDA, and t-SNE to visualize the datasets.
We both worked on all 3 CNNs together.

Related Work(A:60% B:40%)
Our work is related to several areas of research spanning different CNNS.
Datasets. Our work has to first be looking at the datasets provided. The work proposed [1] summarized is stating how classification accuracies are dependent on the class similarities as well as pixel randomization and how dilated convolution can recover pixel correlations. The conclusion of the paper highlights the limitations of standard convolutional neural networks (CNNs) when dealing with images that have undergone pixel-wise permutations. It shows that CNNs trained on natural images significantly differ in performance compared to those trained on permuted images, using a multilayer perceptron (MLP) as a baseline for comparison. The focus is not on the absolute accuracy values but on the relative performances between different setups. This means that Fashion-MNIST has a sample of greater variability of their pixel permutations. This would decrease the performance compared to a MNIST dataset. For CIFAR10 we can see that these data sets have images that are 3 color channels that are not center which means there is natural pixel wise randomization and shows that it is more complex. Relating to our project this shows how our project accuracy is different when comparing Fashion-MNIST and CIFAR10 since CIFAR10 is more complex.

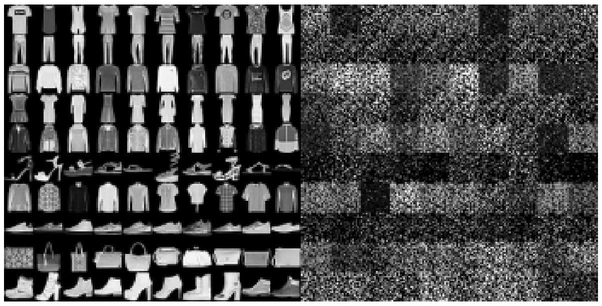


Figure 1. Left Panel: random sample of Fashion-MNIST images; right panel: pixel-wise randomization of the same sample [1].

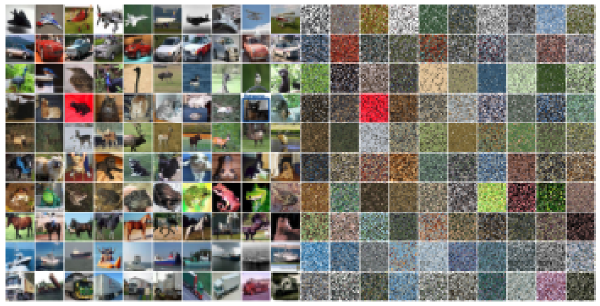


Figure 2. Left Panel: random sample of CIFAR10 images; right panel: pixel-wise randomization of the same sample[1].

LeNet. Our work also has to look at LeNet-5 CNN and how it works. The work proposed [2] summarized is related to gradient-based techniques applied to research to highlight multilayer neural networks trained with the back propagation algorithm as effective as gradient based learning. The networks can generate complex high dimensional patterns. The CNNs are tailored to 2d shapes. The paper introduces graph transformers networks, which enable global training modules such as segmentation and recognition to optimize overall performance across the entire system. The paper describes how it benefits when using different transformers and CNNs such as LeNet.
LeNet-5 explained in this paper consisted of 7 layers designed for 32 x 32 pixel images which was used to explain the algorithms used. LeNet has Input, C1,S2,C3,S4,C5,F6,Output. Used to handle complex pattern recognition. LeNet-5 employs RELU as activation function and the output layer uses Euclidean Radial Basis function units to compute the euclidean distance between input vector and fixed parameter vectors, which interpret the distance as a measurement fit for class identification. This is usually optimized for network effectiveness in real world recognition tasks.

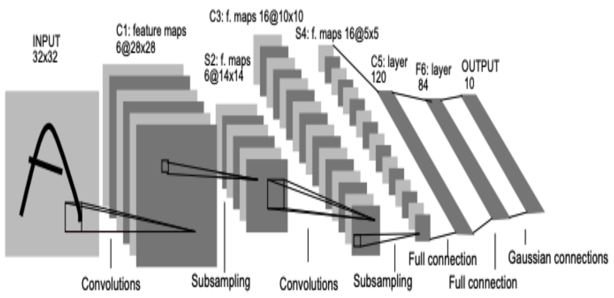
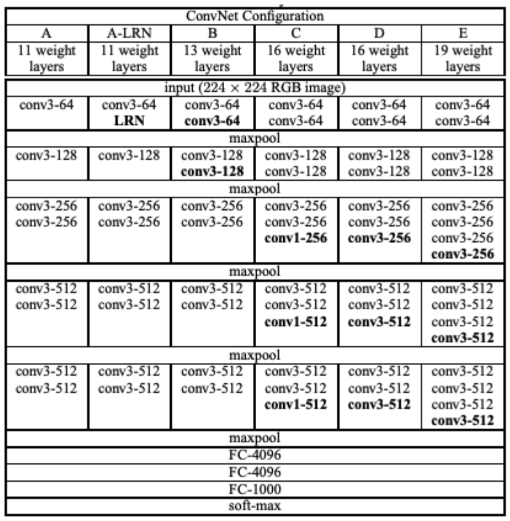


Figure 3. Architecture of LeNet-5 convolutional Neural Network for image recognition. This shows each plane as a feature map with weights[2].

The conclusion of the paper emphasizes the role of learning and enhancing the performance of automatic pattern recognition systems. The paper shows we can eliminate manual feature extractions from CNN, reduce hand crafter heuristics by graph transformer networks and parameter tuning, using gradient based learning to dynamically employ loss function, and integrate multiple recognition tasks to optimize strategies to perform segmentation and recognition in an interconnected process.
Our project uses these techniques to make sure our LeNet-5 works and it runs the program required to fulfill the dataset tasks at hand.

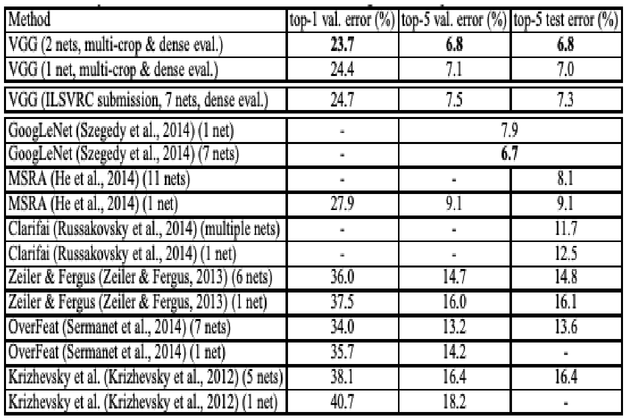
VGG16. Our work also has to look at VGG16 CNN and how it works. The work proposed [3] summarized is investigating the increasing depth of convolution networks on the accuracy of large scale image recognition. The study is primarily focused with 3x3 convolutional filters and finds improvement in accuracy can be achieved by extending the network depth to 16 layers. This is important since VGG16 has 16 layers. The approach significantly outperformed previous configurations and led their team to first place in the ImageNet Challenge 2014. The paper also states that deep network representation generalizes well to other datasets such as the data sets we will be using. VGG16 refers to an architecture that consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers.



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					

Figure 4. ConvNet shown in columns. The depth of the config increases A-E, and more layers are added. The convolutional layer parameter denotes the field size and number of channels[3].

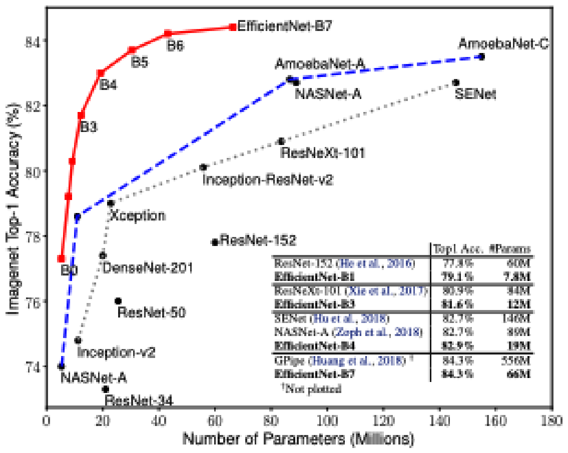
The distinctive aspects of VGG16 in the paper highlights the use of 3x3 convolution filters which our code uses based on the kernel size. Sall filters increase the depth of the network and allow more complex features at a lower cost of the parameters, while improving the network performance in large scale image settings. This depth and architecture style is effective and contributes to the success. The conclusion of the paper evaluates deep convolutional networks. The discussion is based on the top 5 test error and it shows that VGG with 2 nets and a dense evaluation has a top 5 test error of 6.8& which was the lowest compared to GoogleLeNet. This means that is is more effective and the models generalize well to a wide range of datasets and can outperform complex recognition pipelines. This pushes the boundaries of what can be achieved and sets new benchmarks for classification accuracy. The implications show how depper networks can process multiple levels otherwise couldn't before and hits at integration of different models, something our project had to do which is fine tune the VGG16 to our datasets.



Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Figure 5. Shows a table with comparisons with the ILSVRC classifications. The method denoted as VGG is shown as well with 16 and 19 layers[3].

EfficientNet. Our work also has to look at EfficientNet CNN and how it works. The work proposed [4] discusses a new approach to scaling Convolutional Neural Networks for enhanced performance. Usually ConvNets are developed with a fixed resource and scaled when resources are available. THis research identifies a balance that in networks depth, width, and resolution can significantly influence performance. To use the findings introduced, our method has to uniformly adjust 3 dimensions using compound coefficient, thus demonstrating improvements over the traditional methods. They enhance their approach by employing neural architecture to search to design a new baseline network, which is then used to develop models named EfficientNets. These models not only achieve superior accuracy and efficiency, which is highlighted through their models of B0-B7, but also their 85.4% accuracy on ImageNet- that is considerably smaller and faster during inference compared to the best Convolutional Networks currently at that time. The EfficientNets also perform exceptionally across other datasets, including CIFAR-100, with fewer parameters than previous models - these applications are far more accessible with parameters able to be tuned.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) [†]	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

[†]Not plotted

Figure 5. Compares Model Size and ImageNetAccuracy. EfficentNets significantly outperforms other ConvNets[4].

The conclusion of the paper on EfficientNet architecture provides an overview on scaling ConvNets and introduces a novel compound scaling method that is central to the advancements reported. The study begins with a critique of traditional scaling methods, which typically focus on increasing either the depth, width, or resolution of the networks independently. This approach leads to increased computational cost without proportional gains. Also, for limitations the method uses FLOPS which is a measurement of computational efficiency. B0 is used as a foundation and baseline and more layers with compound scaling can scale up to various sizes from B1-B7. The results are shown to achieve higher accuracy rates on ImageNet compared to models such as ResNet, while requiring significantly fewer parameters.

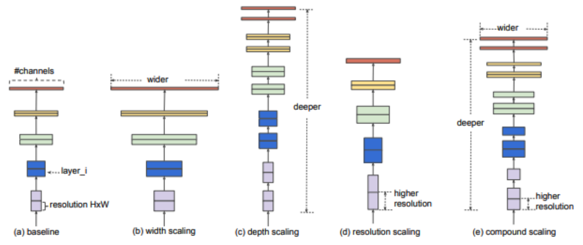


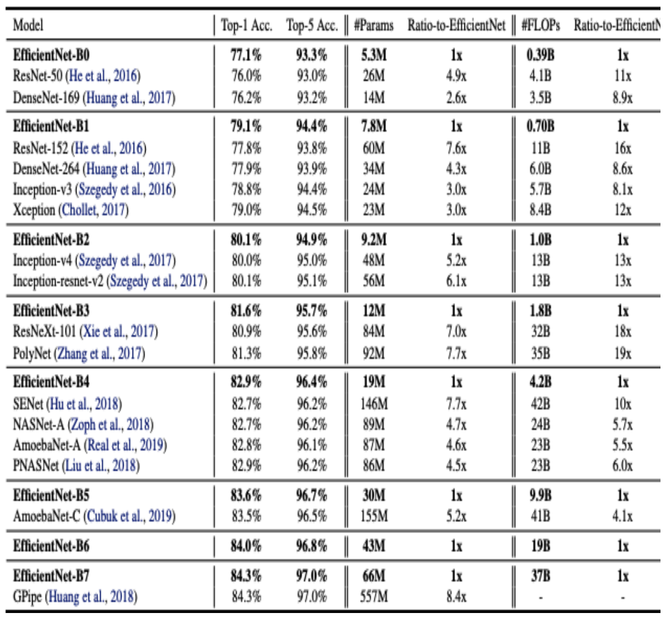
Figure 5. Model Scaling, and it shows a proposed compound scaling method that uniformly scales in three dimensions[4].

Moreover, the paper explores the impact of scaling methods on other popular ConvNets and provides evidence how the compound scaling method enhances performance, as well as fine tuning the datasets to others which our project will be doing. The models achieve and surpass leading models using fewer parameters and demonstrating greater robustness.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 6. Describes the EfficientNet-B0 baseline network[4].

Going back to the research papers' overarching question about the efficiency of ConvNet Scaling. The authors argue that single-dimension scaling methods are suboptimal and multi-dimensional are the way to go in the future. The paper closes by noting potential implications for future research in tailor made applications. The approach not only answers questions of how to improve network performance efficiently but also shows the potential of future studies.



Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientN
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	1x	**0.39B**	1x
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	1x	**0.70B**	1x
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	1x	**1.0B**	1x
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	1x	**1.8B**	1x
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	1x	**4.2B**	1x
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	1x	**9.9B**	1x
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	1x	**19B**	1x
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	1x	**37B**	1x
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Figure 7. Give the overall EfficientNet performance results on ImageNet using FLOPS to measure[4].

Proposed Approach(A:50% B:50%)

The three CNN architectures that are used in this report are LeNet,VGG16, and a modified version of EfficientNet. Looking at LeNet first, the LeNet architecture was invented in 1998 by Yann Lecun. Before LeNet, many neural networks struggled with efficiently processing complex visual data. LeNet consists of 7 layers(3 convolutional layers which apply filters to input data, 2 pooling layers which reduce spatial dimensions, and 2 fully connected layers which act like the classifier). LeNet's architecture is very simple compared to many other modern CNN architectures. LeNet performs well on relatively simple image classification tasks. Due to its small size and few parameters, LeNet can be trained quickly on most hardware. LeNet does have many places where it struggles though. LeNet may have issues with more complex image classification limiting its performance on datasets with more diverse images. It also performs worse compared to more modern architectures, especially on more challenging datasets. The loss function commonly used with LeNet is the cross-entropy loss. What the cross-entropy loss function does is calculate the difference between the predicted probability distribution and the actual distribution in the training data.

Moving on to VGG16, VGG16 is composed of 16 layers and can capture more complex features in images leading to higher accuracy compared to architectures like LeNet. The architecture follows a pattern of 3x3 convolutional filters with a stride of 1 and 2x2 max-pooling layers with a stride of 2. VGG16's architecture allows it to learn hierarchical representation of images like capturing low level features like edges and textures in early layers. VGG16 models that are already pre trained on large datasets can also be used as feature extractors for other tasks through transfer learning. This capability saves time when training datasets for specific tasks.Some disadvantages for VGG16 are that VGG16 is computationally expensive to train and requires more resources. Networks such as VGG16 are also prone to overfitting especially when the dataset is small and regularization techniques are required in order to prevent overfitting and increase performance. The VGG16 network architecture also used the cross-entropy loss function during training. VGG16 demonstrates the power of deep learning in visual recognition tasks and has paved the way for many advancements.

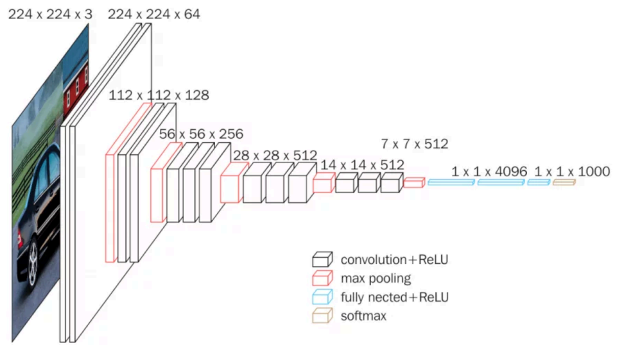


Figure 6. Shows a VGG16 Convolutional Network for Classification and Detection[6].

Lastly, the modified version of EfficientNet. Efficient Net is a CNN that is built upon the concept of compound scaling[5]. The idea behind this compound scaling is to scale three dimensions of a neural network, width, depth, and resolution. It was introduced by a team of researchers at Google AI. We specifically used EfficientNet B0 in our implementation which is the baseline network and it can be scaled up to larger variants which are denoted as EfficientNet-B1 to B7.EfficientNet achieves the peak of performance while requiring fewer parameters compared to other CNN architectures. Despite its efficiency, EfficentNet still maintains its high accuracy on various types of tasks. While EfficientNet offers efficient architecture, understanding and implementing compound scaling may require additional effort. EfficientNet's internal representations may be challenging to interpret.

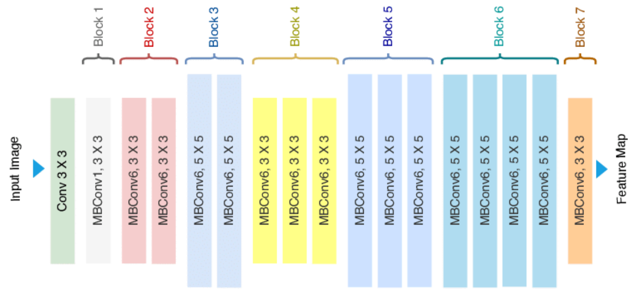


Figure 7. Basic look at EfficientNet Architecture[7]

Experiments(A:50% B:50%)

In this section, we will present our empirical setting and results from training the Fashion-MNIST, and CIFAR10 datasets.

The Fashion-MNIST dataset consists of 60,000 images and has a test set of 10,000 images. The CIFAR10 dataset consists also of 60,000 images and has a test set of 10,000 images. Each dataset is split into 10 classes of 1000 images each. For training we set the amount of epochs to 25.

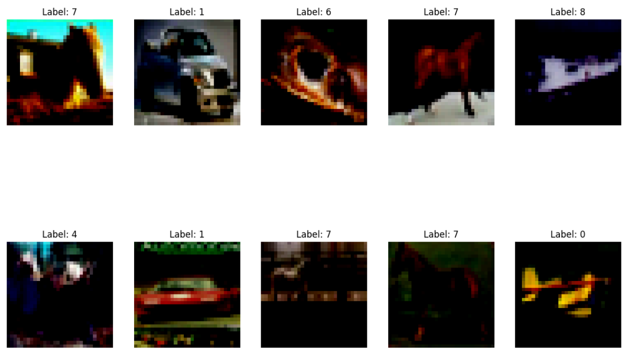


Figure 8. Random images taken from CIFAR dataset with their class labels.

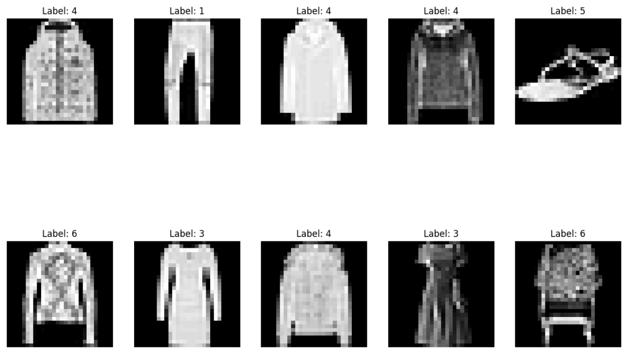
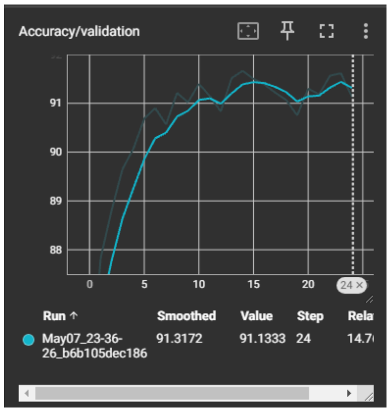


Figure 9. Random images taken from Fashion-MNIST dataset with their class labels.

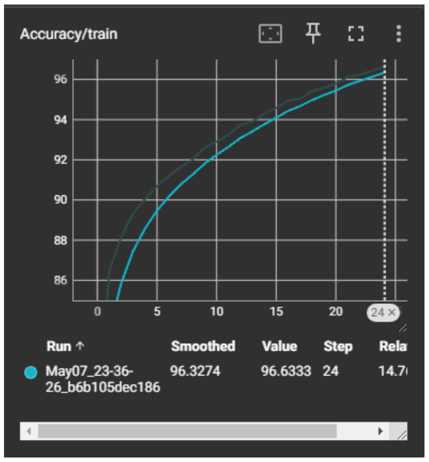
LeNet

Fashion-MNIST dataset results



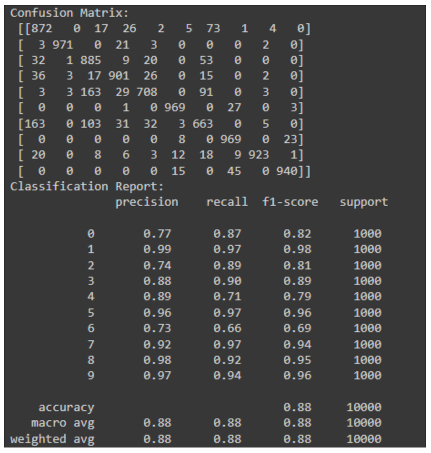
Run ↑ Smoothed Value Step Rela
May07_23-36-26_b6b105dec186 91.3172 91.1333 24 14.7

Validation Accuracy: 91.13%



Run ↑ Smoothed Value Step Rela
May07_23-36-26_b6b105dec186 96.3274 96.6333 24 14.7

Fashion Training Accuracy: 96.63%



```
Confusion Matrix:
 [[872   0  17  26   2   5  73   1   4   0]
 [  3 971   0  21   3   0   0   0   2   0]
 [ 32   1 885   9  20   0  53   0   0   0]
 [ 36   3  17 901  26   0  15   0   2   0]
 [  3   3 163  29 708   0  91   0   3   0]
 [  0   0   0   1   0 969   0  27   0   3]
 [163   0 103  31  32   3 663   0   5   0]
 [  0   0   0   0   0   8   0 969   0  23]
 [ 20   0   8   6   3  12  18   9 923   1]
 [  0   0   0   0   0  15   0  45   0 940]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.87      0.82      1000
           1       0.99      0.97      0.98      1000
           2       0.74      0.89      0.81      1000
           3       0.88      0.90      0.89      1000
           4       0.89      0.71      0.79      1000
           5       0.96      0.97      0.96      1000
           6       0.73      0.66      0.69      1000
           7       0.92      0.97      0.94      1000
           8       0.98      0.92      0.95      1000
           9       0.97      0.94      0.96      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

Figure 12. Classification Report and Confusion Matrix for Fashion-MNIST dataset Testing Accuracy 88%

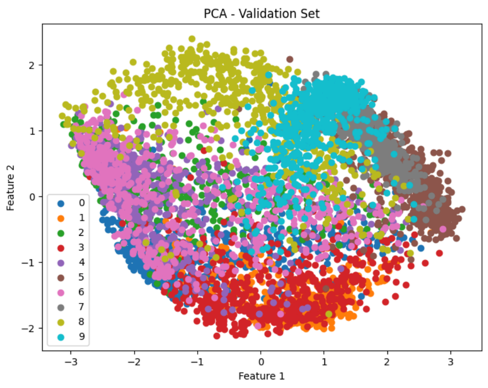


Figure 13. PCA

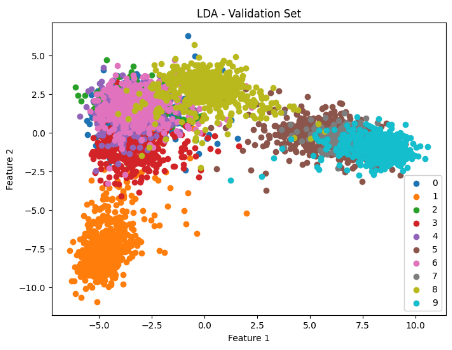
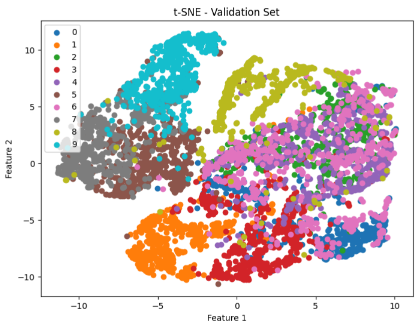


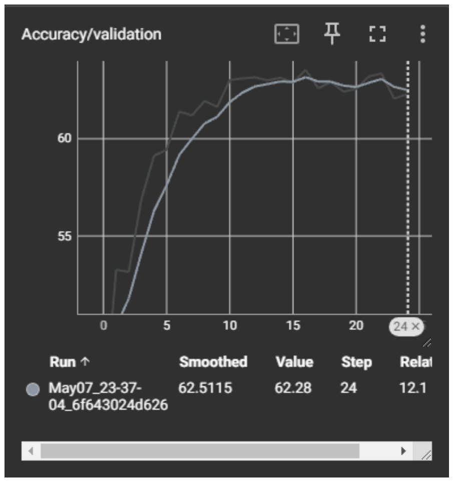
Figure 14. LDA



Hyperparameter Tuning:

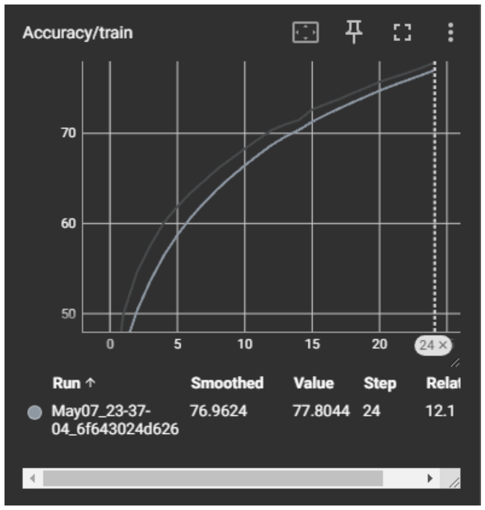
```
LeNet(
  (conv1): Sequential(
    (0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Sequential(
    (0): Linear(in_features=400, out_features=120, bias=True)
    (1): ReLU()
  )
  (fc2): Sequential(
    (0): Linear(in_features=120, out_features=84, bias=True)
    (1): ReLU()
  )
  (classifier): Linear(in_features=84, out_features=10, bias=True)
)
```

CIFAR10 dataset results



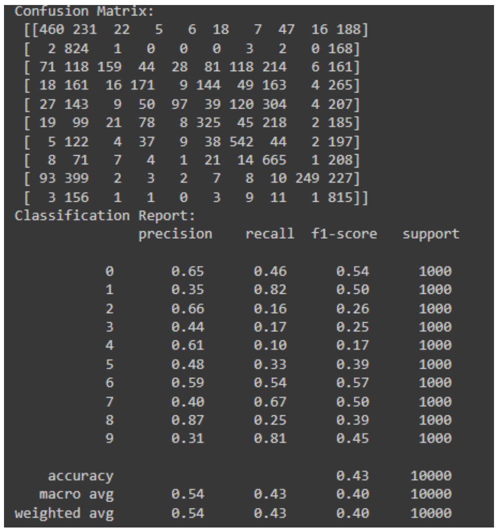
Run ↑	Smoothed	Value	Step	Relat
May07_23-37-04_6f643024d626	62.5115	62.28	24	12.1

Validation Accuracy: 62.28%

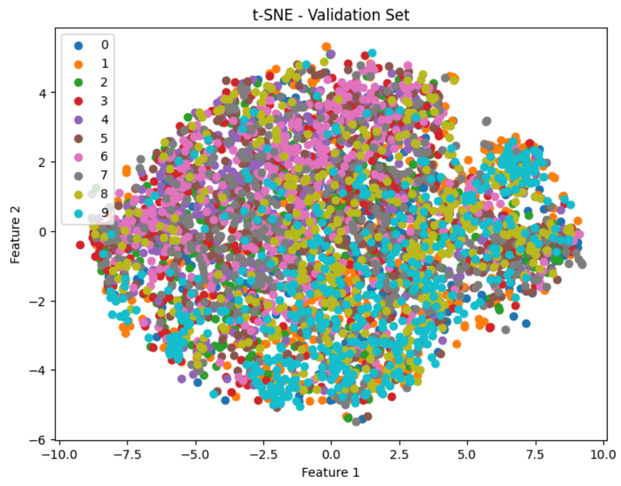


Run ↑	Smoothed	Value	Step	Relat
May07_23-37-04_6f643024d626	76.9624	77.8044	24	12.1

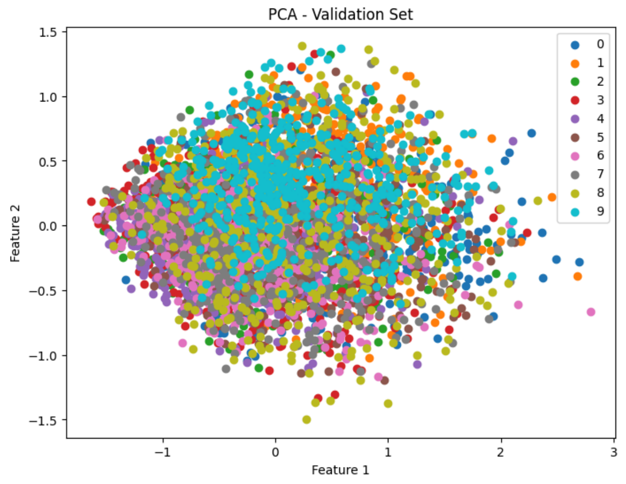
Training Accuracy: 77.80%

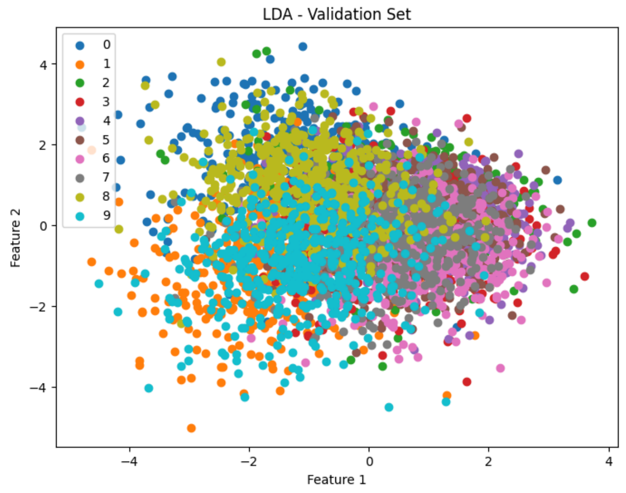


```
Confusion Matrix:
 [[460 231  22   5   6  18   7  47  16 188]
 [  2 824   1   0   0   0   3   2   0 168]
 [ 71 118 159  44  28  81 118 214   6 161]
 [ 18 161  16 171   9 144  49 163   4 265]
 [ 27 143   9  50  97  39 120 304   4 207]
 [ 19  99  21  78   8 325  45 218   2 185]
 [  5 122   4  37   9  38 542  44   2 197]
 [  8  71   7   4   1  21  14 665   1 208]
 [ 93 399   2   3   2   7   8  10 249 227]
 [  3 156   1   1   0   3   9  11   1 815]]
Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.46      0.54      1000
           1       0.35      0.82      0.50      1000
           2       0.66      0.16      0.26      1000
           3       0.44      0.17      0.25      1000
           4       0.61      0.10      0.17      1000
           5       0.48      0.33      0.39      1000
           6       0.59      0.54      0.57      1000
           7       0.40      0.67      0.50      1000
           8       0.87      0.25      0.39      1000
           9       0.31      0.81      0.45      1000

    accuracy                           0.43     10000
   macro avg       0.54      0.43      0.40     10000
weighted avg       0.54      0.43      0.40     10000
```



Testing Accuracy 43%



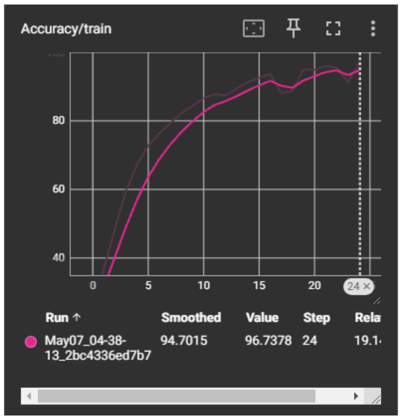


Hyperparameter Tuning:

```
LeNet(
  (conv1): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Sequential(
    (0): Linear(in_features=400, out_features=120, bias=True)
    (1): ReLU()
  )
  (fc2): Sequential(
    (0): Linear(in_features=120, out_features=84, bias=True)
    (1): ReLU()
  )
  (classifier): Linear(in_features=84, out_features=10, bias=True)
)
```

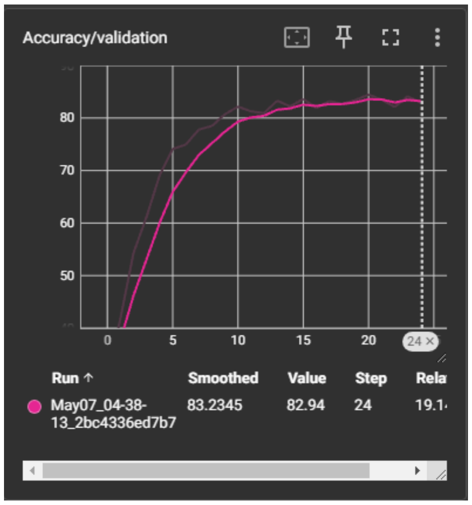
VGG16

CIFAR-10 dataset results



```
Run ↑              Smoothed   Value     Step   Rela
May07_04-38-       94.7015    96.7378   24     19.1
13_2bc4336ed7b7
```

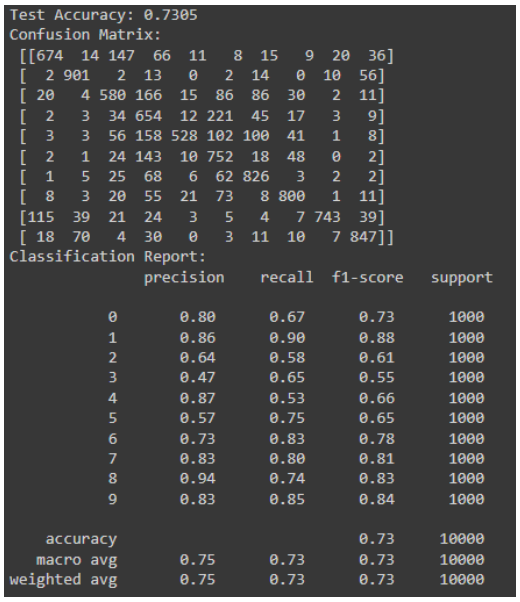
Training Accuracy: 96.74%



Accuracy/validation

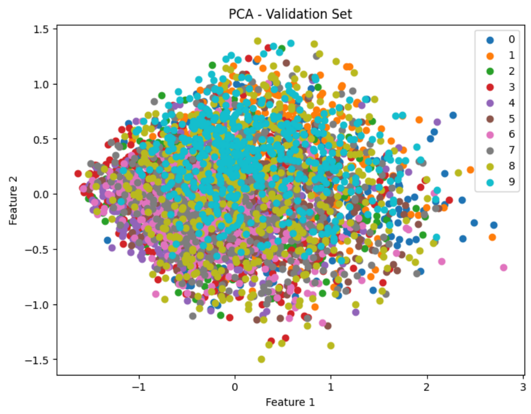
Run ↑	Smoothed	Value	Step	Rela
May07_04-38-13_2bc4336ed7b7	83.2345	82.94	24	19.1

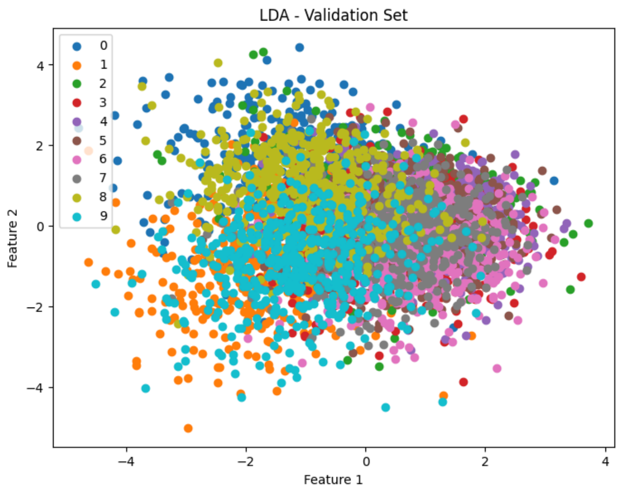
Validation Accuracy: 82.94%

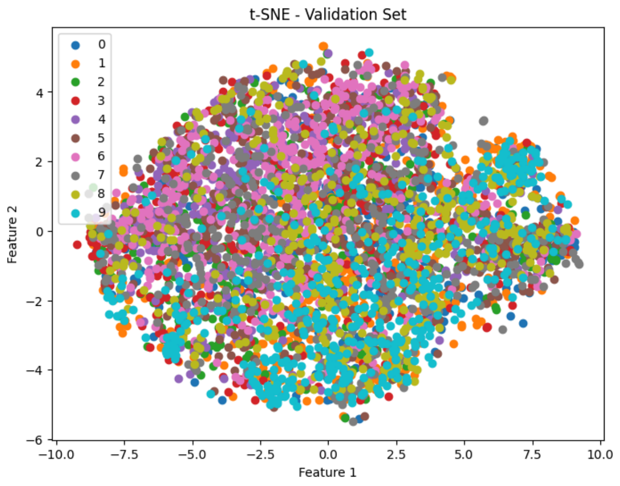


```
Test Accuracy: 0.7305
Confusion Matrix:
 [[674  14 147  66  11   8  15   9  20  36]
 [  2 901   2  13   0   2  14   0  10  56]
 [ 20   4 580 166  15  86  86  30   2  11]
 [  2   3  34 654  12 221  45  17   3   9]
 [  3   3  56 158 528 102 100  41   1   8]
 [  2   1  24 143  10 752  18  48   0   2]
 [  1   5  25  68   6  62 826   3   2   2]
 [  8   3  20  55  21  73   8 800   1  11]
 [115  39  21  24   3   5   4   7 743  39]
 [ 18  70   4  30   0   3  11  10   7 847]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.67      0.73      1000
           1       0.86      0.90      0.88      1000
           2       0.64      0.58      0.61      1000
           3       0.47      0.65      0.55      1000
           4       0.87      0.53      0.66      1000
           5       0.57      0.75      0.65      1000
           6       0.73      0.83      0.78      1000
           7       0.83      0.80      0.81      1000
           8       0.94      0.74      0.83      1000
           9       0.83      0.85      0.84      1000

    accuracy                           0.73     10000
   macro avg       0.75      0.73      0.73     10000
weighted avg       0.75      0.73      0.73     10000
```

Testing Accuracy 73%



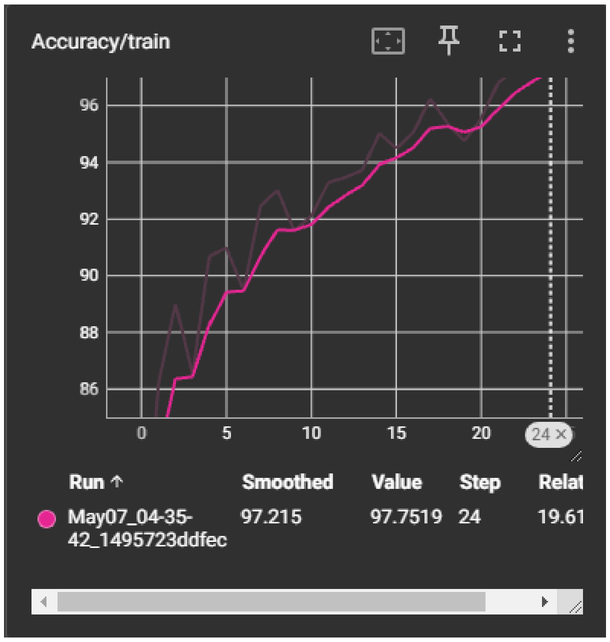




Hyperparameter Tuning:

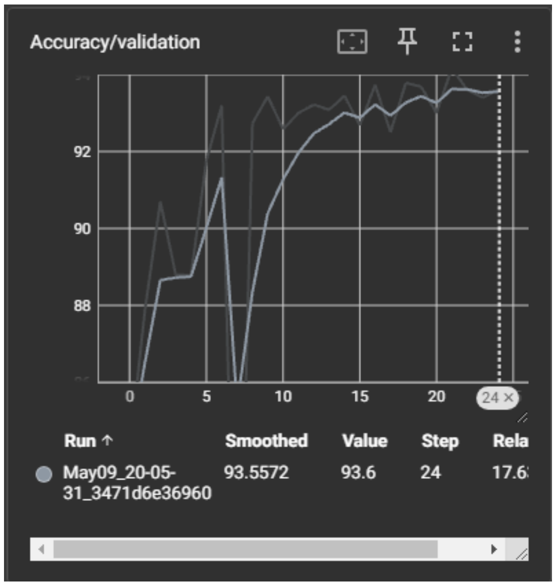
```
VGG16(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (36): ReLU(inplace=True)
    (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (42): ReLU(inplace=True)
    (43): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=512, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

Fashion-MNIST dataset



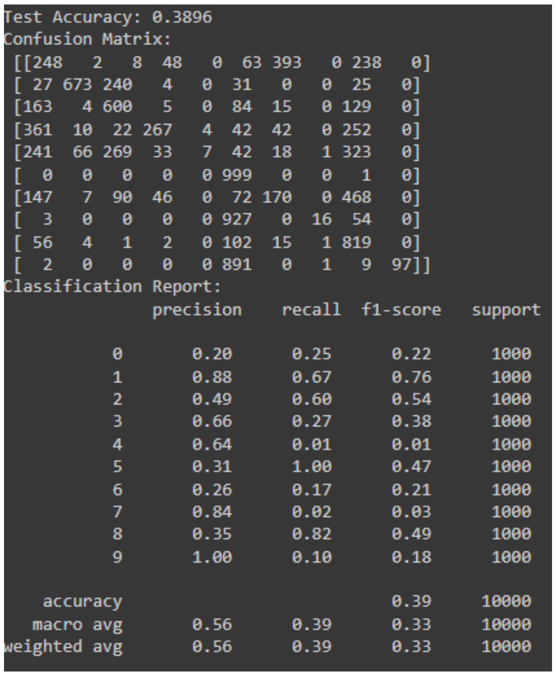
Run ↑	Smoothed	Value	Step	Relat
● May07_04-35-42_1495723ddfec	97.215	97.7519	24	19.61

Training Accuracy: 97.68%



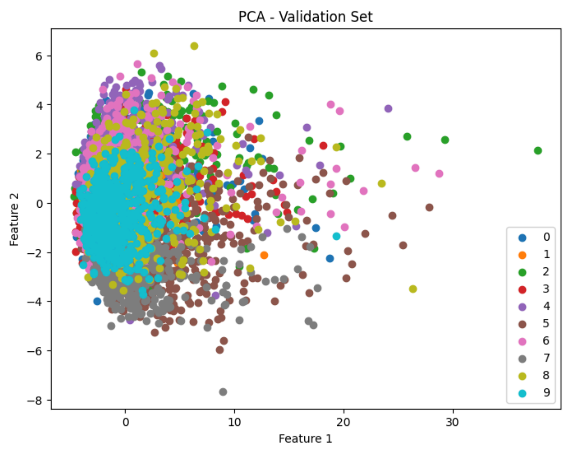
Run ↑	Smoothed	Value	Step	Rela
● May09_20-05-31_3471d6e36960	93.5572	93.6	24	17.6:

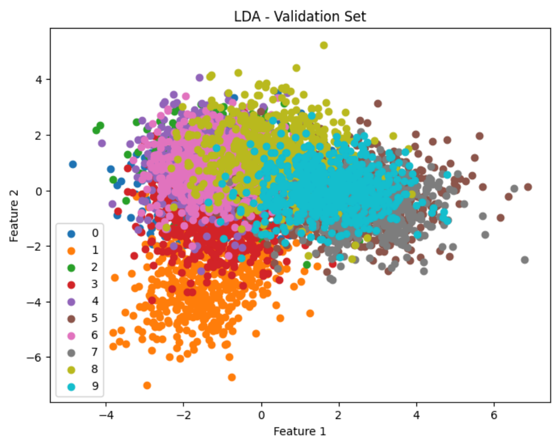
Validation Accuracy: 93.60%

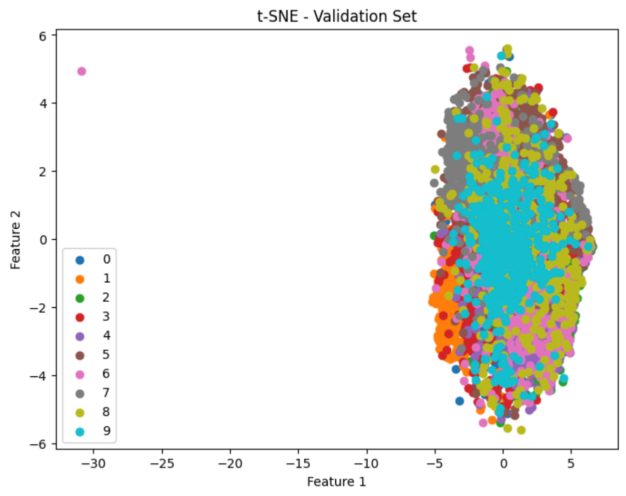


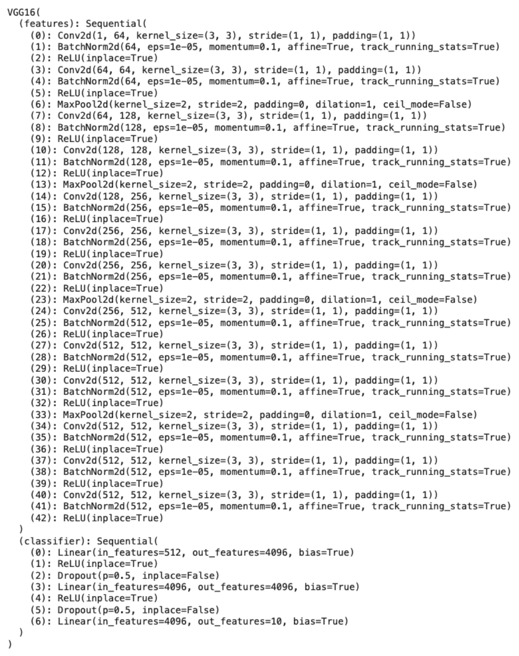
```
Test Accuracy: 0.3896
Confusion Matrix:
 [[248   2   8  48   0  63 393   0 238   0]
 [ 27 673 240   4   0  31   0   0  25   0]
 [163   4 600   5   0  84  15   0 129   0]
 [361  10  22 267   4  42  42   0 252   0]
 [241  66 269  33   7  42  18   1 323   0]
 [  0   0   0   0   0 999   0   0   1   0]
 [147   7  90  46   0  72 170   0 468   0]
 [  3   0   0   0   0 927   0  16  54   0]
 [ 56   4   1   2   0 102  15   1 819   0]
 [  2   0   0   0   0 891   0   1   9  97]]
Classification Report:
              precision    recall  f1-score   support

           0       0.20      0.25      0.22      1000
           1       0.88      0.67      0.76      1000
           2       0.49      0.60      0.54      1000
           3       0.66      0.27      0.38      1000
           4       0.64      0.01      0.01      1000
           5       0.31      1.00      0.47      1000
           6       0.26      0.17      0.21      1000
           7       0.84      0.02      0.03      1000
           8       0.35      0.82      0.49      1000
           9       1.00      0.10      0.18      1000

    accuracy                           0.39     10000
   macro avg       0.56      0.39      0.33     10000
weighted avg       0.56      0.39      0.33     10000
```

Testing Accuracy 38%



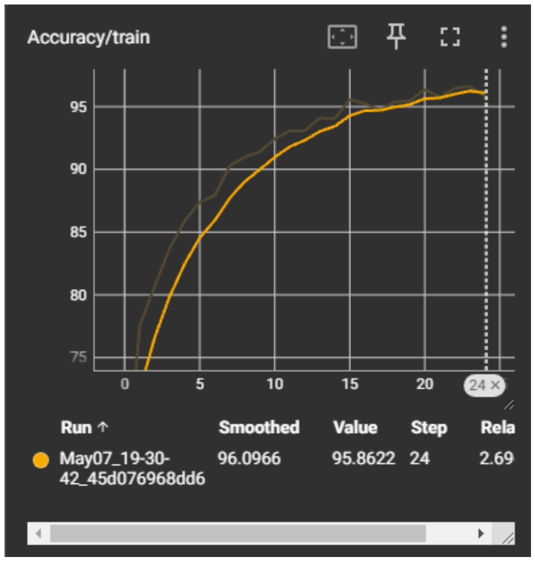





```
VGG16(
  (features): Sequential(
    (0): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace=True)
    (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (19): ReLU(inplace=True)
    (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (26): ReLU(inplace=True)
    (27): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (28): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (32): ReLU(inplace=True)
    (33): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (36): ReLU(inplace=True)
    (37): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (39): ReLU(inplace=True)
    (40): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (41): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (42): ReLU(inplace=True)
  )
  (classifier): Sequential(
    (0): Linear(in_features=512, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

Modified EfficientNet

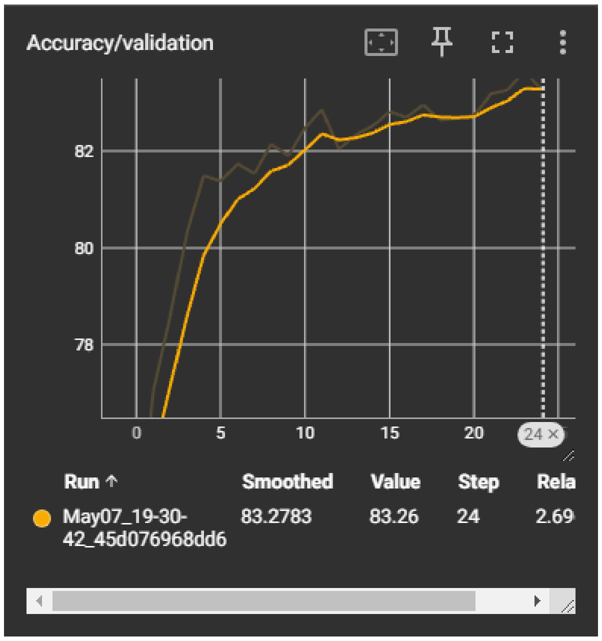
CIFAR10 dataset



Run ↑	Smoothed	Value	Step	Rela
May07_19-30-42_45d076968dd6	96.0966	95.8622	24	2.69

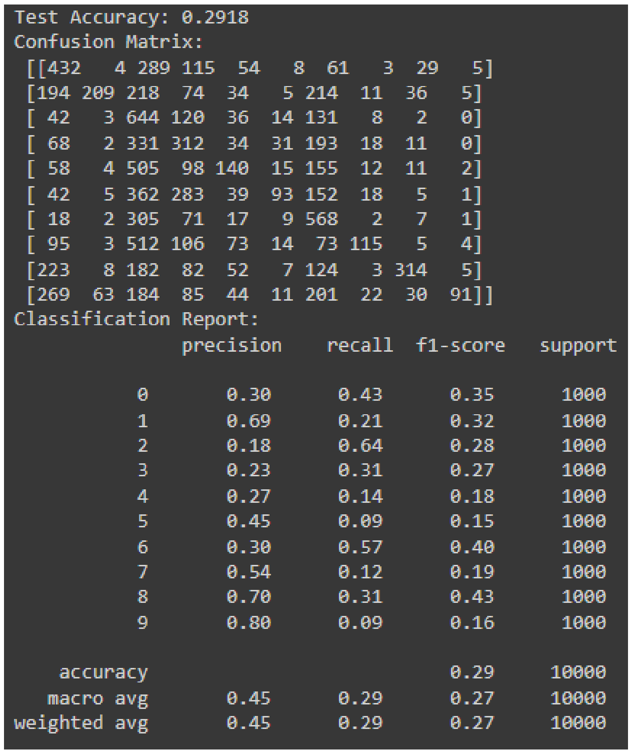
Training Accuracy: 95.86%

Hyperparameter Tuning:



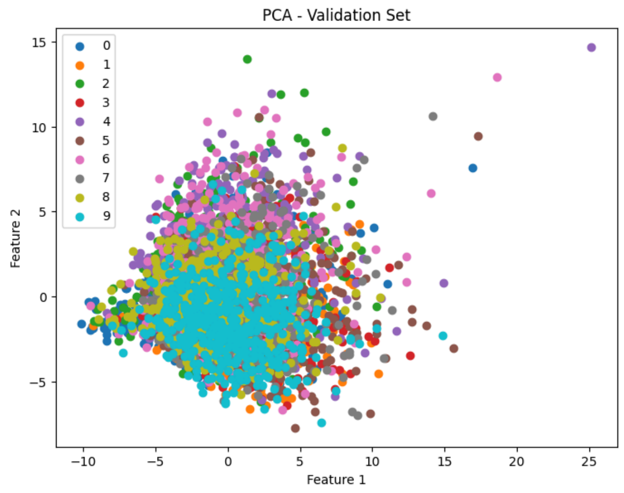
Run ↑	Smoothed	Value	Step	Rela
● May07_19-30-42_45d076968dd6	83.2783	83.26	24	2.69

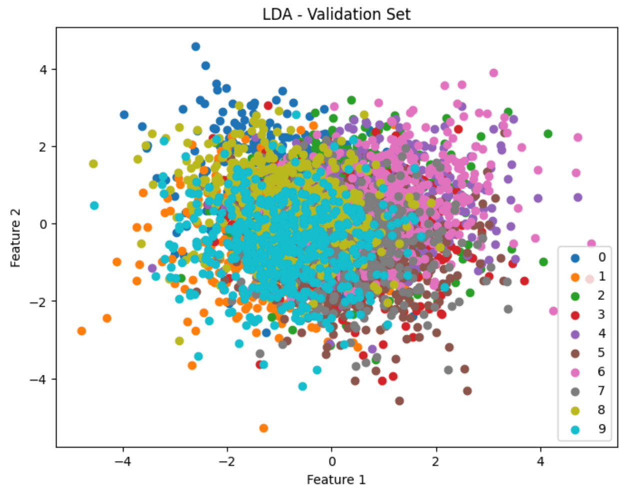
Validation Accuracy: 83.26%

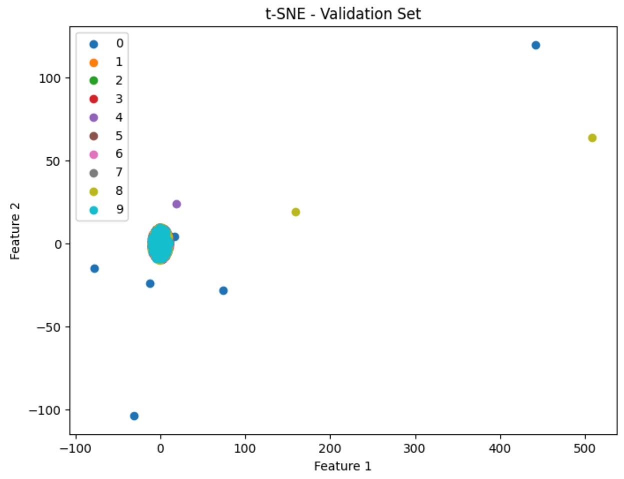


```
Test Accuracy: 0.2918
Confusion Matrix:
 [[432   4 289 115  54   8  61   3  29   5]
 [194 209 218  74  34   5 214  11  36   5]
 [ 42   3 644 120  36  14 131   8   2   0]
 [ 68   2 331 312  34  31 193  18  11   0]
 [ 58   4 505  98 140  15 155  12  11   2]
 [ 42   5 362 283  39  93 152  18   5   1]
 [ 18   2 305  71  17   9 568   2   7   1]
 [ 95   3 512 106  73  14  73 115   5   4]
 [223   8 182  82  52   7 124   3 314   5]
 [269  63 184  85  44  11 201  22  30  91]]
Classification Report:
              precision    recall  f1-score   support

           0       0.30      0.43      0.35      1000
           1       0.69      0.21      0.32      1000
           2       0.18      0.64      0.28      1000
           3       0.23      0.31      0.27      1000
           4       0.27      0.14      0.18      1000
           5       0.45      0.09      0.15      1000
           6       0.30      0.57      0.40      1000
           7       0.54      0.12      0.19      1000
           8       0.70      0.31      0.43      1000
           9       0.80      0.09      0.16      1000

    accuracy                           0.29     10000
   macro avg       0.45      0.29      0.27     10000
weighted avg       0.45      0.29      0.27     10000
```

Testing Accuracy 29%

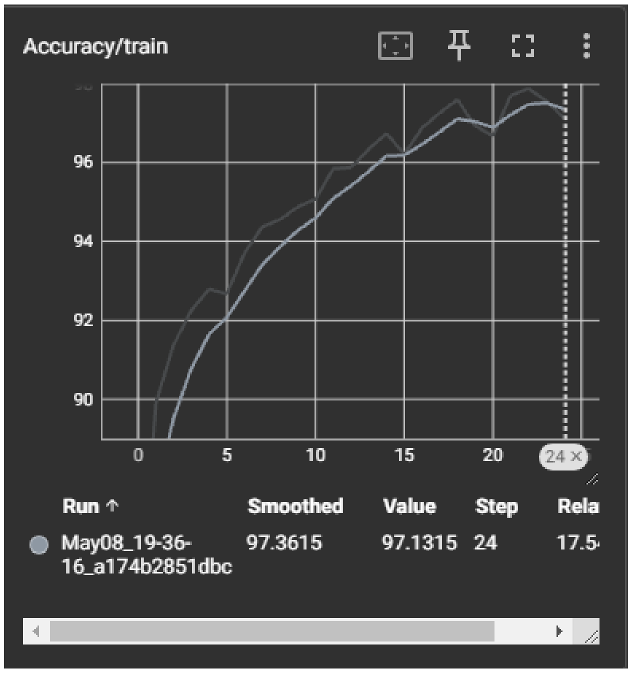






Hyperparameter Tuning:

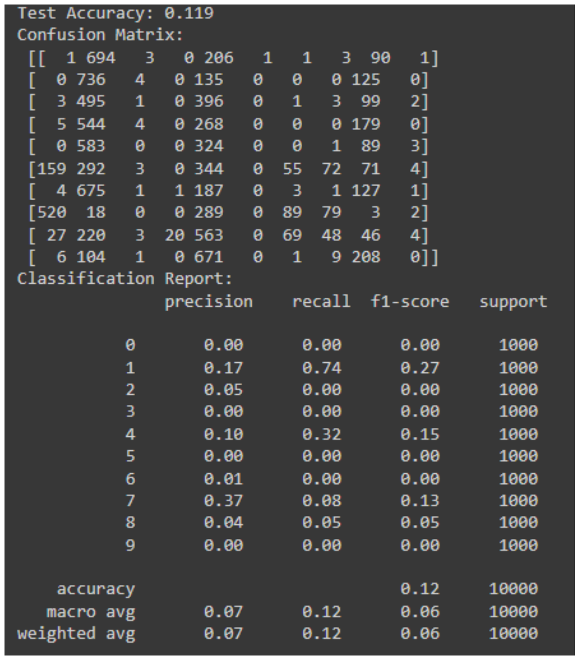
```
EfficientNet(
  (features): Sequential(
    (0): Conv2dNormActivation(
      (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (2): Conv2dNormActivation(
            (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.0, mode=row)
      )
    )
```



Run ↑	Smoothed	Value	Step	Rela
May08_19-36-16_a174b2851dbc	97.3615	97.1315	24	17.5

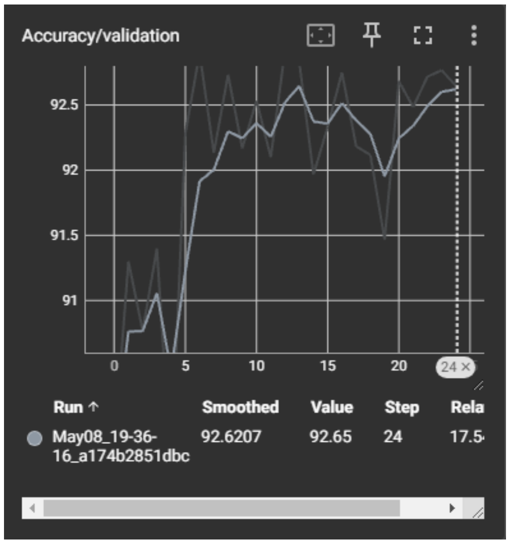
Training Accuracy: 97.13%

Fashion-MNIST dataset



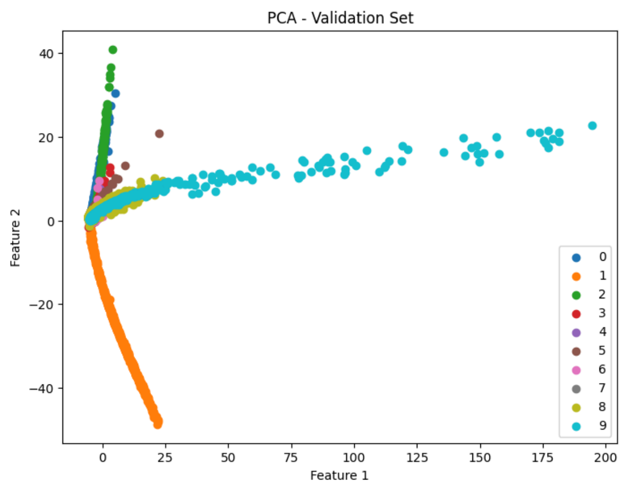
```
Test Accuracy: 0.119
Confusion Matrix:
 [[  1 694   3   0 206   1   1   3  90   1]
 [  0 736   4   0 135   0   0   0 125   0]
 [  3 495   1   0 396   0   1   3  99   2]
 [  5 544   4   0 268   0   0   0 179   0]
 [  0 583   0   0 324   0   0   1  89   3]
 [159 292   3   0 344   0  55  72  71   4]
 [  4 675   1   1 187   0   3   1 127   1]
 [520  18   0   0 289   0  89  79   3   2]
 [ 27 220   3  20 563   0  69  48  46   4]
 [  6 104   1   0 671   0   1   9 208   0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1000
           1       0.17      0.74      0.27      1000
           2       0.05      0.00      0.00      1000
           3       0.00      0.00      0.00      1000
           4       0.10      0.32      0.15      1000
           5       0.00      0.00      0.00      1000
           6       0.01      0.00      0.00      1000
           7       0.37      0.08      0.13      1000
           8       0.04      0.05      0.05      1000
           9       0.00      0.00      0.00      1000

    accuracy                           0.12     10000
   macro avg       0.07      0.12      0.06     10000
weighted avg       0.07      0.12      0.06     10000
```

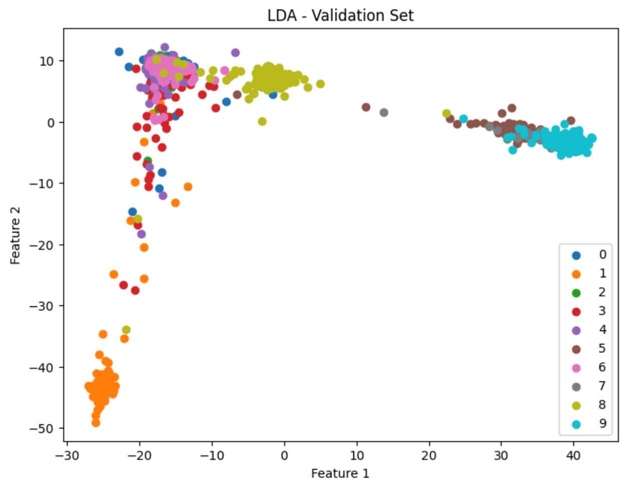
Testing Accuracy:12%

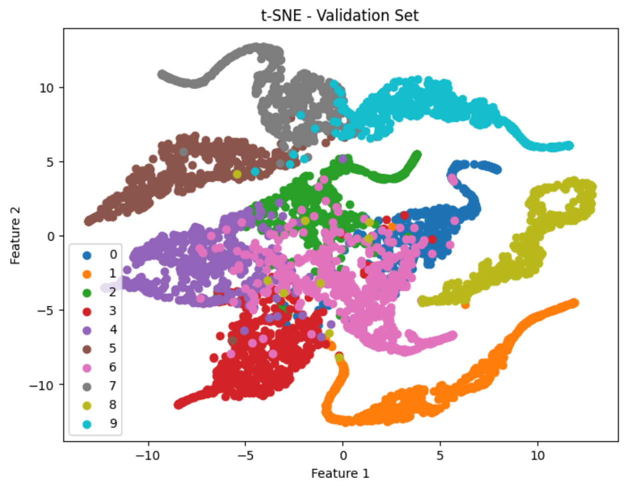


Run ↑	Smoothed	Value	Step	Rela
May08_19-36-16_a174b2851dbc	92.6207	92.65	24	17.5

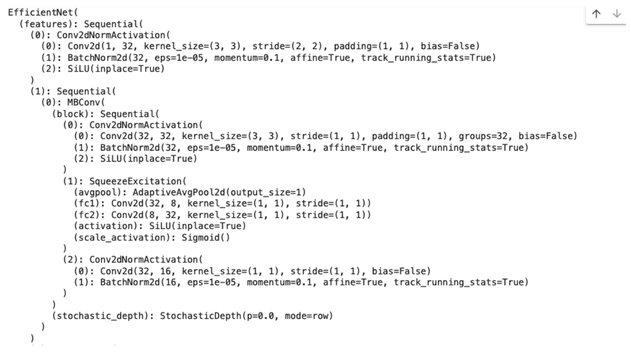
Validation Accuracy: 92.65%







Hyperparameter Tuning:



```
EfficientNet(
  (features): Sequential(
    (0): Conv2dNormActivation(
      (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): SiLU(inplace=True)
    )
    (1): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32, bias=False)
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (2): Conv2dNormActivation(
            (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.0, mode=row)
      )
    )
```

After comparing all the statistics, we see that LeNet was the fastest overall to produce results. This is because it had less layers to go through. Based on how VGG16 and the modified EfficientNet are modeled , it will take a lot more time to produce results. We also see that on the more complicated datasets, the LeNet performs slightly worse. The modified EfficientNet had the best accuracy for the validation set and training set but performed a lot worse for the test set. This could be because of the overfitting of the model that we could see in the t-sne Cifar-10, overlapping data between training data and validation data, or overfitting to the validation set. If we were to do the project again we would re-examine the model's complexity based on the amount of data and keep an eye on performance metrics during training for both the test sets. The CPU was used for VGG16 and LeNet while the GPU was used for the modified EfficientNet. Based on the model the best test accuracy was VGG-16 which means that model did the best based on the data provided which means its more reliable. The datasets provided show that Cifar-10 was much more difficult to train probably because it had to go through 3 RGB channels and it had more complex pixelization representation. Efficient net c does not do well with small to medium-sized datasets which would lead to EfficientNet-B0 not capturing enough complexity from the data, and it is less stable as B0 is just the base model, while B7 model is more balanced to approach the CNN optimization.

If you wanted faster results than choose LeNet because it has less complex features since it only has 2 convolutional layer, 2 pooling layer, and 3 fc-layers while VGG16 has 13 convolutional layers and 3 fully connected layers with more filters than LeNet such as having (3x3) convolutional padding. For EfficientNet, the B0 model uses a technique called compound coefficient to scale the model based on width, depth, or resolution. The reason EffecientNet is modified is because we had to change the model to accept the input channels, and copy the weight of the original models first layer, and then modify it to have the 10 classes (Fine-tuning the model).

What made VGG16 so effective is the increase of layers when using a smaller kernel, which saw an increase in non-linearity. The loss curve shown in the code shows that the loss decreases as the model moves forward. This is due to VGG bringing an improvement in accuracy. The repeated blocks of convolutional layers allows for a robust feature extractor and can be effective in the scenarios where the task benefits deep stacks ina hierarchical format. Also, the deep connectivity between layers can lead to better integration over the entire network which is advantageous.

References

[1]C. Ivan, “Convolutional Neural Networks on Randomized Data.” Accessed: Dec. 14, 2019. [Online]. Available: https://arxiv.org/pdf/1907.10935

[2]Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998, doi: https://doi.org/10.1109/5.726791.

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

[4]M. Tan and Q. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” Sep. 2020. Available: https://arxiv.org/pdf/1905.11946

[5]P. P. Aug 9 and Read 2023 6 M., “What is EfficientNet? The Ultimate Guide.,” *Roboflow Blog*, Aug. 09, 2023. https://blog.roboflow.com/what-is-efficientnet/

[6]“VGG16 - Convolutional Network for Classification and Detection,” *Neurohive.io*, Nov. 21, 2018. https://neurohive.io/en/popular-networks/vgg16/

[7]“What is EfficientNet?,” *skyengine.ai*. https://skyengine.ai/se/skyengine-blog/121-what-is-efficientnet