Deep-Learning in Computer Vision: From CNN to Other Models We are Trying to Understand

Qianxun Xu; Zesheng Liu

December 31, 2022

Abstract

Image classification is an attractive topic in computer vision. There are several methods to do it. Some modifications may also affect the model's performance. We start with a simple 3-Conv-layer CNN model for CIFAR-10 dataset, and discuss how some data augmentation methods and model modifications will affect the final accuracy. We find that these changes could all avoid overfitting problem in our baseline model. As CIFAR-10 is a small dataset, we do not get significant improvement by simply applying data augmentation. We can make an improvement of about 7% with model modifications together with some data augmentation. Then, we discuss the most famous VGG16 model in image classification and test its performance on CIFAR-10 and CIFAR-100. We get a reasonable accuracy: 88.9% for CIFAR-10 and 63.5% for CIFAR-100. Last but not the least, we go through two state-of-the-art model, discuss about their architectures, number of parameters, and see how these models work on CIFAR-10 dataset.

Contents

1 Introduction

This project is our exploration of deep neural network models in the realm of image classification tasks. First inspired by a homework problem, our goal is to build and compare neural network models using various techniques and eventually aim to achieve $85\% - 90\%$ accuracy. The first part of the project will be mainly about trying to tackle the overfitting issue with data augmentation and some model modification techniques. In the second part of the project, we will try to understand the current state-of-the-art models for image classification tasks, including VGG16, MLP mixer, and vision transformer. We will reconstruct the architectures based on publications, and aim to replicate the results.

2 Dataset

The dataset we used through out this project was the CIFAR-10 dataset collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton[1]. The CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck), with 6,000 images per class, in which 50,000 images belong to the training set, and 10,000 images belong to the testing set.

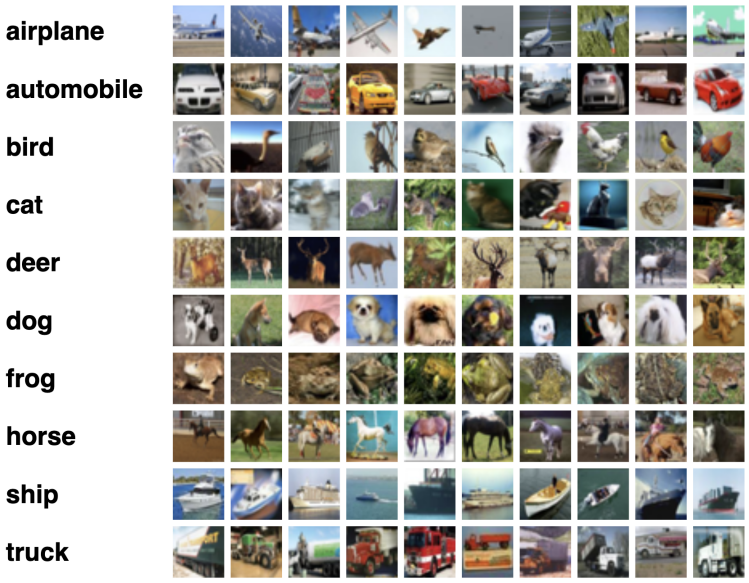


Figure 1: Random sample images from 10 classes

In addition to training and testing the VGG16 on the CIFAR-10 dataset, in order to explore whether VGG16 can be generalized well onto other dataset, we also trained and tested the model on the CIFAR-100 dataset [1]. The CIFAR-100 dataset contains 100 classes and 600 images per class, so resulting in the same amount of images as CIFAR-10. Similarly, there are 50,000 images in the training set, and 10,000 images in the testing set.

3 Baseline CNN Model

3.1 Baseline model architecture

The baseline model we build to run the original and augmented CIFAR-10 dataset, and later to modify and compare with other models is a convolutional neural network (CNN). The CNN is a type of deep learning neural networks that has been used for visual imagery related tasks such as image classification and facial recognition. It is a preferred method for image classification tasks as what we have in this project than other neural network models such as multi-layer perceptron (MLP) since the number of trainable parameters can increase drastically with an increase in the size of the image in a MLP model. In addition, CNN captures the spatial features of an image, which MLP fails to do so.

Our model accepts images as inputs, where each image has size $32 \times 32 \times 3$. Then followed by the convolution layer that consists 6 filters, each with size $5 \times 5 \times 3$ and stride size 1. The convolution layer extracts tiles of the input feature map, applies filters to compute new features, and produces an output feature map. During training, the CNN gradually learns the optimal weights for the filters so that the output feature maps contain useful information. Next is the activation layer which is the rectified linear unit (ReLU) function. The ReLU function, $f(x) = \max(0, x)$, brings nonlinearity into the model, which is also a popular activation function in many other neural networks. Following the activation layer is the pooling layer, where we use 2×2 max pooling with stride 2. The pooling step downsamples the extracted features which reduces the dimensions of feature maps while preserving important feature information. After three convolution modules (convolution, ReLU, pooling), we have three fully connected layers. At the end of the third fully connected layer is a softmax activation function that outputs class probabilities of the ten labels.

The complete baseline CNN architecture is the following:

- Input layer;
- Convolution layer with 6 filters of size $5 \times 5 \times 3$, stride 1;
- ReLU layer;
- Max pooling layer with pooling size 2×2, stride 2;
- Convolution layer with 12 filters of size $5 \times 5 \times 6$, stride 1;
- ReLU layer;
- Max pooling layer with pooling size 2×2, stride 2;
- Convolution layer with 24 filters of size $5 \times 5 \times 24$, stride 1;
- ReLU layer;

- Max pooling layer with pooling size 2×2, stride 2;
- Fully connected layer of output dimension 120;
- ReLU layer;
- Fully connected layer of output dimension 84;
- ReLU layer;
- Fully connected layer of output dimension 10;
- Softmax.

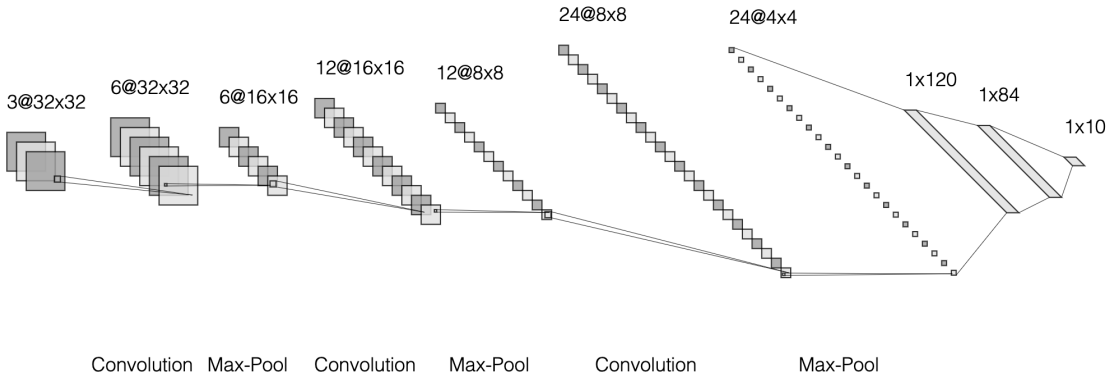
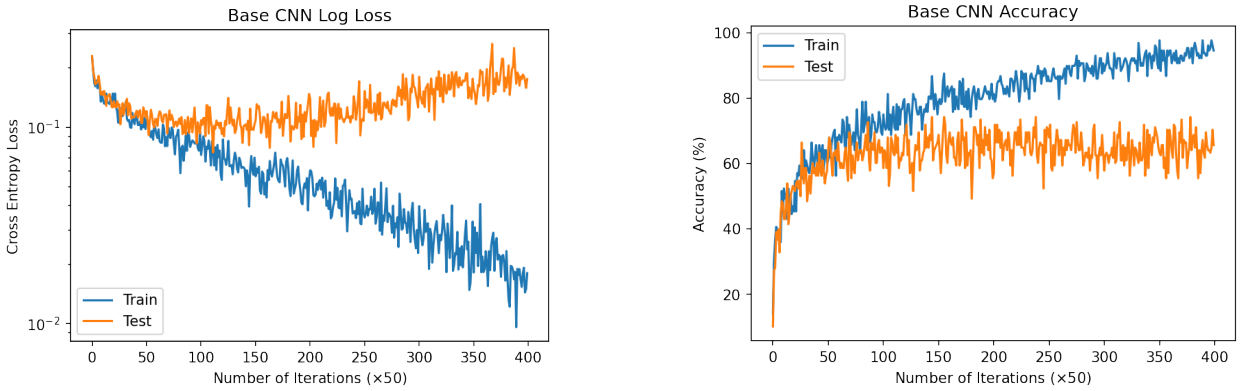


Figure 2: CNN architecture illustration

3.2 Baseline model results



(a) Baseline model loss

(b) Baseline model accuracy

Figure 3: Baseline model results

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline	CIFAR-10	2:40	66706	0.01805	0.945312	0.1752	0.6411

Table 1: Baseline model results

Above (Fig.3) are the results of the baseline CNN model. As we can observe, with 20,000 iterations, the training loss continues to decrease till around 0.018 and training accuracy reaches 94.5%. However, if we look at the performance on testing dataset, we can see the testing loss stops decreasing around 3,700 iterations then it starts to increase, which indicates the model starts to overfit to the training data and the model is poorly generalized to the testing data. Overfitting is a common issue in deep neural network models since with huge amount of parameters, models are capable of fitting really well to the training dataset. When a model overfits, it can recognize images from the training set correctly but fails to generalize patterns. In the following sections, we will try couple methods, such as data augmentation and regularization, to combat the overfitting issue. Our goal is to take measures such that the model can be generalized better to testing set and thus see an increase in the testing accuracy.

4 Data Augmentation

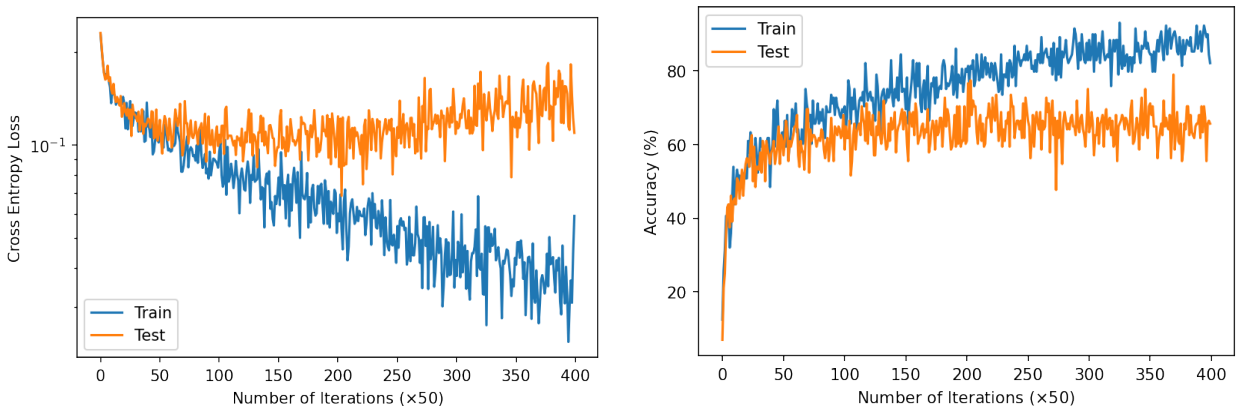
Increasing the dataset size by collecting more data is the first option if it is possible. However, collecting more data is usually extremely time and labor consuming, and thus can be very expensive. Hence, the first approach we tried was to increase the training dataset size by data augmentation through adding color filter, changing image brightness, and mirroring. We proceeded by altering the training set, trained the model with the augmented dataset, then made classifications on the unaltered testing set. It should be kept in mind that data augmentation may not always improve the results. For example, if we randomly scales images and makes them too large to extract any meaningful features, then the model would not learn better on these kind of images. Another situation that data augmentation could become a concern is rotation. For instance, after rotating 180 degrees, the number 6 becomes the number 9. Thus, the label is changed, which is an undesirable aftermath of some data augmentation methods. The methods we used have taken these factors into account and tried to avoid them.

4.1 Grayscale

(a) Original image (b) Grayscale image

Figure 4: Changing the original images to grayscale

The first data augmentation method we tried was adding the black-and-white version of images to the training dataset. Changing the color of the images was done by changing the saturation to 0 using the PIX, which is an image processing library in JAX. The model we used on this dataset was the baseline CNN model.



(a) Loss (b) Accuracy

Figure 5: Baseline model with data augmentation (Grayscale) results

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline	CIFAR-10, grayscale	2:44	66706	0.05897	0.820312	0.109432	0.6423999

Table 2: Baseline model with data augmentation (grayscale) results

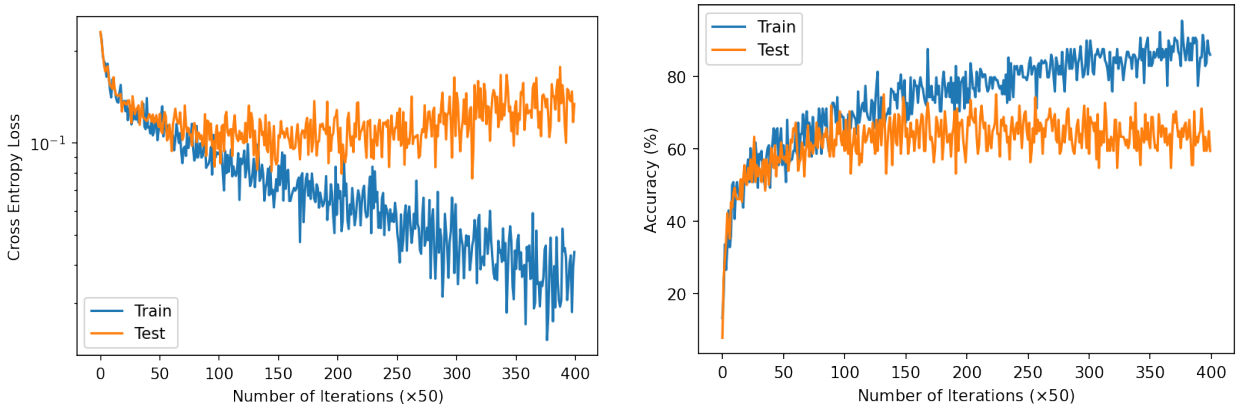
Observing above results, although the turning point of the testing loss is delayed and slope is slightly smoother compared to the first result (Fig.3a), we can still see it stops decreasing and starts to increase after around 5,000 iterations. Even though the final testing accuracy remains around 64%, the slight changes in the testing loss indicate adding grayscale images to the training set can soothe the overfitting problem but not solving it by any means.

4.2 Brightness

(a) Original image (b) 75% brightness (c) 50% brightness (d) 25% brightness

Figure 6: Changing brightness of the original image to 75%, 50%, and 25%

The second data augmentation method we tried was adjusting the brightness of images. The intuition was that the labels should remain the same even when the pictures were taken under different lighting conditions. First, we changed the brightness with PIX to 25%, 50%, and 75% of the original images (Fig.6). But we soon observed that the model performance was noticeably worse than the baseline model. We think it was because when the images were at 25% and 50% brightness (Fig.6c, Fig.6d), there was simply not enough contrast remain in the picture for the model to extract features, or too much noise was added to the images such that the original useful features were destroyed. So the model performed worse than without data augmentation. Therefore, we run the model again with the original and the set of images at 75% brightness (Fig.6b). Here are the results.



(a) Loss (b) Accuracy

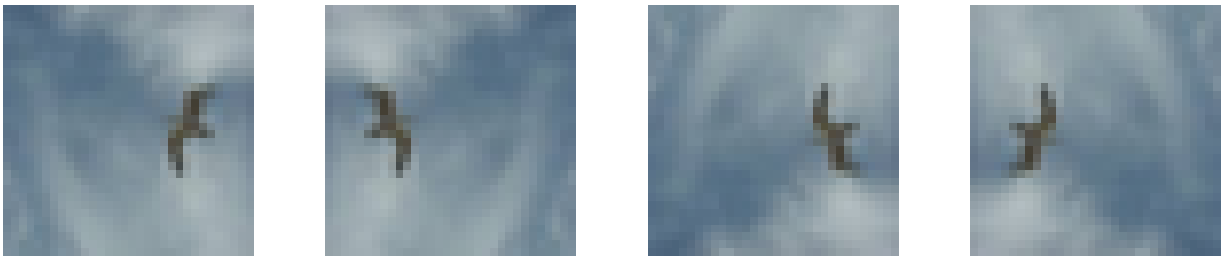
Figure 7: Baseline model with data augmentation (75% brightness) results

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline	CIFAR-10, 75% brightness	3:33	66706	0.04403	0.859375	0.1339	0.6374

Table 3: Baseline model with data augmentation (75% brightness) results

Similar to the results of the baseline model trained with the original and grayscale images, training with different brightness can delay the overfitting but cannot conquer the issue. Testing accuracy is around 63.7%, a slight decrease compared to the baseline result. Changing the brightness has not improved the results so other data augmentation method should be explored.

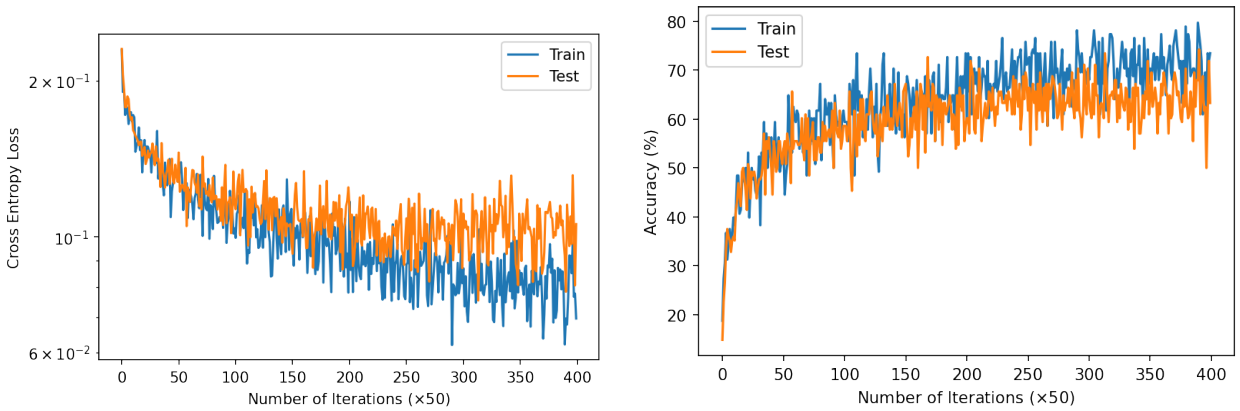
4.3 Mirroring



(a) Original image (b) Horizontal flip (c) Vertical flip (d) Transpose

Figure 8: Flipping the original image horizontally, vertically, and taking the transpose

The third and last data augmentation method we tried was mirroring. The intuition was that the labels of images should remain the same regardless objects' orientations. With rotation, at degrees other than multiples of 90, background noise is added to the pictures. This could be a problem since the network may learn meaningless features from the new background and cause the model to perform worse. With this consideration, we used mirroring techniques that would not introduce new background noise, namely, flipping images horizontally, vertically, and transposes of the images (Fig.8).



(a) Loss (b) Accuracy

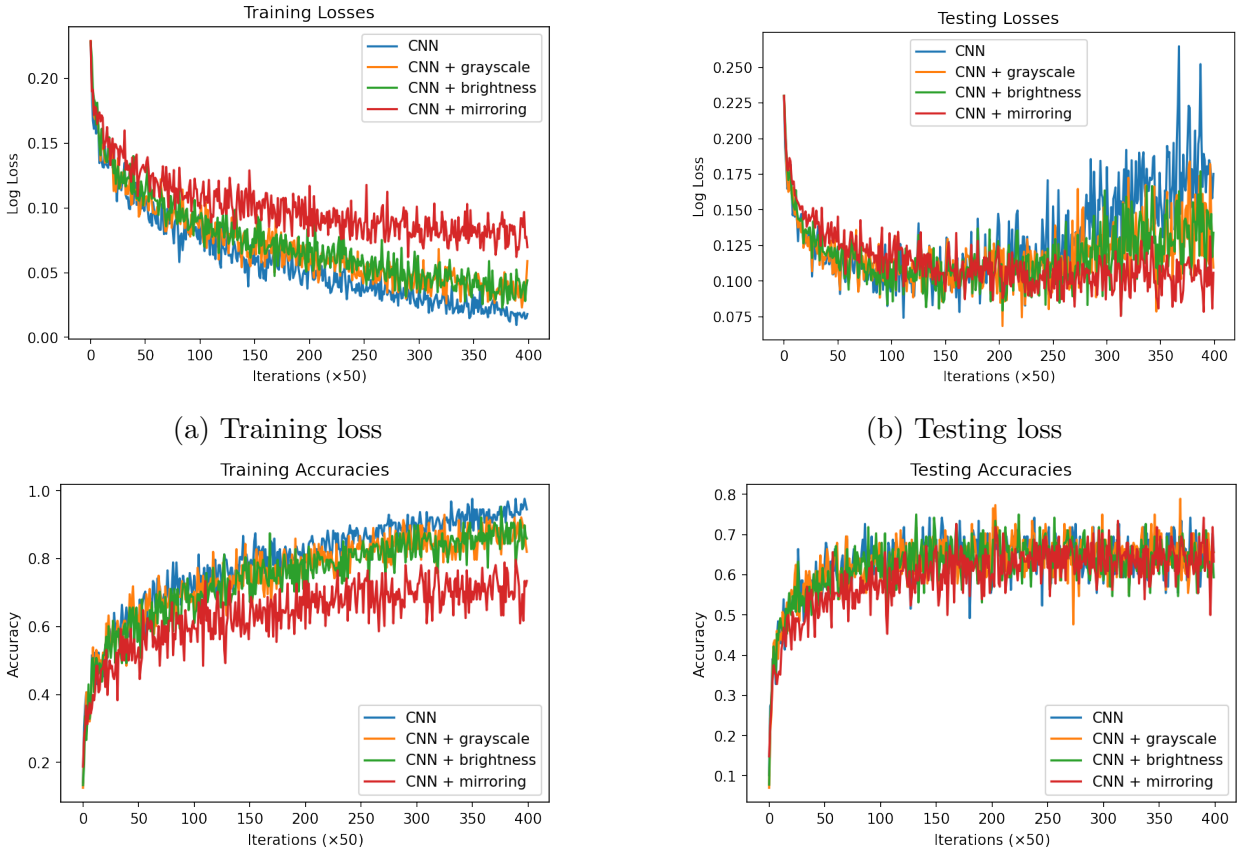
Figure 9: Baseline model with data augmentation (mirroring) results

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline	CIFAR-10, flipped images	3:29	66706	0.06967	0.734375	0.1058	0.6458

Table 4: Baseline model with data augmentation (flipping images) results

The results show a significant improvement from the baseline model's. The testing loss continues to decrease with the training loss and does not show an obvious lift even after 20,000 iterations. This proves that with the flipped and transposed images, our model does not overfit easily to the training dataset, while remain a testing accuracy around 64%. To conclude, among the three data augmentation methods we have tried, the mirroring technique produced the unarguable the best outcomes.

4.4 Summary



(a) Training loss

(b) Testing loss

(c) Training accuracy

(d) Testing accuracy

Figure 10: Baseline model with different types of data augmentation methods results

When we plot the loss and accuracy curves obtained by training on different datasets (Fig.10), it is clear that while the baseline model reaches the lowest training loss, its testing loss is also the highest. Among the three data augmentation methods we have experimented

with, the model trained with flipped images outperforms the others by showing a significant improvement in conquering the overfitting problem while maintaining a similar level of testing accuracy.

5 Model Modification

Training deep neural networks is challenging as the process can be slow and prone to overfit, especially on relatively small datasets. In addition to data augmentation, another approach we took to tackle the overfitting issue was to modify the CNN architecture with batch normalization and dropout layers. In this section, we will explain each method and display the results. In the last part, we will combine batch normalization, dropout, and data augmentation (mirroring technique) to produce the best CNN-based model we built for this project.

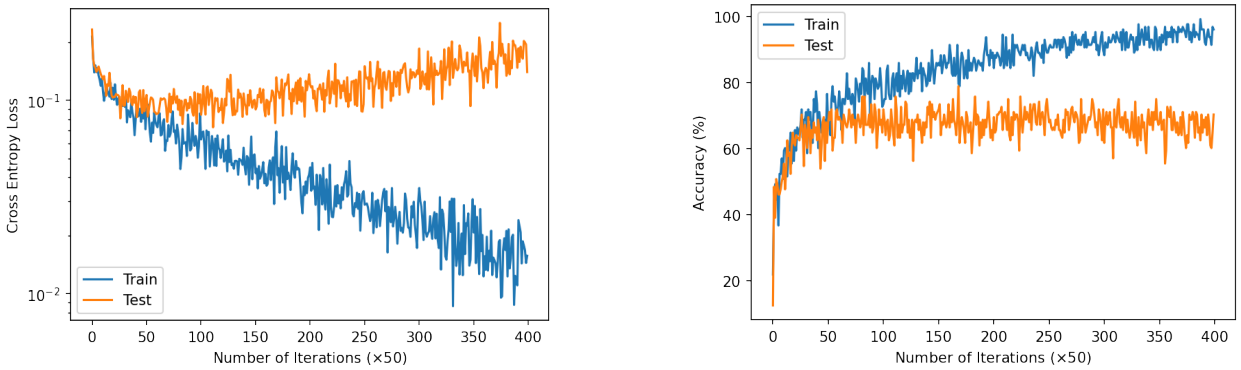
5.1 Batch Normalization

Batch normalization is a technique used often for training deep neural networks. It normalizes the inputs to zero mean and unit variance prior to the activation layer [2], which is also called "whitening" when used in the computer vision setting [3]. The normalization process assumes the spread and distribution of inputs in the subsequent layer do not change drastically while the weights being updated [2]. Another effect of normalization is more efficient model training process [4]. So we expect to see our model to converge faster with batch normalization layers. To summarize, batch normalization accelerates and stabilizes training, and reduces generalization error.

The complete CNN architecture with batch normalization is as follows:

- Input layer;
- Convolution layer with 6 filters of size $5 \times 5 \times 3$, stride 1;
- Batch normalization layer;
- ReLU layer;
- Max pooling layer with pooling size 2×2, stride 2;
- Convolution layer with 12 filters of size $5 \times 5 \times 6$, stride 1;
- Batch normalization layer;
- ReLU layer;
- Max pooling layer with pooling size 2×2, stride 2;
- Convolution layer with 24 filters of size $5 \times 5 \times 24$, stride 1;

- Batch normalization layer;
- ReLU layer;
- Max pooling layer with pooling size 2×2, stride 2;
- Fully connected layer of output dimension 120;
- ReLU layer;
- Fully connected layer of output dimension 84;
- ReLU layer;
- Fully connected layer of output dimension 10;
- Softmax.



(a) Loss (b) Accuracy

Figure 11: Baseline model with batch normalization results

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Batchnorm model	CIFAR-10	2:54	66790	0.01572	0.960937	0.14026	0.673

Table 5: Baseline model with batch normalization layer results

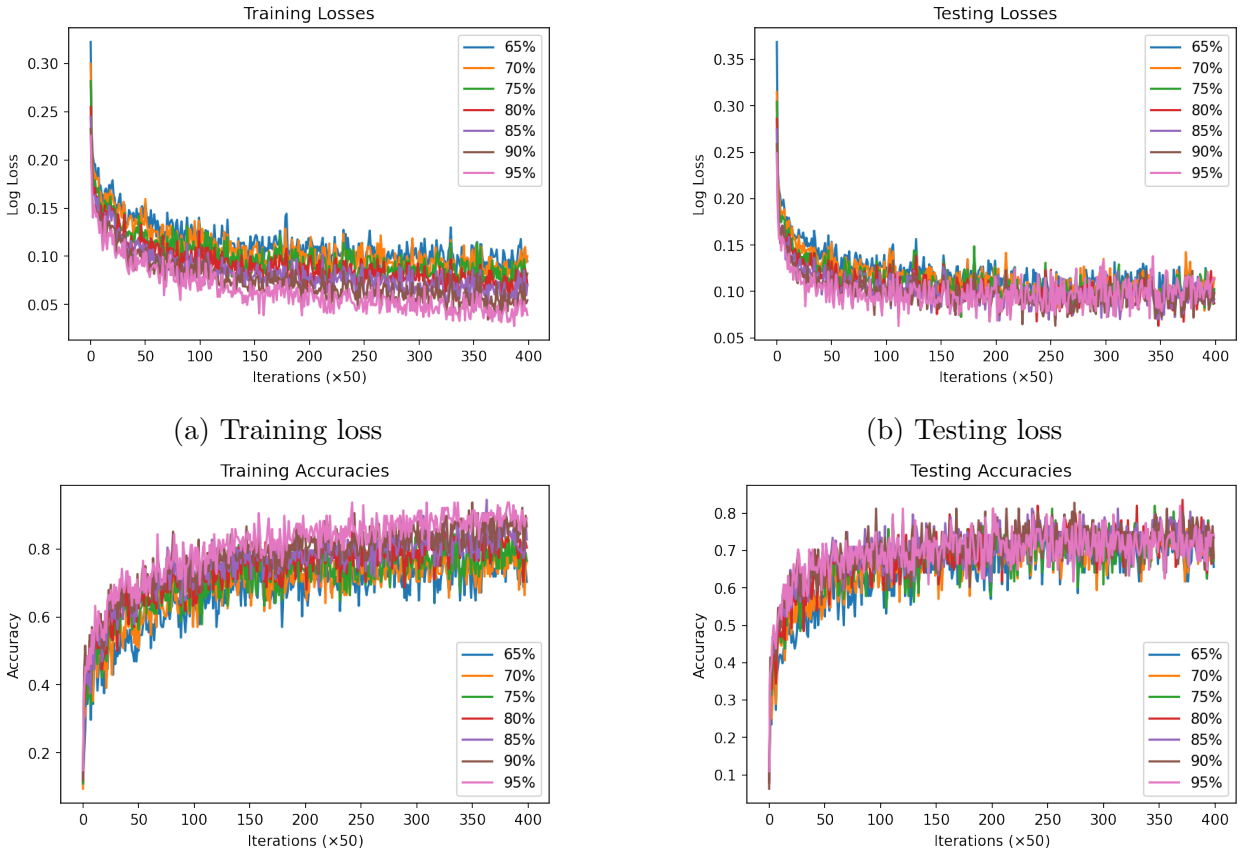
Looking at the loss chart, we observe the testing loss does converge faster at around 2,000 iterations as we expected with the benefits of batch normalization. However, the testing loss continues to increase afterwards so we still encounter the lingering overfitting issue. Another improvement is that the testing accuracy increased about 3%, which indicates with batch normalization, model generalization loss is reduced.

5.2 Dropout

Regularization is a way to optimize a machine learning model by penalizing complex models, hence minimizing loss and achieving better generalization results. Considering neural networks can be extremely complex and tend to overfit to the training data, as we see in the previous section, we apply dropout regularization to the baseline model. The dropout layer random selects connections in the neural network and makes their weights zero during training [5]. By disconnecting the hidden nodes randomly, dropout makes the training process noisy [5]. So we expected to observe a more turbulent loss curve. Because of the random disconnections, some information is lost after the dropout layers [5]. Hence, it is suggested to use a wider network to leverage the thinning effect [5]. However, since we aim to compare the behavior of dropout on the baseline model, we will keep the model otherwise the same as before. To avoid of losing too much information, we should start with a low dropout rate and gradually increase it while observing the outputs, and try to find a balanced dropout rate. It should be note that the dropout layer is only applied to the training architecture, and should not be used while making predictions on the testing dataset. So two CNN architectures were built for this model.

- Input layer;
- Convolution layer with 6 filters of size $5 \times 5 \times 3$, stride 1;
- ReLU layer;
- Dropout layer ("train" mode for training, "test" mode for classification);
- Max pooling layer with pooling size 2×2, stride 2;
- Convolution layer with 12 filters of size $5 \times 5 \times 6$, stride 1;
- ReLU layer;
- Dropout layer ("train" mode for training, "test" mode for classification);
- Max pooling layer with pooling size 2×2, stride 2;
- Convolution layer with 24 filters of size $5 \times 5 \times 24$, stride 1;
- ReLU layer;
- Dropout layer ("train" mode for training, "test" mode for classification);
- Max pooling layer with pooling size 2×2, stride 2;
- Fully connected layer of output dimension 120;
- ReLU layer;

- Dropout layer ("train" mode for training, "test" mode for classification);
- Fully connected layer of output dimension 84;
- ReLU layer;
- Dropout layer ("train" mode for training, "test" mode for classification);
- Fully connected layer of output dimension 10;
- Softmax.



(a) Training loss

(b) Testing loss

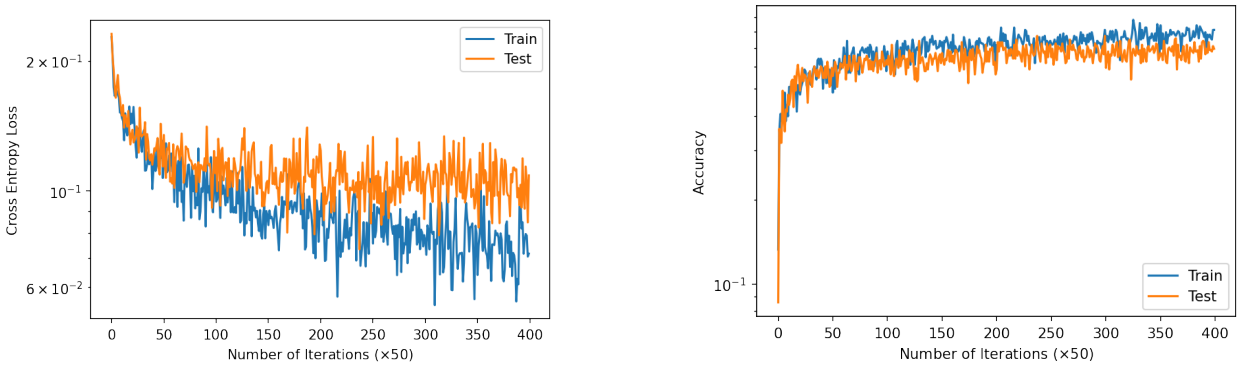
(c) Training accuracy

(d) Testing accuracy

Figure 12: Baseline model with different dropout rates results

We started experimenting with a 5% dropout rate (corresponding to the 95% curve in the charts, 95% indicates keeping 95% of the connections), and decreased the rate by 5% until reaching 35%. The results show that as the dropout rate increases, training losses and training accuracies become worse. This performance coincides with our expectation since as we choose to lose more information, it is less possible to extract meaningful features from each layer. On the other hand, as we retain more information, the model can overfit to the

training data and is poorly generalized on the testing set. Hence, we chose a dropout rate of 15% (keeping 85% of the connections) to balance the trade-off between underfitting and overfitting.



(a) Loss (b) Accuracy

Figure 13: Baseline model with 15% dropout rate results

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Dropout model	CIFAR-10	2:01	66706	0.07173	0.8125	0.108728	0.68349

Table 6: Baseline model with 15% dropout rate results

As we expected, the training and testing loss curves vibrate much more turbulently than previous models due to the randomness of the dropout layers. The testing loss continues to decrease which signifies that dropout can effectively combat the overfitting issue. Testing accuracy curve continues to rise and reach 68% accuracy on the testing set, achieving a 4% increase from the baseline accuracy.

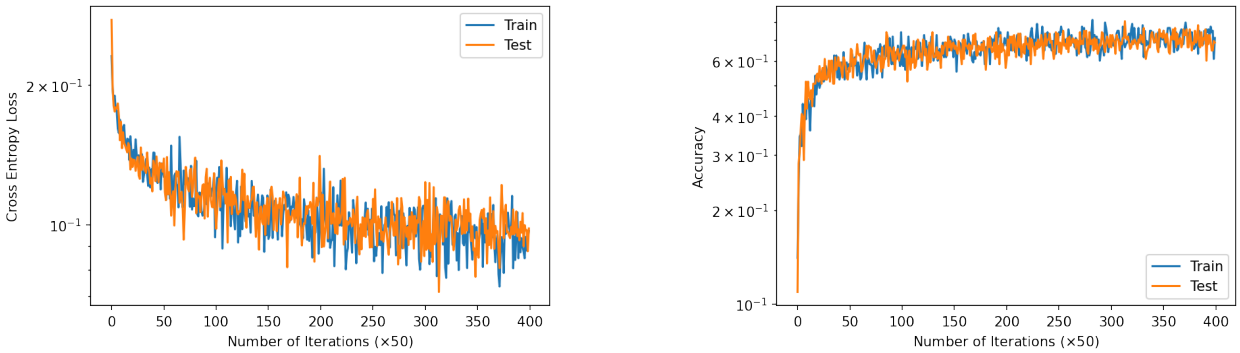
5.3 Batch Normalization, Dropout, and Data Augmentation

Combining all the techniques we have tried so far, we built a CNN model with batch normalization, dropout, and trained the model with augmented dataset (mirroring). Here are the results.

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Batchnorm, Dropout	CIFAR-10, flipped images	1:31	66790	0.09691	0.7109	0.09809	0.70309

Table 7: Baseline model with batch normalization and 15% dropout rate results

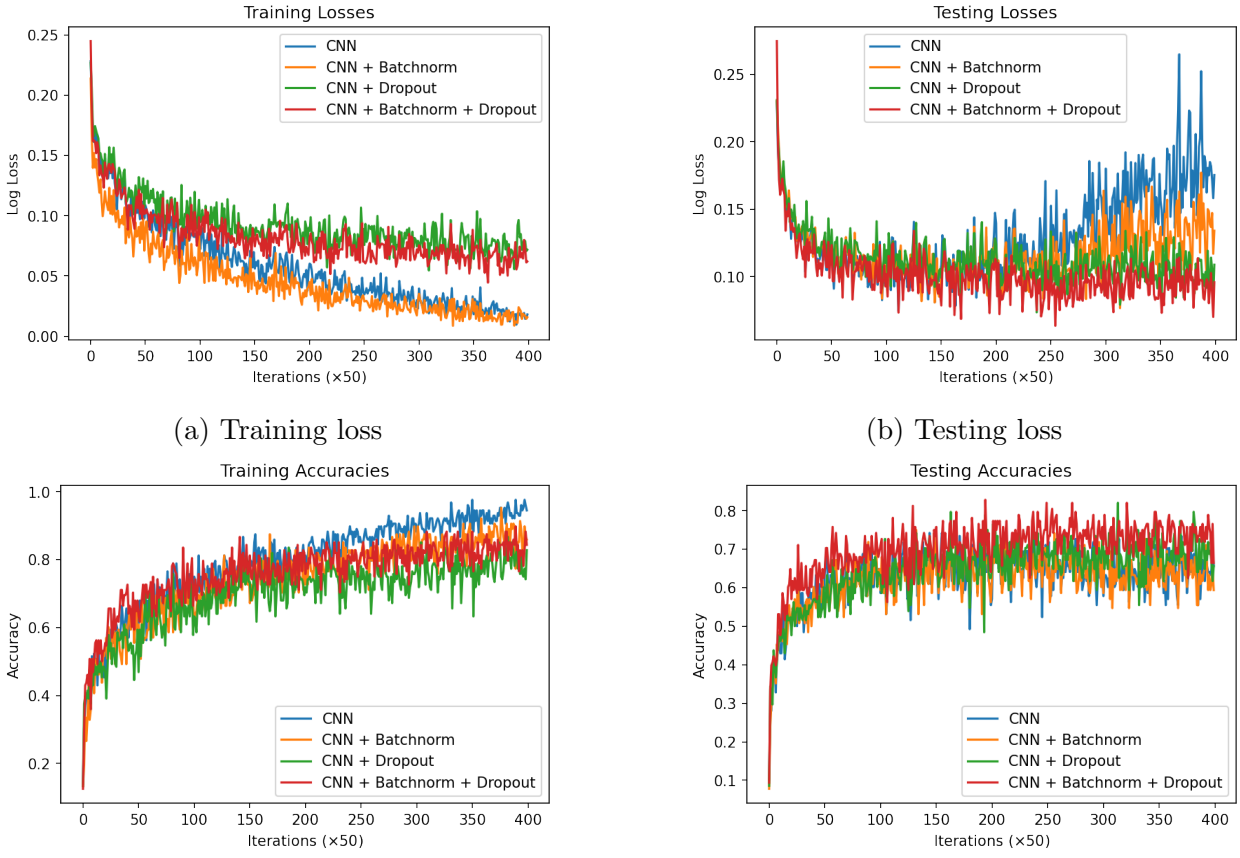
With all the techniques that have shown effective for combating overfitting and reducing generalization loss, our CNN model eventually achieved an accuracy of 70.3%. The continuously decreasing testing loss curve indicates our model has successfully defeated the predominant overfitting problem.



(a) Loss (b) Accuracy

Figure 14: Baseline model with batch normalization and 15% dropout rate results

5.4 Summary



(a) Training loss (b) Testing loss

(c) Training accuracy (d) Testing accuracy

Figure 15: Baseline model compared with models with batch normalization and dropout, trained with original CIFAR-10, results

Plotting the results of each model together, we observe that the baseline CNN model performs the best in terms of training loss, but worst in testing loss. This overfitting problem is significantly alleviated by batch normalization and dropout (Fig.15, red curve). With these two techniques, the model avoids overfitting to the training data, and generalizes well to the testing set, as it can be seen that its curve is above all the other in Fig.15d.

6 VGG 16 Model

6.1 Introduction of VGG Models

Based on our knowledge about CNN, it is easy to find that if we increase the number of convolution layers, it will become more effective in extracting features from the original images. However, increasing the ConvNet depth may cause some unexpected problems like gradient vanishing or overfitting data. Therefore, we need to evaluate its performance after we set the structure. Karen Simonyan and Andrew Zisserman proposed several deep convolutional networks for large-scale image classification [6]. They call it the VGG model. VGG models start with 11 weight layers and have a maximum of 19 weight layers, increasing the depth and having better performance. The most commonly used is the VGG16 model.

The VGG16 model has 13 convolution layers with increasing size of the filters and activation function ReLU, together with some max-pooling layers between them, three fully-connected layers in the last part, and the softmax function for final classification. Compared with other architectures that push to 16-19 layers, it shows significant improvement and has good robustness on different datasets.

Although it was initially designed for the ImageNet dataset, which is 224×224. It shows to work well with other sizes of inputs. But still, it needs to have some modifications on the hyperparameters before we use it on different input sizes or small datasets. Besides, we could make some small modfication to the architectures, like adding batch normalization, dropout or weight decay.

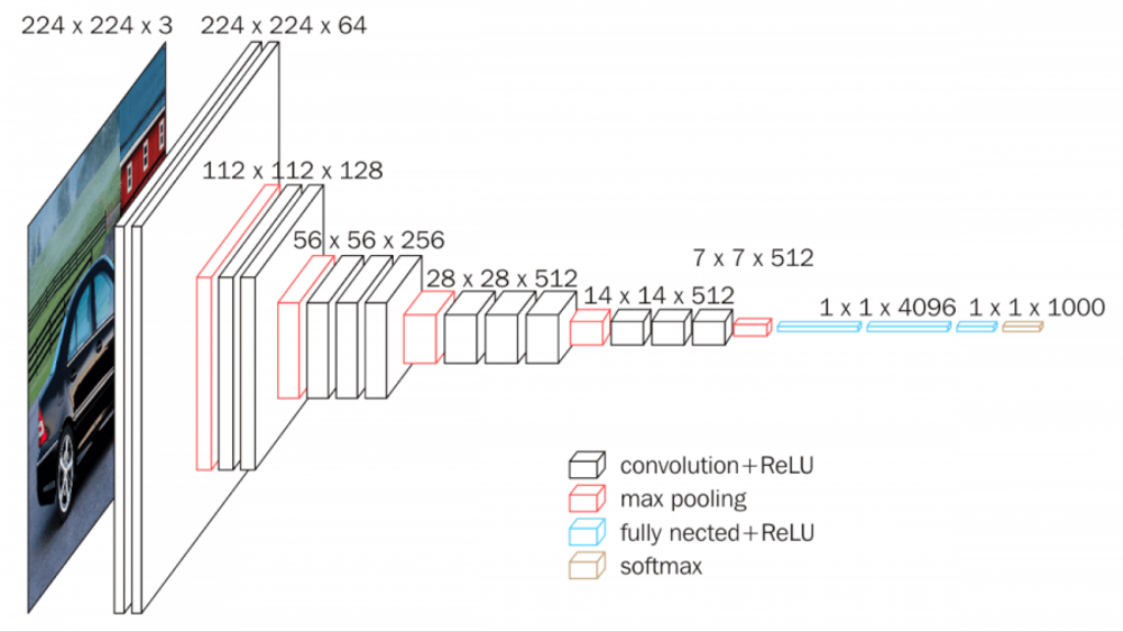
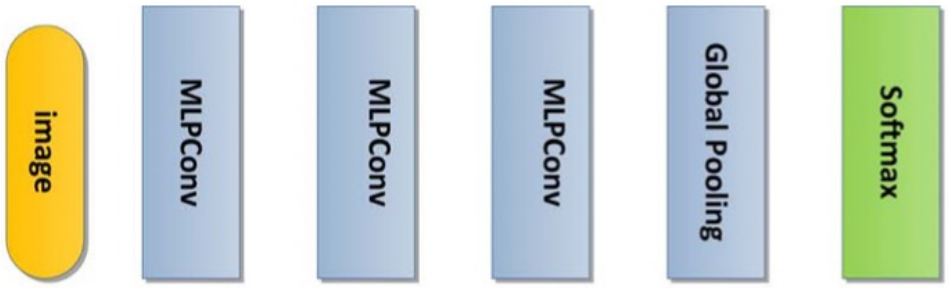


Figure 16: VGG16 Traditional Architectures

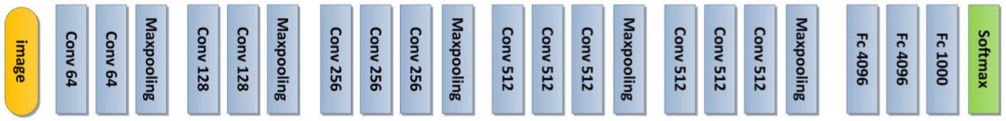
6.2 VGG16 model for Cifar 10

CIFAR-10 dataset is a small dataset, so in order to make VGG16 have a better performance on it, we definitely need some hyperparameter tuning . We briefly follow the work done by Shuying et al. and use their idea in modifying the traditional VGG16 model.[7]

As shown in Fig.17, there are two kinds of famous models, which are network in network and VGG-net models. Some people argue that VGG-net could only be used for a big dataset. Shuying et al. proved that although it may not be able to get a state-of-the-art result, its result is already good enough, as long as we make some modifications.



(a) Network in network model

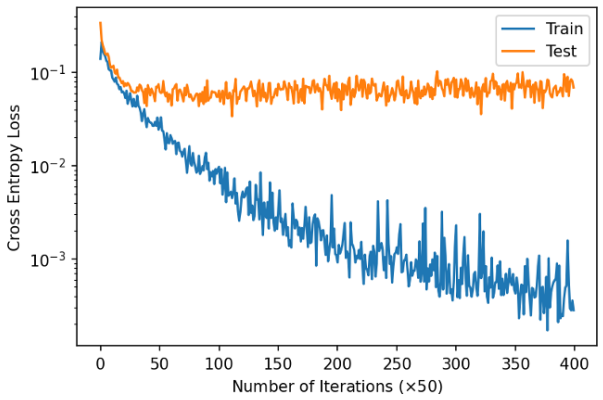


(b) VGG˙net model

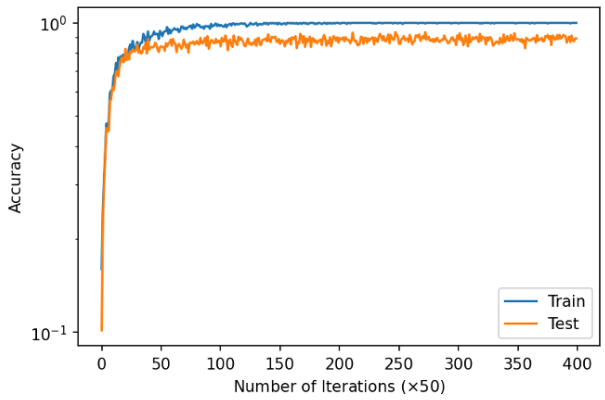
Figure 17: Famous models for image classification.

From our work above, and from the study done by Shuying et al., it is shown that if we add batch normalization and dropout in the model architectures, it could improve the accuracy. Besides, from the above result, we find that there is a best dropout rate for having the highest accuracy. It needs a long runtime and computation power to find a specific best dropout rate, so we just use the model architectures and dropout rate given by Shuying et al..

In our code, we keep the training loop to be 20,000 iterations and change the batchsize from 128 to 256, in order to learning more features at one time.



(a) VGG16 loss



(b) VGG16 accuracy

Figure 18: VGG16 for CIFAR-10

From Fig.19a, we can see that after 20,000 iterations of training,it has a quite low loss

on both the training and the test data. Fig.19a also does not show an overfitting trend. In Fig.19b it shows that after we did such modifications to the traditional VGG16 model, model could have a good performance both on the training dataset and the test dataset. The training accuracy after 20,000 iterations reaches to 1.0 and the test accuracy is 88.9% which is significant higher than our baseline model.

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
VGG16	CIFAR-10	22:59	14990924	0.00028189635	1.0	0.06896409	0.889300003051758

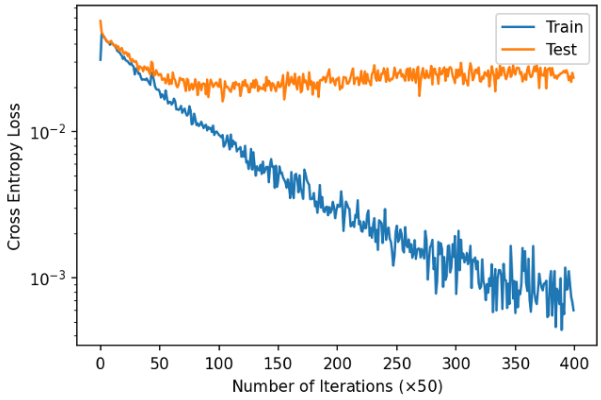
Table 8: VGG16 CIFAR-10 results

From Table.8 we find that as we slightly increase the depth of our model, together with the fact that we change the batch size to be 256, it has a large number of parameters and needs more runtime. Its performance is worth the cost of runtime and memories. In a real-world application, when we need to deal with a dataset bigger than CIFAR-10 for image classification, using the VGG16 model could always give a decent accuracy. However, we should be aware that the drop out rate and other hyperparameters we use in this VGG16 model could only have good performance with the CIFAR-10 dataset. The result would be terrible if we directly use it on prediction for other datasets. When we change to other datasets, we need to do a fine-tune and make some small modification for the model, based on the accuracy we want, the computational resources we have.

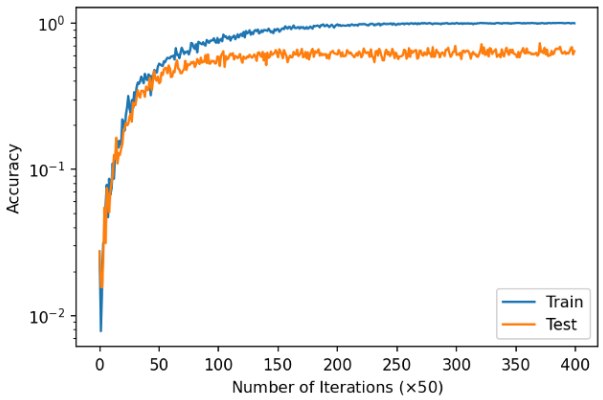
In the work of Shuying et al., they get the accuracy to be about 91.55%, the difference may due to different settings on training method like the choice of the learning rate, how the weight decay applied, or some appropriate data augmentation before.[7]

6.3 VGG16 model for CIFAR-100

In order to test our model robustness, we use the same architectures and do image classification on CIFAR-100. In Fig.19, we can find that it has a similar graph for loss and accuracy changes, which shows the model working properly and do not overfit the data. In Table.9, we find that as now it is 100 classes, the model have a little more parameters compared with VGG16 on CIFAR-10. The accuracy is 63.58%. Cifar is a really small dataset. For CIFAR-100, it has only 600 images for each classes in total, which make the model could not have enough images to learn. Our accuracy is not bad compared to results from several literatures.



(a) VGG16 loss



(b) VGG16 accuracy

Figure 19: VGG16 for CIFAR-100

Model	Dataset	Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
VGG16	CIFAR-100	22:10	15037094	0.00059852446	0.99609375	0.023349063	0.635899963378906

Table 9: VGG16 CIFAR-100 Results

7 State Of The Art Models

The last part will discuss two state-of-the-art models: MLP-mixer and Vision Transformer. These are two models proposed by the Google team. For the image classification task, the last part of the model architectures should be some fully-connected layers and the softmax activation function for prediction. They change the architectures for the feature extraction part to have better performance and more robust. However, we should notice that compared to our CNN models, these SOTA models may need more computational resources and need a pretty long time to train from the beginning. As a result, when we are going to use it for a different dataset, we usually load a file containing those pre-trained weights and then do a tuning process to make it work better for our specific dataset.

7.1 MLP Mixer

Convolutional Neural Network is always a popular computer vision and image classification model. Recently, some attention-based networks are also being studied a lot. Although these models have an outstanding performance, and the improvements in computation resources and the availability of large datasets make it possible to do such extensive training, we do not always want to make the task complex and take several days training it on TPUs. It would be better if there is a model that is not so computational intensive but could give a similar accuracy. That is why we need the MLP-mixer model.[8]

MLP-mixer is based on multi-layer perceptrons, which is relatively easy to implement. The 'Mixer' means to mix different features and gain information from their relations, like the convolution kernel we use in CNN. It only needs several matrix multiplications and introduces nonlinearities. The whole architectures is shown in Fig.20. MLP-mixer contains patch embedding and projection, Mixer layers and fully-connected layers. In Mixer layers, two MLPs are applied in order to deal with different pixel locations called token -mixing and different channels called channel-mixing. There is a GELU activation in the MLP to introduce nonlinearity. Besides, there is also skip-connection and layer norm in the model.[8]

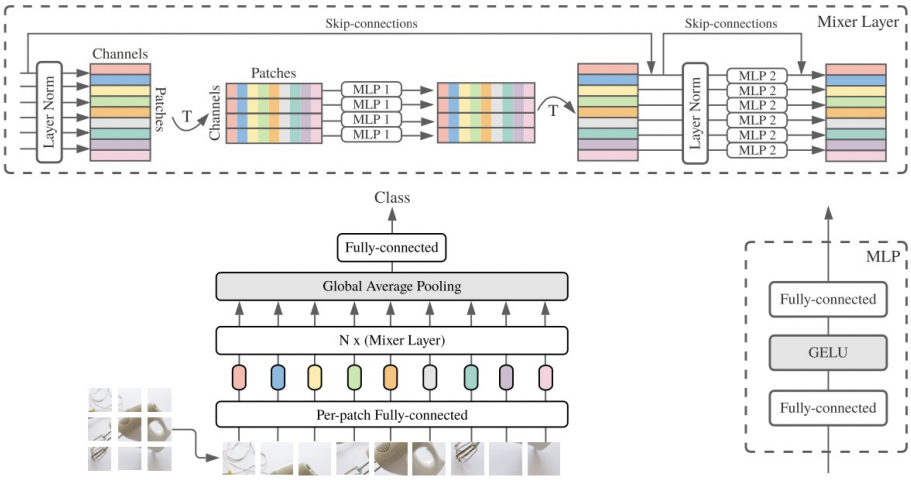


Figure 20: Model Architectures

In the architectures, before we feed our training data into the model, we need to change the raw image data into several patches. Then after a linear patch projection layer, the patches are projected into latent representation and feed into the Mixer layer. In the Mixer layers, there are two MLPs for the token-mixing and the channel-mixing, together with skip connection, layer norm and possibly dropouts. In each MLP, there also exists shared weight along all the columns or rows. Then after the global average pooling layer, we use the standard fully-connect layers to make predictions. Some parts from the MLP-mixer architectures is also been widely used in other models like layernorm, skip connections.

Specification	S/32	S/16	B/32	B/16	L/32	L/16	H/14
Number of layers	8	8	12	12	24	24	32
Patch resolution $P \times P$	32×32	16×16	32×32	16×16	32×32	16×16	14×14
Hidden size C	512	512	768	768	1024	1024	1280
Sequence length S	49	196	49	196	49	196	256
MLP dimension D_C	2048	2048	3072	3072	4096	4096	5120
MLP dimension D_S	256	256	384	384	512	512	640
Parameters (M)	19	18	60	59	206	207	431

Figure 21: MLP-mixer Parameters

From Fig.21, we can see that compared to the parameters in VGG16 model, only large MLP-mixer model could have similar amount of parameters. Usually it has less parameters in training, meaning that we do not require a large memory for computers.

But still, this model is complex than our baseline model, which definitely need more training time. Compared with modified VGG16 model, the runtime is somehow depend on the settings for training, but the runtime of MLP-mixer could not be as long as several days.

Usually there are two ways to apply MLP-mixer model on datasets, which is using pre-trained weight then doing hyperparameter tuning, and training from stretch. Results from google shows that if using pre-trained weight on ImageNet and then do classification for Cifar10, it could reach a accuracy of 97.68%. From other's result, if we train a MLP-mixer model from the very beginning, it could take a longer time and then reach a accuracy of 89.15%

7.2 Vision Transformer

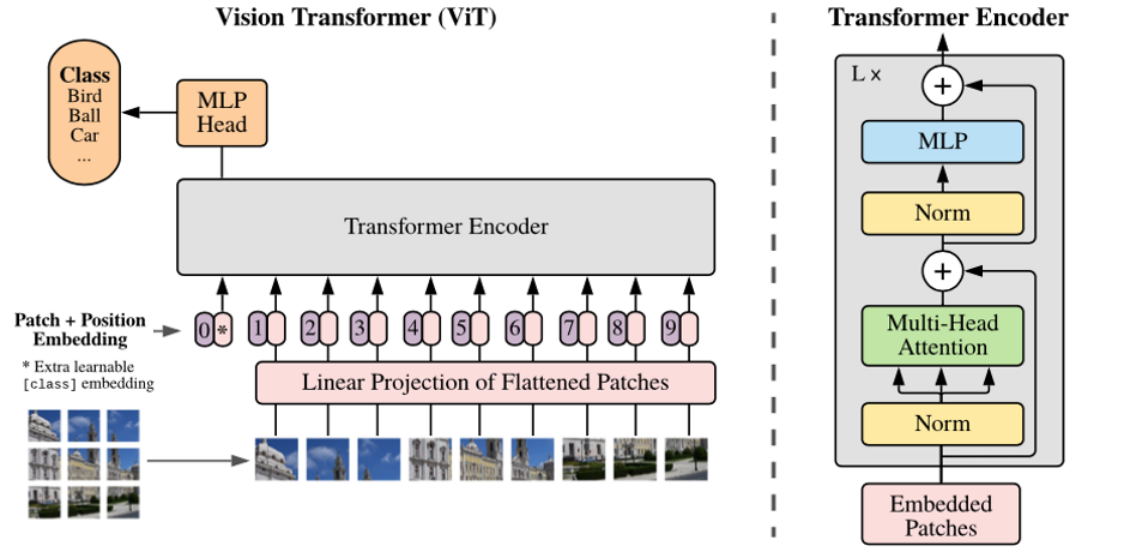


Figure 22: Vision Transformer

Transformer architecture is the most attractive model in natural language processing in recent years. It introduce an attention based mechanisms and are widely used in making sequences to sequences predictions. Vision Transformer is a model proposed by Google, which uses similar idea of transformer into computer vision and image classification area.[9]

The architecture of vision transformer is shown in Fig.22. It includes 3 parts: Patch embedding, Transformer encoder, and classification head. Similar to MLP-mixer, in vision transformer, we also dealing with patches rather than raw image data, considering that one image could be a sequence of patches, so before the transformer encoder, we need to do patch embedding and a linear projection. Transformer is the most common architecture used in NLP. However, we should notice that at here we only need the encoder part, which contains a multi-headed attention(MHA) layer and a MLP, together with some common normalization layer and residual connections. MHA is a combination of several attention layers, which makes the model could have attention on patches from different spatial place. Also we could briefly understand the attention layer as calculating the relation between different word vectors (image patches), deriving a score and getting a output. Then at last we use the classification head and softmax for final prediction.

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Figure 23: Vision Transformer Parameters

Fig.23 shows 3 kinds of VIT model. We notice that VIT-Base has 86M parameters with is about 60% compared with VGG16. VIT-Large and VIT Huge has slightly large number of parameters. Usually, VIT model are pre-trained on large dataset and then fine-tune to small dataset. Besides, in training we usually use a large batch size to let the model have attention on more figures at one time.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** $\pm$ 0.04	87.76 $\pm$ 0.03	85.30 $\pm$ 0.02	87.54 $\pm$ 0.02	88.4/88.5*
ImageNet ReaL	**90.72** $\pm$ 0.05	90.54 $\pm$ 0.03	88.62 $\pm$ 0.05	90.54	90.55
CIFAR-10	**99.50** $\pm$ 0.06	99.42 $\pm$ 0.03	99.15 $\pm$ 0.03	99.37 $\pm$ 0.06	–
CIFAR-100	**94.55** $\pm$ 0.04	93.90 $\pm$ 0.05	93.25 $\pm$ 0.05	93.51 $\pm$ 0.08	–
Oxford-IIIT Pets	**97.56** $\pm$ 0.03	97.32 $\pm$ 0.11	94.67 $\pm$ 0.15	96.62 $\pm$ 0.23	–
Oxford Flowers-102	99.68 $\pm$ 0.02	**99.74** $\pm$ 0.00	99.61 $\pm$ 0.02	99.63 $\pm$ 0.03	–
VTAB (19 tasks)	**77.63** $\pm$ 0.23	76.28 $\pm$ 0.46	72.72 $\pm$ 0.21	76.29 $\pm$ 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Figure 24: Vision Transformer Accuracy

Fig.24 shows the accuracy of applying VIT model on different datasets comparing with other SOTA models. We can find that VIT pretrained by different datasets. It have good performance on the test datasets. For CIFAR-10, its accuracy can reach to about 99.50% and 94.55% for CIFAR-100.

8 Conclusion

In this project, we focus on the image classification task and try different data augmentation methods and model configurations, discuss its effect on the accuracy.Our baseline model is a simple CNN with 3 convolution layers. Its accuracy is about 63%

We find that in Fig.3a, it has a obvious trend of overfitting the data. In order to resolve this problem, we try different data augmentation methods, which are grayscale, brightness and mirroring. We find that all these 3 method could somehow resolve the problem and the most effective method is flipped images. Although we could avoid overfitting in training the model, we only improve a small level of test accuracy.

Then, in order to get a higher accuracy, we explore the most common modifications for the CNN model, which are batch norm and drop out. We get an increase of 3% for simply add batch norm to the original architectures. For adding drop out, as there needs one more hyperparameter: drop out rate (or the rate for keeping the connection), we do a experiment and show that as we increase the drop out rate, the accuracy would increase at first and then for a larger rate it would decrease. The best drop out rate for our model is 15%, and we could get an incrase of 4% for the accuracy. If we add batchnorm and dropout at the same time, together with some data augmentation, we can finally reach a accuracy 70.3%

If we properly increase the depth of the network, we could get better accuracy. We tried the most famous VGG16 model and train it from stretch on Cifar 10 and Cifar 100. VGG model has a large number of parameters and need long training time, compared to our baseline model. We get 88.9% for Cifar10 and 63.5% for Cifar100, which is reasonable. It is also shown that our finding could be generalize to other datasets and have a similar result.

Finally, we go through some state-of-the-art models: MLX-mixer and Vision Transformer. We find that they all has more parameters than our baseline model. They use different architectures for learning features and are all dealing with patches generated from raw features. These guarantees that the model could have better performance. Besides, as they have a large number of parameters and the training process could be very long, we usually pretrain it on large datasets and the do fine-tuning when apply it to small dataset.

References

[1] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[2] Yoshua Bengio Ian Goodfellow and Aaron Courville. In *Deep Learning*, pages 318–320, 425, 2016.

[3] Christian Szegedy Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. 2015.

[4] Andrew Ilyas Aleksander Madry Shibani Santurkar, Dimitris Tsipras. How does batch normalization help optimization? 2018.

[5] Alex Krizhevsky Ilya Sutskever Ruslan Salakhutdinov Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15, 2014.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[7] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[8] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *CoRR*, abs/2105.01601, 2021.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.