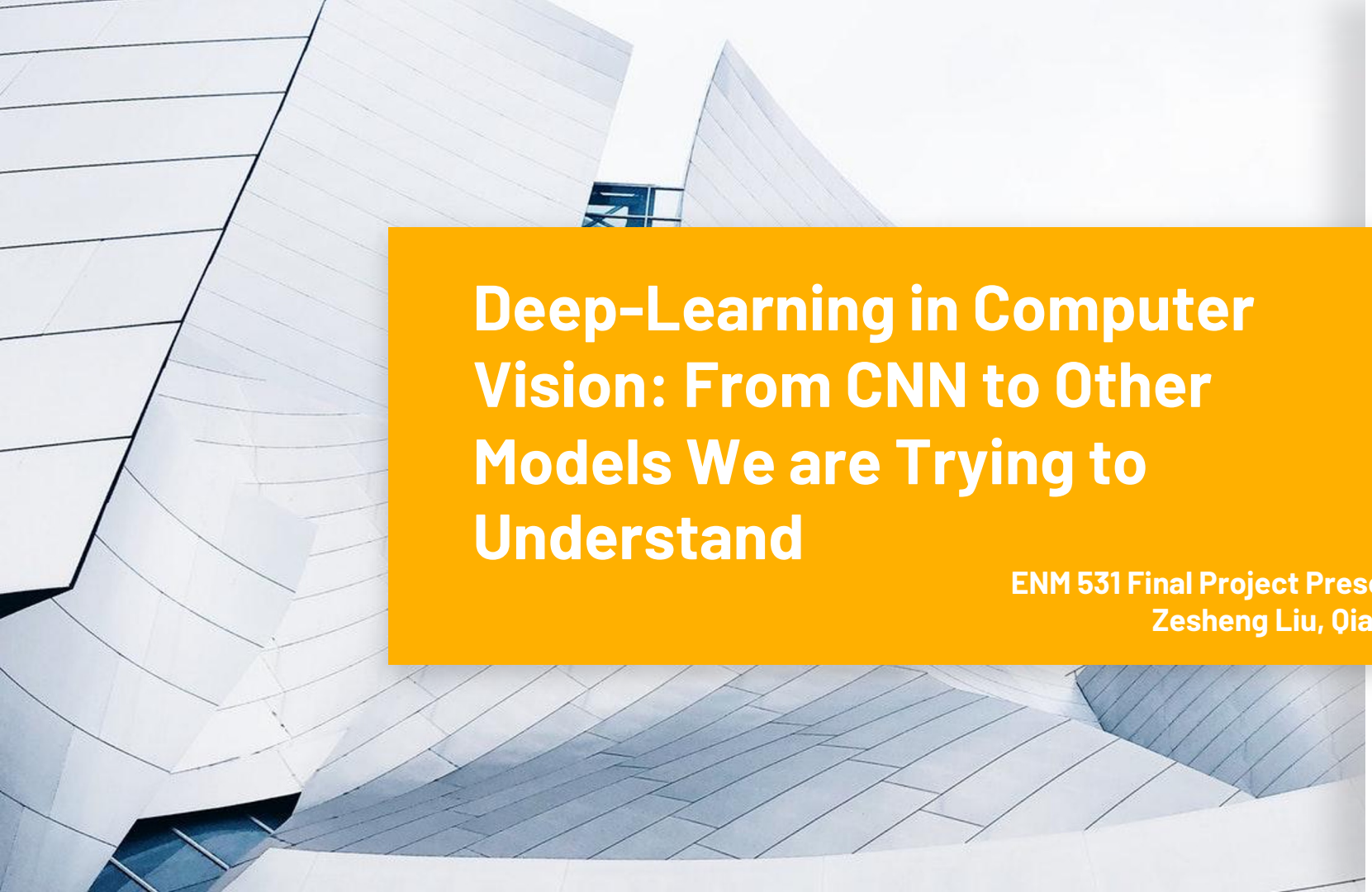


Deep-Learning in Computer Vision: From CNN to Other Models We are Trying to Understand

ENM 531 Final Project Presentation

Zesheng Liu, Qianxun Xu

Outline

- Project Overview
- Baseline Model
- Baseline Model with Data Augmentation
- Model Modifications
- Other Models
  - VGG16
  - MLP-Mixer
  - ViT

1. Project Overview

Project Overview

Objective

- Image classification with neural network models
- Explore other models

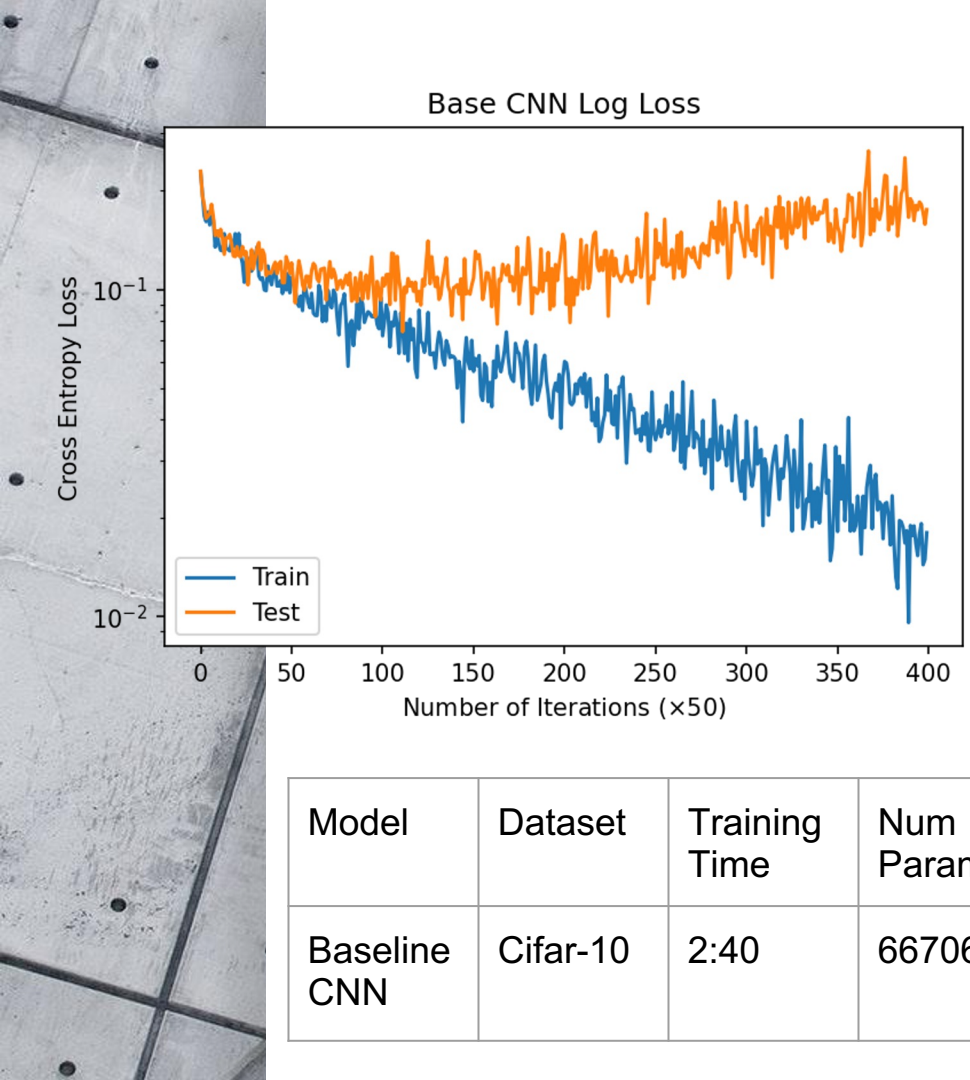
Dataset

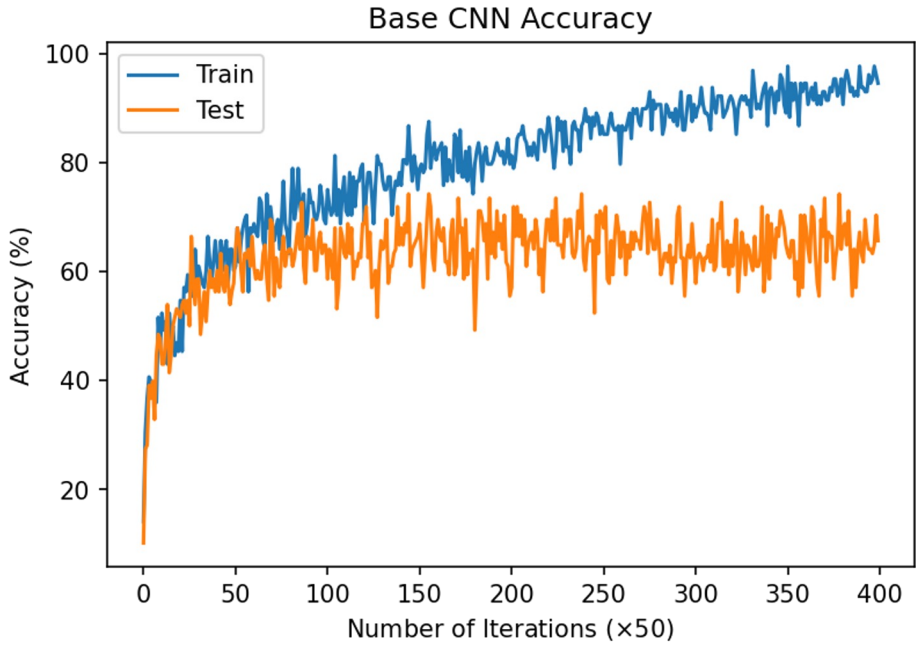
- Cifar-10;
- Training samples 50,000;
- Testing samples 10,000;
- Size 32x32 images;
- 10 classes

2. Baseline Model

CNN

```
# Architecture
def CNN():
    init, apply = stax.serial(Conv(6, (5, 5), (1, 1), padding="SAME"),
                              Relu,
                              MaxPool((2, 2), (2, 2)),
                              Conv(12, (5, 5), (1, 1), padding="SAME"),
                              Relu,
                              MaxPool((2, 2), (2, 2)),
                              Conv(24, (5, 5), (1, 1), padding="SAME"),
                              Relu,
                              MaxPool((2, 2), (2, 2)),
                              Flatten,
                              Dense(120), Relu,
                              Dense(84), Relu,
                              Dense(10), Softmax
                             )
    return init, apply
```





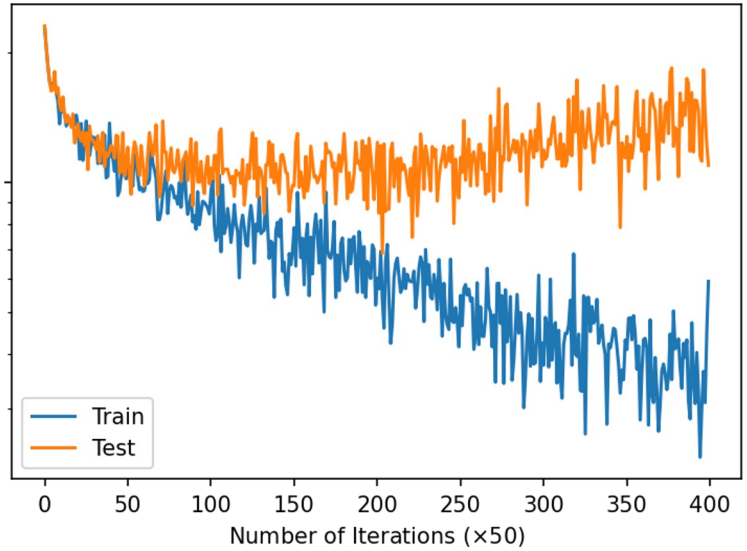
Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline CNN	Cifar-10	2:40	66706	0.01805	0.945312	0.1752	0.6411

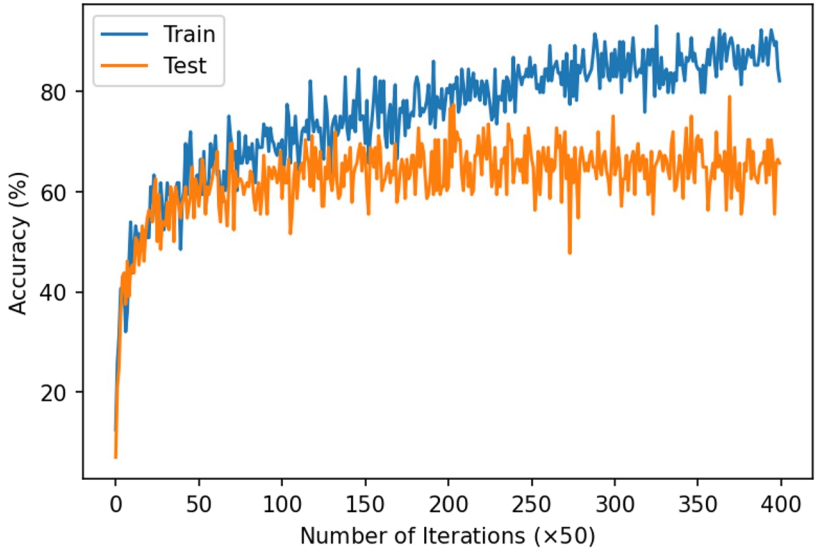
3. Baseline Model with Data Augmentation

CNN + Grayscale Images

Original

Grayscale



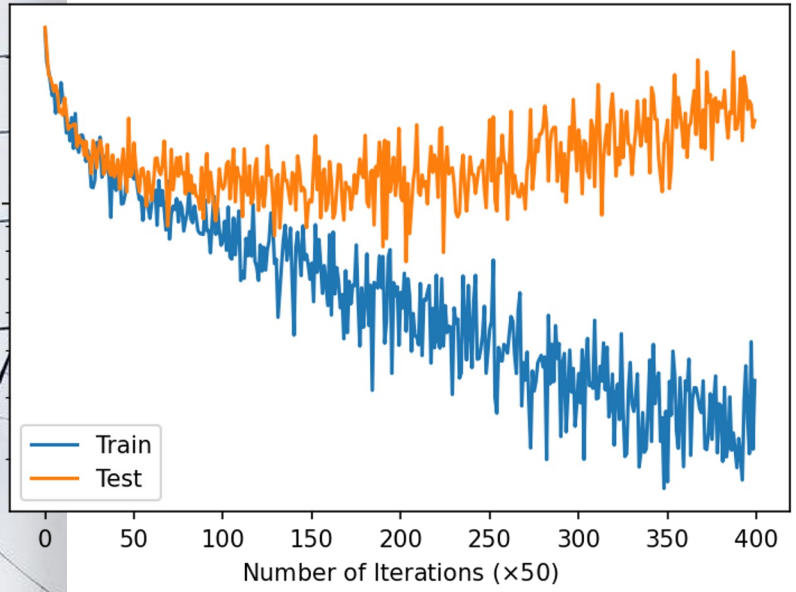


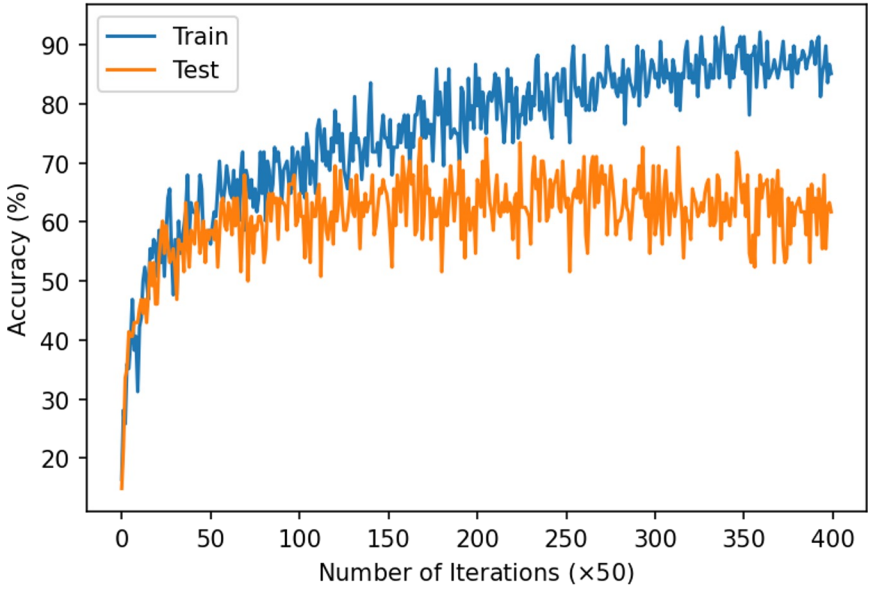
Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline CNN	Cifar-10, Grayscale Images	2:44	66706	0.05897	0.820312	0.109432	0.6423999

CNN + Images with 75% Brightness

Original

75% Brightness

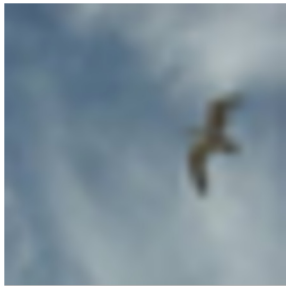


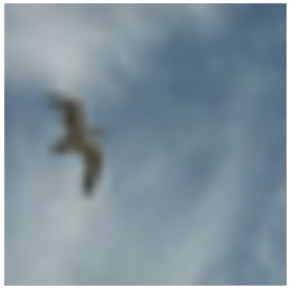


Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline CNN	Cifar-10, 75% Brightness Images	3:33	66706	0.04403	0.859375	0.1339	0.6374

CNN + Flipped Images

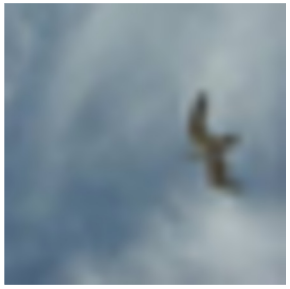
Original

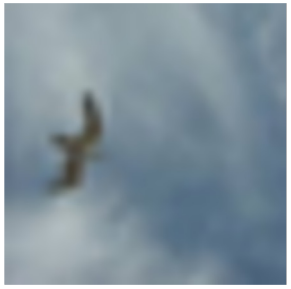




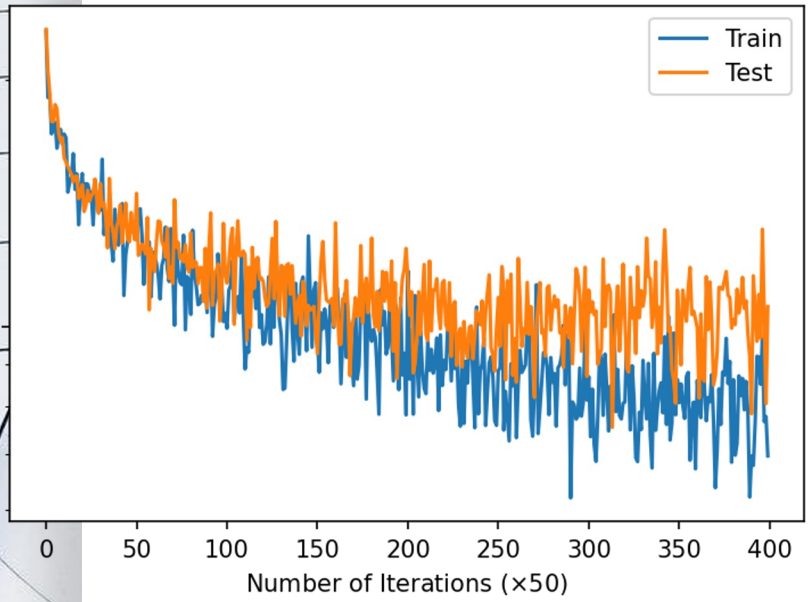
Horizontal Flip

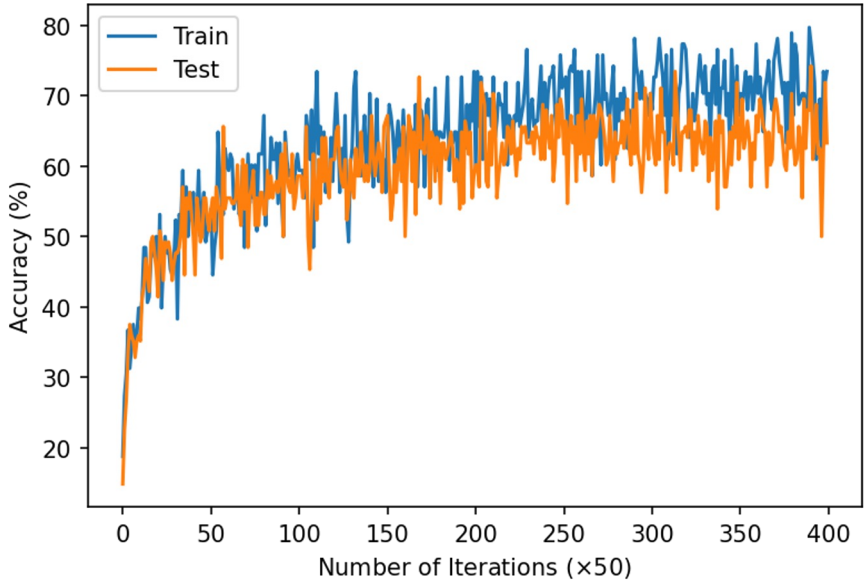
Vertical Flip





Transpose



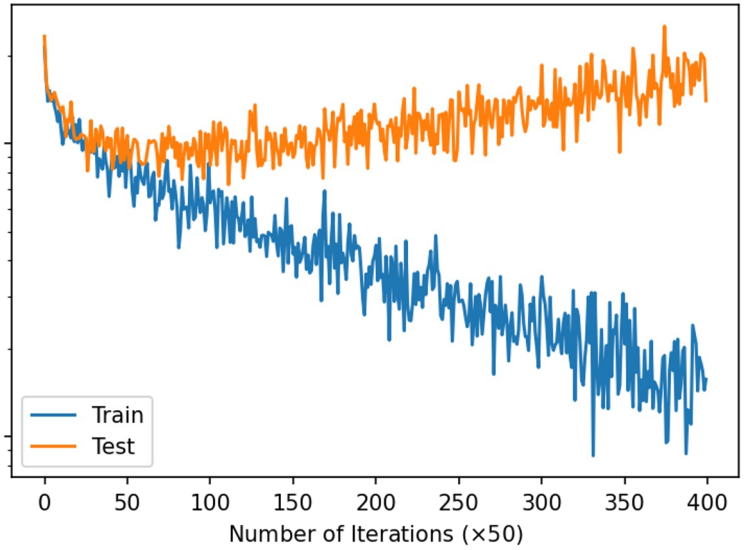


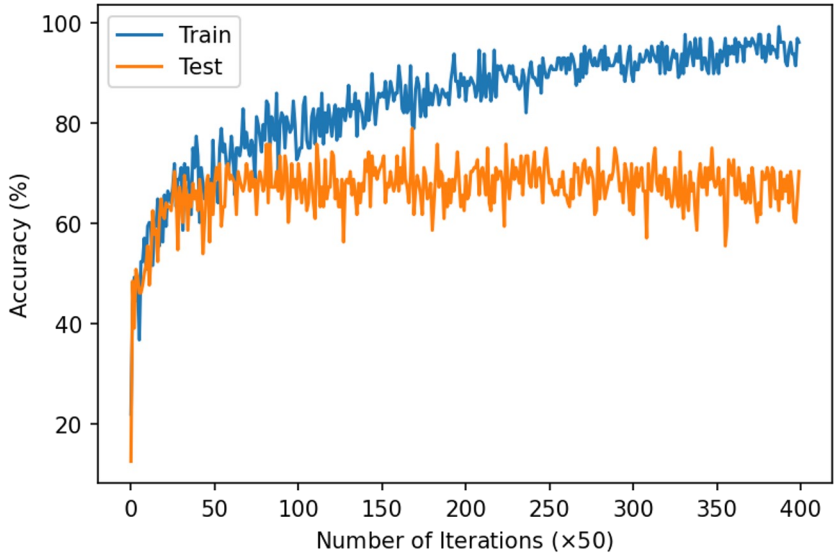
Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
Baseline CNN	Cifar-10, Flipped Images	3:29	66706	0.06967	0.734375	0.1058	0.6458

4. Model Modifications

CNN + Batchnorm

```
# Architecture
def CNN():
    init, apply = stax.serial(Conv(6, (5, 5), (1, 1), padding="SAME"),
                              BatchNorm(),
                              Relu,
                              MaxPool((2, 2), (2, 2)),
                              Conv(12, (5, 5), (1, 1), padding="SAME"),
                              BatchNorm(),
                              Relu,
                              MaxPool((2, 2), (2, 2)),
                              Conv(24, (5, 5), (1, 1), padding="SAME"),
                              BatchNorm(),
                              Relu,
                              MaxPool((2, 2), (2, 2)),
                              Flatten,
                              Dense(120), Relu,
                              Dense(84), Relu,
                              Dense(10), Softmax
                             )
    return init, apply
```



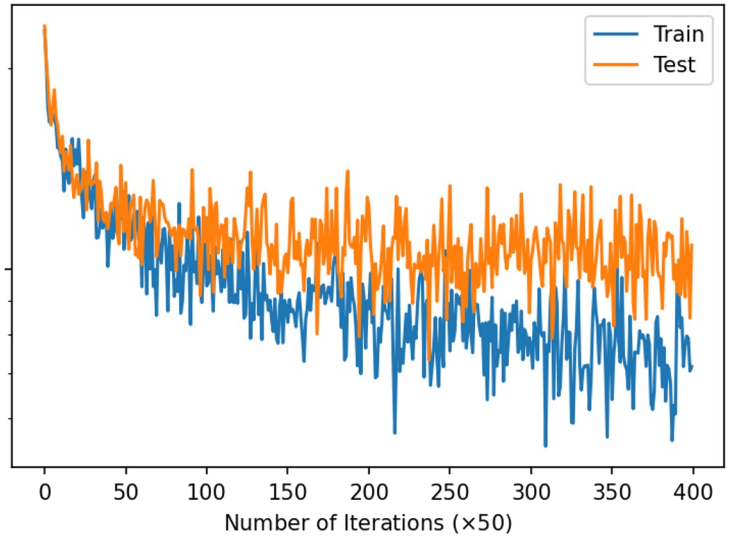


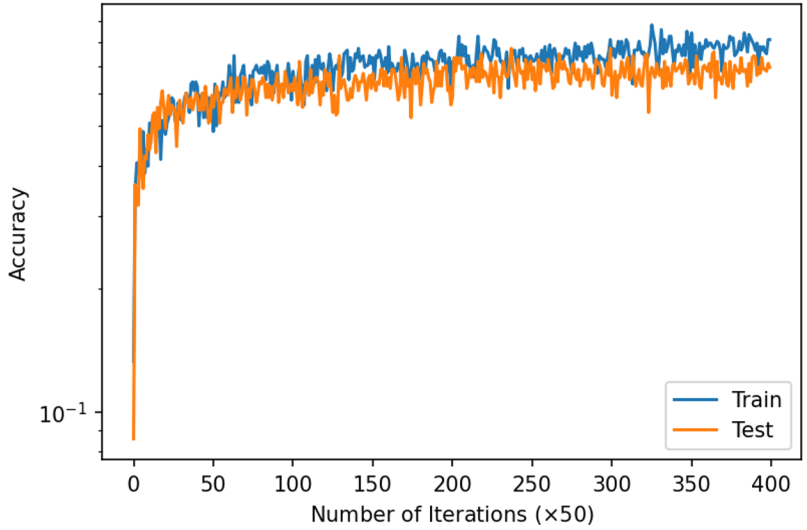
Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
CNN with Batchnorm	Cifar-10	2:54	66790	0.01572	0.960937	0.1402 6694	0.673

CNN + Dropout

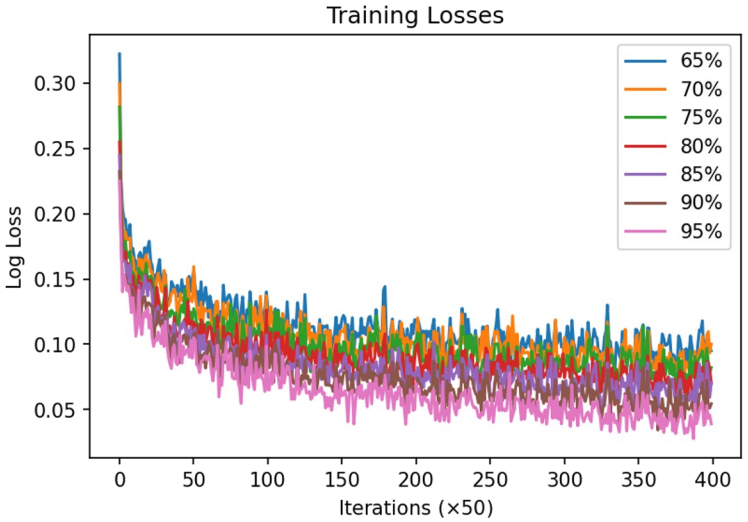
```
# Architecture
def CNN():
    init, apply = stax.serial(Conv(6, (5, 5), (1, 1), padding="SAME"),
                              Relu,Dropout(0.85,mode='train'), MaxPool((2, 2), (2, 2)),
                              Conv(12, (5, 5), (1, 1), padding="SAME"),
                              Relu,Dropout(0.85,mode='train'), MaxPool((2, 2), (2, 2)),
                              Conv(24, (5, 5), (1, 1), padding="SAME"),
                              Relu,Dropout(0.85,mode='train'), MaxPool((2, 2), (2, 2)),
                              Flatten,Dropout(0.85,mode='train'),
                              Dense(120), Relu, Dropout(0.85,mode='train'),
                              Dense(84), Relu, Dropout(0.85,mode='train'),
                              Dense(10), Softmax)
    return init, apply
```

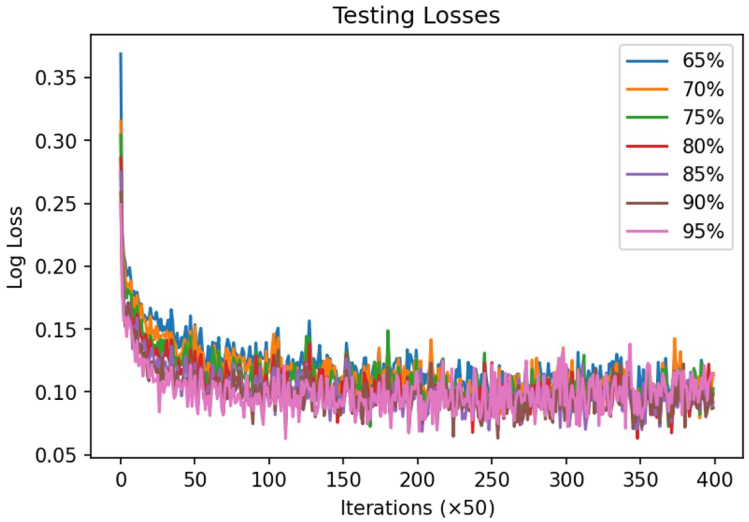
```
# Architecture
def CNN_test():
    init, apply = stax.serial(Conv(6, (5, 5), (1, 1), padding="SAME"),
                              Relu,Dropout(0.85,mode='test'), MaxPool((2, 2), (2, 2)),
                              Conv(12, (5, 5), (1, 1), padding="SAME"),
                              Relu,Dropout(0.85,mode='test'), MaxPool((2, 2), (2, 2)),
                              Conv(24, (5, 5), (1, 1), padding="SAME"),
                              Relu,Dropout(0.85,mode='test'), MaxPool((2, 2), (2, 2)),
                              Flatten,Dropout(0.85,mode='test'),
                              Dense(120), Relu,Dropout(0.85,mode='test'),
                              Dense(84), Relu,Dropout(0.85,mode='test'),
                              Dense(10), Softmax)
    return init, apply
```

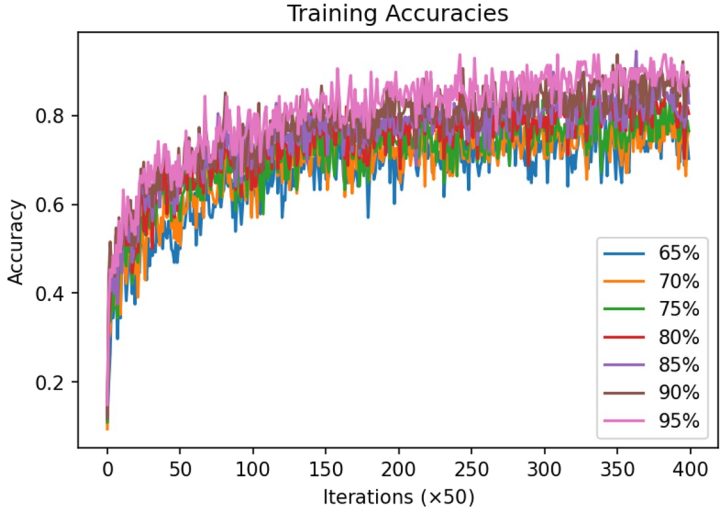


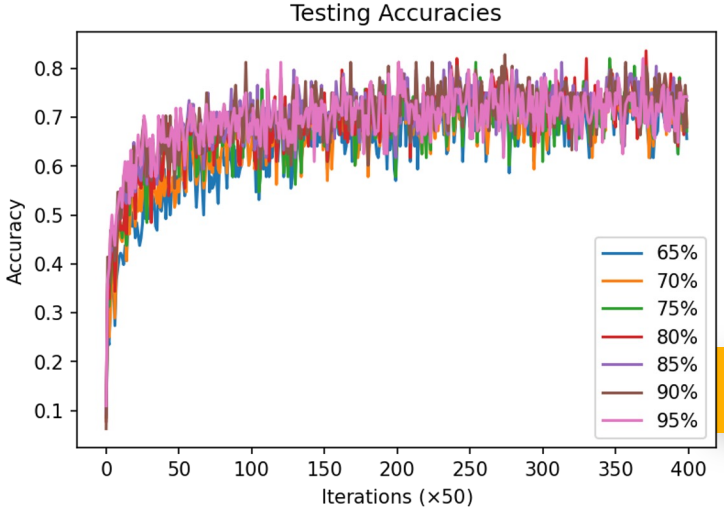


Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
CNN with Dropout	Cifar-10	2:01	66706	0.07173	0.8125	0.10872 8945	0.6834999


Training Losses
Log Loss
Iterations (×50)
65%
70%
75%
80%
85%
90%
95%

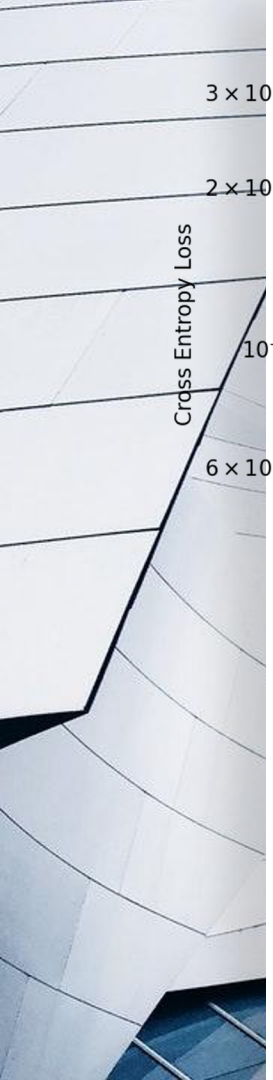

Testing Losses
Log Loss
Iterations (×50)
65%
70%
75%
80%
85%
90%
95%

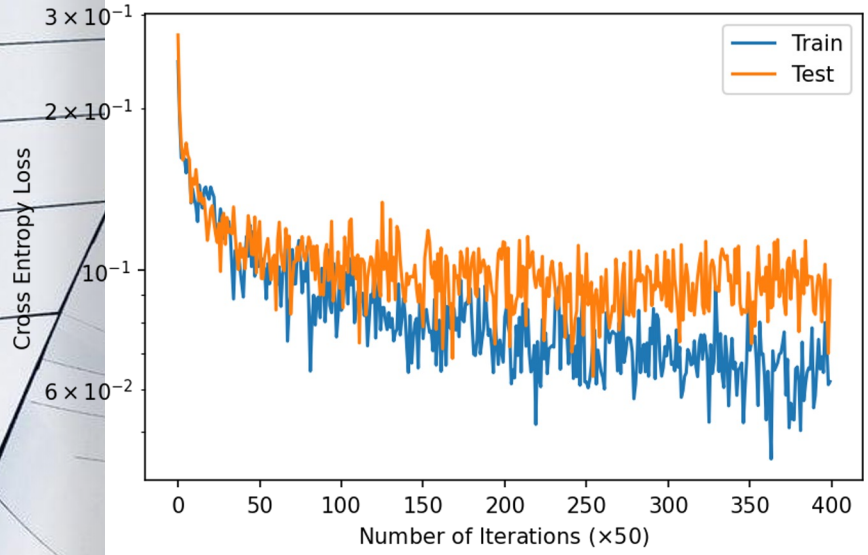

Training Accuracies
Accuracy
Iterations (×50)
65%
70%
75%
80%
85%
90%
95%

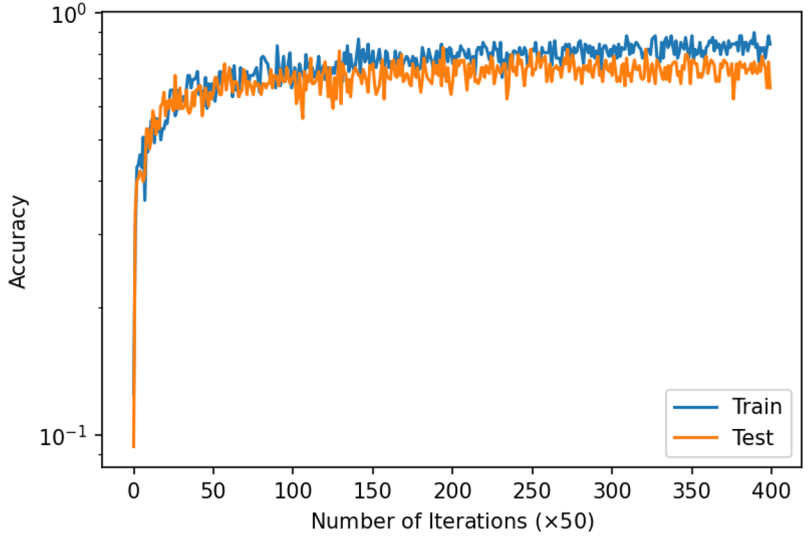

Testing Accuracies
Accuracy
Iterations (×50)
65%
70%
75%
80%
85%
90%
95%

CNN + Batchnorm + Dropout

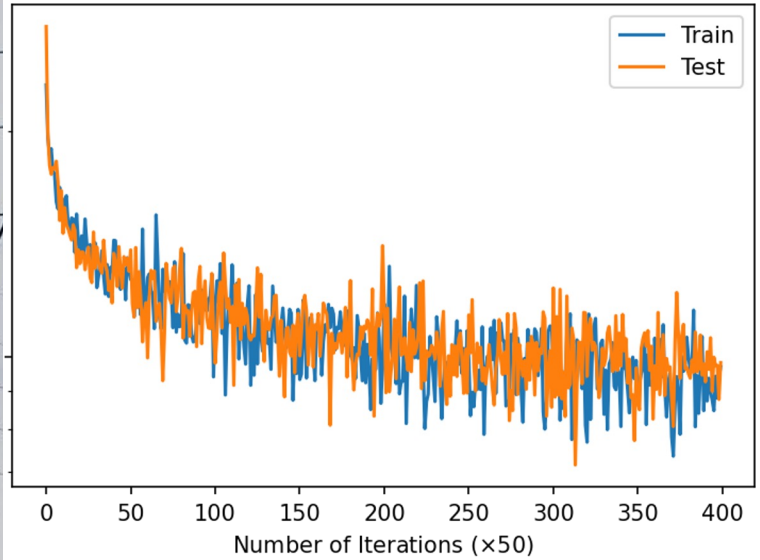
```
# Architecture
def CNN():
    init, apply = stax.serial(Conv(6, (5, 5), (1, 1), padding="SAME"),BatchNorm(),
                              Relu,Dropout(0.85,mode='train'), MaxPool((2, 2), (2, 2)),
                              Conv(12, (5, 5), (1, 1), padding="SAME"),BatchNorm(),
                              Relu,Dropout(0.85,mode='train'), MaxPool((2, 2), (2, 2)),
                              Conv(24, (5, 5), (1, 1), padding="SAME"),BatchNorm(),
                              Relu,Dropout(0.85,mode='train'), MaxPool((2, 2), (2, 2)),
                              Flatten,Dropout(0.85,mode='train'),
                              Dense(120), Relu, Dropout(0.85,mode='train'),
                              Dense(84), Relu, Dropout(0.85,mode='train'),
                              Dense(10), Softmax)
    return init, apply
```

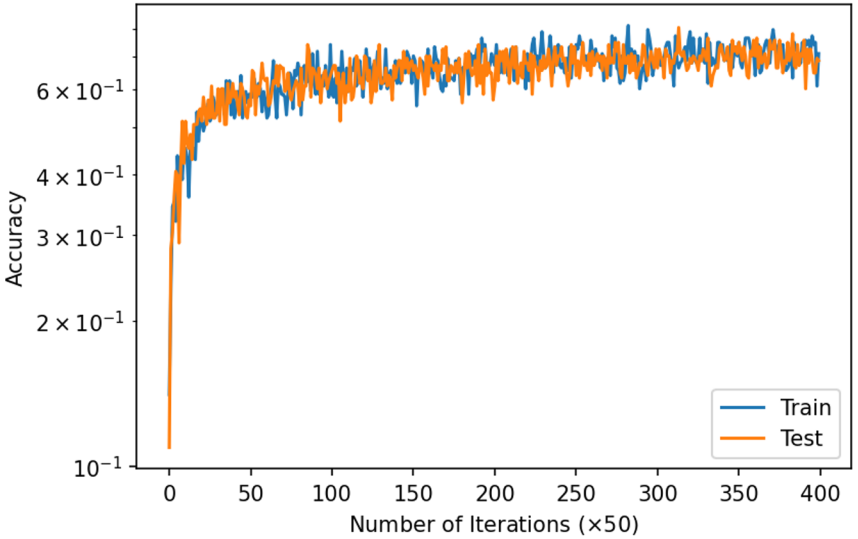




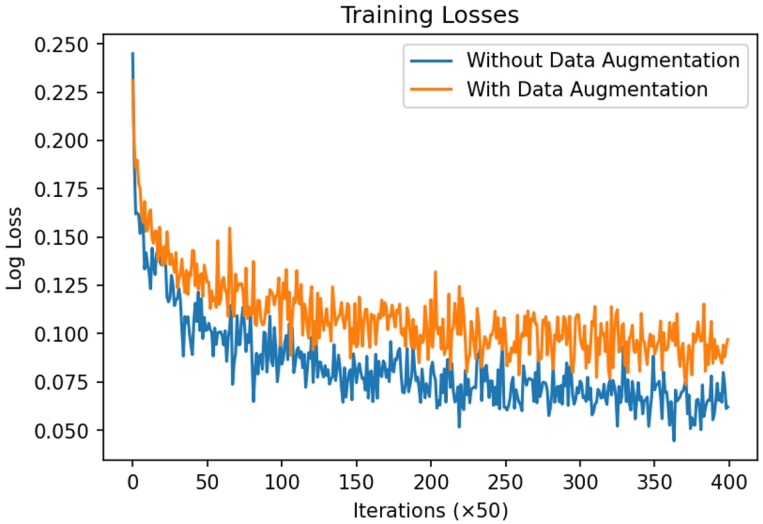


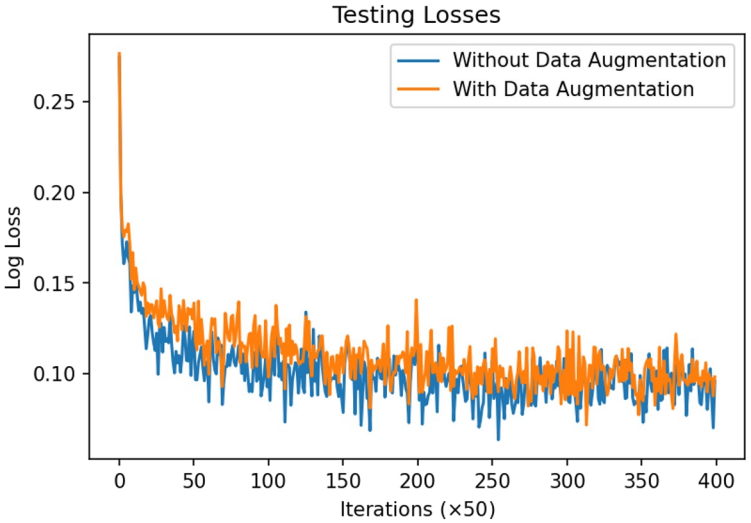
Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
CNN with Batchnorm, Dropout	Cifar-10	1:30	66790	0.06202073	0.84375	0.09572039	0.74059997558

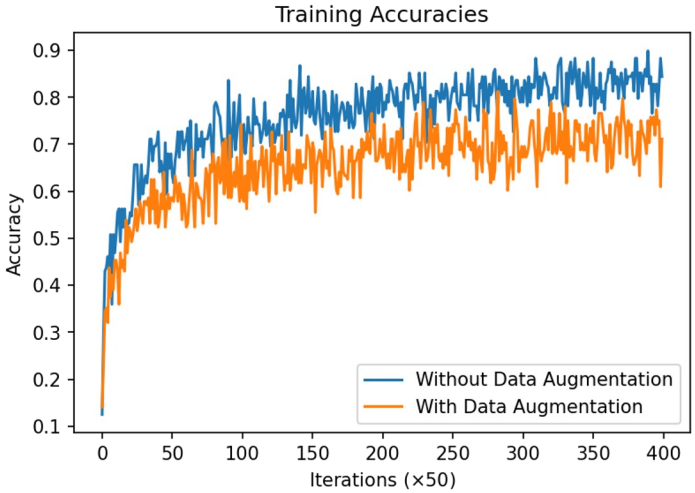


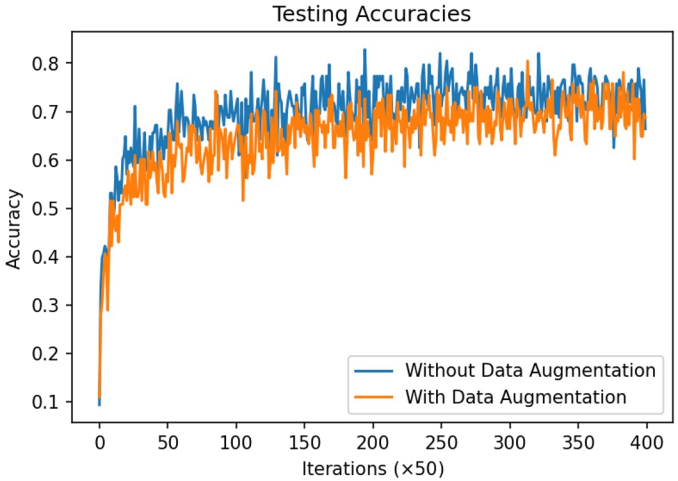


Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
CNN with Batchnorm, Dropout	Cifar-10, Flipped Images	1:31	66790	0.09691 2846	0.710937 5	0.09809 923	0.7030999 7558

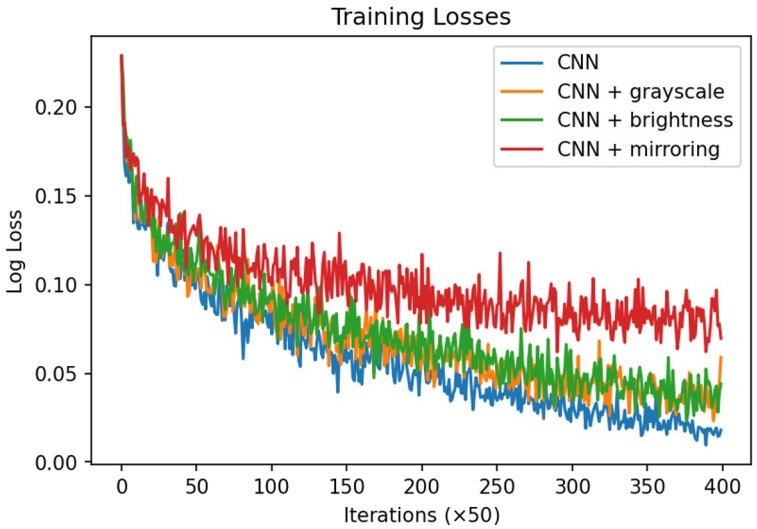

Training Losses
Without Data Augmentation
With Data Augmentation
Log Loss
Iterations (×50)

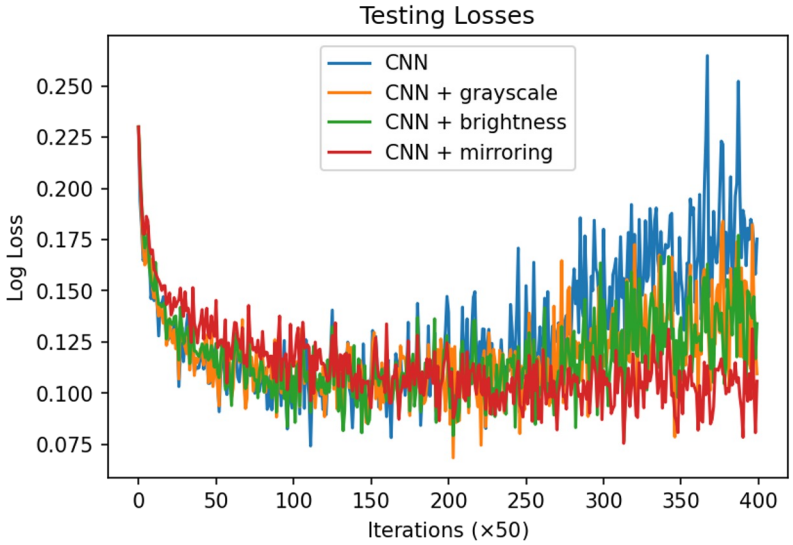

Testing Losses
Without Data Augmentation
With Data Augmentation
Log Loss
Iterations (×50)

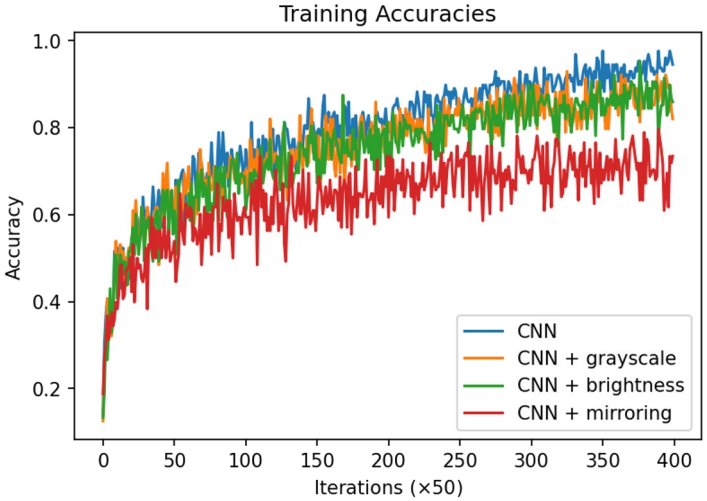

Training Accuracies
Without Data Augmentation
With Data Augmentation
Accuracy
Iterations (×50)

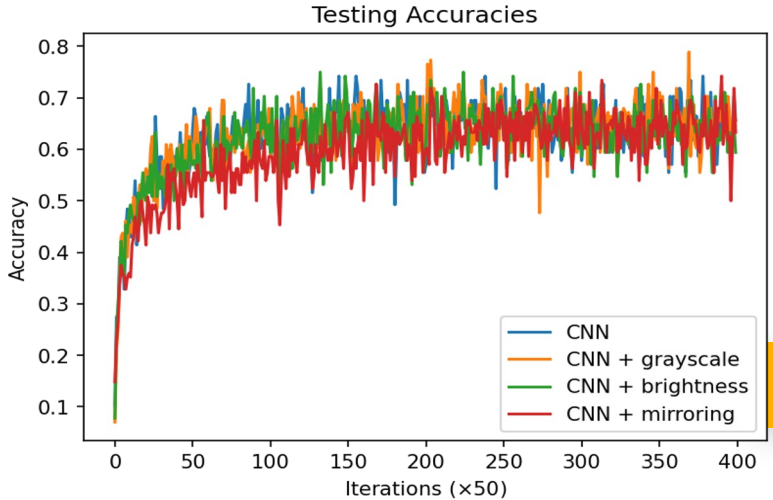

Testing Accuracies
Without Data Augmentation
With Data Augmentation
Accuracy
Iterations (×50)

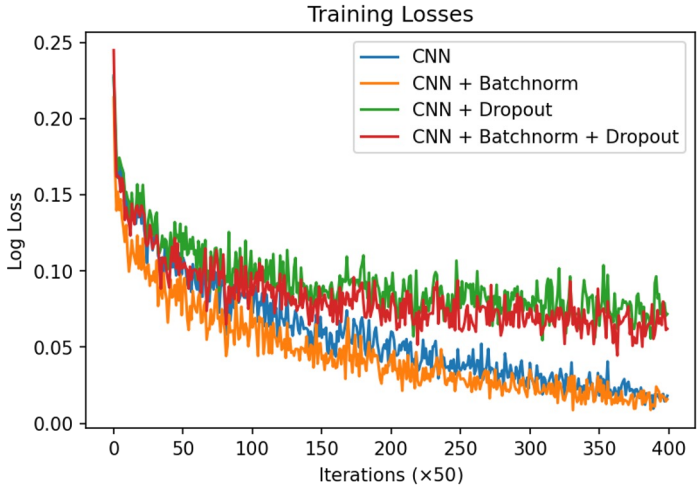
Summary Plots

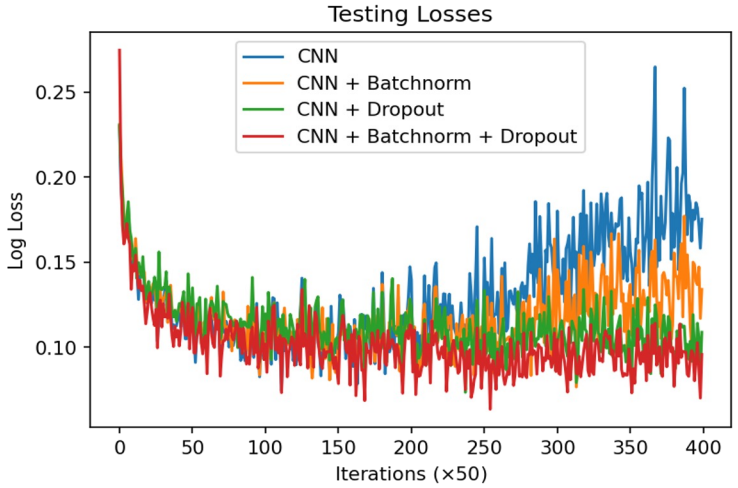

Training Losses
CNN
CNN + grayscale
CNN + brightness
CNN + mirroring
Log Loss
Iterations (×50)

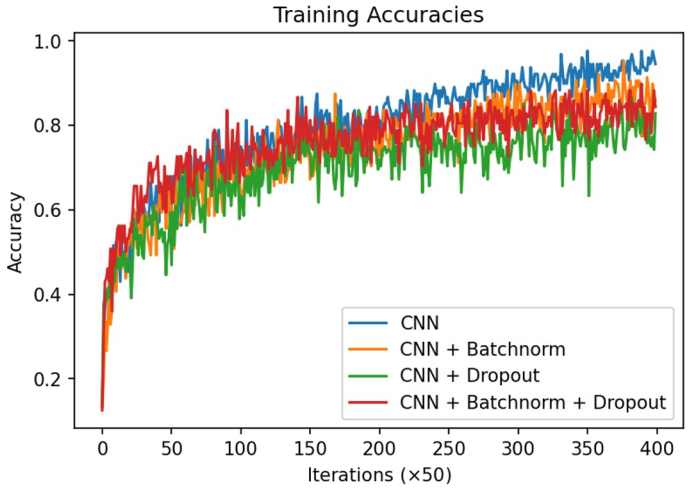

Testing Losses
CNN
CNN + grayscale
CNN + brightness
CNN + mirroring
Log Loss
Iterations (×50)

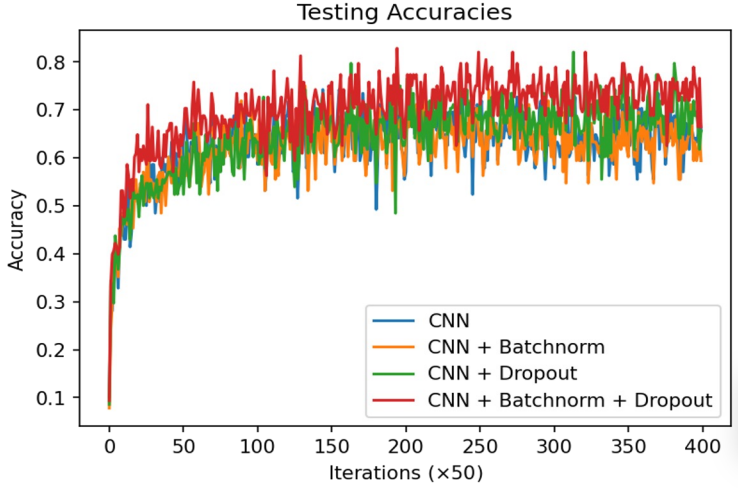

Training Accuracies
CNN
CNN + grayscale
CNN + brightness
CNN + mirroring
Accuracy
Iterations (×50)


Testing Accuracies
CNN
CNN + grayscale
CNN + brightness
CNN + mirroring
Accuracy
Iterations (×50)


Training Losses
CNN
CNN + Batchnorm
CNN + Dropout
CNN + Batchnorm + Dropout
Log Loss
Iterations (×50)

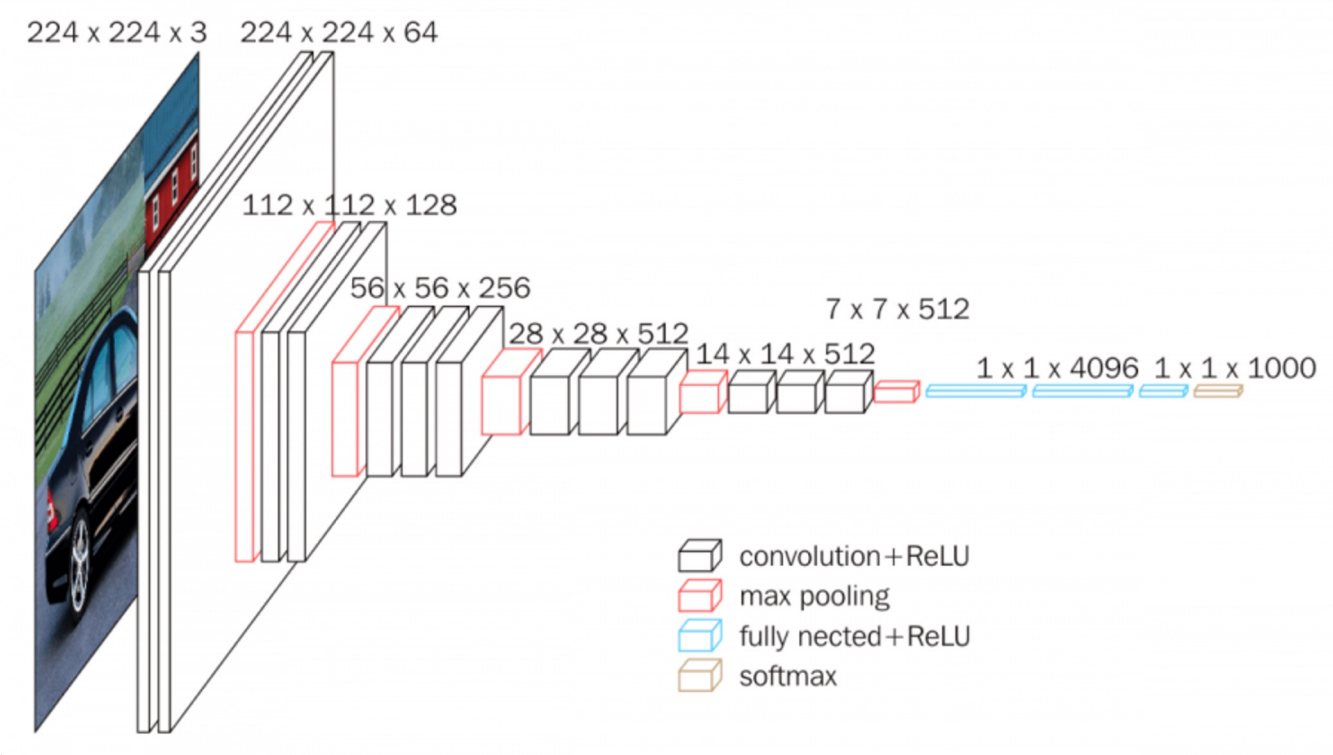

Testing Losses
CNN
CNN + Batchnorm
CNN + Dropout
CNN + Batchnorm + Dropout
Log Loss
Iterations (×50)


Training Accuracies
CNN
CNN + Batchnorm
CNN + Dropout
CNN + Batchnorm + Dropout
Accuracy
Iterations (×50)


Testing Accuracies
CNN
CNN + Batchnorm
CNN + Dropout
CNN + Batchnorm + Dropout
Accuracy
Iterations (×50)

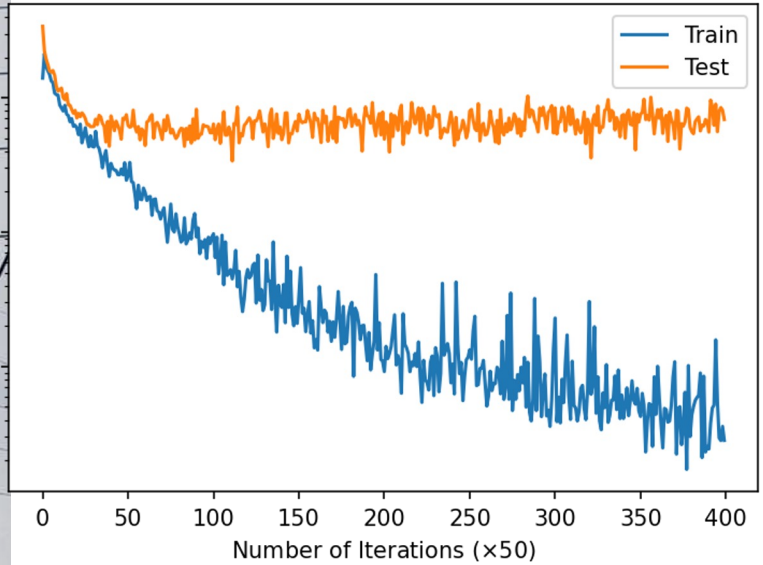
5. Other Models

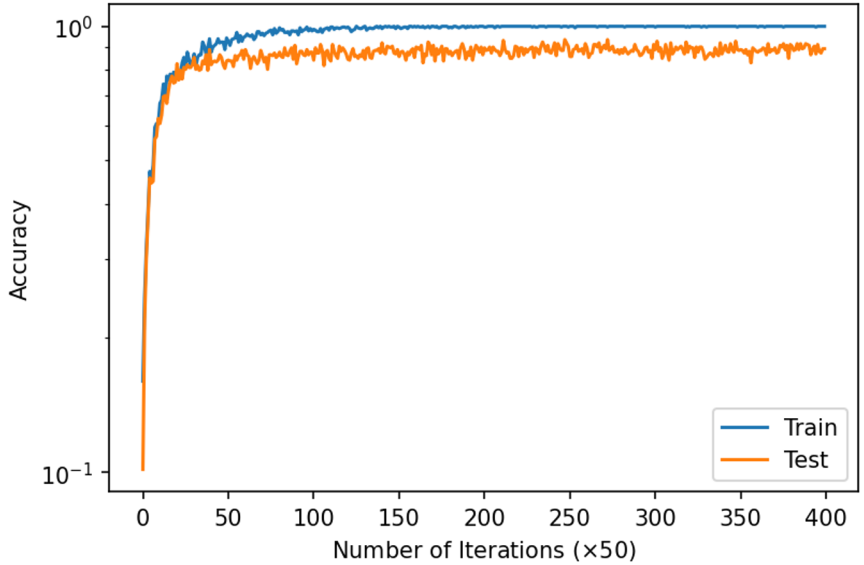
VGG16


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

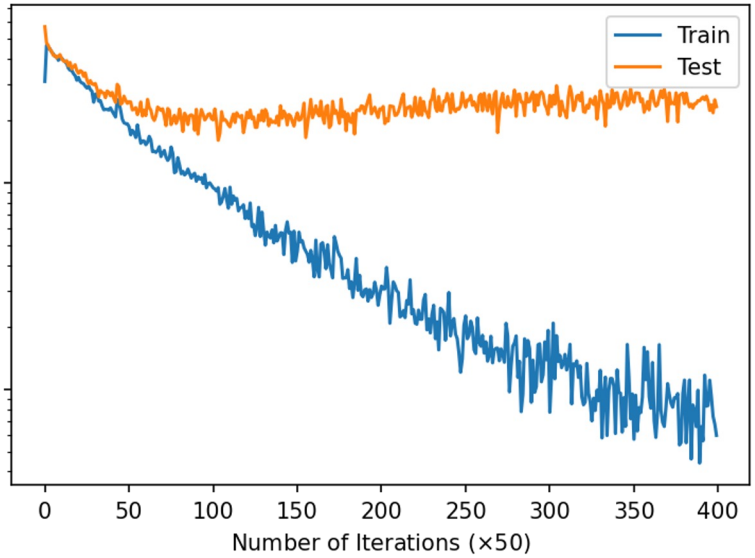
VGG16

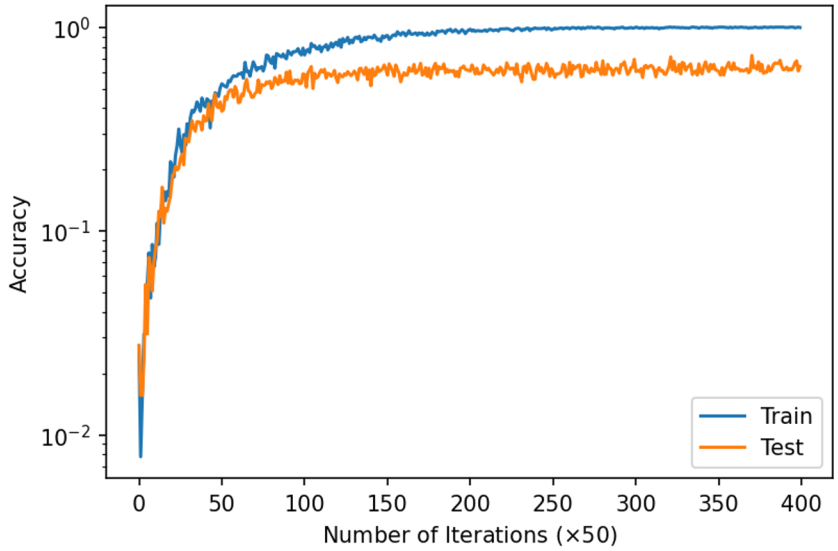
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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





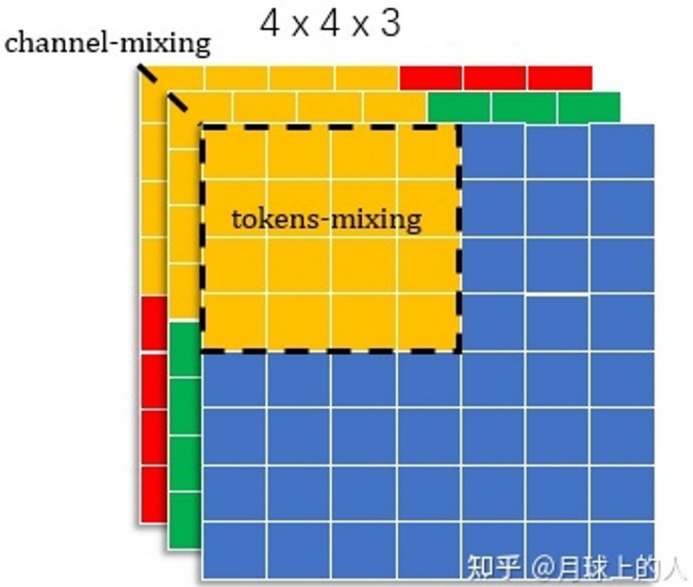
Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
VGG16	**Cifar-10**	22:59	14,990,924	0.00028189635	1.0	0.06896409	0.889300003051758





Model	Dataset	Training Time	Num Params	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy
VGG16	**Cifar-100**	22:10	15,037,094	0.00059852446	0.99609375	0.023349063	0.635899963378906

MLP Mixer

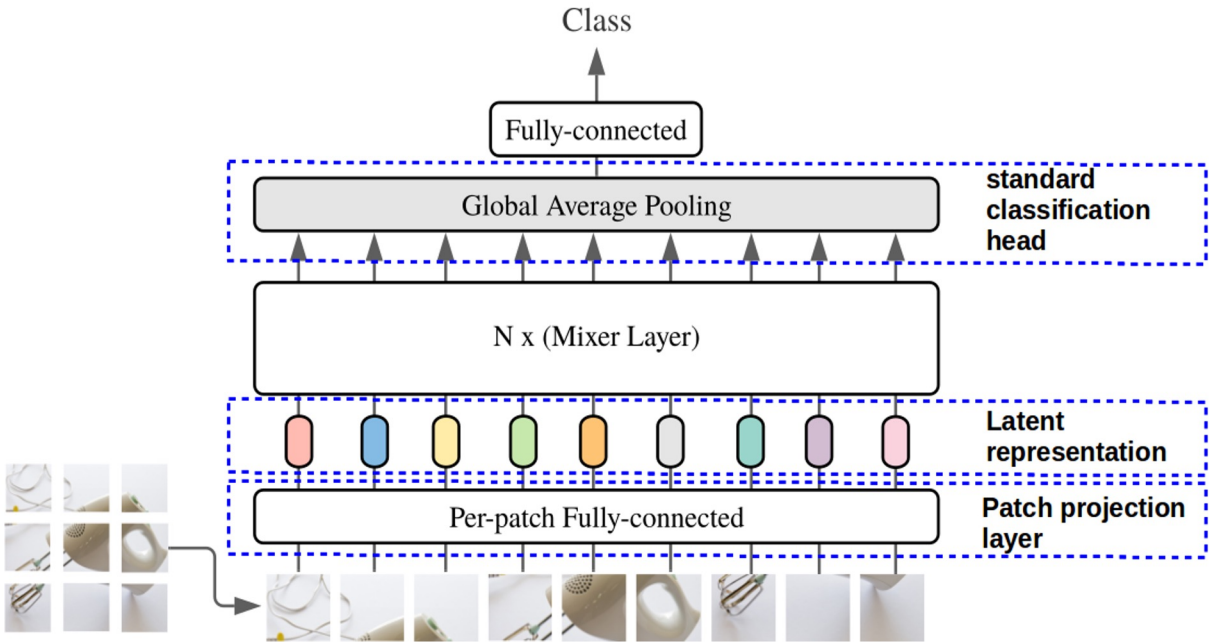


What is Mixer?

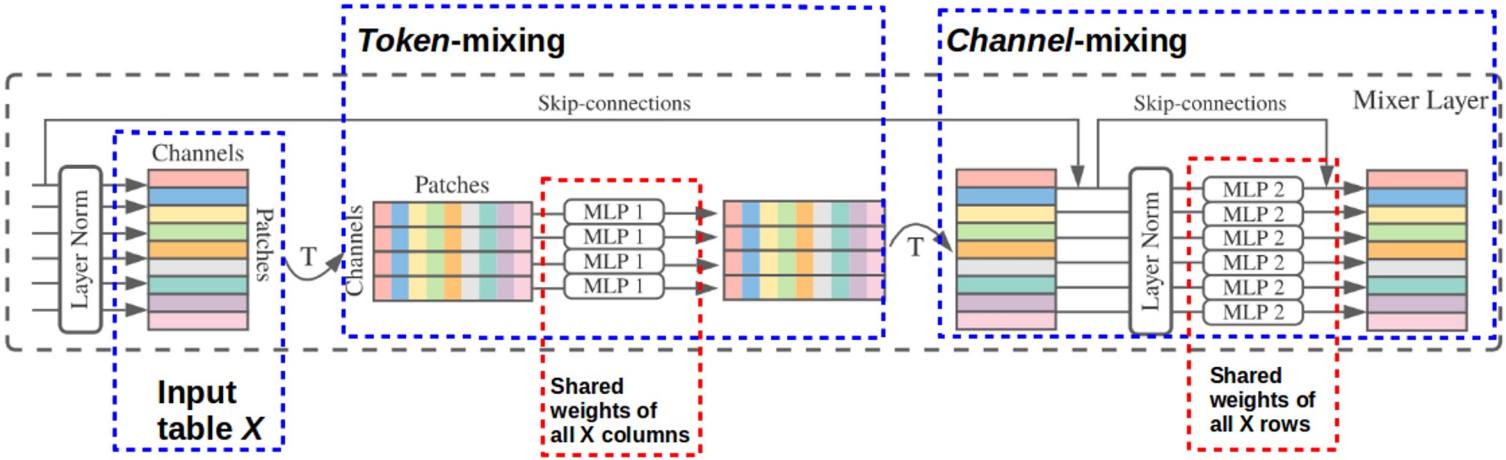
tokens mixing

channel mixing

MLP Mixer


Class
Fully-connected
Global Average Pooling
standard classification head
N x (Mixer Layer)
Latent representation
Per-patch Fully-connected
Patch projection layer

MLP Mixer


Token-mixing
Channel-mixing
Skip-connections
Skip-connections
Mixer Layer
Layer Norm
Channels
Patches
T
Patches
Channels
MLP 1
MLP 1
MLP 1
MLP 1
T
Layer Norm
MLP 2
MLP 2
MLP 2
MLP 2
MLP 2
MLP 2
Input table X
Shared weights of all X columns
Shared weights of all X rows

MLP Mixer

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

MLP Mixer

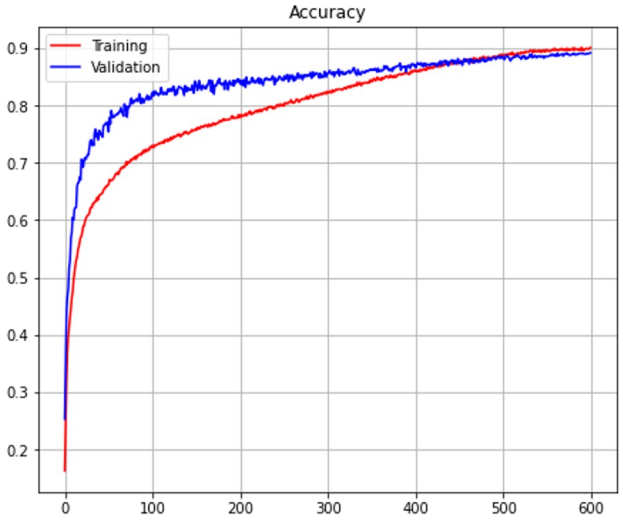
Expected Mixer results

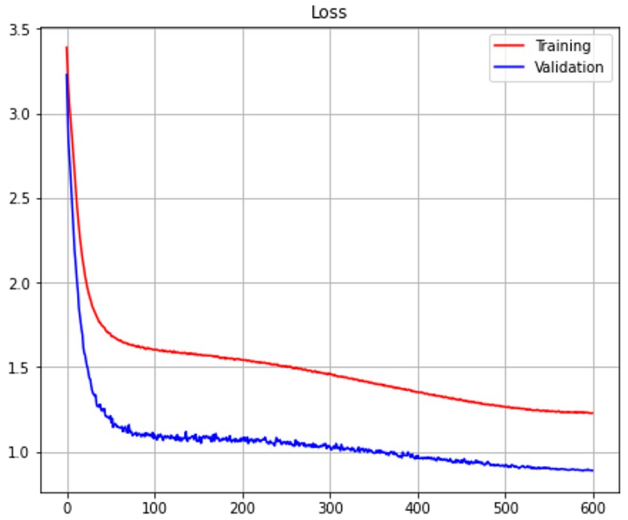
We ran the fine-tuning code on Google Cloud machine with four V100 GPUs with the default adaption parameters from this repository. Here are the results:

upstream	model	dataset	accuracy	wall_clock_time	link
ImageNet	Mixer-B/16	cifar10	96.72%	3.0h	tensorboard.dev
ImageNet	Mixer-L/16	cifar10	96.59%	3.0h	tensorboard.dev
ImageNet-21k	Mixer-B/16	cifar10	96.82%	9.6h	tensorboard.dev
ImageNet-21k	Mixer-L/16	cifar10	98.34%	10.0h	tensorboard.dev

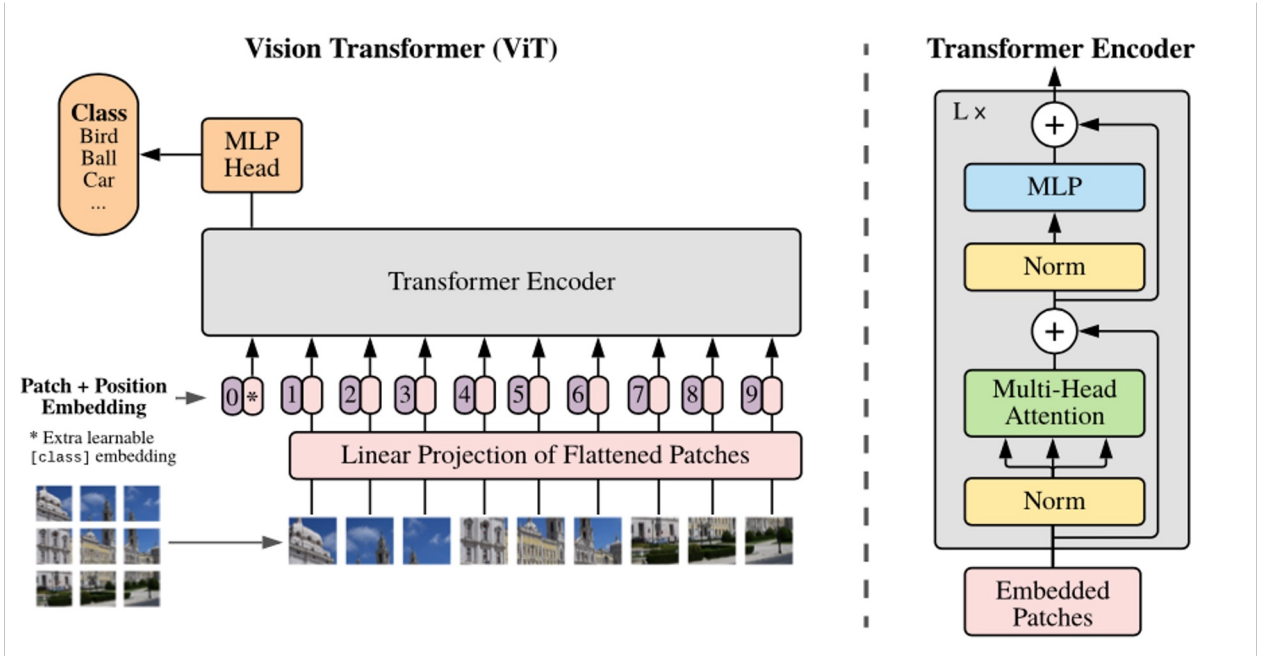
For Cifar10,
accuracy~96.6%(pretrained)
accuracy~89.15%(trained on cifar10 data

MLP Mixer

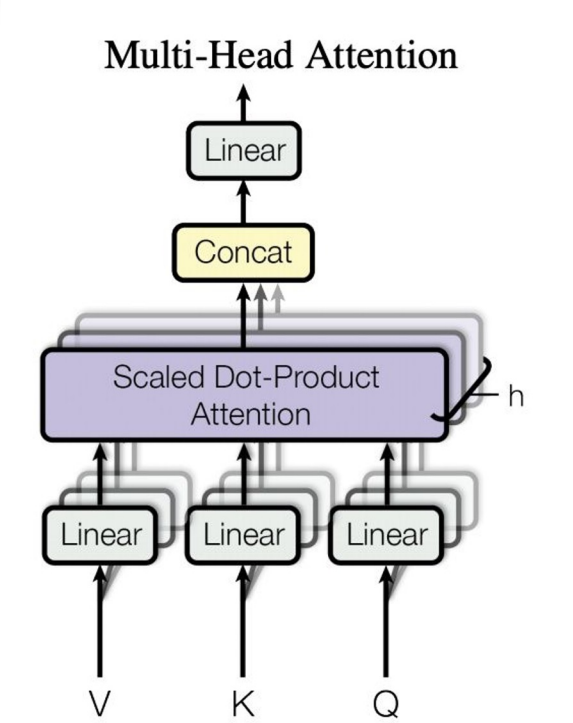

Accuracy
Training
Validation
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
100
200
300
400
500
600

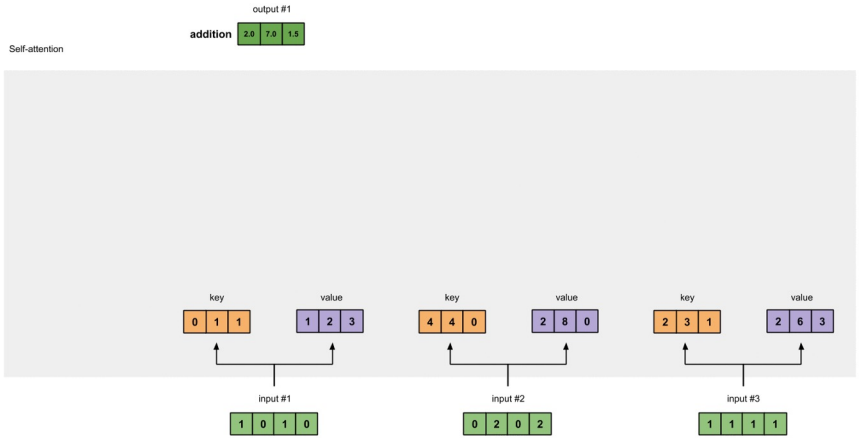

Loss
Training
Validation
3.5
3.0
2.5
2.0
1.5
1.0
0
100
200
300
400
500
600

Vision Transformer



Vision Transformer-Multi-Head Attention





Vision Transformer

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

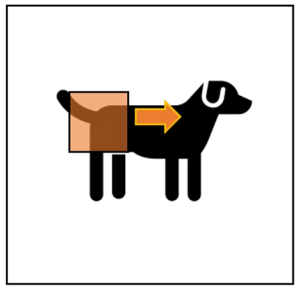
Table 1: Details of Vision Transformer model variants.

Vision Transformer

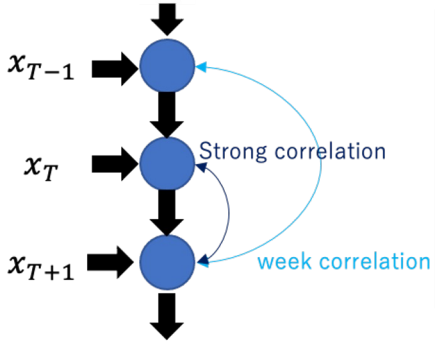
	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15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

Vision Transformer-Why more accurate?

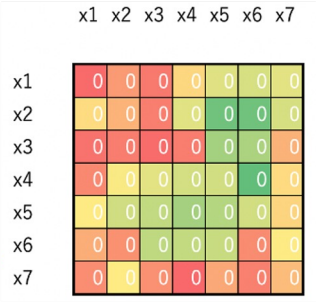
CNN



RNN



Self Attention



Summary

- Baseline CNN model
- Data augmentation
- Model modification
- VGG16
- MLP-Mixer
- ViT

Reference:

[1]Dosovitskiy A, Beyer L, Kolesnikov A, et al. An image is worth 16x16 words: Transformers for image recognition at scale[J]. arXiv preprint arXiv:2010.11929, 2020.

[2]Tolstikhin I O, Houlsby N, Kolesnikov A, et al. Mlp-mixer: An all-mlp architecture for vision[J]. Advances in Neural Information Processing Systems, 2021, 34.

[3]Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

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

Reference:

[5]https://towardsdatascience.com/recent-developments-and-views-on-computer-vision-x-transformer-ed32a2c72654

[6]https://mchromiak.github.io/articles/2021/May/05/MLP-Mixer/#.YnPmvC_4hN1

[7]https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

[8]https://neurohive.io/en/popular-networks/vgg16/