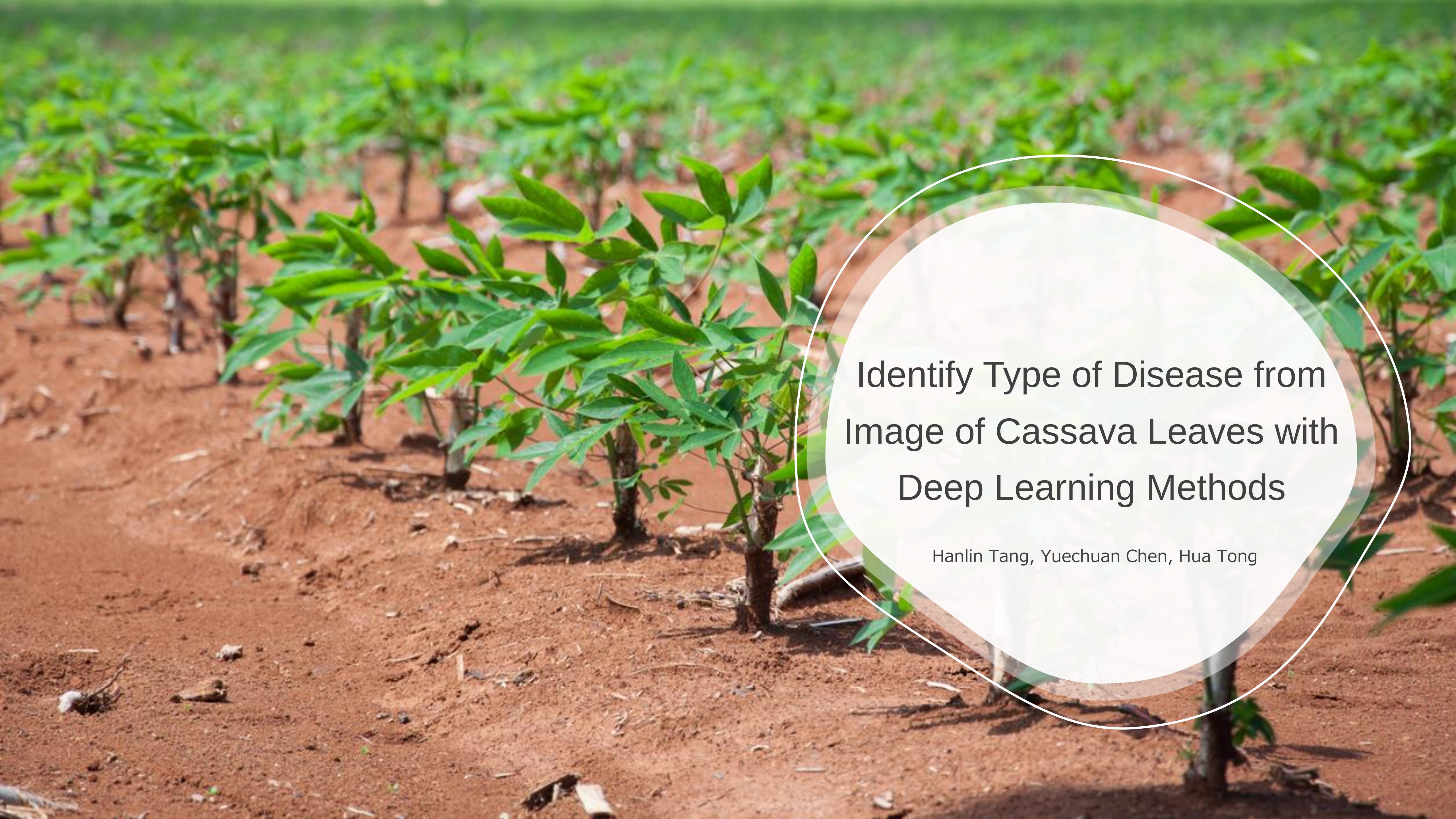


Identify Type of Disease from Image of Cassava Leaves with Deep Learning Methods

Hanlin Tang, Yuechuan Chen, Hua Tong

Contents

Background and Motivation

Background

Cassava, a major staple food in the developing world that plays a vital role in saving the world from hunger.

At least 80% of household farms in Sub-Saharan Africa grow it.

However, there are viral diseases that preventing good yields.

Diseases

Cassava Bacterial Blight (CBB)

Cassava Brown Streak Disease (CBSD)

Cassava Green Mottle (CGM)

Cassava Mosaic Disease (CMD)

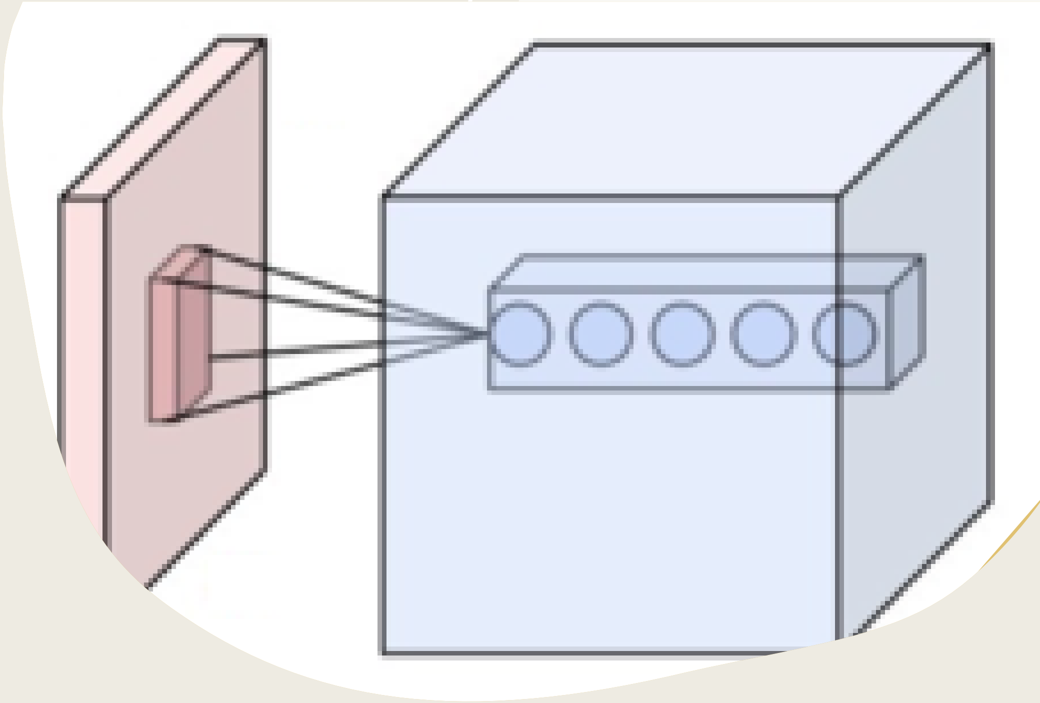
Motivation

- Existing detection methods require the involvement of experts' inspection, which suffers from being labor-intensive, costly, and sometime inaccessible.
- Meanwhile, find out the infected plants sooner could help stopping it from spreading.

- Thus, "identifying disease with image" comes to be an option.

Methods and Models

Models



- Convolutional Neural Network

AlexNet

VGGNet

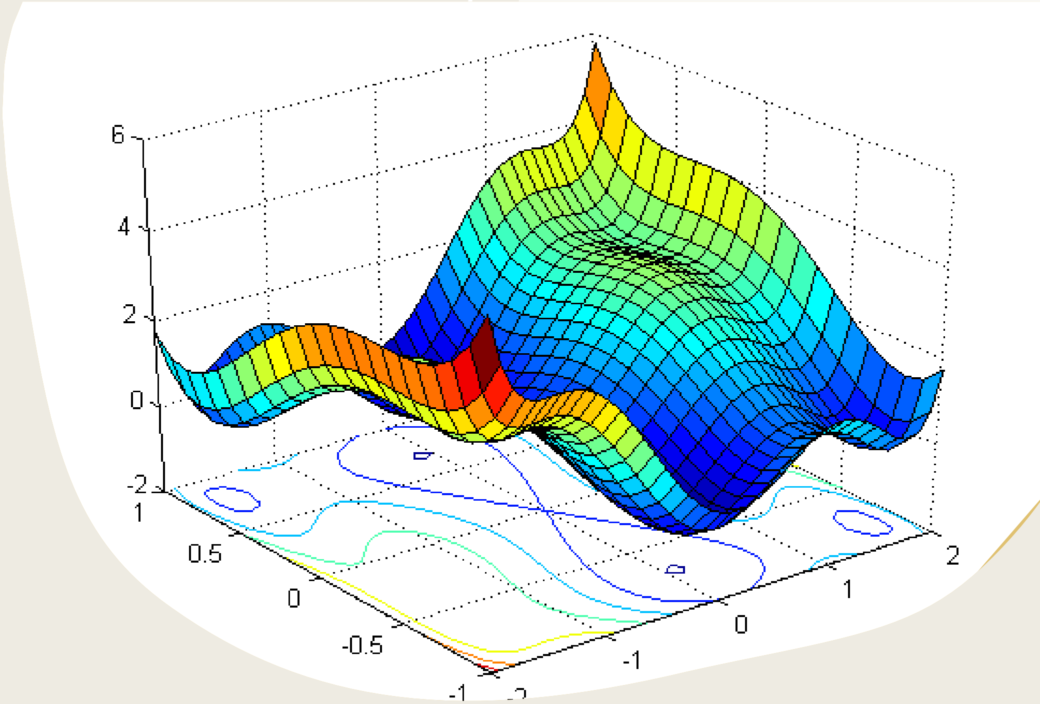
ResNet

GoogLeNet

ResNeXt

Wide ResNet

Loss function



- Cross-entropy

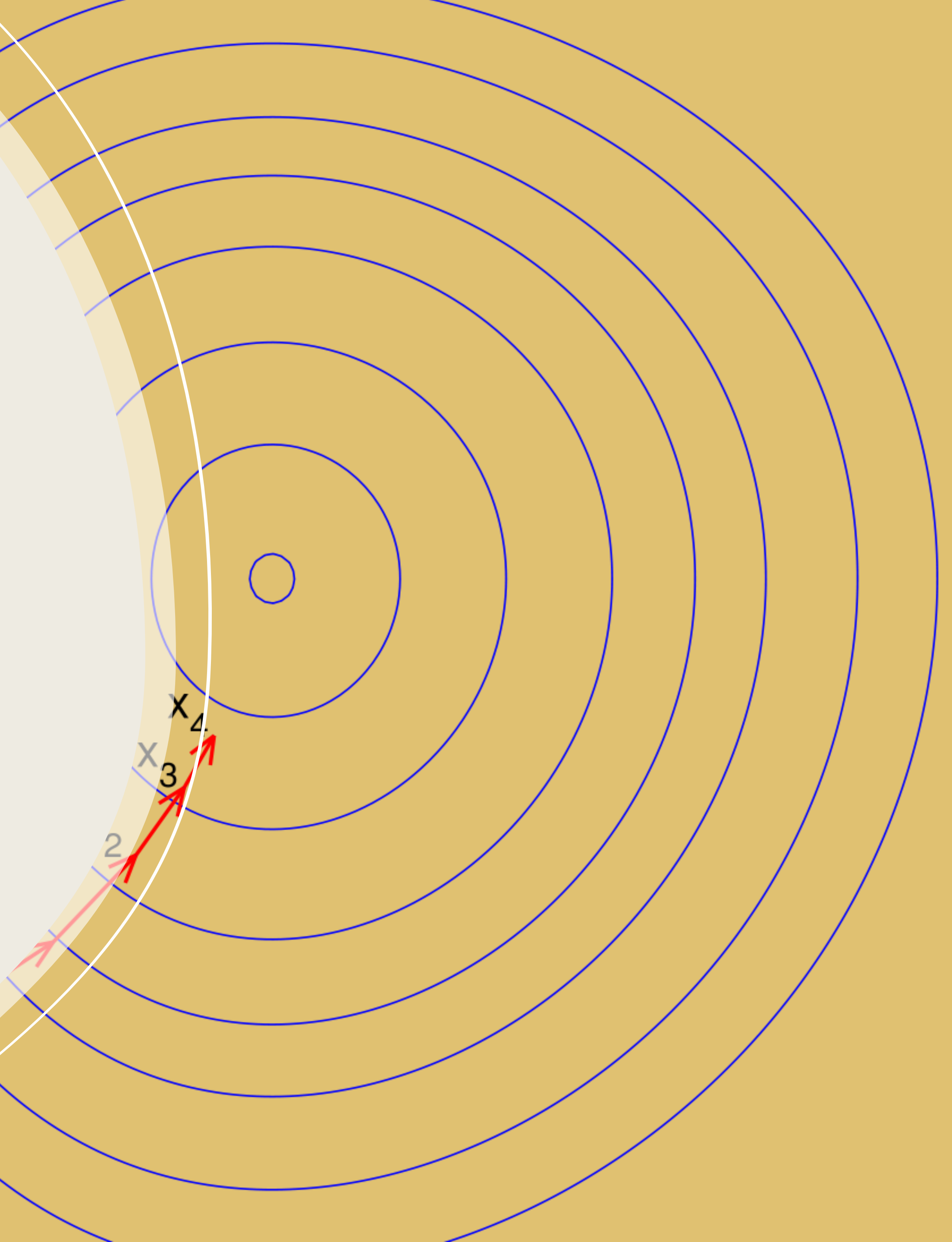
$$\text{cross-entropy} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{k} t_{i,j}\log(p_{i,j})$$

Optimizing

Stochastic gradient descent

Originally referred to the optimizing strategy that update parameters for each single sample.

Nowadays it is more often referred to the mini-batch gradient descent, which take a mini-batch instead of single sample.



Optimizing

Adaptive Moment Estimationan (Adam)

Algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

Optimizing

- Early Stopping

Early stopping, as the name stating, is to stop training earlier than our pre-determined epoch number. There are many criteria for doing early stopping, while most of them are based on the validation accuracy.

We designed such an early stopping strategy: “Stop if the max validation accuracy does not increase for 6 epochs, or does not increase for 2 epochs while train accuracy is larger than 99%.”

Experiment Details

Data Set

NUMBERS

- 80% training data (17118)
- 5% validation data (1070)
- 15% test data (3209)
- Around 15000 online test data (not visible)

EACH DATA

- An RGB 3 channel, 800x600 image
- During initial training, we resize to 400x300 for efficiency
- For pretrained models, we normalize with mean [0.485, 0.456, 0.406] and std [0.229, 0.224,0.225]

Environment

SOFTWARE

- Pytorch on python, including it's embedded models (torchvision.models).

PyTorch

- Overleaf for report, and Powerpoint for slides.

HARDWARE

- One GTX 1650, one RTX 2080, and one RTX 2080 Ti.

General procedure

INITIAL TUNING

- Load model from pytorch package
- Train at most 40 epochs with the early stopping setting
- Generally, use SGD with 0.001 learning rate or 0.0001 learning rate, depends on the result.

FINE-TUNING

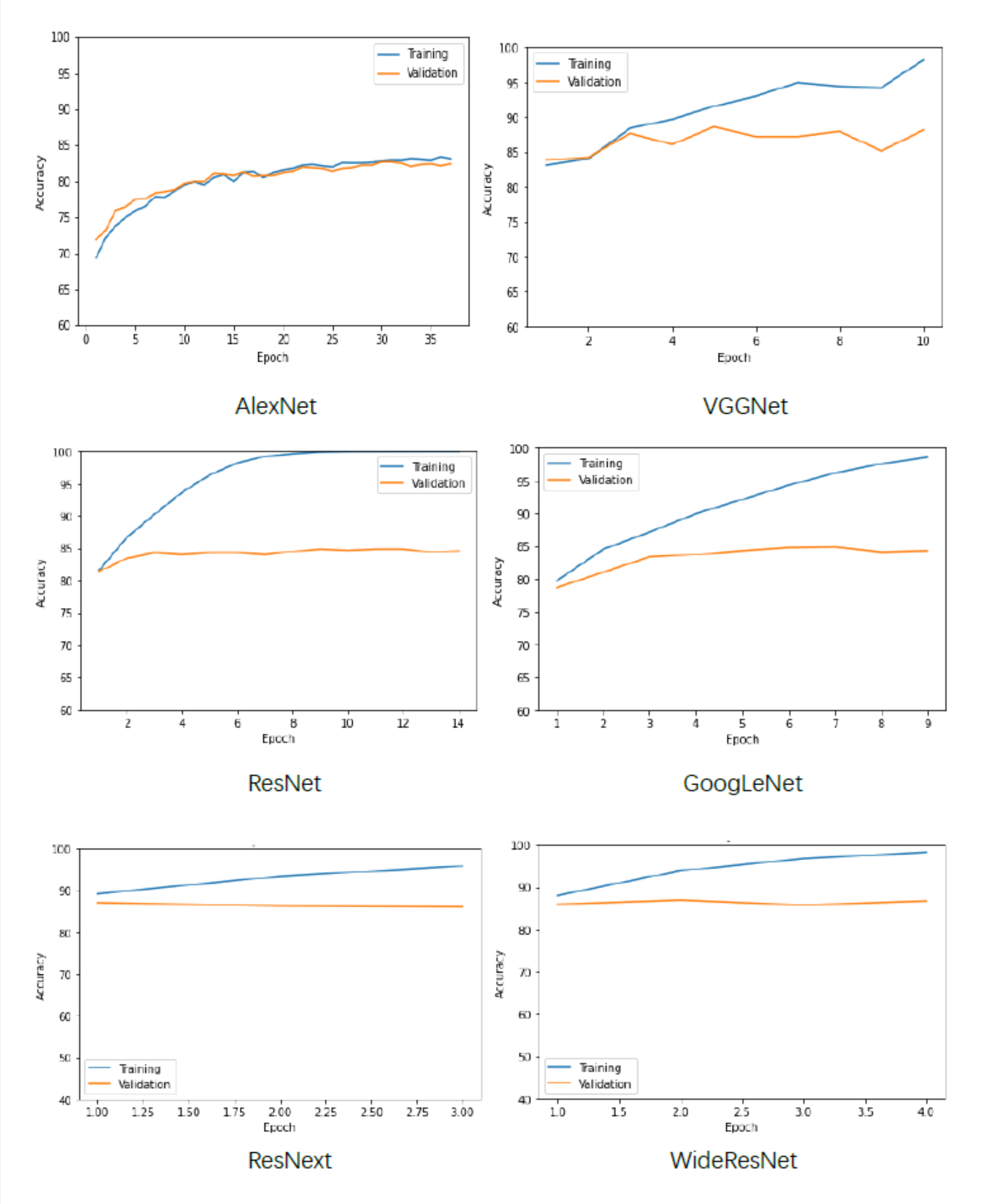
- Only tried on Resnet due to lack of resources.
- Make necessary change to the architecture
- Manually adjust learning rate
- Run less epoch each time and keep improving

Result and Summary

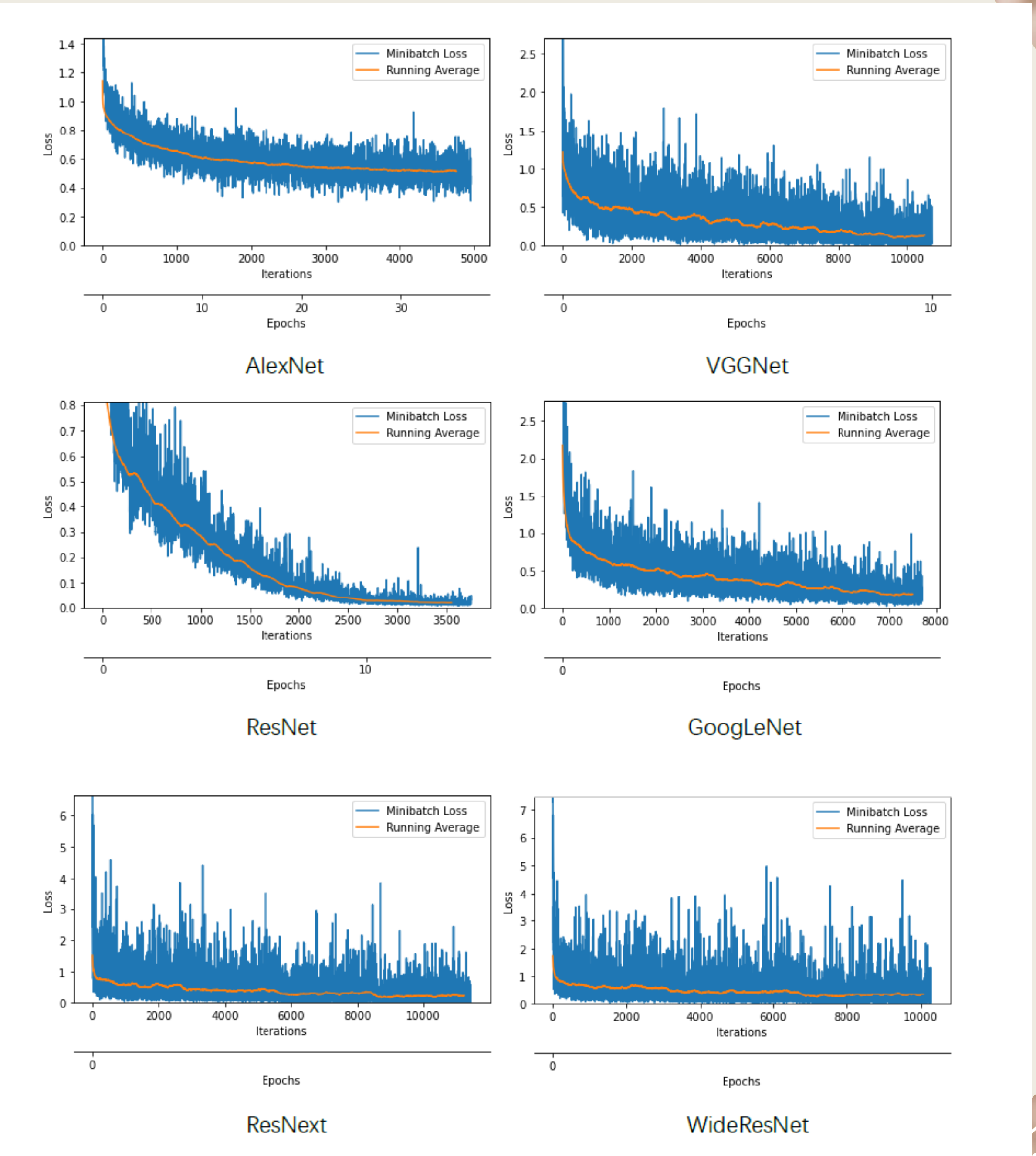
Accuracy

	Training acc	Validation acc	Local test acc	Online test acc
AlexNet	83.07 %	82.43 %	79.37 %	78.21 %
VGG16	98.18 %	88.13 %	85.63 %	84.72 %
Resnet34	99.99 %	84.58 %	82.08 %	80.55 %
GoogLeNet	98.59 %	84.30 %	82.61 %	82.48 %
ResNext	98.26 %	86.73 %	85.76 %	84.37 %
WideResnet	95.86 %	86.17 %	84.11 %	83.78 %

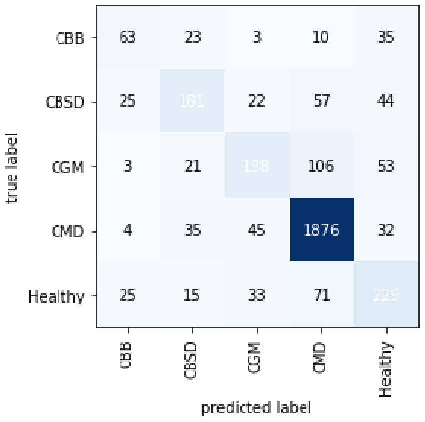
Training Accuracy



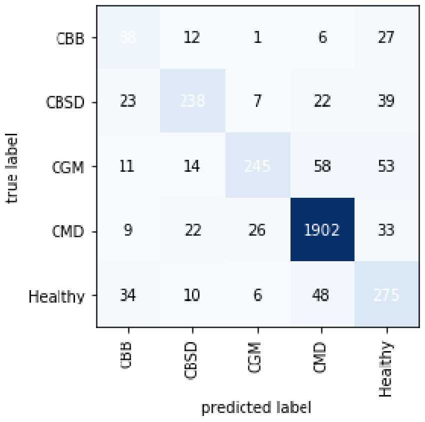
Mini-batch Loss



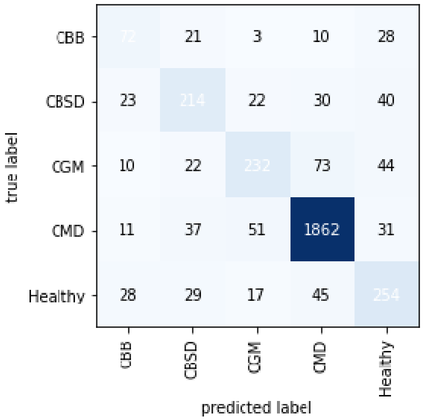
Confusion matrix



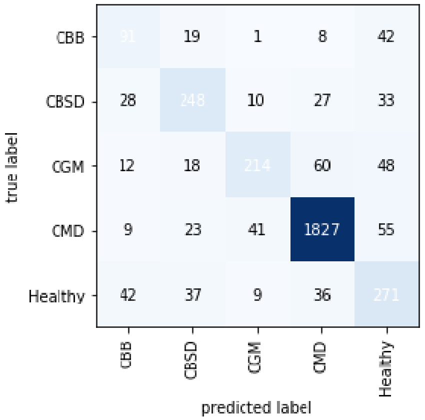
ALexNet



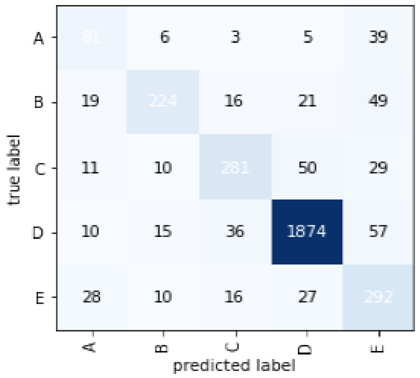
VGGNet



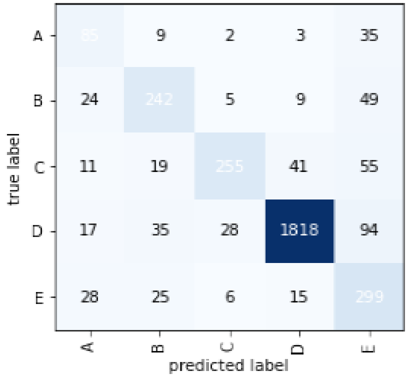
ResNet



GoogLeNet



ResNext



Wide ResNet

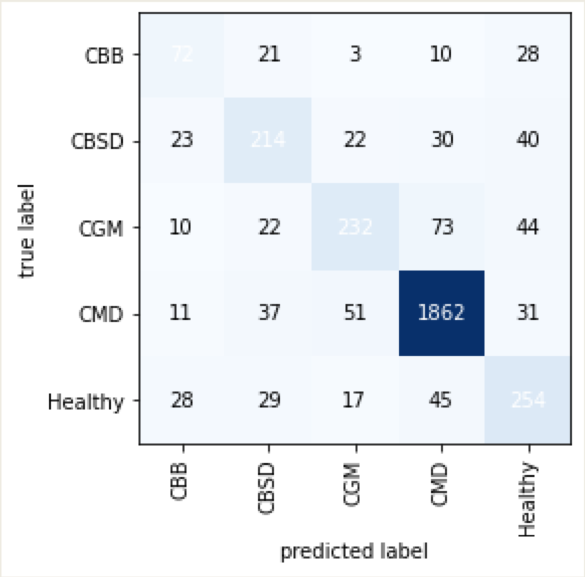
Fine-tuning model

PROCESS

- Due to the limitation of time and resource, we only managed to fine-tuning one model: Resnet.
- Input resized to 512x512 instead of 400x300
- Start with pretrained model
- Replace the last fc layer (2048,1000) with fc (2048,5).
- Trained with smaller mini-batch (8 instead of 16)
- After each epoch, manually decrease learning rate on acc fluctuating
- Eventually run 6 epochs

RESULT

- Train acc: 91.15 %, validation acc: 88.60 %
- Test acc: 87.19%, online test acc: 86.75 %



Summary

We practiced our skills in deep learning and meet the target we set at the beginning of the project.

We also learned the complexity of tuning and how cost it could be to run deep learning training.

The final result, though not outstanding, is very much usable and helpful.

Yet, more fine-tuning is expected if time and resource allowed.

Thank You

Hanlin Tang, htang79@wisc.edu

Yuechuan Chen, ychen959@wisc.edu

Hua Tong, htong24@wisc.edu

Github: https://github.com/Tonghua35/453-Final-Project