50.039 Deep Learning

Small Project Report

Information Systems Technology and Design (ISTD)

Amarjyot Kaur Narula	1003084
Gladys Chua Shi Ying	1003585
Pang Luying	1003631

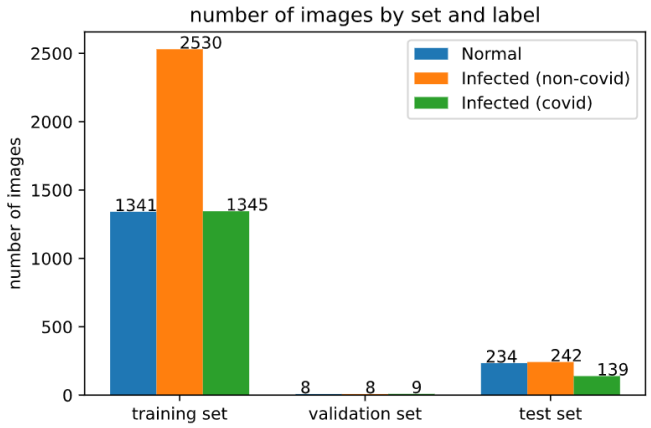
Table of Contents

Project Content and Instructions to Run Code

- Learning curve figures of different models are kept in the **learningCurve** folder
- Trained models are kept in **final_saved_models** folder, you can reload the saved models to obtain the learning curve and make prediction
- See **README.md** for detailed instructions to run the code

1. Dataset and Dataloader

Distribution of Dataset



There is an unequal distribution of classes in the training set and test set, hence this is an imbalanced classification. We can see from the bar chart above, there are more infected(non-covid) images as compared to the normal and infected(covid) images.

Data Processing Operations

One of the best practices for training a Neural Network is to normalize the data such that we will be able to obtain a mean close to zero (on average over the dataset). It was also observed that normalizing the data generally speeds up learning and leads to faster convergence.
Other operations are not crucial as our dataset is good enough for general classification.

2. Proposed Model

Expectations

The objective of this project is to propose, train and evaluate a Deep Learning model, which attempts to classify the X-ray images of patients and help doctors with the diagnosis of COVID or non-COVID pneumonia. We have trained 2 models using the three-classes and two binary classifiers architecture.

Differences Between the Two Provided Architectures

Binary classifier architecture

The first binary classifier classifies normal and infected images first and the second classifier classifies the covid and non-covid of the infected images. This model may give a better classification between non-covid infected and covid infected images as the second classifier learns more specific differences between non-covid infected and covid infected images.

Three-classes classifier architecture

The classifier classifies the images to one of the three mutually exclusive classes (normal, covid infected, non-covid infected) at once. In terms of code implementation, this model is easier to implement than the binary classifier as there is only one classifier here. However,

this model may give a higher percentage of predicting covid images as non-covid infected because the classifier may not be able to learn detailed differences between them as much as it would in the previous architecture.

Conclusion between the two architectures

We have tried out both architectures, and we found that with similar accuracy (around 70%), the binary classifier architecture gives a lower percentage of predicting covid wrongly. Detailed data are shown in the Performance Report section.

Our Choice of Architecture

We follow Resnet18 to structure our model at the beginning. Instead of using the residual block, we use dropout in our model. However, we found that the model is overfitting and does not give a proper learning curve, so we decide to reduce the layers of the model. In the end we found that 1-3 convolutional layers is good enough for classification.
Our basic structure is as shown,

```
Classifier(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=82944, out_features=2, bias=True)
  (drop): Dropout(p=0.5, inplace=False)
  (fl): LogSoftmax(dim=1)
)
```

We will add blocks in the middle if needed. Block structure is shown below:

```
(blk1): Block(
  (layer1): Sequential(
    (0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (layer2): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (dropout): Dropout(p=0.5, inplace=False)
)
```

Three-classes Classifier: Using the basic structure only.

```
Classifier(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=165888, out_features=3, bias=True)
  (drop): Dropout(p=0.5, inplace=False)
  (fl): LogSoftmax(dim=1)
)
```

- ➢ Since we focus on the central part of the image so we let padding = 0
- ➢ Max Pooling layer sets stride = 2 to reduce the number of parameters in the fully connected layer
- ➢ Dropout layer to avoid overfitting
- ➢ Use ReLU as activation function

- ➢ Kernel size = 5, to detect features in a small area
- ➢ Output channels of conv2d = 32:
  We have tried other numbers as well. Large numbers will cause overfit within 1 epoch, where training loss is much smaller than validation loss. Small numbers may not be able to learn the features well in one layer.

Binary classifier 1: Using basic structure only

```
Classifier(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=82944, out_features=2, bias=True)
  (drop): Dropout(p=0.5, inplace=False)
  (fl): LogSoftmax(dim=1)
)
```

- ➢ Output channels of conv2d = 16: the difference between normal and infected can be distinguished easily so we do not need very large numbers.
- ➢ Other settings are set with the same reasons as the Three-classes Classifier above.

Binary classifier 2: Basic structure + Block

```
Classifier2(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (blk1): Block(
    (layer1): Sequential(
      (0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
    )
    (layer2): Sequential(
      (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (fc1): Linear(in_features=82944, out_features=2, bias=True)
  (drop): Dropout(p=0.5, inplace=False)
  (fl): LogSoftmax(dim=1)
)
```

- ➢ Add blk1:
  Since the covid infected and non-covid infected images are similar to each other, we need more layers to learn more detailed features to classify them.
- ➢ Other settings are set with the same reasons as the Three-classes Classifier above.

Value of Mini-Batch Size

Batch Size = 64:

Considering the amount of data (5216) in the training dataset, A large batch size such as 256 or 512 may lead to poor generalization, and a small batch size such as 4 or 8 may cause the training time to be long. Each epoch will input about 82 batches of data to the model with batch size 64, so that the model can converge with an appropriate time.

Choice of Loss Function
Loss function = NLLLoss():
Normally we will use CrossEntropy as a loss function for image classification. Since the last layer of the model is log softmax, we use NLLLoss() as our loss function, which will have the same effect as applying CrossEntropy without activation function at the last layer of the model.

Choice of Optimiser
Optimizer = Adam(lr=0.001):
We use Adam optimizer so that the model will not stop learning further when it encounters saddle points or local minima. An extremely small learning rate may cause the convergence process to be slow. A large learning rate may prevent convergence to the minima by oscillating around so we choose to start with a small learning rate and an adaptive learning rate optimizer Adam.

Choice of Model Parameters Initialisation
Kaiming_uniform_ initialization:
This is default initialization for conv2d in pytorch, where we will not have zero or any constant value weight initialization. Constant value weight initialization may cause all the neurons to learn the same features, which may cause the model to learn nothing.

3. Analysis

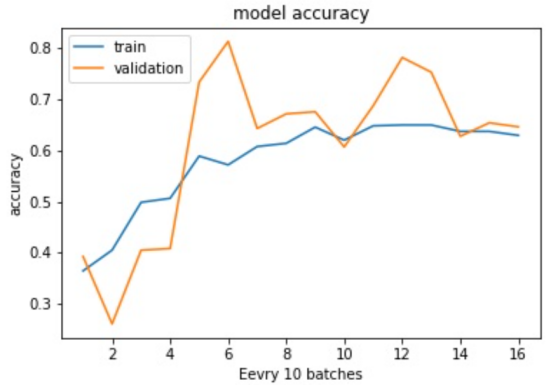
Challenges in Differentiating between Non-Covid and Covid X-rays
It is expected to be more difficult to differentiate between non-covid and covid x-rays, rather than between normal x-rays and infected (both covid and non-covid) people x-rays as both covid and non-covid have symptoms of pneumonia which can be detected through the chest x-ray.
Currently, the diagnosis for covid19 widely used is through the detection of the virus's genetic code, known as nucleic acid where a sample is taken by swabbing the nasal passages or throat. To find evidence of the virus, researchers use Polymerase Chain Reaction (PCR) to copy and amplify any segments of viral genetic code found in the sample, which makes it easier to detect. Hence, the chest x-rays are not the primary source of detecting covid by the doctors which suggests that it is expected to have difficulties for differentiating between non-covid and covid x-rays.

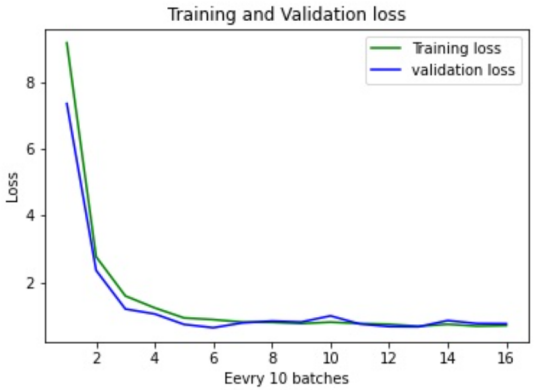
High overall accuracy vs Low True Negatives/False Positives
It would be better to have a model with low true negatives or false positives rates on certain classes than a model with high overall accuracy. In this project, we are predicting whether a person has covid-19, thus, we will be able to tolerate false positives but not false negatives. As the detection of the covid class is more important than the normal and infected (non-covid) classes.

4. Performance Report

Three-classes classifier

- Saved in /final_saved_models/three_classes2.1.pt
  (You can reload the model to get the same result)
- Testing Accuracy: 72.0%
- Learning Curve: (acc2.1.jpg & loss2.1.jpg)



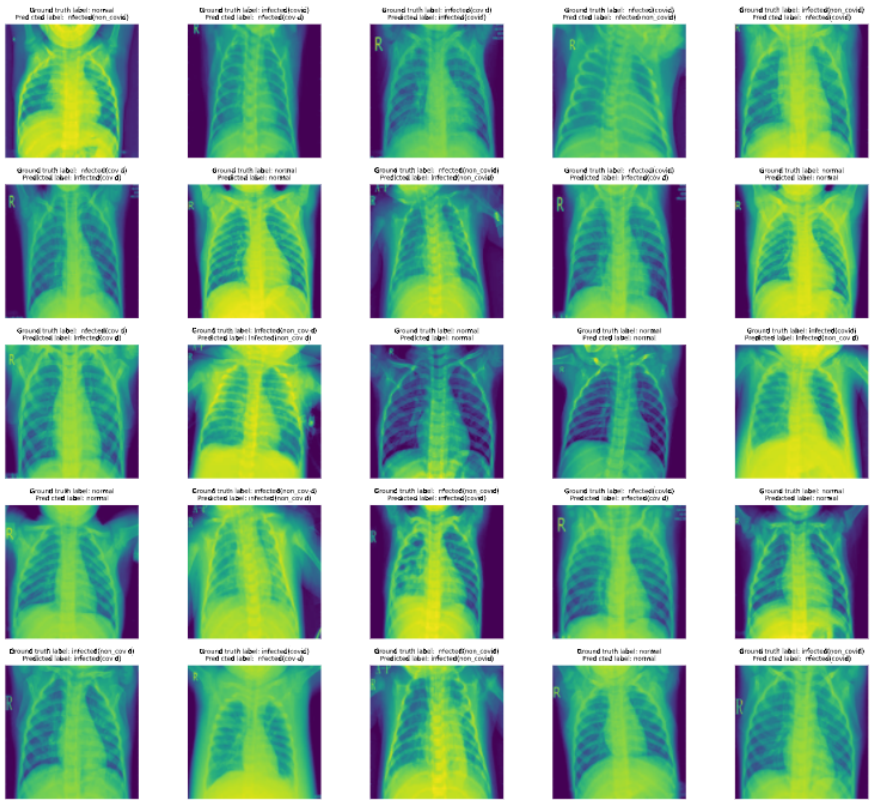


- Confusion Matrix (validation set):

	Predicted Normal	Predicted Infected (non_covid)	Predicted Infected (covid)
Actual Normal	7	1	0
Actual Infected (non_covid)	0	4	4
Actual Infected (covid)	0	2	7

The percentage of predicting covid wrongly: 22.2%

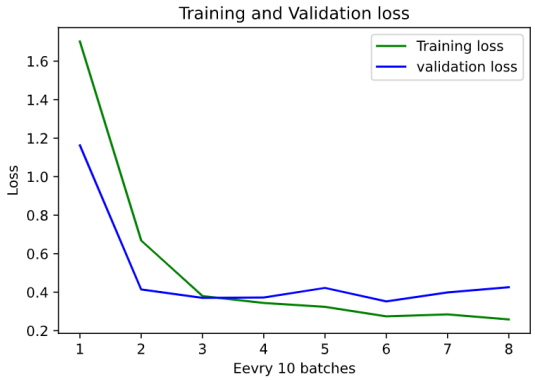
- Predicted validation image (check mainCode.ipynb for clear version)

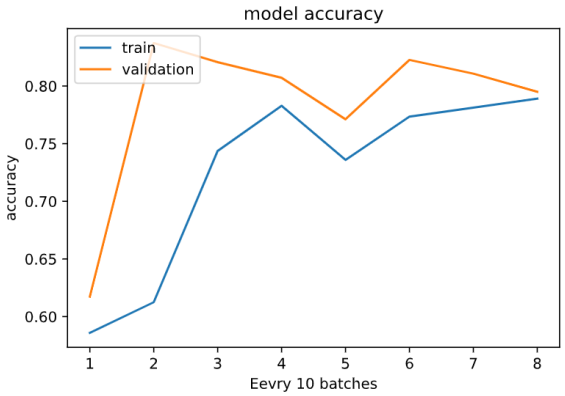


Two Binary Classifier

1. First Binary Classifier (normal vs infected)

- Saved in /final_saved_models/first_binary_classes1.pt
  (You can reload the model to get the same result)
- Learning Curve: (binary1_acc1.jpg & binary1_loss1.jpg)



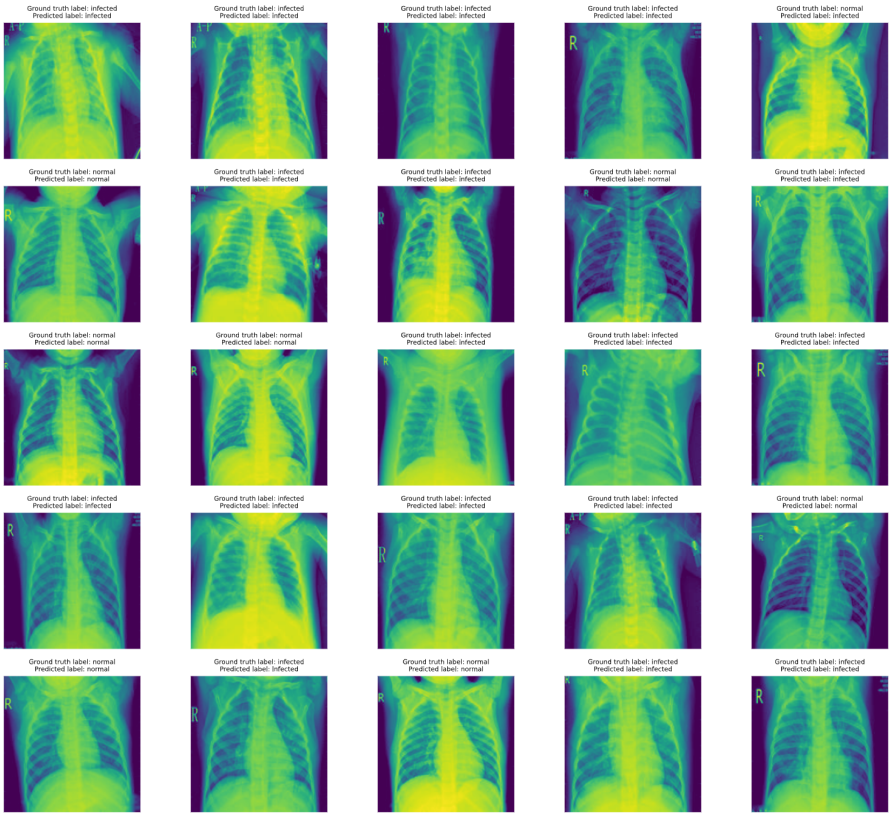


- Confusion Matrix (validation set):

	Predicted Normal	Predicted Infected
Actual Normal	7	1
Actual Infected	0	17

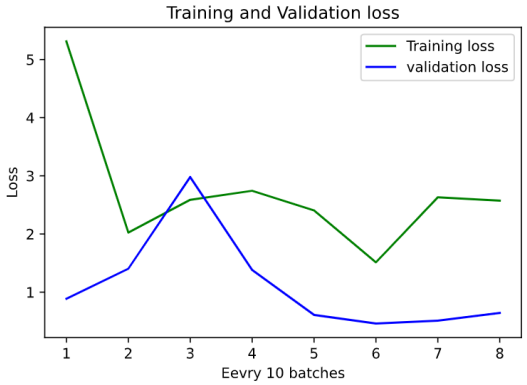
The percentage of predicting infected wrongly: 0.0%

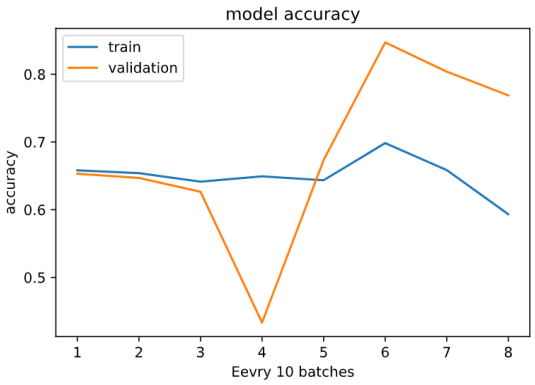
- Predicted validation image (check mainCode.ipynb for clear version)



2. Second Binary Classifier (non-covid vs covid)

- Saved in /final_saved_models/first_binary_classes1.pt
  (You can reload the model to get the same result)
- Learning Curve: (binary2_acc1.jpg & binary2_loss1.jpg)





- Confusion Matrix (validation set):

	Predicted Infected (non_covid)	Predicted Infected (covid)
Actual Infected (non_covid)	3	5
Actual Infected (covid)	1	8

The percentage of predicting covid wrongly: 1/9=11.1%

- Predicted validation image (check mainCode.ipynb for clear version)