University of Sydney

ELEC5622 Signals, Software and Health– Project Report 2

HangYu Chen (490075948)

Table of Contents

Abstract *1*

1 Section 1 Aims and Background *1*

2 Section 2 Contribution of team members *2*

3 Section 3 Methodology *2*

3.1 Data preparation & processing **2**

3.1.1 Data loading and Dataset setting up 2

3.1.2 Data Augmentation and Dataloader setting up 3

3.2 Network set up **5**

3.3 Loss and accuracy measurement methods **6**

3.4 Hyperparameters choosing **0**

3.5 Visualization and Performance matrix **1**

4 Section4 Results *3*

4.1 Data preparation **3**

4.2 Network setting up **4**

4.3 Hyperparameters selection **4**

4.4 Visualization and performance matrix **6**

4.4.1 Visualization 6

4.4.2 Performance matrix 7

5 Section5 Discussion and Conclusion *9*

References *10*

Abstract

Efficient Human Epithelial-2 cell image classification can facilitate the diagnosis of many

autoimmune diseases, which plays an important role in today's clinical applications (Zhimin et.al, 2017) [9].

In this project, we will use CNN so that we could extract more high-level features for cell images classification. In addition to simply training and using edited AlexNet as our network, some introduction to AlexNet structure, Data Augmentation method, hyper-parameters selection and different methods for showing the final accuracy are mentioned in this report.

The final accuracy is 0.880 using the own-defined accuracy function and 0.875 using the performance matrix.

1 Section 1 Aims and Background

HEp-2 cells are used for the identification of antinuclear autoantibodies (ANAs). They are widely used in today's clinical medicine. HEp-2 cells allow for recognition of over 30 different nuclear and cytoplasmic patterns, which are given by more than 100 different auto-antibodies (Petra .et.al, 2002) [10].

The identification of the patterns used to be done manually which need a person inspecting the slides with a microscope. However, as we have pre-knowledge about the CNN, we aim to build up a neural network so that this process could be done automatically by the computer. In this case, we could not only save the time needed for researchers and doctors, but also could minimize the error rate such as misjudgment which is commonly occurs in manually testing. According to Petra (2002), "the differnece between showing and naming" is a well-known problem in image interpretation which makes manually testing hard to follow by novices and even for an experienced physician, it is often hard to decide the right class [10].

In this project, we have a set of HEp-2 cell images hosted by International Conference on Pattern Recognition in 2014 which has already been divided into three datasets (training (8701 images), testing (2720 images) and validation (2175 images)).

The whole process will be:

1) Do the data pre-processing (i.e. setting up datasets and dataloaders, data augmentation).

2) Setting up the AlexNet network and changing the number of final fully-connected layer features.

3) Setting up Loss and Accuracy calculating functions, setting up an optimizer, training our

network as baseline.

4) Do the hyper-parameters selection (i.e. batch size, learning rate, epoch).

5) Do the result visualization and using performance matrix to analysis the result again.

2 Section 2 Contribution of team members

There are four people in our group. Each of us has do some checking and updating tasks for the whole project, but the main task for each person is:

- Hangyu Chen is responsible for setting up the Dataloaders, setting up the AlexNet network, training for batch size=32 and visualize the results, also make a performance matrix for analyzing the accuracy.
- WeiPeng Cui is responsible for hyper-parameters choosing when training the network for batch size = 8.
- ManChun Li is responsible for hyper-parameters choosing when training the network for batch size = 6.
- Aradhika Guha is responsible for hyper-parameters choosing when training the network for batch size = 4.

3 Section 3 Methodology

In this part, the methods that how we approach to find the result and the knowledge behind will be mentioned. The whole process is implemented in Google Colab, and the detail code could be found in “ELEC5622Project(combine).ipynb”.

3.1 Data preparation & processing

3.1.1 Data loading and Dataset setting up

1) We firstly save the zipped image file into our own Google drive. Then unzipping it into the Colab. After that, based on the gt_training.csv file, we could assign the class number for each image in the training, testing and validation files. In this case, each file could be considered as a multi-dimension list which looks like:

e.g., Training_set = [[‘a.png’, class_num], [‘b.png’, class_num], …]

2) Because there are six different kinds of images, the relationship between class number and image classes will be:

Class 0 → Homogeneous; Class 1 → Speckled; Class 2 → Nucleolar; Class 3 → Centromere; Class 4 →NuMem; Class 5 →Golgi.

Example for making NuMem and Golgi class in validation dataset:

```
for i in range (1728,2065):
  vallib5.append(['/content/our_data/validation/validation/'+allValDataName[i],4])
print(len(vallib5))
print(vallib5[0])
print(vallib5[-1])

for i in range (2065,2175):
  vallib6.append(['/content/our_data/validation/validation/'+allValDataName[i],5])
print(len(vallib6))
print(vallib6[0])
print(vallib6[-1])

# The output will be: [[imageName1, class_name], [imageName2, class_name], ...]

# class 1 means: Homogeneous
# class 2 means: Speckled
# etc.
```

Figure 1 Data loading

And the length, the first item and the last item will be printed:

```
337
['/content/our_data/validation/validation/10666.png', 4]
['/content/our_data/validation/validation/12872.png', 4]
110
['/content/our_data/validation/validation/12882.png', 5]
['/content/our_data/validation/validation/13594.png', 5]
```

Figure 2 Items in dataset

3.1.2 Data Augmentation and Dataloader setting up

The aim to do data augmentation is avoiding the overfitting problem, which is one of the drawbacks of CNNs. According to Connor & Taghi (2019), overfitting occurs when a network learns a function with very high variance such as to perfectly model the training data. However, many application domains such as medical image analysis do not have access to big data [1]. In this case, data augmentation is a method which we could artificially expand our datasets by creating modified versions of images.

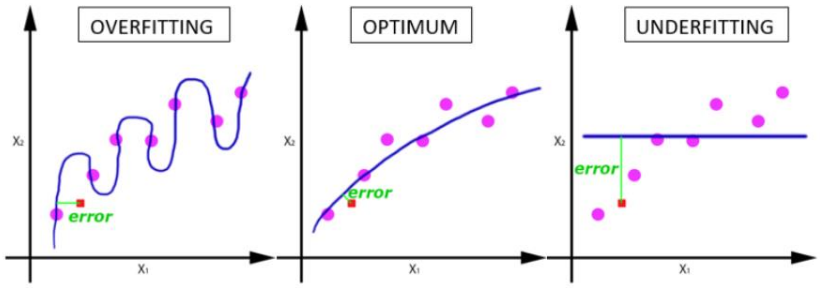

Figure 3 Overfitting

In our project, two methods for images augmentation are applied:

1) Random Horizontal Flip

2) Normalization

1) According to Jason (2019), an image flip means reversing the rows or columns of pixels in the case of a vertical or horizontal flip respectively. And in our project, we set the possibility for random horizontal flip as 0.5.

2) The reason we want to normalize is that neural networks prefer floating point values within a small range, which helps in making training a lot faster (Avinash, 2018) [2].

Firstly we used .Tosensor() to change the intensity value of image in range (0-1). And after we apply Normalize function, this value could be a float number in the range of (-1,1). The formula we calculate the final value after nomalizing is:

image = (image-mean)/std

In this case, if we use transform.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)), the minimum value in range (0,1) will become (0-0.5)/0.5 = -1; The maximum value 1 will become (1-0.5)/0.5 = 1

```
transform1 = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean = (0.5, 0.5, 0.5), std = (0.5, 0.5, 0.5)),
    transforms.RandomHorizontalFlip(p=0.5)
    ]
)
```

Figure 4 Data Augmentation

And during the Data Augmentation process, we also changed the size of the images into [3, 224, 224], which represents the Channel, Height and Width respectively. The reason is that we will use AlexNet as our network, which normal inputs are [224, 224] RGB images.

```
self.resize = torchvision.transforms.Resize((224, 224))
self.apply_resize = apply_resize
```

Figure 5 Image resize

```
image = PIL.Image.open(self.image_paths[index]).convert('RGB') # read the image
image = transform1(image)
image = transforms.ToPILImage()(image).convert('RGB')
```

Figure 6 Covert to RGB

And as we used pytorch for processing our data, we need to create DataLoader objects interfaces to serve up training or test data. According to James (2020), a DataLoader object fetches data from a Dataset and serves the data up in batches.[3]

3.2 Network set up

Our neural network structure is based on the pre-trained AlexNet network. The only difference is that we changed the output features of the **last fully connected layer to 6** because we only have 6 difference classes for Hep-2 images for classification.

AlexNet convolutional neural network was introduced in 2012. It was firstly used for image classification and proved that a deep convolutional neural network consisting of 5 convolutional layers and 3 fully connected layers could classify images efficiently and accurately (Richmond,2020) [4].

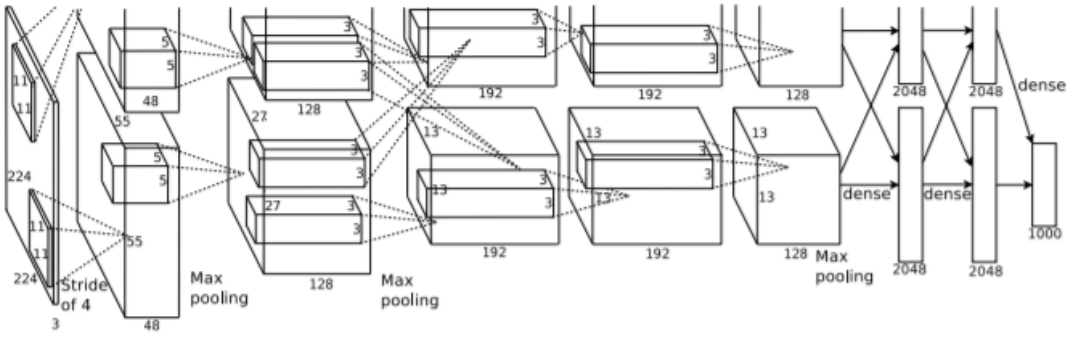

Figure 7 AlexNet structure

We could check the structure of the network by directly print it out. The result is shown in next phase.

One unique characteristic of AlexNet is that ReLU function are used.

ReLU is one of the activation functions which performed for the output of the prior convolution layer. It ensures that values within the neurons that are positive their values are maintained, but for negative values, they are clamped down to zero (see Figure below).

The benefit of ReLU compared with other activation functions is that it could make sure that the training process could be accelerated as gradient descent optimization occurs at a faster rate (Richmond,2020) [4].

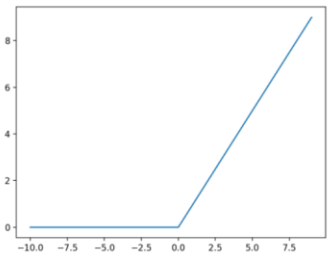

Figure 8 ReLU function

3.3 Loss and accuracy measurement methods

In this project, we used **two different methods** to show the final accuracy after training.

- One is that, we used our own defined function, which calculated the number of images which is correctly be predicted and divide it by the total number of images in the datasets.
- The other one is that we learned to use performance matrix with some scores to represent our accuracy.

And for the loss function, we used CrossEntropy Loss in pytorch.

1) Cross Entropy Loss

According to Jason (2019), the Cross Entropy loss is measure of the difference between two probability distributions for a given random variable or set of events [5].

If we consider a target or underlying probability distribution P and an approximation of the target distribution Q, then the cross-entropy between two probability distributions, such as Q from P could be represented as: H(P,Q), where H() is the cross-entropy function, P may be the target distribution and Q is the approximation of the target distribution Jason (2019) [5].

Then, H(P,Q) could be calculated by $H(P,Q) = -\sum_{x \in X} P(x) * \log(Q(x))$

Where P(x) is the probability of the event x in P, Q(x) is the probability of event x in Q.

The objective is always to minimize the loss. The lower the loss the better the model. In this case, as we used pytorch, **an optimizer** could help us finding the minimum loss.

According to Sanket.D (2019), optimizer is considered as algorithms to reduce the loss and make the result accurate during training by changing the network attributes (weights, learning rate. etc) [6]. If we consider the loss as a curve, optimizer would be used to help us finding the minima point.

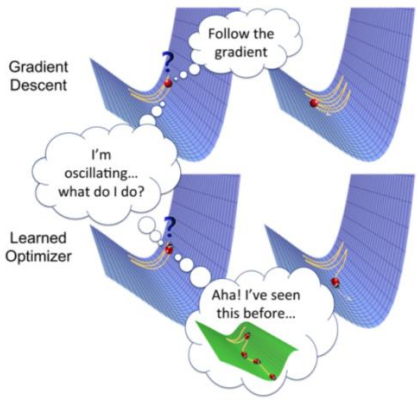

Figure 9 Optimizer

2) Accuracy functions

(a)

Our own defined accuracy function is implemented as below, and it is also normally used in today's deep learning projects.

It is implemented by calculating the number of images which is correctly predicted in a DataLoader. And in order to make the result more persuasive, we choose 128 as the batch size, which is large enough so that there will be more images from different classes in it.

```
def testset_precision(net, testset): # This function is used for calculate the accuracy for the testset
        dl = DataLoader(testset, batch_size = 128) # load test set
        total_count = 0    # total numbers of test set
        total_correct = 0 # total correct predictions of test set
        for data in dl:    # iterate test set
                inputs = data[0].cuda()    # input data, such as images, etc
                targets = data[1].cuda()      # ground truth labels
                outputs = net(inputs)        # get the outputs from network
                predicted_labels = outputs.argmax(dim=1)    # obtain the class with highest score as prediction
                comparison = predicted_labels == targets    # get a [True, False,...] matrix. True means correct prediction
                total_count += predicted_labels.size(0)      # accumulate the test number
                total_correct += comparison.sum()            # accumulate the number of correct predictions

        return int(total_correct) / int(total_count)
```

Figure 10 own-defined accuracy function

(b)

We then used performance matrix to represent the accuracy for the result. This part will be mentioned in the next part in detail, which is **part 3.5**.

3.4 Hyper-parameters choosing

In this part, we will compare the batch size, learning rate and epoch which will influence the training result of our network.

1) batch size

According to Jason (2019), the number of examples from the dataset used in the estimate of the error gradient is called the batch size [11].

A batch size of 16 means there are 16 samples from the dataset will be used to estimate the error gradient at the same time before the model weights are updated.

The more training examples used in the estimation, the more accurate this estimate will be and the more likely that the weights of the network will be adjusted in a way that will improve the performance of the model [11]. The batch size could be in range of 1 to 100.

However, batch size usually depends on the size of the dataset. Although small batch sizes are noisy, they are easier to fit one batch worth of training data in memory (i.e. when using a

GPU).

2) learning rate:

As we need to do the Gradient Descent so that we could find the minima of a curve (e.g. loss function) in order to get the best results. The learning rate is used for gradient descent when we are training our model. However, if the value is too small, it means there will be many updates before we reach the minima point. In this case, it will take a long time to get the result. And if the value is too large, it will cause drastic updates, which will finally lead to divergent behaviors so that we could miss the minimum point (Jeremy Jordan, 2018) and the network cannot be converged [7].

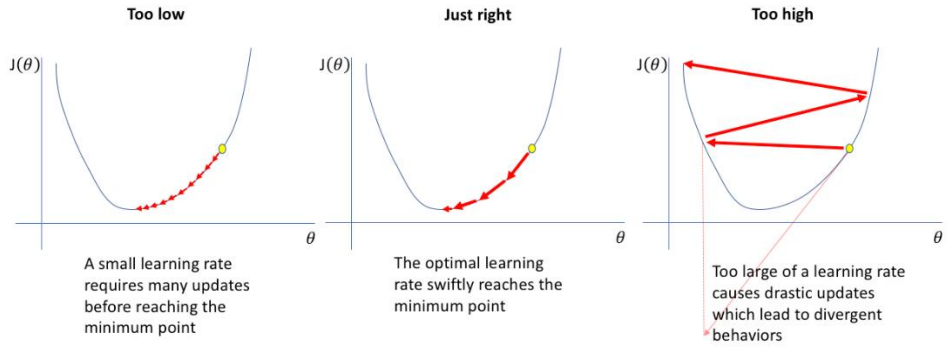

Figure 11 Learning rate effects

3) Epoch

Epoch is the iteration times need for training the network, we could say the network is converged if it achieves a state during training in which loss settles to within an error range around the final value, which means the network will not be improved if we continue training (Daniel & David, 2019) [8].

3.5 Visualization and Performance matrix

3.5.1 Visualization

We could visualize some images with their GroundTruth and Prediction labels after training.

The implement method is quite straightforward.

torch.argmax() will return the indices of the maximum value of all elements in the input tensor.

```
# Check the prediction of the first batch

X_actual = next(iter(train_dataloader))
Y_predic = torch.argmax(net(X_actual[0].cuda()), dim=1)
print(X_actual[1])
print(Y_predic)
```

```
tensor([1, 3, 5, 2, 3, 0, 1, 4, 0, 1, 1, 3, 2, 0, 2, 0, 0, 3, 5, 1, 3, 3, 4, 1,
        3, 1, 2, 2, 0, 3, 5, 5])
tensor([0, 3, 5, 2, 3, 0, 1, 4, 1, 1, 1, 3, 2, 0, 2, 0, 0, 3, 5, 0, 3, 3, 4, 1,
        3, 1, 2, 2, 4, 3, 5, 5], device='cuda:0')
```

Figure 12 Labels printed

Then, using imshow() to show the images and set the GroundTruth & Prediction labels as their title.

```
[ ]
    fig, ax = plt.subplots(nrows=1, ncols=8)   # Visualization
    plt.subplots_adjust(left=8, right=10, wspace=1)
    for idx in range(8):
        ax[idx].imshow(X_actual[0][idx].permute(1,2,0))
        ax[idx].axis('off')
        ax[idx].set_title(f'GroundTruth: {X_actual[1][idx]}\nPredictions:{Y_predic[idx]}')
```

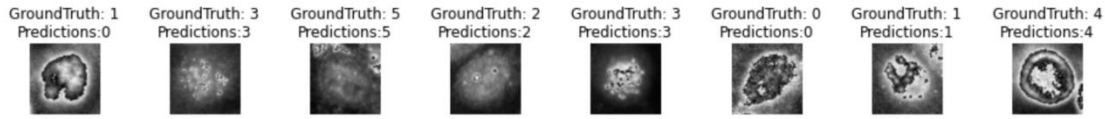

Figure 13 Visualization

With this method, we could make our result easy-to-understand and easy for checking.

3.5.2 Performance matrix

There is a confusion matrix which is a way to measure the performance of classification problem. Two dimensions in this matrix are Actual and Predicted. We could separate it to four parts:

- **True Positives (TP)** − It is the case when both actual class & predicted class of data point is 1.
- **True Negatives (TN)** − It is the case when both actual class & predicted class of data point is 0.
- **False Positives (FP)** − It is the case when actual class of data point is 0 & predicted class of data point is 1.
- **False Negatives (FN)** − It is the case when actual class of data point is 1 & predicted class of data point is 0.

In other words, we want larger TP and TN values. And we could apply these four parts to:

- $Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$
- $Precision = \frac{TP}{TP+FP}$, which is the number of correct documents returned.
- $Recall = \frac{TP}{TP+FN}$, which is the number of positives returned.
- $Specificity = \frac{TN}{TN+FP}$, which is defined as the number of negatives returned.

And with these values, we could get F1 score which is calculated by:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

The best value of F1 would be 1 and worst would be 0.

4 Section4 Results

In this part, some of the results from each part will be shown below in order.

4.1 Data preparation

The lengths of the DataSets are 8701 for training, 2720 for testing and 2175 for testing.

```
train_dataset = OurDataset(mylib)
test_dataset = OurDataset(testlib)
val_dataset = OurDataset(vallib)
```

```
[ ] # print the lenth of each data set
print(len(train_dataset))
print(len(test_dataset))
print(len(val_dataset))
```

```
8701
2720
2175
```

Figure 14 length of each dataset

After setting up three Data loaders (training, testing and validation) with batch size = 6 as our baseline, we could check the items in it.

```
train_dataloader = DataLoader(train_dataset, batch_size=6, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=6, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=6, shuffle=True)


# check some properties in the first batch
first_batch = next(iter(train_dataloader))

print(f'the type of this object is {type(first_batch)}')
print(f'the length of this tuple is {len(first_batch)}')
print(f'the type of the first item in the tuple is {type(first_batch[0])}')    # Tensor!
print(f'the shape of the first item (image tensors) is {first_batch[0].shape}')    # The size of image tensors in this batch!
print(f'the shape of the first item (image tensors) is {first_batch[0][0].shape}')

print(f'the shape of the second item is {first_batch[1].shape}')
print(f'the second item in the tuple is {first_batch[1]}')    # The labels (batch size=6 means there are 6 labels)
```

```
the type of this object is <class 'list'>
the length of this tuple is 2
the type of the first item in the tuple is <class 'torch.Tensor'>
the shape of the first item (image tensors) is torch.Size([6, 3, 224, 224])
the shape of the first item (image tensors) is torch.Size([3, 224, 224])
the shape of the second item is torch.Size([6])
the second item in the tuple is tensor([0, 3, 1, 0, 3, 4])
```

Figure 15 Items in the first batch

We could find that there are two items in one batch, the first is the images, and the second one is the classes of those images.

We can see the 4th output is: [6, 3, 224, 224]. It means there are 6 images in this batch, each image has size [3, 224, 224] as their **Channel number, Height, and Weight**. And the reason we set the height and weight as 224*224 is that it is the normal input size of AlexNet which is the CNN we used in this project.

4.2 Network setting up

The edited AlexNet structure looks like:

```
print(net)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 48, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(48, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(192, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4608, out_features=2048, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=2048, out_features=2048, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=2048, out_features=6, bias=True)
  )
)
```

Figure 16 Network Structure

4.3 Hyper-parameters selection

The results (loss and accuracy) are shown in one diagram with two y-axes (left is Loss, right is Accuracy) at the same time for each learning rate and the x-axis is the epoch. We have three batch sizes for choosing.

1. Batch size = 4

```
The length of the train_loss is 5
The length of the first item in train_loss is 15
<matplotlib.legend.Legend at 0x7fedd035dc90>
```

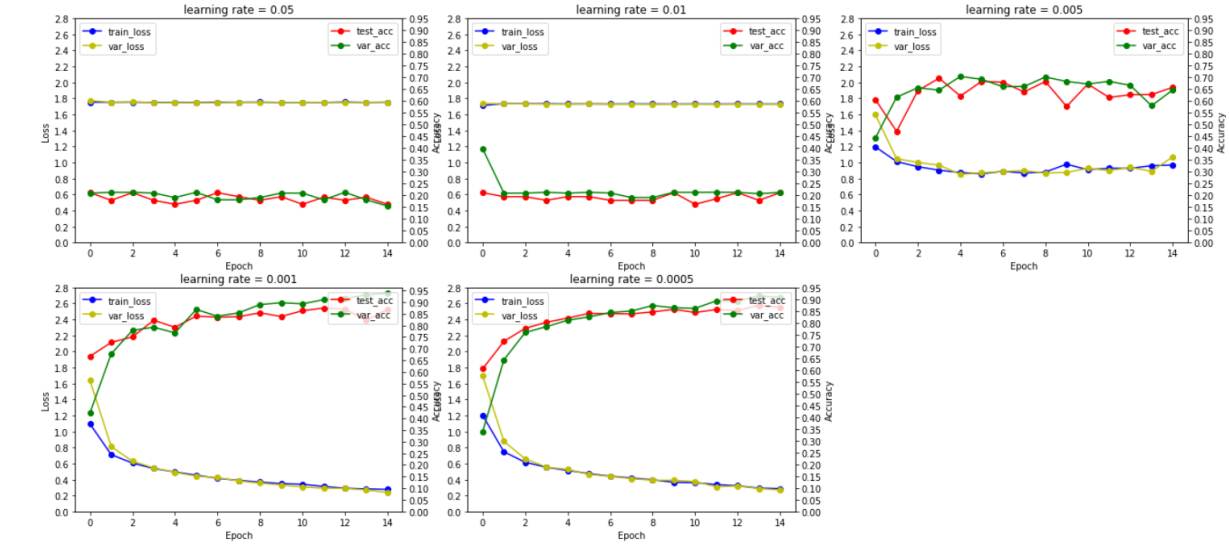

Figure 17 Batch size = 4

We could find the best accuracy occurs when learning rate = 0.001 and epoch = 15.

The highest accuracy for testing and validation set are 0.869 and 0.937 respectively:

```
The val_loss of the 15 epoch is: 0.236
validation set accuracy is: 0.9374712643678161
The train_loss of the 15 epoch is: 0.276
test set accuracy is: 0.8683823529411765
Finished Training for learning rate = 0.00100
```

Figure 18 batch size = 4 result

2. Batch size = 6

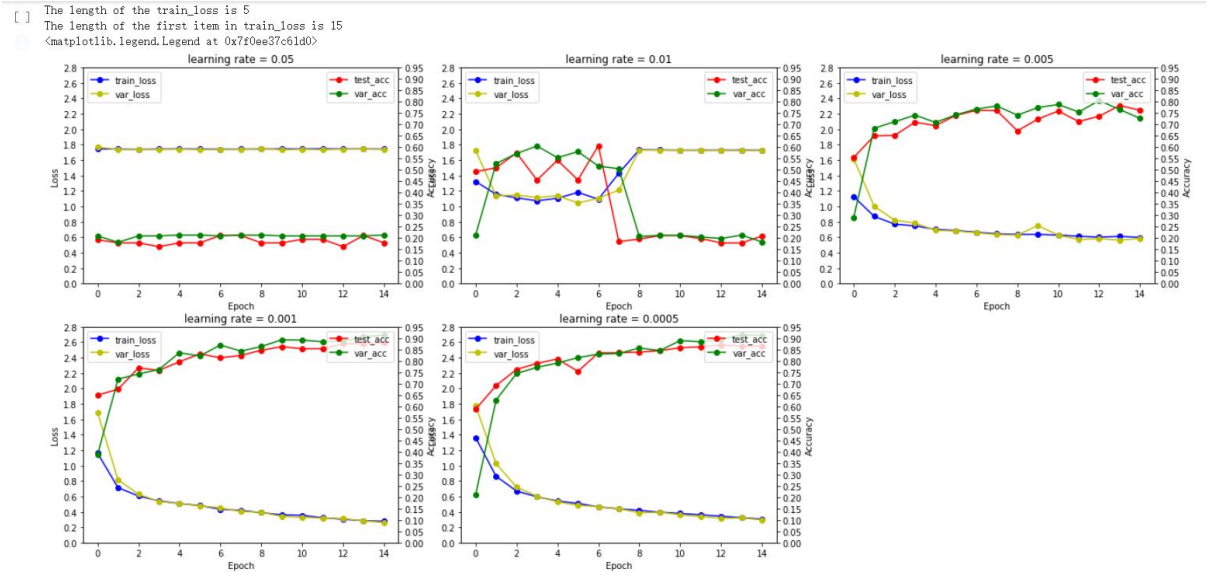

Figure 19 Batch size = 6

We could find the best accuracy occurs when learning rate = 0.001 and epoch = 15.

The highest accuracy for testing and validation set are 0.880 and 0.914 respectively:

```
The val_loss of the 15 epoch is: 0.261
validation set accuracy is: 0.9135632183908046
The train_loss of the 15 epoch is: 0.277
test set accuracy is: 0.8801470588235294
Finished Training for learning rate = 0.00100
```

Figure 20 batch size = 6 result

3. Batch size = 8

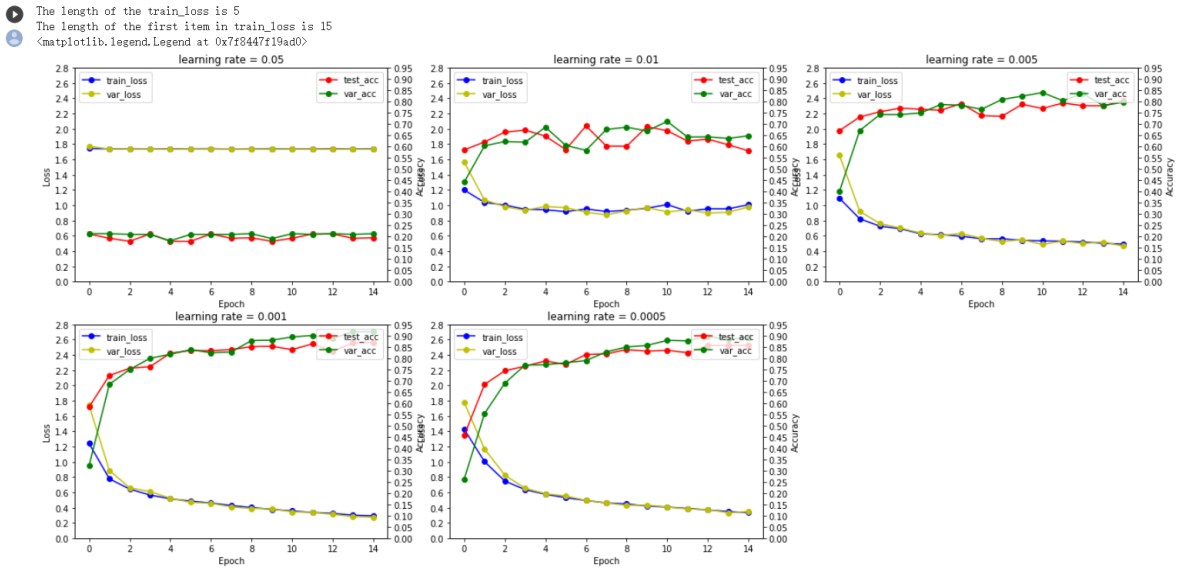

Figure 21 Batch size = 8

We could find the best accuracy occurs when learning rate = 0.001 and epoch = 15.

The highest accuracy for testing and validation set are 0.869 and 0.916 respectively:

```
The val_loss of the 15 epoch is: 0.274
validation set accuracy is: 0.9163218390804597
The train_loss of the 15 epoch is: 0.291
test set accuracy is: 0.8698529411764706
Finished Training for learning rate = 0.00100
```

Figure 22 batch size = 8 result

Overall, the best result (0.880) occurs when batch size = 6, learning rate = 0.001 and epoch = 15. The reasons we did not add more epochs are:

1) Concerning the time needed 2) The accuracy and loss curves are becoming flatten which means it could be considered as already converged.

4.4 Visualization and performance matrix

Notice that in this part, in order to make the performance matrix more persuasive, which means there are more images in different classes are considered at the same time for calculating the accuracy, we finally change the batch size = 32. But the leaning rate and the epoch are still 0.001 and 15 respectively.

4.4.1 Visualization

In this part, we visualized the first 8 images in a batch with their Ground-Truth and Prediction labels. This could help us to identify the result for our network.

```
fig, ax = plt.subplots(nrows=1, ncols=8)      # Visualization
plt.subplots_adjust(left=8, right=10, wspace=1)
for idx in range(8):

        ax[idx].imshow(X_actual[0][idx].permute(1,2,0))
        ax[idx].axis('off')
        ax[idx].set_title(f'GroundTruth: {X_actual[1][idx]}\nPredictions:{Y_predic[idx]}')
```

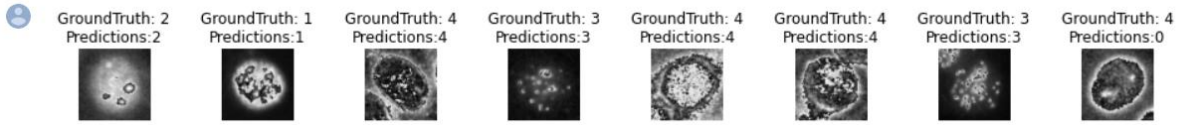

Figure 23 Result visualization

We could find there is only one wrong prediction in the first 8 images in the first batch.

4.4.2 Performance matrix

We firstly print out the GroundTruth and corresponding Prediction labels in the first batch:

```
# Check the prediction of the first batch

X_actual = next(iter(train_dataloader))
Y_predic = torch.argmax(net(X_actual[0].cuda()), dim=1)
print(X_actual[1])
print(Y_predic)
```

```
tensor([1, 3, 5, 2, 3, 0, 1, 4, 0, 1, 1, 3, 2, 0, 2, 0, 0, 3, 5, 1, 3, 3, 4, 1,
        3, 1, 2, 2, 0, 3, 5, 5])
tensor([0, 3, 5, 2, 3, 0, 1, 4, 1, 1, 1, 3, 2, 0, 2, 0, 0, 3, 5, 0, 3, 3, 4, 1,
        3, 1, 2, 2, 4, 3, 5, 5], device='cuda:0')
```

Figure 24 labels printed

In order to use performance matrix, we need change the tensor type to lists:

```
X_actual = X_actual[1].cpu().numpy()
print(X_actual)
Y_predic = Y_predic.data.cpu().numpy()
print(Y_predic)
```

```
[1 3 5 2 3 0 1 4 0 1 1 3 2 0 2 0 0 3 5 1 3 3 4 1 3 1 2 2 0 3 5 5]
[0 3 5 2 3 0 1 4 1 1 1 3 2 0 2 0 0 3 5 0 3 3 4 1 3 1 2 2 4 3 5 5]
```

Figure 25 labels in list

Then, using sklearn.metrics package, we could calculate those results easier:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss

results = confusion_matrix(X_actual, Y_predic)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))
print ('Classification Report : ')
print (classification_report(X_actual, Y_predic))
```

```
Confusion Matrix :
[[4 1 0 0 1 0]
 [2 5 0 0 0 0]
 [0 0 5 0 0 0]
 [0 0 0 8 0 0]
 [0 0 0 0 2 0]
 [0 0 0 0 0 4]]
Accuracy Score is 0.875
Classification Report :
              precision    recall  f1-score   support

           0       0.67      0.67      0.67         6
           1       0.83      0.71      0.77         7
           2       1.00      1.00      1.00         5
           3       1.00      1.00      1.00         8
           4       0.67      1.00      0.80         2
           5       1.00      1.00      1.00         4

    accuracy                           0.88        32
   macro avg       0.86      0.90      0.87        32
weighted avg       0.88      0.88      0.87        32
```

Figure 26 Performance matrix result

From the result shown above, we could find:

1) The confusion matrix could be re-written as:

		Actual class					
		0	1	2	3	4	5
Prediction class	0	4	1	0	0	1	0
	1	2	5	0	0	0	0
	2	0	0	5	0	0	0
	3	0	0	0	8	0	0
	4	0	0	0	0	2	0
	5	0	0	0	0	0	4

2) The final accuracy is 0.88.

3) From the classification report, we could find 6 images in the Homogeneous class (class 0) are chosen, 7 images in Speckled class (class 1) are chosen, 5 images in Nucleolar class (class 2) are chosen, 8 images in Centromere class (class 3) are chosen, 2 images in NuMem class (class 4) are chosen and 4 images in Golgi class (class 5) are chosen.

4) Based on above matrix,

The precision ($Precision = \frac{TP}{TP+FP}$) of:

Class 0 is 4/(4+2) = 0.67, Class 1 is 5/(5+1) = 0.83, Class 2 is 5/(5+0) = 1, Class 3 is 8/(8+0) = 1, Class 4 is 2/(2+1) = 0.67, Class 5 is 4/(4+0) = 1

The recall ($Recall = \frac{TP}{TP+FN}$) of:

Class 0 is 4/(4+1+1) = 0.67, Class 1 is 5/(5+2) = 0.71, Class 2 is 5/(5+0) = 1, Class 3 is 8/(8+0) = 1, Class 4 is 2/(2+0) = 1, Class 5 is 4/(4+0) = 1

The f1-score ($F1 = 2 * \frac{precision*recall}{precision+recall}$) of:

Class 0 is 2*0.67*0.67/(0.67+0.67) = 0.67, Class 1 is 2*0.83*0.71/(0.83+0.71) = 0.77

Class 2 is 2*1*1/(1+1) = 1, Class 3 is 2*1*1/(1+1) = 1, Class 4 is 2*0.67*1/(0.67+1) = 0.8

Class 5 is 2*1*1/(1+1) = 1

Which means our network currently has a good classification for class 2, 3, and 5.

5 Section5 Discussion and Conclusion

The detail analysis and discussion of the results are mentioned after each section in the Result part (Part4).

Overall, in this project, we completed using CNN to do the HEp-2 images classification task. When using AlexNet as our network, choosing the proper hyperparameters (batch size = 6, epoch = 15 and learning rate = 0.001), we could finally got an accuracy for 88%. Also, another performance evaluation method called “confusion & performance matrix” is used for analyzing the result for larger batch size DataLoaders in order to make the result more persuasive and higher credibility and we also got a result for 87.5% as accuracy. Morever, I learned how HEp-2 images play an important role in today’s medical area and the widely usage and benefits of CNNs.

References

[1] Connor Shorten, Taghi M. Khoshgoftaar (2019). *A survey on Image Data Augmentation for Deep Learning. Mathematics and Computers in Simulation*. springer. **6**: 60.

[2] Avinash (Nov,2018), *What is Transform and Transform Normalize? (Lesson 4 — Neural Networks in PyTorch)*, Retired from: https://medium.com/@ml_kid/what-is-transform-and-transform-normalize-lesson-4-neural-networks-in-pytorch-ca97842336bd

[3] James McCaffrey (Oct, 2020), *How to Create and Use a PyTorch DataLoader*, Retired from: https://visualstudiomagazine.com/articles/2020/09/10/pytorch-dataloader.aspx

[4] Richmond Alake (Jul, 2020), *What AlexNet Brought To The World Of Deep Learning*, Retired from: https://towardsdatascience.com/what-alexnet-brought-to-the-world-of-deep-learning-46c7974b46fc

[5] Jason Brownlee (Oct, 2019), *A Gentle Introduction to Cross-Entropy for Machine Learning*, Retired from: https://machinelearningmastery.com/cross-entropy-for-machine-learning/

[6] Sanket Doshi (Jan 2019), *Various Optimization Algorithms for Training Neural Network*. Retired from: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[7] Jeremy Jordan (Mar, 2018), *Setting the learning rate of your neural network*, Retired from: https://www.jeremyjordan.me/nn-learning-rate/

[8] Daniel Kobran and David Banys (2019), *Convergence*, Retired from: https://docs.paperspace.com/machine-learning/wiki/convergence

[9] Zhimin Gao, Luping, Zhou .et.al (2017), *HEp-2 Cell Image Classification with Deep Convolutional Neural Networks*, Faculty of Engineering and Information Sciences - Papers: Part B. 153.

[10] P.Peter, H.Peter, and B.Muller (2002), *Mining knowledge for HEp-2 cell image classification*, Journal Artificial Intelligence in Medicine, 26, pp. 161-173

[11] Jason Brownlee (Jan, 2019), *How to Control the Stability of Training Neural Networks With the Batch Size*, Retired from: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/