Course: DATS 6203 – Machine Learning II

Malaria cell classification using CNN

Exam Number: 2

Instructor: Dr. Amir Jafari

Name: Rehapriadarsini Manikandasamy

RM	04.11.2022
Initials	Date

INTRODUCTION

The exam focused on developing a deep neural network using PyTorch for training to distinguish all types of malaria cells. The network primarily focused on the multi-label classification of seven different types of cell categories [’difficult’, ’gametocyte’, ’leukocyte’, ’red blood cell’, ’ring’, ’schizont’, ’trophozoite’]. The metrics: Accuracy and HLM were used to measure the performance of the models.

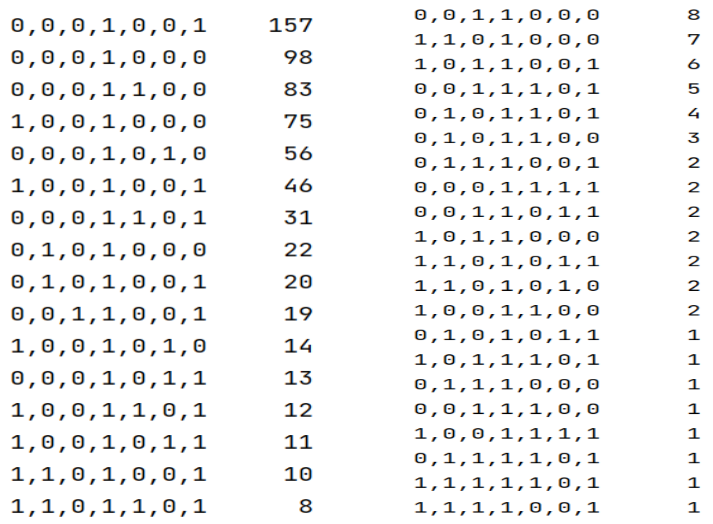
During the seven-day challenge, the best model developed had a sum score (accuracy and hlm) of 0.4495.

DATASET

The dataset provided had 887 cell images and the labels of these images were provided in an excel sheet. The provided dataset had been divided based on train and test labels. The ‘train’ split had 730 images files and ‘test’ split had 157 image files. The dataset was unbalanced, and the images were having different image dimensions (i.e.: 1600X1200, 1944X1833, etc.,)

Preprocessing:

1. **Image Resizing –** The images in the dataset didn’t have unified sizes but the neural networks needed an input vector of same size. Hence, the images were resized to shape of 400 X 400 X 3.
2. **Label encoding –** The targets provided were ‘strings. A label encoding step was implemented to generate a target class. The target classes were one-hot encoded based on the following order: [’difficult’, ’gametocyte’, ’leukocyte’, ’red blood cell’, ’ring’, ’schizont’, ’trophozoite’]. For example, the image containing difficult cells, red blood cells and trophozoite cells, the target will be [1, 0,0, 1, 0, 0, 1]
3. **Data Augmentation:** The dataset was highly unbalanced.



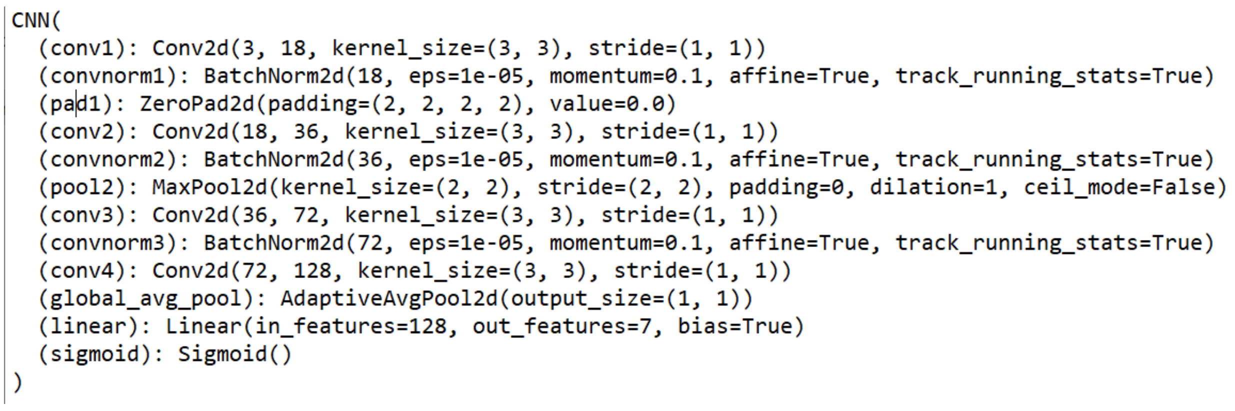
0,0,0,1,0,0,1	157	0,0,1,1,0,0,0	8
0,0,0,1,0,0,0	98	1,1,0,1,0,0,0	7
0,0,0,1,1,0,0	83	1,0,1,1,0,0,1	6
1,0,0,1,0,0,0	75	0,0,1,1,1,0,1	5
0,0,0,1,0,1,0	56	0,1,0,1,1,0,1	4
1,0,0,1,0,0,1	46	0,1,0,1,1,0,0	3
0,0,0,1,1,0,1	31	0,1,1,1,0,0,1	2
0,1,0,1,0,0,0	22	0,0,0,1,1,1,1	2
0,1,0,1,0,0,1	20	0,0,1,1,0,1,1	2
0,0,1,1,0,0,1	19	1,0,1,1,0,0,0	2
1,0,0,1,0,1,0	14	1,1,0,1,0,1,1	2
0,0,0,1,0,1,1	13	1,1,0,1,0,1,0	2
1,0,0,1,1,0,1	12	1,0,0,1,1,0,0	2
1,0,0,1,0,1,1	11	0,1,0,1,0,1,1	1
1,1,0,1,0,0,1	10	1,0,1,1,1,0,1	1
1,1,0,1,1,0,1	8	0,1,1,1,0,0,0	1
		0,0,1,1,1,0,0	1
		1,0,0,1,1,1,1	1
		0,1,1,1,1,0,1	1
		1,1,1,1,1,0,1	1
		1,1,1,1,0,0,1	1

The above figure shows that most of the target classes are having samples less than 10. During competition, data augmentation was performed to increase all the samples to count of 100 each. Just to avoid the overfitting due to the repetition of same images random image rotations were performed.

MODELLING

Network Architecture:

The best model is a CNN model with an input dimension of 400X400X3. Every Conv2d layer is developed with layers for batch normalization and max pooling. Some of the major parameters optimized over the training period: Activation function – Relu, Batch Size = 3, Learning rate =0.01 and epoch- 20. The complete model has 4 learning blocks with sigmoid activation to identify the probabilities of each class. The model optimizer used was 'Adam' and the loss used was 'BCEWithLogitsLoss'.



```
CNN(
  (conv1): Conv2d(3, 18, kernel_size=(3, 3), stride=(1, 1))
  (convnorm1): BatchNorm2d(18, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pad1): ZeroPad2d(padding=(2, 2, 2, 2), value=0.0)
  (conv2): Conv2d(18, 36, kernel_size=(3, 3), stride=(1, 1))
  (convnorm2): BatchNorm2d(36, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (pool2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1, ceil_mode=False)
  (conv3): Conv2d(36, 72, kernel_size=(3, 3), stride=(1, 1))
  (convnorm3): BatchNorm2d(72, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): Conv2d(72, 128, kernel_size=(3, 3), stride=(1, 1))
  (global_avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
  (linear): Linear(in_features=128, out_features=7, bias=True)
  (sigmoid): Sigmoid()
)
```

Tuning of the network:

The following tuning operations were performed throughout the seven-day challenge. The performance metrics on test data and held out (validation) datasets were provided in tables below after the discussion

Day 1– The given base model with batch_size of 30 and learning rate of 0.001 was trained. The focus for the day was just to make sure the tarin and test scripts are running as expected. One of the major mistakes, I didn't notice was the image size (100) specified for resizing.

Day 2 – The images were resized to 400X400X3 dimension since the original images were having larger dimensions than the size defined on day1. This was done to make sure the model gets trained using the utmost information from the image. A dropout layer was also introduced in the last three layers. The dropout rate was adjusted from 0.05 to 0.5 to find the best value.

Day 3 – Adding dropout layers in the model reduced the accuracy on held-out set. Hence, they were removed. The batch size and learning rate was increased from 30 to 52, 0.001 to 0.1 respectively. Later a better model with batch size of 33 and learning rate of 0.01 was picked. The neurons in each layer were increased to find the best model.

Day 4 – This day was mainly focused on data augmentation. Each minority class on train split was increased to 100 at the stage data reading.

```
t_list=list(xdf_dset.target_class.value_counts().index[2:])
print(xdf_dset.target_class.value_counts())
print(len(xdf_dset))
x_train=xdf_dset.copy()
```

```
for j in range(len(t_list)):
    x_t=x_train[x_train['target_class']==t_list[j]]
    for k in range(100):
        xdf_dset.loc[len(xdf_dset.index)]=x_t.iloc[0]
```

This helped increasing the 730 training samples to 4000+ samples. Yet, the model wasn't performing well due to the duplicity in samples. Because most of the minority classes were having only 1 sample and increasing the same image to 100 samples clearly overfitted the model. While training the model using this populated data had the training accuracy to be increased to the maximum of 0.60 but the test was dropping close to 0.06. No submission was made for the day as the model was clearly overfitting and the performance wasn't better than the previous model scores.

Day 5 – Pretrained models like resnet18, resnet34, resnet50, desnsenet121 and vgg16 were tried. Out of all the pretrained models resnet34 performed well on the dataset, its performance was comparatively equal to the CNN model from day3. While training the densenet121, an error of 'Cuda memory error' occurred. Reducing the batch size for training didn't help in resolving the error.

Day 6 – The data augmentation implemented on the day4 was again implemented with resnet34 and CNN (day3) model with reduced batch size and learning rate was trained. Both the models were clearly overfitting. Adjusting the parameters on CNN the model performed better than resnet34 on test data.

Day 7 – In order to reduce the overfitting of the model, tried implementation of image rotation on train data using CV2.

```
if  (index%5)==0:
    X = torch.FloatTensor(img)
    X = torch.reshape(X, (3, IMAGE_SIZE, IMAGE_SIZE))
elif (index%2)==0:
    img= cv2.rotate(img, cv2.cv2.ROTATE_90_CLOCKWISE)
    X = torch.FloatTensor(img)
    X = torch.reshape(X, (3, IMAGE_SIZE, IMAGE_SIZE))
else:
    img= cv2.rotate(img, cv2.ROTATE_180)
    X = torch.FloatTensor(img)
    X = torch.reshape(X, (3, IMAGE_SIZE, IMAGE_SIZE))
```

The model performed well but the performance wasn't better than the previous model, yet the model was saved and tested with test values. Later, a transformation using transforms from the torchvision was implemented.

```
transform = transforms.Compose(
    [transforms.ToPILImage(),
transforms.RandomPerspective(distortion_scale=0.5, p=0.5, fill=0),
transforms.ToTensor()])

img = transform(img)
X = torch.FloatTensor(img)
X = torch.reshape(X, (3, IMAGE_SIZE, IMAGE_SIZE))
```

The model was performing well, and the overfitting problem was little bit reduced compared to the previous models. While training with lower batch size of 15 cuda error again occurred. Due to the insufficient amount of time left for retraining the same model the saved results from the previous model of the day was submitted.

Final model:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.conv1 = nn.Conv2d(3, 18, (3, 3))
        self.convnorm1 = nn.BatchNorm2d(18)
        self.pad1 = nn.ZeroPad2d(2)

        self.conv2 = nn.Conv2d(18, 36, (3, 3))
        self.convnorm2 = nn.BatchNorm2d(36)
        self.pool2 = nn.MaxPool2d((2, 2))

        self.conv3 = nn.Conv2d(36, 72, (3, 3))
        self.convnorm3 = nn.BatchNorm2d(72)

        self.conv4 = nn.Conv2d(72, 128, (3, 3))
        self.global_avg_pool = nn.AdaptiveAvgPool2d((1, 1))

        self.linear = nn.Linear(128, OUTPUTS_a)
        self.sigmoid = nn.Sigmoid()
        self.act = torch.relu

    def forward(self, x):
        x = self.pad1(self.convnorm1(self.act(self.conv1(x))))
        x = self.pool2(self.convnorm2(self.act(self.conv2(x))))
        x = self.act(self.conv4(self.convnorm3(self.act(self.conv3(x)))))
        return self.linear(self.global_avg_pool(x).view(-1, 128))
```

Performance measures based on test set

Metric/Day	**Day1**	**Day2**	**Day3**	**Day4**	**Day5**	**Day6**	**Day7**
Accuracy	~0.20	~0.28	0.33	0.06	-	0.19	0.17
HLM	~0.159	~0.15	0.145	0.23	-	0.17	0.19
Test loss	0.36	0.35	0.32	10.95	-	0.37	0.36

Performance measures based on validation set

Metric/Day	**Day1**	**Day2**	**Day3**	**Day4**	**Day5**	**Day6**	**Day7**
Score	0.404	0.38	0.44	-	-	0.34	0.37
Accuracy	0.26	0.22	0.31	-	-	0.17	0.15
HLM	0.14	0.15	0.13	-	-	0.16	0.21
Average	0.20	0.19	0.22	-	-	0.17	0.18
Sum	0.404	0.38	0.44	-	-	0.34	0.37

The above table shows how the model has been gradually improved from the basic to a well-developed classification neural network.

Failed techniques:

The following preprocessing techniques were attempted in this process, but they were not implemented due to the bugs and conceptual errors faced during the development process.

1. The data augmentation performed was overfitting the models during training. A better augmentation technique by creating much difference in the augmented images could be a better implementation
2. Using pretrained models with lesser batch size and better hyperparameters without experiencing memory error would possibly end up with a better classifier

CONCLUSION/ FUTURE WORK

This work helped in understanding the idea behind the development of a deep neural network and optimizing its parameters for a better trained model. The best model (model from day 3) was picked based on the sum and average of all the provided metrics. The best model has a better accuracy score of 0.33 on the test set and sum score of 0.44 on the held-out set. The model in the future can be enhanced using the discussed preprocessing technique like better augmentation practices and usage of pretrained models efficiently would lead to a better model.

REFERENCES:

1. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
2. https://pytorch.org/vision/stable/models.html
3. https://pytorch.org/vision/0.9/transforms.html