DATS – 6203 (Spring 2020)
Final Project – Individual Report
Kartik Kumar Mathur

Group 4 Team Members: Kartik Mathur, Jason Witry, Jane Zeng

1. Introduction

This is an individual report as a part of a group project for the Machine Learning -II class. The project was a multi-class classfication problem on a dataset which contained chest X-Rays of pediatric patients looking for possible Pneumonia treatment options. This dataset was obtained from Kaggle and was not originally meant for a multi-class classification problem. The decision of pursuing a different and a more complex problem statement was unanimous in the team.

2. Dataset

To address the problem, a dataset was obtained from Kaggle (https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia) that contained 5,863 chest X-Rays of pediatric patients aged 1-5 years old from Guangzhou Women and Children's Medical Center in Guangzhou. These images are split into a train set (5,216 images), a validation set (16 images) and a test set (624 images). The images consist of 3 channels of the same pixels, approximately (1200 by 1600) in size.

In each, we see the lungs on the right side of the spine in the image, with the ribcage surrounding them. The images also have a letter in the corner; "R", which indicates that the indicated side is the right side.

3. Problem Statement

The original dataset was intended to perform binary classification on the dataset that classifies the images into two categories: Normal & Pneumonia. Majority of the kernels that we went through on Kaggle have chosen to tackle this as a binary classification problem. This project deals with classifying the data into three categories: Normal, Viral Pneumonia, Bacterial Pneumonia.

4. Experimental Setup

We chose to use PyTorch as our framework of choice as this problem required more control over the parameters; considering it is a complex problem.

We also reorganised the dataset to balance the original validation dataset(since it only contained 16 images – considered very low for our problem statement). I experimented with different number of images for each class in the validation set and tried with various models. An equal amount of images gives a more balanced result during validation as it represents each class equally and makes it better for training.We also resized the images to 256 x 256 pixels for training to be easier.

DATS – 6203 (Spring 2020)
Final Project – Individual Report
Kartik Kumar Mathur

---

5. Individual Approach

My approach towards the problem was to decide a performance index for the problem. Since the most often appropriate performance index is CrossEntropyLoss() for classification problems. first achieve a working model. Which is why I ran the first model on the reorganised dataset and uploaded it. My first approach towards this was to resize the images to a smaller size and then add a basic CNN on the network. I used a CNN with 2 layers, Batch normalization, Relu and max pooling layers. Without a balance of the dataset and the simple architecture of this model, The results were way low and gave a validaiton loss of around ~2.3.

6. Shared Work & Results

The team worked in three tangents before arriving at the best model for our problem statement. The initial approach was to use basic CNN architectures that I built from the scratch. Following poor results in this approach, I started to explore transfer learning approaches. The pre-trained networks were our option and we chose to work with ResNet & DenseNet. These are highly complex networks and would work beter for our solution. The basic Resnet18 & Resnet34 models with 2-4 layers. This gave improved results but not to a great extent.

In both our experiments, these residual networks were shown to be faster than plain networks with no skip connections and achieve similar, and sometimes better, results than plain networks. Therefore, we built several fully connected classifier architectures with ResNet as the backbone, including 1-layer, 2-layer, 4-layer and 6-layer networks named ResNet_fcN, where N is the number of layers in the fully connected classifier. For any network with 2-layer and above classifiers, training the entire network overfit very quickly. Therefore, we froze the ResNet neurons after seeing signs of overfitting, and continued training the fully connected classifiers. However, the network performance stagnated in each of these models on both the training and validation sets, and none outperformed ResNet with one fully connected classifier (ResNet_fc1).

ResNet_fc1

ResNet_fc1 was trained for 125 epochs total and overfitting began at epoch 25. We trained the network for another 100 epochs to be sure the network would not converge to a smaller minimum. It was trained with the following hyperparameters:

Hyperparameter	Value
Learning Rate	1e-3
Batch Size	64
Sampler Weights	Chosen at runtime to balance classes

We used SGD as the optimizer, with a momentum of 0.9 and no weight decay. The validation (labelled test here) and training loss are shown below:

The best model still turned out to be basic Resnet18 model with one classifier in the end. It gave out a validation accuracy of 85% and a test accuracy of 83%.

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 3)
drop = nn.Dropout(0.2)
act = nn.LeakyReLU()
```

7. Individual Contribution & Results

My first contribution was to figure out a basic model with CNN. I started with creating a function to load the data first using the **torchvision.datasets.ImageFolder()** function.

```
def load_dataset():
    train_path = os.path.join(filepath,'chest_xray/train')
    train_dataset = torchvision.datasets.ImageFolder(
        root=train_path,
        transform=torchvision.transforms.ToTensor()
    )
```

Once this dataset was read and created, I created a data loader using t**orch.utils.data.DataLoader() .**

```
def get_loaders(train_dataset,valid_dataset,test_dataset):
    train_loader = torch.utils.data.DataLoader(
        train_dataset,
        batch_size=64,
        num_workers=0,
        shuffle=True
        )
```

My basic CNN model contained two layers

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


        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32,kernel_size=3,stride=1,padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2

        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=32,kernel_size=3,stride=1,padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2))


        self.fc = nn.Linear(in_features=32*64*64, out_features = 3)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        return out

cnn = CNN().to(device)
```

I experimented with multiple layers ranging from 2 – 16 such layers and changed the hyper parameters to match a more complex network and try to get the results. The results for these experiments are in the summary section below.

Since the output were 3 classes, we chose the output layer to be a Linear layer with 3 outputs.

After trying this, Since the results were not improving further than 69% accuracy and a minimum loss of ~1.2, I tried to experiment with pre-trained networks.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	7×7, 64, stride 2
conv2_x	$56 \times 56 \times 64$	3×3 max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	7×7 average pool
fully connected	1000	512×1000 fully connections
softmax	1000	

Fig. ResNet18 structure

I started with applying basic Resnet18 with a classifier as a sequential model.

```
model = torchvision.models.resnet18(pretrained=True)
model.fc = nn.Sequential(
   nn.Linear(model.fc.in_features, 16),
   nn.LeakyReLU(),
   nn.Dropout2d(dropout),
   nn.Linear(16,3)
   )
```

This gave a better result with loss of **~0.9** and a highest validation accuracy of **78%**. This model also gave a test accuracy of **81%** in the end.

I used the parameters as following:

Image Size	**256**

Epochs	**30**
Learning Rate	**0.005**
Batch Size	**128**
Dropout	**0.3**
Optimizer	**SGD**

The ResNet18 seemed like the learning was saturating closer to 15 epochs with the above parameters and started to overfit the training data from ~17th epoch. The training loss fell rapidly after this and provided a training accuracy of 90% around the 20th epoch.

Once ResNet18 gave a better result, I continued pursuing more complex versions of these pre trained models which have more number of trainable parameters. I tried Resnet34 which gave better results but not better than ResNet18. I also noticed that the Resnet34 model seemed to overfit the data much faster than the previous model. The training stopped and saturated around the 9th epoch at a validation accuracy of 72% and continued to stagnate around the same value for the rest of the training cycle while the training loss rapidly grew until the 20th epoch. Finally, The loss minimized at ~0.98 and gave a validation accuracy of 74%.

Meanwhile, I also played around with the batch sizes and the optimizer. The batch sizes were optimized at 128, Lower the batch size, worse were the results. But a batch size higher than 200 was definitely leading to easier overfitting of the data probably because of the ratio of training normal images to the rest of the two classes.

Number of Layers	Number of Parameters
ResNet 18	11.174M
ResNet 34	21.282M
ResNet 50	23.521M
ResNet 101	42.513M
ResNet 152	58.157M

Fig. ResNet models and number of trainable parameters

I also tried DenseNet121 (Simplest of the DesneNet models) and got less valuable results than Resnet. The DenseNet brough the validation accuracy back down to 71% and the minimum loss to ~1.1. I switched back to Resnet and tried to add some transformations to the pre processing.

```
transforms.Compose([
        transforms.Resize(resize),
        transforms.CenterCrop(224) ,
        transforms.RandomHorizontalFlip(),
       transforms.ToTensor(),
       transforms.Normalize([0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Among other transformations was to sheer, FiveCrop, Pad, RandomAffine, RandomCrop, RandomResizedCrop, Scale. Of the options, Sheer and randomAffine seemed to help out with the results.

From there on, Jason and I pursued the Resnet_Fc model to get the final and best result.

8. Summary & Conclusions

CNN Layers	Batch Size	LR	Epoch	Changes	~Validation Loss	~Validation accuracy
Ch = 16,16 (2 Layers, k = 3, MaxPool)	128	0.001	30	SGD	2.23	39%
16,16 (4 Layers, k = 3, MaxPool)	128	0.001	30	#layers	1.94	48%
16,16 (7 Layers, k = 2, MaxPool)	128	0.005	30	#layers #kernels	1.21	69%
16,16 (12 Layers, k = 2, MaxPool)	128	0.001	30	#layers	1.72	49%
16,16 (16 Layers, k = 2, MaxPool)	128	0.005	30	#layers Adams	1.42	52%
16,16 (7 Layers, k = 2, MaxPool)	64	0.001	30	Batch_size	1.35	57%
SWITCH TO PRE-TRAINED NETWORKS						
Resnet18 + 3 Linear Layers	128	0.005	30	SGD	0.91	76%
Resnet18 + 8 Linear Layers	128	0.005	30		1.34	70.5%
Resnet34 + 2 Linear Layers	128	0.005	30	SGD	1.04	72%
DenseNet121	128	0.005	30	SGD	1.10	68.7%

The best accuracy I could achieve was around 76% for vaildation and a test accuracy of 79%. This was further improved by the collaboration between the team.

```
TRAIN
[[1007   25   40]
 [  32 1998  231]
 [  49  480  539]]
Average Accuracy: 80.53 +/- (0.0) %
Average Precision: 79.64 +/- (0.0) %
Average Recall: 77.59 +/- (0.0) %
Val
[[244  20  13]
 [  1 261  15]
 [  5 159 113]]
Average Accuracy: 74.37 +/- (0.0) %
Average Precision: 79.02 +/- (0.0) %
Average Recall: 74.37 +/- (0.0) %
```

Fig : Confusion Matrix in between training loop at 10 epochs with Resnet18 and 4 layers

9. Possible options in approaches

Trying to use more transformation and image augmentations would certainly help in getting better results. Since we were on a time crunch and were developing models for our problem without any prior context, this time frame a shorter than expected. Given a chance, I would explore more of the Pre-trained networks and ways to combine our own codes and models , perhaps append it to them and then train our dataset. It would definitely help make the results better. What was also important was that we learned to experiment with a new concept of transfer learning and experimenting with pre-trained networks in PyTorch.

10. Percentage of Code from Internet

For the basic script that I wrote in my own sense and as a part of the final script, only the model was chosen as the script written as a final script was more efficient than mine.

Excluding the import statements,

% from internet = #copied - #modified / #copied + #added

% = (83 - 43) / (83 + 65)

= 40/148

Code % from internet = 27.02 %

---

11. References

- https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8
- https://www.researchgate.net/figure/ResNet-18-Architecture_tbl1_322476121
- https://pytorch.org/docs/stable/data.html
- https://pytorch.org/docs/stable/torchvision/transforms.html
- https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751
- https://stackoverflow.com/questions/50136032/modify-layers-in-resnet-model
- https://www.kaggle.com/c/cdiscount-image-classification-challenge/discussion/40249
- https://pytorch.org/docs/stable/torchvision/models.html
- https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803