Machine Learning - Assignment 2

GitHub: https://github.com/dank100/ML-A2/blob/master/ML2.ipynb

Hans Christian Milman 13690223
Dan Kristiansen 13514346

September 2019

1 Introduction

In recent years, we have become more and more dependent on the practice of image-recognition. It's usecases range from self-driving cars to facial recognition. Optical Image Recognition (OCR) has also been a usecase for a long time. One of the more recent trends within image recognition is the use of convolutional networks. This method allows for easy and fast classification of images without the need for complex algorithms. On the flipside, this method requires a lot of data in order to be precise.

Since other OCR methods that extract text in images can be very resource demanding , it can benificial to quickly determine if there event exists text in a given image before applying them. This is the core of our business case, to **determine if there is text in a given image**.

2 Exploration

2.1 Idea

We want to implement a Convolutional Neural Network(CNN) and train it on a dataset consisting of images with and without text. This translates to a binary classification problem with the two classes "text" and "no text".

2.2 Challenges

The main challenge of any deep learning project is aquiring relevant and useful data in large enough quantities to train an accurate model. For all practical purposes, we would need at least several thousand datapoints and more computational power than we have access to. Because we lack both of these, we make a point out of the theoretical usecases and accuracy of such a model as the focus of our report.

2.3 Tests

In order to evaluate the accuracy of our model, we seperate our dataset into train- and test-sets. We're doing this because testing on data you've trained on will yield a misrepresentative accuracy compared to real-world instances. Ideally we want to test our model on new data that does not even come from the same dataset. A common way of splitting the data is 80/20, since it allows to preserve a lot of training data for training a decent model, and leaves us with enough test-data to get a representative accuracy score.

Testing would be straight foward: apply model to a set of data and compare predicted classes with actual classes. This yields an accuracy in the form of percentage.

3 Implementation

Firstly we introduce the architecture of the final CNN model in, the model consist of two convolution layers, a maxpool and three linear reduction layers.

Listing 1: CNN implementation and architecture

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, kernel_size = 3)
        self.pool = nn.MaxPool2d(stride = 2, kernel_size = 2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size = 5)
        self.fc1 = nn.Linear((16 * 13 * 13), 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, CLASSES_NUM)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

3.1 Transform

The images are of different dimensions, which eventually are converted into tensors of various dimension through $ToTensor()$. A tensor is a number representation of the converted image, where the channels represent the brightness of its representing color. Based on the three channels; red, green, and blue, the tensor represent three pictures tinned towards the color on each channel. The normalization of the image ensures that all channels in the image is normalized into a range of $([0,1]-0.5)/0.5 = [-1,1]$ giving an equal range to all features in the image. The images are further alternated into a resized image of 32x32 pixels through the $Resize()$ function. The resizing is important as the layering of the model is specified to fixed dimensions.

```
TRANSFORMER = transforms.Compose(
                    [transforms.Resize((32,32)),
                     transforms.ToTensor(),
                     transforms.Normalize((0.5, 0.5, 0.5)
                     , (0.5, 0.5, 0.5))])
```

3.2 Convolutional Neural Network

The Convolutional Neural Network(CNN) model consist of multiple layers, where the layer structure is specified with a list of parameters. A convolution layer consist of a stack of layers determined by the input features in the image. The $kernel\_size$ refers to the $Height \cdot Width$ of the filter moving over the feature maps that outputs a new set of feature maps with the desired amount of features. The stride is implicit set to two, reducing the size to a $32 \cdot 32$ output. Thus, the stride determines the output size from the filter and represent the shifting of the filter reducing the input.

3.2.1 Pooling layer

The pooling layers reduce the dimensions of the feature maps, gained from convoluting the input image, with different filters. This emphasises the important features, compared to potential noise in the image. The reduction in image size also reduces the amount of parameters in the network, which decreases the amount of computations needed. The way pooling reduces the dimension is by, dividing an image into non-overlapping spaces of variable sizes. E.g. from the specified CNN when pooling an image of size 26x26, reducing it to a 13x13 image, the 26x26 image is split into four non-overlapping matrices of size 13x13. Now depending on the chosen pooling strategy, the entries in the 13x13 matrix is filled, with values from the image that is pooled. Max pooling strategt is used in the specified CNN, which takes the maximum number representation of the four 13x13 matrices.

3.2.2 Linear Layers 1-3

The dimensions of the tensor are reduced from $16 \cdot 13 \cdot 13 = 2704$ down to only two representing the two classes.

3.2.3 ReLU

The Rectified Linear Unit function is given by $f(x) = max(0, x)$. This function is applied as it is faster than other actication functions, because there are no expensive operations and it also does not suffer from the vanishing gradients problem.

3.3 Dataset

The dataset is a combined dataset from a natural image dataset and an image with text dataset. The first consist of 6,899 images where the classes are: Airplanes, Cars, Cats, Dogs, Flowers, Fruits, Motorcycles and Persons. Ideally there should be an even distribution of all classes, but good datasets are hard to come by.

We have 6205 images without text and 1525 with text. This is a relatively small datapool for training a classification model, but will have to do in our case. For theoretical purposes, this amount of data will be enough to show the general idea behind the model.

3.3.1 Splitting the data

In order to evaluate our model, we need a testing dataset. Ideally, the testset would come from a different source and be independent from the original dataset, but in our case that wasn't a possibility due to the scarsity of good datasources. Our dataset is not that big, so we chose to do a 80/20 split with 80% of the data being used for training and 20% being used for testing. The images are also shuffled within their respective class because of the smaller, and for this model irrelevant, subclasses that could introduce bias.

Since our solutions relies on the file-structure to locate test and training sets, this was done behind the scenes, and cannot be seen in the code.

3.3.2 Correcting for bias

Ideally we would want to have equal amounts of images with text and without text to avoid a bias towards one or the other. To counteract this bias, we can introduce class weights, which tell the network to weigh the smaller datapool higher, effectively eliminating bias. In order to not weigh any class too high and introduce even more bias, the class-weights are determined by a function of the ratio between the two datapools:

$$ClassWeight1 = \frac{LargerClass}{SmallerClass}$$

$$ClassWeight2 = \frac{SmallerClass}{LargerClass}$$

Alternative solution would be to duplicate data in the smaller dataset, although this might introduce a bias towards the contents of the images duplicated.

Since our data are of pretty similar distributions, correcting for bias might not be relevant for our model. This can be determined in the evaluation section.

3.4 Training

When dealing with supervised learning, such as neural networks, the network must beprovided with a training set that is used to teach the NN how the task should be solved.The training set consists of input and target values. The aim is then for the NN topredict the values of the target based on the input. Thus the NN can evaluate its ownperformance and correct itself based on the training set.

The learning part of the CNN is split into two steps. The first being theprediction step where inputs are passed through the CNN, followed by a backpropagation phase. This phase goes backwards through the layers and updates the weights of the edges according to an error-value, that is determined by a loss function.

When optimizing the model towards a gradient determined by the optimizer, the learning rate then determines the distance towards the gradient. When training a network it is preferable to have a high learning rate in the beginning, as to quickly find an error-minimum, and then have a lower learning rate for finding the global minimum towards the end of training.

The optimizer of this CNN is the implemented Adam optimizer of torch. The method is computationally effecient and has little memory requirement, making it well suited for training of large datasets with limited recourses.

```
EPOCHS = 5
BATCH_SIZE = 4
LEARNING_RATE = 0.001

model = CNN()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
loss_func = nn.CrossEntropyLoss()

for epoch in range(EPOCHS):
    for i, data in enumerate(train_data_loader, 0):

        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = loss_func(outputs, labels)
        loss.backward()
        optimizer.step()

print("Done")
```

3.4.1 Loss function

The Cross Entropy loss function describes the loss between two different probability distributions and is applied in order to avoid learning slowdown.

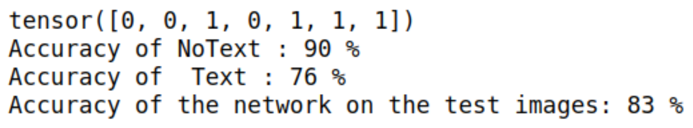
$$H(p,q) = -\sum_x p(x) \cdot log\ q(x) \tag{1}$$

The probability **p** and distribution **q**, respectively symbolise the true value and the predicted value. The loss for this model is determined by the measure of how close the probability distributions are to each other.
If an error is large, the weights learn their values faster, which means that the learning slowdown is circumvented. This is because, when a network starts its training, it usually start out with large errors, which is to be expected as the network has not learned anything yet. The network then learns from those errors faster than the basic loss

function, and therefore, avoid being stuck in a stalemate at the start of the training period.

4 Evaluation

As outlined in sections 2.3 and 3.3.1, we did a 80/20 split for training and testing. The best result we got, was with a 5 epochs, a batchsize of 4 and a learningrate of 0.001. The result can be seen below.



```
tensor([0, 0, 1, 0, 1, 1, 1])
Accuracy of NoText : 90 %
Accuracy of  Text : 76 %
Accuracy of the network on the test images: 83 %
```

Figure 1: Test-results

The results show a slightly better performance when predicting images without text in them. This bias might imply overfitting in favor of the "noText" class, but can also be a result of the quality of data we had available. We found that among the 5000 datapoints in our training data, there were few one-offs that might have included text when they weren't supposed to. Similarly, some of the images with text had a font small enough to not be identifiable when transformed.

We tried a few different combinations of batchsizes and epochs, in our conquest of achieving a better score. Results can be seen in the table below:

	Batchsize 2	Batchsize 4	Batchsize 8
3 epochs	70%	75%	72%
4 epochs	75%	77%	76%
5 epochs	72%	83%	78%

Figure 2: Accuracy with varying epochs and batchsizes

Overall the model shows promising results and would likely be able to perform very well in real-life cases had we had more data and resources available.

5 Conclusion

The Convolutional Neural Network(CNN) architecture have been tweaked throughout the training phase, to conform with the best model for text detection. We can conclude that the model has proven to be very accurate with a 90% accuracy detecting images with text. Moreover, has the model proven to be very time effecient when classifying images, as it only include two classes.

After the training phase and later evaluation, it has become clear for us exactly how important the dataset of a classification model is. As described in section 4 we identified some errors in the dataset, where images, labelled as *noText*, included some font in the image. To further improve the model, will we focus more on the dataset and try to retrieve even more images of different motives, with and without text.