Mobile number identification from forms

Siddharth Shrivastava
IIT Delhi
anz208849@iitd.ac.in

Ashutosh Agarwal
IIT Delhi
csy202452@iitd.ac.in

Abstract

In this assignment, we are supposed to identify a mobile no from a filled form. To do this, we first improve upon LeNet to build a single digit classifier and then use RNN on top of that to identify the full mobile number given the information that the mobile number is written by the same person.

1. Introduction

Extracting information via Machine Learning from filled forms not only saves time but it also makes the information more robust in comparison to a human being feeding the information manually.
In this assignment, we have a dataset that comprises of grids of mobile numbers of Indian population from a form. We have to identify the mobile no correctly from the given the image.The objective of this assignment is to use LeNet-5 [3] as a base model and make design innovations in its architecture to improve the performance on single digit classifier and subsequently identify the entire mobile no making use of the single digit classifier and the fact that one mobile number is written by the same person.

2. Related Work

For single-character recognition, despite forming a model from scratch, we used LeNet-5 [3] convolutional neural network that was pretrained on MNIST dataset as our baseline model, on which we perform our design innovations. LeNet is a five deep layer model with initial two layers extracting features via convolutional neural networks and the trailing layers through fully connected neural networks. Specifically, the convolutional layer uses a 5x5 filter and a tanh activation function. The output features of each convolutional filter are passed through the Max pooling layer to downsample the original input by a factor of 2, respectively. The first convolutional layer outputs six feature maps, and the second extracts 12 feature maps. In order to pass the output from convolutional block to fully connected block, extracted features from convolutional layers are flatten then subsequently these feature vectors are then passed through three fully connected layers to output the final logit of the network, which is normalised by softmax for the final prediction of single-digit recognition.

For recognizing multiple characters in a specific row of an image present in the given dataset, we have built the model inspired by CRNN [4]. In essence, the CRNN model combines the convolutional neural network (CNN) and recurrent neural network (RNN). Recurrent neural networks (RNN) models, another important branch of the deep neural networks family, were mainly designed for handling sequences. One of the advantages of RNN is that it does not need the position of each element in a sequence object image in both training and testing. However, a preprocessing step that converts an input object image into a sequence of image features is usually essential. Thus, for recognizing sequences in an image, CRNN based model has several advantages over CNN, which are as follows:
1. It can be directly learned from sequence labels (for instance, words), requiring no detailed annotations (for instance, characters);
2. It has the same property as CNN on learning informative representations directly from image data, requiring handcraft features nor preprocessing steps, including binarization/segmentation, component localization, etc. neither.;
3. It has the same property of RNN, being able to produce a sequence of labels;
4. It is unconstrained to the lengths of sequence-like objects, requiring only height normalization in both training and testing phases;
5. It achieves better or highly competitive performance on scene texts (word recognition) than the prior arts [2][1]

3. Method

3.1. Single Character Classification

3.1.1 Dataset Preprocessing

All the images were converted to grayscale and then standardized by subtracting the mean activity over the training

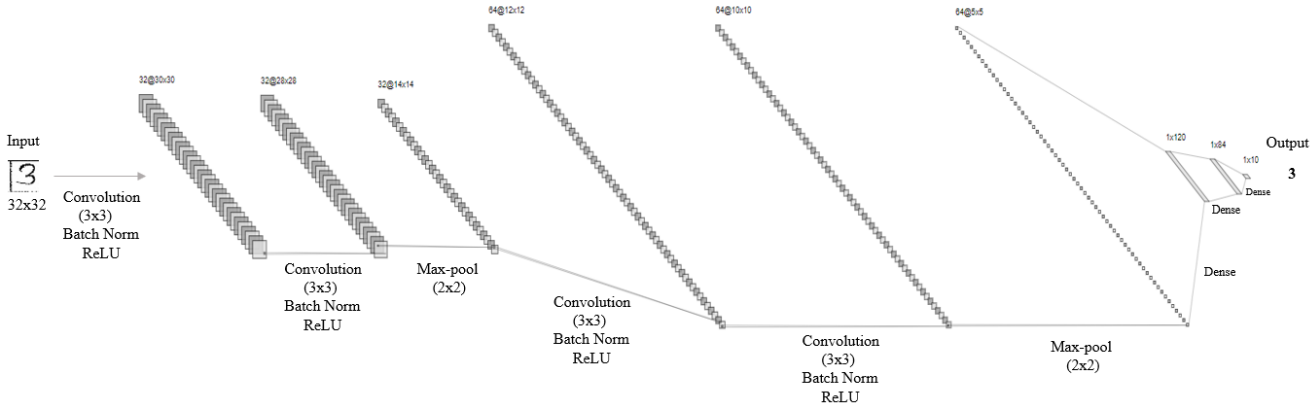

Figure 1. Single Classification Model Architecture

set from each pixel and dividing by their standard deviation. As we know, every image consists of pixel values between 0 and 255. Without standardization, the raw pixel values when fed into the network slows down the learning due to the erratic input nature.

3.1.2 Architecture

The design modifications of the Standard LeNet [3] model to build our network are:

- We have replaced each of the 5x5 kernels in a convolution layer with a stack of two convolution layer comprising a smaller 3x3 kernel. This change preserves the receptive field and increases the non-linearity in the network, due to which the resulting model can generalize better.
- We used ReLU as our activation function instead of Tanh in convolutional layers and inculcated it in Fully connected layers. ReLU overcomes the vanishing gradient problem and increases the model sparsity, which improves the model's faster convergence.
- We have introduced Drop out module in fully connected layers with a drop out rate of 0.1. It has significantly increased the model's generalisation capability.
- Batchnorm is used in Convolutional layers, the primary purpose of layers is to standardize our input features to each have a zero mean and unit variance. Outputs of this layer are normalized across the batch and within each separate image channel. Intuitively, normalization makes the loss function smoother, which helps it converge faster and more stable.
- Additional learning rate scheduler namely step learning rate was adopted for better optimisation of model's performance.

```
self.layer1 = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=(3, 3)),
    nn.BatchNorm2d(32),
    nn.ReLU(),
)

self.layer2 = nn.Sequential(
    nn.Conv2d(32, 32, kernel_size=(3, 3)),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2)
)
self.layer3 = nn.Sequential(
    nn.Conv2d(32, 64, kernel_size=(3, 3)),
    nn.BatchNorm2d(64),
    nn.ReLU(),
)

self.layer4 = nn.Sequential(
    nn.Conv2d(64, 64, kernel_size=(3, 3)),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2)
)
self.fc1 = nn.Linear(in_features=64*5*5, out_features=120)
self.fc2 = nn.Linear(in_features=120, out_features=84)
self.fc3 = nn.Linear(in_features=84, out_features=10)
```

Figure 2. Single Classification Model Architecture code

The final prediction was made for ten classes (0-9) using softmax activation in the final layer logits. The network is optimised using standard cross entropy loss. The architecture code is depicted in Figure 2 and the architecture is shown in Figure 1.

3.2. Sequence Classification Model

3.2.1 Dataset Preprocessing

The preprocessing step in this task is similar to that of Single Character Classification.

3.2.2 Architecture

Our proposed CRNN model used the convolution block (with a replacement of batch norm layer with instance normalisation layer) from our single character classification

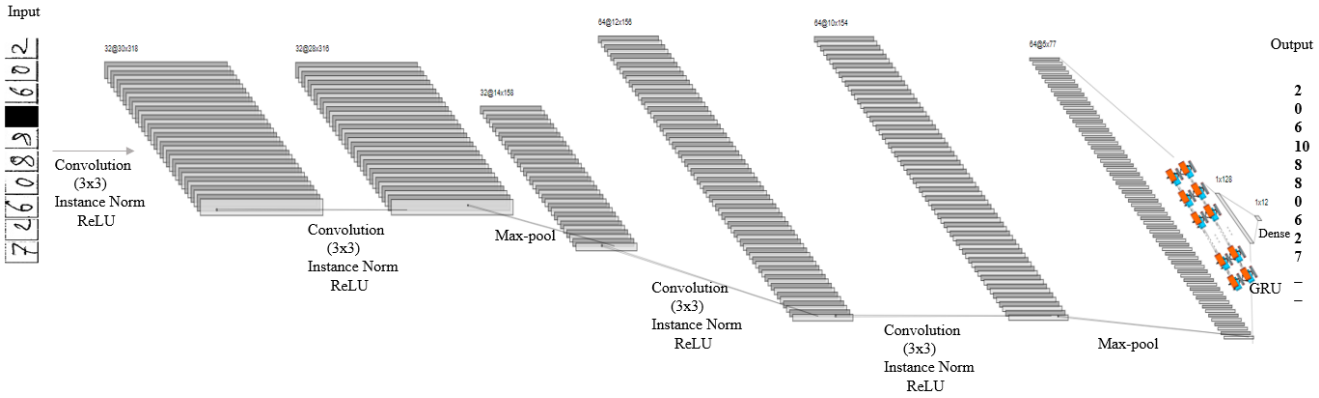

Figure 3. Sequential Classification Model Architecture

```
self.layer1 = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=(3, 3)),
    nn.InstanceNorm2d(32),
    nn.ReLU(),
)

self.layer2 = nn.Sequential(
    nn.Conv2d(32, 32, kernel_size=(3, 3)),
    nn.InstanceNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2)
)
self.layer3 = nn.Sequential(
    nn.Conv2d(32, 64, kernel_size=(3, 3)),
    nn.InstanceNorm2d(64),
    nn.ReLU(),
)

self.layer4 = nn.Sequential(
    nn.Conv2d(64, 64, kernel_size=(3, 3)),
    nn.InstanceNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2)
)
self.gru_input_size = cnn_output_height * 64
self.gru = nn.GRU(self.gru_input_size, gru_hidden_size, gru_num_layers, batch_first=True, bidirectional=True)
self.fc = nn.Linear(gru_hidden_size * 2, num_classes)
```

Figure 4. Sequential Classification Model Architecture code

model as the backbone network for extracting relevant feature maps related to digit recognition. The convolution layer's output feature maps were then transformed into a sequence of feature vectors fed to Bidirectional RNN (Gated Recurrent Unit). The Recurrent layer produces probability distribution for output labels over each of the feature vector. We have used GRU (Gated Recurrent Unit) in place of LSTM (Long Short Term Memory) as a bidirectional RNN module since it is computationally efficient as it required fewer parameters than LSTM, and it out-performed the same in terms of accuracy for our desired task.

We cannot use regular cross-entropy loss because the model can generate multiple sequences of valid output for each ground truth. Therefore, we have utilized the Connectionist Temporal Classification (CTC) loss to overcome the above limitation. The length of the sequence feature vector generated by RNN can be equal to or the greater than the length of the total number of class labels; there can be duplicate characters in the final prediction. To solve this concern, the CTC algorithm is used since it is alignment-free, i.e. it does not require the alignment between the output and input. It introduces a new token to the set of allowed prediction. This new token is referred to as the blank class label.

For a given input, CTC encodes the text output from RNN by inserting many blank labels at any position. In the decoding stage, CTC firstly merges the repeating characters and removes the blank label(s), then the remaining characters will be the final predicted output whose length will be equal to that of the expected ground truth. In this manner, it solves the duplicate characters issue. Many to one signify that one or more input elements can align to a single output element but not vice-versa. Moreover, Monotonic alignments denote that if we advance to the next input, we can keep the corresponding output the same or advance to

the next one.
For single input(X) and output (Y) pair, the conditional CTC probability is calculated by sum of product of probability for a single step in RNN's output over all the vaild alignments. Mathematically it can be represented by,

$$p(Y \mid X) = \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p_t\left(a_t \mid X\right) \tag{1}$$

Where, the $\mathcal{A}_{X,Y}$ is the set of valid alignments and $p_t\left(a_t \mid X\right)$ is the probability of class label a_t at a particular time step t of the RNN's output. For a training set $\mathcal{D}$, the model's parameters are tuned to minimize the negative log-likelihood,

$$\sum_{(\mathcal{X},\mathcal{Y}) \in \mathcal{D}} -\log p(Y|X) \tag{2}$$

The architecture code is depicted in Figure 4 and the architecture is shown in Figure 3.

4. Experiments

For all the experiments we have used 20 epochs form training and have reported 5 fold cross validation accuracy as the accuracy score.

4.1. Single Character Classification

We started with the pretrained LeNet model that was trained on MNIST dataset that resulted in an accuracy of 92.77%.
The standard LeNet model uses Tanh as an activation function and does not have batch norm layers. Using ReLu activation function and adding a batch normalisation after convolution layers resulted in an accuracy of 95.32%. Next, we experimented with increasing the number of feature maps in the convolutional layer. As we can see in table 1.
So, we fixed the number of output feature maps from first layer as 32 and the number of output feature maps from second layer as 32. Next, since the receptive field of two 3x3 kernels is same as a 5x5 kernel, with the added benefit of increased non linearity in the network, instead of using convolution layer with a kernel of size 5x5 we performed 2 convolutions from kernel of size 3x3 one after the other. This resulted in an accuracy of 98.92%.
We further explored the idea of increasing the non linearity by using strided (s = 2) second and fourth convolutional layer instead of max pool layers. In some sense, it makes the pooling layer learn-able. Unfortunately, it deprived the performance of the network to 98.27%. Since, we were achieving a train accuracy beyond 99.5%, the less test accuracy is likely due to overfitting. To overcome this we tried dropout as a technique in both the convolutional layers and the fully connected layers. The results for this

O/P Size of Conv1	O/P Size of Conv2	Accuracy
6	10	95.32
16	32	97.60
32	64	**98.16**
64	128	98.03

Table 1. Changing the number of feature Maps

Experiment	Accuracy
FC layer (p = 0.1)	**99.03**
FC layer (p = 0.1)	98.57
FC layer (p = 0.1)	98.57
FC layer+Conv layer before pooling (p = 0.1)	99.02
FC layer+Conv layers(p = 0.1)	98.80

Table 2. Dropout experiments

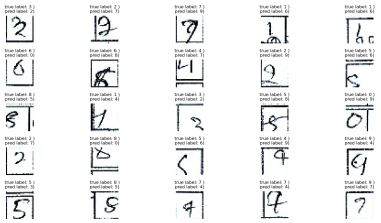

Figure 5. Highest confident Model Miss classifications

is shown in 2. So, we used a dropout rate of 0.1 in the fully connected rate since it gives the best performance. Dropout in convolutional layer causes loss of information since LeNet is a shallow network and there are just 4 convolutional layers in the model.
The other technique to make the network converge to optimum that we have used are learning rate scheduler with decay rate 0.1. We started with the learning rate of 0.0001 and the decreased it by a factor of 10 after 10 epochs. This, again helped in improving the performance of network to 99.19% .
We believed that this accuracy is enough to show that the classifier performs well. To further support this 5 shows the 25 miss classifications by our network where our network was very confident that it has predicted correctly but still there was a miss-classification. As one can see, the miss-classifications are such which can confuse a human too. So, the final accuracy of our model is **99.19%**.

4.2. Sequence Classification Model

Initially, we started with the same convolutional block as that of the single classification model, on top of which we fused GRUs and then fully connected layers. We observed that the training accuracy was about 99%, but the fivefold validation accuracy score was comparatively significantly less. We conclude that this was the case of over-

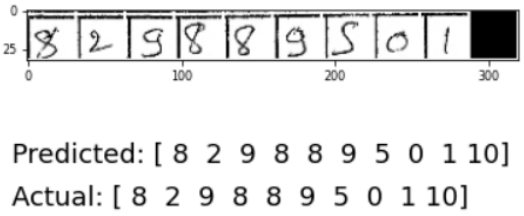

Figure 6. Sequential Classification Result

fitting and confirms that standard batch normalization cannot perform well in RNNs where normalization is applied only ”vertically”, i.e. the output of each RNN. It is not applied ”horizontally”, i.e. between time steps of RNN. [**?**]. Thus, we perform further experiments using the Instance Normalization in the convolutional layer. Then, we have used two different variants of Bidirectional RNN, namely, i.e. GRU and LSTM. Using LSTM, we achieved an accuracy of 93.5%, and with GRU, we got our max reported accuracy of **94.15%**. Despite having fewer parameters, the GRU model was able to achieve a higher accuracy after 20 epochs. The LSTM model displays much greater volatility throughout its gradient descent than the GRU model, which may be because there are more gates for the gradients to flow through, causing steady progress to be more difficult to maintain after many epochs. Results are depicted in Figure 6.

5. Conclusion

In this assignment, we were able to classify a single digit with an accuracy of 99.19% using a convolutional neural network (CNN) and a sequence of mobile number with an accuracy of 94.15% by extending CNN with RNN. The code can be found at github.

References

[1] A. Bissacco, M. Cummins, Y. Netzer, and H. Neven. Photoocr: Reading text in uncontrolled conditions. In *Proceedings of the ieee international conference on computer vision*, pages 785–792, 2013. 1

[2] M. Jaderberg, A. Vedaldi, and A. Zisserman. Deep features for text spotting. In *European conference on computer vision*, pages 512–528. Springer, 2014. 1

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 2

[4] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016. 1