Project: CAPTCHA Code Detection

Submission Date: 29th March 2024

Submitted By: Ahmed Dider Rahat
DSM: 916146

The Goal of the Project

The primary goal of this project is to design and implement a Convolutional Recurrent Neural Network (CRNN) to predict the text content within CAPTCHA images accurately. This involves processing the visual information through convolutional layers to capture the spatial hierarchies in the images and then utilizing recurrent layers to decode the sequential information into a string of text. The project aims to handle a variety of CAPTCHA styles and complexities with high accuracy.

About the Dataset

The dataset is open source and downloaded from Kaggle.

Data Source Link

The dataset can be found on: the link

Properties of the Dataset

- The dataset contains 1070 captcha images.
- All the images have the original labels as a file name.
- Each of the images contains a 5-digit captcha code.

Example Images from the Dataset

2pfpn	3ny45	bgd4m

Implementation Process

The project has three different portions: pre-processing, model training, post-processing

Pre-Processing

The pre-processing section contains all the image processing tasks, and converting images to tensor input. The sub-tasks are described below:

Image Loading and Label Extraction

1. The image is loaded in the environment.
2. Extract the labels from the name of each file.
3. There are a total number of 19 alpha-numeric characters present in the dataset. Each file consists of any 5 combinations from these characters.
4. As I will use Neural Network for this project, I create a map of each character. The map has a unique number(0-18) for each character.

Data Set Splitting

For model training and evaluation I need to split the dataset into 3 portions i.e. Training set (80%), Validation set (10%), and Test set (10%). The number of images in each dataset is:

```
Number of Train set size: 855
Number of Validation set size: 108
Number of Test set size: 107
```

Create DataLoader objects

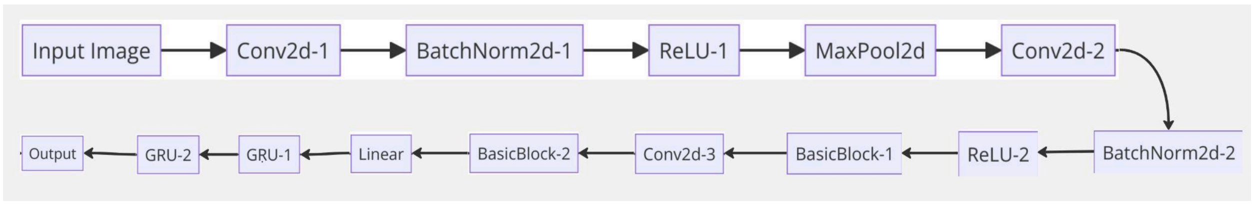
- Data loader objects are created for training, validation, and test datasets.
- For that, I created a custom dataset builder that converts the images to PyTorch tensors and normalizes them with preset mean and standard deviation values, which are standard for pre-trained models trained on the Resnet dataset.

Model Training

I used the CRNN model to predict the CAPTCHA code from the images. The foundation of this model is a pre-trained ResNet-18, which is adapted and extended with custom convolutional and recurrent layers to handle the complexities of OCR tasks.

Model Architecture

The CRNN model is used to exploit both spatial features of text images through convolutional neural networks (CNNs) and sequence dependencies via recurrent neural networks (RNNs).



The model architecture integrates the following components:

Pretrained ResNet-18 Backbone

The initial layers are derived from a pre-trained ResNet-18 model, ensuring robust feature extraction capabilities. This subset of the ResNet architecture excludes the final three layers, optimizing it for the OCR task's unique requirements.

Custom CNN Layers

Following the ResNet backbone, additional convolutional layers are introduced to refine and adapt the feature maps for OCR purposes. This includes a notable convolutional layer with specific kernel size and padding, followed by batch normalization and ReLU activation.

RNN Layers

To capture the sequential dependencies of text, two GRU layers are employed. These bidirectional GRUs effectively process the textual information, considering both forward and backward context within the text sequences.

Linear Transformation

The output from the RNN layers undergoes linear transformation to map the deep features to character predictions, facilitating the decoding of textual content from images.

Dropout

Dropout is incorporated to mitigate overfitting, enhancing the model's generalization capability.

Initialization and Training

Weight Initialization

A rigorous weight initialization strategy is employed, leveraging Xavier initialization for linear and convolutional layers, and normal distribution for batch normalization layers. This ensures a balanced start for the training process.

Training Setup

The model is trained with a specific focus on OCR tasks, involving hyperparameters like learning rate, weight decay, and gradient clipping to manage the training dynamics effectively.

```
num_epochs = 50
learning_rate = 0.001
weight_decay = 1e-3
clip_norm = 5
```

Optimization

An Adam optimizer combined with a learning rate scheduler aids in adjusting the learning rate based on validation loss, optimizing the training process for better convergence.

Loss Function

The CTCLoss function is used, catering specifically to the sequence prediction nature of OCR, allowing the model to handle varying lengths of text efficiently.

Training Process

The training process involves multiple epochs where the model is exposed to batches of text images. During each epoch, the model undergoes optimization to minimize the loss on the training set, followed by evaluation on a validation set to monitor the loss.

Epoch-wise Training

Each epoch consists of a training phase where the model's parameters are updated, and a validation phase to assess performance on unseen data. This cyclical process helps in fine-tuning the model for better accuracy in text recognition.

Gradient Clipping

To prevent the exploding gradient problem, gradient clipping is implemented, ensuring stable and effective training.

Learning Rate Adjustment

The learning rate scheduler dynamically adjusts the learning rate based on the validation loss, enabling the model to adapt its learning pace for optimal performance.

Post-Processing

This section details the post-processing steps of the Optical Character Recognition (OCR) model designed to recognize and decode text from images. The model's raw predictions are refined through a post-processing pipeline to improve accuracy and reduce prediction errors. This process is critical for enhancing the model's practical usability by addressing common OCR challenges, such as character duplication and segmentation errors.

Post-processing Pipeline:

Prediction Results Gathering

For the training, validation, and test datasets, predictions are generated using a dedicated function that processes each batch of images through the model, decodes the logits to text using a custom decoding function, and collates the actual and predicted text sequences into a comprehensive DataFrame. This step is crucial for systematically evaluating and comparing the model's performance across different data subsets.

Duplicate Character Removal

A common issue in OCR predictions is the duplication of characters due to the model's uncertainty or the characteristics of the input images. A specialized function is applied to each predicted string to eliminate consecutive duplicate characters, thus cleaning the predictions. This step significantly improves the readability and accuracy of the model's output.

Post-processing Correction

Another layer of post-processing involves correcting segmented predictions. Predicted strings are split and corrected for internal duplications, further refining the model's output. This is especially useful for correcting errors introduced by character segmentation during the model's prediction phase.

Result Analysis

Interpretation of the Loss Curve

The loss curve of the model provided displays the training and validation loss of a model over 50 epochs.

1. **Initial Rapid Decrease:** At the beginning of training, both training and validation loss decrease rapidly. This suggests that the model is learning quickly and that the initial weights, possibly due to effective initialization, are being adjusted in a way that significantly reduces loss.

2. **Convergence Trend:** After the initial sharp decline, the rate of decrease in loss slows down for both curves. This is expected behavior as the model starts to converge to a solution and the potential for improvement decreases.

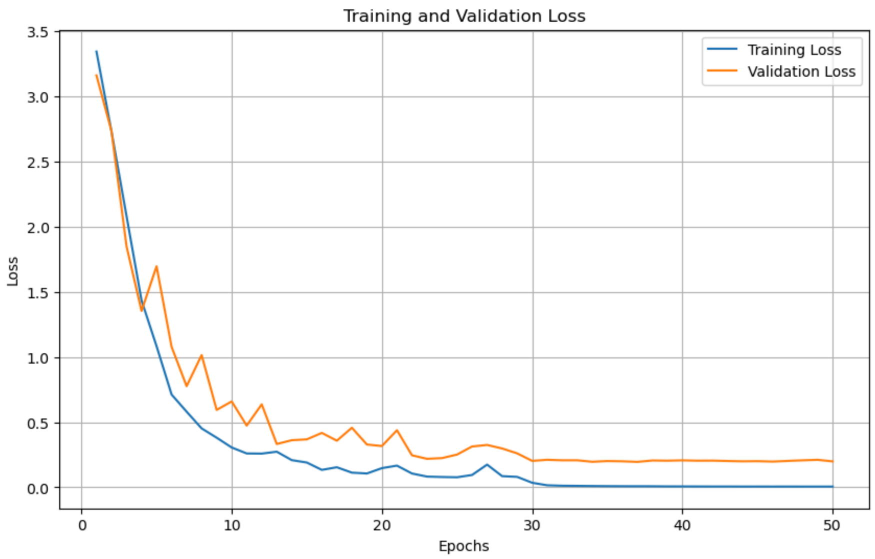


Figure-Tranig and Validation loss of the model

3. **Closeness of Curves:** It is noticeable that the training and validation loss are quite close to each other throughout the training process. This closeness indicates good generalization, as the model doesn't appear to be overfitting to the training data.

4. **Stability after Convergence:** Both curves level out as they approach the later epochs, showing only a minimal decrease in loss. This suggests that the model has largely converged and is making only slight improvements in minimizing the loss function.

5. **End of Learning:** By the final epochs, both curves are relatively flat, which typically suggests that the model will not benefit from further training with the current data and hyperparameters; it has reached its capacity to learn from the given dataset.

The snippet of the correct prediction on the test set

actual	**prediction**	**Prediction (after post-processing)**
w4cdc	w4----cd-c	w4cdc
n4xx5	n4----xx-5	n4x5
e76n4	e7----6n-4	e76n4
ddcdd	dd--d-cd-d	ddcdd
bbymy	bb----ymmy	bymy

The snippet of the wrong prediction on the test set

actual	**prediction**	**Prediction (after post-processing)**
n4xx5	n4----xx-5	n4x5
bbymy	bb----ymmy	bymy
yyg5g	yy----g55g	yg5g
emwpn	em-----wpp	emwp
mx8bb	m-x----8bb	mx8b

Model Accuracy

The accuracy of the model in different sub-sets of the data is given below:

```
--------------------------------------------------------
Model Accuracy on training set:  0.999
Model Accuracy on validation set:  0.806
Model Accuracy on test set:  0.813
--------------------------------------------------------
```

Conclusion

The project aimed to develop an Optical Character Recognition (OCR) model using a Convolutional Recurrent Neural Network (CRNN) architecture. Through extensive training, post-processing, and performance evaluations, the model was refined to interpret and decode text from images with high accuracy. The model has achieved impressive accuracy, demonstrating near-perfect performance on the training set with an accuracy of 99.9%. This high level of accuracy indicates that the model is highly capable of learning from the training data provided. In contrast, the model's performance on the validation and test sets is lower, with accuracies of 80.6% and 81.3%, respectively. While these figures are substantial, they are notably less than the training set performance, suggesting a gap in the model's ability to generalize to unseen data compared to its performance on the training data.

References/Links

1. Project Link: https://github.com/AhmedDiderRahat/captcha_detection/tree/main
2. Presentation Link: https://drive.google.com/drive/folders/14j64s7F32yO5qgSauFk2rZbUT1PbCTl_?usp=sharing