Report

Precog Recruitment Task - Break The CAPTCHA

Name: Krishak Aneja
Mail: krishak.aneja@students.iiit.ac.in

Task Coverage & Decisions

Overview of Task Selection:
Due to the highly interesting nature of the available tasks, I spent the first three days deliberating among the four challenging options:

- NLP and Responsible AI
- Atlas Graph task
- Math+AI
- CAPTCHA Breaking via OCR (the current task)

I ultimately chose the CAPTCHA task because it aligned best with my interests in multimodal systems—as emphasized in my Statement of Purpose—and the challenge of building a solution from the ground up (None of the prerequisite knowledge has yet been taught/covered in our courses and this is in the truest sense a new field for me to explore). This task also neatly complemented my studies in Information Security in ML (with a focus on CNN-based security) from my Introduction to Information Security course.

Parts of the Task Attempted:

- **Task 0 – Dataset Generation:** I synthesized a dataset consisting of (input=image, output=text) pairs, including both an Easy Set (fixed font, capitalization, plain background) and a Hard Set (multiple fonts, fluctuating capitalization, noisy/textured backgrounds).
- **Task 1 – Classification:** I selected a subset of the dataset containing 100 words (classes) and trained a neural classifier using a CNN architecture inspired by TinyVGG and LeNet. I experimented with the number of samples per class, various hyperparameters, and model structures to evaluate performance.

- **Task 2 – Generation:** I extended the CNN architecture to handle variable-length text generation by integrating an RNN (GRU/LSTM) decoder, forming a CRNN model. I experimented with the Connectionist Temporal Classification (CTC) loss to manage alignment between image features and sequential text output.

Parts of the Task Not Attempted:

- **Task 3 – Bonus:** Although I developed a conceptual approach for the bonus task, I did not have sufficient time to implement this part. I hope to explore revisit it later.

---

1. Introduction

The challenge of breaking CAPTCHAs using deep learning techniques pushes the boundaries of classical Optical Character Recognition (OCR) methods, especially when dealing with noisy backgrounds, diverse fonts, and variable text lengths. This project was designed to not only work towards the completion of the task but also to foster a deeper understanding of neural network design through a step-by-step experimental approach.
I prefer to have complete knowledge of what I am doing and accomplish little with understanding rather than doing a lot haphazardly without real understanding or overreliance on AI tools.
Over several experiments, I refined my methodology—from dataset synthesis to CNN classification and eventually to CRNN-based text generation.

Throughout the experiments, I adhered to a scientific methodology: visualizing data and model outputs, monitoring performance metrics, and iteratively adjusting hyperparameters to gain intuition about the underlying processes.

---

2. Methodology

2.1. Preparation and Learning Resources

Before diving into the implementation, I dedicated several days to study the theory behind convolutional and recurrent neural networks. I followed free courses and tutorials, including:

- **3Blue1Brown's Deep Learning Playlist:** For conceptual understanding of neural network functioning.
- **LearnPyTorch.io:** To gain practical PyTorch skills.
- **MIT Deep Learning Course (YouTube):** To understand CNNs and RNNs in particular.

This preparatory work laid the foundation for a methodical approach to dataset synthesis, model design, and hyperparameter tuning.

2.2. Dataset Synthesis (Task 0)

The synthesized dataset was split into:

- **Easy Set:** Single-word images rendered in a fixed font, uniform capitalization, and plain white background.
- **Hard Set:** Images rendered with multiple fonts, varying capitalization on individual letters, and noisy/textured backgrounds.

The decision to start training on the Easy Set and subsequently move to the Hard Set proved pivotal. Training first on the easy examples allowed the model to establish strong class-specific representations. Directly training on the hard examples, by contrast, often led to difficulty in escaping poor local minima.

2.3. Experiments and Observations/Inferences

Experiment 1: CNN for Classification (Task 1)

Architecture:
I designed a CNN architecture inspired by the VGG-style (TinyVGG) and LeNet models. Key elements included:
- Multiple convolutional layers for feature extraction.
- One fully connected layer that maps extracted features to a 100-class output.

Hyperparameter Experiments:
I experimented with:
- **Dataset Size:** Initially, 40 images per class for 10 classes, then expanding to 20 classes with 250 images per class (comprising both easy and hard sets) and finally 100 images per class for 100 classes.
- **Hidden Units:** I tested various numbers (e.g., 16, 24, 32). Notably, I observed that 24 hidden units (and to some extent 16) worked best, while anything other than 24 units (even off by one like 23 or 25) resulted in a training loss that stagnated at 4.605.

Observations on Hidden Units:
The fact that only multiples of 8 (specifically 16 and 24) yielded reasonable performance is intriguing. It may be attributed to:
- **Hardware and Memory Alignment:** Convolution operations and tensor computations in PyTorch often perform optimally with dimensions that are multiples of 8.
- **Network Capacity:** A balance between model capacity and overfitting. Too few hidden units may underfit, whereas too many (e.g., 32) could lead to over-parameterization and difficulty converging, as indicated by a loss value stuck around 4.605.

- **Training Strategy:**
  By training on the Easy Set first, the CNN established a solid baseline representation. Subsequent fine-tuning with the Hard Set allowed the model to generalize better, leading to consistent improvements.
  Analysis: The easier examples likely provided a clearer gradient signal early in training, allowing the model to better establish initial representations before being exposed to more complex examples.

Experiment 2: CRNN for Text Generation (Task 2)

Architecture Extension:
For text generation, I extended the CNN with a sequential decoder:
- CNN Layers: **The same convolutional layers as in Experiment 1 were used to extract spatial features.**
- RNN Component:** An RNN (experimented with both LSTM and GRU variants) was used to model the sequence of characters, handling variable-length text outputs.

Loss Function – CTC (Connectionist Temporal Classification):
Given that text outputs are sequences without a one-to-one alignment with the input image segments, CTC loss was employed. CTC loss provides a mechanism to calculate the probability of the target sequence by summing over all possible alignments, eliminating the need for explicit segmentation.

Character Error Rate (CER):
Instead of standard accuracy, I evaluated performance using Character Error Rate (CER). Particularly accuracy here is letter accuracy which I define as 1 - CER.
CER is defined as the ratio of the number of character errors (insertions, deletions, substitutions) to the total number of characters in the ground truth. This metric is more appropriate for OCR tasks because:
- **Variable-Length Output:** Accuracy per word can be misleading when dealing with sequences of different lengths.
- **Granularity:** CER provides a fine-grained assessment of how many characters are misclassified, making it easier to pinpoint the nature of errors in transcription.

Experimental Insights:
After integrating using GRU instead of LSTM and L2 Regularization (weight decay):
- The test loss improved from 1.5 to 0.9 and test accuracy increased from 75% to 85% (within the same 100 classes).
- When the model was tested on datasets with entirely new classes or new styling (fonts), performance dropped dramatically (to around 22% and 11%, respectively), indicating significant overfitting.
- Additional experiments with L2 regularization and dropout provided some improvement but

underscored that the 100-class dataset was insufficiently diverse for robust generalization. Thus the only remaining option was to expand the dataset- particularly the number of unique words. (I realized quite late that 100 words is quite the measly amount for this task). I constructed a new dataset that contains 10000 distinct words(30k if you count distinct capitalizations). The new dataset wasn't quite as simple as the easy set yet wasn't as augmented as the hard set either. I had to keep the variations in fonts minimal in order to conserve the size of the full dataset.
The model trained on this expanded dataset was alleviated of overfitting, and yet was not as robust to augmentations.
Given more time, the next natural step would be to strike the right balance between number of unique words and the number of variations per word.

---

3. Bonus Task – Proposed Approach

Although the bonus task was not implemented, I propose the following strategy for future work:

- **Challenge:** The bonus set introduces an extra complexity—if the background is green, the word is rendered normally; if red, the word is rendered in reverse.
- **Naïve Preprocessing Issue:**
  - Simply preprocessing the images (identifying the color of the background manually and flipping text where necessary) risks misaligning the model's internal logic with the loss function. Example: The model might predict "elloh" for a red-background image. The label (which we manually flipped back to "hello") is "hello". The loss function penalizes this output because it expects "hello", even though "elloh" was actually correct before flipping.
  - Thus, The model is wrongly penalized for learning the right thing (flipping the reversed text). Over time, the model may learn random workarounds instead of actually understanding the rule of the task.
- **Proposed Architectural Modification:**
  - **Parallel CNN Branch:** Incorporate a branch dedicated to color detection. This branch would learn to identify the background color and, consequently, determine whether the text should be interpreted in its original or reversed order.
  - **Multi-Task Learning:** Integrating color detection into the main network's loss function would encourage the model to learn both tasks simultaneously, ensuring that the transformation logic (reversal of text) is properly integrated.

---

4. Conclusion

This project provided an invaluable deep dive into the design and training of neural networks. To build a neural network for a given task is to embed—or rather, program—the capabilities and logic the model requires into its **architecture**, **hyperparameters** and the **data** used to train it.

- ***Architecture*** refers to the structural design of the model—what components exist and how they are connected. This includes the type and number of layers (e.g., convolutional, recurrent, fully connected), the order and connections of layers (how data flows through them) and the choice of mathematical operations (activation functions, normalization layers, SoftMax).
- ***Hyperparameters*** are the configurable settings that define how the model learns and are not part of the model's fixed structure. These include Layer widths (number of neurons per layer), Dropout rate, Learning rate, Optimizer choices (Adam,SGD), Batch size and Regularization settings(such as Temperature in SoftMax).
- ***Data*** determines how much of the base architecture's potential is realized. Its quality (variety, relevance) and quantity (dataset size) directly impact performance.

And the choice of these 3 pillars is closely linked— they work in tandem. Unlike the traditional programming I have experienced so far, where logic is explicitly coded, neural network engineering requires a more nuanced approach—one where experience and intuition play a crucial role. It blurs the boundary between science and art, a fusion I deeply enjoy, as I've expressed in my SoP.

---

5. References

1. **3Blue1Brown's Deep Learning Playlist**
   [YouTube Channel](https://www.youtube.com/channel/UCYO_jab_esuFRV4b17AJtAw
2. **LearnPyTorch.io**
   LearnPyTorch
3. **MIT Deep Learning Course**
   MIT Deep Learning on YouTube
4. **TinyVGG & CNN Explainer**
   CNN Explainer
5. **CAPTCHA Recognition Using Convolutional Neural Network**
   Medium Post
6. **CTC Loss Explained**
   Distill.pub – CTC

7. **PyTorch Documentation**
   https://pytorch.org/docs/stable/torch.html
8. **Several other blog posts:**
   https://towardsdatascience.com/8-simple-techniques-to-prevent-overfitting-4d443da2ef7d
   https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008
   https://colab.research.google.com/drive/1elYSajW0_qxA_6k-B8w4TGR5ec8vaw5f?usp=drive_link#scrollTo=TM7hyGz3Ry8e
   https://medium.com/analytics-vidhya/image-text-recognition-738a368368f5
   https://keras.io/examples/vision/captcha_ocr/
   https://medium.com/towards-data-science/intuitively-understanding-connectionist-temporal-classification-3797e43a86c