Precog Recruitment Task
Can you break the CAPTCHA?

Arihant Rastogi

February 10, 2025

1 Tasks Attempted

The following tasks were attempted:

- Task 0 - Dataset Generation
- Task 1 - Classification
- Task 2 - Generation

2 Task 0 - Dataset Generation

The implementation for this part has been done in `task0.ipynb`. I have used the Pillow library of Python to generate the dataset and `random_word` to generate random words.

The words were generated with lengths between 5-9 characters to ensure that the text is contained within the image without overflowing.

2.1 Easy Set

- A fixed font is used throughout the dataset.
- Text is positioned randomly in the image for variation.
- Text is black, and the background is white.
- Grayscale images are generated faster, allowing work with larger datasets.

2.2 Hard Set

The following features were added:

- Random capitalization.
- Text color variation with random colors and fonts.
- Noise in the form of random colored dots.
- Centered text to prevent overflow due to varying fonts.
- Background is white.

2.3 Bonus Set

Additional features include:

- Background color set randomly to green or red.
- If the background is red, text is rendered in reversed order.
- Text color is black to ensure significant contrast with the background.

3 Task 1 - Classification

The implementation for this part has been done in `task1.ipynb`. I have used the PyTorch library to train a Convolutional Neural Network (CNN).

3.1 Why Use CNN for Classification?

- Automatically learns hierarchical features (edges, curves, textures) without manual feature engineering.
- Pooling layers help retain character recognition despite shifts, rotations, or resizing.
- Preserves spatial relationships, making it robust to variations in font, style, and size.

3.2 Architecture

The basic architecture was adapted from GeeksforGeeks (referenced in README). The model consists of three repeated convolutional blocks:

- **Low-level**: Learns edges, textures, and simple shapes.
- **Mid-level**: Detects curves, corners, and patterns.
- **High-level**: Recognizes full characters and words.

Each block consists of:

- Convolutional layer for feature extraction.
- `BatchNorm2d` and `MaxPool2d` to reduce computation and training time.
- ReLU activation to introduce non-linearity and prevent vanishing gradient.

Additional improvements include:

- **Dropout** to prevent overfitting.
- **GradScaler** and **autocast** (Automatic Mixed Precision) for faster training and better memory efficiency.
- Cross-entropy loss function for backpropagation and weight updates.

3.3 Hyperparameter Tuning

A test loop was used to determine optimal hyperparameters:

- **Dropout Rate**: 0.2 provided the best balance, preventing overfitting.
- **Activation Function**: ReLU performed best (Sigmoid and Tanh led to information loss).
- **Dataset Size**: 100-125 images per label for easy set; ¿200 per label for hard and bonus sets.
- **Train-Test Split**: 80:10:10 (Train:Validate:Test) yielded optimal performance.

For the hard and bonus sets, modifications were made to handle RGB images.

4 Task 2 & 3 - Generation

The implementation for this part has been done in `task2.ipynb`. I have used the PyTorch library of Python to train a Convolutional Recurrent Neural Network.

The CRNN (Convolutional Recurrent Neural Network) architecture is designed for sequence-based tasks, especially for Optical Character Recognition (OCR) in scenarios where text appears in images. This makes it highly suitable for recognizing words without the need for segmentation into individual characters.

Truth be told, this was the easiest architecture to understand, train and change, as it can be easily understood as an extension of CNN.

The basic architecture was taken from XENON Stack (referenced in the README). The layers remain the same and their function is the one that is highlighted by the linked article.

One different layer from the CNN is the Recurrent layer which is used for sequence modelling.

- After the CNN extracts spatial features, the feature map is reshaped to a sequence representation:

    - The Height dimension (H) is collapsed, and the feature map is reshaped into a (B, W, C) format, treating each column of the image as a time step in a sequence.

- This sequence is passed into a Bidirectional LSTM (Long Short-Term Memory), which helps in modelling context dependencies across time steps.
- Using a bidirectional LSTM ensures the network understands both left-to-right and right-to-left dependencies, which is critical for recognizing text.

4.1 Why This Architecture?

- No Need for Character Segmentation
- Handles Variable-Length Sequences
- Robust to Distortions and Noises

The CTC (Connectionist Temporal Classification) loss is used instead of Cross Entropy Loss because it helps the model map variable-length input sequences to shorter output sequences by introducing:

- Blank tokens (-) to allow flexible spacing
- Collapsing repeated characters
- Computing all possible alignments of the predicted sequence with the target sequence and assigning probabilities.

A new data loader function called `collate` is added as that is not found in the primary PyTorch library. It is changed for handling the variable text lengths. The `decode_predictions` function is used to convert the model's raw output into readable text. This step is necessary because the CTC Loss does not predict each character in order; instead, it outputs a probability distribution over possible characters at each time step.
Here are the outputs.

- Easy $\rightarrow$ 99.6%
- Hard $\rightarrow$ 24.40%
- Bonus $\rightarrow$ 44.0%

The sudden drop in the accuracy is because we are checking full word accuracy but going by characters, the model works much better. The weird change in accuracy between hard and bonus are due to the previous issues of contrast.