DeepCaptcha: A Convolutional Neural Network for Image-Based Captcha Recognition

Abstract - In an era of increasing online security concerns, Image-Based Captchas (Completely Automated Public Turing test to tell Computers and Humans Apart) serve as a vital defense against automated bots. The proposed model architecture - DeepCaptcha is built upon multiple layers of convolutional and pooling operations, coupled with batch normalization and dropout techniques to enhance stability and prevent overfitting. The network is designed to accommodate the diversity of captcha designs and variations, ensuring adaptability to different security measures. The training process involves optimizing the model using categorical cross-entropy loss and the Adam optimizer. Also, regularization techniques such as L1 and L2 regularization, alongside dropout layers, contribute to the model's generalization capabilities and robustness.It demonstrates promising results in decoding captchas, achieving high accuracy on both training and validation datasets.

[1] PROPOSED ALGORITHM

1.1. PREPROCESSING

The initial dataset for captcha recognition comprised a modest sample size of 365 images. Recognizing the need for a more extensive dataset for robust model training, I employed a data augmentation technique to increase the sample size to 1500.

IncreaseDatasetSampleSize

Input:

- Image directory: Path to the directory containing original images.
- TargetSamples: Desired total number of samples.

Output:

- X: Array containing input images.
- y: Array containing corresponding labels.

Initialize Arrays:

- Initialize X as a zeros array with dimensions (targetSamples, IMG_HEIGHT, IMG_WIDTH, CHANNELS).
- Initialize y as a zeros array with dimensions (targetSamples, CAPTCHA_LENGTH, character_count).

Process Images:

- Initialize count to 0.
- For each image in the list of files in the directory:
    - Break the loop if count reaches targetSamples.
    - Read the grayscale img from the file path.
    - Extract label by removing the file extension.
    - If the length of label is equal to 5:
        - Normalize the pixel values of img to the range [0, 1].
        - Reshape img to (IMG_HEIGHT, IMG_WIDTH, CHANNELS).
        - Initialize the target as a zeros array with dimensions (CAPTCHA_LENGTH, character_count).
    - For each character k at index j in label:

        - Find the character index and set the corresponding element in target to 1.
    - Set the count-th element of X to img.
    - Set the count-th element of y to target.
    - Increment count by 1.

Data Augmentation: Duplicate Samples:

- While count is less than targetSamples:
    - Set the count-th element of X to the element at (count mod imageCount) in X.
    - Set the count-th element of y to the element at (count mod imageCount) in y.
    - Increment count by 1.

Output Result: Return X and y as the final augmented dataset.

1.2. SELECTED PARAMETERS

The key training parameters and callbacks include:

Parameters:

➔ batch_size: The number of samples processed in each iteration during training, i.e. the model will update its weights after processing 8 samples.
➔ epochs: The number of times the entire training dataset is passed forward and backward through the neural network. In my case, I set it to 250.

Callbacks:

I. ReduceLROnPlateau:

- monitor='val_loss': Monitors the validation loss.
- factor=0.1: Reduces the learning rate by a factor of 0.1 when triggered.
- patience=3: Waits for 3 epochs with no improvement before reducing the learning rate.
- min_lr=0.000001: Sets the minimum learning rate.

II. EarlyStopping:

- monitor='val_loss': Monitors the validation loss.
- patience=5: Waits for 5 epochs with no improvement before stopping the training.
- restore_best_weights=True: Restores the model weights from the epoch with the best validation loss.

These callbacks are useful for adjusting the learning rate dynamically and stopping the training early if the model performance on the validation set does not improve, preventing overfitting.

1.3. MODEL ARCHITECTURE

➔ Input Layer: The model takes grayscale images as input, with dimensions specified by IMG_HEIGHT, IMG_WIDTH, and CHANNELS.
➔ Convolutional Blocks: Three convolutional blocks are used to capture hierarchical features from the input images. Each block consists of a 2D convolutional layer (Conv2D) with rectified linear unit (ReLU) activation, batch normalization (BatchNormalization), and max-pooling (MaxPooling2D) to downsample the spatial dimensions.
    - Conv1: 16 filters, 3x3 kernel size, L1 regularization (for sparsity).
    - Conv2: 32 filters, 3x3 kernel size, L2 regularization (for weight decay).

- Conv3: 64 filters, 3x3 kernel size, L1 and L2 regularization combined.

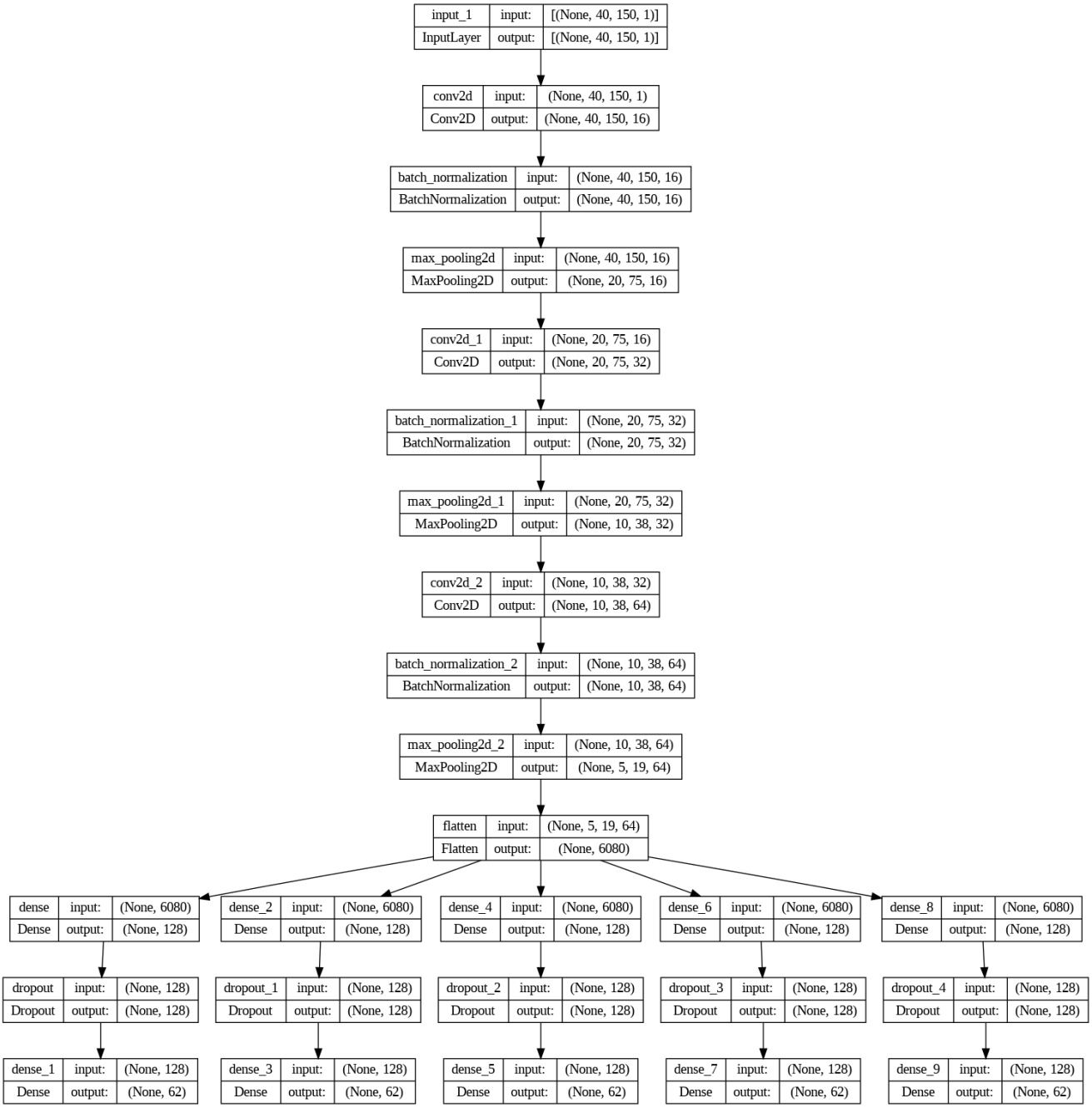
➔ Flatten Layer: Flattens the output of the last convolutional layer to a 1D array.

➔ Fully Connected Layers: For each character in the captcha (specified by CAPTCHA_LENGTH), a set of fully connected layers is applied to capture high-level features and relationships across the flattened representation.

- Dense1: 128 units, ReLU activation, L1 and L2 regularization.
- Dropout: Regularization technique to prevent overfitting (50% dropout rate).
- Dense2 (Output Layer): Produces character predictions using a sigmoid activation function.

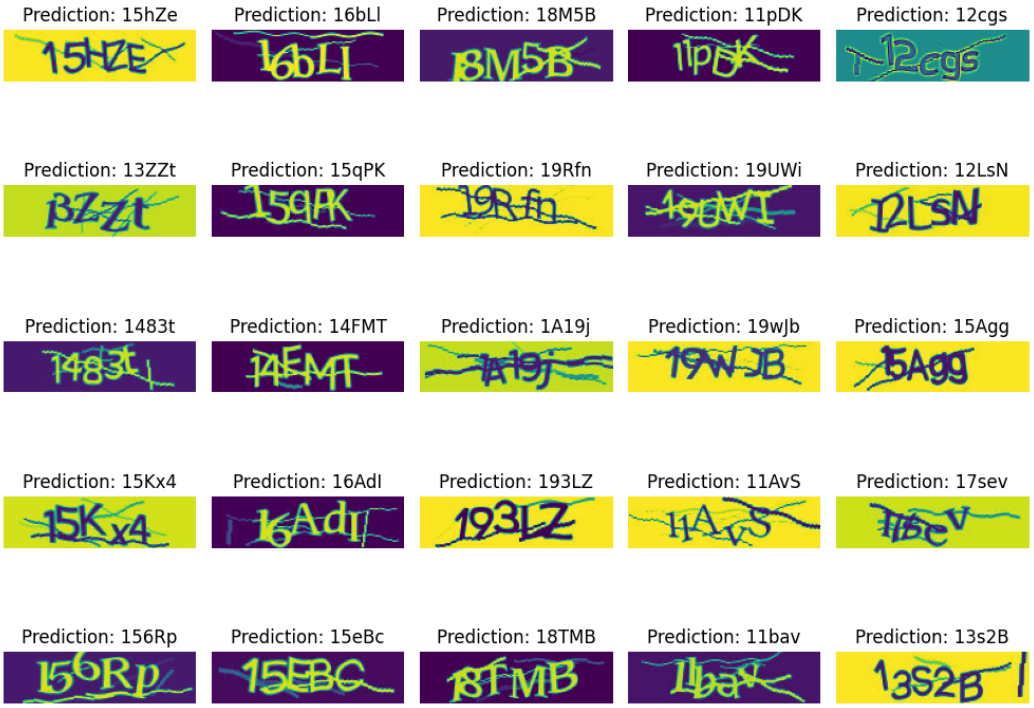
➔ Multiple Outputs: The model has multiple output layers, one for each character in the captcha. This design allows the network to simultaneously predict all characters in a captcha.

➔ Model Compilation: The model is compiled using categorical cross-entropy loss, suitable for multi-class classification problems. The Adam optimizer is used for gradient descent, and accuracy is monitored during training.

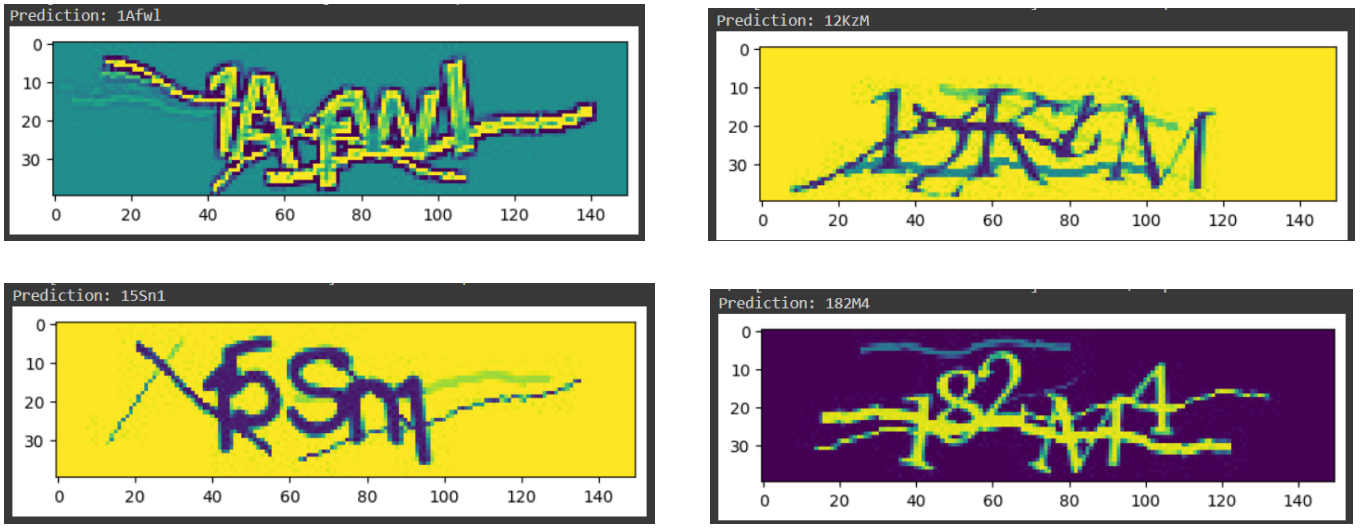


Fig[1] - Model Plot

1.4. VISUALIZATION



Fig[2] Predicted vs Actual Images



Fig[3] Complex captchas Performance

1.5 MODEL EVALUATION

Implemented functions are designed to convert model predictions and ground truth labels into human-readable captcha texts, facilitating the comparison of predicted and actual values during model evaluation.

A. Function return_predictions(preds):

- Input Parameters:
  - preds: The predicted values obtained from the model.
- Functionality:
  - Iterates through each prediction in preds.
  - For each prediction, it extracts the character with the highest probability at each position.

- Constructs the predicted text by mapping character indices to actual characters using the characters list.
- Appends the predicted text to the predictions list.
- Returns the list of predicted texts.

B. Function decode_captcha(y_labels, characters):

- Input Parameters:
  - y_labels: The ground truth labels for the data.
  - characters: The list of characters used in the captcha.
- Functionality:
  - Iterates through each label in y_labels.
  - For each character label, it extracts the character with the highest probability.
  - Constructs the captcha text by mapping character indices to actual characters using the characters list.
  - Appends the captcha text to the captcha_list.
  - Returns the list of captcha texts.

C. Comparison and Printing Loop:

- Transposes y_test to match the shape of predictions.
- Calls return_predictions to get the predicted values.
- Calls decode_captcha to get the actual values from the ground truth.
- Iterates through the number of examples in the test set.
- Prints the predicted and actual values for each example in y_test.

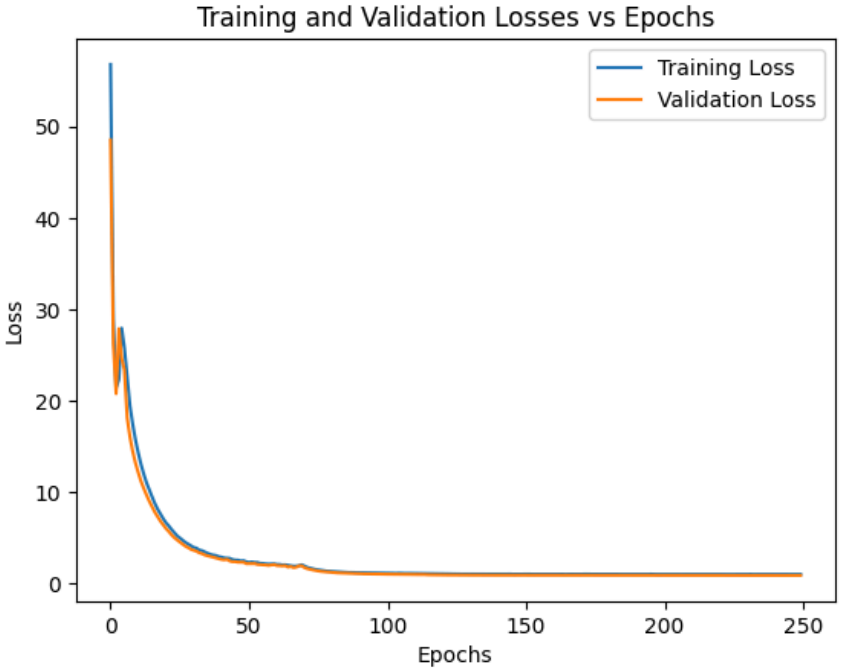
```
Predicted: 15hZe | Actual: 15hZe
Predicted: 16bLl | Actual: 16bLl
Predicted: 18M5B | Actual: 18M5B
Predicted: 11pDK | Actual: 11pDK
Predicted: 12cgs | Actual: 12cgs
Predicted: 13ZZt | Actual: 13ZZt
Predicted: 15qPK | Actual: 15qPK
Predicted: 19Rfn | Actual: 19Rfn
Predicted: 19UWi | Actual: 19UWi
Predicted: 12LsN | Actual: 12LsN
Predicted: 1483t | Actual: 1483t
Predicted: 14FMT | Actual: 14FMT
```

Fig[4] Test set comparison between predicted and actual values

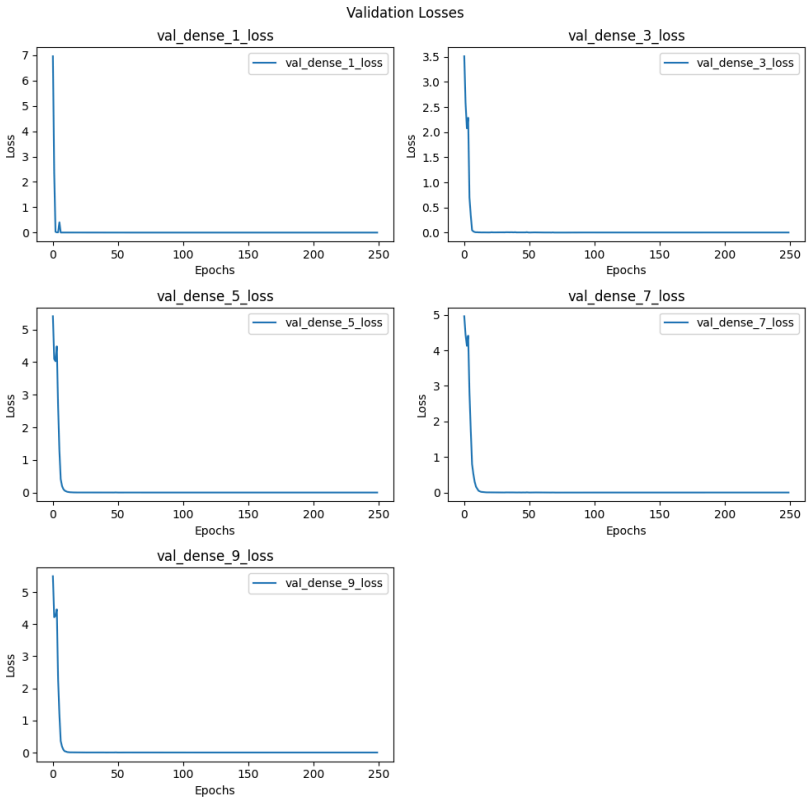
D. Evaluation Plots

Metric Vs Epoch

As visible from the plots, the training and validation losses follow a similar slope and converge. Hence the model has captured the patterns in the data and has generalized well on both the training and validation set. This is attributed to the regularization techniques and hyperparameters.

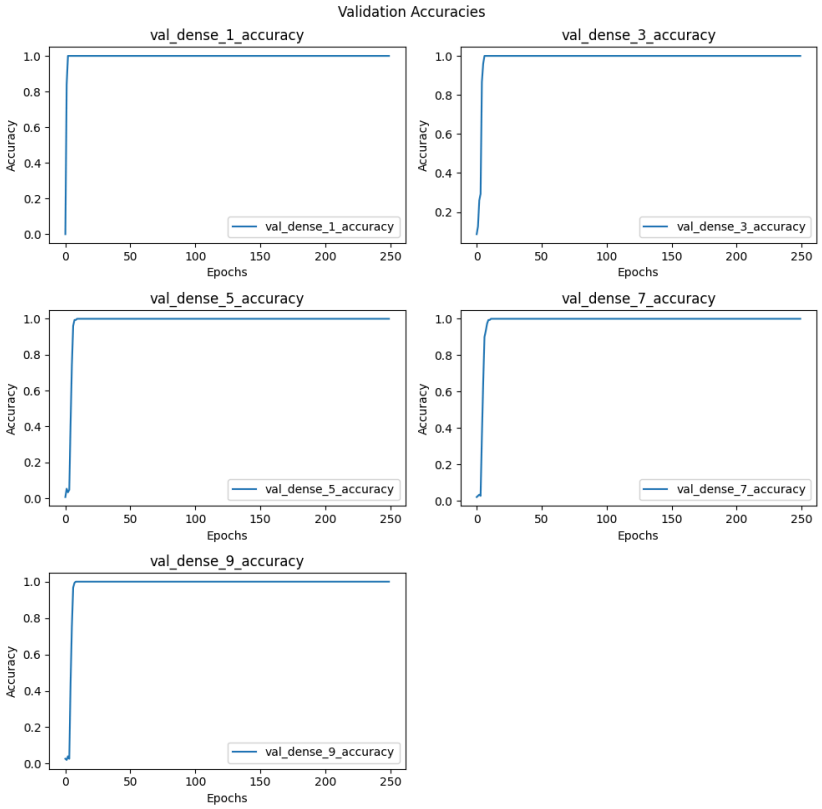


Fig[5] Training and validation loss VS Epochs plot



Fig[6] Validation loss for each Character VS Epochs

The validation loss is decreasing steadily over epochs during training, suggesting that the model is learning and improving.



Fig[7] Validation accuracy for each Character VS Epochs

The validation accuracy is increasing steadily over epochs during training, suggesting that the model is learning and improving.

E. OCR Evaluation Metrics

- Character Error Rate (CER) measures the percentage of incorrectly predicted characters compared to the ground truth, considering insertions, deletions, and substitutions.
- Word Error Rate (WER) extends the evaluation to the word level, calculating the percentage of incorrectly predicted words.

Both metrics provide comprehensive insights into the model's accuracy at character and word levels, essential for tasks like captcha recognition where correct sequence identification is critical. Lower CER and WER values indicate better overall performance.

[2] EXPERIMENTS

Various hyper parameter tuning and regularization experiments were done to select the best model.

	Epochs	**Dataset Size**	**Batch Size**	**Early Stopping Epoch**	**Execution time**	**Training loss**	**Testing Loss**
Model1(without regularization)	200	365	16	36	2min 44s	10.91375	24.98929
Model2(without regularization)	250	400	4	250	47min 23s	1.25643	16.84988
Model3(with l1_l2 regularization)	200	500	16	200	16min 26s	1.95824	1.95977

Model4(with l1_l2 regularization)	150	1500	16	150	38min 32s	1.0944	1.0945
Model5(with l1_l2 regularization)	200	1500	16	200	45min 22s	1.23162	1.23337
Model6(with l1_l2 regularization)	200	1000	16	200	29min 27s	1.37884	1.3792
Model7(with l1_l2 regularization)	200	1500	32	30	5min 49s	8.12204	8.36692
Best Model(with l1_l2 regularization)	**250**	**1500**	**8**	**250**	**1h 21min 27s**	**0.87668**	**0.87829**

Table[2] Comparison of Model Parameters

Some of the key insights from the above models are:

- **Effect of Dataset Size:** The table indicates that as increasing the dataset size, especially with Model4 and Model5 having a dataset size of 1500, the testing loss decreased significantly. Also suggesting that augmenting the dataset led to improved model generalization, as reflected in the lower testing losses.
- **Execution Time and Model Complexity**: Larger datasets and more complex models tend to increase training time. Model2, with a dataset size of 400 and no regularization, has a longer execution time, while Model7, with a smaller dataset but a larger batch size, has a relatively short execution time.
- **Overfitting Control:** Models with l1_l2 regularization (Model3, Model4, Model5, Model6, Model7, and Best Model) tend to perform better in terms of testing loss compared to their non-regularized counterparts (Model1, Model2). This indicates that regularization helps control overfitting.
- **Batch Size Impact:**

A. Model7 with a larger batch size (32) demonstrates a shorter execution time, but it comes at the cost of higher training and testing losses. This highlights the trade-off between execution time and model performance, suggesting that smaller batch sizes may contribute to better generalization.

B. Smaller batch sizes (e.g., Model2 with batch size 4) seem to require more epochs for convergence compared to larger batch sizes. This could be due to the increased variability in the gradients computed from smaller batches.

- **Early Stopping:** Early stopping is an effective technique, as seen in the Best Model. It helps prevent overfitting by stopping training when the validation loss stops improving.
- **Best Model Selection:** The model with l1_l2 regularization, 250 epochs, a dataset size of 1500, a batch size of 8, and early stopping at 250 epochs appears to be the best performer. It achieves the lowest testing loss, indicating better generalization.

[3] RESULTS

Both the curves of training and testing set losses are similar, it suggests that the model is generalizing well to unseen data. The accuracy per character is 100% in the test set.

Character	Accuracy
a	100%
b	100%
c	100%
d	100%
e	100%
f	100%
g	100%
h	100%
i	100%
j	100%
k	100%
l	100%
m	100%
n	100%
o	100%
p	100%
q	100%
r	100%
s	100%
t	100%
u	100%

Character	Accuracy
v	100%
w	100%
x	100%
y	100%
z	100%
A	100%
B	100%
C	100%
D	100%
E	100%
F	100%
G	100%
H	100%
I	100%
J	100%
K	100%
L	100%
M	100%
N	100%
O	100%
P	100%

Character	Accuracy
Q	100%
R	100%
S	100%
T	100%
U	100%
V	100%
W	100%
X	100%
Y	100%
Z	100%
0	100%
1	100%
2	100%
3	100%
4	100%
5	100%
6	100%
7	100%
8	100%
9	100%

Fig[8] Character Vs Accuracy Table from Test Set

Metric	Value
Accuracy	100%
Training set loss	0.8766%
Testing set loss	0.8782%
Character Error Rate (CER)	0.00%
Word Error Rate (WER)	0.00%

Table[2] Final Results

[4] CONCLUSION

From the above experiments, it's essential to consider trade-offs between execution time, model complexity, and performance to choose the most suitable model for deployment. The combination of early stopping, l1_l2 regularization, and the ReduceLROnPlateau callback indicates an approach to model training, incorporating multiple techniques to achieve robust and generalizable results.

[5] LINK

https://colab.research.google.com/drive/1cQtrrbKH__iBcQmBRy7zbXh_6PslTtg5?usp=sharing