

# Machine Learning

## Report

### Handwritten Text Recognition Using CNN

#### Abstract

The main aim of this project is to use model and techniques which are traditionally used on real-life dataset like IAM dataset. I have used Dense layers, LSTM's, CNN's, MaxPooling layers, Dropout layers, Leaky Relu and softmax activation function, Adam optimizer.

In this project, I developed a Hand Text Recognition (HTR) system that effectively digitizes handwritten text using Convolutional Neural Networks (CNNs). Due to several aspects, including picture quality, handwriting quality, and different writing styles, handwritten text identification is a difficult topic in the field of machine learning. By utilizing sophisticated CNN architectures and preprocessing methods, our suggested approach seeks to address these issues and enhance the precision and dependability of handwritten text recognition.

# Introduction

Unlike Handwritten digits recognition(MNIST dataset) which had only 10 classes. Recognizing handwritten writing is a critical problem in many applications, including processing handwritten forms, automating postal services, and digitizing historical documents. This issue is particularly difficult because of the variety in handwriting styles, quality, and picture circumstances. Convolutional neural networks (CNNs), in particular, have demonstrated promising results in solving HTR challenges in recent years.

## Methodology

### 1. Import Dataset

Importing the dataset was very challenging because all the images and labels are in different folder and lot of nested folder are present to access the images. I imported the dataset from <https://fki.tic.heia-fr.ch/databases/iam-handwriting-database>.

```
[ ] def load_and_preprocess_data(data_path):
    labels_path = os.path.join(data_path, 'labels', 'words.txt')
    words_path = os.path.join(data_path, 'words')

    with open(labels_path, 'r') as file:
        lines = file.readlines()

    samples = []
    img_path = []
    for line in lines:
        if line[0] == "#":
            continue
        parts = line.strip().split(' ')
        check = parts[1]
        if check != 'err':
            lab = parts[-1]
            folder1, rest = parts[0].split('-', 1) # Split once to separate the first folder name
            folder2 = rest.rsplit('-')[0]
            folder3 = rest.rsplit('-')[1] + '-' + rest.rsplit('-')[2] # Split once from the right to separate the second folder name and file name

            i_path = os.path.join(words_path, folder1, folder1 + '-' + folder2, folder1 + '-' + folder2 + '-' + folder3 + '.png')

            z = cv2.imread(i_path, cv2.IMREAD_GRAYSCALE)
            if z is not None :
                samples.append(lab)
                img_path.append(i_path)

    #print(img_path)

    return img_path, samples
```

### 2. Data Preparation

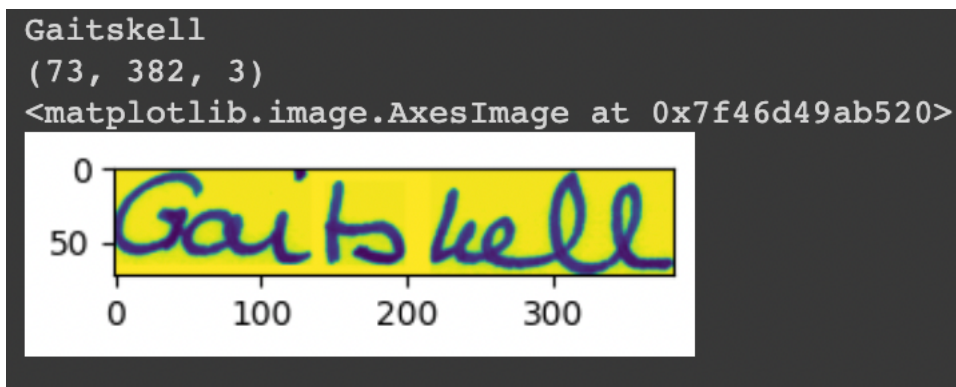
This step took lot of time for me. Because dataset was not labeled properly. All the labels were given in a text file with respective image id's, And images were

given in a separate folder. So, I had to extract all the labels with respective the image.

I preprocessed the images and labels before tokenizing and padding them up. Each image has been assigned as an 3 dimensional array( width x height x channel ). Values of width and height are in the range (0,255) . So, I had to normalize the values to the range (0,1).

```
for i in imgs_path:
    img = cv2.imread(i, cv2.IMREAD_GRAYSCALE)
    if img is not None :
        img = cv2.resize(img, (128, 128))
        img = img.astype("float32") / 255.0
        images.append(img)
```

As you can see above picture, I also re\_sized the image into (128x128). I could have taken different values but, I need 128x128 because, I will be using couple of MaxPooling layers which will re\_size the image at the end. I took the height and width same values because taking different values can cause distortion in image. And, As you can see below (128x128) images were pretty good quality.



As I mentioned before, After resizing the image, I had to collect unique labels with respective to their images. All labels are words (As you can see the below fig), This model is based on predicting the right character at right position. Hence, I also needed to characters dataset which was not provided in the dataset.

```

0s characters = []
max_len = 0
for words in samples:
    for c in [*words]:
        characters.append(c)
        max_len = max(max_len, len([*words]))
    #print(words)

unique = list(set(characters))
print("Max lenght of a word :", max_len)
print("Size of characters:", len(unique))

#labels
samples[:10]

```

```

➞ Max lenght of a word : 21
Size of characters: 78
['A',
'MOVE',
'to',
'stop',
'Mr.',
'Gaitskell',
'from',
'nominating',
'any',
'more']

```

So, I had to extract all the unique characters from the labels and used them to **tokenize** the words.

### 3. Tokenization and Padding

After data preparation, I used tokenization and Sequence Padding to our dataset.

**Tokenization:** In text analysis and natural language processing (NLP), tokenization is a basic step. It is the process of dividing a given text into tokens, which are smaller units. Depending on the application or purpose, these tokens may be single words, phrases, sentences, or even characters.

In this project, even though I used words as labels. I will be tokenizing the characters of the words for our model. Because, I want the model to predict the characters which can later form words rather than some bunch of words.

```

padding_token=99
all_labels_padded=[]
# Create a Tokenizer instance
tokenizer = Tokenizer(char_level=True, oov_token=None)

# Fit the Tokenizer on the characters
tokenizer.fit_on_texts(characters)

print(samples[0])
print(len(samples))
# Convert label to a sequence of integers
for item in samples:
    label_seq = tokenizer.texts_to_sequences([item])

    #print(label_seq)

# Pad the sequence
label_padded = pad_sequences(label_seq, maxlen=max_len, padding='post', value=padding_token)
all_labels_padded.append(label_padded)

```

# Model description

For this dataset, I used a hybrid model such as CNN + LSTM + Dense, Basically this model consists of 9 layers.

## **Convolutional Neural Network**

A particular deep learning architecture called a convolutional neural network (CNN) is made for processing grid-like data, including pictures or time-series data. Convolutional layers, which are intended to automatically and adaptively learn spatial hierarchies of features from the input data, are what define CNNs.

## **Bi Directional Long-Short-term-Memory**

Recurrent neural network (RNN) architectures called bi-directional long short-term memory (Bi-LSTM) networks are made to store both past and future information in a sequence. They work especially well for sequence-to-sequence issues like time-series prediction, speech recognition, and natural language processing. When deep RNNs are trained, the vanishing gradient problem might occur. LSTMs are a form of RNN that solves this issue. Input, output, and forget gates are used by LSTMs to introduce a memory cell and a series of gating mechanisms that enable the network to acquire long-term dependencies and store information over protracted times.

## **Dense Layers**

We know dense layers are in which all units or neurons are connected to each other.

I used 2 CNN layers, 2 Bi-LSTM layers, 1 Dense layers as my main architecture. Additionally, I have used 2 MaxPooling, 2 'Leaky Relu' activation functions for CNN layers. And LSTM have default 'tanh' activation functions. And finally the output layer is the Dense layer with 'softmax' activation.

There is also one layer which is reshape layer, which used between CNN layers and LSTM layers. Used to resize our images from 3 Dimensional to 2 Dimensional as LSTM can only handle 2-D input.

# Experiments & Results

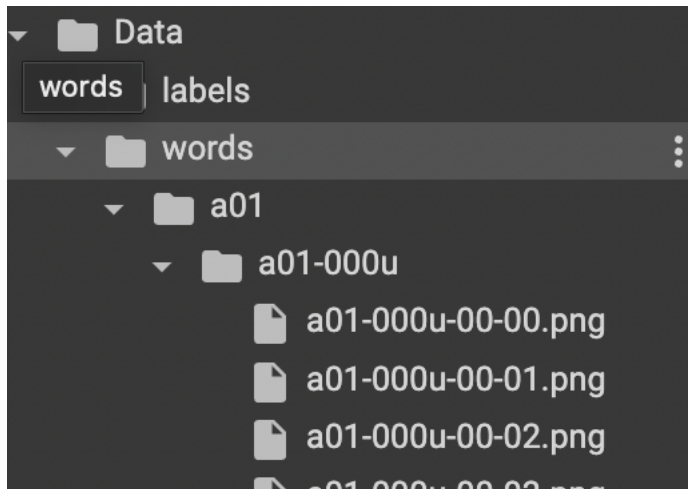
## 1. Database

IAM dataset is the one utilized in this study. The IAM dataset is a great resource for creating and evaluating HTR systems since it includes a wide variety of handwritten text samples from various authors. A popular dataset for studies on handwritten text recognition is the IAM Handwriting Database. The IAM dataset is a great resource for creating and evaluating HTR systems since it includes a wide variety of handwritten text samples from various authors.

This dataset consists of at least 120,000 images and its respective labels. After preprocessing (like removing the inaccurate images). I got the 96,454 number of images and respective labels. Hence, this dataset is quite big and makes it interesting and challenging to try out our machine learning models and technique on them.

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 128, 128, 1)]	0
conv1 (Conv2D)	(None, 128, 128, 64)	640
leaky_relu1 (LeakyReLU)	(None, 128, 128, 64)	0
pool1 (MaxPooling2D)	(None, 42, 128, 64)	0
conv2 (Conv2D)	(None, 42, 128, 128)	73856
leaky_relu2 (LeakyReLU)	(None, 42, 128, 128)	0
pool2 (MaxPooling2D)	(None, 21, 64, 128)	0
reshape_4 (Reshape)	(None, 21, 8192)	0
bidirectional_8 (Bidirectional)	(None, 21, 512)	1730352
bidirectional_9 (Bidirectional)	(None, 21, 512)	1574912
dense_4 (Dense)	(None, 21, 80)	41040

Our dataset includes , images and words in two different folder. One folder consists of all images of written words by different writers. While other folder contains a text file, which has our labels in them.



Images dataset

```
1 #--- words.txt -----
2 #
3 # iam database word information
4 #
5 # format: a01-000u-00-00 ok 154 1 408 768 27 51 AT A
6 #
7 # a01-000u-00-00 -> word id for line 00 in form a01-000u
8 # ok -> result of word segmentation
9 # ok: word was correctly
10 # er: segmentation of word can be bad
11 #
12 # 154 -> graylevel to binarize the line containing this word
13 # 1 -> number of components for this word
14 # 408 768 27 51 -> bounding box around this word in x,y,w,h format
15 # AT -> the grammatical tag for this word, see the
16 # file tagset.txt for an explanation
17 # A -> the transcription for this word
18 #
19 a01-000u-00-00 ok 154 408 768 27 51 AT A
20 a01-000u-00-01 ok 154 507 766 213 48 NN MOVE
21 a01-000u-00-02 ok 154 796 764 70 50 TO to
22 a01-000u-00-03 ok 154 919 757 166 78 VB stop
23 a01-000u-00-04 ok 154 1185 754 126 61 NPT Mr.
24 a01-000u-00-05 ok 154 1438 746 382 73 NP Gaitskell
25 a01-000u-00-06 ok 154 1896 757 173 72 IN from
26 a01-000u-01-00 ok 156 395 932 441 100 VBG nominating
27 a01-000u-01-01 ok 156 901 958 147 79 DTI any
28 a01-000u-01-02 ok 156 1112 958 208 42 AP more
29 a01-000u-01-03 ok 156 1400 937 294 59 NN Labour
30 a01-000u-01-04 ok 156 1779 932 174 63 NN life
31 a01-000u-01-05 ok 156 2008 933 237 70 NNS Peers
32 a01-000u-02-00 ok 157 408 1106 65 70 BEZ is
33 a01-000u-02-01 ok 157 541 1118 72 54 TO to
34 a01-000u-02-02 ok 157 720 1114 113 63 RF he
completed at 2:58 PM
```

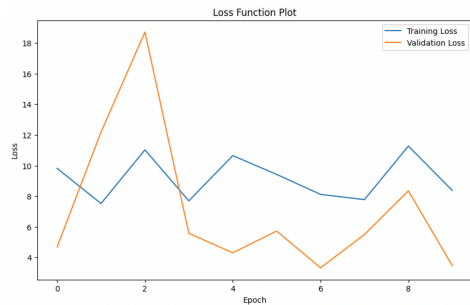
words.text file

## 2. Training and Testing logs

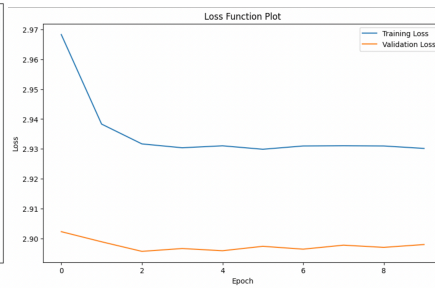
I split the dataset into training and validation dataset. I trained the model with different parameters, such as

1. Optimizer (adam, SGD, DG)
2. Loss functions (categorical\_cross\_entropy, sparse\_categorical\_cross\_entropy\_loss)
3. Epochs (5,10,15)
4. Hybrid layers ( Combination of different layers )
5. Different number of neurons in each layer (64,128,256)
6. Different learning rates (0.1,0.01,0.001)

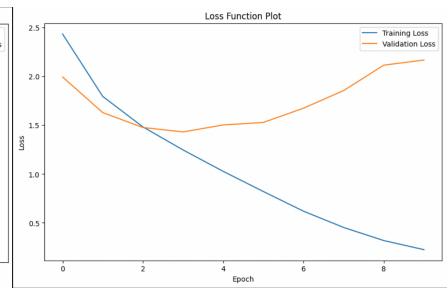
I plotted training and validation losses with different parameters, I got the following results



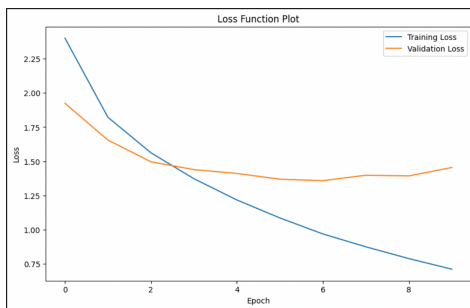
lr=0.1



lr=0.01



Adding Dropout layer



Removing few LSTM layers

Later on, I calculated the accuracy of the model with training dataset and validation dataset

After all combinations and fine tuning parameters, I got the **best** parameters here:

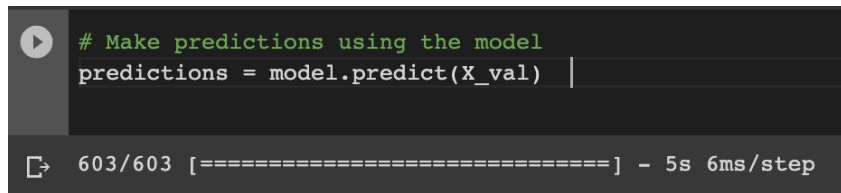
1. Adam Optimizer
2. masked\_sparse\_categorical\_crossentropy\_loss
3. 5 epochs
4. 2 CNN, 2 Bi-LSTM, 1 Dense, 2 MaxPooling layers
5. Learning rate = 0.001

```
Epoch 1/5
1206/1206 [=====] - 33s 23ms/step - loss: 2.3844 - masked_accuracy: 0.2756 - val_loss: 1.9315 - val_masked_accuracy: 0.3961
Epoch 2/5
1206/1206 [=====] - 27s 22ms/step - loss: 1.7614 - masked_accuracy: 0.4482 - val_loss: 1.6288 - val_masked_accuracy: 0.4907
Epoch 3/5
1206/1206 [=====] - 27s 22ms/step - loss: 1.4725 - masked_accuracy: 0.5352 - val_loss: 1.4951 - val_masked_accuracy: 0.5316
Epoch 4/5
1206/1206 [=====] - 27s 22ms/step - loss: 1.2530 - masked_accuracy: 0.6025 - val_loss: 1.4723 - val_masked_accuracy: 0.5490
Epoch 5/5
1206/1206 [=====] - 27s 22ms/step - loss: 1.0578 - masked_accuracy: 0.6629 - val_loss: 1.4861 - val_masked_accuracy: 0.5617
```

Training log



I also predicted the images:



```
# Make predictions using the model
predictions = model.predict(X_val)
```

603/603 [=====] - 5s 6ms/step

## Discussion and Comparison

In this section, I will discuss about the training v validation loss graphs ,

- **lr=0.1:** In this graph, you can see that training and validation loss have very unpredictable curve. Hence, it is not recommended to have this loss , it is best to have some curve which is very predicted for future testing data.
- **lr=0.01:** In this graph, gap between training and validation loss curve are too much, I want any future data(In this case validation loss) very close to training data.
- **Adding Dropout layer:** In this graph, As epochs increase, training loss curve decreases while validation loss curve increases, this only means that model is not able to predict the new data, while its modeling on training data, This phenomenon is called **Overfitting**.
- **Removing few LSTM layers:** In this graph, It is little similar to what happened in the graph of Dropout layer, Training and validation loss are going in different directions hence, the model is overfitting.

As you can see the above training log picture, I got 66% accuracy and both training and validation decreased for each epoch. Considering the facts that, I used techniques taught in class on real-life large dataset , I got reasonable accuracy.

Accuracy might increase if I had use other techniques like CTC loss as an output layer, edit distance for each an every character, using cross validation on the dataset.

## Conclusion

After all observations of above graphs, I can list out the possibilities of why model at par :

- I can clearly tell the model is overfitting, this means the model is too complex for the dataset.
- There are limitations for this model, I have used only the traditional methods. So , I can try out other parameters like CTC loss and edit distance etc.
- Using Vision Transformers, might lead to better computation results.

## References

<https://fki.tic.heia-fr.ch/databases/iam-handwriting-database.>

[https://en.wikipedia.org/wiki/Stochastic\\_gradient\\_descent](https://en.wikipedia.org/wiki/Stochastic_gradient_descent)

[https://en.wikipedia.org/wiki/Loss\\_function](https://en.wikipedia.org/wiki/Loss_function)

<https://vevesta.substack.com/p/here-is-what-you-need-to-know-about>

<https://towardsdatascience.com/build-a-handwritten-text-recognition-system-using-tensorflow-2326a3487cd5>