Handwritten Character Recognition using CNN

Sriaditya Venkatesh
Department of Electrical and Computer Engineering, Univerisity of Florida
Gainesville, Florida
svenkatesh1@ufl.edu

Ajay Sriram Raghavendran
Department of Electrical and Computer Engineering, University of Florida
Gainesville, Florida
a.raghavendran@ufl.edu

***Abstract*— This project's primary focus is set on the classification of handwritten characters. The characters included for classification are upper-case and lower-case letters starting from 'a/A' until 'h/H' and special characters '$' and '#'. A total of 6720 handwritten characters ranging across all the classes are included in training from which the model is validated with the help of cross validation techniques. The model is trained using convolutional neural networks (CNNs). CNN is a powerful technique that can compute classification to a high degree of accuracy. This report consists of the following components: introductory background of the given problem statement, implementation of the model, the experimental setup of the project, the results and conclusions derived from the project.**

Keywords— Convolutional Neural Network (CNN), Character Recognition, Deep Learning, Classification

I. Introduction

A. Overview

A considerable part of machine learning research has been and is being driven towards finding patterns in data and improving their classification capabilities. One such area where pattern recognition can be used is to classify a given dataset of alphanumeric characters. This project has its specific interest in classifying alphabetical characters from 'a' through 'h', both in uppercase and lowercase, and special characters '$' and '#', all written by hand. There are various approaches one can lean towards to achieve this task. There are techniques such as k-Nearest Neighbors (kNN) algorithm, Support Vector Machines (SVM), Neural Network algorithms, to name a few. We have chosen to work with the Neural Network approach, specifically, the Convolutional Neural Networks (CNN) because of its high-performance capabilities.

The data collection was carried out with each team involved in the project submitting 400 images and associated labels of handwritten data consisting of 40 images from each of the 10 classes as shown in Table I. The amount of training data used for the model is 6720 pairs of handwritten characters and their corresponding labels. The model is built subsequently and the results are validated using accuracy score and confusion matrix.

B. Literature Review

A convolutional neural network can have multiple layers namely [1]:

1. Convolutional layer
2. Non-Linearity layer
3. Pooling layer
4. Fully connected layer

TABLE I
DATA AND CLASS LABELS

Data	Label
a/A	0
b/B	1
b/C	2
d/D	3
e/E	4
f/F	5
g/G	6
h/H	7
$	8
#	9

An important assumption made by CNN is that the features representing the dataset are not spatially dependent. The first step in our algorithm is to preprocess the data. Subsequently, the CNN architecture is built. The final step is estimating probabilities of each data belonging to each class and the pair with the highest probability is chosen as the label for that data.

This rest of the paper is organized in the following manner -Section II, we discuss about the details of the model, its basic architecture, the flow of the model and number of trainable parameters. Section III talks about the experimental design which includes preprocessing of the data, resampling of the data, details about how the model is integrated, results of training which includes Accuracy scores, Confusion matrix and Validation accuracy. The conclusions drawn from the project are presented in Section IV.

II. Model Implementation

The model used for the classification of handwritten characters makes use of Convolutional Neural Networks. The Convolutional Neural network makes use of Image convolution as shown in the image below [2] –

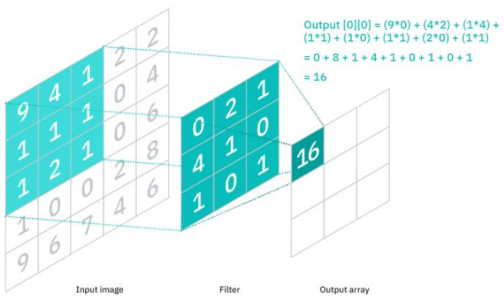

Fig 1: The Convolution Operation

The weighted window used for convolution is known as a *kernel.* The size of the kernel determines the features to be highlighted in the output. The output of the convolution is referred to as *feature map.* The operation of convolution is followed by Max-Pooling layer. The Max-Pool layer has the task of down-sampling the convolved output image. This layer calculates the maximum value for patches of the feature map. The below figure delineates the max-pool operation [3] –

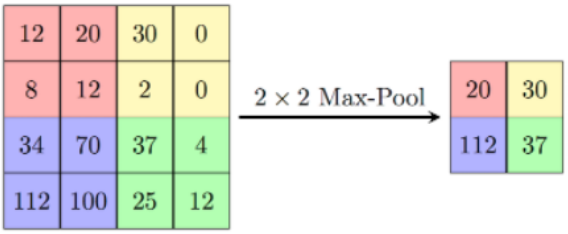

Fig 2: Max-Pooling

Once the max-pooling operation is done, the output is flattened and sent to a fully-connected neural network, also known as the *Dense* Network. This layer performs the task of classification by considering the features extracted by the previous CNN Layers. The figure below gives a brief of this design flow [4] –

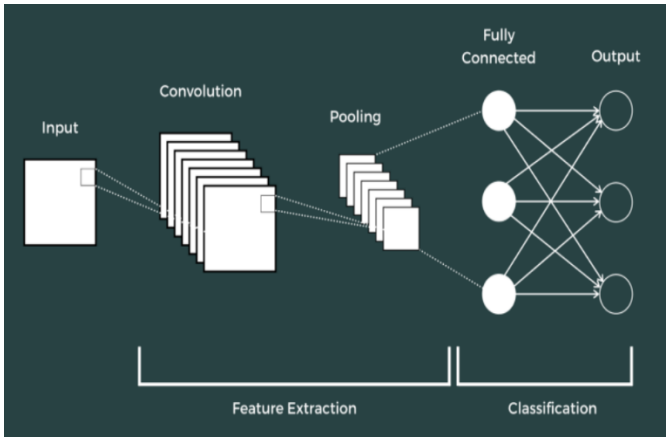

Fig 3: CNN Model Flow

In the current implementation, we have the resampled input entering the first CNN structure. The size of the input image data is 6720x50x50. 6720 input samples each resampled to 50x50 size. The first CNN structure consists of the 2D Convolution layer with 32 output channels and kernel size 5x5. This is followed by one more 32 output 2D convolution layer with kernel – 5x5. Both these utilize the RELU activation. This is followed by the Max-Pool layer. The output of this first structure is succeeded by two 64-output channel 2D convolution layer with kernel size – 3x3. There is a Max-Pool layer once again for down-sampling this output. The need to use Max-Pooling layer after every Conv2D layer comes from the fact that down-sampling of the feature maps to condensed maps gives rise to two advantages – it reduces number of parameters, thus reduces computational cost and controls the overfitting of the network [5]. The basic architecture flow for one image is shown in Figure 3.

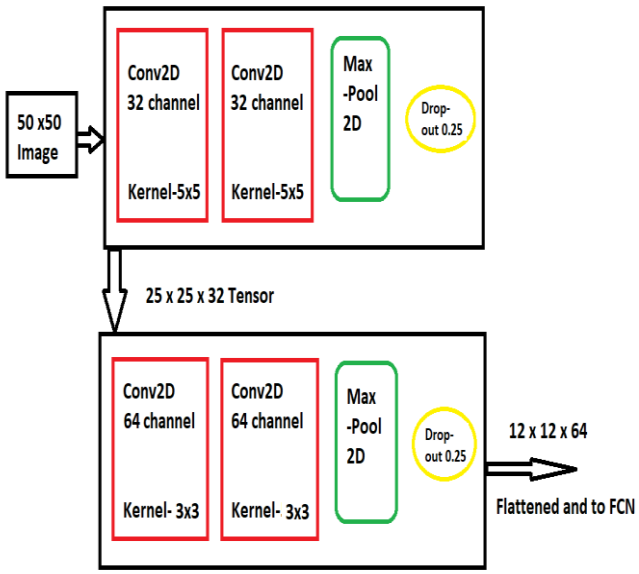

Fig 3: Basic Architecture

The output of the CNN stack after flattening will be an array of size 6720 x 9216. This is followed by a 256 – unit Dense layer and lastly a 10 – unit dense layer for one-hot encoded prediction.

The aforementioned model is built using the Keras Sequential library which allows stacking of the layers one below the other. The use of Dropout is made after each set of layers. The current used Dropout Rate is 0.25. Dropout layer randomly equates input units to zero with a frequency of "rate" at each step during training time. This helps prevent overfitting. Inputs which are non-zero are scaled up by 1/ (1 – "rate") such that the sum over all inputs is unchanged. [6]

The below figure shows the model summary –

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 50, 50, 32)        832

conv2d_9 (Conv2D)            (None, 50, 50, 32)        25632

max_pooling2d_4 (MaxPooling  (None, 25, 25, 32)        0
2D)

dropout_6 (Dropout)          (None, 25, 25, 32)        0

conv2d_10 (Conv2D)           (None, 25, 25, 64)        18496

conv2d_11 (Conv2D)           (None, 25, 25, 64)        36928

max_pooling2d_5 (MaxPooling  (None, 12, 12, 64)        0
2D)

dropout_7 (Dropout)          (None, 12, 12, 64)        0

flatten_2 (Flatten)          (None, 9216)              0

dense_4 (Dense)              (None, 256)               2359552

dropout_8 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 10)                2570

=================================================================
Total params: 2,444,010
Trainable params: 2,444,010
Non-trainable params: 0
_________________________________________________________________
```

Fig 4: Keras Model Summary

III. Experimental Design

A. Data Preparation and Preprocessing

The data files are in the form of NumPy Files. The first task is to load the training data and the corresponding labels into NumPy arrays. The NumPy library offers the load functionality to achieve this. The files provided for training have names "data_train.npy" and "labels_train.npy".

The pre-processing of the data includes Normalization. The Normalization is achieved by Min-Max Scaling. The formula for normalization used for a vector X is as follows –

$$X_{norm} = \frac{(X - \min(X))}{\max(X) - \min(X)} \quad (1)$$

After the pre-processing of the data, the 90000x6720 vector is converted to 6720 images of size 300x300. It is then resampled to 50x50. The image resolution at this point reduces but the key features are not obliterated. The resolution of the images used is sufficient for the CNN to extract distinguishing features.

One important pre-processing technique is the one-hot encoding of the output labels. This makes the labels expressive of the classes they represent.

B. Training

- The 80-20 train test split data is passed to the compiled model whose implementation has been previously explained. The train data is data augmented which includes rotated, zoomed, vertically and horizontally shifted versions of the images along with the main dataset
- The model compilation includes setting the key parameters for the model like Optimizer, Loss and metric of evaluation.
- In the current experiment, the model is compiled with RMSProp Optimizer. This optimizer maintains the moving average of the square of gradients and divides the gradient by the root of this average [7].
- The loss function is the Categorical Cross-Entropy. This loss function aids in problems involving multi-class classification. The only activation function to be used with cross-entropy is the *SoftMax* activation function which forms the end activation of the model.
- The evaluation metric used for every epoch is the Accuracy measure. There are a total of 100 epochs and a batch-size of 64. The model is fit with an adaptive learning rate updation rule. Keras provides with a rule called "*ReducedLROnPlateau*" which allows to reduce the learning rate when the metric has stopped improving [8]. The minimum learning rate used is 0.00005.
- Validation is performed on the twenty percent test data at each epoch.
- At the end of the training, the accuracy score of the last cross-validation split test data is printed. A snippet of the training output is shown in Figure 5. Both training loss and validation loss can be observed in the snippet shown below.

```
Epoch 45/100
84/84 - 112s - loss: 0.2652 - accuracy: 0.9152 - val_loss: 0.2440 - val_accuracy: 0.9360 - lr: 2.0000e-04 - 112s/epoch - 1s/step
Epoch 46/100
84/84 - 113s - loss: 0.2783 - accuracy: 0.9085 - val_loss: 0.2190 - val_accuracy: 0.9353 - lr: 2.0000e-04 - 113s/epoch - 1s/step
Epoch 47/100
84/84 - 113s - loss: 0.2530 - accuracy: 0.9213 - val_loss: 0.2176 - val_accuracy: 0.9397 - lr: 2.0000e-04 - 113s/epoch - 1s/step
Epoch 48/100
84/84 - 113s - loss: 0.2742 - accuracy: 0.9111 - val_loss: 0.2133 - val_accuracy: 0.9360 - lr: 2.0000e-04 - 113s/epoch - 1s/step
Epoch 49/100
```

Fig 5: Training output Snippet

C. Results

The final validation accuracy obtained was **95.01** percent. This can be seen in the figure below. The final validation accuracy is the accuracy of hundredth epoch's cross-validation test split.

```
84/84 - 117s - loss: 0.2231 - accuracy: 0.9299 - val_loss: 0.1943 - val_accuracy: 0.9516
Epoch 100/100
84/84 - 117s - loss: 0.2256 - accuracy: 0.9323 - val_loss: 0.1968 - val_accuracy: 0.9501
Final Test Accuracy: 95.01%
```

Fig 6: Final Validation Accuracy

The below two figures show the training and validation accuracy and training and validation loss against the number of epochs.

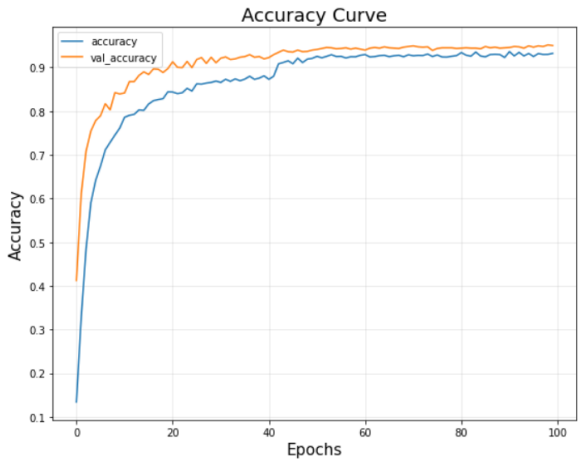

Fig 7: Accuracy Curve

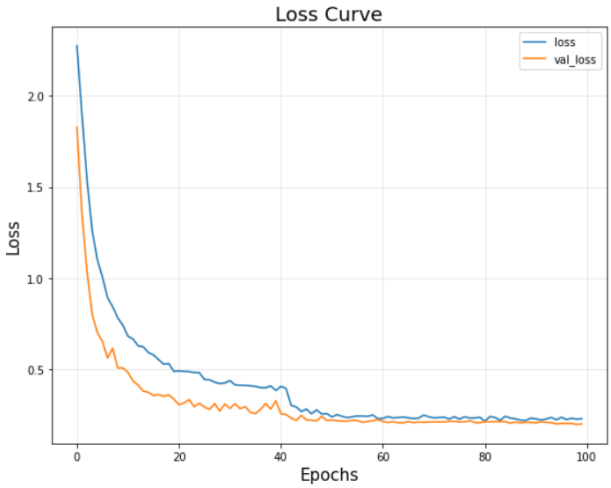

Fig 8: Loss Curve

For the predicted labels and the actual targets, the Confusion matrix was calculated and is shown in Figure 9.

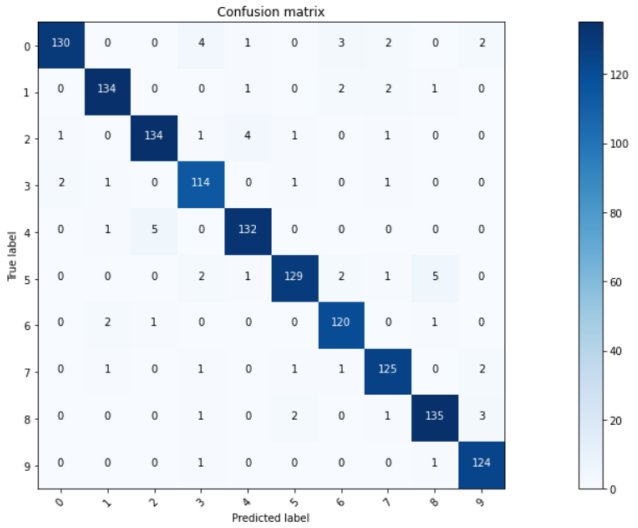

Fig 9: Confusion matrix for the validation test set.

IV. Conclusions

In this paper, we proposed a method to classify handwritten characters using Convolutional neural networks. As CNN achieves high accuracies by preserving the spatial information, it was a very suitable candidate for the task. Along with the proper parameters for the model, a proper optimizer and apt learning rate updation rules were used to together achieve an accuracy of **95.01** percent.

References

[1] S. Albawi, T. A. Mohammed and S. Al-Zawi, "*Understanding of a convolutional neural network*," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[2] https://www.ibm.com/cloud/learn/convolutional-neural-networks

[3] https://paperswithcode.com/method/max-pooling#:~:text=Max%20Pooling%20is%20a%20pooling,used%20after%20a%20convolutional%20layer.

[4] https://www.theclickreader.com/introduction-to-convolutional-neural-networks/

[5] Hossein Gholamalinezhad1 , Hossein Khosravi, "*Pooling Methods in Deep Neural Networks, a Review*"

[6] https://keras.io/api/layers/regularization_layers/dropout/

[7] https://keras.io/api/optimizers/rmsprop/

[8] https://keras.io/api/callbacks/reduce_lr_on_plateau/