

ML MINI PROJECT REPORT

Handwriting Recognition using Convolutional Neural Network



Team Members

Subhiksha S 185001172

Sukeerthi M 185001175

Vaishnavi M R 185001187

Varshinee M M 185001190

PROBLEM STATEMENT

ABSTRACT

One of the computer related problems that are being sought and researched is how can an image be recognized and classified by the computer just like the humans do. One that can be recognised from an image is the handwriting which can help in handwritten form processing and in other fields. The goal of this project will be to create a model that will be able to recognize the handwritten characters using Convolutional Neural Networks.

OBJECTIVE:

The objective of this task is to identify the handwritten alphabets correctly. The basic idea is to split the dataset into training and testing dataset and then it will get trained with the training set. The output of the testing dataset is used to give feedback to the model to accordingly increase the accuracy of the model again by training it and validating the accuracies and losses.

MOTIVATION:

In a world that is computationally exceeding the limits of human outreach, there is an immense necessity for the digitalisation and documentation of information in a raw transcript format written by various people. Despite the exponential growth of technological writing tools, people still prefer to take handwritten notes. Each of us have different handwriting and sometimes it becomes difficult to understand them. These variations in handwriting pose quite a lot of problems. So the importance of detecting and identifying not only computer fonts but also the handwritten alphabets correctly is evident from the growing digitalisation all over the world. In this machine learning project, we will recognize handwritten characters, i.e, English alphabets from A-Z. We are going to achieve this by modeling a neural network that will have to be trained over a dataset containing images of alphabets.

DATA

The data set that we have chosen contains 26 folders (A-Z) containing handwritten images in **size 28 * 28 pixels**, where each image is a gray scale image. The data is taken from the NIST database

(<https://www.nist.gov/srd/nist-special-database-19>)

and MNIST large dataset and few other sources, downloaded from

<https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format>

The overall collection of data was split into train and test data in the **ratio 80 : 20**.

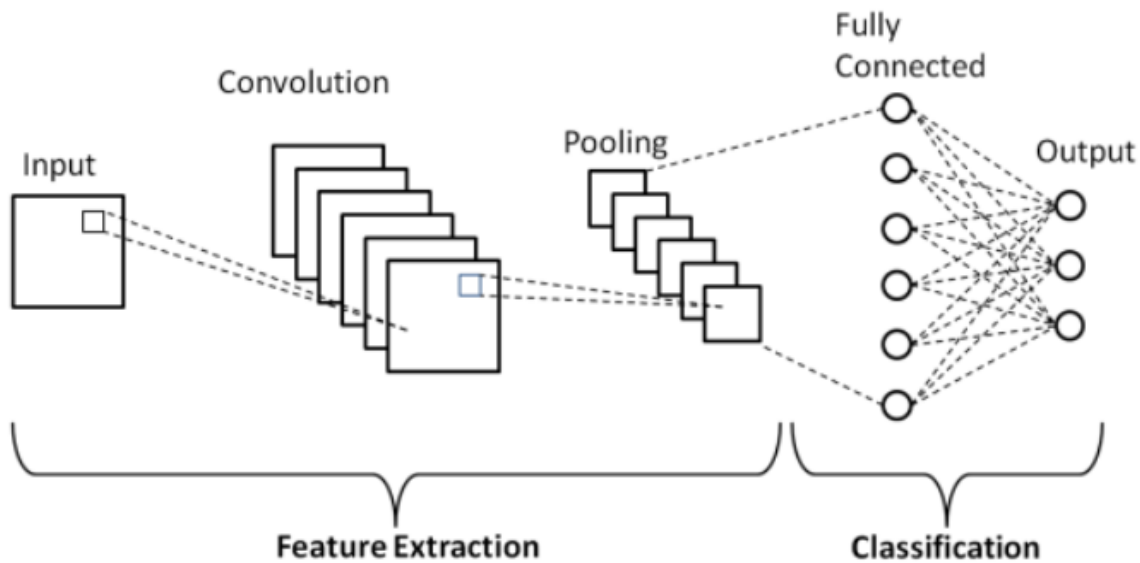
DATA FILES:

1. **A_Z Handwritten Data.csv** - Labelled dataset of 372450 images each of 28*28 pixels in a 1d array form.
2. **Training data** : 297960
3. **Testing data** : 74490

Here, 20% percent of the has been assigned for testing purposes and the rest 80% for training purposes.

METHODOLOGY

The deep neural networks have the state-of-art performance in solving many classification and recognition problems. In this project we have used Convolution Neural Network for handwriting recognition. A CNN can take an input image, assign learnable weights and biases to various aspects of the image and be able to differentiate from one another. CNN are very effective in perceiving the structure of handwritten characters or words in a way that helps in automatic extraction of distinct features. This makes them more suitable for handwriting recognition. In addition to this it also helps in achieving better accuracy with low computational cost and testing complexity. It has the ability to automatically detect the important features of an object without any human supervision or intervention. A CNN integrates the feature extraction and classification steps and requires minimal preprocessing and feature extraction efforts.



ARCHITECTURE

A basic convolutional neural network comprises three components, namely, the convolutional layer, the pooling layer and the output layer. The pooling layer is optional sometimes. For handwriting recognition we use CNN with three convolutional layers. It consists of the input layer, multiple hidden layers (repetitions of convolutional, normalization, pooling) and a fully connected and an output layer. In a CNN model, the input image is considered as a collection of small sub-regions called the “receptive fields”. A mathematical operation of the convolution is applied on the input layer, which emulates the response to the next layer. The response is basically a visual stimulus.

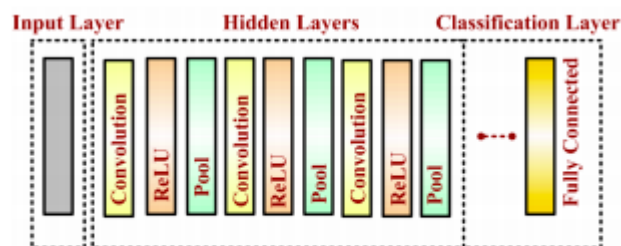


Figure 1. Typical convolutional neural network architecture.

INPUT LAYER : The input is stored and loaded here. It also describes the height, width and number of channels of the input image (RGB information).

HIDDEN LAYER: They perform a feature extraction process where a series of convolution, pooling, and activation functions are used. The distinct features of the handwritten character are identified here. This layer is the backbone of CNN.

CONVOLUTION LAYER: It is the first layer placed above the input image. It is used for extracting features of an image. The main contributors of the convolutional layer are receptive field, stride, dilation and padding.

POOLING LAYER: It is placed between two convolutional layers to reduce input dimensionality which in turn reduces the computational complexity. Pooling allows the selected values to be passed to the next layer while leaving the unnecessary values behind. The pooling layer also helps in feature selection and in controlling overfitting. The common types of pooling operations are max-pooling and avg-pooling (where max and avg represent maxima and average, respectively).

ACTIVATION LAYER: It contains activation functions to introduce non linearity in the system. The sigmoid function, rectified linear unit (ReLU) and Softmax are some activation functions that can be used. The activation function used in the present work is the non-linear rectified linear unit (ReLU) function, which has output 0 for input less than 0 and raw output otherwise.

CLASSIFICATION LAYER: It is the last layer in the CNN architecture. It is a fully connected feed forward network, mainly adopted as a classifier, where the neurons in the fully connected layers are connected to all neurons of the previous layer. This layer calculates predicted classes by identifying the input image, which is done by combining all the features learned by previous layers. In this project, the classification layer uses the 'softmax' activation function for classifying the gestures of the input image from the previous layers into various classes based on training data.

EVALUATION METRICS:

1. CLASSIFICATION ACCURACY:

Ratio of number of correct predictions to the total number of input samples

$$Accuracy = \frac{\text{Number of Correct predictions}}{\text{Total number of predictions made}}$$

Training Accuracy - 0.9858370423316956;

Testing Accuracy - 0.9858370423316956

2. LOGARITHMIC LOSS

Works by penalising the false classifications. Suppose there are N samples belonging to M classes , then the Log Loss is calculated as below:

$$LogarithmicLoss = \frac{-1}{N} \sum_{i=1}^N \sum_{j=1}^M y_{ij} * \log(p_{ij})$$

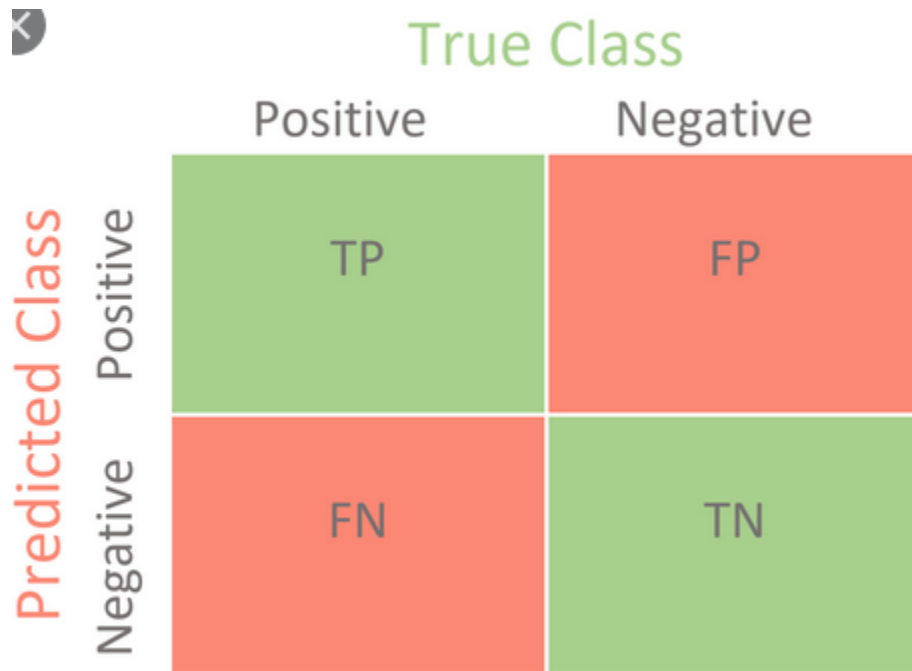
Y_{ij} indicates whether sample i belongs to class j or not

p_{ij} , indicates the probability of sample i belonging to class j

We have 297960 training data and our log loss is - 0.078903928399086

3. CONFUSION MATRIX

The output is a matrix that describes the complete performance of the model



		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

The 4 important terms:

- **True Positives** : The cases in which we predicted YES and the actual output was also YES.

TP: 73339.0

- **True Negatives** : The cases in which we predicted NO and the actual output was NO.

TN: 1861362.0

- **False Positives** : The cases in which we predicted YES and the actual output was NO.

FP: 888.0

- **False Negatives** : The cases in which we predicted NO and the actual output was YES.

FN: 1151.0

4. AREA UNDER CURVE

a. True Positive Rate (SENSITIVITY):

TPR corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

$$TruePositiveRate = \frac{TruePositive}{FalseNegative + TruePositive}$$

$$TPR = 73339/1151+73339= 0.984548261511$$

b. True Negative Rate(Specificity)

TNR corresponds to the proportion of negative data points that are correctly considered as negative, with respect to all negative data points.

$$TrueNegativeRate = \frac{TrueNegative}{TrueNegative + FalsePositive}$$

$$TNR = 1861362/1861362+888= 0.999523157470$$

c. False positive Rate

False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$FalsePositiveRate = \frac{FalsePositive}{TrueNegative + FalsePositive}$$

FPR: $888/1861362+888= 0.00047684253$

5. F1 SCORE

F1 score is used to measure a test's accuracy. It is the harmonic mean between precision and recall. The range for F1 score is [0,1]. It tells us how precise our classifier is and how robust it is.

High precision but lower recall, gives you an extremely accurate, but it then misses a large number of instances that are difficult to classify. The greater the F1 Score, the better is the performance of our model.

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

- a. **Precision:** It is the number of correct positive results divided by the number of positive results predicted by the classifier.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$\text{Precision} = 73339.0 / 73339.0 + 888.0 = 0.988036698236$$

- b. **Recall:** It is the number of correct positive results divided by the number of ***all*** relevant samples

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{Recall} = 0.9845482707023621$$

$$\text{F1 Score} = 0.98628939990581646338331255721355$$

PROPOSED SOLUTION

Project Prerequisites

1. Python (3.8.5 used)
2. IDE - Jupyter Notebook with Anaconda Prompt

Required Frameworks

1. Numpy
2. cv2 (openCV)
3. Keras
4. Tensorflow (Keras runs on top of TensorFlow)
5. Matplotlib
6. Pandas

Steps to be followed :-

IMPORT LIBRARIES AND LOAD DATASET:

Import the necessary libraries according to the frameworks mentioned above.

The data pertaining to images of handwritten alphabets is depicted in the form of pixels. The dataset consists of 372450 images, across 784 columns of pixel data which can be reshaped into 28x28 pixels for image feature extraction and representation.

Read the dataset file, by using the dataframe functions provided by pandas and visualize a brief notion by extracting the first few image pixel details.

SPLITTING OF TRAINING AND TESTING DATA

Using scikit-learn's inbuilt function "train_test_split", we can split the data arrays into two subsets: for training data and testing data instead of allocating them manually.

DATA RESHAPING FOR IMAGE FEATURE EXTRACTION

Initially, the image has been represented as a pixel data spanning across 784 columns which should now be reshaped as 28x28 resolution of pixels.

CREATING A DICTIONARY FOR ALPHABET LABELS

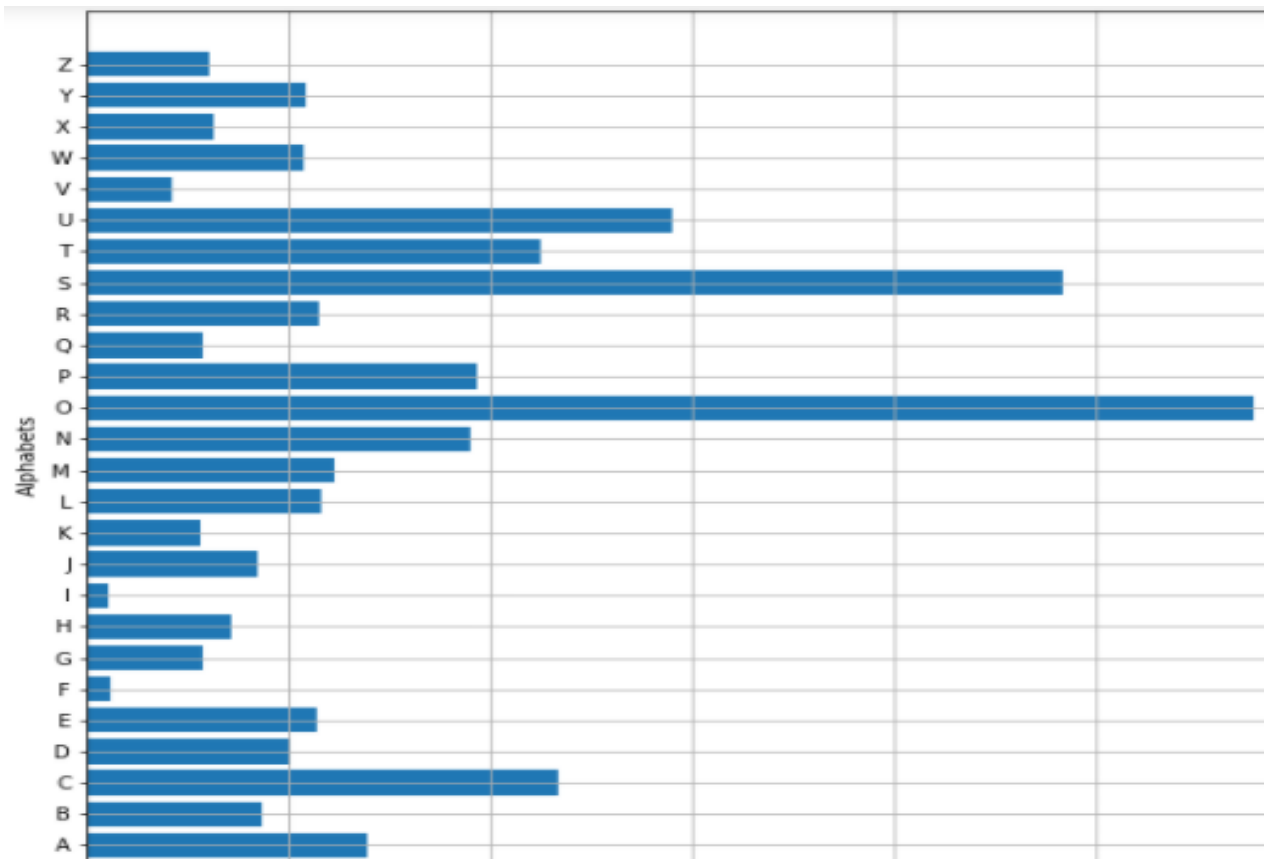
The labels for classifications are present in the form of floating point numbers. To classify them as alphabets for better representation of results, we map the type casted integers to their corresponding alphabets as key value pairs.

A	B	C	D	E	F	G	H	I	J	K	L	M
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
13	14	15	16	17	18	19	20	21	22	23	24	25

PLOTTING OF FREQUENCY HORIZONTAL BAR GRAPH

To have more insight into the dataset, we describe the distribution of alphabets and represent a count for each alphabet label.



MATPLOTLIB

It is a 2D plotting library for creating graphs and bars to visualize and represent data. Most of the Matplotlib utilities lie under the pyplot submodule, and are usually imported under the plt alias. The subplot function can be used to plot the above frequency bar graph which inputs parameters such as the number of rows and columns and the graph size. We can customize the x axis and y axis parameters as frequencies and alphabets labels respectively.

PLOTTING OF 28x28 RESOLUTION IMAGES AFTER RESHAPING

We need to obtain the images from the values of pixels using thresholding.

Thresholding is a technique in openCV which is the assignment of pixel values in relation to the threshold value provided. Each pixel value is compared with the threshold value, if the pixel is smaller then it is set to 0 else it is set to a maximum value of 255.

Here the function used is `cv2.thresh_binary` where-in - if the threshold intensity is greater than the set threshold, the pixel is set to 1 else set to 0.

After flattening the images to reduce nested for loop iterations, we apply the threshold function with parameters - `cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)`.

source: is the input image array.

thresholdValue: the value of Threshold below and above which pixel values will change accordingly.

maxVal: Maximum value that can be assigned to a pixel.

thresholdingTechnique: The type of thresholding to be applied.

We can thus plot the images in subplots of size 28X28 pixels for pictorial representation.

ADD THE BIAS INPUT

To fit into the model, add the corresponding bias input vector thereby increasing a dimension. The shape() function offered by the numpy library. NumPy arrays have an attribute called shape that returns a tuple with each index having the number of elements pertaining to each dimension in this scenario.

The new shape of training data would be- (297960, 28, 28, 1)

The new shape of testing data would be - (74490, 28, 28, 1)

DEFINING THE CATEGORICAL LABELS

Since the CNN model takes input of labels & generates the output as a vector of probabilities, we need to convert the single float values to categorical values. The to_categorical function can be used to convert a vector or single column matrix of class labels into a matrix with n columns for each category.

CREATE A SEQUENTIAL MODEL USING CNN LAYERS AND ACTIVATION FUNCTIONS

The sequential API allows us to create models layer-by-layer, a plain stack of layers where each layer has exactly one input tensor and one output tensor.

The add() method allows to create a sequential model incrementally.

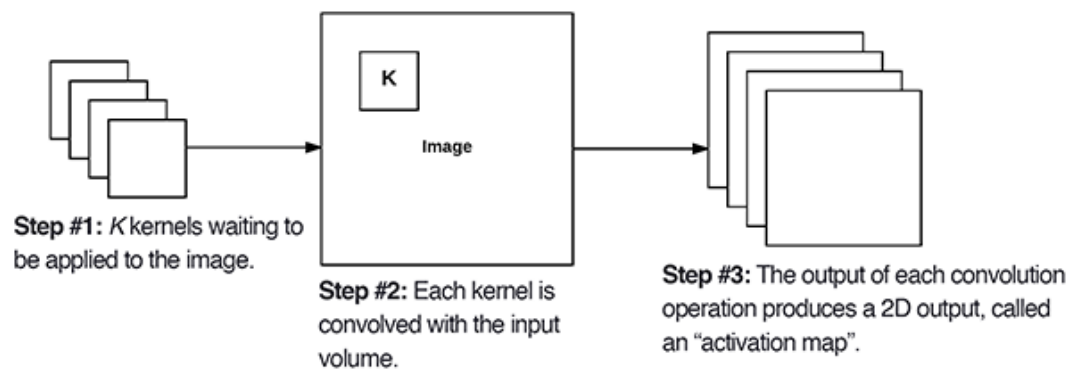
Here we are adding 3 convolutional layers along with a max pooling layer, 1 layer to flatten the database, 2 hidden layers of fully connected neural network, and one output layer called the softmax layer.

Layer 1 of conv2D - explicitly specified the input shape as (28,28,1)

Layer2 and layer3 of convolutional layer takes input from the previous layer

Conv2D parameters:

Filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).



kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

Strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value $\neq 1$ is incompatible with specifying any `dilation_rate` value $\neq 1$.

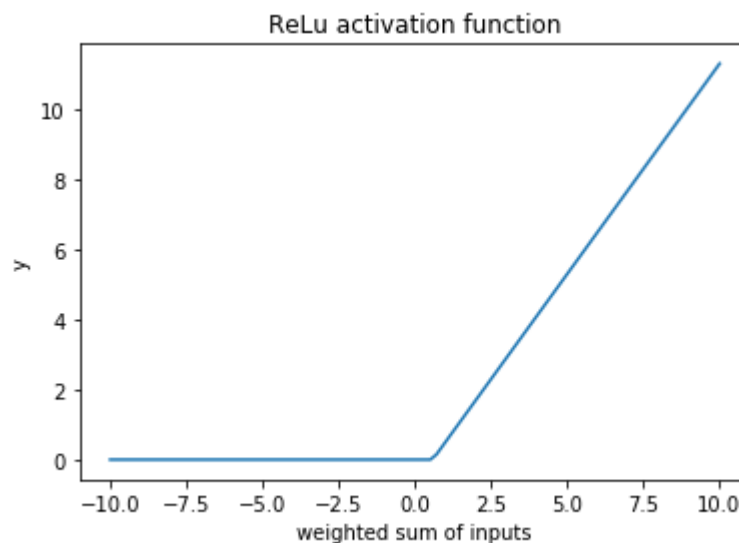
Its default value is always set to (1, 1) which means that the given Conv2D filter is applied to the current location of the input volume and the given filter takes a 1-pixel step to the right and again the filter is applied to the input volume and it is performed until we reach the far right border of the volume in which we are moving our filter.

Padding: one of "valid" or "same" (case-insensitive). "valid" means no padding. "same" results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.

Activation: Activation function to use. In our model we have used ReLU (Rectified Linear Units) activation function which is defined as:-

$$f(x) = \max(0, a) = \max(0, \sum_{i=1}^n w_i x_i + b)$$

If the input is a positive number the function returns the number itself and if the input is a negative number then the function returns 0.



Because of its simplicity and easy computation, *ReLU* is used as a standard activation function in CNN and it does not saturate for the positive value of the weighted sum of inputs.

After each convolution layer there is a maxpool layer to reduce the spatial dimensions of the output of the conv layer

Max pool:

Max pooling 2D: downsizes the input representation by taking the maximum value over the window defined by the pool_size for each dimension along the feature axis. The window is shifted by strides in each dimension

output_shape = (input_shape - pool_size + 1) / strides) while using valid option

output_shape = input_shape/strides

flatten() will reshape the array to a single vector to be given as input to the NN

Dense layer:

Dense implements the operation: $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$ where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer

argument : unit: Positive integer, dimensionality of the output space.

input: input_dimensions

output_dim: the units

COMPILE THE MODEL

Before training the model we need to compile it and define the optimizers, loss function and metrics for prediction. The model is compiled using `compile()` method. Here the optimizer, loss, and metrics are the necessary arguments.

Optimizers are important as we use that to adjust input weights. The input weights are optimized by comparing the prediction and the loss function. Among the various available, we use Adam. The learning rate has to be specified for the optimizers which is set to 0.001. **Adam** combines the best properties of the AdaGrad and RMSProp algorithms to provide an **optimization** algorithm that can handle sparse gradients on noisy problems.

Loss function is used to find error or deviation in the learning process. There are various loss functions available in which `categorical_crossentropy` has been used here. Categorical cross entropy is a loss function that is used in multi-class classification tasks. These are tasks where an example can only belong to one out of many possible categories, and the model must decide which one.

Metrics specifies the evaluation criteria for the model to check for its performance efficiency. There are various available metrics in Keras such as accuracy, precision, recall, TrueNegatives, TruePositives, FalsePositives, FalseNegatives that have been mentioned in the training process.

FIT THE MODEL TO FEED INTO THE NEURAL NETWORK FOR TRAINING

Models are trained by NumPy arrays using `fit()`. The main purpose of this fit function is used to evaluate your model on training. This can be also used for graphing model performance. It has the following syntax –

```
model.fit(X, y, epochs = , batch_size = )
```

Here, X, y – It is a tuple to evaluate your data.

epochs – no of times the model is needed to be evaluated during training.

batch_size – training instances.

The fit() method adjusts weights according to data values so that better accuracy can be achieved. Metric values are displayed during fit() and logged to the History object returned by fit().

OBTAIN THE MODEL SUMMARY DESCRIPTION

The model.summary() function outlines the different layers defined in the model. It makes a tabular description of :

- The layers and their order in the model.
- The output shape of each layer.
- The number of parameters (weights) in each layer.
- The total number of parameters (weights) in the model.

In the above model we have used Conv2D, MaxPool2D and Dense Layers.

EXTRACT THE PERFORMANCE METRIC VALUES FROM THE LAST EPOCH

The history callback provides the capability to register callbacks when training a model. It records training metrics for each epoch. This includes the loss and the accuracy for training as well as the val_loss and val_accuracy for the validation dataset, if it is set. Metrics are stored in a dictionary in the history member of the object returned. To list the metrics collected in a history object, we can use history.history.keys(). We can print the values of whichever epoch we want. Using history.history, the metrics and the corresponding values can be accessed for all the epochs and can be printed.

For example- to access the accuracy metric over all epochs, we use:

`“History.history[‘accuracy’]”`

PRINT THE CONFUSION MATRIX

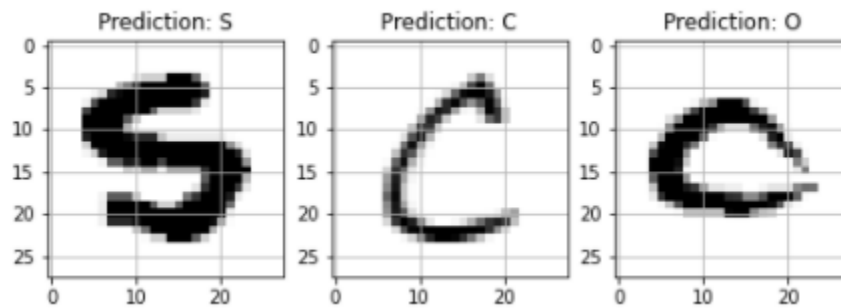
After obtaining the final epoch’s results for TruePositives, TrueNegatives, FalsePositives and FalseNegatives, we append them into a list named `Confusion_matrix` and reshape it as a 2x2 matrix that is represented in the following format.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

```
CONFUSION MATRIX
[[7.333900e+04  1.861362e+06]
 [8.880000e+02  1.151000e+03]]
```

DISPLAY FEW PREDICTIONS CORRESPONDING TO TESTING SET IMAGES

We first create subplots of the shape using the `plt.subplots()` function. This function is used to create a figure and a set of subplots. It returns the layout of the figure(`fig`) and axes objects or array of axes objects(`axes`). Then we flatten the array using `axes.flatten()` method to iterate over them easily. We then enumerate the array and reshape the image as 28X28 using `np.reshape()` function. Use `np.argmax()` function which returns the index of the class with the highest predicted probability. We then predict the character from the word dictionary and display it.



LOADING EXTERNAL IMAGE FOR PREDICTION

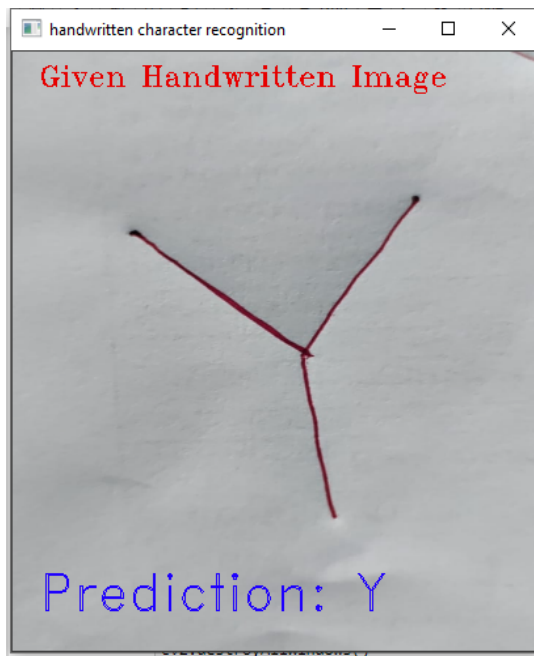
It is also possible to do prediction on external images. The function `cv2.imread()` loads an image from the specified path. Then a copy of the image is made using the `copy()` function. Only this copy is fed to the model. The image that is read is then converted from BGR representation to RGB as OpenCV reads in BGR format using the function `cv2.cvtColor()` which converts image from one color space to another. The image is also resized to the preferred dimensions for display.

PROCESSING THE IMAGE

The copied image is blurred to reduce noise. It is then converted from BGR to Grayscale Using `cv2.cvtColor()` function and `cv2.COLOR_BGR2GRAY` parameter. Then thresholding is applied to keep the image smooth so that it wouldn't lead to any wrong predictions. The image is then resized to the dimensions that is accepted by the model using `cv2.resize()` and it is then reshaped using `np.reshape()` so that it can be fed as input to the model.

PREDICTING THE CHARACTER

After the image is processed, we are going to make a prediction with it. `np.argmax()` function returns an index of the max value along the axis. Here it is used to return the index of the class with the highest predicted probability. Once the index is returned we can know the exact character through the word dictionary that we created earlier. The character is displayed on the frame by using `cv2.putText()` function which is used to draw a text string on an image. The frame is displayed until the ESC key is pressed. This is done using the `waitKey()` function.



RESULTS:

<u>METRICS</u>	<u>VALUES</u>
Training Accuracy	0.9858
Training Loss	0.0789
Testing Accuracy	0.9858
Testing Loss	0.0789
Precision	0.9880
Recall	0.9845
True Positives	73339.0
True Negatives	1861362.0
False Positives	888.0
False Negatives	1151.0
TPR	0.9845
TNR	0.9995
FPR	0.0004
F1 Score	0.9862

CONCLUSION:

For the Handwriting Recognition problem statement, we have detected the alphabets using Conventional Neural Network. An accuracy of 98.58% was obtained using ADAM optimizer for the MNIST database. By increasing the number of epochs, the accuracy percentage improved through more iterations of training. A frequency distribution bar graph has been plotted to visualize the dataset and corresponding labels have been depicted in the form of a dictionary. A sequential model has been developed for extracting features and passed through the convolutional, maxpool and dense layers to form a fully connected neural network that is capable of solving our multi-classification problem to the corresponding matching alphabet labels. The performance metrics were extracted after the model was trained across multiple epochs, and a confusion matrix was created using the values of these metrics. In order to extend the limits of the algorithm rather than confining to the dataset samples alone, a real-life handwritten image and a cursive-style alphabet image was taken and given as input to the model. After some refinements to the image to make it compatible with the pre-defined functions, an image pop-up is created and the prediction is shown to the user. Thus, we have implemented a convolutional neural network model for classifying the data-set of handwritten character images using TensorFlow and other machine learning libraries with optimal accuracy and efficient performance through suitable parameters.