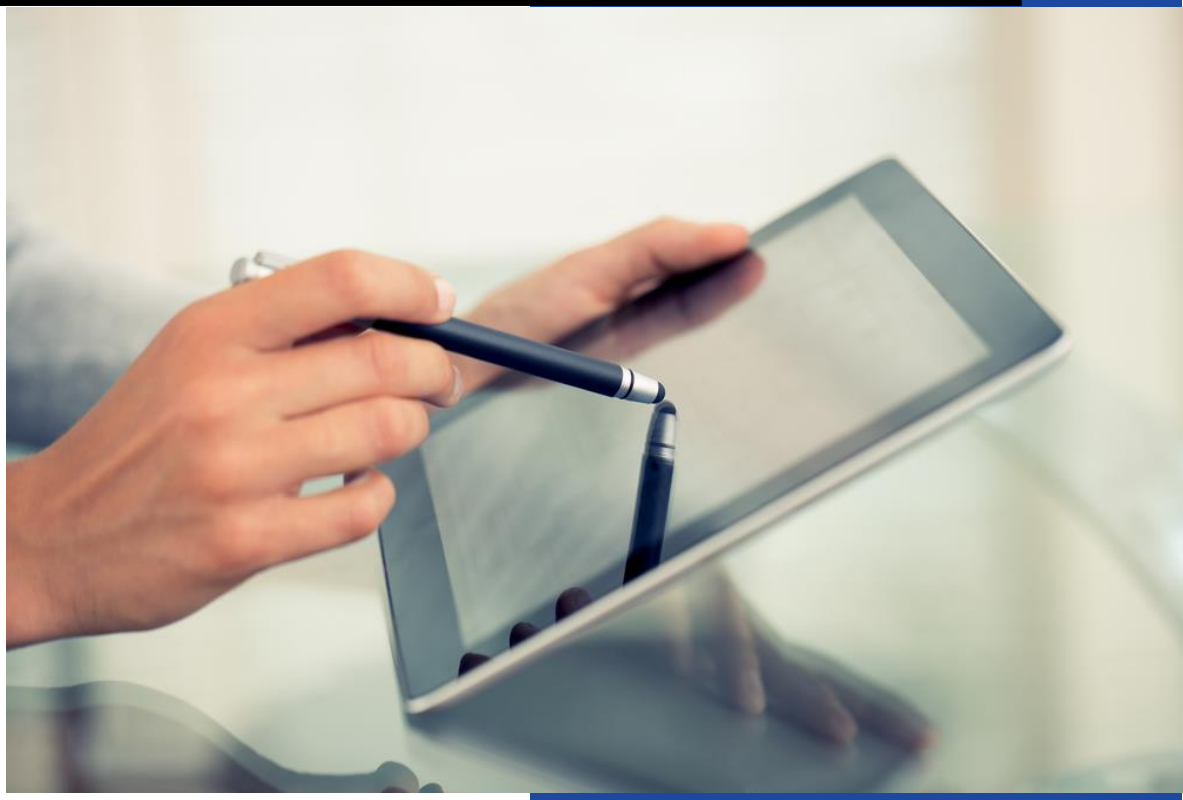
BUSINESS ANALYTICS
Master of Science

# HANDWRITING RECOGNITION




**Athens University of Economics & Business**

*Team Partners:*

- Evangelia Arseniou (p2822026)
- Paraskevi Barmpa (p2822013)
- Sotiria Ligkou (p2822023)

*Course:* Machine Learning & Content Analytics

*Academic Year:* 2021-2022

*Professor:* Haris Papageorgiou (xaris@ilsp.gr)

*Assistant responsible for this assignment:*

George Perakis (gperakis@aueb.gr)

EBS Solutions

# Contents

EBS Solutions

# Abbreviation List

| | |
|---|---|
| ML: | Machine Learning |
| AI: | Artificial Intelligence |
| OCR: | Optical Character Recognition |
| HTR: | Handwritten Text Recognition |
| FNN: | Feedforward Neural Network |
| CNN: | Convolutional Neural Network |
| CRNN: | Convolutional Recurrent Neural Network |
| VGG: | Visual Geometry Group |
| ResNet: | Residual Network |
| NA: | Not Available |
| ANN: | Artificial Neural Network |
| SNN: | Simulated Neural Network |
| RMSprop: | Root Mean Square Prop |
| MAE: | Mean Absolute Error |
| ReLU: | Rectified Linear Unit |
| LSTM: | Lost Short-Term Memory |
| CTC: | Connectionist Temporal Classification |
| NN: | Neural Network |

EBS Solutions

# List of Figures

## List of Tables



## List of Plots

# 1. Introduction

Nowadays, more and more companies have to keep pace with exploding process complexity and accelerating innovations. As the world becomes more digitized, the business world tries to enable some of their recurring tasks to run with less manual effort. There is a growing demand for business process automation more than ever due to the fact that this could help in cost minimization and efficiency increase. At this point, it is crucial to mention the important role of Machine Learning (ML). Machine Learning is a subset of Artificial Intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. In ML, there are different algorithms (e.g., neural networks) that help to solve such kind of problems.

A very interesting topic that many companies have to deal with is the Optical Character Recognition (OCR) which is a technology that converts any kind of images containing written or printed text into a machine-readable format. Whether it is automatic text extraction, document conversion to PDF or translation with a prone camera, character recognition apps stand behind it. But what about Handwritten Text Recognition (HTR)? Here, a whole spectrum of complexity is added. While the problem of OCR for machine text has effectively been solved, in most applications, for a long-time human handwriting displays a near-endless range of fonts and styles. Correctly recognizing these is way beyond classical pattern matching algorithms, which is why handwritten text recognition remains at the cutting edge of science.

The aim of this paper is to thoroughly examine the application of handwriting recognition in a real case scenario using a variety of deep learning techniques. Some of these techniques used below are the Feedforward Neural Networks (FNN), the Convolutional Neural Networks (CNN) – with and without data augmentation methods – and the Convolutional Recurrent Neural Networks (CRNN). Finally, we have implemented some pre-trained models (VGG-16, VGG-19, ResNet50) in our dataset in order to see how a model that was already used in such topics like OCR, works in handwritten text images.

EBS Solutions

## 1.1. Who We Are

Today, manually transcribing large amounts of handwritten data is an arduous process that's bound to be fraught with errors. Automated handwriting recognition can drastically cut down on the time required to transcribe large volumes of text and also serve as a framework for developing future applications of machine learning. So, we came up with a new project idea called "DigitPen". We are a consulting start-up company called "EBS Solutions", based in New York city, specializing in app development, especially for issues concerning the Machine Learning area and we have created a new app that we would like to present describing the technological tools on its background with a detailed analysis.

## 1.2. Business Case

We are the Business Analytics team of EBS company and our aim is to present our latest app called "DigitPen", a handwriting recognition technology that allows users to upload (through mobile, tablet or computer) in a specific platform a handwritten image in English converting the handwritten text into digital form. We have created this innovative app as in our company we have meetings with many clients daily and during those sessions, we all write down notes. These notes are formal contracts between us and our clients and we want to transform them into digital documents. Based on our research and findings, this solution is very effective in terms of time saving and simplicity. It is a high-tech solution and its development will be an asset.

## 2. Methodology

The programming language we used for the whole project was Python. We chose to use the Jupyter notebook to implement our python code with the help of Keras and Tensorflow. Jupyter is a client-based interactive web application that allows users to create and share codes, equations, visualizations, as well as text. Keras is a deep learning API written in Python, running on top of the machine learning platform Tensorflow.

The first step of the digital transformation we used was the OCR processing that converts any kind of images containing written or printed text into a machine-readable format. In general, our investigation showed us that image classification problems and image preprocessing can be suitably solved with neural networks such as Feedforward Neural Network (FNN), Convolutional Neural Network (CNN), Data augmentation on data - training a neural network model, Convolution Recurrent Neural Network (CRNN) and also pre-trained models that are appropriate for handwriting recognition. Later on, we will describe with more details every methodology.

### 2.1. Data Overview

The purpose of this assignment is to use an image recognition technology in order to predict the text that each image contains. To get insights about our purpose, we used data from Kaggle platform that contains more than four hundred thousand handwritten names collected through charity projects. More specifically, there are 206,799 first names and 207,024 surnames in total. The dataset is divided into a training set (331,059), testing set (41,382) and validation set (41,382) respectively. The downloaded Kaggle zip file contains 3 csv files (written_name_train_v2.csv, written_name_validation_v2.csv and written_name_test_v2.csv) and 3 folders (train_v2, validation_v2 and test_v2) with handwritten images. Each csv file consists of two columns: Filename column which has the image names (e.g., TRAIN_00001.jpg) and Identity column which has the name that each image contains (e.g., BALTHAZAR). Each folder has a subfolder (train, validation and test) which contains black-white images of handwritten names (characters were black and thin and the background was white). The dataset is available in https://www.kaggle.com/landlord/handwriting-recognition.

## 2.2. Data preprocessing

The initial step was loading the 3 csv files and the images of the three datasets we referred previously on Jupyter. Figure 1 represents some of these images on which we could observe that there are some redundant words like Nom (= Surname) and Prenom (= Name) that are not handwritten text:


Figure 1. *Handwritten Images Visualization*

### 2.2.1. Missing values and unreadable words

After that we converted all the characters in Identity column of the three csv files into upper case letters as we wanted to create the labelling of these images in a next step. Labels are predetermined by a machine learning engineer and are chosen to give the computer vision model information about what is shown in the image. The process of labelling images also helps machine learning engineers hone in on important factors that determine the overall precision and accuracy of their model. Every row in the three files with no label (NA values = empty cells) was removed. We also indicated some '*Unreadable'* labels in Identity column that were also removed as they are not offering any additional information. A sample of unreadable images are shown in Figure 2.

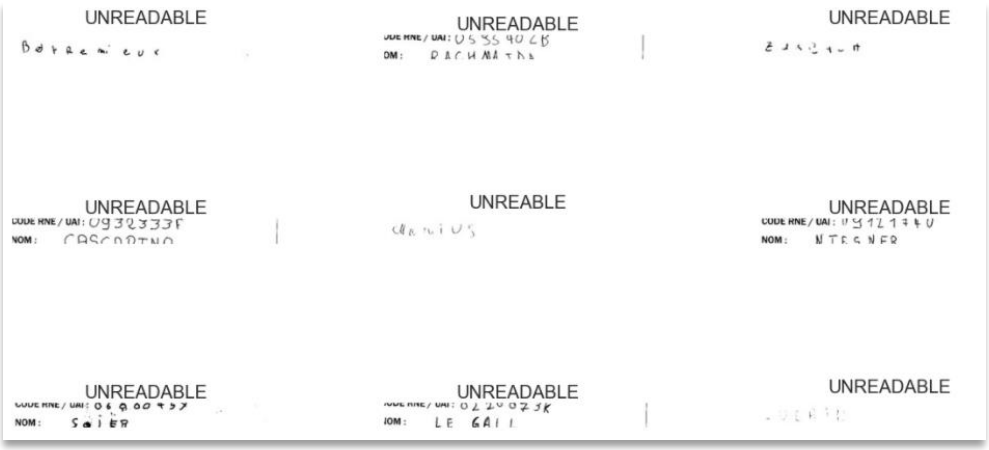



Figure 2. *Unreadable Images Visualization*

### 2.2.2. Image preprocessing function

After the data cleaning procedure, we defined a preprocess function in order to load the images with specific characteristics as follows:

- The images were loaded as grayscale format. A grayscale image is simply one in which the only colors are shades of gray. The reason for gray scaling the images from any other sort of color image is that less information needs to be provided for each pixel. The grayscale intensity is stored as an 8-bit integer giving 256 possible different shades of gray from black to white. Here, the color has no significance on our images so it is better to go for grayscale them to avoid false classification and complexities later.
- We reshaped the images to height 64 and width 256 so as to have all the same shape. The height and the width are cropped on the images if they are greater than 64 and 256 accordingly. If they are smaller than the referred values, then the image is padded with white pixels.
- Finally, the image is rotated clockwise (90 degrees) to bring the image shape to (x, y). Rotating the image this way, makes the models we will construct to predict first the first character of the word.

### 2.2.3. Training/ Validation/ Test dataset

At this point, we defined the size of train, validation and test set, as the files we have, include a huge number of images that cannot fit into our memory. The data has already been separated into training, validation and test set.

- **Training** dataset is the sample of data used to fit the model. The model sees and learns from this data.
- **Validation** dataset is the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
- **Test** dataset is the sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.

This separation is done to prevent the model from overfitting and to accurately evaluate it. For our dataset, we decided to get 30,000 as train set, 3,000 as validation set and 1,000 as test set. Now that the split of sets is done, we will need to further use our training, validation and test set to create the input images arrays and label arrays for the model.

### 2.2.4. Defining the input and output variables of the algorithm

Machine learning algorithms are described as learning a target function that best maps input variables (X) to an output variable (Y).

**a)** Input variables

A fit machine learning model takes inputs and makes a prediction. In our case, the inputs must be defined as arrays of numbers that represent images. Firstly, we created the images' arrays train_x, valid_x and test_x (x-axis data = images) reshaping the data arrays to have a single-color channel. For example, train_x is an array that represents 30,000 images of 256 width, 64 height and single color.

At this point, we have to mention that when we read all the images using the preprocess function, we divided them by 255. The reason is that in each pixel of an image, we have a

number. This number goes from 0 to 255 so 0 is the total white and 255 represents the total black. The division of image by 255 is used to transform any number in these images between 0 and 1 as the neural networks prefer to deal with small input values. When we perform a forward pass (refers to calculation process, values of the output layers from the inputs data) we do matrix multiplications and if we do matrix multiplications with numbers that exceed 1 then these numbers tend to become very large and they will not fit in memory. That is why we normalize these numbers by dividing by the maximum value on every pixel.

**b)** Output variables

The y-axis data that we will use are the labels of the images and we need them to predict the characters of the words. The y-axis data is just a series of numbers associated with each class label. Therefore, we need to define the class labels manually. At this point we created the definition “characters” creating an alphabet that included all the letters and the special characters we found ("-'`" and space). We also created two functions in order to deal with the labels so each character is represented by a number but also a number is represented by a character. In Figure 3 we could see some examples:

```
BALTHAZAR [ 1  0 11 19  7  0 25  0 17]
SIMON [18  8 12 14 13]
BENES [ 1  4 13  4 18]
LA LOVE [11  0 28 11 14 21  4]
DAPHNE [ 3  0 15  7 13  4]
```

Figure 3. *Conversion of character to number*

The name SIMON has the label [18 8 12 14 13] so starting the count from number 0, number 18 represents the S letter of alphabet, number 8 represents the I letter etc. In this way we also have the information about which letter comes first and we don’t lose any history information.

After this, we set the maximum label length equal to 24 as almost all of the names have length up to 24 in our dataset and then we create the arrays train_y, valid_y and test_y which contain

the encoded labels with the integers for each image. For example, train_y array represents 3,000 images of maximum label length 24.

In case of feedforward neural network and pre-trained models, we have to define new input matrices. For feedforward network we created the matrix train_xmlp as in this kind of network we have to do flattening on the input images. Flattening means that we flatten the array to one dimension so we convert the input data into a (30,000, 256×64×1 = 16,384) array. For the pre-trained models, we created new input shapes as they deal with arrays of a specific size which in our case is 128×128×3 where 3 is the number of channels. In case of the pre-trained models, we defined the size of train and validation set equal to 10,000 and 1,000 accordingly due to memory issues. We also created new label matrices as the size changed. On the next sections we are going to discuss every methodology in depth.

## 3. Neural Networks

Neural Networks reflect the behavior of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning and deep learning. They are also known as Artificial Neural Networks (ANNs) or Simulated Neural Networks (SNNs) which are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

A neural network has many layers. Each layer performs a specific function and the complex the network is, the more the layers are. The purest form of a neural network has three layers: the input layer, the hidden layer and the output layer. As the names suggest, each of these layers has a specific purpose. These layers are made up of nodes. There can be multiple hidden layers in a neural network according to the requirements. The input layer picks up the input signals and transfers them to the next layer. It gathers the data from the outside world. The hidden layer performs all the back-end tasks of calculation. A network can even have zero hidden layers. However, a neural network has at least one hidden layer. The output layer transmits the final result of the hidden layer's calculation.

### 3.1. Feedforward Neural Network (FNN)

A Feedforward Neural Network is an Artificial Neural Network in which connections between the nodes do not form a cycle. FNN was the first and simplest type of artificial neural network. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden node and to the output nodes. It does not form a cycle. In FNN inputs are fed by a series of weights which is then computed by Hidden Layers. The hidden layers job is to transform the inputs into something that the output layer can use. Hidden Layers use Activation Function to maps the resulting values in 0 or 1.

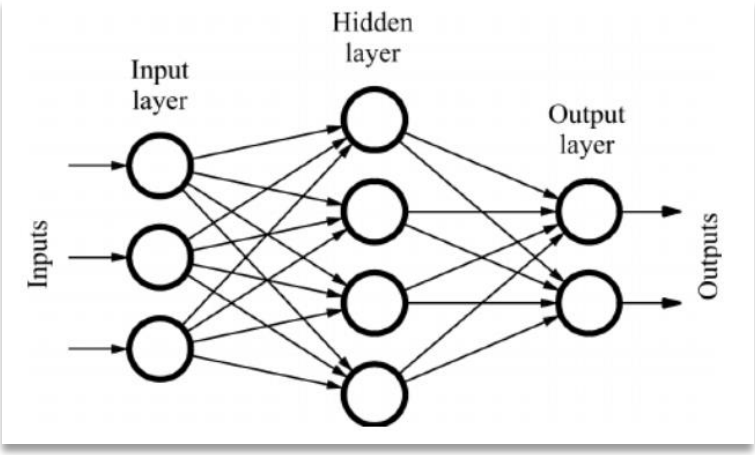



Figure 4. *Feedforward Neural Network Visualization*

In multi-layered perceptrons, the process of updating weights is nearly analogous, however the process is defined more specifically as backpropagation (method for calculating derivatives inside deep feedforward neural networks). In such cases, each hidden layer within the network is adjusted according to the output values produced by the final layer. When training a neural network by gradient descent, a loss function is calculated, which represents how far the network's predictions are from the true labels. Backpropagation allows us to calculate the gradient of the loss function with respect to each of the weights of the network. This enables every weight to be updated individually to gradually reduce the loss function over many training iterations.

Regarding the terminologies that we have already referred we could say that:

- ***Activation Function*** is a function used in artificial neural networks which outputs a small value for small inputs and a larger value if its inputs exceed a threshold. In other words, an activation function is like a gate that checks that an incoming value is greater than a critical number. Activation functions are useful because they add non-linearities into neural networks, allowing the neural networks to learn powerful operations.
- ***Weight*** is the parameter within a neural network that transforms input data within the network's hidden layers. A neural network is a series of nodes, or neurons. Within each node is a set of inputs, weight and a bias value. As an input enters the node, it gets multiplied by a weight value and the resulting output is either observed or passed to the

next layer in the neural network. Often the weights of a neural network are contained within the hidden layers of the network.

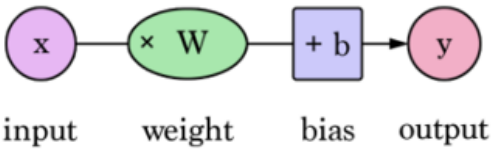



Figure 5. *Input, Weight, Bias and Output schema*

- ***Bias*** represents how far off the predictions are from their intended value. Biases make up the difference between the function's output and its intended output. A low bias suggest that the network is making more assumptions about the form of the output, whereas a high bias value makes less assumptions about the form of the output.

### 3.1.1. Building the Feedforward neural network model

After preprocessing correctly, the dimensions of train_xmlp especially reshaped for this kind of neural network and doing encoding on labels creating train_y, we are ready to start building the Feedforward Neural Network with a single hidden layer.

At first, we define an empty model using *Sequential ()* meaning that each layer that we add per line will use as input the output of the former layer added to the model. Sequential means that we have stacked layers. We build a layer upon another layer, upon another layer and so forth. There are 256 × 64=16,384 neurons in the input layer, 512 neurons in the hidden layer and 24 neurons in the output layer, one for each character as we defined the maximum predicted label length equal to 24. Softmax is a special activation function that transforms the output into probability values of each class. Therefore, with *Dense (24),* we will have 24 neurons each representing the probability of a given character.

Afterwards, we compile the model. The compilation is the final step in creating a model and when compilation is done, we can move on to training phase. In compilation of the model, we use as loss function the Mean Absolute Error, as optimizer the RMSprop, and as a metric the accuracy. Below you could see the aforementioned terms in more detail:

**Loss function:** is used to determine the loss between the output of our algorithms and the given target value. Here we use ***Mean Absolute Error (MAE)*** loss function which gives the absolute difference between the actual values and the values predicted by the model for the target variable. MAE values closer to zero are better.

**Optimizer:** ***RMSprop*** is a gradient based optimization technique used in training neural networks. Gradients of very complex functions like neural networks have a tendency to either vanish or explode as the data propagates through the function (vanishing gradients problem). RMSprop was developed as a stochastic technique for mini-batch learning and deals with the above issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size (momentum), decreasing the step for large gradients to avoid exploding and increasing the step for small gradients to avoid vanishing. Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. This means that the learning rate changes over time.

**Metrics:** we use the ***accuracy*** inside compilation. *Accuracy* is the ratio of the number of correct predictions to the total number of input samples, which is usually what we refer to when we use the term accuracy.

Before we fit the model, we have to specify the number of epochs and the batch size. Also, we have to decide if we will use an early stopping method. Regarding these terms:

**Early stopping:** is a method that allows us to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a holdout validation dataset.

**Epoch**: indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model.

**Batch size:** refers to the number of samples utilized in one iteration.

In order to train our model, we choose to have 30 epochs, batch size equal to 128, callbacks (early stopping) and validation split equal to 0.1 meaning that Keras will split apart a fraction (10%) of the training data to be used as validation data. The results implies that the model is not good enough in terms of train and validation accuracy which are both low and close to **15**%. The train and validation loss are close to 3.5. In Figure 6, we could see the model architecture of Feedforward neural network.

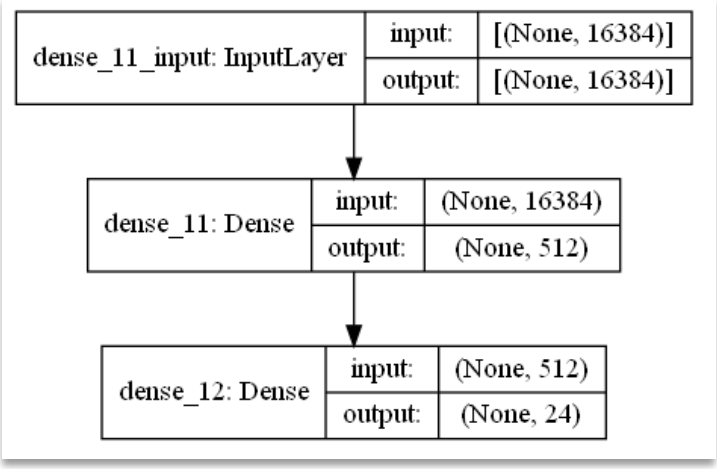



Figure 6. *Architecture of Feedforward neural network model*

### 3.1.2. Evaluation of Feedforward neural network model

In order to evaluate our model, we make predictions on the test dataset reshaping the test_images array (x) in the same way we reshaped the train_xmlp array. The dimensions of the test_images array after the reshaping are (1,000, 256 × 64). Also, the test_y array was reshaped in the way that train_y array was reshaped. Finally, the test accuracy is 14.90%, a percentage very low for predicting handwritten words.

```
32/32 [==============================] - 0s 8ms/step - loss: 3.4611 - accuracy: 0.1490
```

Table 1. *Test Accuracy on Feedforward neural network model*

## 3.2. Convolutional Neural Network (CNN)

A *Convolutional Neural Network* (CNN) is a class of deep learning model that is predominantly used for image recognition. CNN is successful in identifying faces, objects, traffic signs and also used in self-driving cars. This network is similar to ordinary neural networks but has multiple hidden layers and a filter called the convolution layer. The hidden layers have 4 different procedures to complete the network. Each one is explained in detail.

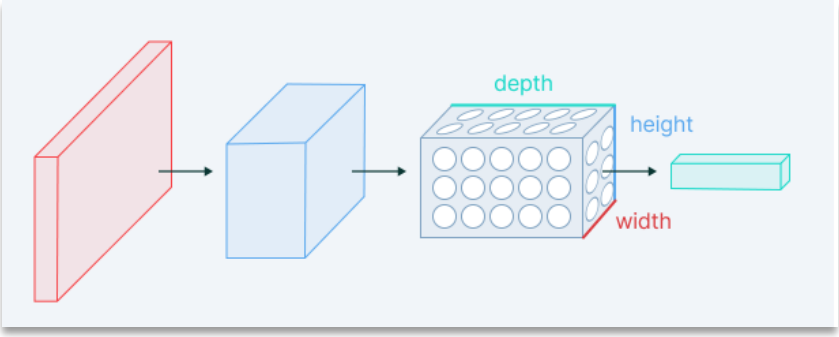



Figure 7. *Convolutional Neural Network Visualization*

**Convolution layer**

Convolution layer is the heart of a Convolutional Neural Network, which does most of the computational operations. The name comes from the "convolution" operator that extracts features from the input image. These are also called filters. The matrix formed by sliding the filter over the full image and calculating the dot product between these 2 matrices is called the 'Convolved Feature' or 'Activation Map' or the 'Feature Map'. Suppose that in table data, different types of features are calculated like "age" from "date of birth." The same way here also, straight edges, simple colors and curves are some of the features that the filter will extract from the image. During the training of the CNN, it learns the numbers or values present inside the filter and uses them on testing data. The greater the number of features, the more the image features get extracted and recognize all patterns in unseen images.

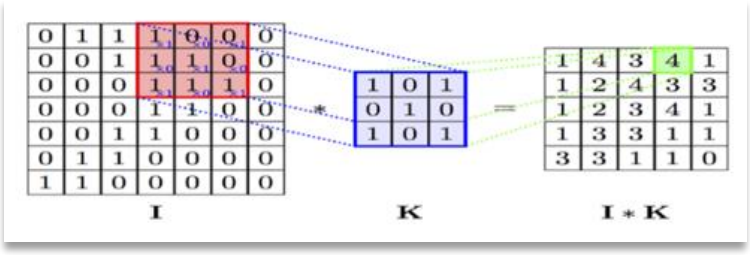



Figure 8. *Convolutional Layer*

**Pooling Layer**

Pooling or subsampling is used to decrease the dimensionality of the feature without losing important information. It is done to reduce the huge number of inputs to a full connected layer and computation required to process the model. It also helps to reduce the overfitting of the model. It uses a 2 x 2 window and slides over the image and takes the maximum value in each region as shown in the figure. This is how it reduces dimensionality.

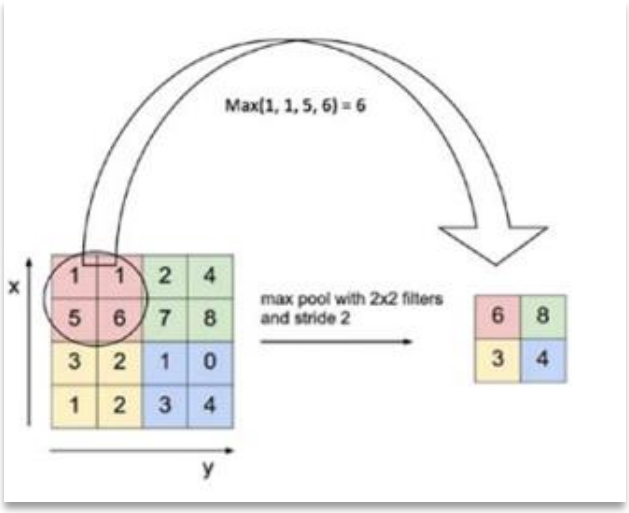



Figure 9. *Pooling layer*

**ReLU Activation Function**

Rectified Linear Unit (*ReLU)* is a nonlinear function that is used after a convolution layer in CNN architecture. It replaces all negative values in the matrix to zero. The purpose of ReLU is to introduce nonlinearity in the CNN to perform better.

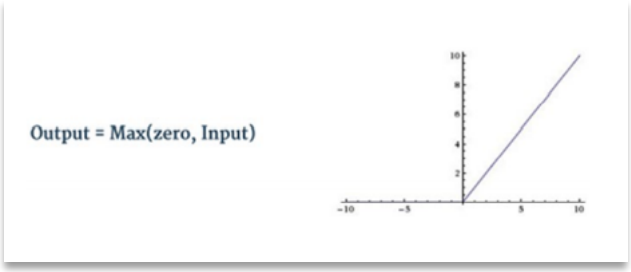



Figure 10. *Non-Linearity (ReLU)*

Finally, the last layer is a ***dense layer*** that needs feature vectors as input. But the output from the pooling layer is not a 1D feature vector. This process of converting the output of convolution to a feature vector is called flattening. The *Fully Connected layer* takes an input from the *flatten layer* and gives out an N-dimensional vector where N is the number of classes. The function of the fully connected layer is to use these features for classifying the input image into various classes based on the loss function on the training dataset. The ***Softmax function*** is used at the very end to convert these N-dimensional vectors into a probability for each class, which will eventually classify the image into a particular class.

### 3.2.1. Building the Convolutional Neural Network model

At first, we begin with an empty model using *Sequential ()* model and then we construct it layer by layer specifying some parameters of each layer.

**First Layer:** The first layer in the model is a 2-dimensional convolutional layer. This layer will have 64 output filters each with a kernel size of 3x3. We also use the ReLU activation function. Note that the choice for the number of output filters specified is arbitrary and the chosen kernel size of 3x3 is generally a very common size to use. *Keras Conv2D* is a 2D Convolution Layer which creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs. In image processing kernel is a convolution matrix which can be used for blurring, sharpening, embossing, edge detection, and more by doing a convolution between a kernel and an image. Mandatory Conv2D parameter is the numbers of filters that convolutional layers will learn from. It is an integer value and also determines the number of output filters in the convolution. Here in our case, we are learning a total of 64 filters and then we use *MaxPooling2D* to reduce the spatial dimensions of the output volume. As far as choosing the appropriate value for number of filters, it is always recommended to use powers of 2 as the values, that is why the both values of the two Conv2D layers we put have numbers that are power of 2 (e.g., 64, 128, 256 etc.). On the first layer only, we also specify the input_shape, which is the shape of our data. Our images are 256 pixels high and 64 pixels wide and have 1 color channel: grayscale. This gives us an input_shape of (256, 64, 1). We then add a MaxPooling2D layer to pool and reduce the dimensionality of the data. After this step, we add a *BatchNormalization* method. Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data improving the learning speed of Neural Networks and provides regularization, avoiding overfitting.

**Second Layer:** We follow by adding another convolutional layer with the different specs as the earlier one. The choice of 128 output filters of size 3x3 here is again arbitrary, but the general choice of having more filters in later layers than in earlier ones is common. As we can see, both of the Conv2D layers we have used, have padding defined as same so zero padding. *Zero padding* is a technique that allows us to preserve the original input size. This is something that we specify on a per-convolutional layer basis. With each convolutional layer, just as we define how many filters to have and the size of the filters, we can also specify whether or not to use padding. The second Conv2D layer is again followed by activation ReLU, the same type of MaxPooling2D layer and before we use again Batch Norm technique, we use a dropout layer. *Dropout* is a technique used to prevent a model from overfitting. *Dropout* works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase. In passing 0.3, every hidden unit (neuron) is set to 0 with

a probability of 0.3. In other words, there is a 30% change that the output of a given neuron will be forced to 0. After Batch Norm, we used Flatten layer.

**Third Layer**: Flatten layers allow us to change the shape of the data from a vector of 2D matrixes into the correct format for a dense layer to interpret. It's simply allowing the data to be operable by this different layer type. Then we use again the same dropout layer we referred previously.

**Forth Layer:** We have again 128 output filters of size 3x3 but this time we define another argument called kernel initializer inside this layer. Initializers define the way to set the initial random weights of Keras layers. Initializer *he_normal* draws samples from a truncated normal distribution centered on 0. Then we use again the activation function ReLU and a dropout layer setting inside it the number 0.2 meaning that there's now a 20% change that the output of a given neuron will be forced to 0.

**Fifth Layer**: The last Dense layer is the output layer of the network and so it has 24 nodes, one for each letter in the maximum length of 24. We use again activation ReLU and dropout(0.2) layer and finally we use the Softmax activation function on our output so that the output for each sample is a probability distribution over the outputs of each of the letters.

Afterwards, we compile the CNN model. Here we use again Mean Absolute Error as loss function and accuracy metric but with a different **optimizer** called **Adam**. Adam is an alternative optimization algorithm that provides more efficient neural network weights by running repeated cycles of adaptive moment estimation. Adam extends on stochastic gradient descent to solve non-convex problems faster while using fewer resources than many other optimization programs. It is most effective in extremely large data sets by keeping the gradients "tighter" over many learning iterations.

For the fit of the model, we use the same parameters as before (30 epochs, 128 batch size, early stopping, 0.1 validation split). Again, the train and validation accuracy are not as expected with values close to 22% and loss values close to 3.5. In Figure 11, we could see the model architecture of CNN.

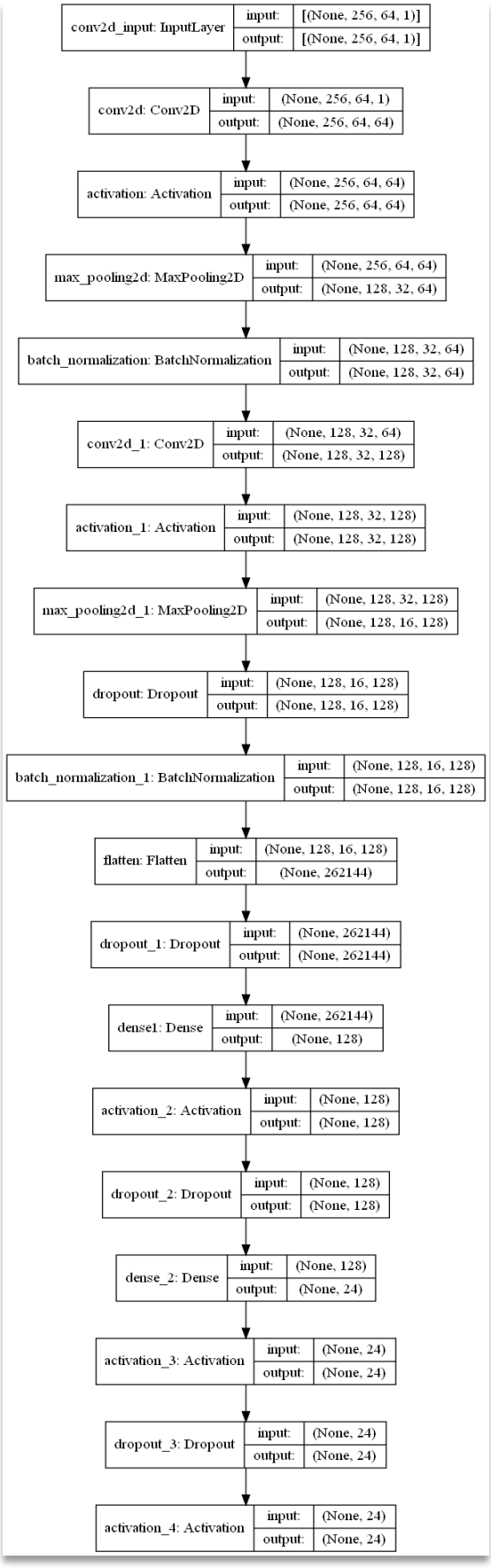



Figure 11. *Architecture of Convolutional Neural Network*

### 3.2.2. Evaluation of Convolutional Neural Network model

In order to evaluate our model, we make predictions on the test dataset. The test accuracy is 23.50% as you could see from Table 2.

```
32/32 [==============================] - 5s 147ms/step - loss: 3.4452 - accuracy: 0.2350
Test loss: 3.4452083110809326:
Test accuracy: 23.500 %
```

Table 2. *Test accuracy on Convolutional Neural Network*

## 3.3. Data Augmentation on Convolutional Neural Network

In this section we have performed data augmentation on CNN. Data augmentation encompasses a wide range of techniques used to generate new training samples from the original ones by applying random jitters and perturbations, but at the same time ensuring that the class labels of the data are not changed. We perform this technique in order to increase the generalizability of our model. Since the network is constantly seeing new, slightly modified versions of the input data, the network is able to learn more robust features.

There are many ways to perform augmentation on the CNN model. Here, we explore horizontal flips, random brightness, zoom and rotations of images. In order to perform the methods, we have used *ImageDataGenerator* from Keras. This technique is appropriate because we can augment the images while the model is still training. It is a very well know method in deep learning and also it requires lower memory usage.

In order to see how augmentation techniques are applied in our data, before the fitting of the models, we choose to display the first image of the validation dataset as an example.

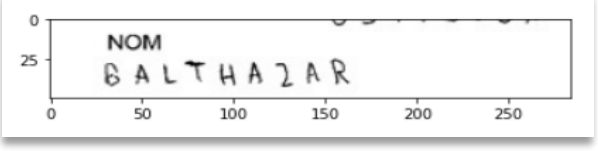



Figure 12. *First image of validation dataset*

✓ **Horizontal flip augmentation**

Reversing the entire rows and columns of an image pixels in horizontally is called horizontal flip augmentation. In Figure 13 we could see generally the differences of the Horizontal and the Vertical flip. Horizontal flip is like a 'mirror' image of the actual image.

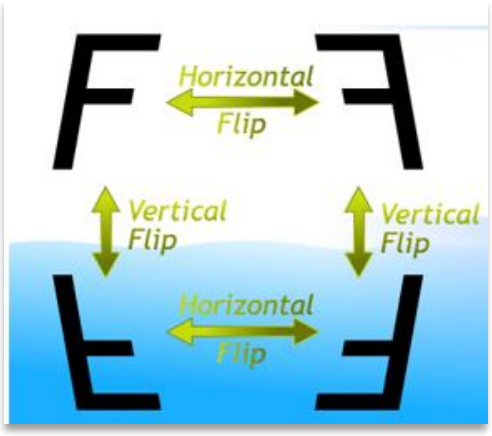



Figure 13. *Difference of Horizontal and Vertical Flip*

An example of how horizontal flip works in our data can be seen below:


Figure 14. *Horizontal flip augmentation in the image of Balthazar name*

✓ **Random Brightness augmentation**

The brightness of an image could be augmented by either randomly darkening images, brightening images or both. The intent is to allow a model to generalize across images trained

on different lighting levels. This could be achieved by specifying the brightness range argument to the ImageDataGenerator () constructor that specifies min and max range as a float representing a percentage for selecting a brightening amount. Values less than 1.0 darken the image, e.g. [0.5, 1.0], whereas values larger than 1.0 brighten the image, e.g. [1.0, 1.5], where 1.0 has no effect on brightness.

In the image of BALTHAZAR name, we use brightness range [0.4, 1.0] so the letters will have a contrast with the image background. In Figure 15 we could see the method 6 different times.

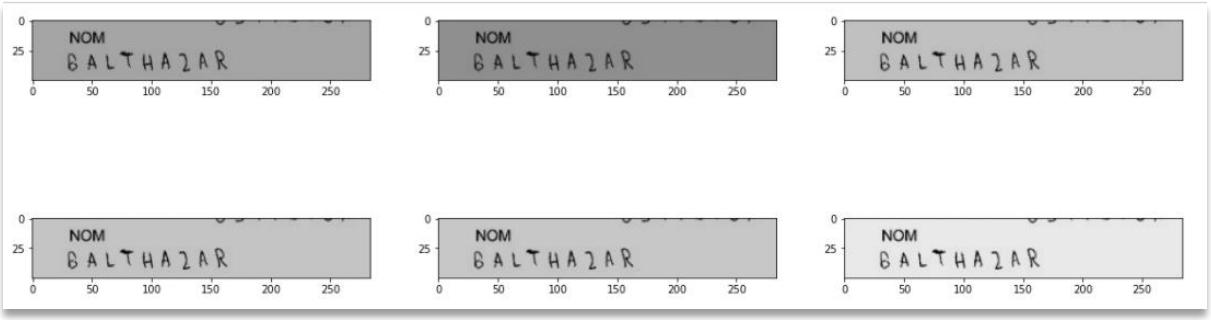



Figure 15. *Random Brightness augmentation in the image of Balthazar name*

Even though the letters are in black colour, the dark background helps recognize them easier. This method could be very helpful in the model that we are going to create afterwards. The model could probably learn more patterns through this contrast.

✓ **Random Zoom augmentation**

Random Zoom Image augmentation is used to generate images with varying zoom levels for feeding our deep learning model. Image zooming can be configured by the zoom range argument to the ImageDataGenerator constructor. Someone can specify the percentage of the zoom as a single float or a range as an array or tuple. If a float is specified, then the range for the zoom will be [1-value, 1+value]. For example, if we specify 0.3, then the range will be [0.7, 1.3], or between 70% (zoom in) and 130% (zoom out).

The zoom amount is uniformly randomly sampled from the zoom region for each dimension (width, height) separately. The zoom may not feel intuitive. That zoom values less than 1.0 will zoom the image in, e.g. [0.5,0.5] makes the object in the image 50% larger or closer. Values larger than 1.0 will zoom the image out by 50%, e.g. [1.5, 1.5] makes the object in the image smaller or further away. A zoom of [1.0,1.0] has no effect.

In the image of BALTHAZAR name, we set the zoom range between 0.5 and 1.0, so as to establish a 50% random zoom.

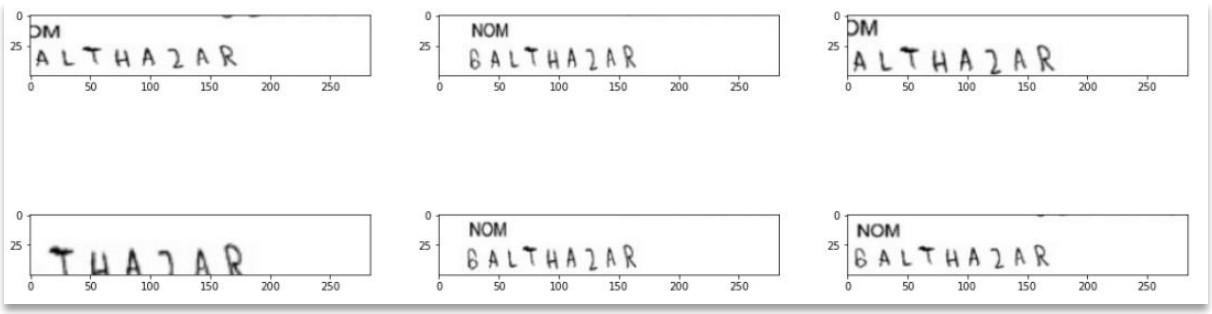


Figure 16. *Random Zoom augmentation in the image of Balthazar name*

Random zoom as well as brightness augmentation are the most important techniques for image classification based on handwriting recognition. Zooming on images could clear the pixels and as follows the letters. On the other hand, brightness could help the situation while the letters will be in high contrast. These were examples of how the techniques work, especially in the image of Balthazar name. Below we are going to work with zoom and brightness techniques and train CNN models in order to generate more data, except the original ones.

### 3.3.1. Data augmentation on CNN using Zoom and Brightness techniques

We used ImageDataGenerator in order to generate new training samples from the original training dataset with zoom range from 0.4 to 1.0 and brightness range from 0.2 to 1.0. In Figure 17 we could see a small overview of the images that were produced.


Figure 17. *Zoom and brightness data augmentation in an image of training dataset*

For model fitting we use 30 epochs and128 batch size with callbacks. The model achieved an accuracy of 22.46% on the fourth epoch (where it stopped) and a better validation accuracy of 23.17%.

✓ **Evaluation of CNN model with Zoom and Brightness data augmentation**

In order to evaluate our model, we make predictions on the test dataset. The test accuracy is 12.70% as you could see from Table 3.

```
32/32 [==============================] - 5s 159ms/step - loss: 3.4720 - accuracy: 0.1270
Test loss: 3.4719583988189697:
Test accuracy: 12.700 %
```

Table 3. *Test accuracy of CNN model with Zoom and Brightness data augmentation*

### 3.3.2. Data augmentation on CNN using rotation technique

A rotation augmentation randomly rotates the image clockwise by a given number of degrees from 0 to 360. The rotation likely rotates pixels out of the image frame and leave areas of the frame with no pixel data that must be filled in. In order to perform a model like this, we have used again the ImageDataGenerator, where we used a rotation range of 30. This means that the

rotations to the image are between 0 and 30 degrees. Also, we use a horizontal and vertical flip for the image processing.

An overview of how image look can be seen below, where the name has different views and more patterns are generated.

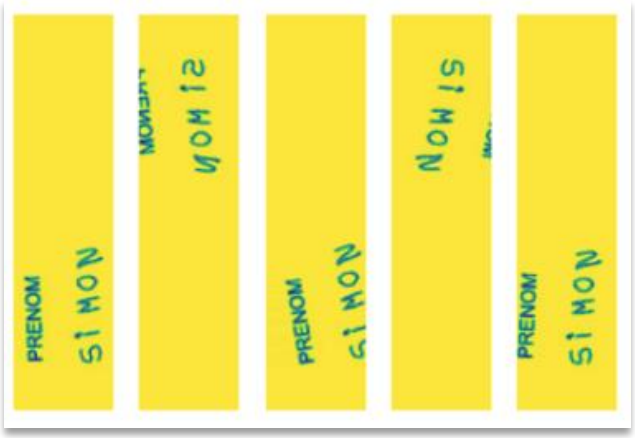



Figure 18. *Rotation data augmentation in an image of training dataset*

Afterwards we fit the CNN model, by using the same parameters as before. The model stopped again at the fourth epoch, with an accuracy of 16.26% in the training set and 15.13% in the validation set.

✓ **Evaluation of CNN model with rotation data augmentation**

We evaluate the model in the test set, where achieve a higher accuracy of 23.5%.

```
32/32 [==============================] - 5s 157ms/step - loss: 3.4452 - accuracy: 0.2350
Test loss: 3.4452078342437744:
Test accuracy: 23.500 %
```

Table 4. *Test accuracy of CNN model with rotation data augmentation*

## 3.4. Convolutional Recurrent Neural Network (CRNN)

On this chapter, we are going to see another kind of neural network which is the *Convolutional Recurrent Neural Network* (CRNN). CRNN involves CNN followed by the RNN (Recurrent neural networks) to process images containing sequence information such as letters. It is mainly used for OCR technology and has some advantages.

1. End-to-end learning is possible: a technique where the model learns all the steps between the initial input phase and the final output result.
2. Sequence data of arbitrary length can be processed because of LSTM (long short-term memory) which is free in size of input and output sequence.
3. There is no need for cropping technique to find each character one by one.

While CNN help us extracting relevant features in the image, *RNN* help the Neural Network to take into consideration information from the past in order to make predictions or analyze. RNN are like other ANN abstractions of biological nervous systems, yet they differ from them in allowing using their internal memory of the training to be fed recurrently to the neural network. RNNs are widely used for sequence analysis because they are constructed to extract contextual information from sequences by modelling the dependencies between different time steps where time is the portion of a pass in which node inputs are processed into outputs, and then those outputs are fed to the next node. A high-level representation of an RNN can be seen in the following figure:

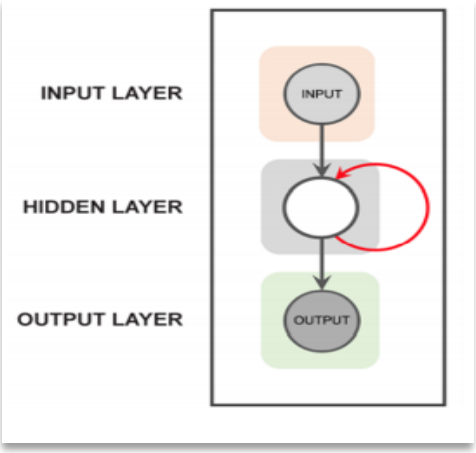



Figure 19. *Structure of RNN*

Here we will use a combination of convolution and recurrent neural network as we understood how these networks work. Most of the time, the Convolutional Neural Network analyzes the image, sending it to the recurrent part of the important features detected. The recurrent part analyzes these features in order, taking into consideration previous information in order to realize what are some important links between these features that influence the output. In Figure 20, we could see a representation of the structure of CRNN.

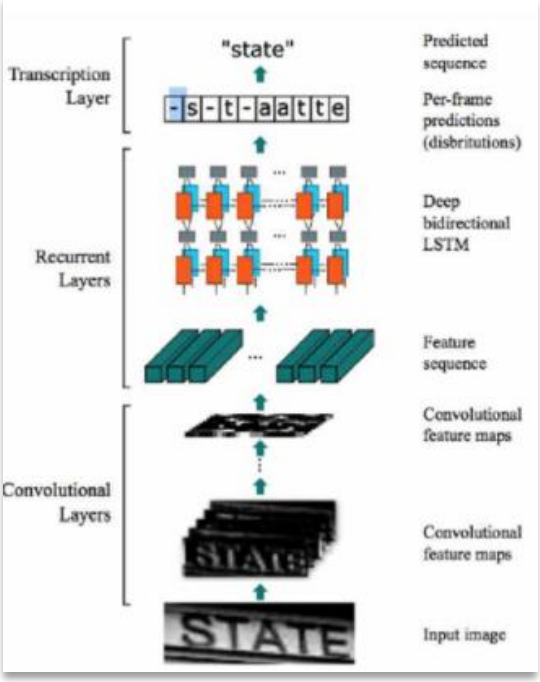



Figure 20. *Structure of CRNN*

### 3.4.1. Building the Convolutional Recurrent Neural Network model

At first, we begin with an input image of height 64 and width 256. Here we used three convolution layers having kernel size (3,3). The number of filters is increased from 32 to 128 layer by layer. Three max-pooling layers are added with two of them having size (2,2) and the last one with size (1,2) to extract features with a larger width to predict long names. Also, we used batch normalization layers after every convolution layer in order to accelerate the training process and dropout with a dropout rate of 30%. The convolutional layers are followed by the

reshape layer which is very necessary for CRNN as the shape of the feature vector differs from CNN to RNN. The convolutional layers are developed on 3-dimensional feature vectors, whereas the recurrent neural networks are developed on 2-dimensional feature vectors. Then we used two Bidirectional LSTM (BiLSTM) layers each of which has 256 units. BiLSTM layers learns bidirectional long-term dependencies between sequence data. This RNN layer gives the output of size (batch_size, 64, 30) where 64 is the total number of output classes including blank character. Finally, the output of the bidirectional layers is fed to the time distributed dense layers followed by the Fully connected layer.

**Loss Function**

Now we have prepared model architecture, the next thing is to choose a loss function. In this text recognition problem, we will use the CTC (Connectionist Temporal Classification) loss function. CTC loss is very helpful in text recognition problems. It helps us to prevent annotating each time step and help us to get rid of the problem where a single character can span multiple time step which needs further processing if we do not use CTC.

A CTC loss function requires four arguments to compute the loss, predicted outputs, ground truth labels, input sequence length to LSTM and ground truth label length. To get this we need to create a custom loss function and then pass it to the model. To make it compatible with our model, we will create a model which takes these four inputs and outputs the loss.

**Optimizer and evaluation metrics**

To train the model we use RMSprop (Root Mean Square Prop) optimizer with learning rate 0.001. RMSProp works by keeping an exponentially weighted average of the squares of past gradients. RMSProp then divides the learning rate by this average to speed up convergence. We also tried to use Adam optimizer but the results were better with RMSprop.

For evaluation we choose accuracy metric.

For the fit of the model, we use 30 epochs and 128 for batch size. In Figure 21, we could see the model architecture of CRNN.

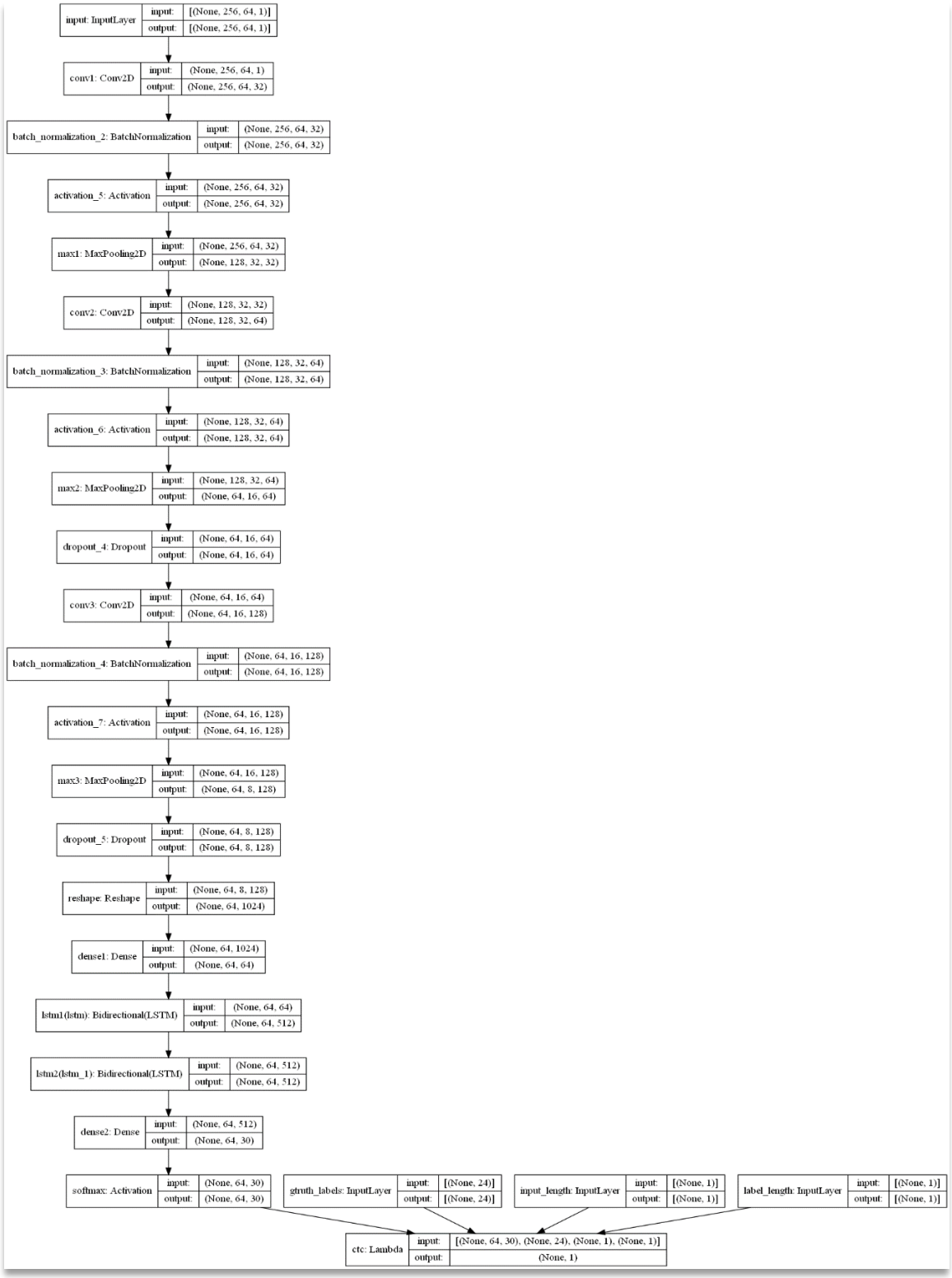


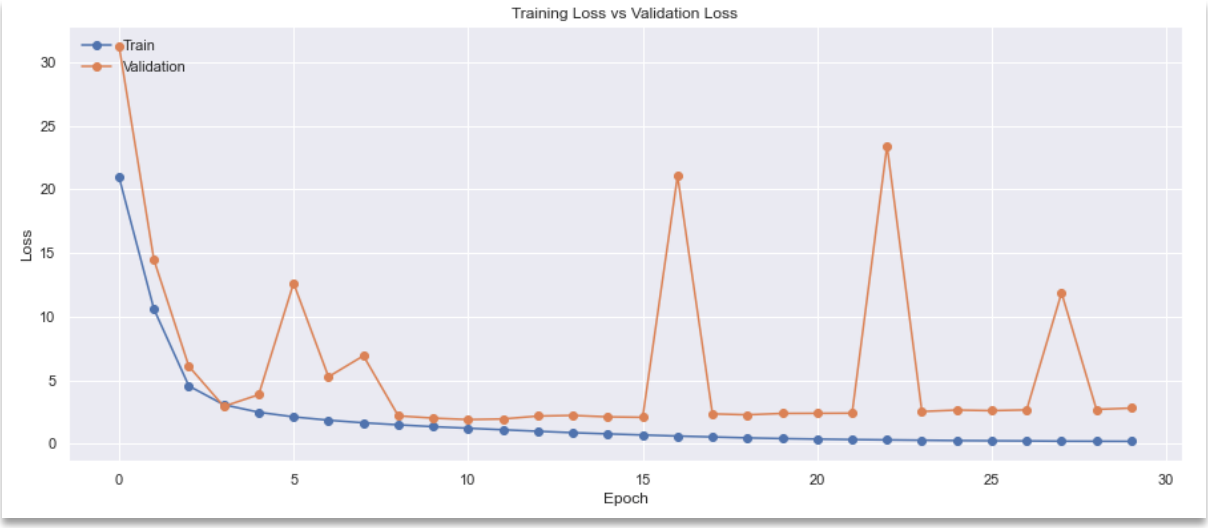

Figure 21. *Architecture of Convolutional Recurrent Neural Network*

### 3.4.2. Results

In Plot 1 we could see the evolution of the accuracy of the model within the 30 epochs. The blue line represents the train data while the orange the validation. The peak performance of the accuracy of the train dataset is approximately **91%**, while the same measure for the validation dataset is about 75%. We observe that the accuracy of the train dataset is higher in general, but this result is unreliable as the accuracy of the train dataset is the result of overfitting. For this reason, we focus on the accuracy of the validation dataset. It can be seen that for both datasets the accuracy is increasing till the epoch 4, then for the train dataset the accuracy is still increasing but with a decreasing rate while for the validation dataset the accuracy has many fluctuations.


Plot 1. Accuracy metric of CRNN model

Plot 2 shows the evolution of the loss of the model within the 30 epochs. The blue line represents the train data while the orange the validation. We observe that the loss of the train dataset is less than the loss of the validation. It can be seen that for both datasets the loss is decreasing till the epoch 2, then for the train dataset the loss remains stable while for the validation dataset the loss has few fluctuations.




Plot 2. Loss of CRNN model

### 3.4.3. Evaluation of Convolutional Recurrent Neural Network model

As our model predicts the probability for each class at each time step, we need to use some transcription function to convert it into actual texts. Here we will use the CTC decoder to get the output text with greedy which performs much faster path search.

Finally, the CRNN model predicts efficiently 87.97% of characters and 72.20% of words in the test dataset. The percentages are high enough which implies that CRNN is a sufficient model and could predict correctly the handwritten words.

| | ***Percentage*** |
|---|---|
| ***Correct characters predicted*** | *87.97%* |
| ***Correct words predicted*** | *72.20%* |

Table 5. *Percentage of correct predictions in the test dataset using CRNN model*

### 3.4.4. Classification report and confusion matrix

The classification report shows a representation of the main classification metrics on a per-class basis. This gives a deeper intuition of the classifier behavior over global accuracy which can mask functional weaknesses in one class of a multiclass problem. More specifically, True Positives, False Positives, True negatives and False Negatives are used to predict the metrics of a classification report.

The report shows the main classification metrics precision, recall, f1-score and support on a per-class basis. The metrics are calculated by using true and false positives, true and false negatives. Positive and negative in this case are the characters for the predicted classes. There are four ways to check if the predictions are right or wrong:

| | |
|---|---|
| **TN / True Negative** | When a case was negative and predicted negative |
| **TP / True Positive** | When a case was positive and predicted positive |
| **FN / False Negative** | When a case was positive but predicted negative |
| **FP / False Positive** | When a case was negative but predicted positive |

Using this terminology, the metrics are defined as follows:

- **Precision:** Precision can be seen as a measure of a classifier's exactness. For each class, it is defined as the ratio of true positives to the sum of true and false positives. Said in another way for all instances classified positive, what percent was correct.
- **Recall:** Recall is a measure of the classifier's completeness; the ability of a classifier to correctly find all positive instances. For each class, it is defined as the ratio of true positives to the sum of true positives and false negatives. Said in another way for all instances that were actually positive, what percent was classified correctly.
- **F1 score:** The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Generally speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation.
- **Support:** Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported

scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

Figure 22 shows the classification Report of CRNN model.

Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| | 0.50 | 0.71 | 0.59 | 35 |
| ' | 0.00 | 0.00 | 0.00 | 1 |
| - | 0.69 | 0.58 | 0.63 | 19 |
| A | 0.90 | 0.90 | 0.90 | 799 |
| B | 0.81 | 0.89 | 0.85 | 128 |
| C | 0.93 | 0.89 | 0.91 | 214 |
| D | 0.81 | 0.87 | 0.84 | 225 |
| E | 0.88 | 0.89 | 0.89 | 810 |
| F | 0.78 | 0.85 | 0.81 | 61 |
| G | 0.79 | 0.88 | 0.83 | 106 |
| H | 0.85 | 0.86 | 0.85 | 186 |
| I | 0.87 | 0.87 | 0.87 | 526 |
| J | 0.70 | 0.80 | 0.75 | 46 |
| K | 0.86 | 0.96 | 0.91 | 46 |
| L | 0.91 | 0.92 | 0.92 | 477 |
| M | 0.84 | 0.83 | 0.83 | 258 |
| N | 0.89 | 0.88 | 0.89 | 520 |
| O | 0.91 | 0.87 | 0.89 | 411 |
| P | 0.82 | 0.74 | 0.78 | 95 |
| Q | 0.94 | 0.76 | 0.84 | 21 |
| R | 0.90 | 0.88 | 0.89 | 450 |
| S | 0.90 | 0.86 | 0.88 | 240 |
| T | 0.88 | 0.85 | 0.87 | 310 |
| U | 0.86 | 0.88 | 0.87 | 280 |
| V | 0.82 | 0.76 | 0.79 | 89 |
| W | 0.58 | 0.75 | 0.65 | 20 |
| X | 0.91 | 0.87 | 0.89 | 23 |
| Y | 0.82 | 0.78 | 0.80 | 89 |
| Z | 0.76 | 0.81 | 0.79 | 43 |
| accuracy | | | 0.87 | 6528 |
| macro avg | 0.80 | 0.81 | 0.80 | 6528 |
| weighted avg | 0.87 | 0.87 | 0.87 | 6528 |

Figure 22. *Classification Report of CRNN model*

From Figure 22 we could see that for characters C and Q the precision was 0.93 and 0.94 respectively. That means that of the things the model classified as C, 93% of them really were and of the things the model classified as class Q, 94% of them really were. For special character ' the precision was zero meaning that of the things the model classified as special character ',

0% of them really were. The macro average precision is 0.80, and the weighted average is 0.87. The weighted average is higher for this model because the place where precision fell down was for special character ', but it is underrepresented in this dataset (only 1/6528), so accounted for less in the weighted average.

**Confusion Matrix**

A confusion matrix is a table that is often used to describe the performance of a classification model on a test dataset for which the true values are known. In the main diagonal are the values that are correctly classified. In the confusion matrix below we could observe that the most characters are correctly classified and only a few of them is misclassified. For instance, 95 characters of the test dataset are correctly predicted as K and 91 are correctly predicted as L. Regarding the misclassification, 10 characters are predicted as N while they belong to the W class.

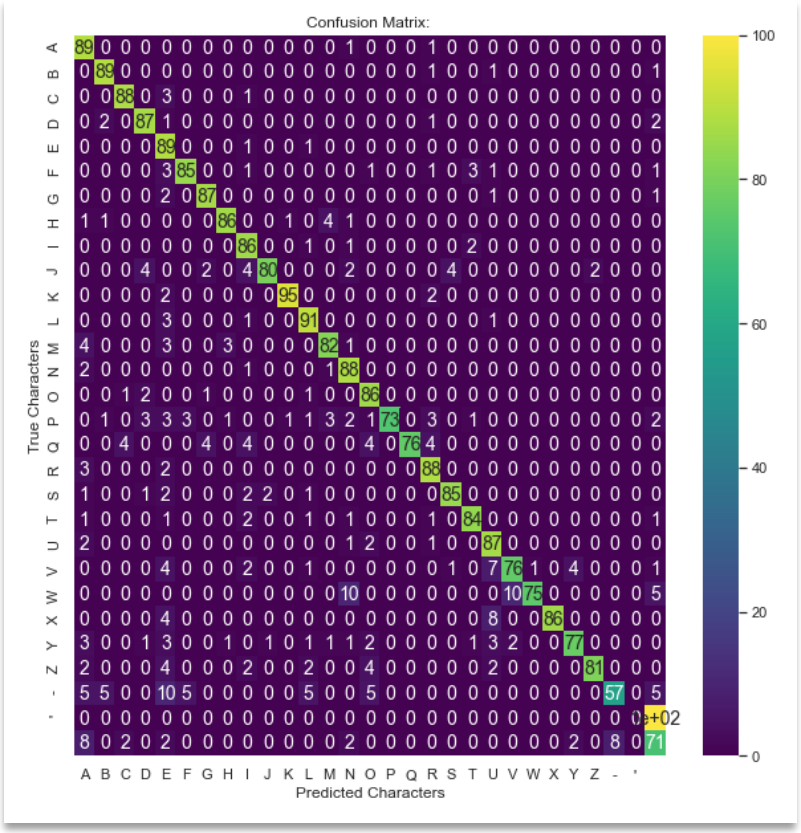



Figure 23. *Classification Report of CRNN model*

### 3.4.5. Jaro – Winkler Similarity

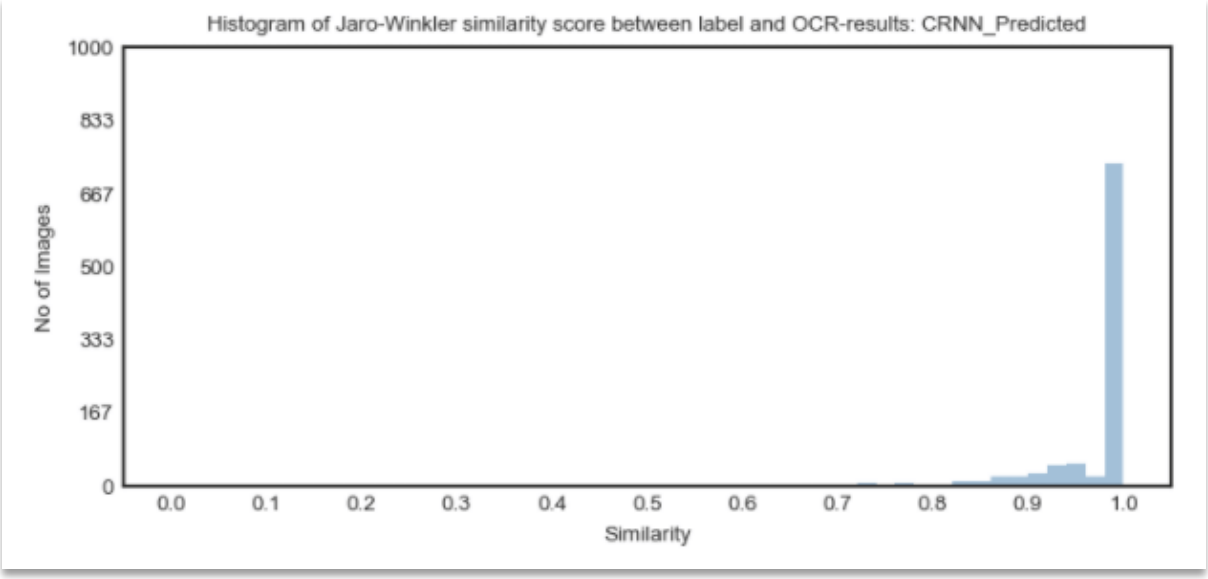
The Jaro-Winkler distance is a string metric for measuring the edit distance between two sequences. Informally, the Jaro distance between two words is the minimum number of single-character transpositions required to change one word into the other. The Jaro-Winkler distance uses a prefix scale which gives more favorable ratings to strings that match from the beginning for a set prefix length. From the below table we could see the first 25 names of the test dataset with their predicted values and the similarity score. The most names are completely similar strings having 1 as similarity score while only one declares a weaker similarity (i.e., CLOTAIRE => WVCIBTENSS with similarity score 0.55).

Mean Squared Error: CRNN_Predicted 22.45

| | IDENTITY | CRNN_Predicted | SIMILARITY_SCORE |
|---|---|---|---|
| 0 | KEVIN | KEVIN | 1.000000 |
| 1 | CLOTAIRE | WVCIBTENSS | 0.550000 |
| 2 | LENA | LENA | 1.000000 |
| 3 | JULES | JULES | 1.000000 |
| 4 | CHERPIN | CHERPIN | 1.000000 |
| 5 | MARTIN | MARTIN | 1.000000 |
| 6 | VALENTINE | VALENTINE | 1.000000 |
| 7 | LORAS | LORAS | 1.000000 |
| 8 | THIBAULT | THIBAULT | 1.000000 |
| 9 | AZABI | UAZABLN | 0.790476 |
| 10 | GORTCHAKOFF | GORTCHAKOTFEN | 0.989277 |
| 11 | MAHENTHIRAN | MAHENTHIRAN | 1.000000 |
| 12 | FRANSOISSISEPH | FFRANZOISSPZEBH | 0.787532 |
| 13 | JEANNE | JEANNE | 1.000000 |
| 14 | DEBORAH | DEBORAH | 1.000000 |
| 15 | DROUCS | DROUES | 0.933333 |
| 16 | JADE | JOSE | 0.700000 |
| 17 | CORNILFRERROT | CORNIL FRERROT | 0.990476 |
| 18 | CLEMET | CLEMENT | 0.976190 |
| 19 | RELIS | AELIS | 0.866667 |
| 20 | NAMIZATAFATIM | WNAMIZATA FATIMUE | 0.921569 |
| 21 | FOURNEL | FOURNEL | 1.000000 |
| 22 | DICINTIO-ILLESCA | DICINTIOIILLESC | 0.916667 |
| 23 | BARDOT | BARDOT | 1.000000 |
| 24 | DUVAL | DUYAL | 0.893333 |

Table 6. *Jaro – Winkler Similarity Score*

The results of the table are more easily understandable in the plot below:




Plot 3. *Histogram of Jaro – Winkler similarity score*

### 3.4.6. Predictions on test dataset using CRNN model

In Figure 24 we could observe some predictions of the test dataset using the CRNN model. The handwritten names are the images of the test dataset and the titles above them the predicted values.

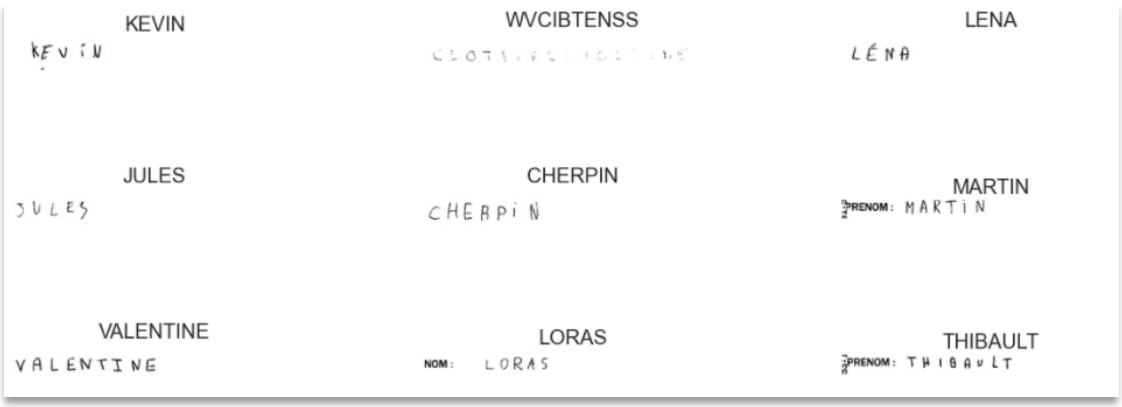



Figure 24. *Predictions on test dataset using CRNN model*

## 3.5. Pre-trained Models

In this section, we will explore how pre-trained models work in our datasets. So far, we have learned a lot about designing and training our own NN models. As we have shown some of our models (like FNN and CNN) did not perform very well. This can be due to multiple reasons such as the small dataset or the more training our models requires. However, training some models like CNN without callbacks takes a lot of time. Transfer learning is an option where we could reuse an existing architecture that has already been trained. Transfer Learning is the kind of practice where we use pre-trained models in addition to a couple of layers.

The pre-trained models are models that already exist and are appropriate for similar topics, image classification in our case. We do not create a model from scratch but we use them as a starting point. The pre-trained models are trained on very large-scale image classification problems and due to the huge number of images, they tend to have very good fit. The past few years we have seen many architectures being developed that have made tremendous progress in the field of image classification. In this project we use VGG-16, VGG-19 and ResNet50 pre-trained models. VGG-16 and VGG-19 with 16 and 19 layers, respectively, are the most used architectures. The difference between VGG-16 and VGG-19 is that each number stands for the number of weight layers in the network.

✓ **Processing the Data**

In order to apply the models, we have to process the data to be compatible with these models. More specifically, we reshape the images into 128×128×3 for both train and valid samples. Finally, due to memory issues we ended with 10,000 images for training, 1,000 for validation for 24 classes as we have predefined in the basic pre-process.

### 3.5.1. Visual Geometry Group – 16 (VGG-16)

The VGG-16 model is one of the most popular pre-trained models for image classification. The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. The model is sequential in nature and uses lots of filters. At each stage, small 3 × 3 filters are used to reduce the number of parameters all the hidden layers use the ReLU activation function. Even then, the number of parameters is 138 billion – which makes it a slower and much larger model to train than others.

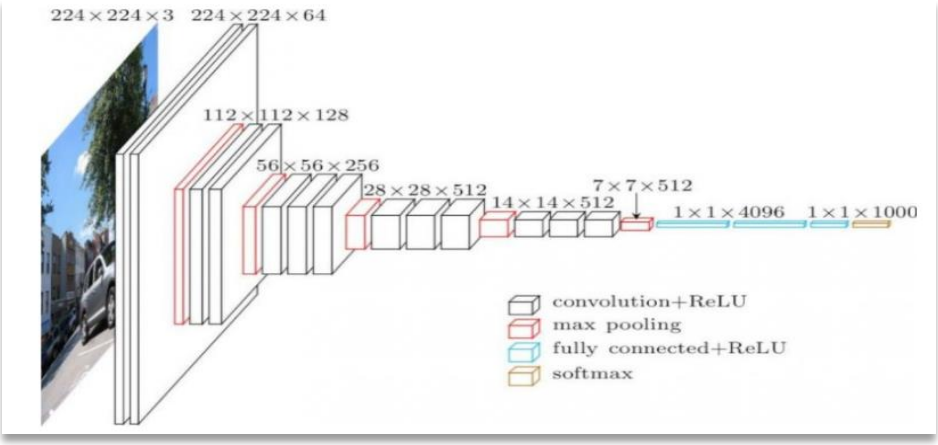



Figure 25. *VGG-16 visualization*

In order to compile the model, we use for loss function the Mean Absolute Error, Adam optimizer, accuracy for metric. For model fitting we use 30 epochs, 128 batch size, verbose equal to 1 and callbacks. Finally, our model produces an accuracy of **20.41**% on the train dataset, while in validation dataset is about **18**%. The results are not those that we expected by a model like this, which is a simple one. In Figure 26, we could see the model architecture of VGG-16 pre-trained model.

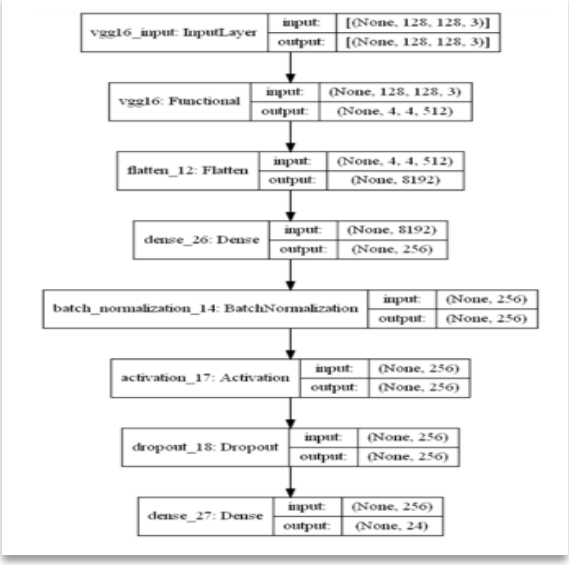



Figure 26. *Architecture of VGG-16 pre-trained model*

### 3.5.2. Visual Geometry Group - 19 (VGG-19)

VGG-19 is a 19-layer (16 convolution layers, 3 fully-connected) CNN that strictly used 3×3 filters with stride and pad of 1, along with 2×2 max-pooling layers with stride 2. To reduce the number of parameters in such deep networks it uses small 3×3 filters in all convolutional layers. The VGG-19 is trained on more than a million images and can classify images into 1000 object categories.

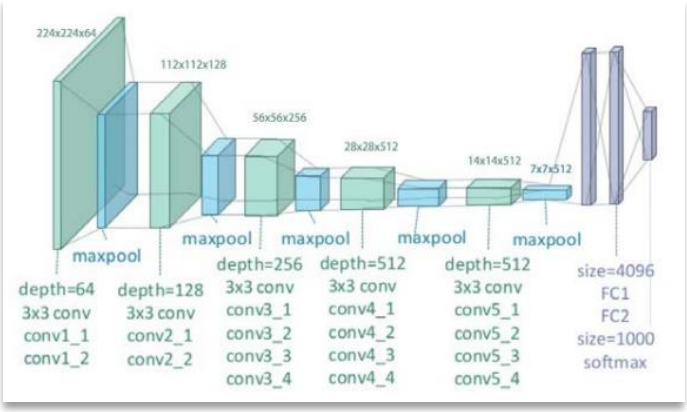



Figure 27. *VGG-19 visualization*

In order to compile the model, we use as previous for loss function the Mean Absolute Error, Adam optimizer and accuracy for metric. For model fitting we use 30 epochs, 128 batch size, verbose equal to 1 and callbacks. Finally, this model achieved **18.58**% accuracy in the train dataset and **18.9**% in validation. The architecture of VGG-19 pre-trained model is the same of VGG-16 in Figure.

The results are about the same in both VGG models. Their predictive ability is not as good as we expected, so in our case, these models are not appropriate for the image classification.

### 3.5.3. Residual Network 50 (ResNet50)

The ResNet50 is a 50 layer of Residual Network. It is a classic neural network used for many vision tasks. Like VGG the size of convolutional layers is 3×3 filters, the input size of this model is fixed as 224×224. and they follow some simple designs such as: For the layers having the same number of filters has the same output. The number of filters is doubled if the convolved output size is halved such that the time complexity per layer is preserved. The model ends with an average pooling layer and a 1000-way fully-connected layer with Softmax. In comparison with VGG, ResNet50 is deeper but the size of the model is smaller because a global average pooling operation is used instead of full-dense layers. *Global Average Pooling* is a pooling operation designed to replace fully connected layers in classical CNNs. Figure 28 show the configuration layers of ResNet50 network.

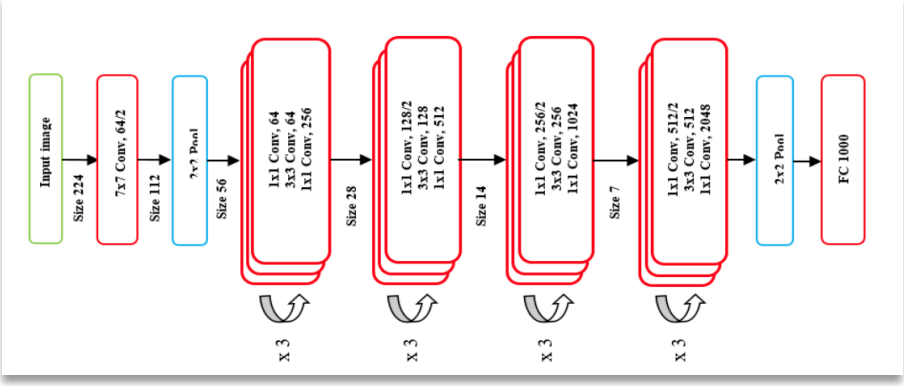



Figure 28. *ResNet50 visualization*

To compile the ResNet50 model we use MAE loss function and accuracy metric. Also, we choose Adam optimizer with learning rate equal to 0.001. For the fitting of the model 30 epochs, 128 batch size and callbacks are used. Finally, ResNet50 model achieved **19.08**% accuracy in train set and **16.80**% in the validation set.

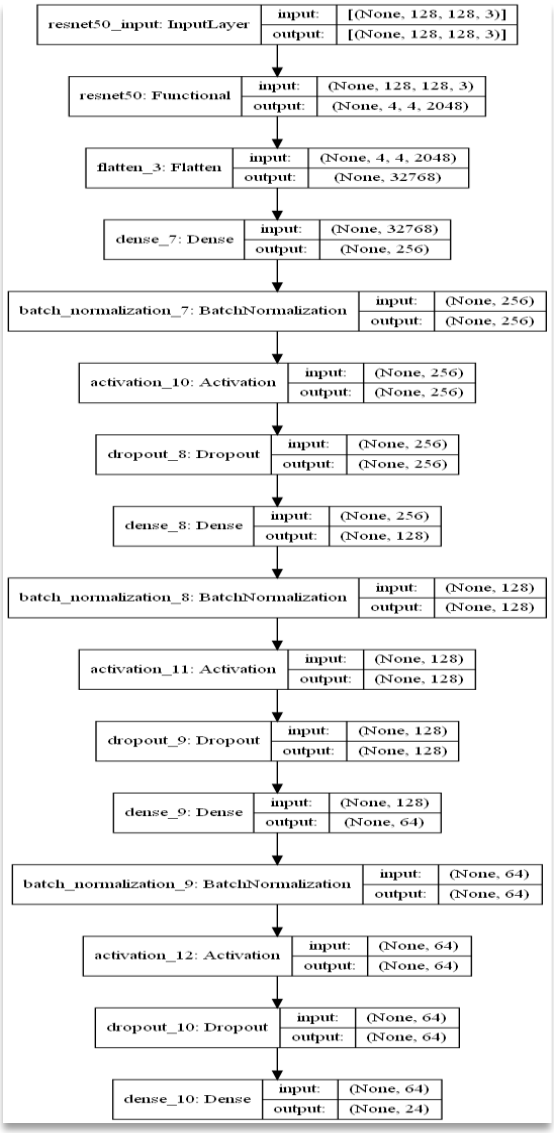



Figure 29. *Architecture of ResNet50 pre-trained model*

These models had been used in order to see how well they fit to our images, since they are very well known for their results in image classification. However, they produce very small accuracy which means that the predictive ability is not suitable for the image classification of the handwriting names that we have.

# 4. Conclusion

The aim of this project was to identify the best image recognition technology in order to predict the handwritten text that an image contains examining simultaneously a variety of deep learning techniques. To get insights about the solution that we were seeking as a company, we downloaded data from the popular Kaggle platform that contained csv files and images with handwritten names. Of course, the approach that we found as the best for our business purpose, can be applicable on handwritten images that contain just words (or a sequence of letters) which also consist of alphabet letters.

During the project, we faced some difficulties. To be more precise, one of the major problems that we encountered during this assignment was related with the number of images which was very large for the already split sets. We could only get a specific number of each set (train, validation, test) as we described on a previous chapter. Another crucial problem we faced in terms of time performance was the long duration (almost 9 hours) of fitting the CRNN model on which the callbacks were not included for early stopping of the training process. As a result, we had to wait until the end of this procedure in order to complete also other methods.

At first, we had to deal with missing value and unreadable content in the csv files and after several adjustments (converting the labels on identity column into upper case letters), the datasets were ready for the initial analysis. After the reshaping of the arrays properly for each method separately, we started our examination with a baseline model which was the Feedforward Neural Network. This model had low prediction ability based on the test set, so we decided to proceed with another model which was the Convolution Neural Network that contained more layers. On this model, we got higher prediction percentage although this percentage was not the expected one. Moreover, we applied Data Augmentation technique on the Convolutional Neural Network which improved slightly our results. At this point, after thorough examination, we found out that the most indicated technique for handwritten images is the Convolutional Recurrent Neural network which includes the combination of the Long Short-Term Memory architecture and CTC loss function that gave us the highest prediction percentage almost 90%. We also used some pre-trained models (VGG-16, VGG-19, ResNet50) that we discovered during our research and we implemented them for educational purposes.

Furthermore, there were many positive aspects to this project. First, the literature and the help that can be found for the subject in the web and bibliography is rather impressive. In addition,

the cooperation between the members of our team was very efficient and harmonic. The entire implementation of this project was quite demanding and at the same time fascinating. Through its process the members of our team learned a lot of important aspects of the Deep Learning field and its challenges. It was an end-to-end process which made us realize the struggle and the aspects of an entire Machine/Deep Learning project, which is a very educating experience.

In conclusion, we could say that we could also re-adjust some hyperparameters and reconstruct the architecture of each model to better achieve our main goal. Finally, we hope that the present project of our work will help others to improve the handwritten recognition method they use as there is always room for improvement in the field of computer vision.

## 4.1. Members/Roles

The team contributing to this project was consisted of three members, namely Evangelia Arseniou, Paraskevi Barmpa and Sotiria Ligkou. All the members are students of the MSc in Business Analytics of Athens University of Economics and Business. Statistics degree was the first degree of all our members, so it was very unprecedented and in the same time fascinating for us the whole procedure as we did not have a technical background. As far as the role of each member is concerned, each role cannot be well-defined as all of us dealt with this project touching all the chapters in the same manner.

## 4.2. Time Plan

| Tasks | 26/7-1/8 | 2/8-8/8 | 9/8-15/8 | 16/8-22/8 | 23/8-29/8 |
|---|---|---|---|---|---|
| Web & Bibliography Research | ✓ | | | | |
| Finding Dataset | ✓ | | | | |
| Finding Methodologies | ✓ | ✓ | | | |
| Building the Models | | ✓ | ✓ | | |
| Evaluating the Models | | | | ✓ | |
| Report | | | | | ✓ |

EBS Solutions

## 4.3. Bibliography


- ✓ Akshay Kulkarni & Adarsha Shivananda, published: 2019, Natural Language Processing Recipes, Unlocking Text Data with Machine Learning and Deep Learning using Python, doi: 10.1007/978-1-4842-4267-4
- ✓ Mirza Rahim Baig et al., Published: 2020, THE DEEP LEARNING WORKSHOP, Learn the skills you need to develop your own next-generation deep learning models with TensorFlow and Keras
- ✓ Navin Kumar Manaswi, published: 2018, Deep Learning with Applications Using Python, Chatbots and Face, Object, and Speech Recognition with TensorFlow and Keras, doi: https://doi.org/10.1007/978-1-4842-3516-4
- ✓ Yuxi (Hayden) Liu & Saransh Mehta, published: 2019, Hands-On Deep Learning Architectures with Python, Create deep neural networks to solve computational problems using TensorFlow and Keras
- ✓ https://builtin.com/artificial-intelligence
- ✓ https://serokell.io/blog/ai-ml-dl-difference
- ✓ https://softengi.com/blog/web-development/object-character-recognition-to-digitize-your-business/
- ✓ https://readcoop.eu/insights/ocr-vs-htr/
- ✓ https://analyticsindiamag.com/why-jupyter-notebooks-are-so-popular-among-data-scientists/
- ✓ https://labelbox.com/image-annotation-overview
- ✓ https://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm
- ✓ https://blog.roboflow.com/train-test-split/
- ✓ https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7
- ✓ https://www.ibm.com/cloud/learn/neural-networks
- ✓ https://www.upgrad.com/blog/neural-network-tutorial-step-by-step-guide-for-beginners/
- ✓ https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network
- ✓ https://deepai.org/machine-learning-glossary-and-terms/activation-function
- ✓ https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network
- ✓ https://deepai.org/machine-learning-glossary-and-terms/rmsprop

- ✓ https://deepai.org/machine-learning-glossary-and-terms/loss-function
- ✓ https://deepai.org/machine-learning-glossary-and-terms/evaluation-metrics
- ✓ https://medium.com/@ewuramaminka/mean-absolute-error-mae-machine-learning-ml-b9b4afc63077
- ✓ https://www.tutorialspoint.com/keras/keras_dense_layer.htm
- ✓ https://radiopaedia.org/articles/epoch-machine-learning
- ✓ https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/
- ✓ https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
- ✓ https://www.geeksforgeeks.org/keras-conv2d-class/
- ✓ https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab
- ✓ https://deeplizard.com/learn/video/qSTv_m-KFk0
- ✓ https://faroit.com/keras-docs/2.0.6/initializers/
- ✓ https://deepai.org/machine-learning-glossary-and-terms/adam-machine-learning
- ✓ https://www.mdpi.com/2072-4292/9/3/298
- ✓ https://www.analyticsvidhya.com/blog/2020/11/a-short-intuitive-explanation-of-convolutional-recurrent-neural-networks/
- ✓ https://towardsdatascience.com/handwriting-to-text-conversion-using-time-distributed-cnn-and-lstm-with-ctc-loss-function-a784dccc8ec3
- ✓ https://www.mathsisfun.com/definitions/horizontal-flip.html
- ✓ https://www.charterglobal.com/image-data-augmentation-and-ai/
- ✓ https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
- ✓ https://theailearner.com/2019/05/29/creating-a-crnn-model-to-recognize-text-in-an-image-part-2/
- ✓ https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html
- ✓ https://paperswithcode.com/method/global-average-pooling