MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

SUMMER PRACTICE REPORT

CENG 400

STUDENT NAME : Onur ŞİMŞEK

ORGANIZATION NAME : Spark Calibration

ADDRESS : ODTÜ Teknokent,Silikon Blok No:34 Çankaya /ANKARA

START DATE : 6 July 2020

END DATE : 7 August 2020

TOTAL WORKING DAYS : 20

STUDENT'S SIGNATURE **ORGANIZATION APPROVAL**

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION

This report document includes what I have done during my internship at Spark Calibration which is located in METU Technopolis including the difficulties that I faced and how I solved them. During my internship, software engineer Uğurcan AKYÜZ was my mentor. In this report, you can find what my project is about and the technical details of my project.

My internship was in between 06.07.2020 and 07.08.2020. There was a holiday in between these days, and after the holiday we continued our project.This was not my first internship experience,so my aim for this internship was to have some knowledge and experience with concepts such as deep learning, computer vision.

In Spark Calibration, I have responsibilities in a project of the company in which I try to recognize the information which is written on the screen of a multimeter by using deep learning. All of the internship has been done online due to conditions of pandemic.However, I had some meetings with my mentor in a week to discuss what I have done and what I will be doing during the following week.

2. INFORMATION ABOUT PROJECT

My project is called the Text Recognition Application. It is a project where we are trying to recognize or read the information on the multimeter.

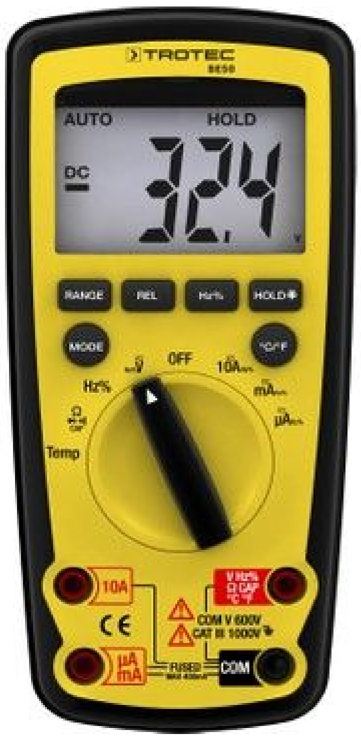


Figure 1: The multimeter and information on the multimeter

As you can from Figure 1, a multimeter has information such as the value on the screen and postfix which is indicated AC in the above figure.

In my internship, first I need to understand how this can be done by using deep learning.So, I finished two courses in coursera platform which were given by Laurence Moroney. The first course was *Introduction to TensorFlow for Artificial Intelligence,Machine Learning, and Deep Learning*. The second was *Convolutional Neural Networks in TensorFlow* .

After finishing the courses, I started the project of the company. We take a project as an example for our project. The project is called SimpleHTR. My mentor wanted me to adapt the images and labels that we have to this project so that we can start training on this model.

2.1 ANALYSIS PHASE

While I was creating the Text Recognition Application, I have used many tools and libraries.The most important ones are Visual Studio, Google Colab, Jupyter

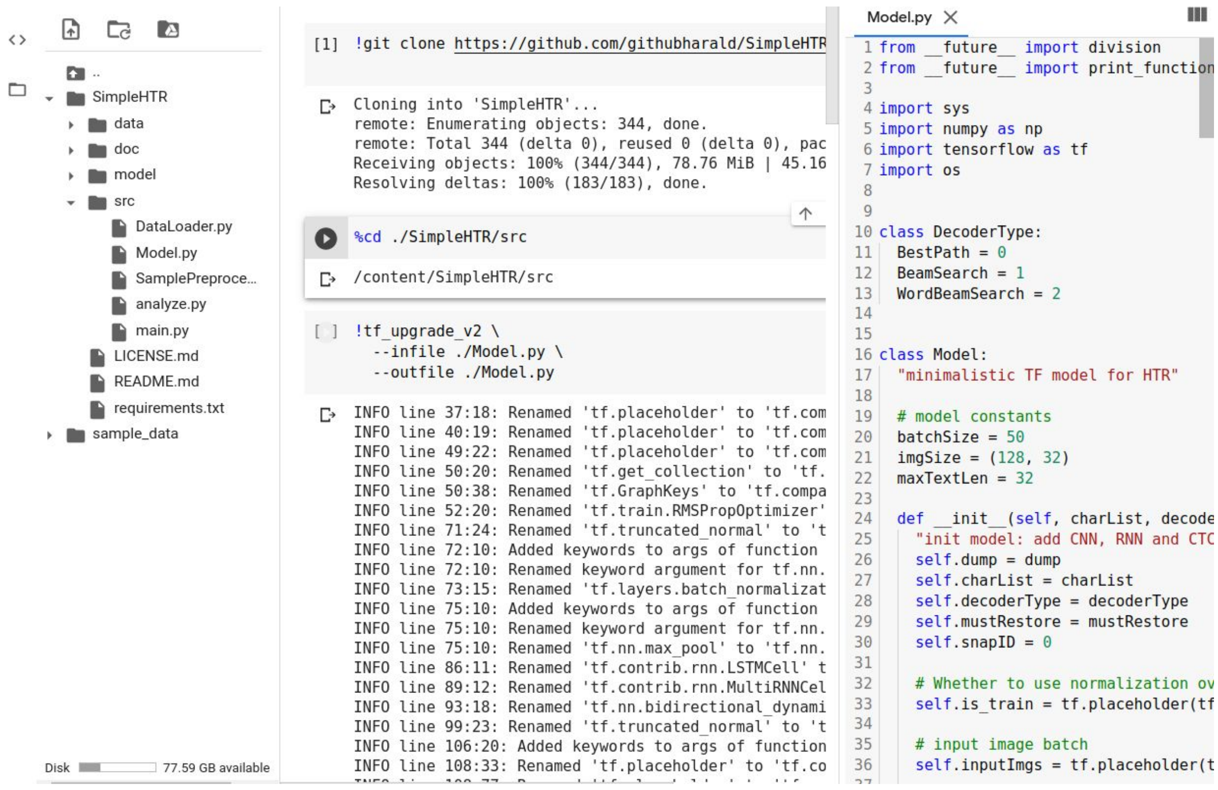
Notebook, numpy, os, openCV, matplotlib and tensorflow. Here, you can find brief information about these tools and how I have used them.

2.1.1 Visual Studio

I used Visual Studio for the basic Python codes of my project that doesn't require any complex library. I used the terminal of Visual Studio to test these codes.

2.1.2 Google Colab

Google Colab is an environment in which you can put your Python codes, jupyter notebook files, images, .txt files. Also, it enables you to upload or remove the files and edit the files that you upload. You can execute your codes, you can write Linux commands in this environment. The most important feature is if I didn't have Google Colab, I would have to do all of the installation to my computer which will certainly take a lot of time and effort.



```
[1]  !git clone https://github.com/githubharald/SimpleHTR

     Cloning into 'SimpleHTR'...
     remote: Enumerating objects: 344, done.
     remote: Total 344 (delta 0), reused 0 (delta 0), pac
     Receiving objects: 100% (344/344), 78.76 MiB | 45.16
     Resolving deltas: 100% (183/183), done.

     %cd ./SimpleHTR/src

     /content/SimpleHTR/src

[ ]  !tf_upgrade_v2 \
       --infile ./Model.py \
       --outfile ./Model.py

     INFO line 37:18: Renamed 'tf.placeholder' to 'tf.com
     INFO line 40:19: Renamed 'tf.placeholder' to 'tf.com
     INFO line 49:22: Renamed 'tf.placeholder' to 'tf.com
     INFO line 50:20: Renamed 'tf.get_collection' to 'tf.
     INFO line 50:38: Renamed 'tf.GraphKeys' to 'tf.compa
     INFO line 52:20: Renamed 'tf.train.RMSPropOptimizer'
     INFO line 71:24: Renamed 'tf.truncated_normal' to 't
     INFO line 72:10: Added keywords to args of function
     INFO line 72:10: Renamed keyword argument for tf.nn.
     INFO line 73:15: Renamed 'tf.layers.batch_normalizat
     INFO line 75:10: Added keywords to args of function
     INFO line 75:10: Renamed keyword argument for tf.nn.
     INFO line 75:10: Renamed 'tf.nn.max_pool' to 'tf.nn.
     INFO line 86:11: Renamed 'tf.contrib.rnn.LSTMCell' t
     INFO line 89:12: Renamed 'tf.contrib.rnn.MultiRNNCel
     INFO line 93:18: Renamed 'tf.nn.bidirectional_dynami
     INFO line 99:23: Renamed 'tf.truncated_normal' to 't
     INFO line 106:20: Added keywords to args of function
     INFO line 108:33: Renamed 'tf.placeholder' to 'tf.co
```

```python
from __future__ import division
from __future__ import print_function

import sys
import numpy as np
import tensorflow as tf
import os


class DecoderType:
  BestPath = 0
  BeamSearch = 1
  WordBeamSearch = 2


class Model:
  "minimalistic TF model for HTR"

  # model constants
  batchSize = 50
  imgSize = (128, 32)
  maxTextLen = 32

  def __init__(self, charList, decode
    "init model: add CNN, RNN and CTC
    self.dump = dump
    self.charList = charList
    self.decoderType = decoderType
    self.mustRestore = mustRestore
    self.snapID = 0

    # Whether to use normalization ov
    self.is_train = tf.placeholder(tf

    # input image batch
    self.inputImgs = tf.placeholder(t
```

Figure 2 : Google Colab interface

2.1.3 Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create documents that contain codes, equations and narrative text. I had experience with Jupyter Notebook while I'm doing programming assignments of the courses. Also, I had some knowledge since I have used it while I'm doing signal processing programming assignment of *Signals And Systems* course which is given in our department .

2.1.4 Numpy

Numpy is the library for the Python programming language to do calculation for multi dimensional arrays and matrices. In my project, I have recognized that having for loops in our code will increase the time it takes in order for our code to execute. Rather than having explicit for loops, when we use numpy's built in functions, there is a significant time difference. In deep learning, when we are dealing with large numbers of data, this time difference might be very important. Therefore, for numerical operations Numpy is a highly optimized library.

2.1.5 Os

This Python library provides operating system dependent functionality. In my project, I had to read images, read the labels, process them and save them into a file. I use the OS library for these purposes.

2.1.6 OpenCV

OpenCV is Open Source Computer Vision library. As its name implies, it is widely used in computer vision and machine learning. In my project, I used openCV to perform image related operations such as reading the image or processing the image.

2.1.7 Matplotlib

Matplotlib is a library in Python that enables interactive visualizations. In my project I used it to show the images, plotting loss and cost functions that shows how well we are doing in our model in deep learning.

2.1.8 Tensorflow

Tensorflow is a platform that is used to build and deploy ML models. I used Tensorflow to create models, compile models, and to fit the data into the model. It is also important that since we are using convolutional neural networks, we are also adding a layer to accomplish convolution.

2.2 DESIGN PHASE

In our project, our aim is to recognize :

1. The prefix information as plus or minus
2. The value on the screen
3. The postfix information as unit such as AC

We need very high accuracy in our project since we want to check whether something has been done correctly or not. Therefore, my advisor wants me to use deep learning. To achieve the above three goals, we need to create a model and we need to train our model with some data. We could simply continue with this if we had enough data. However, we have only 129 images and labels for them. This number will not be sufficient for a model to be trained. We could apply some **Image Augmentation** techniques to our model and have more data, but even this will not be sufficient for us.Therefore, to achieve above goals my aim was to use somebody else’s model to train our model. This model is designed by Harald Scheidl.

Since we are using deep learning, all we need is data to train our model. We need data, however this data should be in correct form in order for training to start. Our data, or images and labels are not in the form that Harald’s model can understand. Doing my internship, I tried to perform some image processing, and read the .txt files in the correct sense that Harald’s project can work with.

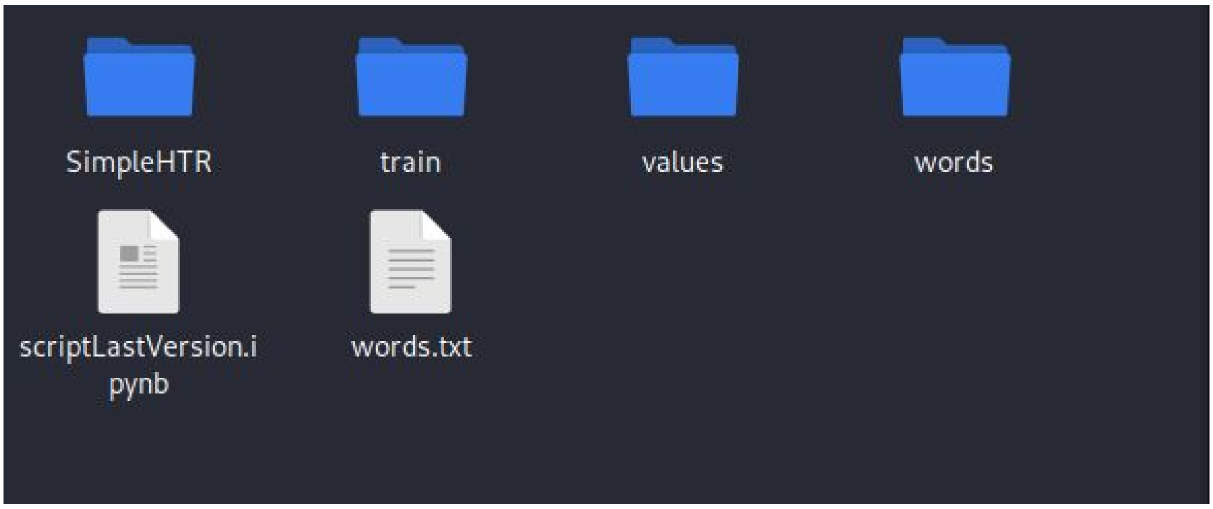


Figure 3: The content of the project

As it can be seen in Figure 2, there are four directories and two files in my project. I was mostly interested in the script which creates words.txt. The file words.txt is just for our example project to be used while starting training. A brief information of each file or directory can be found below:

SimpleHTR: This is the directory where our project is located in. It contains subdirectories for data, documentation, codes.
Train: All of our training data is in here. You can see an example image in the figure below. It is different from Figure 1, since some algorithms are applied before.

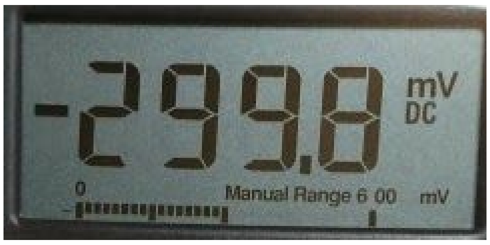


Figure 4: An example image from our training data

Values: Our labels are located in this directory. There are 129 .txt files each of which contains information about the image. One example of this can be :-;299.8;mVDC. This is the label for the below figure. As you can see, each information is splitted by ; character.

Words: This is the directory that is needed in order to run Harald's model. In order for that project to be runned, our training data must be located in this directory. Harald's training data was very large, about 600 MB in the form which is compressed. Therefore, it is reported that it will take 18 hours if we start training.

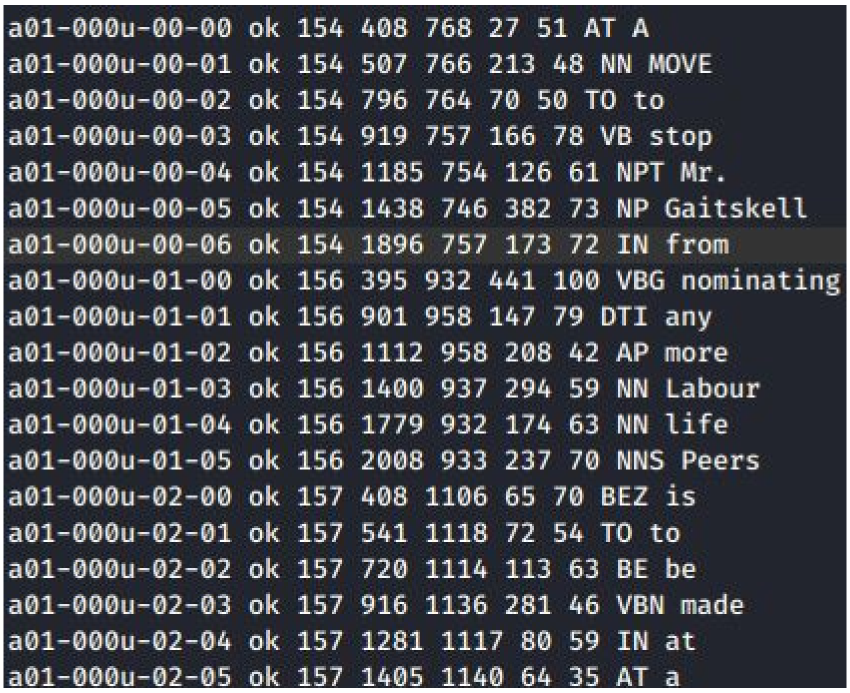
Words.txt: This file is generated when my script is executed. The file structure is very similar to the one which is in Harald's project.

ScriptLastVersion.ipynb: This jupyter notebook file is very important for us in order to process our data so that it can then work with Harald's model. With this picture, and with this images and labels structure, it is impossible for our project to start training. My advisor and I wanted to start training using Harald's model. Therefore, I should have created a script which converts our data into some good forms.So, in a step by step manner, this file should be able to :

- Process all of the images that we have in the train directory, make them represented by not three channels but two channels.
- Process all the labels in values directory.
- Read all the information.
- Combine all of the information that is read and write this information to just a single .txt file which is called as words.txt.
- Each line of the words.txt should be the image's information each of which is separated by a blank character.

The reason that we are doing above items is just because we want to process our data to start training.

I want to show how words.txt should be in order for our model to fit the data. In the below figure, you can see an example part of how Harald's labels are looking like :

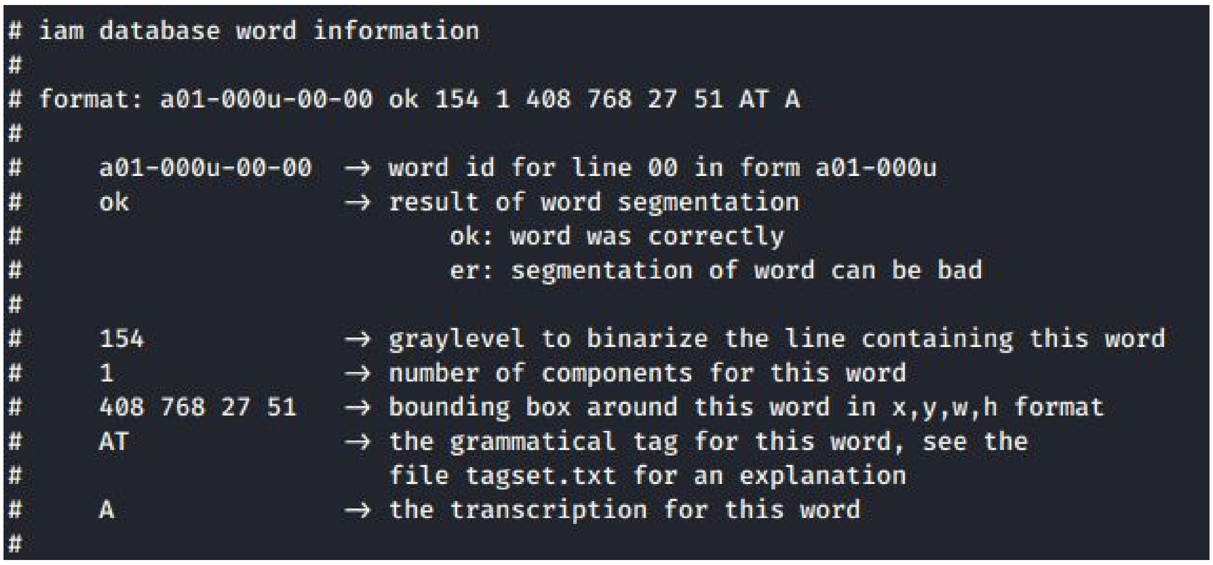


```
a01-000u-00-00 ok 154 408 768 27 51 AT A
a01-000u-00-01 ok 154 507 766 213 48 NN MOVE
a01-000u-00-02 ok 154 796 764 70 50 TO to
a01-000u-00-03 ok 154 919 757 166 78 VB stop
a01-000u-00-04 ok 154 1185 754 126 61 NPT Mr.
a01-000u-00-05 ok 154 1438 746 382 73 NP Gaitskell
a01-000u-00-06 ok 154 1896 757 173 72 IN from
a01-000u-01-00 ok 156 395 932 441 100 VBG nominating
a01-000u-01-01 ok 156 901 958 147 79 DTI any
a01-000u-01-02 ok 156 1112 958 208 42 AP more
a01-000u-01-03 ok 156 1400 937 294 59 NN Labour
a01-000u-01-04 ok 156 1779 932 174 63 NN life
a01-000u-01-05 ok 156 2008 933 237 70 NNS Peers
a01-000u-02-00 ok 157 408 1106 65 70 BEZ is
a01-000u-02-01 ok 157 541 1118 72 54 TO to
a01-000u-02-02 ok 157 720 1114 113 63 BE be
a01-000u-02-03 ok 157 916 1136 281 46 VBN made
a01-000u-02-04 ok 157 1281 1117 80 59 IN at
a01-000u-02-05 ok 157 1405 1140 64 35 AT a
```

Figure 5: An example portion of words.txt to illustrate

If you look at Figure 4, the difference is very clear. Our data is in the form of separate .txt files each of which contains the information about images. In addition we don't have information like in Figure 4 such as image name. However, to train the model, our labels should be read in a way that we have exactly the same view as in Figure 4. We need to change our way of labeling the data without losing the data information.

If we have a look at just one single line to understand which information is for which purposes, we would have the following figure:



```
# iam database word information
#
# format: a01-000u-00-00 ok 154 1 408 768 27 51 AT A
#
#     a01-000u-00-00  → word id for line 00 in form a01-000u
#     ok              → result of word segmentation
#                           ok: word was correctly
#                           er: segmentation of word can be bad
#
#     154             → graylevel to binarize the line containing this word
#     1               → number of components for this word
#     408 768 27 51   → bounding box around this word in x,y,w,h format
#     AT              → the grammatical tag for this word, see the
#                       file tagset.txt for an explanation
#     A               → the transcription for this word
#
```

Figure 6: IAM dataset word information

The dataset that we are working with is IAM Dataset. So, my script, when it is runned, should create a file which contains lines each of which contains these information. A line in our .txt file looks like the following :

a01-000u-00-00 ok 154 1 408 768 27 51 AT A

Image Name: a01-000u-00-00 is the image name. a01 , 000u, 00 are the directory names, and our image will have a path like this.

Image Status: It is represented either ok or err. To achieve this, all of the images should be observed and the images with bad quality should be stored in an array. Their status will be err.

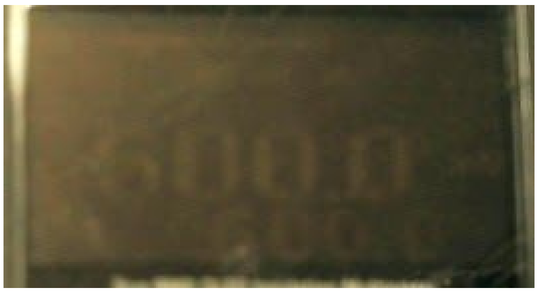


Figure 7: An example of an image whose status is err

Gray Level: We have chosen this as 154 for all of the images.

Number of Components: It is one for all of the images. It is only mentioned in the description. There is no information about it in the lines.

Bounding Box: Four numbers which are taken by using image shape.
Labels: Last two information should be the labels in the image. At most, we might have three labels; however there are two labels in each line. Therefore, I found a solution by combining the prefix information and values on the image.

Since we have these information in the IAM dataset, my script should be created in a way that it also has information like image name, bounding box, status of the image in addition to the labels. Remember that I have only labels in a very crowded form.
Creating a script and having a good .txt file which is very similar to IAM dataset format is not sufficient. The model has its own dataLoader in it. We have a different configuration, so we must also modify some files in the src subdirectory of SimpleHTR shown in Figure 2.

This being said, it is time to talk about the design of our project that we take as an example.

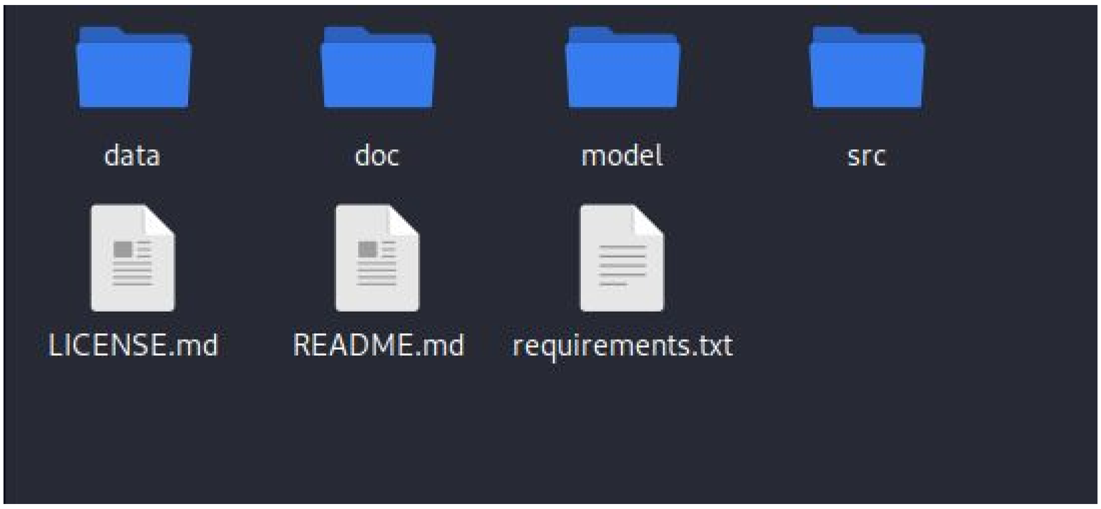


Figure 8: The structure of our sample project

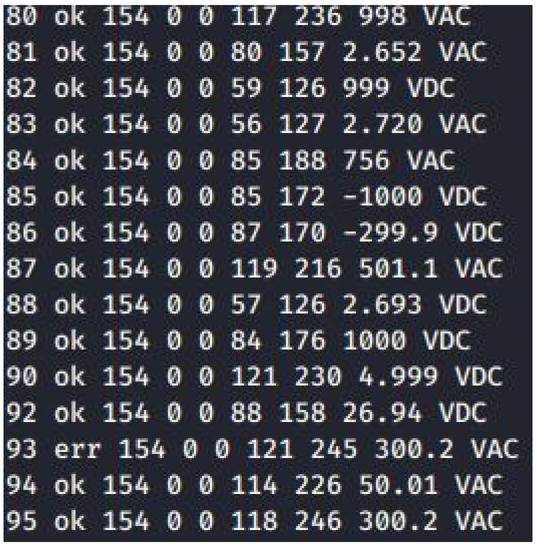
The *doc* directory in Figure 7, includes the images that are used in the documentation of our project. The *src* directory contains the source code of our project. We should mainly look at the *dataLoader.py* file in *src* directory.

There are also two main directories that we need to focus on. One of them is the *data* directory. This directory is important because it contains our training images under the directory called *words* and the file words.txt which can be created by running my script. There is also test.png which our model works with and try to recognize the text on the images. One of the most beautiful indicators of doing well with training of the model is the file *corpus.txt.* After I am able to start the training, the content of the corpus.txt changes with the labels of the images. So, this file could be a good checkpoint for us. The second of them is the *model* directory. This directory includes a .zip file. In order to train the model, we should unzip this file. There are *charList.txt* and *wordCharList.txt.* When training starts, the content of the files changes with the labels that I provide in words.txt. If we check charList.txt after training starts, it is as follows:

-.0123456789ACDHMVZkmz

If we look at this, they are simply all of the characters in our image dataset. However, when there was an error, and I was not able to start training, the content of this file is so different than our characters or the above line.

Lastly, I want to talk about what will happen when my script is runned and how created *words.txt* is similar to the IAM Dataset format.



```
80 ok 154 0 0 117 236 998 VAC
81 ok 154 0 0 80 157 2.652 VAC
82 ok 154 0 0 59 126 999 VDC
83 ok 154 0 0 56 127 2.720 VAC
84 ok 154 0 0 85 188 756 VAC
85 ok 154 0 0 85 172 -1000 VDC
86 ok 154 0 0 87 170 -299.9 VDC
87 ok 154 0 0 119 216 501.1 VAC
88 ok 154 0 0 57 126 2.693 VDC
89 ok 154 0 0 84 176 1000 VDC
90 ok 154 0 0 121 230 4.999 VDC
92 ok 154 0 0 88 158 26.94 VDC
93 err 154 0 0 121 245 300.2 VAC
94 ok 154 0 0 114 226 50.01 VAC
95 ok 154 0 0 118 246 300.2 VAC
```

Figure 9: An example part of words.txt

Desired script should create a big version of Figure 8. A couple things to notice:

- The image name is just like 80.jpg. I got rid of the .jpg extension.
- The order of the images was wrong even though I have used the built in *sorted* function. Therefore, I spent extra effort and created my own function to

sort these numbers. The problem was the fact that all of the data, that was coming, was in the form of strings and we can not sort the strings like the way that we expect. Therefore, there should be a conversion to the integer and after sorting operation, all of the sorted integers should be converted back to strings.

- In Figure 8, you can see that the image 93.jpg is the image with bad quality since its status is reported as err. The identification is done by me and the script is created to perform this identification.
- Some images like 81.jpg contain floating point values and it is crucial that the character '.' should be in our character list.
- If you look at the information about 86.jpg, you can see that some images have a prefix information as minus which is indicated by the character '-'. Actually, we were storing the prefix information as a separate information. However, after I saw errors related to it, I found a solution by combining the prefix information with the value information. Therefore, the last two information becomes related to our labels.

I perform the above items since our data is not like what we want. With these changes, our data becomes compatible with the IAM dataset. By saying *processing the data* such as images and labels, I refer to this.

Related to the image processing part, my advisor sent a function that converts the RGB image to Binary Image. Our image data was consisting of images with three channels; however they should be two channel images. The function works very well for some images; extracts the details on the images.However, for some of them I lost the postfix information when I use this method. Therefore, I found another way of representing our images with two channels without having a loss in the image.In below, you will see how I lost the information on the image when I applied this function.

```
['', '299.9', 'VDC']
```

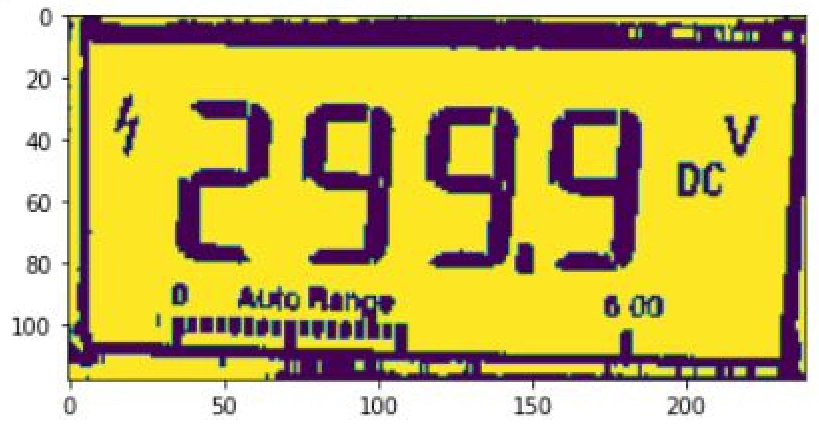


Figure 10: Example where function is very successful

```
['', '300.4', 'mVAC']
```

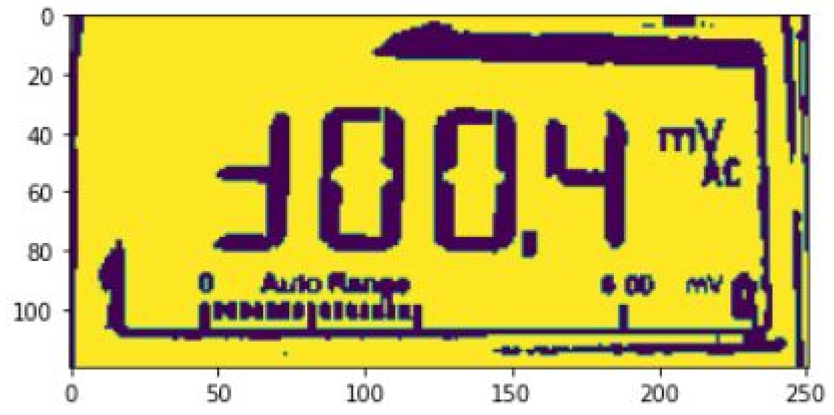
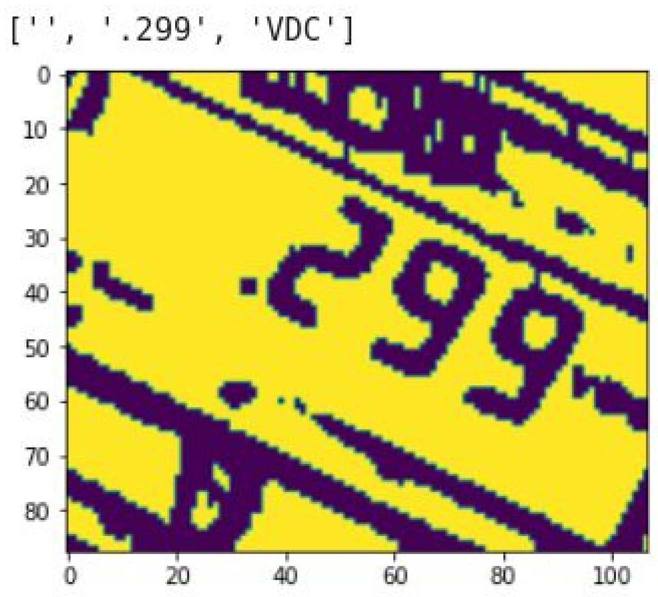


Figure 11: The value information is lost



```
['', '.299', 'VDC']
```

Figure 12: The postfix information is lost

Despite these images, I can say that the function works well for most of the images without having any information lost. We can use it when we consider the fact that its benefit is higher than its side effect.

2.3 IMPLEMENTATION PHASE

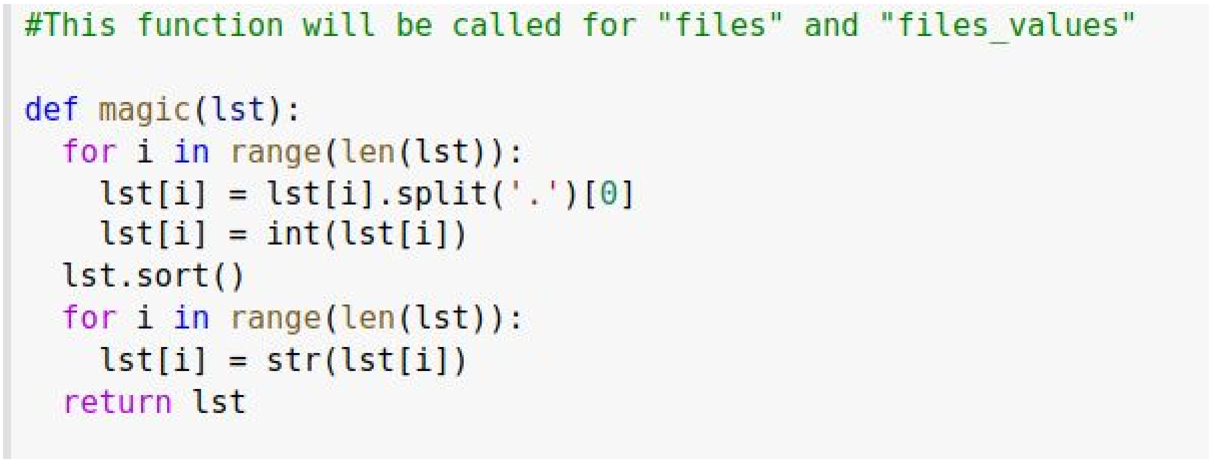
Since my internship is about deep learning, I want to express a few important concepts that I have learned in the courses in order for later usage. These concepts are crucial key factors that we should apply to our project.

- **Image Augmentation:** When we don't have enough data, we apply image augmentation to our dataset. This will enlarge our dataset. For example, with the help of image augmentation, we can shift the figure on the image, or we can rotate the image by some range.
- **Callback:** In deep learning when we have a lot of data, training of the model can take so much time. Therefore, we might end up in a situation where we wait so much time even though we have reached the desired accuracy. Callback is just for this purpose. When a desired accuracy is reached, with the help of callback, we stop training our model.
- **Overfitting:** Sometimes there might be the case that we are doing very well with training data, but when it sees new data that it has not seen before, the results are not as we expected. This is called overfitting. Overfitting is most likely to happen if we have less data, so to avoid overfitting, we should use more data.

- **Transfer Learning:** Learning of a new task is done by using previously learned tasks. For example, if somebody already trained his/her model by his/her data, then we can use this already trained data in addition to our data. We only apply training to **our data**. By doing this, we have more data, the training time is decreased and the accuracy of our model is higher.
- **Broadcasting:** What do you think will happen if we want to add two matrices whose dimensions are 3x4 and 1x4. In mathematically speaking, this operation can not be performed since the matrix dimensions should be the same. However, in Python this operation is possible with the help of broadcasting.
- **Convolution:** In convolution, we are applying a filter to our image. This filter, extracts some of the features in our image. The same model, with the same data will give a different accuracy result if we use convolution. In this case, training of the model is done with high accuracy. However, since we have a lot of images and we are applying our filter to all of the images, training time will increase.

After this brief introduction, it is time to move on how I implement the processing of the images and labels. To start the implementation, I want to say that we need to create two files which are *train* and *values* under the *tmp* directory of *Google Colab files*. The output of the script which is *words.txt* will be created under the directory *content.*

The first part that I want to talk about is ordering the file name part. Remember that our file names are just like *24.jpg* or *24.txt.* The below figure will explain how I got rid of the extensions and order to number since we need a set of ordered numbers.

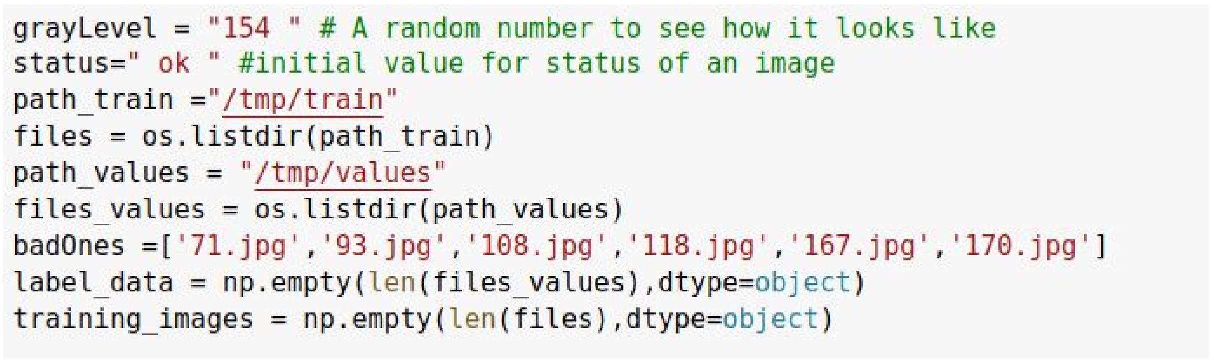


```python
#This function will be called for "files" and "files_values"

def magic(lst):
  for i in range(len(lst)):
    lst[i] = lst[i].split('.')[0]
    lst[i] = int(lst[i])
  lst.sort()
  for i in range(len(lst)):
    lst[i] = str(lst[i])
  return lst
```

Figure 13: How did I sort numbers which are of string data type?

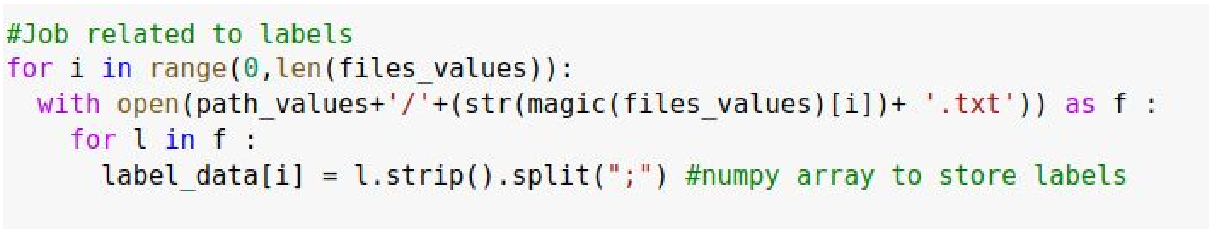
As you can see from Figure 13, the information is coming as strings and I want to sort the numbers. For this reason, I convert them to integers, I order them then return a list whose elements are just string.



```
grayLevel = "154 " # A random number to see how it looks like
status=" ok " #initial value for status of an image
path_train ="/tmp/train"
files = os.listdir(path_train)
path_values = "/tmp/values"
files_values = os.listdir(path_values)
badOnes =['71.jpg','93.jpg','108.jpg','118.jpg','167.jpg','170.jpg']
label_data = np.empty(len(files_values),dtype=object)
training_images = np.empty(len(files),dtype=object)
```

Figure 14: Initializations and creating numpy arrays

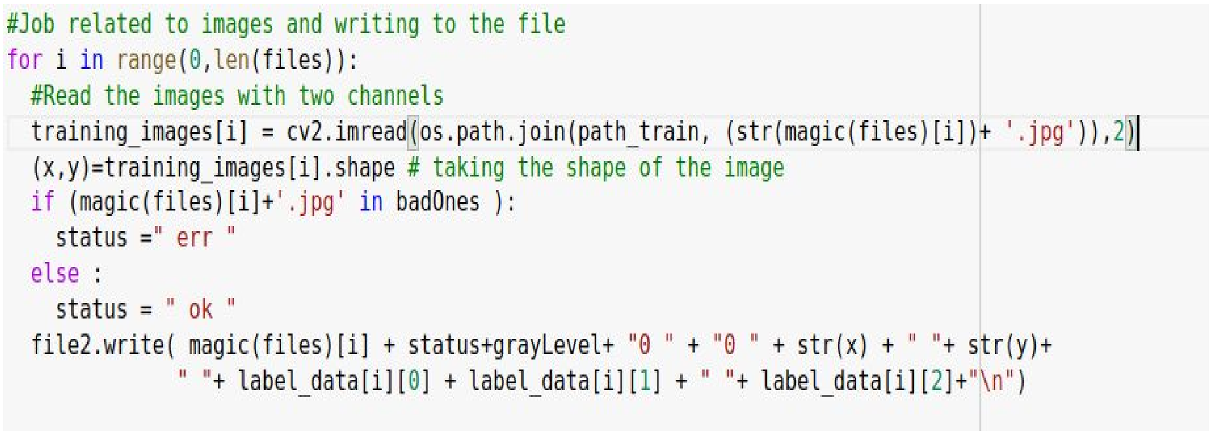
Figure 14 shows how I can perform the image status. The items which are in the *badOnes* list are the images with bad quality. It is bad in the sense that either it is very difficult to recognize the information on the image or the image is just an empty screen.



```
#Job related to labels
for i in range(0,len(files_values)):
  with open(path_values+'/'+(str(magic(files_values)[i])+ '.txt')) as f :
    for l in f :
      label_data[i] = l.strip().split(";") #numpy array to store labels
```

Figure 15: Storing labels in a numpy array

Figure 15 explains how I process the label information. Remember that our data was so messy that one could not use it to convert the IAM dataset format. Here, I'm opening the files one by one to read the information on them, and I split each line by character ';' since each information is separated by this character.

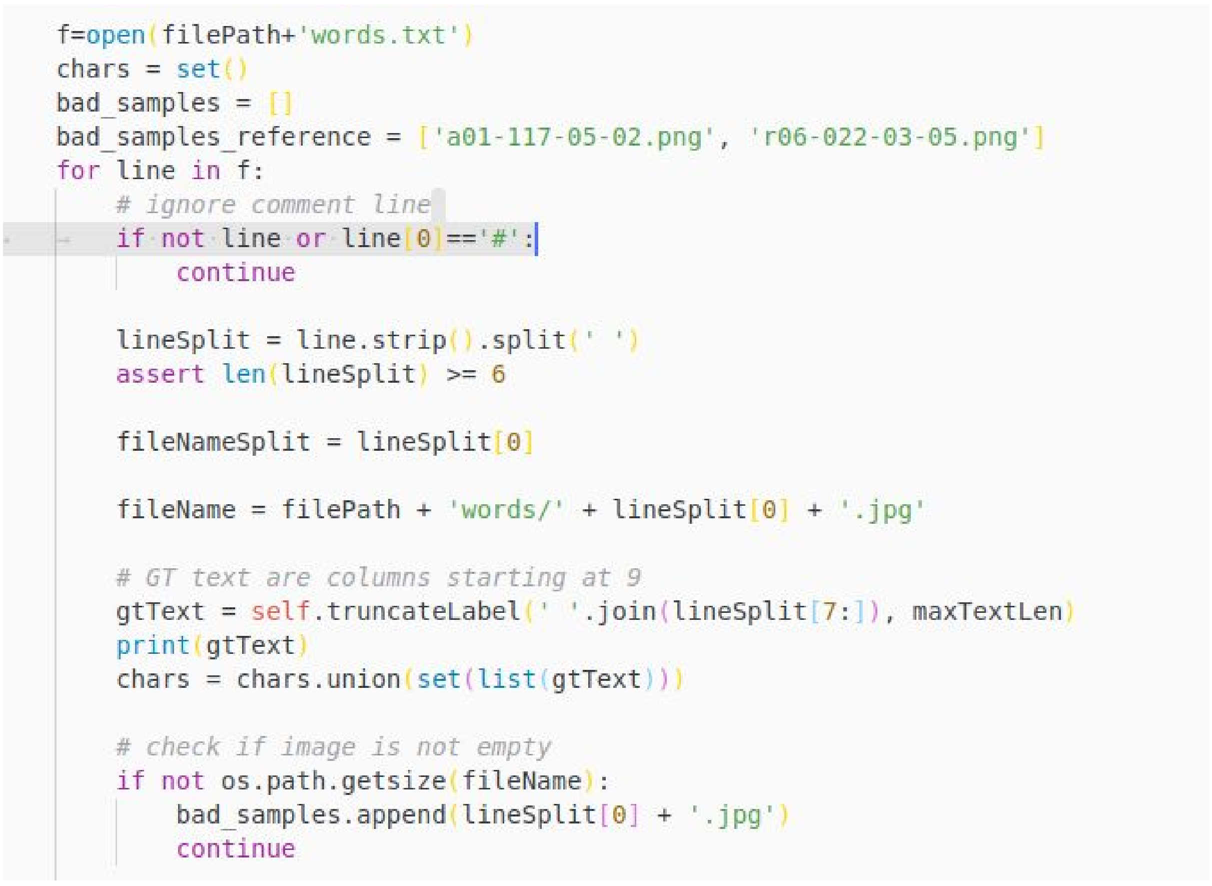


```
#Job related to images and writing to the file
for i in range(0,len(files)):
  #Read the images with two channels
  training_images[i] = cv2.imread(os.path.join(path_train, (str(magic(files)[i])+ '.jpg')),2)
  (x,y)=training_images[i].shape # taking the shape of the image
  if (magic(files)[i]+'.jpg' in badOnes ):
    status =" err "
  else :
    status = " ok "
  file2.write( magic(files)[i] + status+grayLevel+ "0 " + "0 " + str(x) + " "+ str(y)+
            " "+ label_data[i][0] + label_data[i][1] + " "+ label_data[i][2]+"\n")
```

Figure 16: Taking image related information and writing them to a file

The implementation details about how I read the images, obtain the related information and write all of the needed information to a file which is in the IAM dataset format can be seen in Figure 16. Here, I read the images as a grayscale image, so *image.shape* will return a tuple with **two items.** In the if statement, the identification of images is done based on image quality.

After this script, we have a words.txt that our example project can work with. Now, my job was to do the change on this example project so that we can start training our model. In the below figure you can see this change.



```
f=open(filePath+'words.txt')
chars = set()
bad_samples = []
bad_samples_reference = ['a01-117-05-02.png', 'r06-022-03-05.png']
for line in f:
    # ignore comment line
    if not line or line[0]=='#':
        continue

    lineSplit = line.strip().split(' ')
    assert len(lineSplit) >= 6

    fileNameSplit = lineSplit[0]

    fileName = filePath + 'words/' + lineSplit[0] + '.jpg'

    # GT text are columns starting at 9
    gtText = self.truncateLabel(' '.join(lineSplit[7:]), maxTextLen)
    print(gtText)
    chars = chars.union(set(list(gtText)))

    # check if image is not empty
    if not os.path.getsize(fileName):
        bad_samples.append(lineSplit[0] + '.jpg')
        continue
```

Figure 17: How the data in words.txt is loaded?

There are 129 lines in *words.txt* each of which starts with the image name. Therefore, if I split each line by blank character and take the first element I will get the image name. However, there is also other information such as image labels. *LineSplit* will also give us this information if we take starting from the seventh element. After these changes, I was able to start training, and the content of the file such as *charList.txt* changed accordingly.

2.4 TESTING PHASE

I was not responsible for creating any test for this project. The testing can be divided into two parts as the testing of the scripts and testing of the model. The second one can be done by using accuracy of our model in the validation data. On the other hand, if we want to check whether the scripts are working exactly as it should be, we could execute our jupyter notebook or Python files. If we have a similar words.txt which is looking like IAM Dataset, then our scripts are working perfectly. Remember that the aim of the scripts is to process data in a way that it looks like IAM Dataset.

In deep learning, in a situation where we are able to successfully start training of our model with our data, the measure of how we are doing with our model is the **accuracy result** that is coming from the validation set. Since the training data is seen by our model before, it is very important for a model to have a high accuracy in **validation data** which it has not seen before. To do these, it is a must for us to have a lot of data so that we could then use some of the data for training and some of them for validating. For example, in an even very simple case where we are trying to recognize numbers, our training data has 60000 images and validation data has 10000 images. From here, you can understand how important it is to have a large number of data, if we want to avoid something called overfitting.

3. ORGANIZATION

3.1 ORGANIZATION AND STRUCTURE

Since I conduct my internship as an online internship, I couldn’t have an idea about the company. However, from their websites, all of the information is simply accessible.

The company divides their work into two jobs namely Spark Measurement Technologies and Spark Calibration Services. You can find brief information about them in below from their websites:

As Spark Calibration Services, in the electronics industry RF and different devices; while continuing our mission to meet the maintenance, repair and calibration needs with a service, we maintain our vision of becoming the leading company in the sector with our understanding of high quality, fast service and customer-oriented service. We continue our R&D studies in cooperation with universities, and we aim to be an innovative company that makes a difference in domestic and foreign markets with the calibration software we have developed.

The history of the company goes back to the 1980's,Keysight Technologies (formerly Agilent Technologies), a world leader in the field of Electronic Testing and Measurement Devices, has been represented in Turkey since 1980s under different structures in order to meet the electronic test and measurement needs of the Turkish Electronics Industry.

In 1989, Hewlett Packard opened an office in Turkey and started providing services in an international structure.

In 1999, “Electronic Measurements Group” separated from Hewlett Packard and started operating worldwide and in Turkey as Agilent Technologies and now as Keysight Technologies.In 2001, Agilent Technologies decided to change the structure in Turkey and Spark Group of companies took over in order to serve the Turkish Electronic Industry with more flexible solutions.

The working areas of the company are :

- Defence industry
- Telecommunication
- Homeland Security
- Electronics Manufacturing
- Education and Research Institutions
- Broadcasting

Figure 18: Spark Calibration Laboratory

3.2 VISION AND GOALS

The quality policy of the company can be summarized as below items :

- Customer Focused Understanding
- Continuous Improvement in Service Quality
- Continuous Investment in Personnel Training and Laboratory Technologies
- Compliance with TS EN ISO / IEC 17025 Rules
- Adherence to Manufacturer Methods for Repair, Calibration and Adjustment Processes
- Commitment to Complying with Staff, Impartiality and Confidentiality Rules

In addition to this, vision and mission of the company :

Mission: The mission of Spark Group is to become the solution partner preferred by customers, by meeting the test and measurement instrument needs of Turkish Electronics Industry with the value added services it produces.

Vision: The vision of Spark Group is to become an indispensable solution partner in the Turkish Electronic Industry as regards Test and Measurement Systems and Solutions.

4. CONCLUSION

In conclusion, after having an internship at Spark Calibration and creating the Text Recognition Application, I have learned the technologies that I haven't known before. In addition to technologies, I had a chance to see how a computer engineer can work remotely without cut off communication.

During my internship , the times that I have difficulties were :

- The time when I try to order the name of images
- The time when I loss information on the images while converting them to binary image
- The time when I try to modify data loader of our project
- The time when I try to start the training
- The time when I want to test our model with some image

However, this was my first time having adventures with deep learning, computer vision and data processing. I felt that every information that I learned was something that I can use in my future career. Having finished two courses from Coursera, made me believe that I can do a lot more with these knowledge.

In addition to these, during my internship I recognized that Python is a very advanced tool for deep learning developers. It has a lot of built in functions in its library, and difficult jobs can be done with even one line of code. However, this doesn't mean that training a model or testing it with some validation set is very simple. It is very difficult because all of your data should be in some form which your model can fit into. Also, it might be the case that you have a very less number of data where overfitting is likely to happen while training. Therefore, you should perform data processing with which you can have data that your model can work with like what I have done during my internship.

5. REFERENCES

- About The Company.(n.d.). Retrieved from: http://www.sparkkalibrasyon.com.tr/kurumsal-1-hakkimizda.html

- Brief History.(n.d.). Retrieved from: https://www.sparkmeasure.com/eng/kurumsal-1-brief-history.html

- Working Areas.(n.d.). Retrieved from: https://www.sparkmeasure.com/eng/kurumsal-4-working-areas.html

- Quality Policy.(n.d.). Retrieved from: http://www.sparkkalibrasyon.com.tr/kurumsal-2-kalite-politikamiz.html

- Vision and Mission.(n.d.). Retrieved from: https://www.sparkmeasure.com/eng/kurumsal-3-our-vision-and-mission.html