# Sign Language Digits Recognition

João Vasconcelos, Nº Mec: 88808
*DETI*
*University of Aveiro*
j.vasconcelos99@ua.pt

Vasco Ramos, Nº Mec: 88931
*DETI*
*University of Aveiro*
Aveiro, Portugal
vascoalramos@ua.pt

## I. Introduction

This report has the goal of exposing and explaining our project developed for the course of Topics of Machine Learning which focuses on Sign Language Digits Recognition.

## II. Concepts

### A. Sign Language Digits

Hand gestures and sign language are the most commonly used methods by deaf and non-speaking people to communicate among themselves or with speech-able people. However, understanding sign language is not a universal skill.

For this reason, building a system that recognizes hand gestures and sign language can be very useful to facilitate the communication gap between speech-able and speech-impaired people.

In this research we are going to try to recognize digits from sign languages images.

### B. Data Set Overview

The data set we used is a modified version of the one available at *Sign Language Digits Dataset*. The one we are using is available at *Kaggle - Sign Language Digits Dataset* in the Data Sources section. The images are already normalized so we didn't have the necessity to normalize it.

This data set contains 2062 sign language digits images and, as it is known, digits range from 0 to 9, so, there are 10 unique signs.

Figure 1 shows two examples of the images present in the data set.

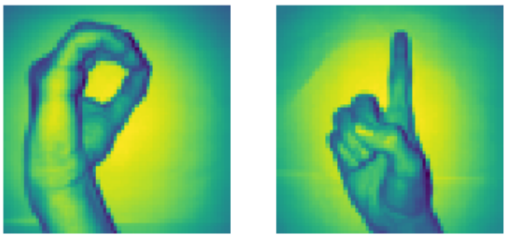

Fig. 1. Example of the data used in our model

### C. Models Used for Classification

In an initial approach, we used (Binary) Logistic Regression with just two classes (signs 0 and 1) and then used a Convolution Neural Network to recognize and classify the images considering all 10 classes (signs).

## III. Logistic Regression

### A. Classification Scripts

Logistic Regression was our first approach to the classification model. We decided to use just two of the classes so we can test the model accuracy and then decide what to do next.

We used 60% of the images of those two classes to train our model, 20% to evaluate the model with cross validation and the remaining 20% to test the model. We started with lambda as 0.1 and theta as [0, 0, ...] . To summarise:

TABLE I
Initial Configuration

| | |
|---|---|
| **Train Data** | 60% |
| **Cross-Validation Data** | 20% |
| **Test Data** | 20% |
| **Lambda** | 0.1 |
| **Theta** | [0, 0, ...] |

To train our model, we used the sigmoid, costFunctionReg and gradientDescent functions so that we could calculate the cost and gradient associated with the used thetas and lower the cost using the Gradient Descent algorithm.

The first step was to run this code with the previously defined initial values, after which we plotted the variation of the cost function using gradient descent, resulting in figure 2:

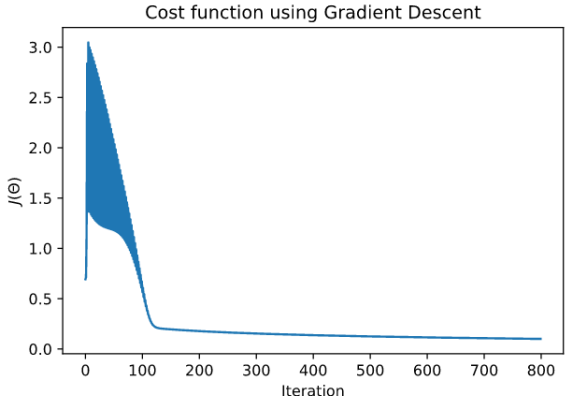


Fig. 2. Cost Function using Gradient Descent

In this representation, we can see that at first the gradient descent is not stable, but after 100 iterations it starts to

stabilize, which means the optimization of the parameters has improved.

After this, to check if the values we chose, mainly Lambda, were correct, we ran a cross-validation test with multiple lambdas so that we could decide what was the best fit to our problem. To do this, we used the classifierPredict and validationCurve functions. The obtained result was a diagram that showed us that the best lambda value was 0.01, as can be seen in figure 3.

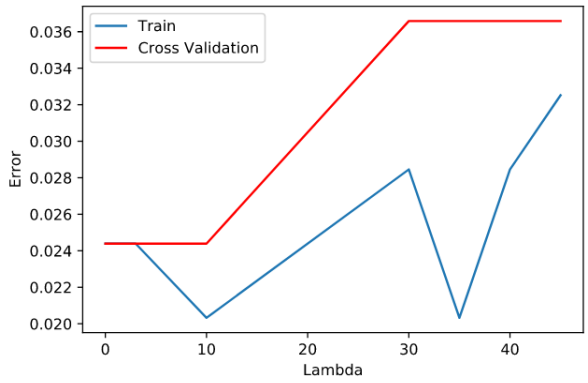


Fig. 3. Cross-Validation Test using Multiple Lambdas

The last step was to re-train our model using the optimized values we could obtain. In order to perform this, we combined the train and cross-validation data-sets as the new train data-set.

We re-run gradient descent to re-calculate the optimized theta values, which resulted in the graphic showed in figure 4 (which is a lot similar to the one showed in figure 2).

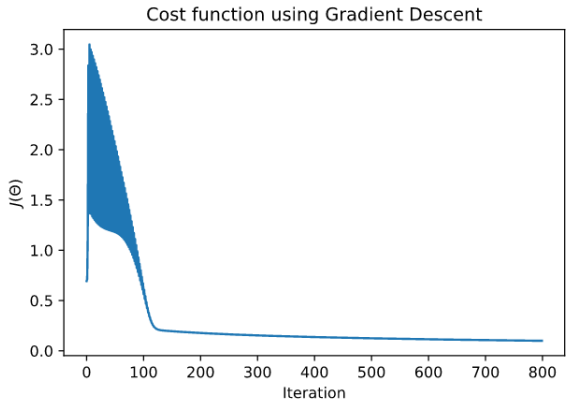


Fig. 4. Cost Function using Gradient Descent (Re-trained Model)

Since the output is similar to the previous results, we can deduce the optimizations were not too significant.

At last, we tested our model by predicting the results of our training and testing data-sets and compared those results to the real values. The values of error and accuracy were the following.

TABLE II
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.02743902 |
| **Test Error** | 0.03658536 |
| **Train Accuracy** | 97.3% |
| **Test Accuracyy** | 96.3% |

With these values we were able to understand that the model was not fully optimized and still missed some examples.

### *B. Scikit-Learn*

In our second approach, we applied the Sklearn library and its model interfaces to train a model to predict the previously chosen classes.

First, we tried to predict the data with a simple Logistic Regression model provided by Sklearn and obtained the following values:

TABLE III
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.0 |
| **Validation Error** | 0.02439024 |
| **Train Accuracy** | 100.0% |
| **Validation Accuracy** | 97.6% |

With just a simple model application, this model already performed better than the one we previously implemented. The next step was to use K-fold Cross Validation (also provided by Sklearn) to calculate the optimized values and train this optimized model with the amplified train data-set (train + cross-validation data-sets) and finally test it with the test data-set. This approach provided the following results:

TABLE IV
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.0 |
| **Test Error** | 0.0 |
| **Train Accuracy** | 100.0% |
| **Test Accuracy** | 100.0% |

We were satisfied with these results, however, as we couldn't get values or historical data to understand or discuss the internal optimizations of the Sklearn model, we decided to not further scrutinize this approach. Instead, we did some additional research in order to understand how we could benefit from other models and what the next steps should be.

Thus, given to the somewhat low success rate of our own model given just two classes and after doing some online research, we decided to move on to creating a model to solve this problem based on CNN as we are going to discuss and analyse in the next section.

## IV. Convolution Neural Network (CNN)

After using the Logistic Regression model, we decided to use the Convolution Neural Network (CNN) to try to lower the error found in the previous approach. With this said, we started by splitting the data like we did in the last approach (60% for training, 20% for cross-validation and 20% for testing).

To implement the CNN model we used the Keras library and built two functions to avoid code redundancy and confusion in the analysis of the work. The function show_model_history shows information about the loss and accuracy of a certain model and the function evaluate_conv_model trains the model with the provided data.

We started by building a simple model with one Convolution layer and got the following error and accuracy values:

TABLE V
Error and Accuracy Values

| | |
|---|---|
| **Train Error** | 0.054 |
| **Validation Error** | 0.203 |
| **Train Accuracy** | 94.5% |
| **Validation Accuracy** | 79.7% |

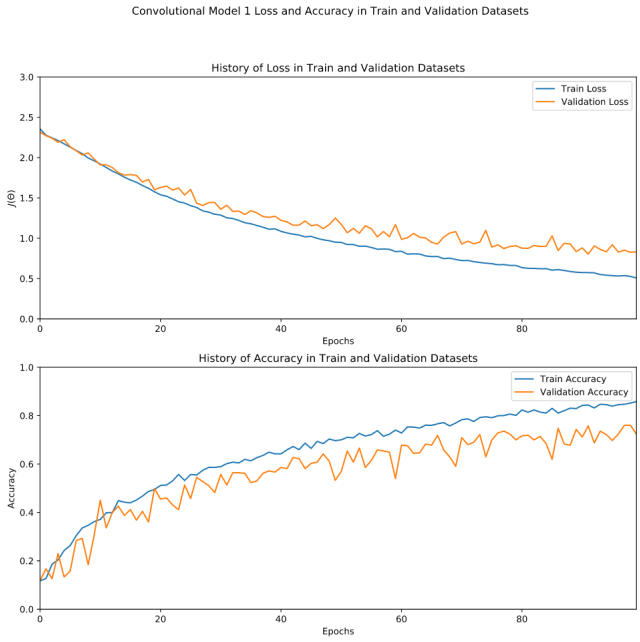


Fig. 5. Convolutional Model 1 Loss and Accuracy in Train and Validation Datasets

By analysing the previous graphs, we can clearly see that the model has a low training accuracy rate and a lower validation accuracy rate. Besides that, the fluctuating growth in the validation graph show that the robustness of the validation results is very low.

Considering the above conclusions, we tried to add a new Convolution layer to the model and got the following error and accuracy values:

TABLE VI
Error and Accuracy Values

| | |
|---|---|
| **Train Error** | 0.002 |
| **Validation Error** | 0.123 |
| **Train Accuracy** | 99.8% |
| **Validation Accuracy** | 87.7% |

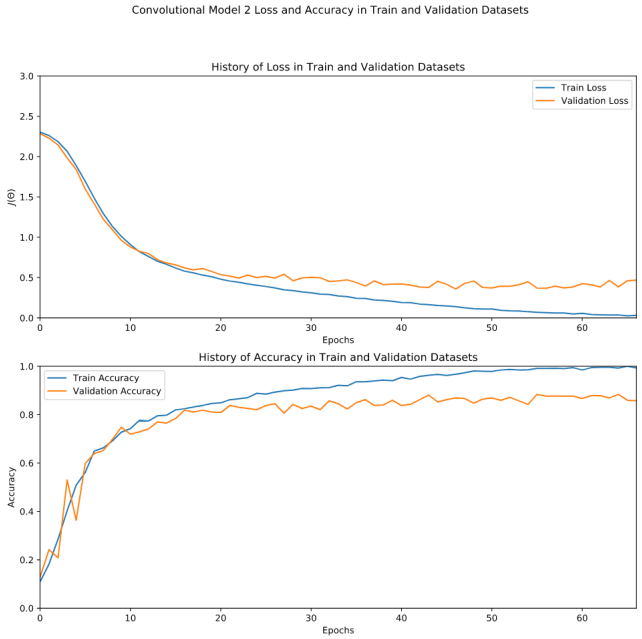


Fig. 6. Convolutional Model 2 Loss and Accuracy in Train and Validation Datasets

By analysing the previous graphs,we can clearly see that the model has a high training accuracy rate and a lower validation accuracy rate. This means we have a model with high variance that might be overfitting the training data. Besides that, although the fluctuation in the validation chart are reduced, they still exist. We can deduce that the robustness of validation results is still low.

Considering these conclusions, we added another Convolution layer to the model and got the following error and accuracy values:

TABLE VII
Error and Accuracy Values

| | |
|---|---|
| **Train Error** | 0.0 |
| **Validation Error** | 0.085 |
| **Train Accuracy** | 100.0% |
| **Validation Accuracy** | 91.5% |

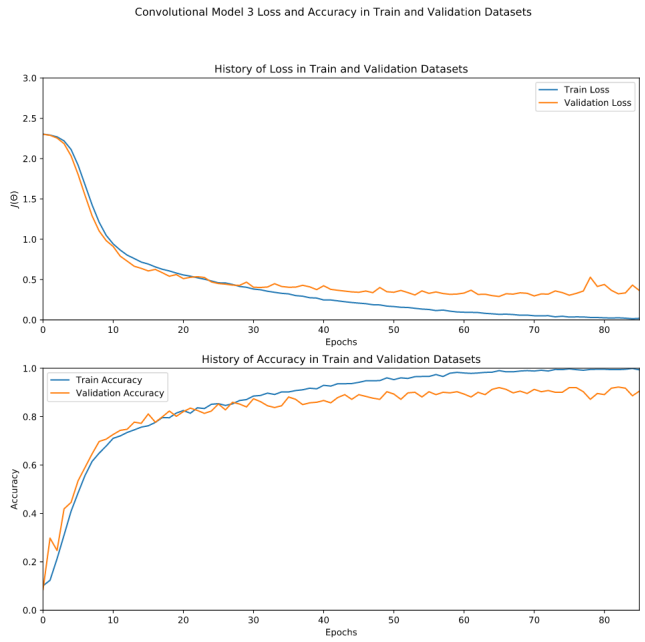


Fig. 7. Convolutional Model 3 Loss and Accuracy in Train and Validation Datasets

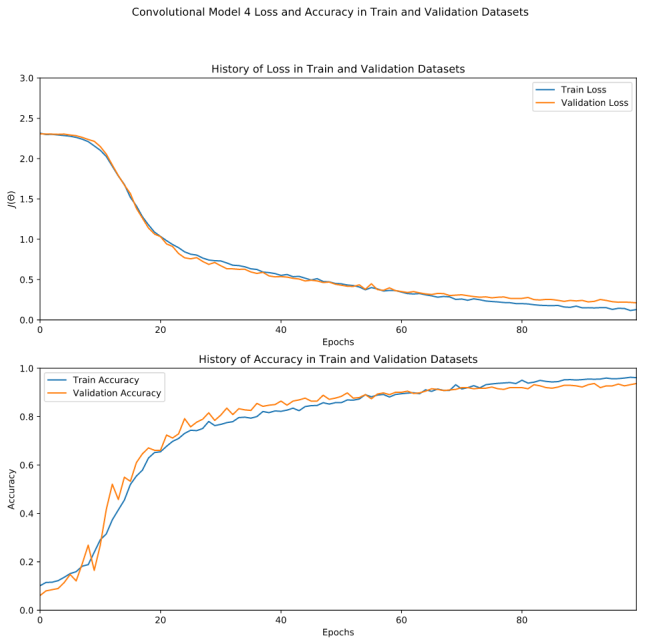


Fig. 8. Convolutional Model 4 Loss and Accuracy in Train and Validation Datasets

The validation accuracy has increased but the problem of overfitting of the model still exists and although the fluctuation in the validation line reduced, they still exist and we can assess that the robustness of the validation results is still low. With this information we inferred that adding a new Convolutional layer was not very useful.

Taking into account what we previously concluded, we used a Dropout layer in the new version of the the model and got the following error and accuracy values:

TABLE VIII
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.008 |
| **Validation Error** | 0.075 |
| **Train Accuracy** | 99.2% |
| **Validation Accuracy** | 92.5% |

As we can see in figure number eight, this change didn't produce results much different from the previous approach because the validation and training accuracy didn't change a lot. However we can see an improvement in the zigzags of the validation chart so we can assess that the robustness of the validation results is improving.

Considering the above conclusions, we added a new Convolutional layer, with max pooling and a dropout layer and got the following error and accuracy values:

TABLE IX
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.055 |
| **Validation Error** | 0.065 |
| **Train Accuracy** | 94.5% |
| **Validation Accuracy** | 93.5% |

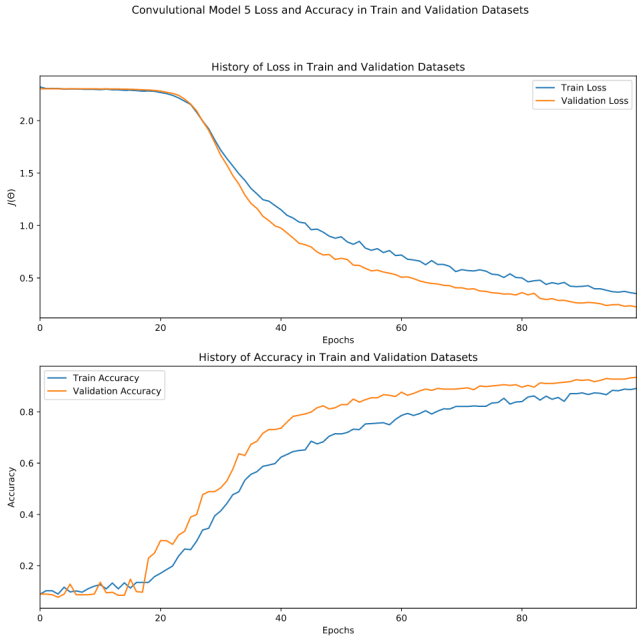


Fig. 9. Convolutional Model 5 Loss and Accuracy in Train and Validation Datasets

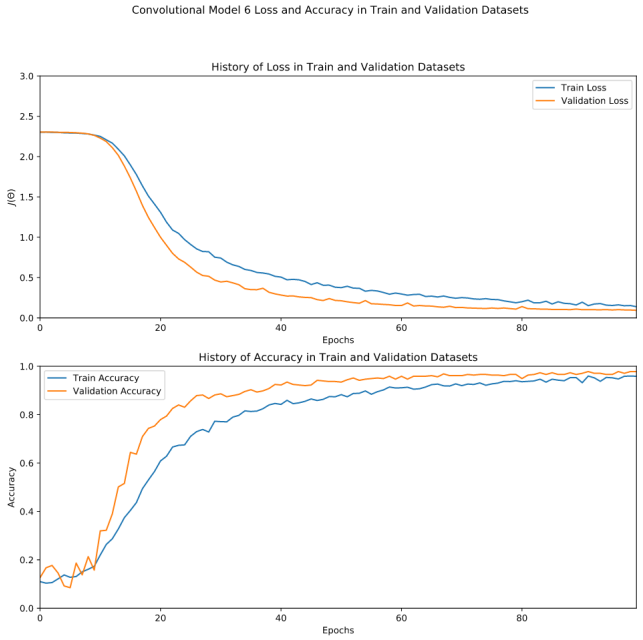


Fig. 10. Convolutional Model 6 Loss and Accuracy in Train and Validation Datasets

As we can see in figure nice, we reduced the overfittig with this model but the training performance decreased and the validation performance is still very poor.

Because of this, we created a new model, where we applied the dropout layer only in the end and increased the number of nodes from 128 to 256 in the full connected layers and got the following error and accuracy values:

We reduced the overfittig, the low robustness and increased the training and validation data performance with the sixth version of our model.

With the objective of developing a better version of this model we added a batch normalization layer that extended our training time but had a positive effect in our results:

TABLE X
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.009 |
| **Validation Error** | 0.036 |
| **Train Accuracy** | 99.1% |
| **Validation Accuracy** | 96.4% |

TABLE XI
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.0 |
| **Validation Error** | 0.029 |
| **Train Accuracy** | 100.0% |
| **Validation Accuracy** | 97.1% |

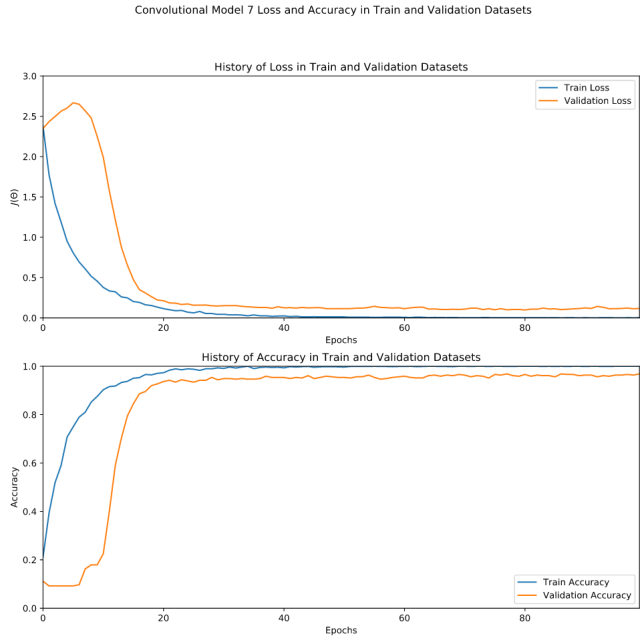


Fig. 11. Convolutional Model 7 Loss and Accuracy in Train and Validation Datasets

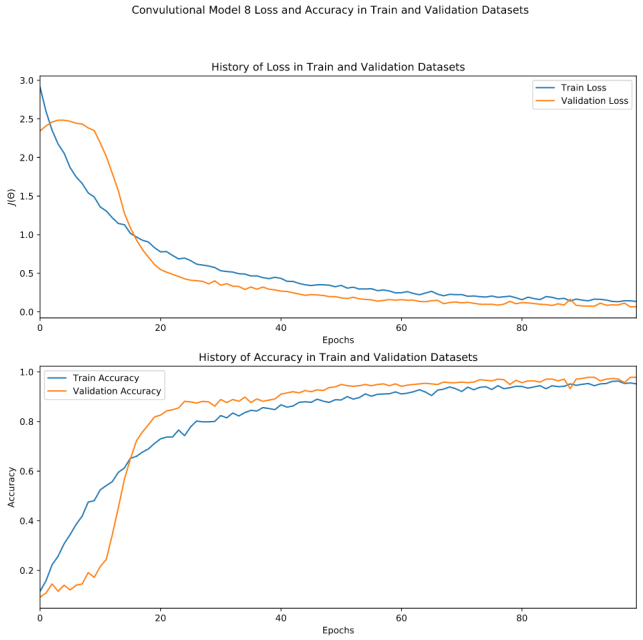


Fig. 12. Convolutional Model 8 Loss and Accuracy in Train and Validation Datasets

As we previously stated, batch normalization had a positive effect but it also introduced some overfitting in our model as we can see in the above graphs and error and accuracy values.

To deal with this problem, we added a dropout layer to each one of our convolutional layers and got the following error and accuracy values:

With this last change we reduced the overfitting and increased a bit our validation performance.

After eight iterations, we developed our best model yet, so we joined our validation and training sets, retrained the model 8 with this set and tested it with the test set. We got the following results:

TABLE XII
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.006 |
| **Validation Error** | 0.022 |
| **Train Accuracy** | 99.4% |
| **Validation Accuracy** | 97.8% |

TABLE XIII
ERROR AND ACCURACY VALUES

| | |
|---|---|
| **Train Error** | 0.0 |
| **Test Error** | 0.022 |
| **Train Accuracy** | 100.0% |
| **Test Accuracy** | 97.8% |

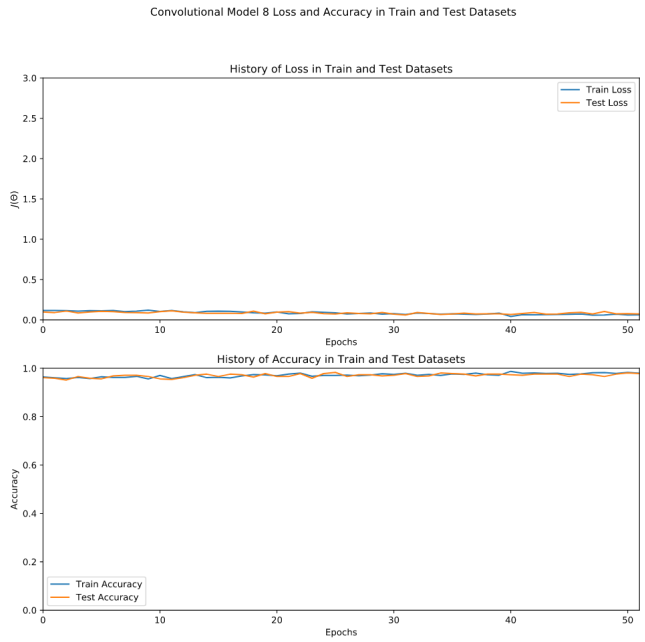


Fig. 13. Convolutional Model 8 Loss and Accuracy in Training and Testing Datasets

As we can see in figure thirteen and training and testing results, the performance of the model was very good, achieving a 0 training error and 0.022 testing error.

To observe in which classes our model had the lowest success rate, we calculated the confusion matrices in figures 14 and 15.

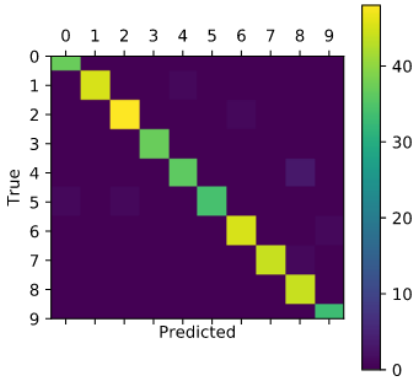


Fig. 14. Confusion matrix as image

```
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 45  0  0  1  0  0  0  0  0]
 [ 0  0 48  0  0  0  1  0  0  0]
 [ 0  0  0 37  0  0  0  0  0  0]
 [ 0  0  0  0 36  0  0  0  3  0]
 [ 1  0  1  0  0 34  0  0  0  0]
 [ 0  0  0  0  0  0 45  0  0  1]
 [ 0  0  0  0  0  0  0 44  1  0]
 [ 0  0  0  0  0  0  0  0 44  0]
 [ 0  0  0  0  0  0  0  0  0 33]]
```

Fig. 15. Confusion matrix as text

By analysing the confusion matrix we can see the model made most of its mistakes predicting the number four, confusing it three times with the number eight.

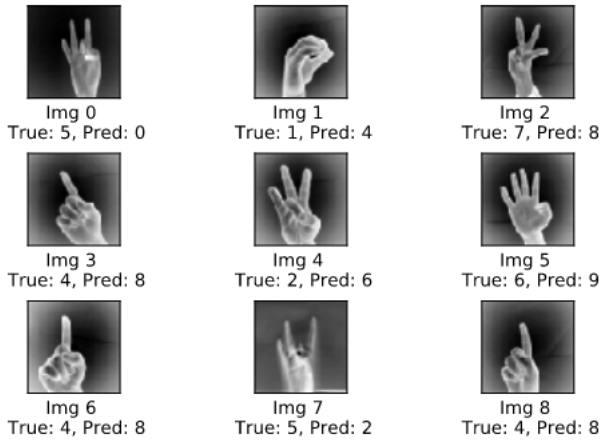


Fig. 16. Error image plots

By plotting the nine images where our model made mistakes predicting the real number we can see that some of them have low quality, for example image number seven, what could have influenced the predictions.

Besides that, some images have similarities with other numbers, as in image zero, the representation of the number five is similar to the representation of the number zero.

This problem could be mitigated with data augmentation: the data-set is quite small and has a small number of examples per class. With data augmentation, by rotating, flipping, cropping and translating (moving the image along the X e/or Y direction), we could create more examples per class that would not be the same as the ones we already have, giving the model the opportunity to better understand the differences that sets apart the most similar classes.

## V. Conclusion

Comparing with the previous results obtained with the logistic regression model and the one implemented with sklearn, there are some final conclusions we can enumerate:

- The Sklearn model is the most efficient one, mainly due to the internal optimizations applied and usage of more efficient algorithms present in the library
- The other topic is the unquestionable better performance of the CNN model comparing to the Logistic Regression model we implemented by hand.

Bellow we have a table where we summarize the training and testing errors of each model we used to solve our problem:

TABLE XIV
LOGISTIC REGRESSION AND CNN MODEL TRAINING AND TESTING ERROR

| | **Training Error** | **Testing error** |
|---|---|---|
| **Logistic Regression Model** | 0.027 | 0.036 |
| **Logistic Regression Sklearn** | 0.0 | 0.0 |
| **CNN Model** | 0.0 | 0.022 |

## WORK DONE BY EACH STUDENT

Due to the restrictions of Covid-19 virus, we were not able to always develop the project together (in the same place or close to each other).

Therefore, João Vasconcelos was responsible for the implementation of the Convolution Neural Network and Vasco Ramos was responsible for the Logistic Regression and research on possible models to solve the problem. The report and PowerPoint presentation were done in collaboration.

Overall, we consider the work was split equally between both members.

## REFERENCES


[1] Petia Georgieva, *Lecture 3: Classification and Logistic Regression*, 2019
[2] Petia Georgieva, *Lecture 4: Neural Networks*, 2019
[3] Petia Georgieva, *Lecture 6: Model Selection and Validation*, 2019
[4] Kalam, Md & Nazrul, Md & Mondal, Islam & Ahmed, Boshir., *Rotation Independent Digit Recognition in Sign Language*, 2019
[5] Kaggle, *Deep Learning Tutorial for Beginners*
[6] Kaggle, *Convolutional Neural Networks Tutorial Tensorflow*