IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Understanding Sign Language using Neural Networks

BERNARDO FALÉ 93331[1], (DETI, UA), HUGO DOMINGOS 98502 [2], (DETI, UA)
[1]Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, Aveiro (e-mail: mbfale@ua.pt)
[2]Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, Aveiro (e-mail: xxxx@ua.pt)

This work was done for the Fundamentals of Machine Learning course with the help of class materials

ABSTRACT Sign language is the primary mode of communication for millions of people around the world. Sign language recognition systems have the potential to significantly improve communication and accessibility for deaf and hard of hearing people; However, this is a challenging task due to the variability of sign languages and the complexity of hand gestures.
In this paper, we propose an approach that uses artificial neural networks to identify digits in sign language.

INDEX TERMS Neural Networks, Machine Learning, Artificial Intelligence, Sign Language Recognition

I. INTRODUCTION

IN today's interconnected world, effective communication stands as a cornerstone of mutual understanding and harmonious coexistence. For many, verbal language serves as the primary medium of expression; however, for the deaf and non-speaking community, hand gestures and sign language emerge as essential tools for conveying thoughts, emotions, and information. Despite its widespread use within this community, sign language remains an enigma to the majority, leading to a palpable communication hole between those who rely on it and those who don't. Addressing this divide is not just about fostering inclusivity, but also about unlocking a realm of untapped interactions and shared experiences. This paper delves into the realm of machine learning and artificial neural networks (ANN); ANN are used for speech recognition, image analysis, adaptive control, and several problems in data science [1]. By employing algorithms tailored for specific data science tasks, we endeavor to create a system that is capable of recognizing and interpreting sign language. Through analysis, graphical representations, and conclusive insights, this study aspires to shed light on the transformative power of technology in making sign language universally comprehensible.

II. DATA DESCRIPTION AND PREPROCESSING

The used dataset consists of a matrix X with 2062 images of hand gesture digits. Each image is of size 64 by 64 and it is in grayscale, so that it means that each image has 4096 total features, each one being a floating point number larger than 0. The dataset also contains a matrix with the correct labels but those are not in the correct order relatively to the X matrix, so that we had to manually do the labeling of the images.

A. NORMALIZATION

The first technique we applied was Z-score normalization where $Z = \frac{(X-\mu)}{\sigma}$ or $\mu = \frac{1}{N}\sum_{i=1}^{N} X_i$, $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i-\mu)^2}$, and $Z_i = \frac{(X_i-\mu)}{\sigma}$. The second technique we tried was simple scaling $X_i' = \frac{X_i - \min(X)}{\max(X)-\min(X)}$, as Z-score might not have been the best technique to implement for grayscale images; This was noted after the results were not good, as we'll see below.

B. FEATURE SELECTION

In order to optimize the training process, as the original data has 4096 features, we decided to perform feature selection in order to reduce the number of features to 2025. To do that, we used an algorithm called Principal Component Analysis(PCA). The motivation behind this algorithm is that there are certain features that capture a large percentage of variance in the original dataset. So it's important to find the directions of maximum variance in the dataset. These directions are called principal components and PCA is essentially a projection of the dataset onto the principal components. By applying PCA to an image dataset, the code is essentially reducing the dimensionality of the images, retaining the most significant features (principal components) while reducing the noise or less significant information. [3] We found this important since the number of features made the training really slow in our ANN; Thus, feature selection was a must

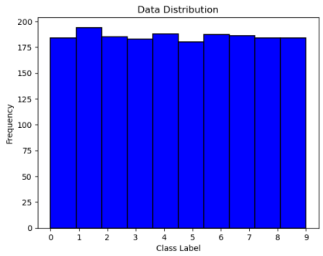

FIGURE 1. Original Data distribution of the training set

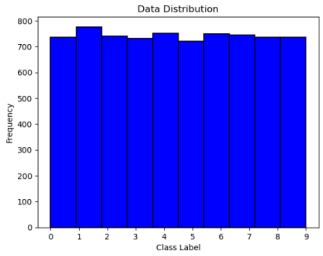

FIGURE 2. Data distribution of the training set after data augmentation

apply technique. However, and as we will see below, it made our model less accurate.

C. DATA AUGMENTATION

In the other hand, we tried to increase the dataset size by performing data augmentation. Data augmentation is a technique where slight modifications are applied to the original data, such as rotations, so that we can increase the number of training images. In our case, we copied each image three times and made random rotations between -10 and 10 degrees in each of the copies, resulting in a training set three times bigger than the original. In figures 1 and 2 we can observe the number of training examples on the original data and the number of training examples after the data augmentation process. We conclude that we increase substantialy the number of training examples, while maintaining the distribution of the data uniform.

III. IMPLEMENTED MODELS

A. ARTIFICIAL NEURAL NETWORK

This section will talk about the first model we developed, and why we had the need to make another one for comparison

1) Architecture

This ANN was very similar to model developed in class 4 and, to us, is seen as the entry point of our work. It is comprised of feedforward propagation and backpropagation, regularized and unregularized cost functions using gradient descent, and the sigmoid activation function.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{k}[-y_k^{(i)}log(h_\theta(x^{(i)})_k) - (1 - y_k^{(i)}log(1 - h_\theta(x^{(i)})_k)] + \frac{\lambda}{2m}[\sum_{j=1}^{25}\sum_{k=1}^{400}(\theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\theta_{j,k}^{(2)})^2]$$

When we first developed the model, we just straight up trained without normalization and feature selection and with the default class hyper-parameters; This was unsustainable, since it took about 20 minutes for a model with 25 units on the hidden layer and 4096 units on the input layer, Figure 3. We tried several combinations of hyper-parameters. We also tried training our model with normalization and without PCA and the other way around. We tried several combinations of architectures as figure III-A1 displays. We decided to stick with 2052 features for the most part of our experiments as the results were the best in comparison with the other reduced feature architectures.

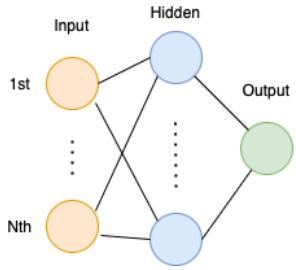

FIGURE 3. ANN architecture

Input	Hidden Layer	Output
400	10	1
4096	25	1
2052	15	1

TABLE 1. Different Architectures

2) Data Split

We divided the data set in two sets: 80% and 20% for training and testing, respectively. We used [4] to split the data.

B. TRAINING

In this model training, we fed the data set as a whole, only splitting as said in III-A2. Since our first try of training the model as a whole took so much time (Because of our feedforward neural network cross-entropy loss function), we decided to apply the techniques mentioned in II. We used Z-Score to normalize our data, and decided to reduce our features to almost half with PCA; Our dataset looked like in figure 4 after the techniques were applied. As we can see, the first pixels are represented with most "information" of the image". This way, we would be able to train it fast, and take key indicators of what would be going wrong and what would be going right. Important to say that we also didn't use adaptable learning result, thus only being hard coded, and changed for more richness and quantity of results. Our strategy for weight initialization was $\epsilon_{init} = \frac{\sqrt{6}}{\sqrt{L_{in}+L_{out}}}$. For optimization of weights and parameters we used a gradient descent algorithm shallow neural network implementation.

C. RESULTS

Training this artificial neural network was extremely frustrating, as the results were constantly what we were not

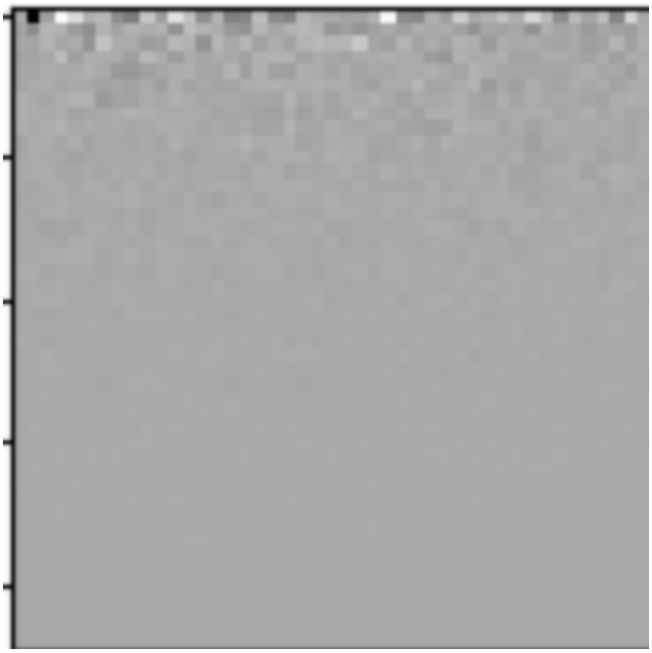

FIGURE 4. Image after normalization and feature selection

expecting. Almost all the iterations that we made of our training were something that we could not present as "good". Most of our training results are in table III-C;

I	Feat	Hunits	α	λ	TrainAcc	TestAcc
1000	4096	25	1	*NaN	10%	7%
2000	2052	15	0.1	*Nan	90%	40%
1500	2052	15	0.1	0.1	90%	42%
2000	400	10	0.1	*NaN	70%	*NaN
1500	2052	15	0.1	0.5	57%	52%
1500	2052	15	0.1	*NaN	65%	64%

TABLE 2. Hyper-parameters of different training *Nan - Not implemented

As we can see in figure 5 the loss cost function converged ultra fast; This is not a good sign, since the accuracy on the test training data was about 10.7%. This gives evidence that the model is too simple for the number of features/complexity of the dataset. In the second iteration we applied the normalization and feature selection algorithms. Note that, from the 2nd iteration through the 4th iteration, we selected the features before we normalized them; After research, we concluded that this might not have been the best approach and not the standard procedure, although it made sound sense to us at the time of training.

After the results of the first iteration we decided to reduce the number of features and introduced the data splitting for better measurements, as well as reducing the number of units in the hidden layer. As seen in 7, the loss function converges much more naturally; In spit of the good percentage of the training accuracy against the data set, the predictions on data that was never seen before were not good at all. We concluded that the model had learned the random fluctuations in the training data rather than the underlying pattern. It definitely

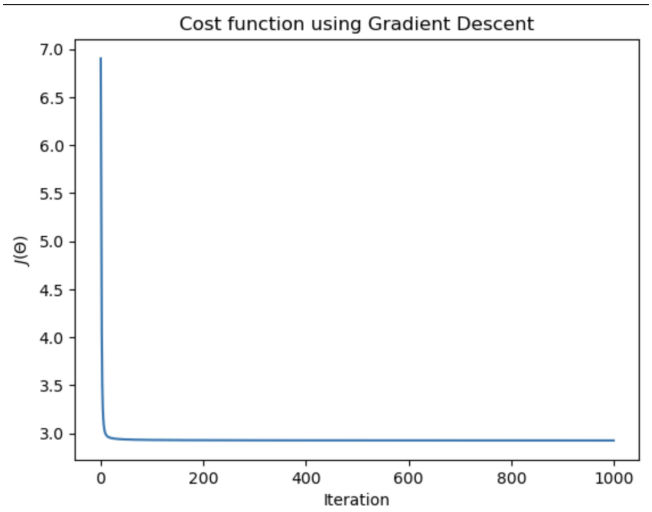

FIGURE 5. First iteration Loss cost function

showed evidence of overfitting. In that case, we took the standard measure: We tried to regularize the function. We can observe in 9 that cost function converges naturally, but that the confusion matrix 10 is indeed confusing; Regularizing the cost function did not help at all at getting the best test results. We also did a brief test for what would happen if we reduced the features to 400; We concluded that the image would lose a lot of its information, and decided that this was definitely not the approach to follow. At this point, we started normalizing the data before we did feature selection, and this meant that results would be a lot more different now. Because we thought that regularizing the function was the way to go, we decided to increase the λ value to 0.5. We can observe in 11 that cost function looks "funky", and diverges in a ridiculous manner. We are back to the underfitting problem, we concluded that the bias value might have been too high, but decided not to test further, as this would be a training black hole, a hole that would take at least 6/7 minutes to train. We tried one last time to train the model on an unregularized function; We based this decision on the fact that the bias values we tried before were rather large, or rather low. This meant that maybe the model might not actually need regularization. Despite the poor results 13 14, they were the most conclusive, and the best ones in terms of training/testing accuracy ratio. This left us wondering if the poor results were due to the simple implementation of the neural network, and if we added more hidden layers, more units per layer and different activation algorithms we would have much better results. As such, we decided to implement a different, more robust, and overall much more complex in III-D.

D. CONVOLUTIONAL NEURAL NETWORK

1) Architecture

This model consists of an input layer that gets the 4096 features of an image, then it passes through two convolutional

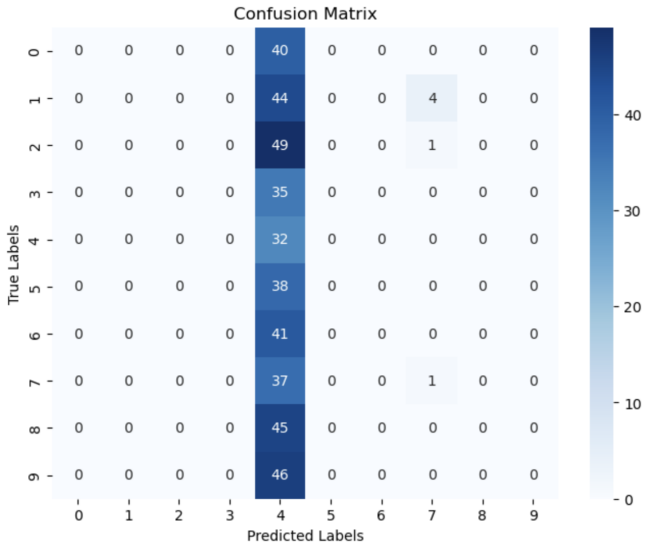

FIGURE 6. First iteration Confusion matrix

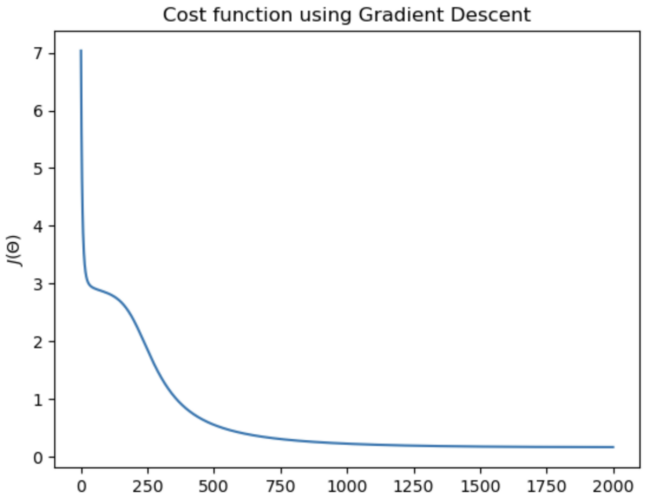

FIGURE 7. Second iteration loss cost function

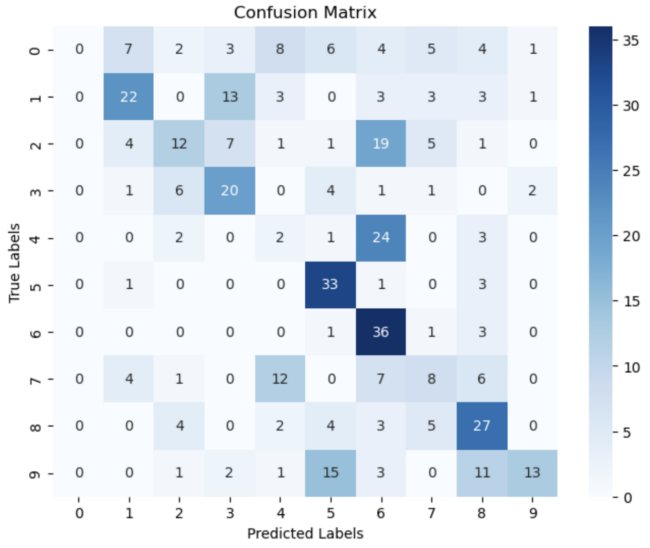

FIGURE 8. Second iteration Confusion matrix

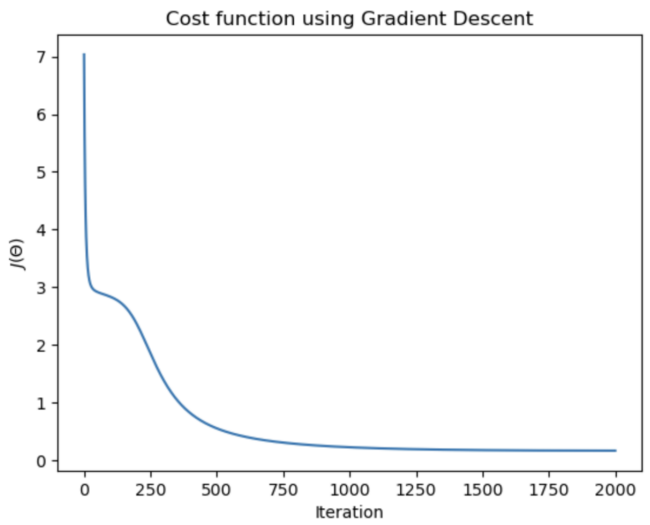

FIGURE 9. Third iteration loss cost function

layers, each one followed by a max-polling layer. The first convolutional layer applies 32 filters with size of seven by seven. What this means is that those 32 filters will slide through the image and help to detect patterns. This layer have $Relu(x) = max(0, x)$ as activation function to introduce non-linearity to the model. After this first convolutional layer, a max-pooling layer is used to down-sample the spatial dimensions. It reduces the size of the feature maps by taking the maximum value within each 2x2 region. This helps in reducing computational complexity and focusing on the most important features.

After this, there are another two layers identical to the previous two in the same order. The only difference is that the convolutional layer now applies 64 filters. This decision to start with fewer filters on the first convolutional layer and then increase in the second one was taken so that an hierarchical feature extraction was made by the network where simpler patterns are learned earlier to help prevent overfitting.

In figures 16 and 17 we can check the feature map results after the aplication of the filters on the first and seconds convolutional layers, respectively, on the image 15. We can clearly see that on the first one there are more general features of the image and in the second that the features are more specific.

As our input layer accepts matrixes with shape 64x64, to connect to fully connected layers we need to transform the data into a vector. This was done with flatten layer that converts the 2 dimension array to a vector with 1 dimension.

After, we have three fully connected layers, also with ReLU as activation function, with the number of neurons on each one being 128, 64 and 32, respectively. This layer help the model to proccess the features learned in the convolu-

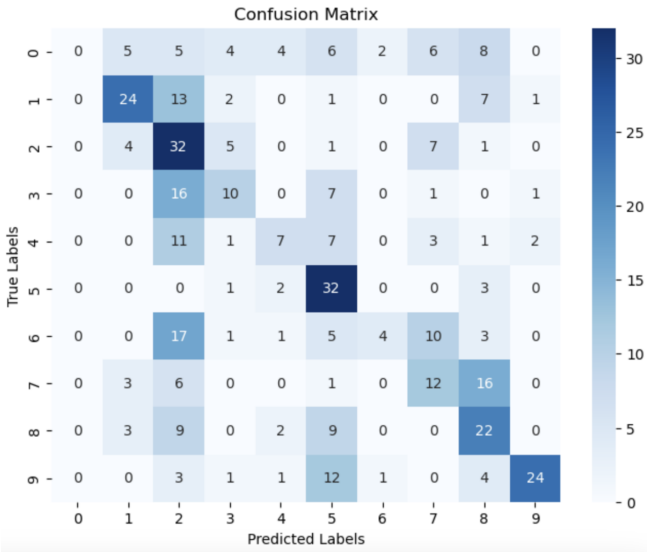

FIGURE 10. Third iteration Confusion matrix

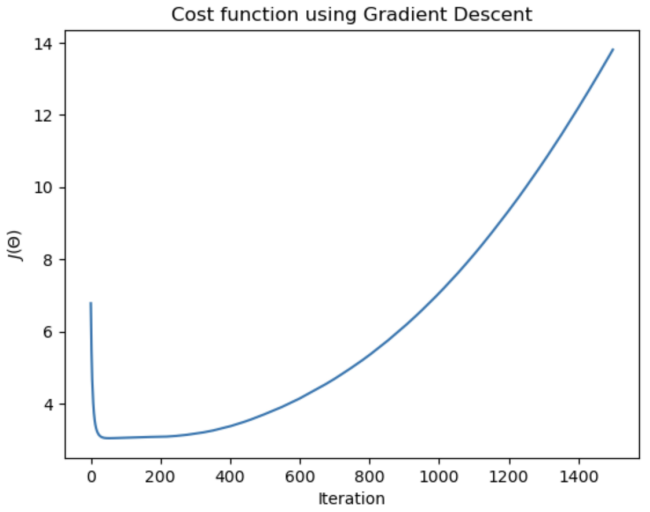

FIGURE 11. Fifth iteration loss cost function

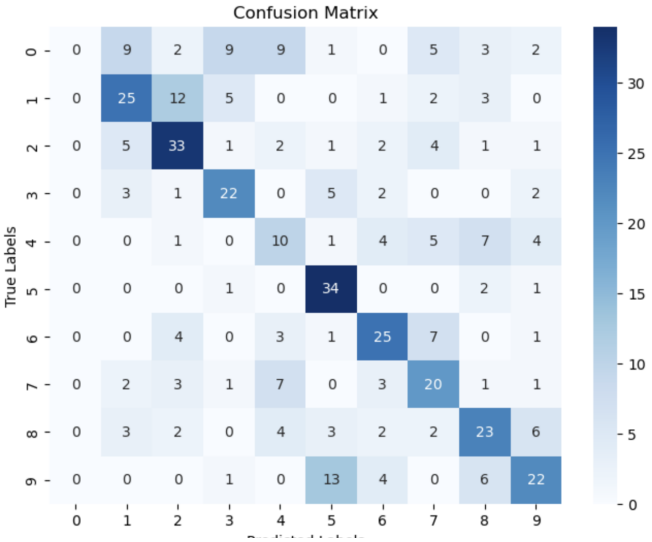

FIGURE 12. Fifth iteration Confusion matrix

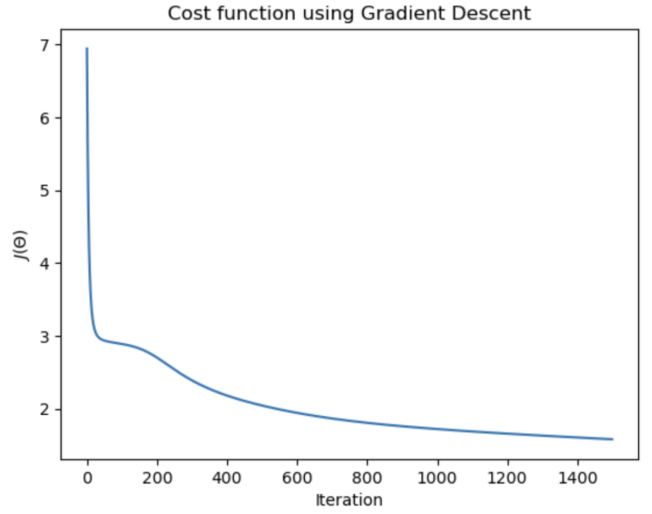

FIGURE 13. Sixth iteration loss cost function

tional layers.

Finally we have another fully connected layer with 10 neurons, representing the 10 classes on this problem. Here we have chosen softmax

$$\sigma(y_i) = \left(\frac{e^{y_i}}{\sum_j e^{y_j}} \right) j = 1, ..., n \qquad (1)$$

as the activation function, so that it produces the output probabilities of each class, that is more suitable to this classification problem as each image has only one correct class.

This model uses the Adam algorithm as optimizer, that is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments. This algorithm was chosen because it is computationally efficient, as it has little memory requirement, and is well suited for problems that are large in terms of data or parameters, as it is this case.

2) Data Split

In this case the data was splitted in three sets: training, validation and test. Firstly we shuffled the original data and divided it two parts, the training and testing sets, the earlier with 90% of the original data and the former one with 10%. Then, 30% of the testing data was separated to be used as validation data.

3) Training

To train this model we introduced the concept of mini-bacthing that instead of training the model with the whole training data and, only then, adjusting the weights on the neural network, the data is divided in mini-batches that are passed to the neural network, and the weights are updated

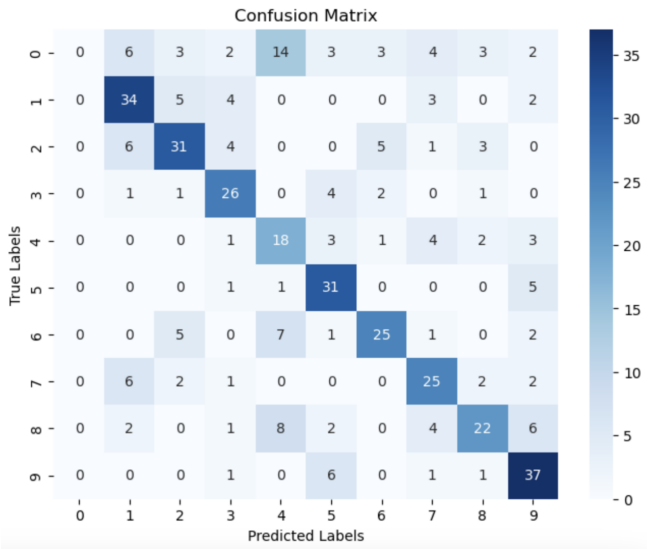

FIGURE 14. sixth iteration Confusion matrix

FIGURE 15. Original Image before the convolutional layers

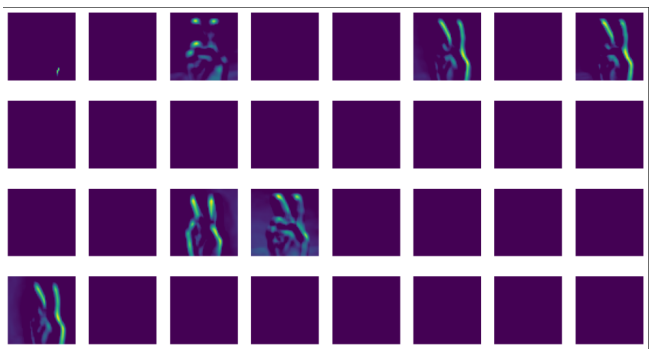

FIGURE 16. Image after the first convolutional layer

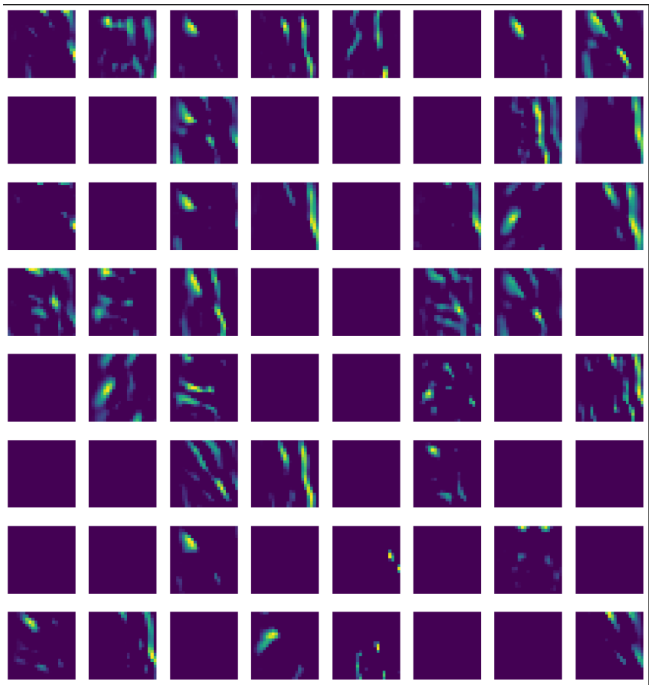

FIGURE 17. Image after the second convolutional layer

whenever the training is completed for that batch. Then we have the concept of epochs, that is when all the batches have feed the neural network to train it.

In this way we can have the best of the two worlds as mini-batch gradient descent strikes a balance between the extremes of batch and stochastic gradient descent.

By dividing the training data into smaller subsets or mini-batches and updating the model's parameters with each mini-batch, we can enjoy several advantages such as faster convergence, as we update the the weights more frequently than with batch gradient descent; and more stability than we using stochastic gradient descent because in that case the weights would be updated in every training example.

We opted for mini-batches of size 10 and 40 epochs to train the neural network as it was the hyper-parameters that gave us the best results.

The training process is initialized with the learning rate setted to 0.001, but we added a callback function that reduces the learning rate by half whenever there is a plateau on the accuracy of the validation data. We defined the plateau to be detected whenever there are two consecutive epochs where the accuracy on the validation data does not increase.

We also added another callback function that enables the training algorithm to perform an early stop. We opted for this approach because the model was getting overfitted. The early stop callback instructs the algorithm that when there are 5 consecutive epochs where the loss of the validation data does not decrease by at least 0.001, the model stops and the the best weights are restored in the model.

The cost function used in this model is the categorical cross-entropy, that calculates the loss in the cross-entropy

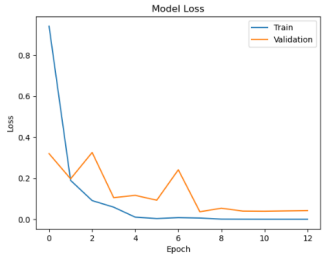

FIGURE 18. Convolutional Neural Network Loss over epochs

between the predicted and the correct labels. We have chosen to use this cost function as it is indicated for multi-classification problems because it encourages the model to produce class probabilities that are as close as possible to the one-hot encoded target labels.

The cost function has the following equation,

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2}$$

, where M is the number of classes, y is the binary indicator (0 or 1) if class label c is the correct classification for observation o and p is the predicted probability observation o is of class c.

4) Results

When training this neural network we get consistent results between 93% and 98% for the accuracy on the test data.

In Figure 18 we can conclude that the model is not over-fitting because the cost loss for both train and validation data are not getting apart over the epochs. Also in Figure 20 we can validate that the same happens.

Observing figures 19 and 21 we have the loss and accuracy for the same model and hyper-parameters but we can check that the cost function is not converting and the model is not getting better results when compared with the results without regularization.

This can mean that the regularization hyper-parameters are applying too much regularization not allowing the model to learn, so that we tried to reduce those but without success. Other approach was to add complexity to the model, adding more layers and neurons on each layer but, again, we where not getting better results. Thus, we choosed to keep with the approach without regularization.

In the confusion matrix in figure 22 we observe that most off the predicted labels are correct, only having 8 false positives in 144 total predictions on the test data.

IV. CONCLUSION

With the work presented in this report we took some conclusions.

Firstly we verified that convolutional neural networks are a more suitable approach for the task of multi-class image classification than a neural network without convolutional layers.

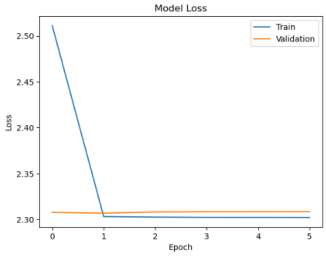

FIGURE 19. Convolutional Neural Network Loss with regularization over epochs

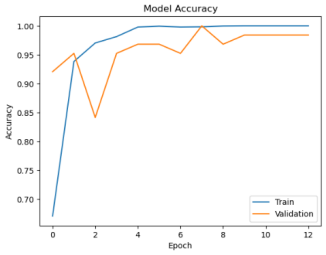

FIGURE 20. Convolutional neural network accuracy over epochs

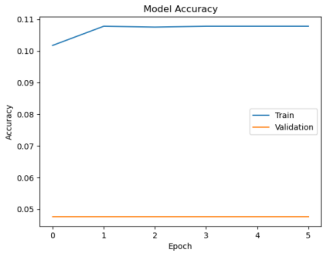

FIGURE 21. Convolutional neural network accuracy with regularization over epochs

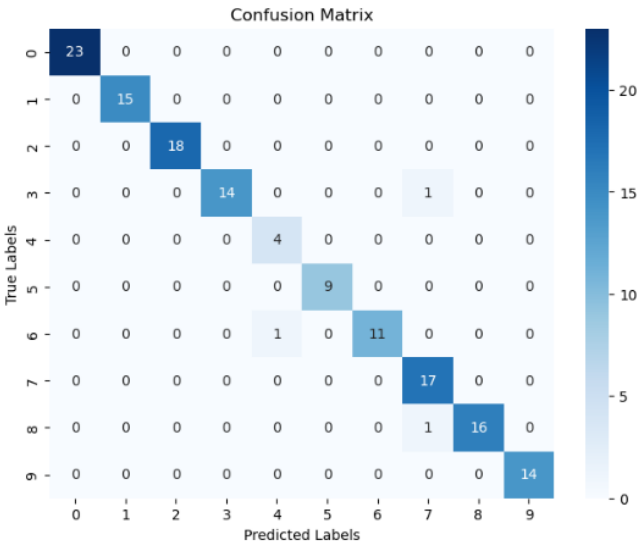

FIGURE 22. Convolutional Neural Network Confusion Matrix

We can also conclude that the use of mini-batch was the best trade-off between performance and results.

The techniques applied to our data-set, such as feature selection and normalization didn't help achieving better results. On the other hand, data augmentation let us achieve better results by increasing the training data-set size.

We also learned to choose the best trade-off between the time spent on training the model and the accuracy by using, for example, feature selection for the data preprocessing and mini-batch for the training process.

To conclude, we consider to have successfully developed a system that classifies the images with great accuracy.

• • •

REFERENCES

[1] Ms. Sonali. B. Maind, "Research Paper on Basic of Artificial Neural Network" in International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169 Volume: 2 Issue: 1 , January 2014
[2] I.A. Adeyanju, "Machine learning methods for sign language recognition: A critical review and analysis" in Intelligent Systems with Applications 12, 26 May 2021
[3] https://builtin.com/data-science/step-step-explanation-principal-component-analysis
[4] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html