Translating American Sign Language with Convolutional Neural Networks

Juan David Torres Velasco, A01702686, Tecnológico de Monterrey

Abstract. The following document shows the implementation and explanation of the use of convolutional neural networks to translate images of American Sign Language to actual letters.

I. Introduction

The hearing-impaired community has been always present in our lives and to communicate, due to their disability, they use hand gestures also known as sign language. There's no universal sign language but rather many variations of sign language that differ between countries. The American Sign Language is used mostly in the United States, in some places in Canada and very few others in Mexico. Even though the hearing-impaired community is very large the people that aren't part of this group can't communicate by Sign Language and this affects the communication.

Given that most of the people in the US have phones they could use a translator that uses the camera of their phone to translate the letters of the American Sign Language, so the communication between these people can improve. Therefore, I decided to base my project on the use of Convolutional Neural Networks to create a model that can translate images of American Sign Language to the letters that they correspond.

II. Convolutional Neural Networks

Convolutional neural networks were introduced firs in the 1980s by Yann LeCun, a computer science researcher, but because of their need for a lot of processing power they couldn't be scaled and therefore weren't used a lot. In 2012 a convnet called AlexNet evidenced in an ImageNet challenge that the processing power was now able to do deep learning and that maybe it was time to use convnets more. Since then, researchers have been using them to perform computer vision tasks. [1]

Convnets are mostly used on image and video models because of their ability to detect features and patterns from visual representations.

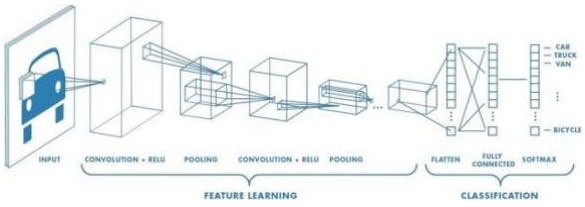


Figure 1 CNNs Representation

As it can be seen in the Figure 1 The first layer is usually the input and to this layer as well to the next feture learning layers a kernel is applied as a sliding window for each of the pixels in the image as shown in the figure 2 where the K multiplies the I and the result is then the dot product of these numbers. The result of this convolution process is then activated by an activation function, usually ReLU known also as the Rectified Linear Unit for image. The resulting layer then will be smaller than the one before and a maxpool layer is applied then to decrease the size of the resulting layer, this process can be done multiple times. After feature learning layers
are done, the classification layers are used first to flatten the result layer, then to create a neural network and then using an activation function for the classification. For categorical values the SoftMax function is used and for binary categories the Sigmoid function is used. [2]

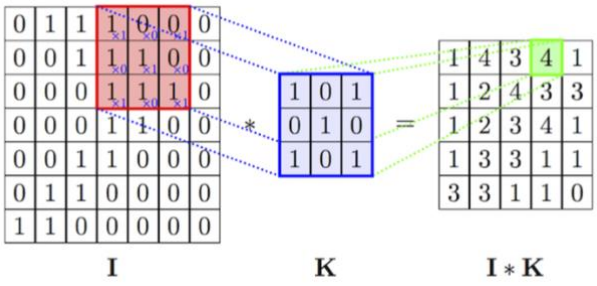


Figure 2 Convolution Process

III. Data set

The initial dataset used is the Significant ASL Alphabet found on Kaggle on the following link: https://www.kaggle.com/kuzivakwashe/significant-asl-sign-language-alphabet-dataset

As it can be seen in the figure 3.1 the same letter was represented different for the initial dataset and I decided to change it because of the overfitting.

Figure 3.1 Sample of K for the initial dataset

But because of the overfitting I used the dataset ASL Alphabet and can be found on Kaggle on the following link: https://www.kaggle.com/grassknoted/asl-alphabet

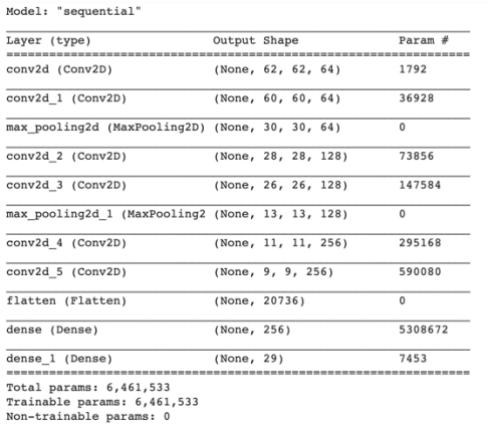
This dataset I used at the end contains approximately 87,000 images of 200x200 pixels. It has 29 classes of which 26 are the letters from A-Z and another three classes, one for the space, delete and nothing. Given that my intention with the model is to translate one by one the letters, I removed the space and delete classes from the actual data set. All the letters are on American Sign Language and in the Figure 3.2 it can be seen what each of them mean. In the figure 4 is an example of a sample of the dataset for the letter C.

Figure 3.2 ASL Alphabet

Figure 4 Sample of the Dataset (C word)

III. Initial Model

The initial model I created was by using a sequential model which had two convolutional 2d layers and then a 2dmaxpool which cuts the output of the layers by the number given which in this case is in half. In the figure 5 can be evidenced the sequential model used initially. The optimizer used is the Adam Optimizer which is an extension for the stochastic gradient descent that updates the weights of the model iteratively with the training data set and has an adaptive learning rate. [3]



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 64)        1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 60, 60, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 9, 9, 256)         590080
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0
_________________________________________________________________
dense (Dense)                (None, 256)               5308672
_________________________________________________________________
dense_1 (Dense)              (None, 29)                7453
=================================================================
Total params: 6,461,533
Trainable params: 6,461,533
Non-trainable params: 0
```

Figure 5 Initial Model

IV. Overfitting and Validation

Overfitting happens when a model has high accuracy on the training but when the model is tested it has a very low accuracy, this means that the model is very bad when a new input is given because it can only be correct for the training samples.

To check overfitting, it is necessary to use validation to check whether the model is performing well with the training dataset as well as with the validation data. But because I didn't have validation data, I used a technique called cross-validation which uses a percentage of the training dataset to train and another portion to validate whether the model is doing well or not. For this model I used the 10% of the training set for validation and 90% for training.

After the training was done with the initial dataset the graph for training - validation accuracy and loss were the ones shown on the Figure 6. While the training accuracy was very high the validation accuracy was going very low, when the training loss decreased the validation loss got very high and a significant difference on both loss and accuracy could be perceived.

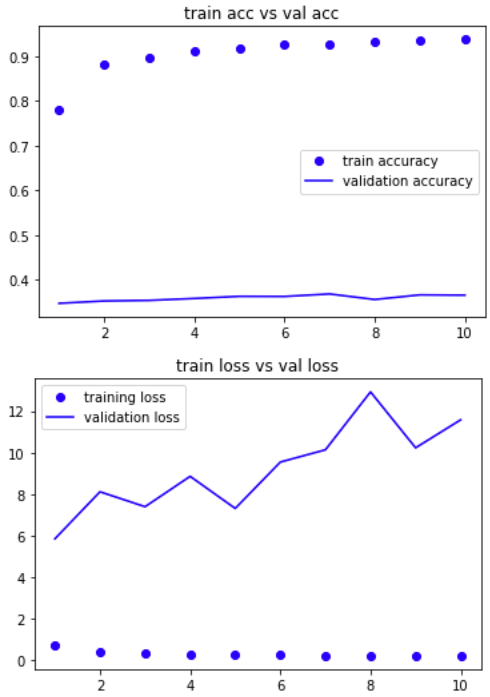


Figure 6 Overfitted Model

V. Transfer Learning VGG16

Given that the initial model was having overfitting, it was necessary to solve the issue. To do so I used the VGG16 pretrained model using transfer learning.

The VGG16 is a convolutional neural network model created by K. Simonyan and A. Zisserman from the University of Oxford. This model achieved the 92.7% in an ImageNet test accuracy. It improved over AlexNet by replacing the large kernel with multiple 3x3 filters in a sequence, the architecture is shown in the figure 7. This model then was trained for weeks and this is one of the most used pretrained models for image classification [4]. The use of transfer learning evidently helped to solve the overfitting of the model and this will be seen in the following figures of the document.

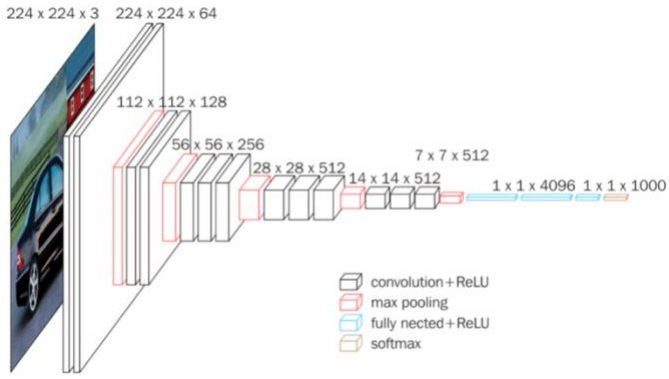


Figure 7 VGG16 Architecture

VI. Data Augmentation

Overfitting can be caused as well because there are not so many samples on the data set and this prevents the model to be good at generalizing as it is not exposed to changes of the images that could happen in real life. Therefore, a great way to solve this problem is by using data augmentation as it creates new samples for training by changing certain characteristics of the original training data set like the brightness, the rotation, the horizontal flip, the vertical flip, withing others [5].

Consequently, I used data augmentation to create new samples based on the data set samples and changed their brightness because sometimes there can be conditions of low lighting when taking a picture, added a horizontal flip because there can be cases in which a person is left-handed and the horizontal flip can do a great job managing that condition, I changed the zoom range as well because there can be cases where the actual person can be near or far from the subject it wants to translate, changed the rotation range too but only 10 units because many letters are very similar to each other when rotated, and rescaled the image so that the process of training can go faster.

VII. Dropout

Dropout is mostly used to prevent a model of overfitting. This is done by ignoring a percentage of nodes chosen at random so that they are not considered for the back and feed forward propagation of the process of training and decrease the overfitting. In figure 8 there is an example of how a dropout can help a model to be more flexible and generalize better. Given the following I used a dropout layer of 50%, so the overfitting could be mitigated. [6]

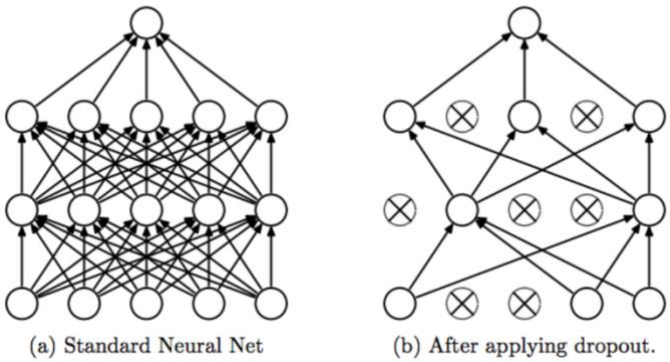


Figure 8 Before and After Dropout

VIII. Non-Overfitting

The use of the new dataset also helped so the model didn't overfit because the initial dataset had different signals for some letters affecting the actual predictions for the model.
The new sequential model is shown in the figure 9.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 6, 6, 512)         14714688
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dropout (Dropout)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9437696
_________________________________________________________________
dense_1 (Dense)              (None, 27)                13851
=================================================================
Total params: 24,166,235
Trainable params: 9,451,547
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 9 Final Model

After doing so the new graphs for the accuracy and loss for the validation and training set changed and now the loss and accuracy for validation and training data were very close to each other as it can be seen in the figure 10.

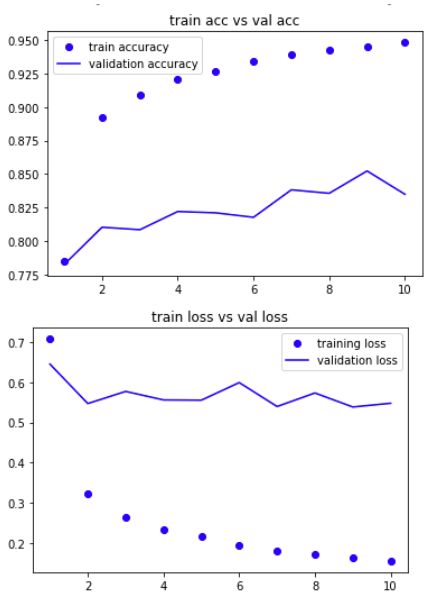


Figure 10 Not Overfitted Model

IX. Application

Once the model was finished, I had to save the model and export it, so I could use it for the mobile application implementation. I used the CoreMLTools provided by apple to export the Keras and TensorFlow model to use it as an MLModel for iOS development and query the model using the mobile application I was going to create.

Once the model was exported, I started by implementing an image selector and a camera to use the photos from the camera or the library of the iPhone. This is how the home page looks like.

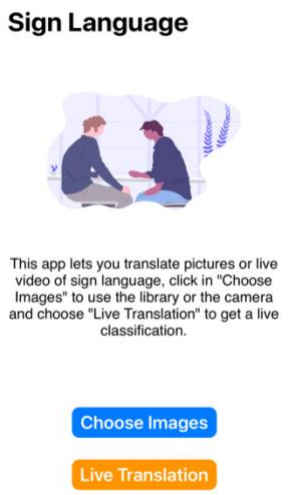


Figure 11 Home page

The section for choosing an image looks like the following when you choose an image of the library or take a picture.

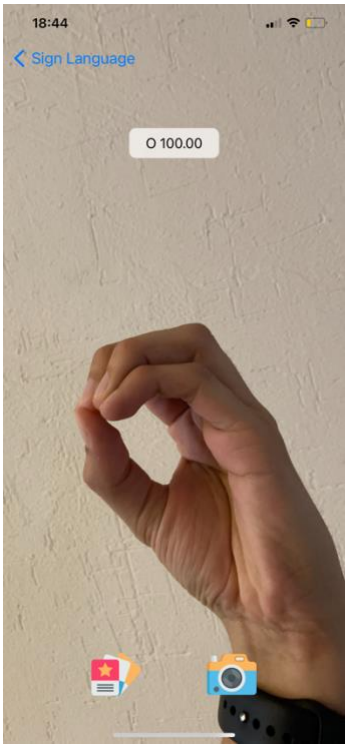


Figure 12 Choose Image Section

The live translation section uses a real time camera output to query the model and predict the sign language letter is representing the camera footage.

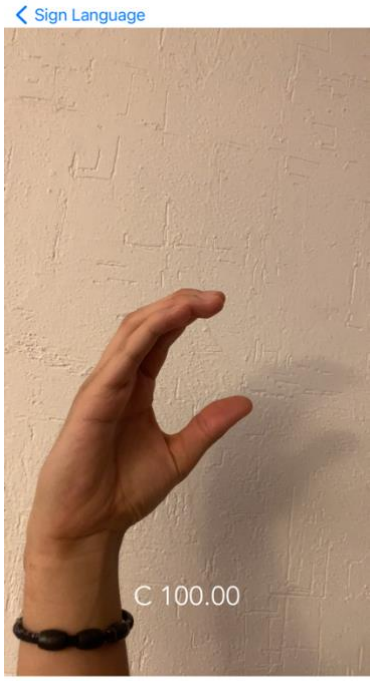


Figure 13 Live Translation Section

Finally, the last step of the process is to deploy the application but given that Apple has a long deployment time for applications it takes time for the app to be published on the Apple Store.

X. Problems Faced

The problem with the image classification of American Sign Language Alphabet is that many of the letters are very similar to each other and this can create a lot of confusion to the actual model, as well as the positioning of the hand and how the camera perceives it.

One of the letters are the K and V that because of the similarity as it can be seen in the figure 14 the model can get confused with those two letters.

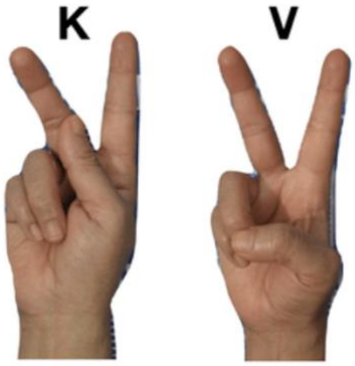


Figure 14 K and V comparison

Other letters that can be confused between each other are the N and M because they only change one finger as well as the K and V and this can be seen in the figure 15.

Figure 15 M and N comparison

The A and S also had problems with their translation because they are as well very similar as it can be appreciated in the figure 16.

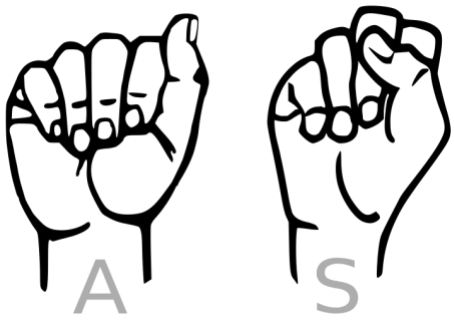


Figure 16 A and S comparison

XI. Conclusion

The use of this application can help millions of people in the US to translate American Sign Language at any given situation and help improve the communication with hearing impaired community. Convolutional Neural Networks can be used to solve many problems that could help improve people's lives immensely, keeping this in mind is very important to keep improving the use of these Networks and even research more ways in which deep learning can get better and be applied to even more complex models.

To solve the problems I faced during this implementation, I could use data sets with different images as the ones used in the training, but given that these problems are present in many of the actual models for ASL translation it is in the present very difficult for a CNN model to predict with 100% accuracy all the ASL alphabet for real life situations.

Maybe a new feature for this app could be to save the sequence of letters that appeared more time on the screen and then create the word that the person is trying to communicate.

The potential of these project is very high, and I will continue in the process of improving the app so more people could benefit from these and solve a problem that has been present in our lives for many years.

References

[1] Draelos, B., 2019. *The History of Convolutional Neural Networks.* Available at: https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/

[2] Saha, S, 2018. A Comprehensive Guide to Convolutional Neural Networks Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] Brownlee, J., 2017. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[4] Popular Networks, 2018. VGG16 – Convolutional Network for Classification and Detection Available at: https://neurohive.io/en/popular-networks/vgg16/

[5] F. Chollet, Deep learning with Python, p. 138. Shelter Islands: Manning, 2018.

[6] Budhiraja, A 2016. Dropout in (Deep) Machine learning Available at: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5