A Convolutional Neural Network to classify the American Sign language

project for the Context Aware Security

Analytics in Computer Vision course

by Antonio Garofalo

Abstract

The American Sign Language is used approximately by 250.000 Americans on a daily basis. A tool to recognize such language would facilitate the lives of those who use the ASL and those close to them, for various purposes such as communicating with people who can't understand the ASL. In this document various techniques are analyzed in order to produce a Convolutional Neural Network aimed at classifying fingerspelling images.

Related works

A study from Brandon Garcia and Sigberto Alacron Viesca from Stanford University [**1**], focuses on developing an ASL recognition system used to translate a video of a user's ASL signs into text, which uses a Convolutional Neural Network (CNN) to solve the underlying multi-class classification problem. The solution proposed on their paper aims at removing the constraints imposed by 3-D capture elements or motion-tracking devices, as their system works through a simple web application. Or in a study by Ameen, SA and Vadera, S, from the University of Salford [**2**], where a similar study has been conducted: a classification problem based on classifying the American Sign Language fingerspelling using depth and colour images. On both these cases a Convolutional Neural Network has been adopted in order to classify the underlying problem. Furthermore, other studies on this subject have been conducted, and a big portion on them relies on the use of Convolutional Neural Networks, with accuracies varying from 80% to as much as 99.7%. Some considerations on hyperparameter tuning have been taken based on various blog posts present on medium.com, for instance the effect of batch size on training dynamics.[**3**]

Proposed method

Upon accurate research on the studies quoted in the previous section, it has been decided that the classification of fingerprints relies on the extraction of features from a certain picture, therefore the most suitable classifier for this task is the Convolutional Neural Network. The pictures used on this project were downloaded on kaggle.com. The dataset consists of 87000 pictures for the training set but just a mere 29 pictures for the test set, therefore some pictures were taken personally to be used in the test set, specifically, 3 picture for each class were taken.

Input shape

An input shape of 64x64x1 has been adopted because the original dataset downloaded from kaggle.com consists of pictures with a resolution of 200x200, and resizing the pictures to 64x64 would not compromise the aspect of the picture in a major way. Also, the pictures provided are in RGB but the color is considered useless for the purpose of this project, as the CNN needs to learn how to classify the American Sign Language based on the shape of the fingerprints alone. Therefore, a grayscale color mode was used. In such a way it was possible to save a considerably high amount of computational power. Eventually, some experiments using a RGB color scale have been conducted anyways.

building the CNN

Different hyperparameters were tested for different configurations of the network, such as learning rate, number of epochs, and batch size. In order to load the pictures, a generator was considered: it allows the processors to avoid performing computation-intensive and memory-intensive operations by breaking down the dataset into smaller batches and working on it batch by batch, the one available on the Keras API [**4**] also performs data agumentation on the batches. However, since the dimension of the dataset is extremely large, data augmentation was declared not necessary and so the pictures have been loaded using openCV, which is a Python library designed to solve computer vision problems. the contents of the images and

labels lists were simply taken using glob, which is a Python library used to return all file paths that match a specific pattern, by iterating each folder of the dataset. Regarding the validation, it was chosen to simply split the dataset into 80% training set and 30 % validation set. A stratified K-fold cross validation was considered, however, implementing such solution in a dataset as big as 87.000 pictures would weigh heavily on the performance. In order to split the dataset the train_test_split function from sklearn was used. So after the initial setup this is the size of the splitted dataset:

```
Train data :  60900 60900
Validation data :  26100 26100
Test data:  116 116
```

Below some samples for the training set and the test set are reported, on figure 1 and fugure 2 respectively.

Figure 1: Samples of the training set.

Figure 2: Sample of the test set.

When it comes to activation function, Rectified Linear Unit (ReLu) was used for the input and hidden layers, in order to let them be either completely active or non active at all. Obviously the same one cannot be adopted for the output layer as it is necessary to provide a certain percentage of activation for each output neuron, therefore a Softmax activation function was used.

Experiments

This section is used to report the carried out experiments and some relative information. The experiments were made on a Macbook Air with M1 processor. The method used to evaluate the performance of the model is by using a confusion matrix with the test set, and also to use the method .evaluate.

Experiment #1

The architecture for the first experiment wasn't choosen using any particular technique. The approach during this experiment was to see how the model would react on a first try, to make up a general idea on how to improve it.

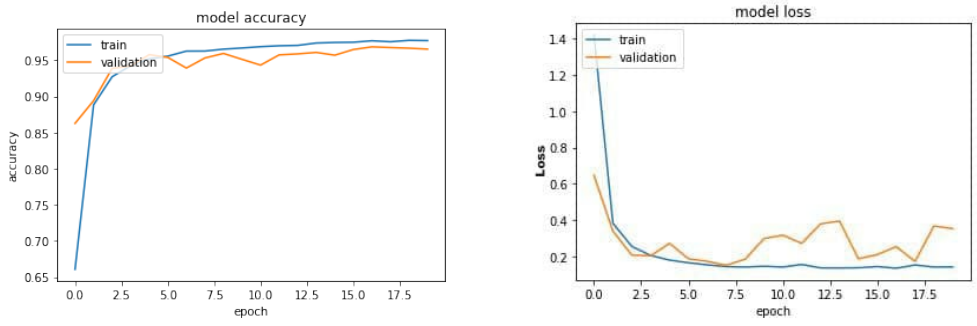


Figure 3: accuracy and loss graphs for the first experiment

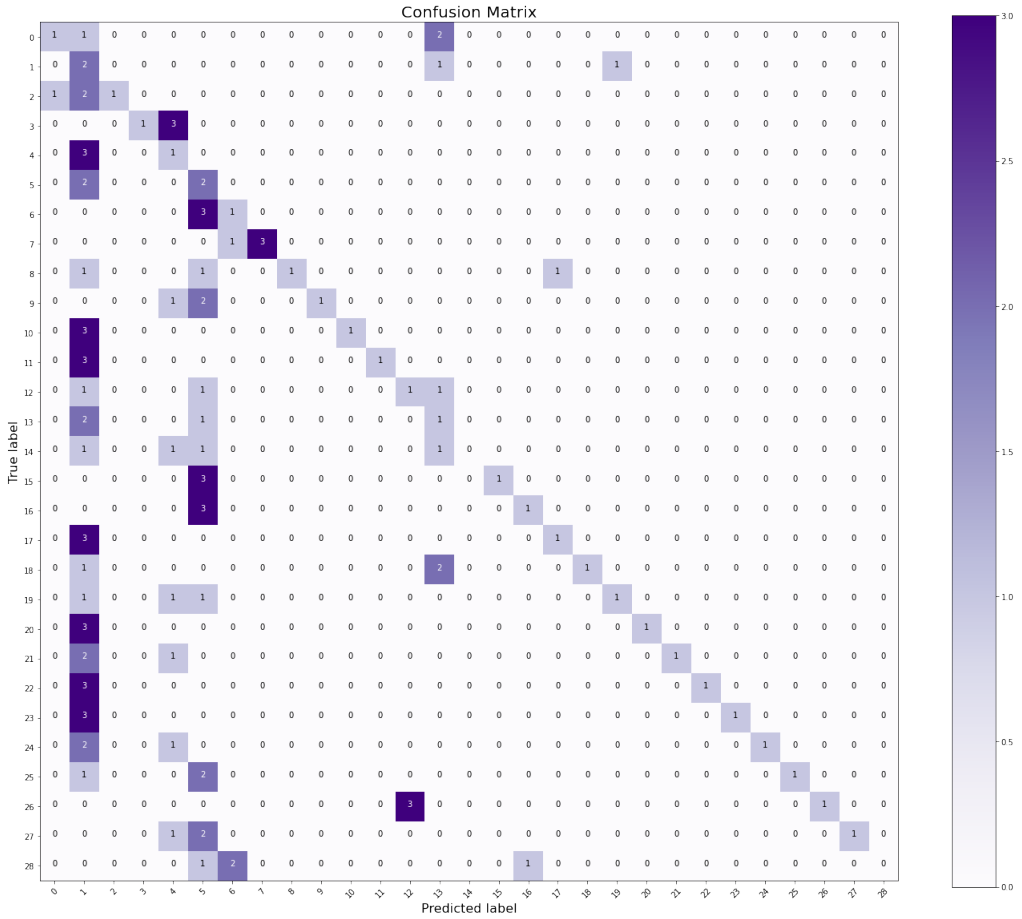


Figure 4: confusion matrix for the first experiment1.

The training accuracy and loss are respectively 0.9772 and 0.1570. The validation accuracy and loss are respectively 0.9654 and 0.3542. By calling the .evaluate method, the precision of the first experiment for the test images is 26.724%. Even though this was the first experiment, the result was relatively good. A 97% accuracy on the training set is very good, however, a difference as small as 1% between training and validation could mean that the model was starting to overfit.

Experiment #2

On the second experiment, the number of epochs was reduced to 10 just to try and prevent that little overfitting noticed in the first experiment.Then, the batch size was increased to 64 and the number of neurons was increased for each layer.

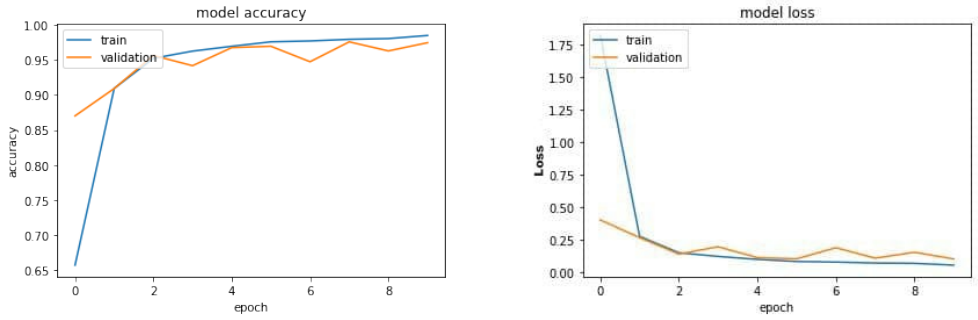


Figure 5: accuracy and loss graphs for the second experiment

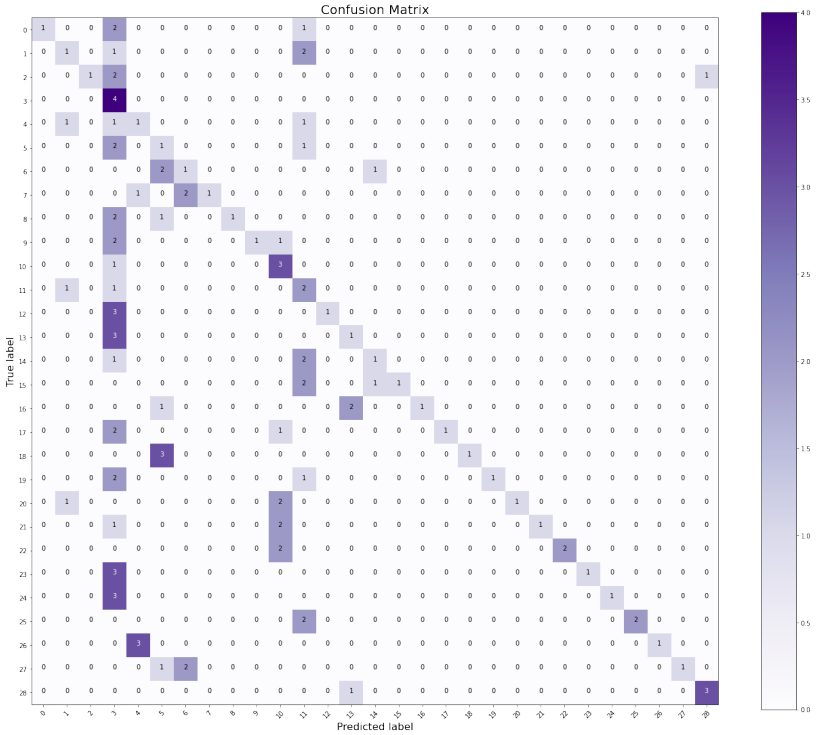


Figure 6: confusion matrix for the second experiment.

The training accuracy and loss are respectively 0.0518 and 0.9844. The validation accuracy and loss are respectively 0.9741 and 0.0994. By calling the .evaulate method on the test set, the accuracy for the test images is 33.621%. Compared to the fist experiment, an improvement of 6% is a huge success. However, the validation accuracy was still a little lower than the training one, which could again mean that the model was experiencing some overfitting again.

Experiment #3

Incremented the epochs to 20, and decreased the batch size to 10. Furthermore, a new Convolution layer, followed by a Max Pooling layer was added. The reason behind this hyperparameters tuning was mainly based on testing how the model would react with an additional convolution layer. While it's true that adding layers adds weights, and therefore complexity, to the model, the dataset is large enough to allow this.

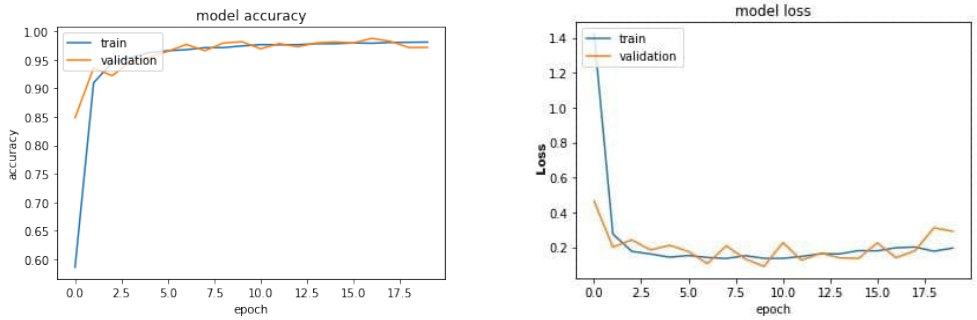


Figure 7: accuracy and loss graphs for the third experiment

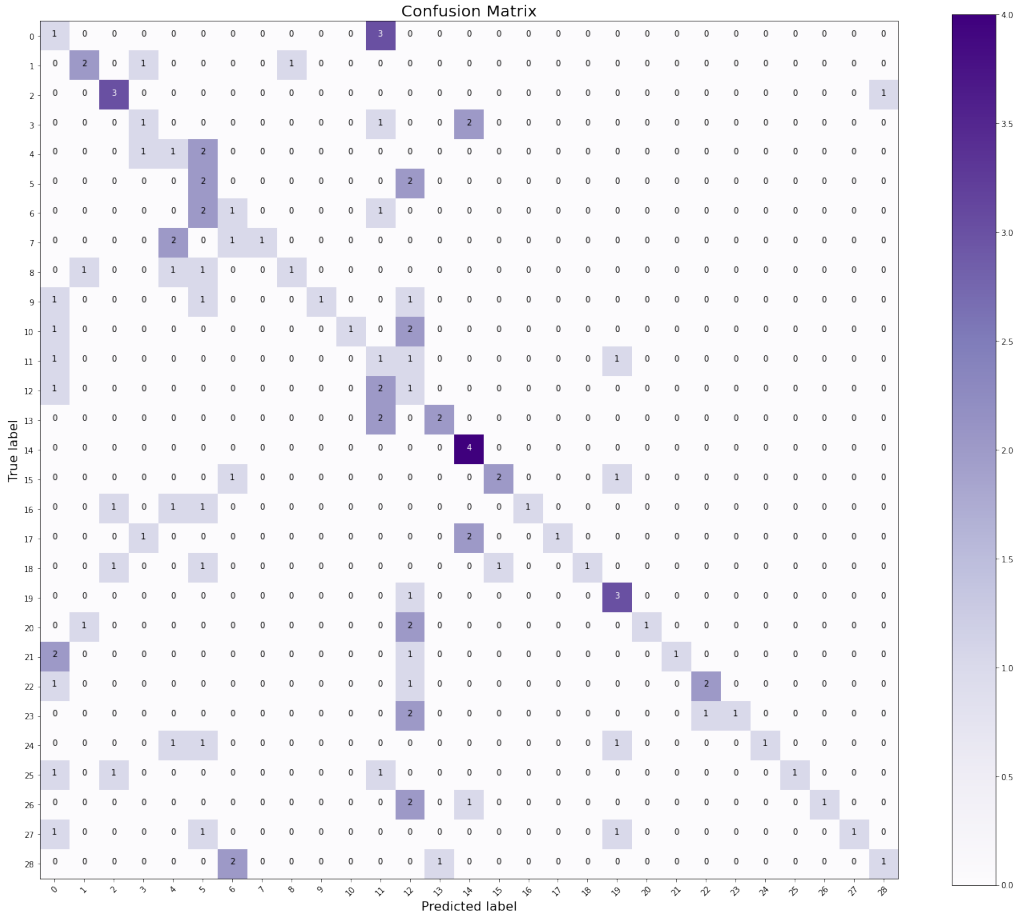


Figure 8: confusion matrix for the third experiment.

The training accuracy and loss are respectively 0.9813 and 0.2908. The validation accuracy and loss are respectively 0.9720 and 0.2908. By calling the .evaluate method, the precision of the first experiment for the test images is 35.345 %. This experiment resulted in an improvement of 2% on the test set evaluation, the training accuracy is still, even by an extremely little amount, bigger than the validation accuracy.

Experiment #4

Decreased the epochs to 10 mainly to try and prevent the very small overfitting that was mentioned in the previous experiment, increased the batch size to 64 in order to update the weights less frequently, and trying a new optimizer, the stochastic gradient descent with a learning rate of 0.01. Also tried lowering the number of layers.

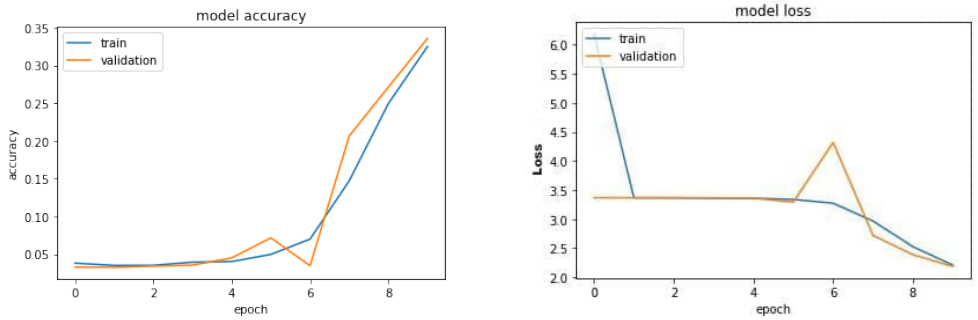


Figure 9: accuracy and loss graphs for the fourth experiment

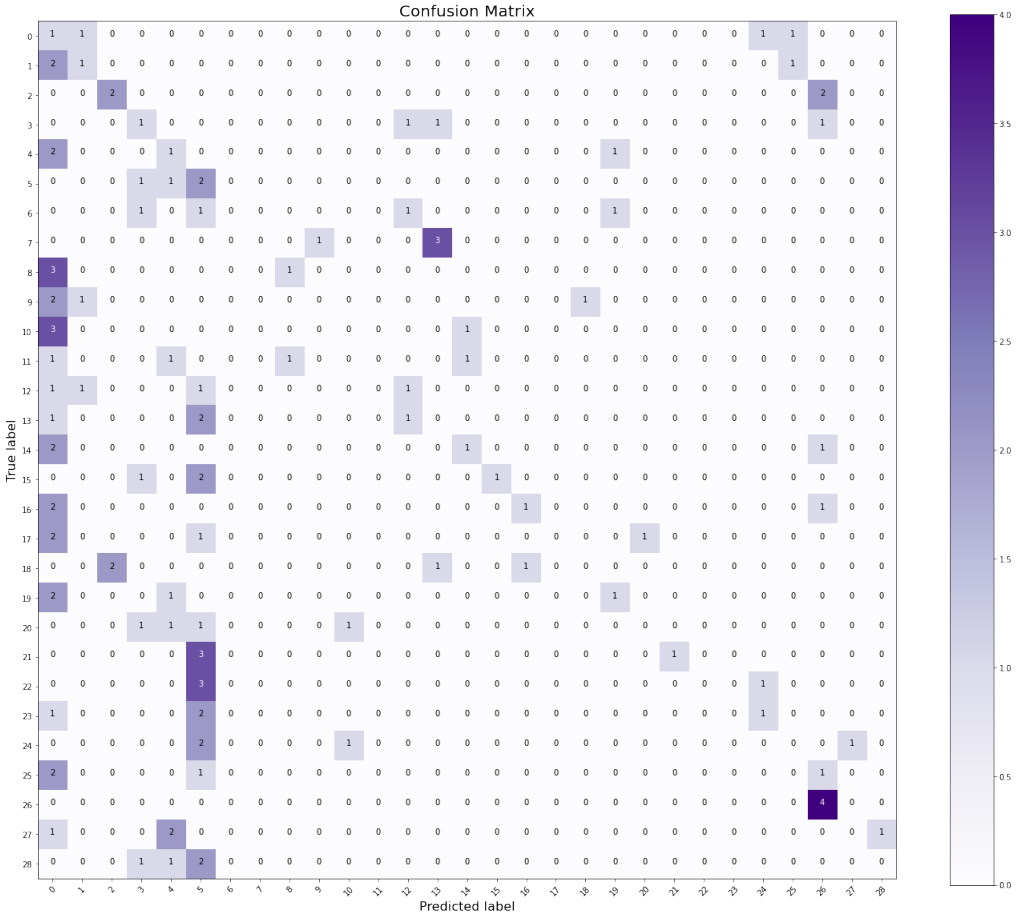


Figure 10: confusion matrix for the fourth experiment.

The training accuracy and loss are respectively 0.3250 and 2.2111. The validation accuracy and loss are respectively 0.3356 and 2.1871. By calling the .evaluate method, the precision of the fourth experiment for the test images is 16.379%. The performance on this experiment was not good, and some considerations were made about it. The most obvious one is because of the different optimizer, however, upon further research it has been declared that the main reason as to why the accuracy wasn't growing much is because of the learning rate, which was too high.

Experiment #5

Reusing the previous model architecture but incrementing the number of nodes per layer and trying with a high batch size of 256, in order to see the impact it has on the model performance, since there are thousands of samples in the dataset.

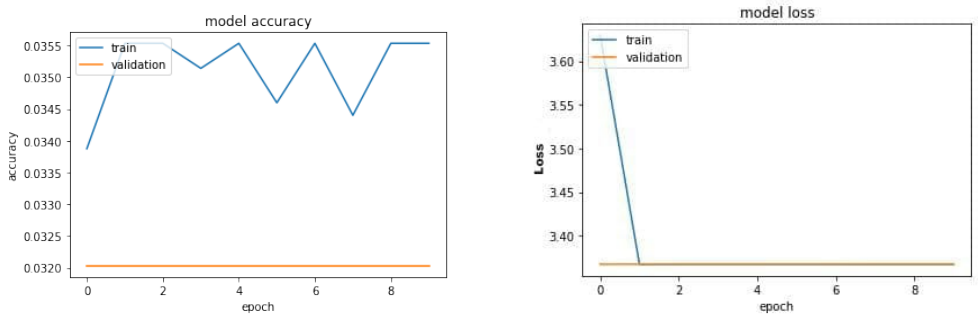


Figure 11: accuracy and loss graphs for the fifth experiment

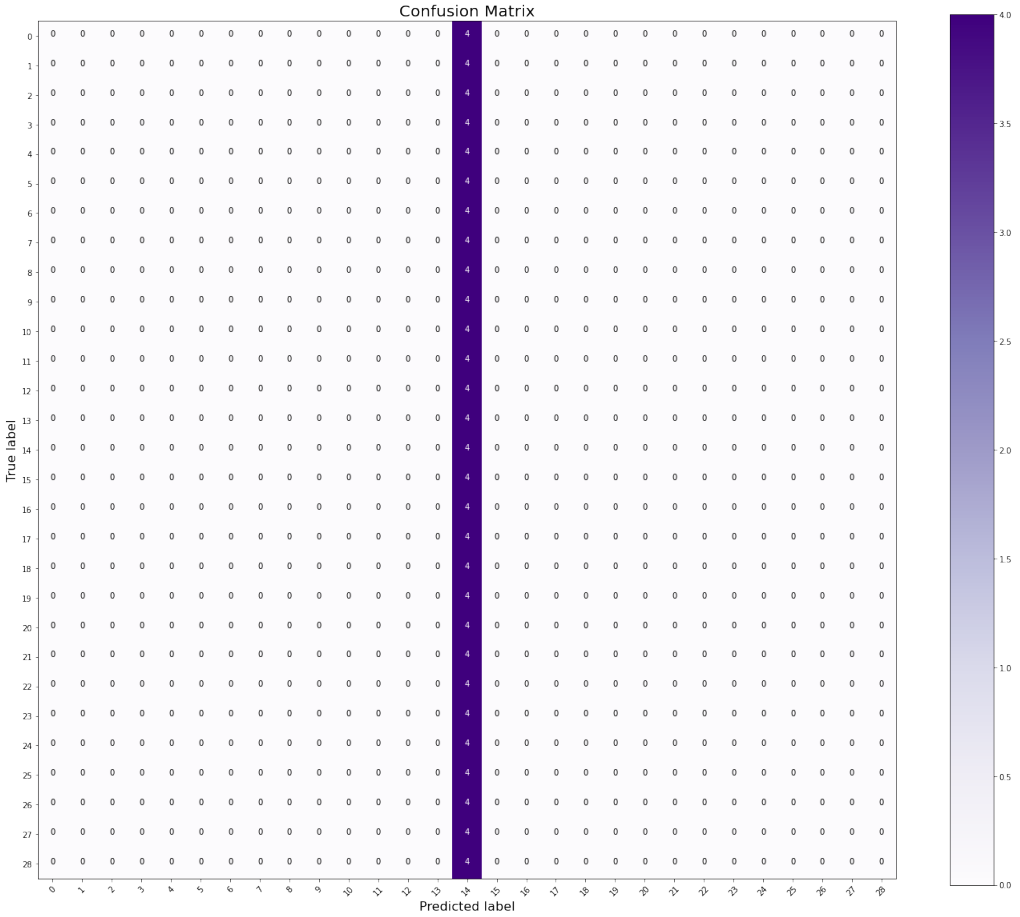


Figure 12: confusion matrix for the fifth experiment.

The training accuracy and loss are respectively 0.0355 and 3.3500 The validation accuracy and loss are respectively 0.0320 and 3.3500 By calling the .evaluate method, the precision of the fourth experiment for the test images is 3.448%. Clearly, a batch size of 256 is not optimal for the model, a higher batch size means that the weights get updated less frequently, and this explains why the accuracy of this experiment is extremely low.

Experiment #6

Just lowering down the batch size for the reasons explained in the previous experiment.

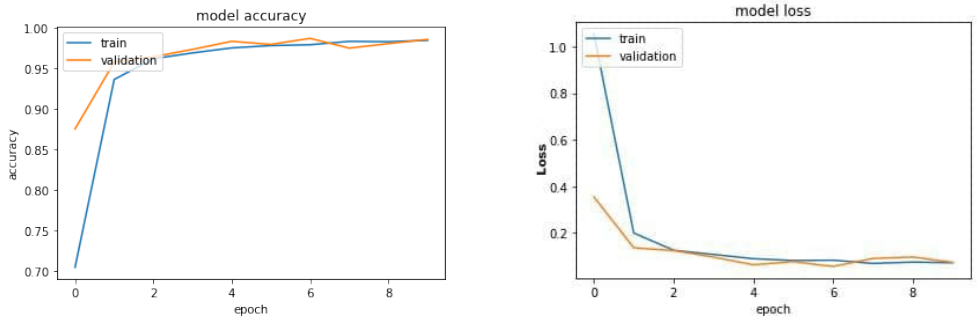


Figure 13: accuracy and loss graphs for the sixth experiment

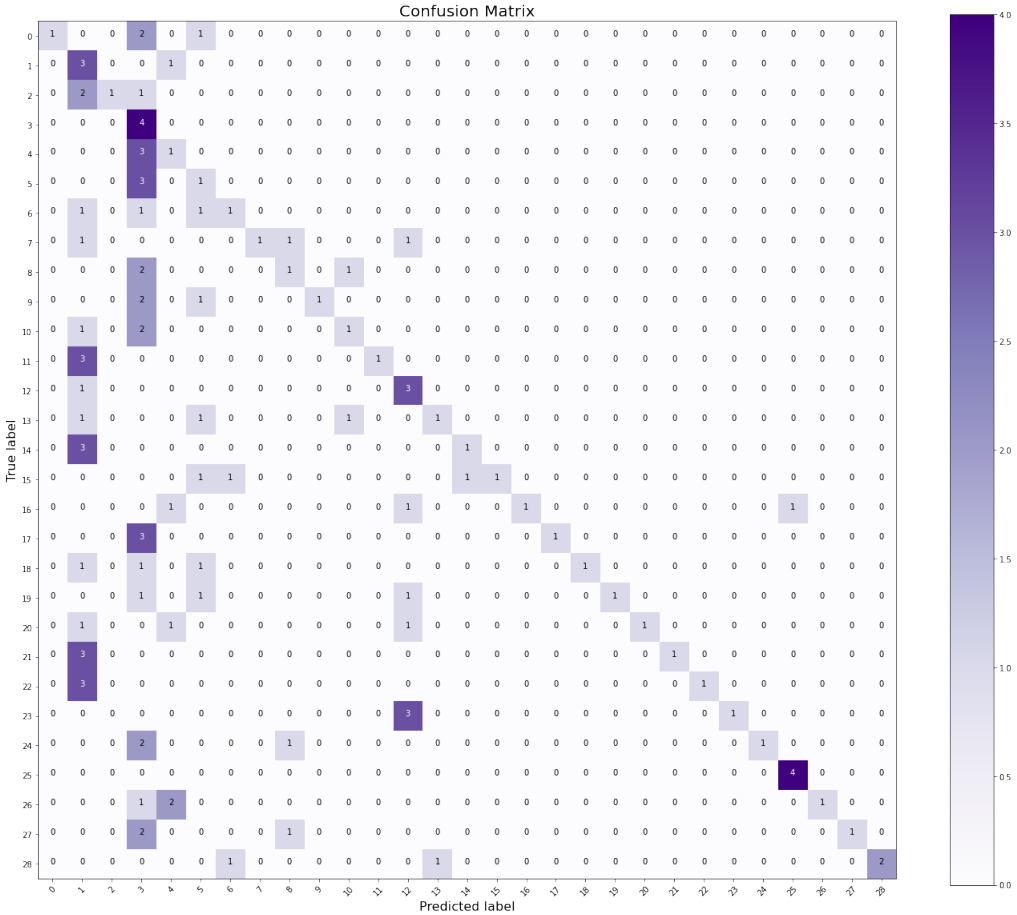
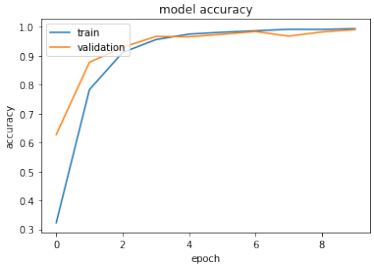


Figure 14: confusion matrix for the sixth experiment.

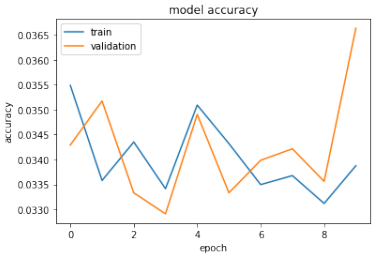
The training accuracy and loss are respectively 0.9847 and 0.0713. The validation accuracy and loss are respectively 0.9859 and 0.0729. By calling the .evaluate method, the precision of the fourth experiment for the test images is 34.483%. The evaluation score is very similar to the one obtained in the third experiment, because essentially the only difference between the third and sixth experiment is in the number of neurons for each layer. Batch size and epochs are also different, but not by much. The subsequent experiments will take place on this kind of model architecture, tuning some hyperparameters in order to get a better performance.

Experiment #7,8,9

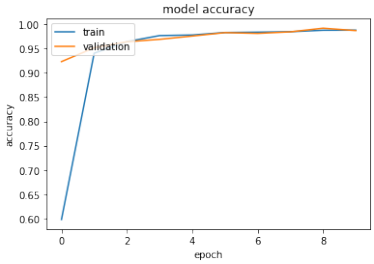
The only thing changed in these experiments was the learning rate, just to determine the impact that the learning rate has on the model, using the adam optimizer, and the batch size which is set to 30.



(a) learning rate = 0.0001

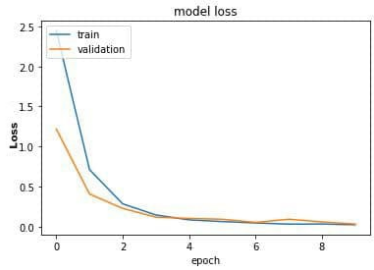


(b) learning rate = 0.01

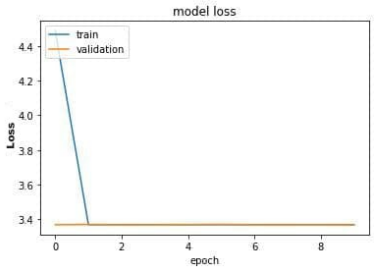


(c) learning rate = 0.001

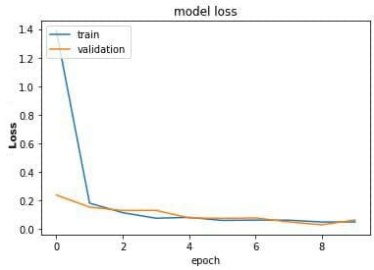
Figure 15: accuracy graphs for the 7th,8th and 9th experiment



(a) learning rate = 0.0001

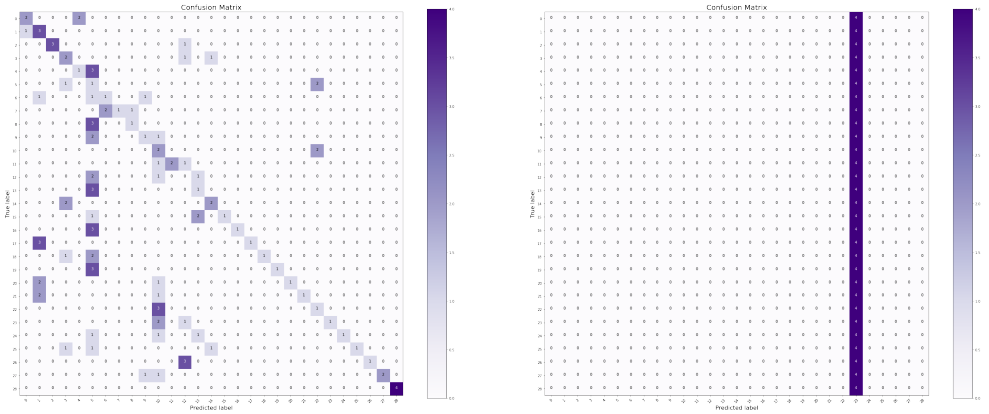


(b) learning rate = 0.01



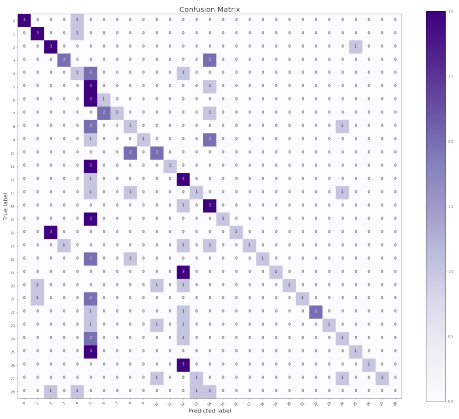
(c) learning rate = 0.001

Figure 16: loss graphs for the 7th,8th and 9th experiment



(a) learning rate = 0.0001

(b) learning rate = 0.01



(c) learning rate = 0.001

Figure 17: confusion matrixes for the 7th,8th and 9th experiment

The results of this experiments were:

- training loss 0.0226 and training accuracy 0.9936, validation loss 0.0315 and validation accuracy 0.9907 with learning rate 0.0001.
- training loss 3.3700 and training accuracy 0.0339, validation loss: 3.3691 and validation accuracy 0.0366 with learning rate 0.01.
- training loss 0.0497 and training accuracy 0.9880, validation loss 0.0631 and validation accuracy 0.9871, learning rate 0.001.

By calling the .evaluate method on the test set, the results were:

- 35.345% accuracy with learning rate 0.0001.
- 3.448% accuracy with learning rate 0.01.
- 37.06% accuracy with learning rate 0.001.

In this experiment it was concluded that using this model architecture and a learning rate of 0.0001 it was possible to achieve an accuracy of 99% on both training and validation. A lower learning rate implies that smaller steps are taken on the gradient descent, which ensures the possibility to find the minimum point. For this particular problem, the local minimum point encountered using a learning rate of 0.0001 appears to be also the global minimum: in some particular cases a low learning rate could get the problem stuck in a local minimum point.

Experiment #10

Even if the training and validation accuracy reported in the previous experiment is above 99%, the evaluation wouldn't go above 37%. At this point some consideration were made about wether the pictures taken personally for the test set were dependent on the color rather than the shape. So, the color map of the pictures was changed from grayscale to RGB.

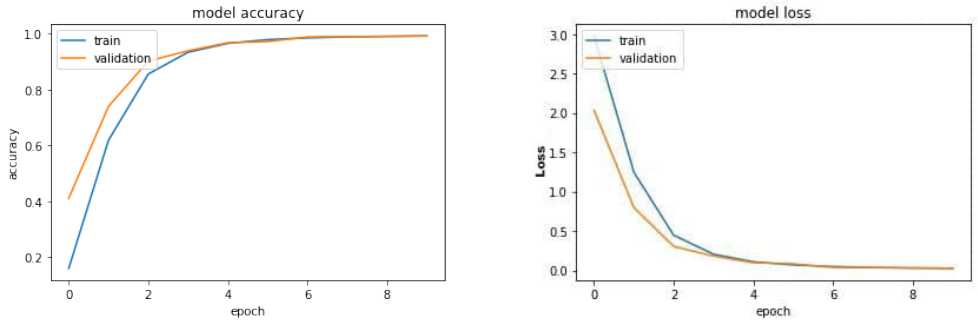


Figure 18: accuracy and loss graphs for the tenth experiment

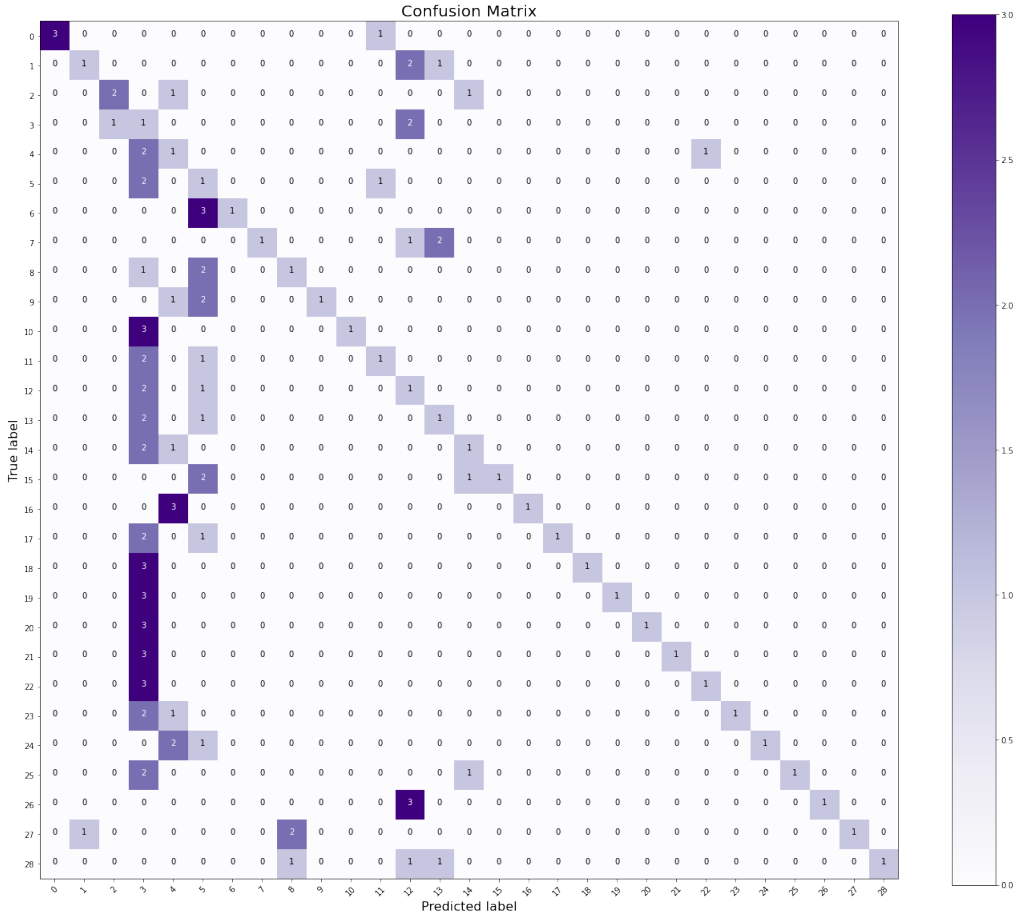


Figure 19: confusion matrix for the tenth experiment.

The training accuracy and loss are respectively 0.9924 and 0.0263. The validation accuracy and loss are respectively 0.9926 and 0.0239. By calling the .evaluate method, the precision of the tenth experiment for the test images is 37.586%. There is a slight improvement compared to the previous experiment, but not an important one.

Experiment #11

Changing the color scheme of the pictures didn't improve the result, so, in this experiment the test set is changed from the pictures taken personally to the test set provided on kaggle.com, the one consisting of just 29 pictures. The model used is the same as the previous, so there are just reported the changes on the confusion matrix and the evaluate method.

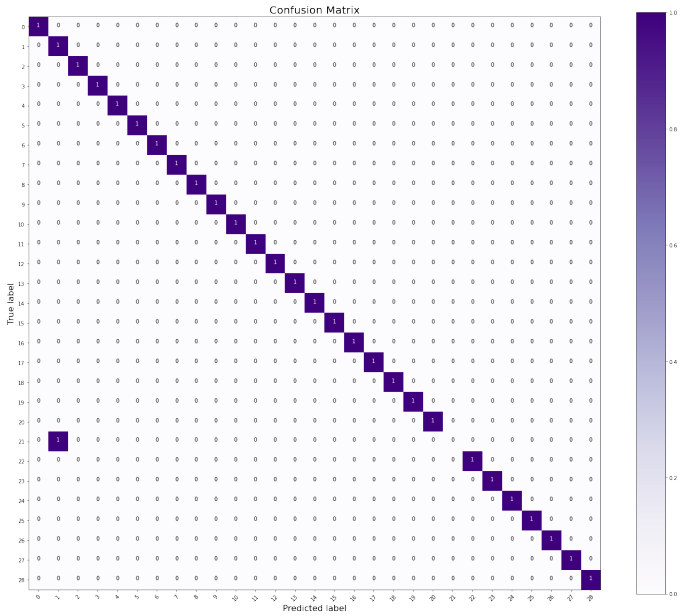


Figure 20: confusion matrix for the eleventh experiment.

By calling the .evaluate method, the precision of the evelnth experiment for the test images is 96.552%.

Conclusion

After all the carried out experiments, it was concluded that the most suitable model architecture was the one reported in the seventh experiment reaching an accuracy of 99%. In the future, this trained model could be implemented in other projects related to the ASL fingerprints recognition.

References

[1] Real-time American Sign Language Recognition with Convolutional Neural Networks

[2] A Convolutional Neural Network to classify American Sign Language fingerspelling from depth and colour images

[3] keras.io/api/

[4] effect of batch size on training dynamics