Recognition of the American Sign Language Alphabet using Deep Learning

Micro Module ECIU Machine Learning

1st Ana Sofia da Silva Nunes
Department of Physics, University of Aveiro, Portugal
Universiade de Aveiro
Aveiro, Portugal
asilvan@ua.pt

***Abstract*—This project presents training a deep learning model to correctly classify the gestures of the American sign language alphabet. Image classification is a common problem approached by the fields of computer vision and machine learning. The main goal of the project was to classify every image in the dataset using neural networks. A conventional neural network is baseline of this investigation, followed by the use of a convolution neural network (CNN) and data augmentation techniques. The final model configuration included augmented data which boosted the validation accuracy to 98.56% and the training accuracy to 99.14%. This result showed that the configuration proposed performed better and reduced overfitting.**

***Index Terms*—Augmentation, Convolutional Neural Network, Machine Learning, Sign language**

https://github.com/ana-sofia-silva/eciu-machine-learning

I. Introduction

Sign language is a visual communication that involves the combination of hand shapes and movements. Image classification is a task in which a program is asked to classify an image it has never seen before into the correct class. For this type of task, Deep Learning methods are particularly well suited for pattern recognition through trial and error. Deep Learning is a subfield of machine learning that focuses on learning successive layers of increasingly meaningful representations [1].

This work is about developing the implementation of deep leaning methods for image classification. The dataset used is American Sign Language Dataset. The dataset with handwritten digits is commonly used for machine learning methods, i.e., it is designed to learn human features for sign interpretations.The dataset consists of 24 letters and its discussion can be found in the follwing section [2].

Deep convolutional neural networks (CNN) perform computer vision tasks when compared to traditional neural networks. These networks are rely on big dataset to avoid overfitting. Overfitting, refers occurs when a network learns a function with very high variance such as to perfectly model the training data [3]. The depths of the CNN, the combination of CNN layers and the augmentation techniques alter the performance of the system. Therefore, this project focus on recognition of static hand gestures. Starting on a conventional neural network, the main goal is to investigate how the validation and training accuracy can be increased.

II. Dataset

This section provides the description of the dataset. The sign language MNIST dataset can be found at Kaggle and it presented in CSV format with labels and pixel values in individual rows. As mentioned earlier, the dataset consists of 24 different letters. Thus, it is a multi-class problem with 24 letter classes except J and Z (these two letters require motion). The data contains a total of 27,455 cases. A selection of images can be seen in Fig. 1.

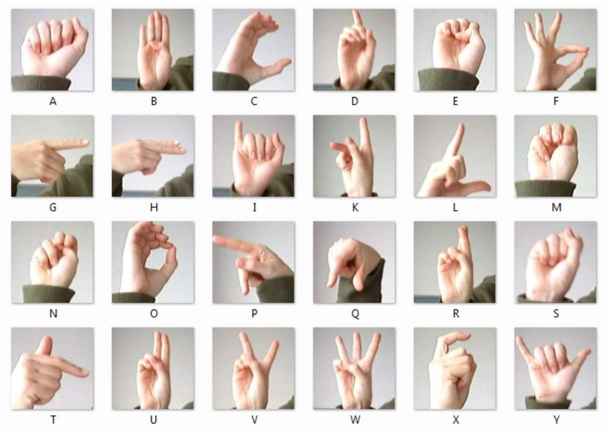

Fig. 1. The manual alphabet used in American Sign Language [2]

Data were provided as pixel-to-pixel intensity [0-255] class-distributed XLS files. Data preprocessing steps included converting said data to image format using Python's open source libraries Pandas, Numpy, and others to obtain images in PNG format with 28x28 grayscale [2] .

III. Implementation

A. Training and Validation Data and Labels

In Deep Learning, the images used must meet certain requirements. Each image is denoted by x and each of its associated labels by y. Also, both x and y are used to train the model and then a separate set of x and y are used for validation [1]. For validation, the performance of the model must be evaluated after training. Therefore, 4 data segments are considered:

1) x_train: Images used for training the neural network
2) y_train: Correct labels for the x_train images, used to evaluate the model's predictions during training
3) x_valid: Images set aside for validating the performance of the model after it has been trained
4) y_valid: Correct labels for the x_valid images, used to evaluate the model's predictions after it has been trained.

IV. Loading the Data Into Memory

The dataset used is in CSV (Comma Separated Values) format and can be loaded as a DataFrame with Pandas. Pandas uses the function *read_csv* to return the DataFrame: *train_df* the training DataSet and *valid_df* the validation DataSet.

A. Exploring the Dataset

When you examine the dataset, the DataFrame represents each row as an image, which has a label column and also 784 values representing each pixel value in the image. You can see some of the first values in the following table,

TABLE I
First images of the DataFrame.

	label	pixel1	pixel2	...	pixel784
0	3	108	118	...	202
1	6	155	157	...	149
2	2	187	188	...	195
...	...	...	...	...	163

The y_valid and y_train variables are defined and the labels are deleted from the original data set. For the variables x_valid and x_train the values of DataFrame are defined directly. So there are 27,455 frames for training and 7,172 frames for validation, both with 784 pixels.

In order to implement the system, python was used as the main programming language and Google Colab was used to write and run code. Matplotlib was used to visualize model accuracy plots.

B. Preparing the Data for Training

In Deep Learning, it is common practice to transform the data with certain requirements to simplify the classification problem. There are two tasks that should be performed before training: normalizing the image data and categorizing the labels. To make it easier for the model to work with the image data, the data must be normalized. To normalize the data (convert integer values to floating point values between 0 and 1) all pixel values must be divided by 255. The *keras.utils.to_categorical* can be used to categorically encode the labels. Note that *keras* must be imported and the number of categories must be defined.

C. Model

Once the data preparation for both training and validation has been successfully completed, the model can be built. First, a sequential model is considered. The model has a dense input layer with 512 neurons, uses the activation function *relu*, and expects an input shape of 784 - number of pixels per image. The second layer, similar to the previous one, has 512 neurons and uses the same activation function. And for the dense output layer with neurons equal to the number of classes, the activation function *softmax* is used. For composing the model the categorical cross entropy to reflect that it should fit into one of many categories, and measuring the accuracy of the model.

D. Results

This section describes the results obtained using a the configuration describe above. The resulting training accuracy is quite high while the validation accuracy is not. Table I bellow presents the accuracy for training and validation are presented for *Relu* and *Sofmax* in the last layer.

TABLE II
Tranning and Validation accuracy

Accuracy	Trainning	Validation
Sofmax	0.9797	0.8337
Relu	0.9923	0.8352

The model learned to categorize the training data, but performed poorly on new data for which it has not been trained. Summarily, it is memorizing the data set but fails to gain a robust and general understanding of the problem - sign of overfitting. The accuracy graph is shown on Fig. 2.

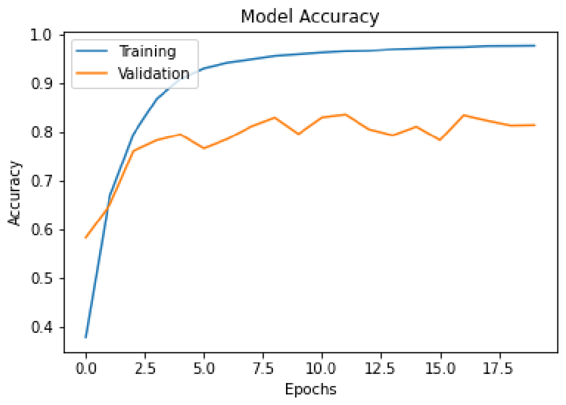

Fig. 2. Model accuracy.

V. Convolutional Neural Network (CNN)

This section provides the description of the CNN configuration that was used. As mention previously, the images are composed by list of 748 pixels and in order to use CNN to detect the features they need to be reshaped - 28×28 pixel. The creation of the convolutional model includes many different layers such as *Conv2D*, *BatchNormalization*, *Dropout*, *Dense* among others. The compilation of the model and its training it is similar to what was describe in the previous sections.

A. Results

The training and validation accuracy acquired was, respectively, 0.9999 and 0.9683, for 20 epochs. The training accuracy is very high, and the validation accuracy has improved as well. The accuracy graph is shown on Fig. 3.

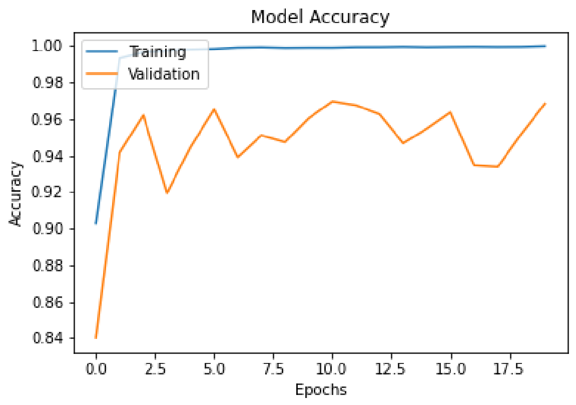

Fig. 3. Model accuracy using CNN.

In this section, new types of layers were used to implement a CNN. This implementation performed better than the model is significantly improved.

VI. Data Augmentation and Deployment

Although the use of CNN improved the performance of the model implemented, the validation accuracy is still bellow the training accuracy.The validation error must continue to decrease with the training error and Data Augmentation is a powerful method of achieving this. Augmentation techniques provided a more comprehensive set of possible data points, thus minimizing the distance between the training and validation set [3].

A. Model

For the creation of the model the same approach presented before was used. Before the model was compiled a set of techniques were applied by the use of Keras, which has an image augmentation class called *ImageDataGenerator*. Some basic geometric transformations were applied on the dataset. The set of manipulations included: randomly rotate images in the range (degrees, 0 to 180), randomly zoom image, randomly shift images horizontally and vertically and randomly flip images horizontally. The class *ImageDataGenerator* provides the use of *Batch Size* to control the stability of training process. The batch size is, by definition, the number of examples from the training dataset used in the estimate of the error gradient and it can influence the learning algorithm. The details of its this use can be found in *"How to Control the Stability of Training Neural Networks With the Batch Size"* [4].

B. Results

The training and validation accuracy acquired was, respectively, 0.9914 and 0.9856, for 20 epochs. The accuracy of augmented model was higher than non-augmented model at each epoch. This means that the model is no longer overfitting in the way it was in previous sections. Therefore is generalizing better, i.e., making better predictions on new data. The accuracy graph is shown on Fig. 4.

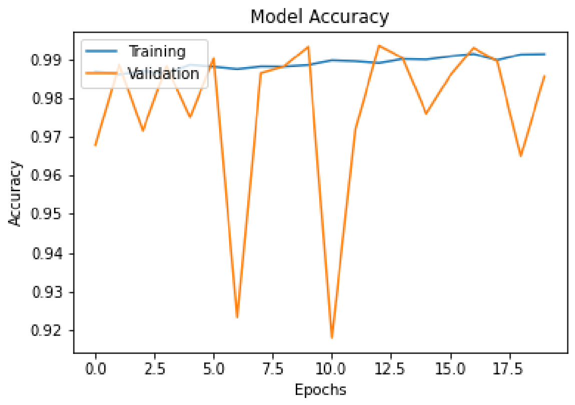

Fig. 4. Model accuracy with augmented data.

VII. Results

So far, we have compared three strategies, i.e., one using a baseline neural network; other directly with CNN without taking any pre-processing; and, finally a CNN with augmented data. In order to provide a more comprehensive comparison between these strategies, Tables III exhibits the main results.

TABLE III
Summary of the Results

Accuracy	Trainning	Validation
Baseline	0.9797	0.8337
CNN without augmentation	0.9923	0.8352
CNN with augmentation	0.9914	0.9856

VIII. Conclusion

This project explores traditional and Deep Learning techniques that can be used in image classification. It also analyzes the effect of data augmentation. Recalling the results obtained, the following conclusions can be withdrawn: the use of CNN is advantageous for this type of systems, and the use of augmentation techniques can highly impact the accuracy of the validation dataset - validation accuracy 98.56% and training accuracy 99.14%. The accuracy reached with the final configuration, CNN plus augmented data, performed better than the baseline by reducing over fitting.

If given more time, more complex architecture and more varied datasets shall be explored. It is suggested to replicate the same study with: increased number of epochs and different augmentation methods; and, the use of pre-trained models, such ResNet50 or VGG16 [5].

References

[1] F. Chollet, *DEEP LEARNING with Python*. Shelter Island, NY: Manning Publications Co., 2018.

[2] J. Brownlee, "Sign Language MNIST." https://www.kaggle.com/datamunge/sign-language-mnist/, 2018. [Online; accessed 5-January-2021].

[3] K. Shorten, C., "A survey on Image Data Augmentation for Deep Learning." https://link.springer.com/article/10.1186/s40537-019-0197-0#citeas, 2019. [Online; accessed 28-January-2021].

[4] J. Brownlee, "How to Control the Stability of Training Neural Networks With the Batch Size." https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/, 2020. [Online; accessed 30-January-2021].

[5] P. Huilgol, "Top 4 Pre-Trained Models for Image Classification with Python Code." https://www.analyticsvidhya.com/blog/2020/08/top-4-pre-trained-models-for-image-classification-with-python-code/, 2020. [Online; accessed 2-February-2021].