Sign-Language MNIST and Classification with Machine Learning

Diogo Marto
DMAT/DETI
Universidade de Aveiro
diogo.marto@ua.pt

Pedro Azevedo
DMAT/DETI
Universidade de Aveiro
pgca@ua.pt

***Abstract*—This paper explores the training outcomes of a neural network and a convolutional neural network aimed at recognizing hand-digit images. We will also compare both types of models and apply some techniques to increase the performance of them.**

***Index Terms*—Neural Networks, Sign-Language MNIST, CNN**

This report was made for the FAA course with Petia Georgieva (petia@ua.pt) as its instructor.

I. Introduction

Around the world, millions of people suffer speech or hearing impairment. More than ever, it is important to use the technology we have in our reach to improve commodity and provide the means to interconnect those with hearing loss and/or speech impairment and the rest of non-sign Language speakers. In that order, using the capabilities of machine learning, we've developed a system able to predict with high accuracy sign language signals from images. We've trained our models based on the dataset from MINST about American Sign Language using Neural Networks and Convolutional Neural Networks techniques.

II. Dataset Description and Transformations

American Sign Language is expressed by movements of the hands and face. It is the primary language of many North Americans who are deaf and hard of hearing and is used by many hearing people as well. The dataset format is patterned to match closely with the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27455 cases) and test data (7172 cases) are pre-split. They are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2….pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255. The original hand gesture image data represented multiple users repeating the gesture against different backgrounds. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest as shown in Fig. 1.

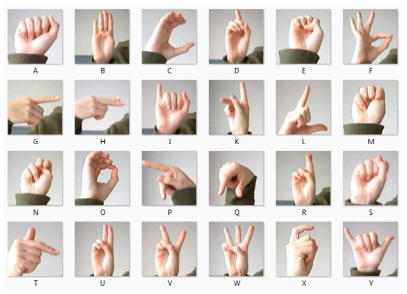


Fig. 1: Original Pictures Examples.

To create new data, an image pipeline was used based on ImageMagick and included cropping to hands-only, gray-scaling, resizing, and then creating at least 50+ variations to enlarge the quantity as shown in Fig. 2.

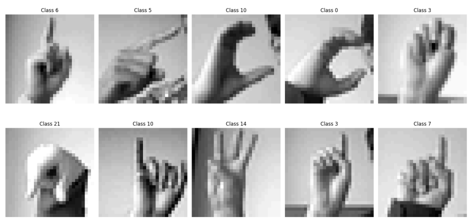


Fig. 2: Examples of the Dataset

As a way to get a better feeling about the data we would be working with, we analyzed class distribution across the training dataset to detect any discrepancy in the dataset that could penalize the generalization in the training of our model to all 24 desired classes. From the obtained histogram on Fig. 3, we can see the dataset is distributed evenly among all classes, not requiring data balancing.

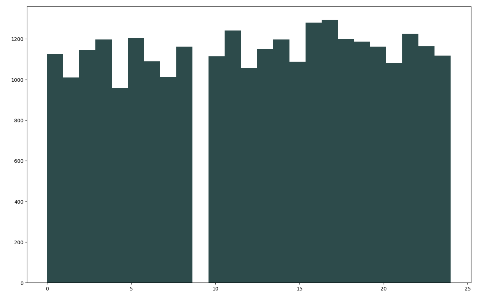


Fig. 3: Distribution of Classes and their count (Train)

From the obtained histogram on Fig. 4, we can see that, unlike the training dataset, the test dataset is more unbalanced as such a metric like F1-score is more suitable for our problem but regardless we also provide for our models accuracy metrics as a way to compare with models from other authors.

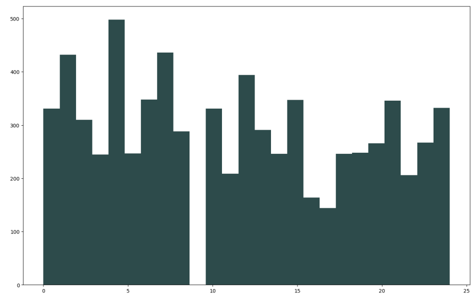


Fig. 4: Distribution of Classes and their count (Test)

We also applied normalization to the pixels such that their values are now from [0,1] when before they were from [0,255]. This generally helps the speed of convergence of CNNs.

To enhance the generalization capability of the models, data augmentation was applied in some models using the ImageDataGenerator class in Keras. In the models of the NN (where we didn't use TensorFlow so we couldn´t use ImageDataGenerator), we made a similar function using the PIL library in Python, adjusting the brightness, contrast, and position of the image. The augmentation strategy included small geometric transformations such as random rotations within a range of ±10 degrees, zooming by up to 10%, and horizontal and vertical shifts by up to 10% of the image dimensions. These transformations simulate real-world variations while preserving the semantic integrity of the input data. This augmentation pipeline was specifically tailored to introduce variability in the training data while maintaining the integrity of the sign language gestures, to improve the model's robustness and reduce overfitting.

III. State of the Art

[1] The study conducted by Ranjeet Jain focused on developing a convolutional neural network (CNN) model for American Sign Language classification. The architecture employed a relatively small CNN with 85912 parameters, designed to balance computational efficiency and accuracy. The model was trained without incorporating data augmentation techniques, which could limit its ability to generalize effectively to unseen variations in the dataset. Despite this limitation, the model achieved a commendable accuracy of 0.8387 (83.87%) but a validation of accuracy of 0.999 (99.90%). Since the construction of the validation dataset was made by splitting the training dataset this suggests that there might be some differences between the training and the test dataset. This result demonstrates the feasibility of lightweight architectures for sign language classification, albeit with room for improvement, particularly in enhancing generalization through techniques like data augmentation.

[2] Sayak Dasgupta proposed a robust CNN-based approach with 319352 parameters to American Sign Language classification, achieving an impressive accuracy of 99.4%. Their methodology was distinguished by the integration of data augmentation, which artificially expanded the diversity of the training dataset, thus enhancing the model's robustness to variations in input data. Additionally, the use of an adaptive learning rate optimization algorithm allowed the network to dynamically adjust its learning process, contributing to more efficient convergence. This combination of techniques highlights the critical role of data preprocessing.

[3] Vítor Santos and Tiago Pereira developed a high-performing CNN model for sign language classification with 8262649 parameters, achieving an accuracy of 93% without data augmentation and 97.53% with data augmentation. The model leveraged a significantly larger architecture compared to other approaches, enabling it to capture more complex features from the input data. Additionally, using an adaptive learning rate further enhanced training efficiency and accuracy. In parallel, a secondary experiment employed a K-Nearest Neighbors (KNN) classifier, achieving an accuracy of 84%. This demonstrates the flexibility of the dataset for supporting both deep learning and traditional machine learning approaches. However, the CNN significantly outperformed the KNN in capturing nuanced features necessary for accurate classification.

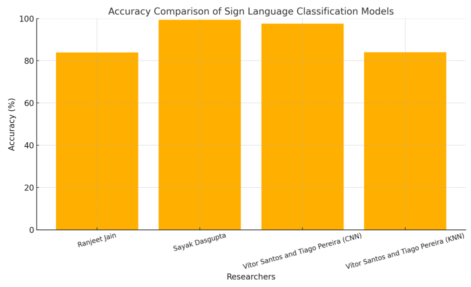


Fig. 5: Comparasion between models

Unfortunately, we didn't find a Neural Network applied to this problem so we don't have a direct comparison. But we hope our proposed NN model can become a reference.

IV. Models

A. Neural Network

a) *NN description:* A Neural Network is a computational model inspired by the way biological neural systems work, such as the human brain. It is a key component in machine learning specifically in deep learning, where it is used to recognize patterns, make predictions, or perform tasks like image recognition, natural language processing, and more. The basic unit of a neural network is a neuron (also known as a node), analogous to the neurons of the brain. Each neuron will take some input, pass it through a mathematical function, and pass it along to another neuron. This will create multiple layers of neurons. A basic structure will include three types of layers: an input layer, that takes the data in, and one or more hidden layers, where its neurons will process this data and try to extract patterns or features from the dataset. Lastly, there is an output layer, where the model will create its last prediction or classification, depending on the goal. Neurons have connections between each other, and these connections can have weights and biases associated with each neuron, and these are adjusted during the model's training to improve the network's performance.

First model:

The first approach was to just create a simple NN and to understand what kind of accuracy it would achieve. We created a simple MLPClassifier using scikit-learn's neural network library. This model would have a single hidden layer with 100 neurons and a limit of 300 iterations. Since MLPClassifier trains iteratively, updating its parameters along each iteration, it would stop when it finds convergence. Using the data directly from the dataset, without any normalization or data augmentation, this model achieved an underwhelming 18.61% accuracy and 0.156 F1-Score (15.6%) as seen in Fig. 6. The training accuracy for the model achieved a result of 34.52%, showing the model had difficulty finding patterns in the data.

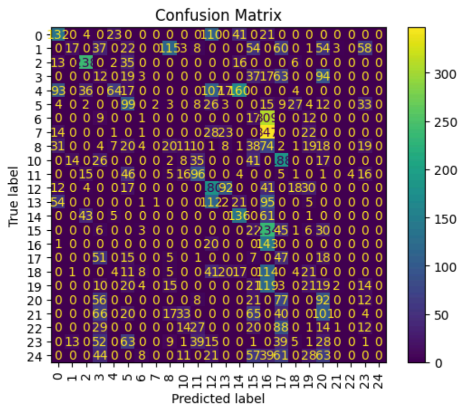


Fig. 6: 1st Model's Confusion Matrix

Looking at the loss curve for this model in Fig. 7, we can see that it converges after around 100 iterations, but maintains a loss of about 2%, which complements the bad accuracy achieved by the model.

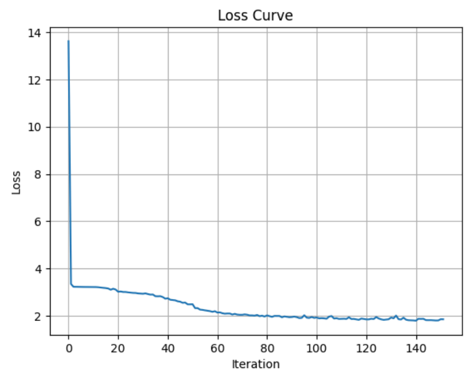


Fig. 7: Loss Curve of the 1st Model of the NN

Second model:

For the second model, we took some steps to improve the model's performance. We applied data normalization, transforming the pixel values from range [0, 255] to range [0, 1], which will help the model's performance. We also enabled early_stopping, meaning the model would end training when its validation performance stops showing improvement, to avoid overfitting to the training set. We also ran some tests, to evaluate the best values for hidden layers, reaching 2 layers, a first one with 256 neurons and a second with 128 neurons, as described in Table I.

TABLE I: Comparison between hidden layers

Hidden Layers	Accuracy	F1-score
(64, 32)	73.38%	0.732
(128, 64)	74.80%	0.7215
(256, 128)	77.96%	0.7577
(512, 256)	73.93%	0.7134

We looked for combinations of powers of 2 and found that going above these numbers would only lead to a much computationally heavier model that gets the same or lower values in accuracy.

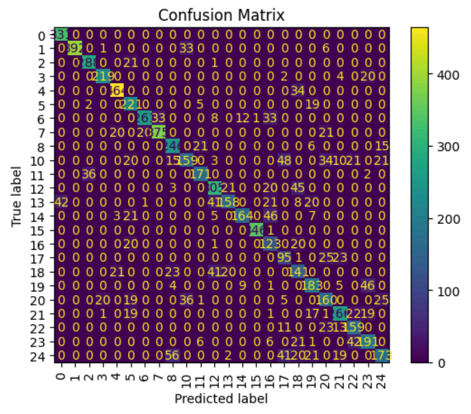


Fig. 8: 2nd Model's Confusion Matrix

This model performed with an accuracy of 77.96% and a f1-score of 0.7577 (75.77%). This is an improvement compared to the first model, as seen in Fig. 8, but shows clear signs of overfitting since the training accuracy for this model was 100%. With overfitting being the problem, we decided to manipulate the alpha value, thus impacting the existing loss curve, as seen in Fig. 9 .

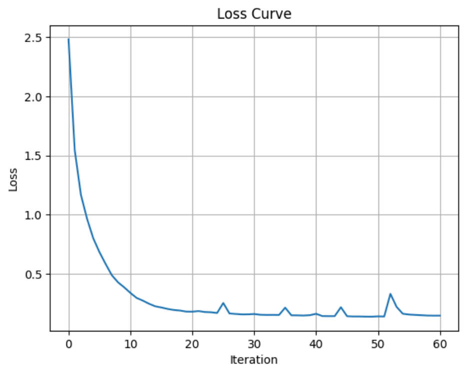


Fig. 9: Loss Curve of the 2nd Model of the NN

Third model: We added a hyper-parameter called alpha, which represents the L2 regularization term, used to manipulate the size of the weights associated with each node. Alpha by default is 0.0001. After experimenting with multiple values of alpha (0.0001, 0.1, 0.2, 1, 2, 3, 10), we decided to use 0.1 as alpha. A higher value would increase the risk of underfitting the model, and smaller values displayed clear overfitting. Although this value still shows signs of overfitting, it had the best accuracy out of all the values of alpha with a value of 77.86%.

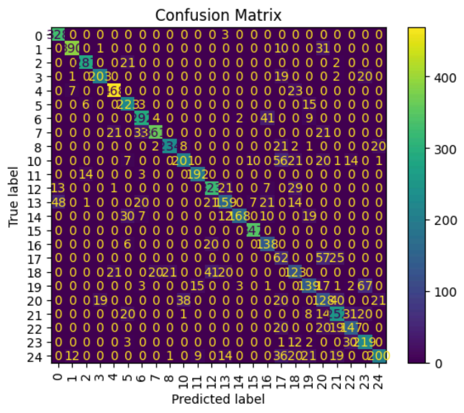


Fig. 10: Confusion Matrix of the Fourth Model

The addition of the alpha value to the model also has a clear impact on the loss curve, reducing the number of iterations, as seen in Fig. 11. Given the existing limitations of the model, we turned to data augmentation to improve our model's performance.

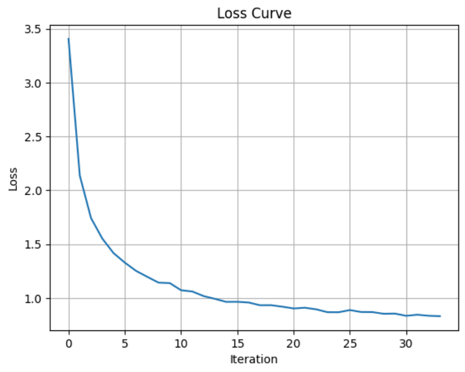


Fig. 11: Loss Curve of the 3rd Model

Fourth model:

To further improve our model and produce a model capable of accurately predicting hand signs used in sign language, we used data augmentation to increase our training dataset and train a model more generalized, capable of having a better performance when predicting the classes of the test dataset. To do this, we added 3 new versions of each image to the training set, manipulating brightness, contrast, and overall position of the hand sign by shearing the original version of the image. This allowed the model to have a more complete training dataset. Maintaining all other configurations of the 2nd model, it reached a 99.48% accuracy and 0.9949 f1-score. This, together with a cross-validation accuracy of 0.9983 (99.83%), would mean our model has reached its peak performance, although other improvements could still be considered.

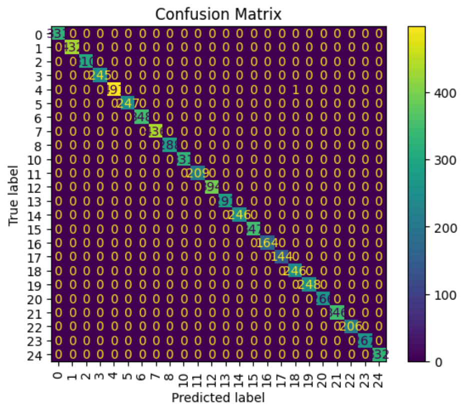


Fig. 12: Confusion Matrix of the Fourth Model

We also diversified the testing set, by performing the same manipulations to the images as we did in the training set, to learn if this almost perfect accuracy would be confirmed. The augmented testing set achieved an accuracy of 99.48%, having an F1-Score of 0.9979. There isn't a big difference in terms of loss comparing to the 2nd model, reaching convergence in about 30 iterations, as described in Fig. 11

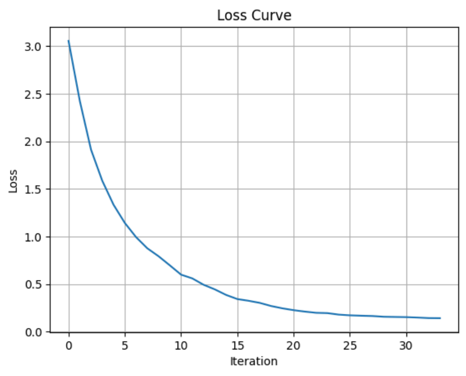


Fig. 13: Loss Curve of the Fourth Model

B. *Convolutional Neural Network*

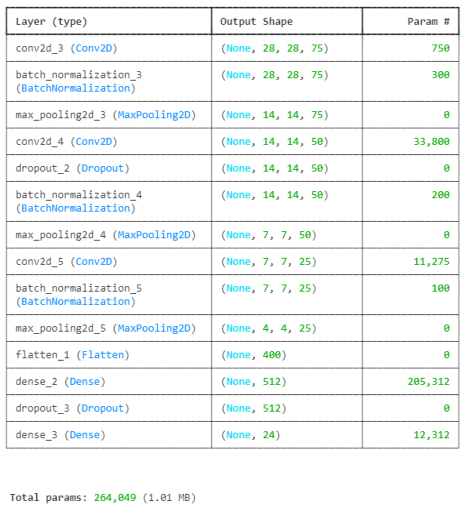
a) *CNN description:* A Convolutional Neural Network (CNN) is a specialized type of artificial neural network specifically designed for processing data that has a grid-like topology, such as images. The core building block of a CNN is the convolutional layer, which applies filters (kernels) to the input data, effectively detecting patterns and features like edges, corners, and textures. These features are then aggregated through pooling layers, which reduce dimensionality while retaining essential information. The final layers of a CNN typically consist of fully connected layers that perform classification or other high-level tasks based on the extracted features. This hierarchical structure allows CNNs to learn increasingly complex representations of the input data, making them highly successful in image recognition such as this problem. For the architectures of CNNs, we propose we use the following layers:

- Convolutional Layer (Conv2D): Extracts spatial features from input images by applying learnable filters (kernels) that scan over the image. Each filter activates specific patterns, such as edges or textures, helping the model learn hierarchical features.
- Batch Normalization: Normalizes the outputs of a layer to have zero mean and unit variance, improving training stability and convergence.
- Max Pooling (MaxPool2D): Down-samples the feature maps by selecting the maximum value from patches of the input, reducing spatial dimensions. Helps in capturing dominant features while reducing computational complexity and the risk of overfitting.
- Dropout: Randomly sets a fraction of input units to zero during training to prevent overfitting. Forces the network to learn more robust features by not relying on specific neurons.
- Dense Layer: A Fully connected layer where each neuron is connected to all neurons in the previous layer, enabling high-level feature representation. Essential for integrating extracted features and performing classification.
- Dense with Softmax Activation: The final layer in multi-class classification, converting raw output scores into probability distributions across target classes. Ensures that the sum of predicted probabilities equals 1, allowing the most likely class to be selected.

For our models, we built the CNNs with the Python library TensorFlow, this library splits training into epochs and batches for our graphs of loss and accuracy curve the x-axis represents the epochs. All models were also compiled using Adam has the solver.

First model:

For our first approach, we decided based on the state of the art to construct a CNN with the architecture shown on Fig. 14. This is a decently large model with 264079 parameters.



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 75)	750
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 75)	300
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 75)	0
conv2d_4 (Conv2D)	(None, 14, 14, 50)	33,800
dropout_2 (Dropout)	(None, 14, 14, 50)	0
batch_normalization_4 (BatchNormalization)	(None, 14, 14, 50)	200
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 50)	0
conv2d_5 (Conv2D)	(None, 7, 7, 25)	11,275
batch_normalization_5 (BatchNormalization)	(None, 7, 7, 25)	100
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 25)	0
flatten_1 (Flatten)	(None, 400)	0
dense_2 (Dense)	(None, 512)	205,312
dropout_3 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 24)	12,312

```
Total params: 264,049 (1.01 MB)
```

Fig. 14: Architecture of first CNN

The training of this CNN had no regularisation, no adaptive learning rate, or data augmentation based on ImageGenerator and a learning rate of 0,001. Based on Fig. 15 we can see that the training loss converged very fast to a value near 0 while the validation loss had chaotic jumps. This suggests that small changes between epochs of training can lead to massive differences in validation and we have overfitting. Furthermore, we observed that retraining the model returns very different values for accuracy and f1-score, namely, we observed that accuracy can be between 90.90% and 96.16%, and f1-score 0.9019 and 0.9587.

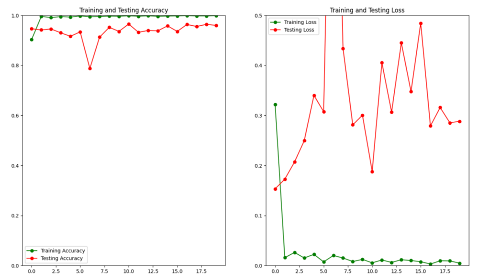


Fig. 15: Loss curve and accuracy of first CNN

Some techniques:

Due to the poor and unstable performance of the model we applied 2 techniques to help combat the weird behavior namely:

- Adaptive Learning Rate: Adaptive learning rate techniques dynamically adjust the learning rate during training, enabling faster convergence and improved performance, making training more efficient and robust.
- Data Augmentation: Data augmentation expands the training dataset by applying transformations such as rotation, scaling, and shifting to simulate real-world variations. These augmented samples enhance the model's ability to generalize, reducing overfitting and improving performance on unseen data.

For our problem, the majority of the gains are due to data augmentation which combats overfitting but the adaptive learning rate allows us to reduce the learning rate on our last epochs to squeeze out a bit more accuracy.

When we retrain our model 1 with these techniques we get perfect results as shown in Fig. 16 where the accuracy on test converges to 100%. Retraining the model can give small variations in the accuracy of a maximum of 0.4%. Since the accuracy is 100% the F1-score is 1.

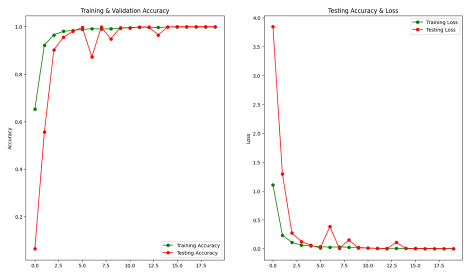


Fig. 16: Loss curve and accuracy of second CNN with data augmentation and adaptive learning rate

The confusion matrix on Fig. 17 only has non-zero entries on diagonals which is to be expected since accuracy is 100%.

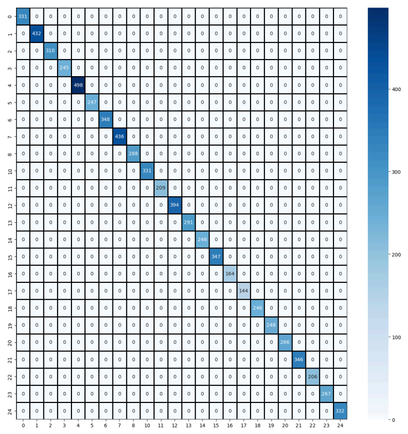
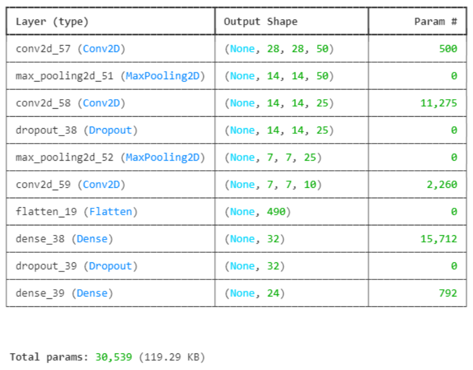


Fig. 17: Confusion Matrix for first CNN with data augmentation and adaptive learning rate

Second model:

For our second model, we wanted to explore a smaller model and regularisation both of which reduce overfitting. The proposed architecture for this model is shown in Fig. 18 and it has L2 regularisation on the dense layers with a strength of 0,01. This architecture is the result of iterations where increasing the L2 strength dropped the accuracy on both training and test data but reduced their difference thus combating overfitting and decreasing the size of the CNN which resulted in worse accuracy but less overfitting namely the chaotic behavior seemed to lessen with one less convolution layer compared to model 1. The proposed architecture is the result of this iteration and it achieved an accuracy of 91,64% and an F1-score of 0.9106, the total size of this model is 30539 parameters which is much less than the 264 thousand of model 1.



Layer (type)	Output Shape	Param #
conv2d_57 (Conv2D)	(None, 28, 28, 50)	500
max_pooling2d_51 (MaxPooling2D)	(None, 14, 14, 50)	0
conv2d_58 (Conv2D)	(None, 14, 14, 25)	11,275
dropout_38 (Dropout)	(None, 14, 14, 25)	0
max_pooling2d_52 (MaxPooling2D)	(None, 7, 7, 25)	0
conv2d_59 (Conv2D)	(None, 7, 7, 10)	2,260
flatten_19 (Flatten)	(None, 490)	0
dense_38 (Dense)	(None, 32)	15,712
dropout_39 (Dropout)	(None, 32)	0
dense_39 (Dense)	(None, 24)	792

```
Total params: 30,539 (119.29 KB)
```

Fig. 18: Architecture of second CNN

The training of this CNN has no adaptive learning rate, or data augmentation based on ImageGenerator and a learning rate of 0,001. Based on Fig. 19 we can confirm that overfitting is much less pronounced compared to model 1.

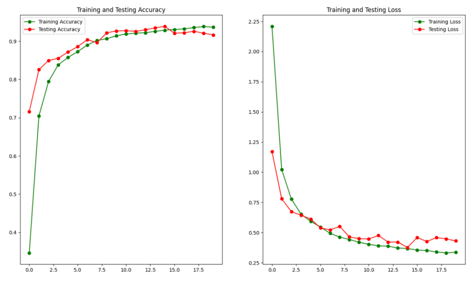


Fig. 19: Loss curve and accuracy of second CNN

If we apply an adaptive learning rate and data augmentation we can increase the accuracy to 97.65% and the F1-score to 0.9737. On Fig. 20 we can see a weird phenomenon where the testing accuracy is higher than the training accuracy, this is most likely because the generated images are different between epochs so the training data is somewhat unseen but due to the variations on the generation of images, they have higher variance than the test dataset and the model can perform better on the test dataset.

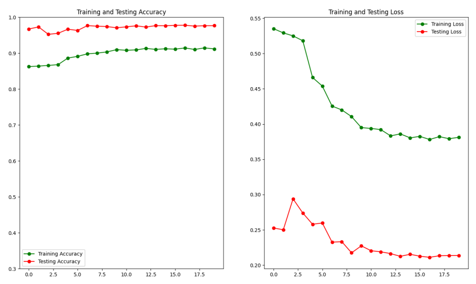


Fig. 20: Loss curve and accuracy of second CNN with data augmentation and adaptive learning rate

The confusion matrix on Fig. 21 shows that the model misclassified class 11 by class 12 nearly 25% of the time and class 16 by class 9 nearly 34% of the time. For the remaining classes, the error isn´t as high, and some always got classified correctly.

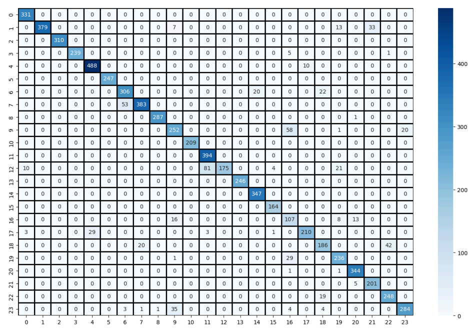
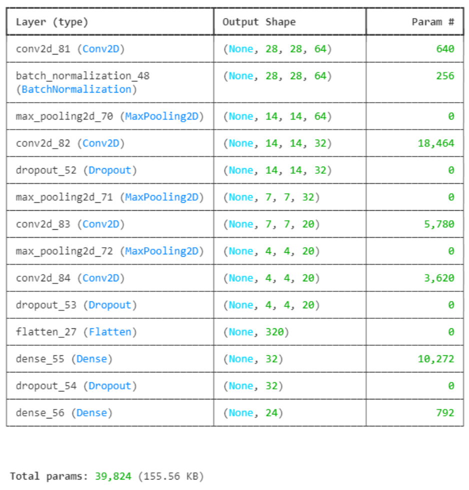


Fig. 21: Confusion Matrix for second CNN

Third model:

For this model, we wanted a model with near-perfect accuracy but much smaller than model 1, since model 2 wasn't able to achieve >99% accuracy. The proposed architecture shown on Fig. 22 has 39824 parameters which is 6.6x less than model 1. In comparison to model 2, we added the convolution layer we removed thus the slight increase in parameter count, this means that without data augmentation will get overfitting but for this model, we made the consideration that we are using it with data augmentation on training. We obtained an accuracy of 99.90% and a F1-score of 0.9989.



Layer (type)	Output Shape	Param #
conv2d_81 (Conv2D)	(None, 28, 28, 64)	640
batch_normalization_48 (BatchNormalization)	(None, 28, 28, 64)	256
max_pooling2d_70 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_82 (Conv2D)	(None, 14, 14, 32)	18,464
dropout_52 (Dropout)	(None, 14, 14, 32)	0
max_pooling2d_71 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_83 (Conv2D)	(None, 7, 7, 20)	5,780
max_pooling2d_72 (MaxPooling2D)	(None, 4, 4, 20)	0
conv2d_84 (Conv2D)	(None, 4, 4, 20)	3,620
dropout_53 (Dropout)	(None, 4, 4, 20)	0
flatten_27 (Flatten)	(None, 320)	0
dense_55 (Dense)	(None, 32)	10,272
dropout_54 (Dropout)	(None, 32)	0
dense_56 (Dense)	(None, 24)	792

```
Total params: 39,824 (155.56 KB)
```

Fig. 22: Architecture of third CNN

On Fig. 23 we can see that without data augmentation our model overfitted as expected but it still achieved better accuracy than our larger model 1 under the same conditions getting an accuracy of 96.52% and F1-score of 0.9660.

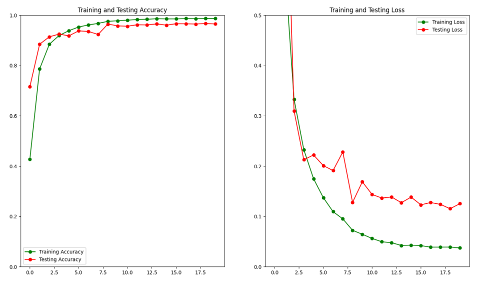


Fig. 23: Loss curve and accuracy of third CNN

On Fig. 24 we can see that the model plus data augmentation converged almost on and the same phenomenon occurred as in model 2 where test accuracy is slightly higher at 99.90% and the training accuracy is 98.15%, based on these values we can say that overfit didn't occur.

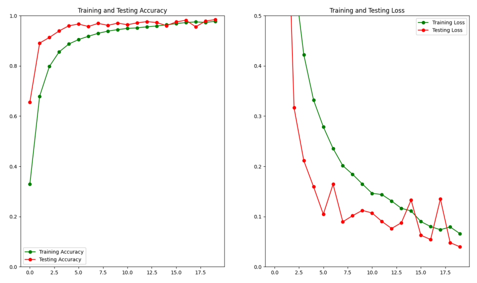


Fig. 24: Loss curve and accuracy of third CNN (Data Augmentation)

The confusion matrix on Fig. 25 shows almost perfect results but all misclassifications were due to the model classifying class 18 as class 6 this occurred 7 times out of 255 examples of class 18 or about 2.7% of the times.

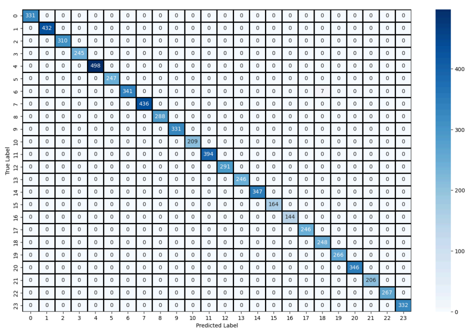


Fig. 25: Confusion Matrix for third CNN

V. Comparison

The results of all our trained models are on Table II and Fig. 26. Without data augmentation techniques we can see that our NNs performed worse compared to the CNNs, but with data augmentation, we were able to significantly improve the performance of our NN. In our CNN 2, we tried to explore regularization techniques like L2 regularization but with CNN 1 we can see that data augmentation can solve this problem, and model CNN 3 without data augmentation and L2 regularization despite overfitting achieved better accuracy than model CNN 2. CNN 1 achieved 100% accuracy but it is 6.6x larger than CNN 3 which achieved a similar result of 99.90% accuracy. Our Neural Networks didn't perform well without data augmentation and always seemed to overfit if we wanted to keep a performance of over 70%. NN 1 highlights the importance of normalizing data when working with Neural Networks since its accuracy is so poor compared to the similar model NN 2.

TABLE II: Comparison between models

Model	**Accuracy**	**F1-score**
NN 1	0.1861	0.156
NN 2	0.7796	0.7577
NN 3	0.7786	0.7559
NN 4 (data augmentation)	0.9948	0.9979
CNN 1	0.9090	0.9019
CNN 1 (data augmentation)	1.0000	1.0000
CNN 2	0.9164	0.9106
CNN 2 (data augmentation)	0.9765	0.9737
CNN3	0.9652	0.9660
CNN3 (data augmentation)	0.9990	0.9989

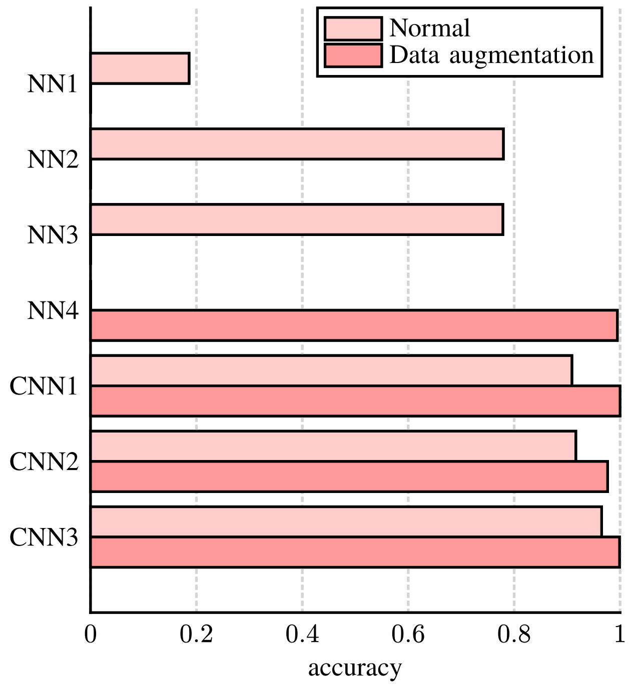


Fig. 26: Barchart of accuracy per model (Our models)

Comparing ourselves with our references as shown in Fig. 27 we can see that both our NN and CNN achieved accuracy close to the best model (Sayak Dasgupta CNN), but our CNN 3 model has 8x fewer parameters than the best model and our NN 4 is slightly smaller at 237584 parameters vs 319352 of Sayak Dasgupta CNN. Ranjeet Jain CNN performed much worse than our CNN 3 despite being a model of the same dimension.

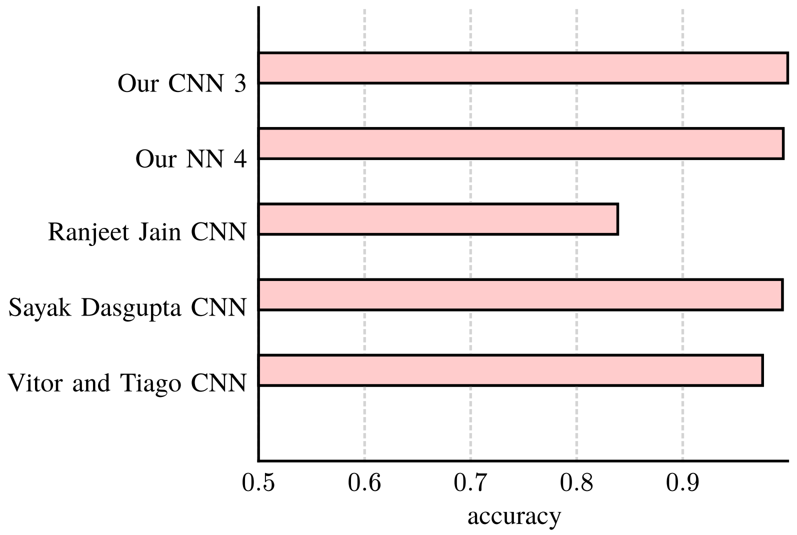


Fig. 27: Barchart of accuracy per model (Our best models vs references)

VI. Conclusion

For the NN, we reached several conclusions. First, normalization of the data is essential, as without it the models seem to achieve very bad results, with our first model getting an 18,61% accuracy. Adding normalization, multiple hidden layers, and a good value for alpha were important factors to create a good model, but the determinant factor was the data augmentation, making the last NN model achieve 99.48% accuracy. Talking about CNNs we are happy with our results where we made a small model with very high accuracy and once again an important factor in boosting performance is data augmentation. For future improvements to this model, we could consider using PCA or another method for dimensionality reduction, in an attempt to create a smaller Neural Network with the same capabilities or even an ensemble of models.

References

[1] R. Jain, “Deep Learning Using Sign Language.” [Online]. Available: https://www.kaggle.com/code/ranjeetjain3/deep-learning-using-sign-langugage

[2] S. Dasgupta, “Sign Language Classification - CNN (99.40% Accuracy).” [Online]. Available: https://www.kaggle.com/code/sayakdasgupta/sign-language-classification-cnn-99-40-accuracy

[3] V. Santos and T. Pereira, “Exploring Models for Sign Language Classification.” Aveiro, Portugal, 2023.