Ciências ULisboa

Faculdade de Ciências da Universidade de Lisboa

Aprendizagem Automática Avançada

2022/2023

# Real-time American Sign Language Alphabet Recognition using Convolutional Neural Networks

João Terroa - 53117

David Conceição - 52518

**Docentes**:

Luís Correia

Helena Tomás

Pedro Cotovio

# Contents



# List of Figures

**Abstract**

In this report, we present the development of a deep convolutional neural network (CNN), as well as an autoencoder, for the recognition of the American Sign Language (ASL) alphabet. The primary objective of this project was to gain hands-on experience working with deep learning frameworks and machine learning techniques. We have employed the TensorFlow and Keras libraries to train our CNN model, which were subsequently integrated with a webcam-based application using the OpenCV library for real-time predictions. This report details the design, implementation, and evaluation of the proposed system. The full code for this implementation can be found in the **Github** repository.

# 1 Introduction

The rapid advancements in artificial intelligence and machine learning have resulted in the development of numerous applications that aid human interaction and communication. One such domain that can benefit from these advancements is the recognition of sign languages, particularly the American Sign Language (ASL)[1;2]. ASL is a vital means of communication for the deaf and hard-of-hearing community, and the development of robust recognition systems can significantly enhance the quality of life and accessibility for these individuals.

## 1.1 Objective and scope of the project

The primary objective of this project is to provide an educational exploration of deep learning techniques and machine learning principles by developing a CNN model for ASL alphabet recognition. The project's scope encompasses the design and implementation of a convolutional neural network, as well as an autoencoder, the training and validation of said models, using TensorFlow[3] and Keras[4], and the integration of the trained model with a webcam-based application for real-time predictions using OpenCV[5]. The project also aims to evaluate the model's performance and discuss results regarding the project's findings and challenges.

# 2 Methodology

## 2.1 Dataset descriptions

Two datasets were used for this project. The first dataset is the ASL (American Sign Language) Alphabet Dataset from Kaggle, provided by the user debashishsau (figure 1). This dataset is a collection of images representing the alphabets from American Sign Language, contained within 29 folders. Each folder signifies a distinct class. It comprises $\approx$ 223,000 training images without a set resolution. The dataset is divided into 29 classes, including 26 classes for the letters A-Z and three classes for SPACE, DELETE, and NOTHING[6].

The second dataset used for this project is the Hand Gesture Recognition (HGR1) dataset (figure 2) available from the Silesian University of Technology. It comprises 899 images, collected from 12 individuals performing 25 distinct gestures. The dimensions of these images vary, ranging from 174x131 pixels up to 640x480 pixels. The background and lighting conditions of these images are uncontrolled, reflecting real-world conditions. Each image in this dataset is paired with ground truth binary skin presence masks and hand feature point locations, providing additional data for the training of the autoencoder[7].

## 2.2 Preprocessing

### 2.2.1 CNN Task

In preprocessing, the dataset undergoes a series of modifications. Firstly, the images are loaded from their respective folders, and each image is resized to a standard size of 100 $\times$ 100 pixels, to ensure uniformity in image size prior to input into the model.

The images are subsequently converted to grayscale which are favored as they simplify the model and reduce computation due to the presence of a single channel, as opposed to the three channels (Red, Green, Blue) in colored images. These grayscale images are then normalized by dividing each pixel by 255, which serves to convert the original pixel values from a range of 0 to 255 to a range of 0 to 1. This normalization process aids in accelerating the training process and can contribute to improved model performance.

Each image is then assigned a label corresponding to the class it belongs to. Following the labelling, the dataset is partitioned into training, validation, and test sets, with an 70%-10%-20% split, respectively, allowing the model to be evaluated on unseen data during the training phase, aiding in the early detection and prevention of overfitting

Lastly, the image labels are one-hot encoded, which allows the model to interpret the labels as categorical data, rather than numerical data.

Upon the completion of these preprocessing steps, the dataset is prepared for input into the CNN model for training.

### 2.2.2 Autoencoder Task

The preprocessing for the autoencoder model initiates by loading the original images and the skin masks from their respective directories. Each image and mask is resized to a uniform size of 128 $\times$ 128 pixels to maintain consistency across all input data. The pixel values of these images are, once again, normalized to a range of 0 to 1.

The dataset is subsequently split into a training set and a test set, with an 80%-20% split, respectively.

Following this, data augmentation is performed on the training set. Each image and corresponding mask is rotated by 90, 180, and 270 degrees, effectively multiplying the training set size by four. This augmentation process serves to introduce more variability into the training set, which can enhance the model's ability to generalize to unseen data.

Finally, the augmented training set is partitioned into a training set and a validation set, with an 87.5%-12.5% split.

At the end of these preprocessing steps, the dataset is ready for the training of the autoencoder model.

## 2.3 Model Architectures

### 2.3.1 Design of the CNN architecture

The CNN model, shown in figure 3, designed for the ASL alphabet recognition task, is composed of several layers. The input layer accepts grayscale images of size 100 $\times$ 100 pixels as input, with each pixel in the image being a feature, resulting in a total of 10,000 features. The first convolutional layer has 32 filters and uses a 3 $\times$ 3 kernel size with the ReLU (Rectified Linear Unit) activation function. It is followed by a max pooling layer which reduces the spatial dimensions (width and height) of the input volume for the next convolutional layer, using a pool size of 2 $\times$ 2.

This is followed by the second and third convolutional layers, both having 64 filters and using a 3 $\times$ 3 kernel size with the ReLU activation function. Another max pooling layer with a pool size of 2 $\times$ 2 follows these convolutional layers.

Next, we have the fourth convolutional layer with 128 filters, using a 3 $\times$ 3 kernel size with the ReLU activation function. This is followed by a third max pooling layer, which again uses a pool size of 2 $\times$ 2.

After the convolutional and pooling layers, a flattening layer is used to convert the 2D matrix to a 1D vector, which can be used as input for the dense layers. The first dense layer has 512 nodes and uses the ReLU activation function, and this is followed by a second dense layer with 84 nodes also using the ReLU activation function.

The final layer is the output layer, which is another dense layer with 29 nodes (corresponding to the 29 classes). It uses the softmax activation function to output a probability distribution over the 29 classes.

The model uses the Adam optimizer with a learning rate of 0.001 and uses categorical cross-entropy as the loss function. Early stopping is employed as a form of regularization to prevent overfitting, which halts the training process when the validation loss stops improving.

### 2.3.2 Design of the Autoencoder architecture

The Autoencoder model, shown in figure 4, is designed based on the U-Net architecture, which is a type of Convolutional Neural Network (CNN) originally designed for biomedical image segmentation[8]. This architecture is notable for its symmetric, expansive path which allows precise localization and hence is ideal for tasks such as image-to-image translation, which is what we aim for in our skin mask generation.

The U-Net Autoencoder accepts colored images of size 128 $\times$ 128 pixels as input. The architecture consists of a contracting path (encoder) and an expansive path (decoder).

The encoder part of the network consists of four blocks, each containing two convolutional layers with kernel size of 3 $\times$ 3 followed by a 2 $\times$ 2 max pooling operation. The number of filters in the convolutional

layers doubles with each block, starting with 64 filters in the first block and reaching 512 in the last block. After the last block, dropout is applied with a rate of 0.5 to prevent overfitting.

The decoder part of the network also consists of four blocks, each containing an upsampling of the feature map followed by a 2 $\times$ 2 convolution ("up-convolution") that halves the number of feature channels, a concatenation with the corresponding cropped feature map from the contracting path, and two 3 $\times$ 3 convolutions, each followed by a ReLU activation function. Dropout is applied to the third block with a rate of 0.5.

The final layer of the U-Net is a 1 $\times$ 1 convolution which maps each component feature vector to the desired number of classes, in this case 3, corresponding to the Red, Green, Blue channels of the image.

The model uses the Adam optimizer with a learning rate of $1e-4$ and the binary cross-entropy loss function. Once again, early stopping is employed as a form of regularization.

### 2.4 Integration with webcam for real-time predictions using OpenCV

To make use of real time hand gesture recognition, we make use of the OpenCV library in Python to integrate our model with a webcam feed. The webcam captures the video in real time, and each frame is processed individually to recognize the hand gesture.

First, we set the input dimensions for the images that will be fed to the model and create functions to preprocess and predict the class of each image.

Next, we load the model and initialize a video capture object using OpenCV. We also load the segmentation model, which we use to segment the hand from the rest of the image in each frame.

Inside the main loop, the program captures each frame from the video and uses MediaPipe's Hand solution to identify the hand landmarks. These landmarks are then used to create a Region of Interest (ROI) around the hand. This ROI is segmented using the segmentation model, and the hand gesture is recognized using the main model.

The program then displays the real-time video feed with the recognized hand gestures and their corresponding prediction probabilities. The loop continues until the user presses the "q" key.

## 3 Results and Evaluation

### 3.1 CNN Model

The overall accuracy of the model on the test dataset was 99.09%, a strong indication of the efficacy of the CNN for this classification task. The test loss was also significantly low, measuring 0.0395, further confirming the successful training process and the model's ability to generalize well to unseen data.

A detailed analysis of the model's performance across individual classes was conducted using precision, recall, F1-score, as well as examining the diagonal of the confusion matrix, which represents correctly classified instances for each class. The precision for all classes ranged between 0.98 and 1.00, and recall ranged between 0.97 and 1.00. These high scores demonstrate the model's capacity to correctly identify positive examples while minimizing both false positives and false negatives.

The confusion matrix, in figure 5, provides further insights into the model's performance for each hand gesture class. The correctly classified instance rates were consistently high, with a range from 0.967 for class 'S' to 1.00 for class 'U'. This suggests that the model was highly effective in distinguishing the different ASL hand gestures, even though some classes might have been slightly more challenging to classify than others, due to the nature of the language.

Overall, the weighted averages for precision, recall, and F1-score were all at 0.99, indicating a balance between positive prediction and actual positive occurrence for each class. This means the model has performed with a high degree of reliability and consistency across all classes.

### 3.2 Autoencoder Model

The Autoencoder model was applied as an unsupervised learning approach, which stands in contrast to the supervised Convolutional Neural Network model used earlier. Instead of learning to classify hand gestures, the autoencoder was trained to learn to segment a hand from the background, in images. It is used to extract the hand from the ROI.

The performance of the autoencoder model was evaluated primarily based on the reconstruction error (figure 6), a standard metric used to measure the difference between the original input and the recon-

structed output. The reconstruction error for the test dataset was quite low, at 0.0088, indicating that the model was generally successful in reconstructing the original data.

However, the metric 'accuracy' in the context of an autoencoder has a different interpretation compared to its usage for the CNN model. Here, it refers to the proportion of correctly reconstructed features for each instance in the test dataset, rather than the proportion of correctly classified instances. The achieved accuracy was 0.68, implying that the model correctly reconstructed approximately 68% of the features in the test instances.

# 4 Discussion

Prior to an in-depth exploration of the project's findings, it is important to initially clarify why a second model, the autoencoder, was incorporated into the real-time application pipeline, as counter-intuitive as it might seem, especially given the high performance of the standalone Convolutional Neural Network (CNN) model.

While the results obtained by the Convolutional Neural Network (CNN) are very good, it is important to consider the limitations and potential challenges in practical applications. The image dataset used for training and validation presents ASL hand gestures in an idealized condition, with hands posed against a plain and uniform background. This setup allows the model to learn and recognize the nuances of the ASL alphabet with high accuracy. However, it does not adequately represent the variety and complexity of real-world scenarios where the image of a hand may be set against diverse, cluttered, or distracting backgrounds. In realistic situations, the visual input for the model is likely to contain numerous elements beyond the intended hand gesture - such as other objects in the background, variations in lighting conditions, shadows, and even the presence of other body parts. We observed these factors interfering with the model's performance, as they were not part of the training conditions. This limitation resulted in reduced prediction accuracy, in our real-time application. To address these issues, we included an autoencoder for hand segmentation, designed to help the model isolate and focus on the hand gesture while minimizing the impact of background and other distracting elements. We observed a substantial improvement in terms of prediction accuracy following the integration of the autoencoder. Simultaneously, this also significantly increased the user's freedom of hand placement, resulting in a more natural and unrestricted user experience. This integration demonstrated that combining different AI models can sometimes lead to better results, as each model can focus on a particular task and compensate for the limitations of the other.

Regarding the project's findings, the CNN model performed remarkably well in classifying the ASL alphabets. This is attributable to the comprehensiveness of the dataset, including the variety and number of images for each class, and the consistency of the image backgrounds. In addition, the architectural design of the CNN model, which included multiple convolutional, pooling, and dense layers, facilitated the learning of complex patterns and relations among the features. The effectiveness of the chosen architecture is demonstrated by the high accuracy and low loss on the test dataset.

The precision, recall, and F1-score metrics also reflected the effectiveness of the model in classifying each class of gestures. The metrics showed consistently high values for all classes, indicating that the model was not biased towards any particular class, and performed equally well across all classes. The confusion matrix further confirmed this finding, as it displayed a high rate of correct classifications for each gesture, and few misclassifications.

The Autoencoder model, although it had a lower accuracy compared to the CNN model, performed adequately given its more challenging task of learning to generate the skin mask of a hand. The reconstruction error was low, indicating that the model was able to generate images that closely matched the original inputs. The success of the Autoencoder model can also be attributed to the architecture of the model, which was based on the U-Net architecture, a structure known for its efficacy in image segmentation tasks. It's worth noting that these results highlight the inherent differences between supervised learning tasks, such as classification, and unsupervised learning tasks like autoencoders. While precision, recall, and F1-score were appropriate for evaluating the CNN model's ability to correctly classify instances, our primary measure of performance for the autoencoder model was the reconstruction error. Furthermore, unlike the CNN model where higher accuracy indicated better performance, in the case of the autoencoder, achieving a perfect accuracy isn't necessarily the ultimate goal. The model should ideally be able to reconstruct the data accurately enough to preserve important features while still maintaining a reduced dimensional representation in the latent space.

In addition, the Autoencoder's performance, evaluated in terms of reconstruction error and accuracy,

was measured in a controlled environment similar to the one for the CNN model. Therefore, it is still subject to some of the limitations of the CNN model.

In conclusion, both the CNN and the Autoencoder models demonstrated good performance on their respective tasks. However, their performance is subject to the limitations of the datasets used, and might not necessarily translate to diverse, real-world conditions.

# 5 Conclusion

This project has demonstrated the effective application of deep learning techniques for the recognition of American Sign Language (ASL) hand gestures and the generation of binary skin presence masks. The developed Convolutional Neural Network (CNN) and Autoencoder models have shown very good performance in their respective tasks, with the CNN achieving an accuracy of 99.09% on the ASL Alphabet Dataset and the Autoencoder achieving a low reconstruction error on the Hand Gesture Recognition (HGR1) dataset.

The integration of these models into a real-time application using OpenCV and a webcam feed has further demonstrated their practical utility. However, this real-world implementation also highlighted some limitations in the performance of the CNN when presented with complex and varying backgrounds. This has led to the inclusion of the Autoencoder for hand segmentation, which has proven effective in mitigating these limitations and improving the overall accuracy of real-time predictions, despite it still suffering from some of the limitations itself.

Overall, the results of this project support the use of deep learning techniques for ASL recognition.

# References

[1] Sakshi Sharma and Sukhwinder Singh. Vision-based hand gesture recognition using deep learning for the interpretation of sign language. *Expert Systems with Applications*, 182:115657, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.115657. URL https://www.sciencedirect.com/science/article/pii/S0957417421010484.

[2] Yulius Obi, Kent Samuel Claudio, Vetri Marvel Budiman, Said Achmad, and Aditya Kurniawan. Sign language recognition system for communicating to people with disabilities. *Procedia Computer Science*, 216:13–20, 2023. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2022.12.106. URL https://www.sciencedirect.com/science/article/pii/S1877050922021846. 7th International Conference on Computer Science and Computational Intelligence 2022.

[3] Martín Abadi et al. Tensorflow. https://www.tensorflow.org, 2015.

[4] François Chollet et al. Keras. https://keras.io, 2015.

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Debashish Sau. Asl (american sign language) alphabet dataset. https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-aplhabet-dataset, 2021.

[7] Michal Kawulok, Tomasz Grzejszczak, Jakub Nalepa, and Mateusz Knyc. Hand Gesture Recognition (HGR1) Dataset. https://sun.aei.polsl.pl/~mkawulok/gestures/, 2017.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

# A Dataset Images

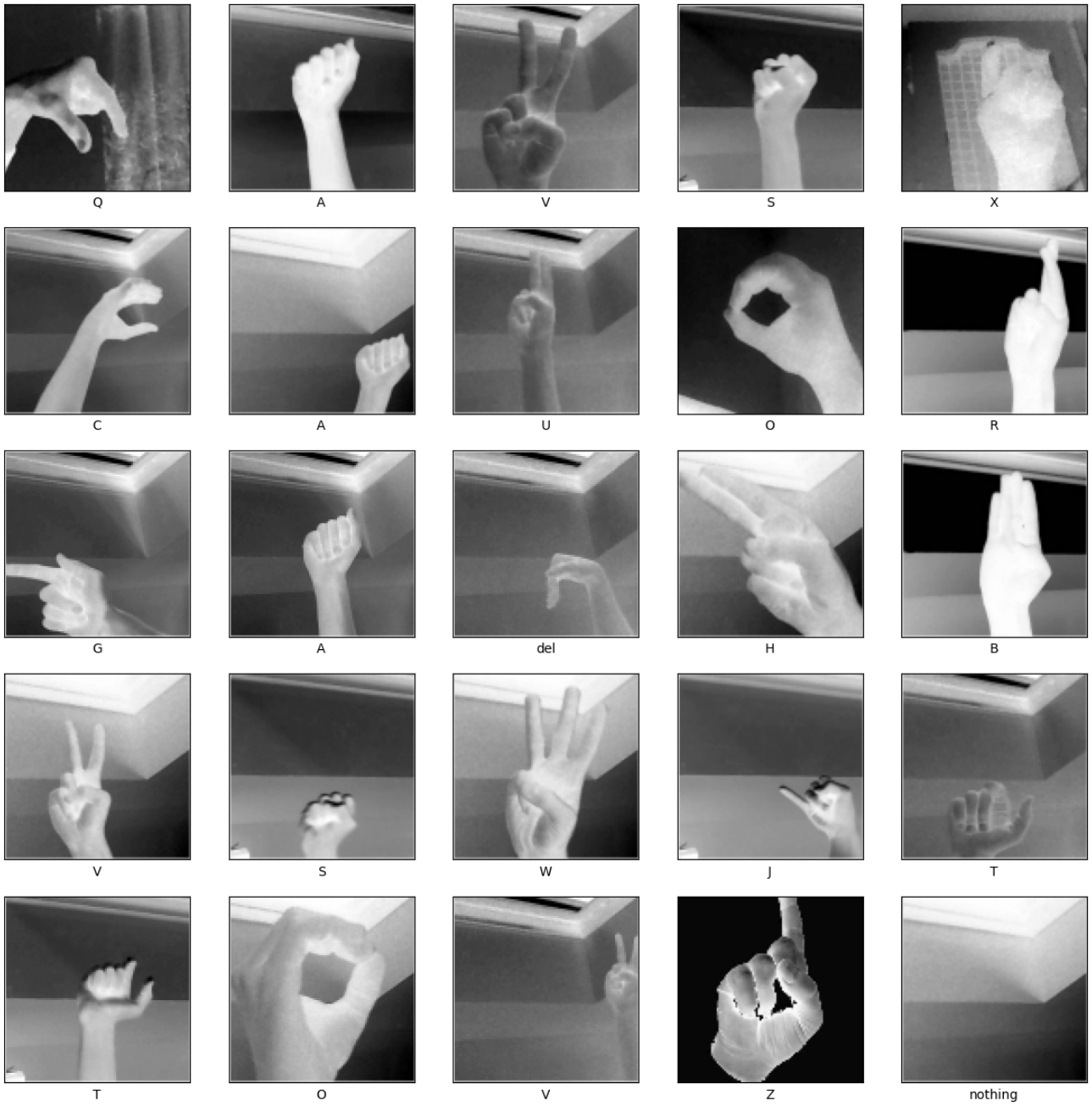


**Figure 1:** Sample for each class taken from the ASL dataset.

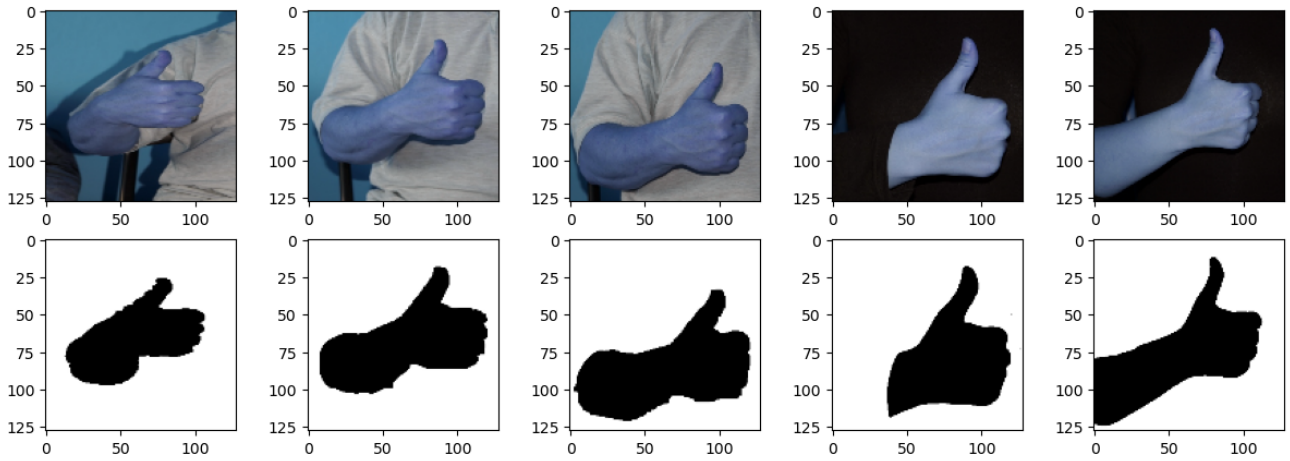

**Figure 2:** Samples taken from the HGR1 dataset.

# B Model Architectures

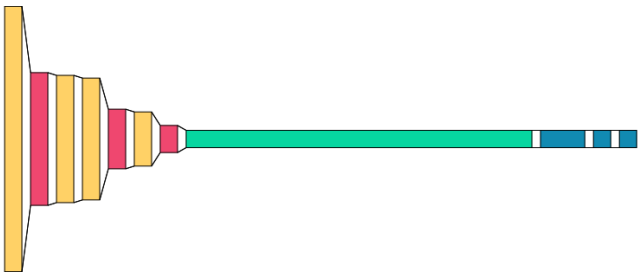

**Figure 3:** Architecture of the convolutional Neural Network used.

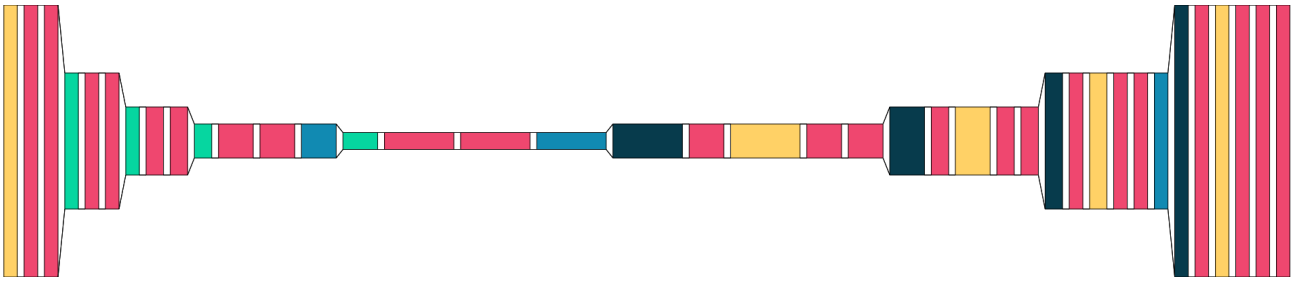

**Figure 4:** Architecture of the Autoencoder used.

# C Training Results

## C.1 CNN Model

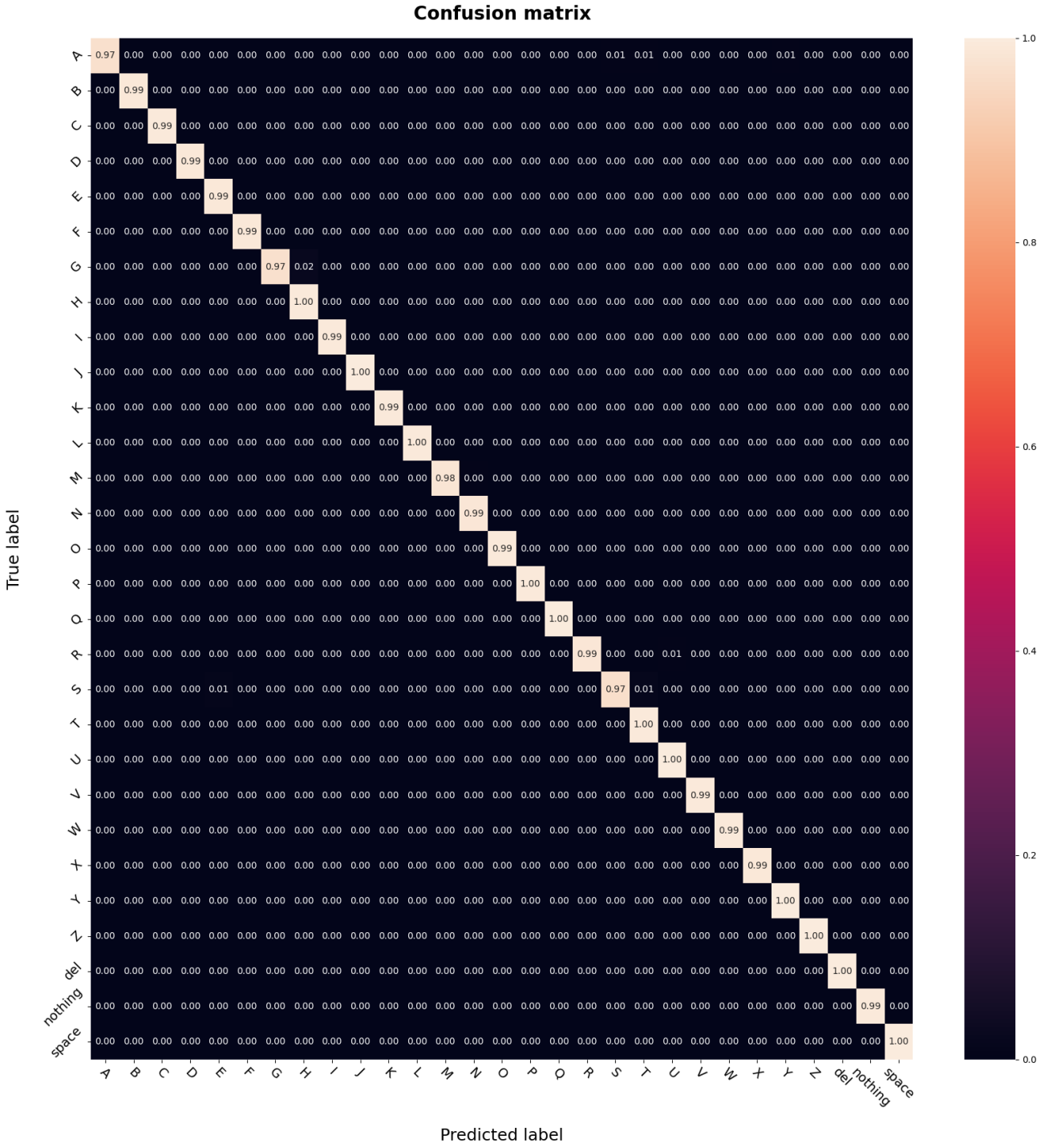


**Figure 5:** Confusin matrix for CNN model.

## C.2 Autoencoder Model


**Figure 6:** Examples of predicted masks using the autoencoder. The first row corresponds to the original images, the second row is the original masks and the third row corresponds to the predicted masks.