School of Information Technology and Engineering at the
ADA University

School of Engineering and Applied Science at the
George Washington University

DATA COLLECTION, AUGMENTATION, CLASSIFICATION, AND GENERATION FOR A SIGN LANGUAGE

A Thesis
Presented to the Graduate Program of Computer Science and Data Analytics
of the School of Information Technology and Engineering
ADA University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science and Data Analytics
ADA University

By Samir Dadash-zada

April, 2022

THESIS ACCEPTANCE

This Thesis by: Samir Dadash-zada
Entitled: *Data Collection, Augmentation, Classification, and Generation for a Sign Language*

has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

Dr. Jamaladdin Hasanov		28.04.2022
(Adviser)		(Date)
Dr. Abzatdin Adamov		28.04.2022
(Program Director)		(Date)
Dr. Sencer Yeralan		28.04.2022
(Dean)		(Date)

Abstract

This thesis discusses and brings a wide range of solutions for different aspects of the data collection, augmentation and generation for an automatic sign language to text translation system. Even though the data collection, augmentation and generation parts were implemented for the Azerbaijani Sign Language (AzSL) dataset, all of the mentioned stages can be used for a different set of sign language images and videos with minimal adjustment to the parameters of the corresponding models.

In order to reach out to as many subscribers as possible and collect an original set of images and videos from a public, an exclusive Telegram bot was developed and used during the course of 7 months. It was shared on various social networks in order to maximize the variance of features for each single sign. Multiple fixes and updates were applied on the bot in order to lead the bot subscribers to upload an image or a video that was more scarce.

Validation of the original dataset was done by running a few clustering algorithms. The results of the clustering revealed many similar signs, some anomalies, and feature patterns from the dataset and led to some important takeaways.

As the dataset will be mainly fed into Machine Learning models, the amount of original images and videos were not enough, especially after dropping out invalid media files among them. That was the main driving force to bring the data augmentation and generation parts. The image and video augmentation techniques used in the thesis are essentially selected because of their feature preservation and speed features.

In addition to the augmentation techniques, a state-of-the-art technique for generating synthetic images is also used called Generative Adversarial Network (GAN). A very specific version of GAN called StyleGAN is slightly customized to the dataset which is mainly used for generating photo-realistic high-quality images.

Contents

Figures

Tables

1 Introduction

The life of people with disabilities has continuously been eased with advancements in software development. During the last few decades, an increasing amount of new sophisticated models in Machine Learning has made it possible to reach a very isolated group of people and make their life easier in some way. Advancements in Computer Vision and Natural Language Processing directly affect the life of deaf and mute people in many countries. In order to bring easier alternatives for communication with those people, many countries have designed various Sign Language Recognition (SLR) systems. Some of the solutions require physical tools as gloves, some relate hand and lip synchronization techniques, and so on. Recent improvements in Computer Vision, Natural Language Processing and Machine Learning has brought new solutions which can work faster, scale easily and are more accurate. As these implementations are being used by many people, many improvements have been done and published. The one thing that gets in demand exponentially with each new solution is a data. Considering that SLR is a research area of Computer Vision, the data formats are usually image and video files. Generally speaking, creating a public dataset for a Computer Vision problem requires a lot to consider, as accessibility of the original dataset, its relevance to the domain, the size of the dataset, not having standards, security and privacy issues, solving storage related issues, and many more. The project of AzSL is one of the social projects that can change the life of 50,000 deaf-mute and hard of hearing people of Azerbaijan.

Generally, a SLR system is a multi-stage pipeline of models each with a unique set of requirements. At its core, SLR is the task of recognizing sign language elements from video streams. Since on simple hand gesture recognition CNN has achieved remarkable results, recognition of sign language is a more complex task and exceeds its limits. Any sign language consists of different one-step and multi-step gestures, and hand movements make it difficult to determine boundaries between glosses. The dynamism brings vagueness to the recognition and requires a methodology that could handle transitions of different signs. SLR systems can be classified into isolated and continuous based on whether the video streams contain an isolated gloss or a gloss sequence that corresponds to a sentence [11]. A successful SLR system can be considered one which can translate the information existing in image and video files simultaneously to a single channel. By this we mean that a sign language consists of sentences made by words, special words made by letters, and implicit transitions from one to another. Therefore the final goal of any SLR is an end-to-end mapping from source channels to an output channel.

1.1 Methodology

This research can be classified into many categories based on the classification criteria.

- Based on the target of the research this research can be classified as a scientific applied research as it will enable us to predict/detect signs from the input stream based on predefined measures. The prediction part in this case is more similar to classification of the sign.
- Based on the type of data used we can classify this exploration as primary quantitative research because the data is collected right from the source for this purpose and analyzed with the help of many mathematical formulas and algorithms.

In general, it should be emphasized that this research serves as a starting point for many other dependent researches. The dependency is based on the data availability. Among those models we can enumerate some advanced Machine Learning models as Deep Neural Network (DNN), Convolutional Neural Network (CNN), The Long Short-Term Memory (LSTM) which are used in sign recognition and sequence generation phases.
As AzSL has 24 static letters and 8 dynamic letters, the users should upload an image for each static letter and a video for each dynamic letter. Dynamic letters require a video because there should be a movement (aka a specific flow of the movement) by a user. The data was collected in a sequential and systematic way in order to have a balanced dataset with the same amount of images/videos per each category/letter. Considering that the data was uploaded to the server from various digital devices with different resolutions and technical specifications, we needed to apply a few preliminary cropping/encoding transformations before saving the data.
Analysis of the data has been conducted for multiple purposes, especially for clustering and anomaly detection. As static and dynamic letters have different file formats, they should be analyzed in a different way with the help of multiple clustering models. It should be highlighted that even between a few of the static letters there is a high visual similarity which makes the job of clustering algorithms challenging. That's why 3 different feature extraction model and clustering algorithms have been applied on the dataset of static letters in order to find out the one that can distinguish the most of them. Visual Geometry Group with 16 layers (VGG16) [14], InceptionV3 [16] and MediaPipe [18] were used to generate a feature vector for each image in the dataset, cluster them by k-means algorithm and reveal anomalies in the dataset. The selection of these models was based on the design of neural network, training focus, and the availability of a transfer learning to our dataset.

1.2 The purpose of the study

The AzSL project has been initiated by the National Observatory on Labour Market and Social Protection Affairs. Currently, it is supported by The Ministry of Labour and Social Protection of the Population of the Azerbaijan Republic and ADA University. The first and the foremost problem in the beginning of this project was data collection, augmentation, generation, and anomaly detection. Without a valid and standardized dataset it was impossible to carry out primary research of AzSL project.

The ongoing data-related research also brought many new challenges and got separated into an isolated primary quantitative research. The main target of this research is to prepare a ready-to-use dataset for AzSL with the help of various computational methods and remove visible impurities from it.

2 Related Work

Building a sign language recognition system requires a lot of effort as it is based on Computer Vision, Deep Neural Network, and Natural Language Processing techniques. Getting input from a live camera, detecting and tracking hand gestures, converting to words, and generating meaningful sentences are needed to be developed with care. In this section, we will talk about the challenges of developing such systems, work that was previously done, and different approaches. For any Machine Learning model, data (both quality and quantity) has a great deal of importance. Depending on the approach, the dataset to be used needed to be retrieved or collected according to criteria. To give an example, will your model also consider facial gestures of the actor or only his/her hand movements? Will he/she act in a specific location with specific background or it can be anywhere? Only one actor at a time or multiple? In general, training deep learning models requires a good amount of data recorded in different environments and with different actors. A few sign languages which have sufficient number of users have already available datasets such as "RWTH-PHOENIX-Weather (German)", "Boston ASL LVD (American)", "DEVISIGN-L (Chinese)" and others. Still for different purposes, datasets are being collected from scratch even if they are available open-source datasets. The countries which don't use widely spread sign languages and have their own may experience shortage of data which is the number one challenge in building such recognition systems.
Today, several sign language recognition systems are developed such as American, German, French, and Russian sign language, and some are still in development. Mainly, there are 2 approaches; hardware-based approach and pure ML techniques. In the former one, several hardware nodes are used to extract features of hands. One of them is the use of sensor gloves [1]. Sensor gloves resemble the kinds we usually wear, with the addition of cables connected to computers. It consists of multiple sensors that capture position of joints and their correlation while making moves. In other words, these gloves are extracting important features of hands which then will be used for training. Another example is the use of Microsoft Kinect to estimate human body poses [2]. Using a depth sensor it extracts important features to generate a 3D virtual version of a hand. Hardware based approaches are kind of old ones and especially for data collection, it became costly, and also Microsoft stopped support for Kinect devices.

With the development of ML tools and frameworks, hardware-based techniques started to disappear. CNN (Convolutional Neural Network) models are the widespread technique for sign language recognition systems. It takes images as input and classifies it to the corresponding letter or word.

One of the successful approaches in Sign Language Recognition is a component-based vocabulary-extensible Sign Language Gesture Recognition Framework. Generally speaking a gesture can been seen as a group of components. These components can include a shape of a hand, location of the hand, a rotation degree of the hand relative to the primary axis of the image, trajectory of

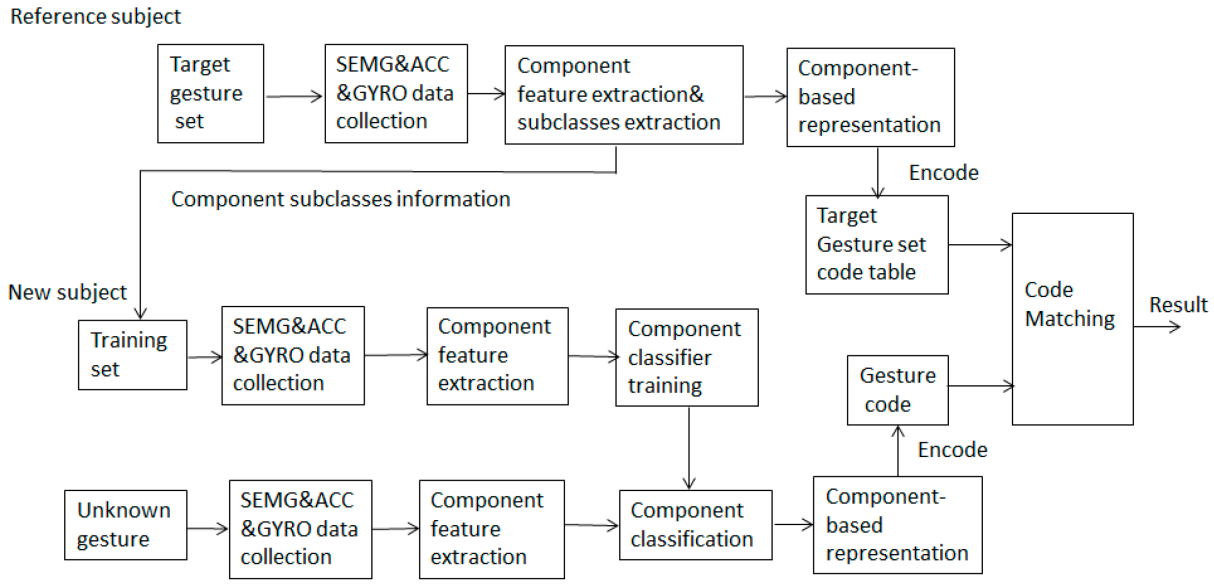

Figure 1 Component model

a movement, etc. Since most gestures share some specific and visible components, a component-based approach provides feasible solutions to the recognition of large-scale gesture set [17]. They are approaching to the solution by detecting those components in each dataset input and then divide signs into several subwords. In order to create a united feature set 238 subwords were extracted from 5113 signs as the basic units for large vocabulary CSL recognition [17]. The full training phase consists of withdrawing these common features from the whole dataset and grouping the components per each sign. After this step a code table of the target sign is created which is used for components - sign mapping in the recognition step. As a hand-shape classifier the Hidden Markow Model (HMM) was used in order to classify the resulting components as the HMM is a very powerful method for modelling sequential data. As we can see from the Figure 1 the training and the recognition flow has many similar modules. A similar idea to the component-based model will also be used in this thesis in order to extract all unique features from the image and cluster the images based on these features per each sign.

The Turkish Sign Language recognition system built in CNN + FPM (Feature pooling Model) + BLSTM (Bi-directional Long-Short Term Memory) + Attention model gave the best results [9]. Moreover, “MediaPipe” framework is a recent trend for feature extraction of both pose and hand landmarks. Then, those features are trained in CNN, LSTM, HMM and in other models for sign language recognition [4].

3 Data Collection

3.1 Initial stage of the data collection

The data collection part consists of multiple improvement stages. For the first step of the image collection, we had to come up with a tool that was easily accessible to the public and required a very small initial cost. Considering an increasing use of Telegram bots in Azerbaijan, we decided to develop an easy-to-use bot called JestDiliBot with a single command which was only responsible for data collection. A user is asked to select a letter, review a sample image or video based on the selection, take a picture or record a video of the letter according to the shown gestures, and upload the file to the bot.
Even though it sounds like a proper set of instructions for subscribers, after a few days the dataset became biased as many people were selecting the initial letters of the Azerbaijani alphabet and uploading those samples only. The result of this uncontrolled data collection is shown in 2.

This led to an improved controlled version of the data collection process where the following steps were executed after each successful file upload:

- a file is saved in the server
- an internal file responsible for caching the statistics is updated with a new count of images/videos
- when a **start** command is selected the next time, only 10 letters with the minimum number of data is shown to the user

A better distribution of the dataset was achieved after a few months as shown in 3

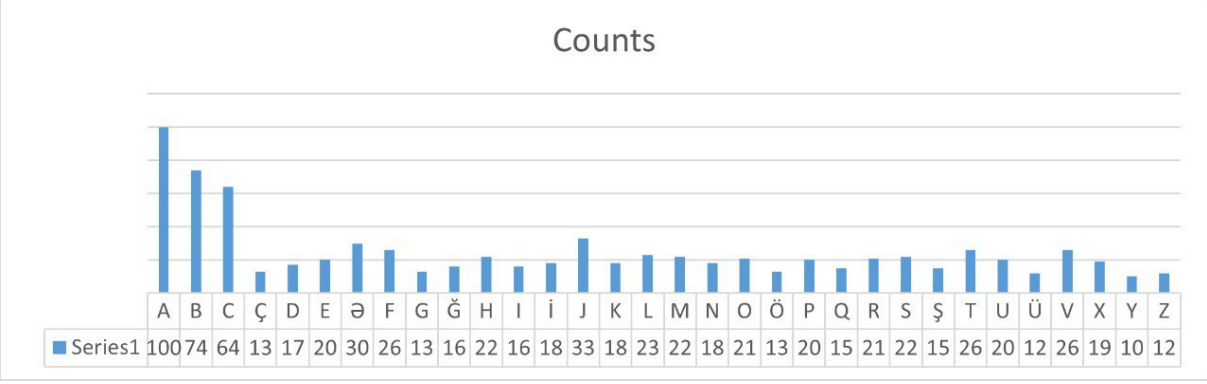

Figure 2 Unequal distribution of the collected dataset. 11-August-2021

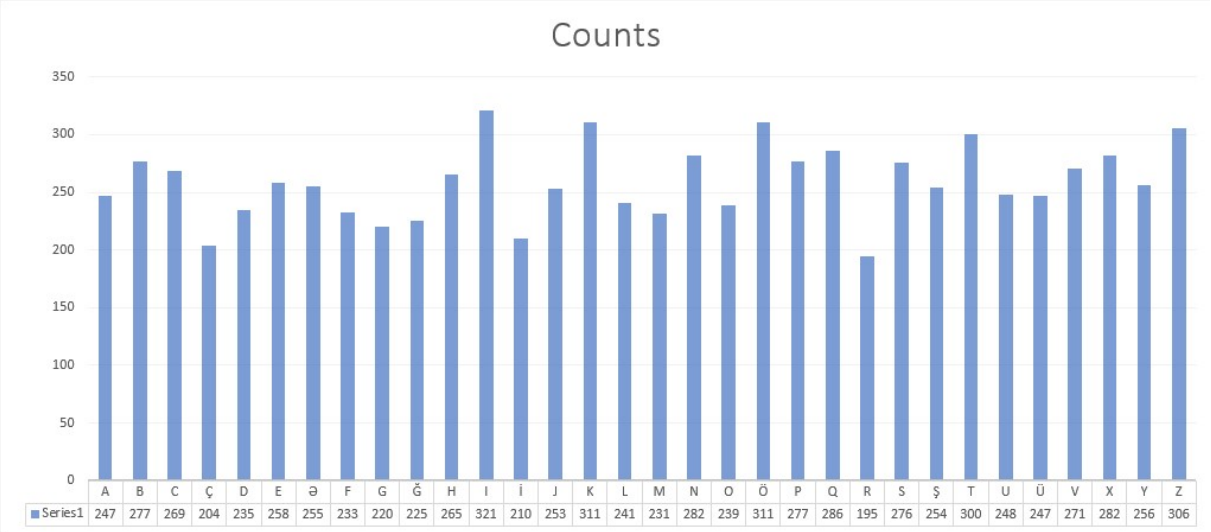

Figure 3 Better distribution of the collected dataset. 01-December-2021

3.2 Collected raw dataset

As the AzSL consists of static and dynamic letters, the raw dataset was stored and analyzed as two separate dataset. It is also a logical separation of the dataset as video analysis, clustering, and augmentation can require a separate set of tools. In summary, we could build a dataset of 14,144 images and 3,558 videos collected from 221 volunteers. The dataset has been described in detail and made publicly available for the SLR community. There is a part of the dataset that can be categorised as invalid and some of the issues are mentioned below.
The below images show the raw dataset in terms of the static letters:

Figure 4 Letter A. 835 images in total. A front view [left] and a side view [right].

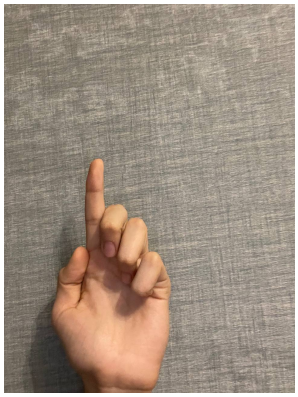
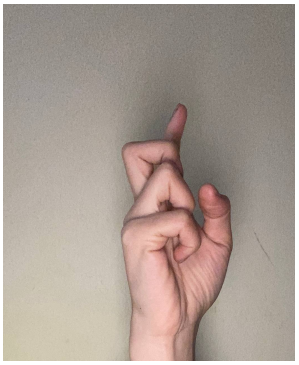

Figure 5 Letter B. 729 images in total. A front view [left] and a side view [right].

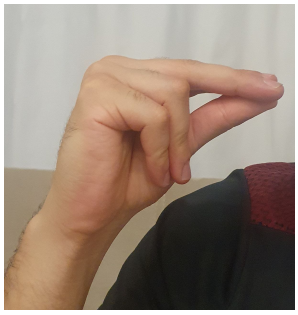


Figure 6 Letter C. 621 images in total. A side view [left] and a front view [right].

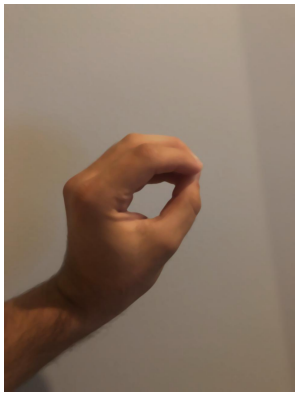
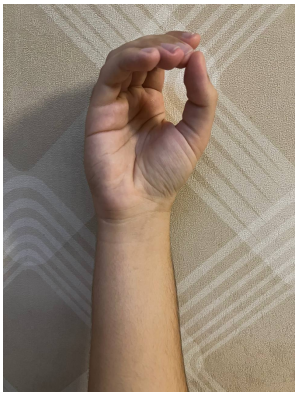

Figure 7 Letter E. 462 images in total. A side view [left] and a front view [right].

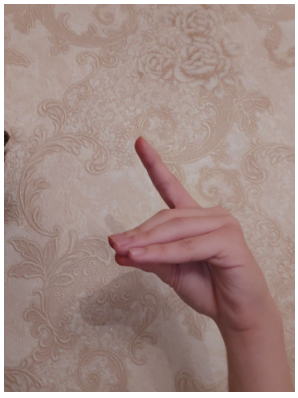


Figure 8 Letter EE. 201 images in total. A side view [left] and a front view [right].


Figure 9 Letter F. 443 images in total. A front view [left] and a side view [right].

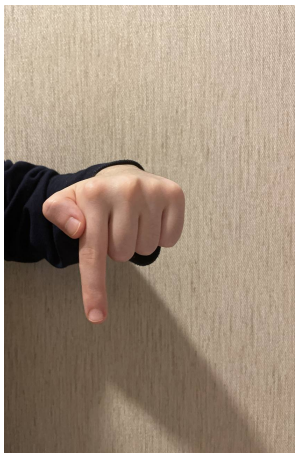


Figure 10 Letter GH. 142 images in total. A front view [left] and a side view [right].

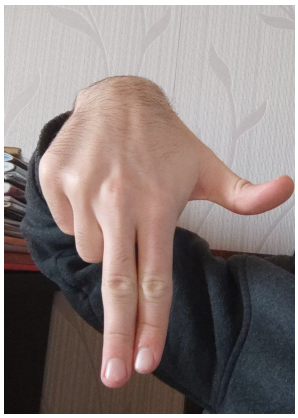
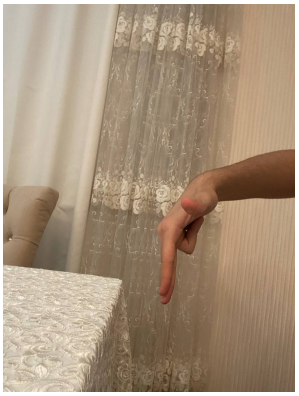

Figure 11 Letter H. 548 images in total. A front view [left] and a side view [right].

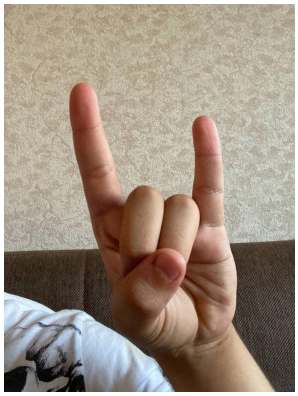
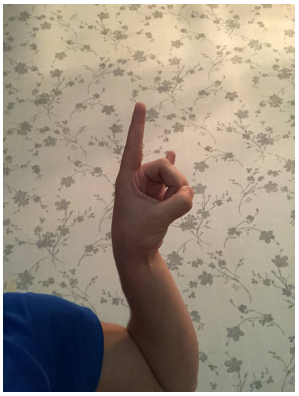

Figure 12 Letter I. 916 images in total. A front view [left] and a side view [right].

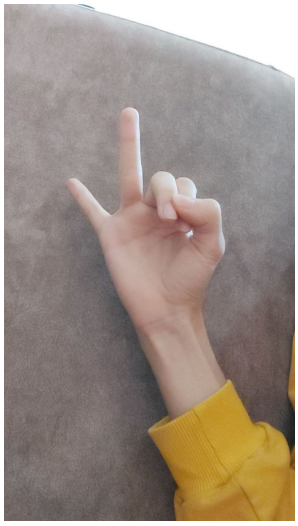
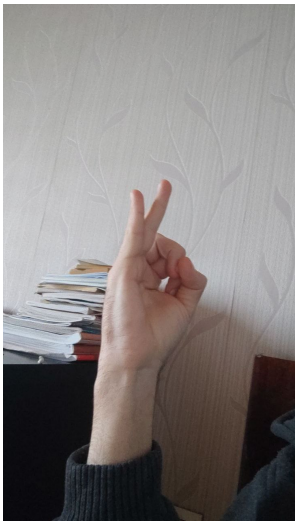

Figure 13 Letter II. 86 images in total. A front view [left] and a side view [right].


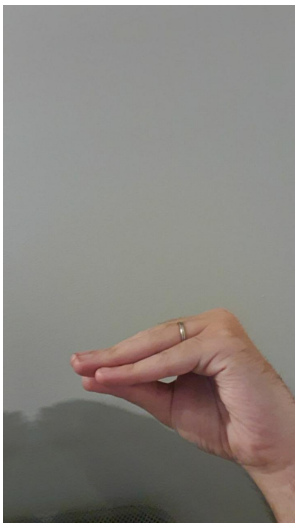

Figure 14 Letter J. 511 images in total. A front view [left] and a side view [right].

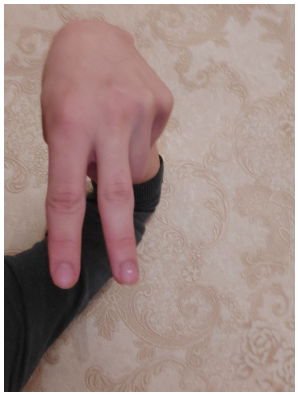
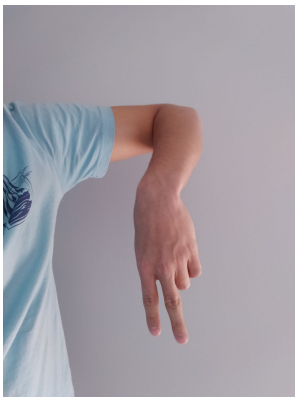

Figure 15 Letter L. 451 images in total. Front views.


Figure 16 Letter M. 594 images in total. Front views.

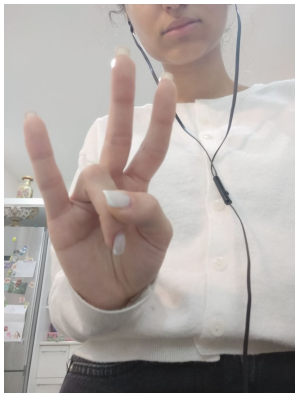


Figure 17 Letter N. 434 images in total. A front view [left] and a back view [right].

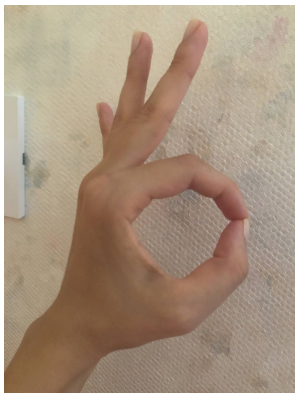


Figure 18 Letter O. 647 images in total. A front view [left] and a back view [right].


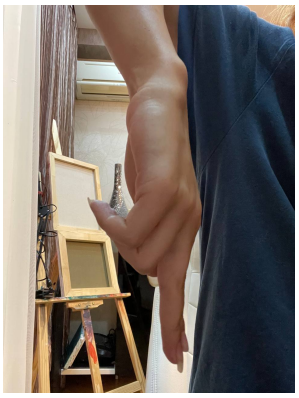

Figure 19 Letter P. 468 images in total. A front view [left] and a side view [right].


Figure 20 Letter Q. 420 images in total. A front view [left] and a side view [right].


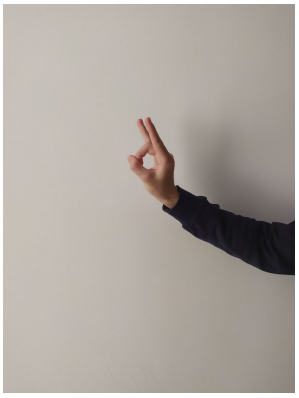

Figure 21 Letter R. 435 images in total. A front view [left] and a side view [right].

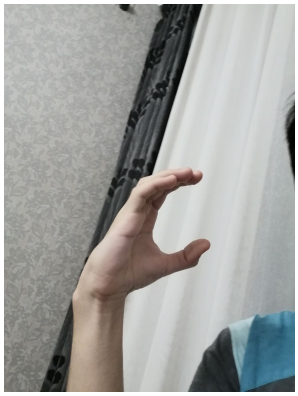
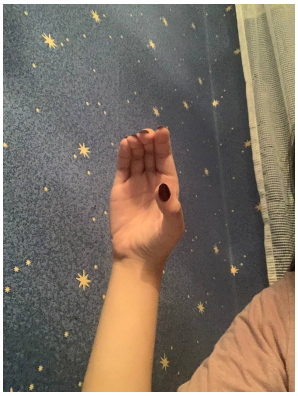

Figure 22 Letter S. 485 images in total. A side view [left] and a front view [right].


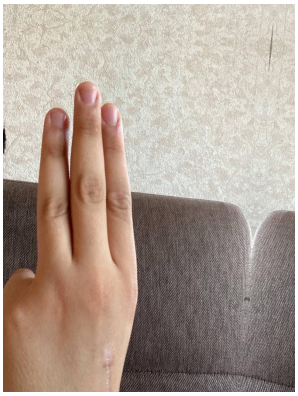

Figure 23 Letter SH. 157 images in total. Front views.

3.3 Analysis of the raw dataset

In depth visual analysis of the raw dataset revealed many invalid, duplicate images and videos that made data cleansing necessary for the dataset. In order to avoid a manual intervention fully, a fully automatized anomaly detection has been developed by testing multiple Machine Learning models as will be described in the upcoming section.
As we can see from the structured classification of the images, there are many important points to consider during the data cleansing process:

- There are some letters in AzSL with very high similarity ratio so that a human eye can hardly distinguish them. For example, a side view of the **letter I** (12) and a side view of the **letter R** (21) or a front view of the **letter L** (15) and a front view of the **letter P** (19). Having many indistinguishable features between two actually different dataset makes it very complicated for classification models to set them apart. We will see many samples of this challenge in the image classification part.

- In some letters dataset consists of front/back and left hand-right hand views. It gives us the advantage of having 360 degree recognition of the letter and letting any model learn it this way in the future. However, it brings many features for classification models and can cause wrong identification of the letter by the model.

- The depth of image features are very different from one another. As an example we can show **letter R** samples (21) or **letter H** samples (11). In the section regarding classification models, I will show which models can focus on the hand only disregarding the depth of the hand in the image and which cannot.

- Another issue with the hands in the image is its location. It can be seen from many examples that the zoom level and location of the hand in the image varies extremely between two samples. It makes it complicated to apply many pre-processing techniques or augmentation techniques as cropping, changing focus, etc. as we can end up with an image without a hand being fully shown in it.

- Video files, on the other hand, have a different set of challenges. There are some slices of the videos where the hand goes out of the frame and it can be considered an invalid data for models which try to find a sequence between consecutive frames of the video. This issue also makes some of the augmentation techniques not applicable for videos as we will discuss in the section 5.
  The second issue of working with videos is the extra need for computing power. For example, a simple average video of 3 seconds can have 60-70 frames on average which can require 70 times more resources than an average image to work with.

4 Data Classification and Anomaly Detection

Having a labeled dataset collected from an uncontrolled source such as a Telegram bot brings many issues with itself. Having the same image with different labels and having a lot of mislabelled images are the most challenging ones. In order to avoid any manual touch, I have applied a few classification techniques in order to find a correct label for each image and also detect hidden anomalies. However, as I have also mentioned in the previous section, considering a high rate of similarity between images makes the application of classification algorithms even more complicated.
When it comes to image classification, one can find many algorithms and Machine Learning models to apply and see the result. However, considering the time and resource availability and the domain of the applied models, I have decided to opt out for the following three models:

- **VGG16** [14]: It is a widely used CNN Architecture for ImageNet (a large visual database project used in visual object recognition consisting of over 14 million images belonging to 1000 classes). The model achieved 92.7% top-5 test accuracy in ImageNet. It is preferred for many image recognition and classification tasks because of its simplicity to apply on the dataset and implementation. In the case of AzSL, VGG16 will be used as a pre-trained network for feature extraction. The model gets its name from its architecture having 16 layers of network as shown below:

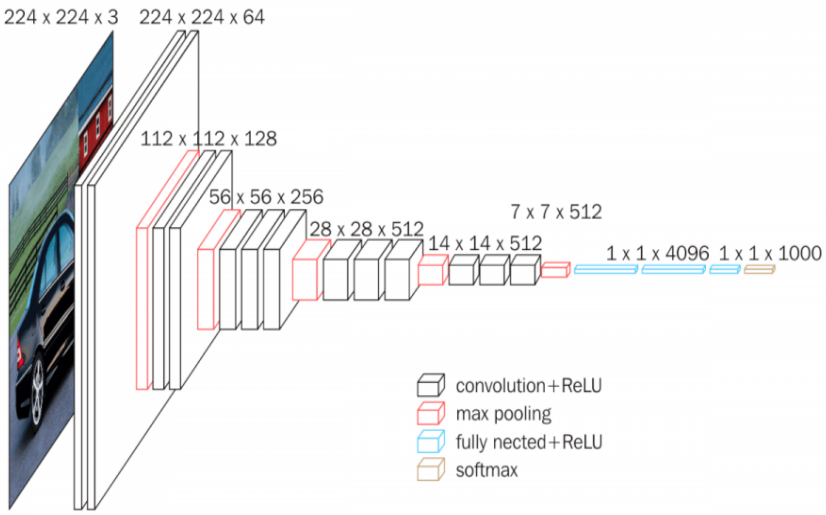

Figure 24 VGG16 Architecture

- **Inception V2** [16]: It is also a deep CNN architecture that was introduced in 2014 by Google researchers. It won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC14) and became one of the famous convolution models. As it is also a pre-trained model, using this model as transfer learning and applying on your custom dataset is not very difficult. The main difference of Inception V2 from its parent version V1 was a division of one large 5x5 convolution into two small sequential 3x3

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0

flatten (Flatten)            (None, 25088)             0

fc1 (Dense)                  (None, 4096)              102764544

fc2 (Dense)                  (None, 4096)              16781312

predictions (Dense)          (None, 1000)              4097000

=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

Figure 25 VGG16 Model summary

Algorithm 1 VGG 16 Feature extraction and clustering algorithm

Require: from keras.preprocessing import image
Require: from keras.applications.vgg16 import VGG16
Require: from keras.applications.vgg16 import preprocess_input

i, $imagepath$ in enumerate($filelist$)
(1) img = image.load_img($imagepath$, target_size=(224, 224))
(2) img_data = image.img_to_array(img)
(3) img_data = np.expand_dims(img_data, axis=0)
(4) img_data = preprocess_input(img_data)
(5) $features$ = np.array(model.predict(img_data))
(6) $featurelist$.append($features$.flatten())
$kmeans$ = KMeans(n_clusters=$number\_clusters$, random_state=0).fit(np.array($featurelist$))

convolution operations to improve computational speed. With the help of this trick, a faster execution has been achieved roughly 3 times more.

Algorithm 2 Inception V2 Feature extraction algorithm

Require: from keras.preprocessing import image
Require: from keras.applications.inception_v3 import InceptionV3
Require: from keras.applications.inception_v3 import preprocess_input

$model$ = InceptionV3(weights='imagenet')
▷ Remove the last layer (output softmax layer) from the inception v3
$inceptionModel$ = Model($model.input$, $model.layers[-2].output$)
i, $imagepath$ in enumerate($filelist$)
(1) $keypoints$ = encode($imagepath$)
(2) $featurelist$.append($keypoints$)
$kmeans$ = KMeans(n_clusters=$number\_clusters$, random_state=0).fit(np.array($featurelist$))

- **MediaPipe** [18]: It is a huge library of different Machine Learning solutions for live and streaming data made by Google. One of the solutions is the **MediaPipe Hands** which is a high-fidelity hand and finger tracking solution. Based on the results of the classifications done on the AzSL dataset, this model has shown the best classification performance by detecting the maximum number of unique letters. MediaPipe Hands utilizes a ML pipeline consisting of multiple models working together: a palm detection model that operates on the full image and returns an oriented hand bounding box, and a hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints. [8] As we can see from the Figure 27 the model can detect 21 precise keypoints including the inside of a palm. This highly accurate keypoint detector can recognise the hand and the mentioned localizations for hands in an almost orientation-invariant manner as visualized in the Figure 28.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 299, 299, 3       0         []
                             )]

conv2d (Conv2D)              (None, 149, 149, 32       864       ['input_1[0][0]']
                             )

batch_normalization (BatchNorm  (None, 149, 149, 32    96        ['conv2d[0][0]']
alization)                   )

activation (Activation)      (None, 149, 149, 32       0         ['batch_normalization[0][0]']
                             )

conv2d_1 (Conv2D)            (None, 147, 147, 32       9216      ['activation[0][0]']
                             )

batch_normalization_1 (BatchNo  (None, 147, 147, 32    96        ['conv2d_1[0][0]']
rmalization)                 )

activation_1 (Activation)    (None, 147, 147, 32       0         ['batch_normalization_1[0][0]']
                             )

...
Total params: 21,802,784
Trainable params: 21,768,352
Non-trainable params: 34,432
```

Figure 26 Inception V3 Architecture

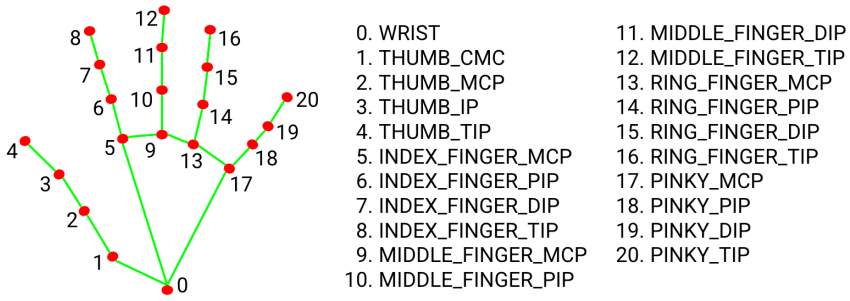

Figure 27 MediaPipe hand detection

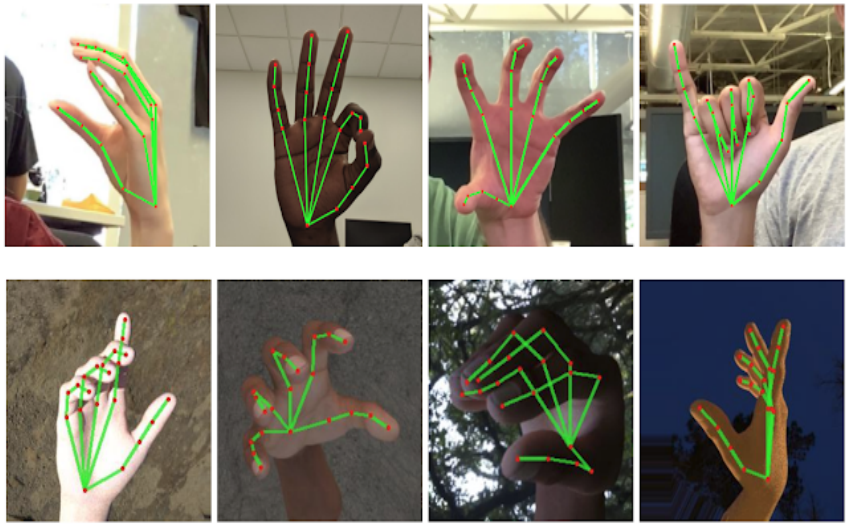

Figure 28 Top: Aligned hand crops passed to the tracking network with ground truth annotation. Bottom: Rendered synthetic hand images with ground truth annotation.

Algorithm 3 MediaPipe Feature extraction algorithm

Require: import mediapipe as mp

$drawingModule$ = mp.solutions.drawing_utils
$handsModule$ = mp.solutions.hands
mp_model = handsModule.Hands(static_image_mode = True)

$filePath$, $loadPath$ in enumerate($filelist$)
    img, $results$ = mediapipe_detection($loadPath$, mp_model)
    $hand$ = hand_detection($results$)
    $keypoints$ = extract_keypoints($hand$, $results$)
    $featurelist$.append($keypoints$)
    $filelist$.append($filePath$)
$kmeans$ = KMeans(n_clusters=$number\_clusters$, random_state=0).fit(np.array($featurelist$))

5 Data Augmentation

5.1 Purpose of the data augmentation

Despite the fact that ML models are considered the state-of-the-art approach in many data driven applications, they have some significant drawbacks. The first and foremost is the amount of data which is the bottleneck for many solutions. The other drawback is the labelling of the dataset. Most of the time, the labelling defines the model that is going to be applied for the dataset. That is why we seek to increase the size of the dataset by different means. There are a variety of solutions based on the domain of the problem. Augmentation techniques come in very handy as an initial solution to increase the number of input dataset by preserving the accuracy of the labels and the uniqueness of the dataset.
However, it is not an easy job to come up with a pipeline of augmentation techniques that generate a valid image/video and can be used in the dataset. The following issues should be considered during the selection of the augmentation techniques:

- A technique needs to preserve the main landmarks/features/points in the image
- A technique should support being randomized with the help of parameter tuning so the same technique can be used multiple times in a pipeline
- A technique should return the same type of the input data in order to be used in a pipeline of techniques

5.2 Image augmentation

The following augmentation techniques have been selected and applied on the images with a 0.5 probability:

- Flip horizontally. Horizontal flipping is essential for this dataset as we would never want any model to understand that a sign is related to a right hand or a left hand only. It would just add a suspected bias to the database if most of the subscribers were right-handed or left-handed. By introducing horizontal flips to the augmentation, we are fully eliminating a possible effect of that. However, no vertical flips have been added to the augmentation pipeline. The rationale behind that is also related to the models which will be trained on this dataset. It is not realistic to assume that any model needs to be trained on signs which are vertically flipped. In actuality, it doesn't mean anything to view a sign as such.
- Crop width/height between 0-10% of the original image. This augmentation trick is also among the crucial ones in order to divert the hand from the center as we need to

train models to be able to detect the hands in any part of the image and extract all the features from the hand.

- Affine transformations as scaling between (-10%) – 10% over x and y axis, translation over x axis between (-10%) – 10% and over y axis between (-15%) – 15%, rotation of 0-7 degree clockwise and counter-clockwise.

In addition to the above, five randomly selected augmentations from all will be applied on the image:

- Superpixel representation of some pixels (between 20-200 pixels)
- One of the blurring techniques: Gaussian blur with sigma between 0 and 0.3, uniform blur with kernel size 2x2 and 7x7, median blur with kernel size 3x3 and 11x11.
- Coloring of some edges with black and white with edge detection techniques
- Additional Gaussian Noise as increase in brightness
- Addition and multiplication of some pixels with a constant to increase the color range or brightness
- Apply grayscale
- Distort local areas with varying strength

5.3 Video augmentation

Even though the video augmentation can be thought to be the same as an image augmentation, there are many crucial points that need to be considered separately. The following issues should be addressed during the selection process of the augmentation filters for a video:

- **Processing time.** Any image augmentation technique can be used on video augmentation as long as its processing time is very short. As a video consists of multiple frames (average frame per second (fps) in the dataset is 23), augmentation of a single video multiplies the duration of a single image by 20 on average. Therefore, advanced and time consuming augmentation techniques should be avoided as much as possible.
- **Random selection of filters in a sequence.** In contrast to the image augmentation, we cannot use a random selection of filter for each frame. The same set of filters have to be used for each video file in order to end up with a consistent and valid video file. Otherwise, each frame will be very different from the previous one and the sequence will not make sense for any viewer (also for any model).

The following augmentation techniques have been applied on the videos:

- Flip horizontally (no vertical flip has been used)
- Crop width/height between 0-10% of the original image. Bigger cropping ratios are not applicable as the position of the hand varies from frame to frame and can be cropped by mistake
- Gaussian and average blurring
- Linear contrast, edge detection and sharpening
- Addition and multiplication of pixels with a constant
- Various affine transformations as translating, scaling, rotating, shearing

It should be mentioned that for each new video input file the implementation picks a new set of filters with random parameter values generated from the predefined range for each filter and then all of the filters are applied sequentially to all the frames of the video file.

6 Data Generation and GAN

6.1 Purpose of the data generation

As mentioned in the section regarding Data Augmentation (5) the more and quality data we have for the Machine Learning models, better the results are going to be. However, the available techniques for increasing the size of the dataset is limited. Additionally, most of them are creating some kind of variations of the original images which can result in different levels of biases in the dataset. A few different ML technologies have been introduced in order to mitigate the visual data scarcity issue and add a large amount of synthetic unique data. The most recent and advanced models are mainly based on two broad ML models: GAN and Autoencoders. Each of them requires a bunch of papers to read in order to understand the idea behind and to be able to customize the model for your dataset. As the following sections will provide a better understanding for GAN, I will give a short explanation for Auto-encoders and why it was not suitable for the AzSL dataset.

In short , an Auto-encoder is a type of algorithm with the primary purpose of learning an "informative" representation of the data that can be used for different applications by learning to reconstruct a set of input observations well enough. [10] In general, auto-encoding is a lossy compression algorithm where encoder (compression) and decoder (decompression) functions learn the most optimized set of parameters from the data itself with the help of examples. In other way, an Auto-encoder established a distribution of unique features from the samples encountered and then tries to generate the same samples based on the distribution. During its training phase, the model tries to find the best distribution possible in order to make the loss (the difference between the input samples and the generated output) minimum. In most typical architectures, the encoder and the decoder are neural networks since they can be easily trained with existing software libraries such as TensorFlow or PyTorch with backpropagation. [10] The trained model can generate synthetic images similar to the ones in the provided input dataset. The most interesting part is that we can make each artificially generated image as unique as possible with the help of a random vector which the decoder requires as a parameter. This random vector adds a unique flavour to each generated output not depending on the previous generated ones.

Even though Auto-encoders have been applied on various domains for datasets with different number of classes, it is not a silver bullet for all kinds of the data augmentation problems. The AzSL dataset has a unique characteristics of having 24 classes (for static letters) with a higher similarity between classes than most of the well-known publicly available datasets which Autoencoders model are usually trained on. But the SytleGAN model which is going to be explained in the sections below have a focus on generating better output images based on the datasets with many classes with similar features.

6.2 Generative and Discriminative Algorithms

It is necessary to understand the logic behind the generative and the discriminative algorithms before diving deep into the implementation details of GAN network. Actually it is a lot easier to understand this pair of algorithms by comparing and contrasting them with each other.

The generative model (G) can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model (D) is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from the genuine articles. [3] The discriminative algorithms are subject to classify input data where the features of the instance of data are given and by this way, they do prediction of a category or a label that data belongs to (31). For instance, considering the data instance in case of the words given in an email, a discriminative algorithm might predict if the received messages are spam or not spam by labeling them accordingly. Spam or not spam are those labels so that the input data is constituted by the features coming from the email which contains those words to be labelled. From a mathematical point of view, we can call the label "a" and the features can be defined as "b".

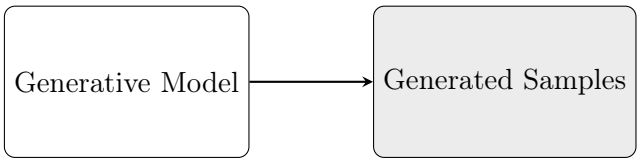

Figure 29 Model with generative capabilities, which running on the dataset to generate fake samples

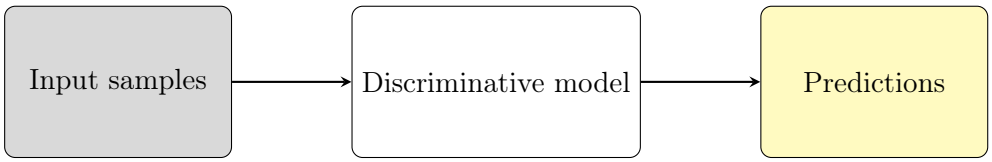

Figure 30 Discriminative algorithms receive the samples to identify their authenticity

The formulation will be P(a|b) which is used to mean that "the probability of a given b". In the case of our example, "the probability that the email is considered spam given the words it contains, meaning that discriminative algorithms try to map features to appropriate labels.
However, generative algorithms try to do the opposite of what discriminative ones do. They perform a prediction of features based on a certain label instead of doing prediction-given features. A generative algorithm attempts to answer, supposing the email is spam, how likely those features are. While discriminative models care about the relation between "a" and "b", generative models care about how you actually get "b". Clearly, here the probability P(b|a) is captured - the probability of "b" given "a".

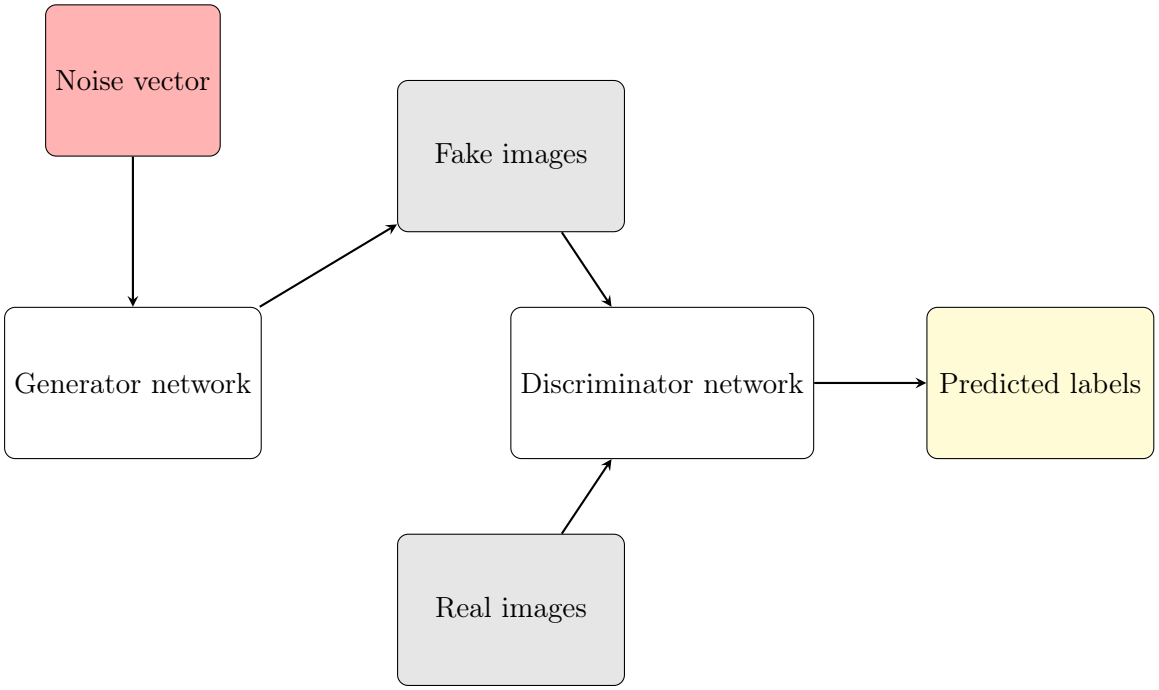

Figure 31 Discriminative algorithms receive the samples to identify their authenticity

There are many issues that should be carefully monitored during training of both models. The primary concern is the synchronization of the D with G as we don't want the discriminator to become optimized in a few epochs and prevent the generator to improve its performance. Another issue is related to a failure case where the generator starts giving out samples from a single or a few classes only (low variety) and stops producing any samples from other classes. This is called *a mode collapse* failure case and can be addressed using one of following solutions:

- Experience replay where we repeatedly add previous fake samples to the training dataset of the discriminator
- Having multiple GANs for different modes and combine the results.

6.3 GAN - Loss function for Discriminator and Generator

In some sense, GAN can be seen as a good implementation of game theory, where the generator and the discriminator are competing with each other and making one another more stronger in each iteration. However, there are many failure cases for both of the models when it comes to the actual training because of this competitive environment. That's why we need to monitor the training losses carefully during the training phase. The following subsections will discuss the mathematical concepts of the loss functions for the discriminator and the generator.
A few notations will be established before analysis of mathematical expressions in order to make the application of formulas clear. The following parameters will be used in the formulas of loss functions:

- G = Generator
- D = Discriminator
- z = Input noise from a latent space
- $P_z(x)$ = Probability distribution of an input noise
- $P_d(x)$ = Probability distribution of an original data
- $P_g(x)$ = Probability distribution of a generated data

The original loss function introduced in [3] can be derived from the *binary cross-entropy loss* formula as shown below:

$$L(\hat{y}, y) = [ylog(\hat{y}) + (1 - y)log(1 - \hat{y})]$$

6.4 Discriminator Loss

In the case of the training phase of a discriminator with the original data, the above function converges to the first part of the formula because of the $(y = 1)$. And the resulting loss function for D becomes as following considering $\hat{y} = D(x)$ (original data):

$$L(D(x), 1) = log(D(x))$$

At the same time, the discriminator needs to be trained on the synthetic data coming from the generator. The label for the fake data is $y = 0$ and the data is $x = D(G(z))$, so the loss function becomes:

$$L(D(G(z), 0) = log(1 - D(G(z)))$$

We should emphasize one more time that the discriminator will reach an optimal performance when it can identify fake images from real images without any loss. In terms of the above formulae, it means maximizing both of them at the same time. As a result, the final loss function for the discriminator has the following components:

$$L^{(D)} = max[log(D(x)) + log(1 - D(G(z)))]$$

6.5 Generator Loss

As it is a competitive game between a discriminator and a generator, the generator should have a goal of minimizing the result of the above loss function for the discriminator simultaneously.

As a result of this principle, the loss function for the generator should be a minimized version of the above formula:

$$L^{(G)} = min[log(D(x)) + log(1 - D(G(z)))]$$

So, during a training phase we are required to calculate both of the formulas for each data point. We can write the combination of the two formulas as following:

$$L = minmax[log(D(x)) + log(1 - D(G(z)))]$$

As we are trying to apply the loss function for the whole dataset, we need to derive the expected result of the equation. In other words, D and G play the following two-player minimax game with value function $L(G, D)$ [3]:

$$min_G max_D L(G, D) = E_{x \sim p_r(x)}[logD(x)] + E_{z \sim p_z(z)}[log(1 - D(G(z))]$$

$$= E_{x \sim p_r(x)}[logD(x)] + E_{x \sim p_q(x)}[log(1 - D(x)]$$

[3]

6.6 StyleGAN and its differences

Even though GAN models are very recent technologies in Machine Learning, they have shown breathtaking results in many different domains. However, there are some areas that require more specific implementations of GAN where customization made to the model can generate some part of the image more realistic even with high-resolutions. The main challenge in GAN models is more control on the output coming out of the implementation, as some specific features like pose of the face, shape of the face and hair style in a face image. A Style-Based Generator called StyleGAN is able to solve this issue of having more control over image features. The main idea behind this solution is to create many images starting from low resolution to high resolution through an iterative and gradual change.

Actually, there are a few important components in the StyleGAN network ([6]) that makes generating highly realistic synthetic images possible:

- Progressive GAN - The idea of progressive training used in StyleGAN [6] has very important takeaways: if the layer is lower, then the resolution is also lower and it can create more low frequency features. By progressively training from lower to higher features we can achieve an image with extensively visible details composed of lower and higher frequencies. In order to do that the StyleGAN paper divides the image features into three types of different resolutions (as shown in the Figure 32):
  - **coarse type**: the resolution is up until 8px x 8px

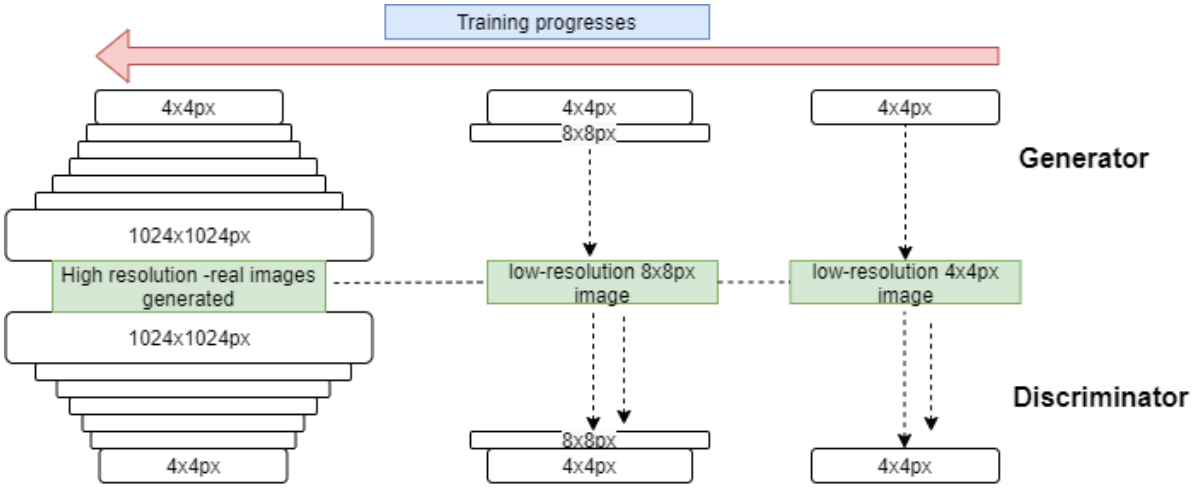

Figure 32 StyleGAN step by step growing from very low resolution to higher resolution

  - **middle type**: the resolution is from 16px x 16px up until 32px x 32px
  - **fine type**: the resolution is from 64px x 64px up until 1024px x 1024px

- Improved baseline by using bi-linear up/down sampling operations
- The mapping network is a bunch of fully connected layers where the mapping occurs between two latent spaces and this is used in style mixing
- AdaIN operations - It is specific normalization technique where each channel was scaled based on a zero mean and a unit variance and after that style based biases and scaling are applied
- Style mixing - switching from one latent space (vector) to another randomly while generating an image. This regularization technique prevents the network from assuming that adjacent styles are correlated. [6]

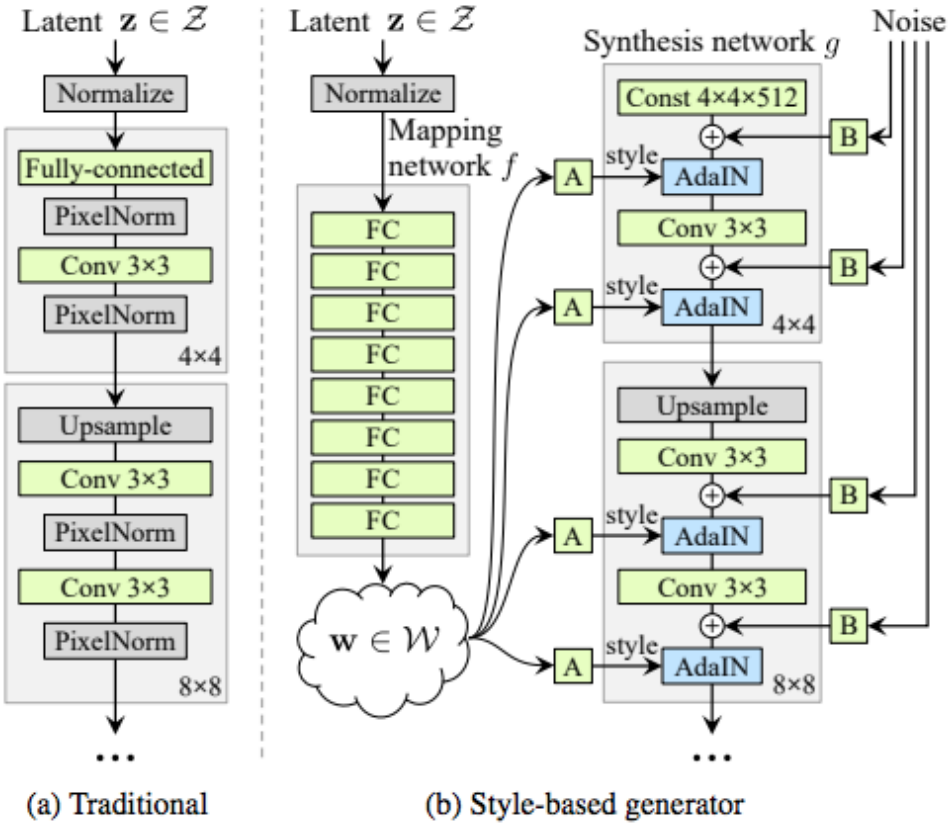

Figure 33 In a traditional generator [7] the latent code was fed though the input layer only. In the StyleGAN [6] first they map the input to an intermediate latent space W, which then controls the generator through adaptive instance normalization (AdaIN) at each convolution layer. Gaussian noise is added after each convolution, before evaluating the non-linearity. Here "A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input. The mapping network f consists of 8 layers and the synthesis network g consists of 18 layers — two for each resolution (42 10242). The output of the last layer is converted to RGB using a separate 1 × 1 convolution, similar to Karras et al. [7]. StyleGAN generator has a total of 26.2M trainable parameters, compared to 23.1M in the traditional generator.

7 Experiments, Results, Analysis

7.1 Results of the Clustering

The clustering part for the collected dataset has been done in order to find out the uniqueness of the signs from each other and help the augmentation techniques to possibly consider them. Even though it is not challenging for the deaf-mute people to distinguish some very similar signs visually, it is an extremely difficult task for any clustering algorithm to successfully determine the class of the image even after a few phases of feature extraction. The classification part is also one of the biggest parts for future research as many improvements can be applied and there are some more techniques that are worth implementing and testing the effects.
In this section all the results of image classification will be demonstrated with images and statistics as a rate of identification of unique signs, and the comparison will be made among the implemented algorithms.

As mentioned in the section 4 a few different classification algorithms have been reviewed and implemented on the dataset. Each of them has been applied with the help of transfer learning on the dataset and the results are very different in terms of accuracy, success rate, and the execution speed. The following tables are describing the end result for all of the 3 different feature extractions and clustering algorithms. Each table has got three columns only with the following description:

- **Label**: A unique identity of the label assigned to the clustered images. As there are 24 static letters in AzSL the number of end classes (also labels) are no more than 24 in each table.
- **MPL - Most probable letter**: One of the 24 static letters that have the maximum number of occurrences in the clustered label. For example, if the **Most probable letter** for the label 2 is A, it means that among the images that have been labeled as 2 by the clustering algorithm, the sign A has the maximum number of occurrences. Intuitively, it is a right way to assume that based on the similar feature classification of the images with the true label A, they have got thrown into the same bucket (in this case, the bucket labeled with number 2).
- **Max**: the number of the images found in the specified label that is corresponding to the most probable letter.

Table 1 VGG16 clustering results

Label	Max	MPL
6	236	A
20	300	A
11	213	N
0	106	A
15	68	S
4	49	EE
2	36	J
14	128	B
1	18	V
3	367	M
8	7	B
13	9	B
5	12	V
23	21	J
22	8	T
16	103	I
7	104	I
18	103	I
10	5	L
12	24	M
19	40	M
21	27	O
9	27	O
17	35	O
# of unique letters		**5**

Table 2 InceptionV3 clustering results

Label	Max	MPL
18	108	O
9	271	A
7	72	B
3	206	SH
23	129	B
5	112	T
20	137	A
13	81	A
1	44	O
16	48	N
4	84	H
12	205	M
6	134	F
2	47	H
10	166	N
0	6	X
17	49	EE
19	108	M
24	7	B
14	50	N
11	136	O
22	103	I
21	103	I
8	104	I
15	103	I
# of unique letters		**6**

Table 3 MediaPipe clustering results

Label	Max	MPL
2	129	A
21	118	A
4	177	I
22	97	EE
6	99	B
3	80	SH
13	73	U
12	105	A
8	167	O
14	103	II
20	58	O
7	121	I
0	129	N
18	174	I
15	150	H
23	54	E
10	150	F
19	76	M
9	65	M
11	151	EE
16	59	L
5	130	M
17	87	P
1	96	Q
# of unique letters		**11**

Table 4 MediaPipe (foreground extracted) clustering results

Label	Max	MPL
0	86	B
2	62	S
3	61	E
14	24	O
22	63	O
6	43	V
1	90	U
11	51	A
7	114	A
21	15	B
18	50	II
12	15	EE
8	51	C
15	122	A
19	67	A
23	18	SH
16	84	M
20	80	EE
17	48	EE
4	89	H
13	44	L
5	69	T
9	68	T
10	40	Q
# of unique letters		**11**

When it comes to the evaluation of the classification algorithms and their comparison, I will describe a customized metric to find the winner among them. It goes without saying that what we are trying to maximize in any of the classification algorithms in this dataset is a number of the uniquely detected letters by the model. In other words, the metric is the number of labels where the most probable letter of that label doesn't appear among the list of most probable letters of the rest of the labels. We can find out the values of the custom metric for each classification algorithm in the last raw of each table. We can summarize them in a small stats table as in the following:

Table 5 Comparison of the metrics of the classification algorithms.

Algorithm	**# of unique letters**
VGG16	5
Inception V3	6
MediaPipe	11
MediaPipe (foreground extracted)	11

As we can see from the statistics table, the MediaPipe model has shown a better result than any of the experimented models. Actually, there is a significant rationale behind this result. As we already know from the section 4, the MediaPipe Hand recognition model has been trained exclusively to detect hands, gestures, fingers, and even very detailed segment lines and points on the fingers. So, when we run the model on our dataset, it first tries to extract all the important and attention-worthy features from the images related only to the hand shown in the image. Another crucial point about the MediaPipe is that it can be configured to ignore some unrelated, though generally important parts from the image such as if it is a left hand or a right hand, the color of the hand, depth of the hand relative to the image, position of the hand in the image, etc. It is very useful in our case, as our main concern is the unique form of the fingers which we understand as a sign.
A higher emphasis also should be given to the last row of the table describing comparison of the models. In order to improve the correct segmentation of the MediaPipe model, we have tried to preprocess the whole dataset based on the depth of the image and extract only foreground from the image. We have applied it based on the assumption that when people take pictures of their hands, they locate the camera in a way that their hand becomes one of the initial layers of the resulting image. After the extraction the whole focus in the image gathers around the hand and it helps us to reduce the noise in the image. The following figure (Figure 34) displays some of the samples from the foreground extracted dataset.

In order to extract the foreground from the images, the GrabCut algorithm [12] has been applied on the whole dataset. The GrabCut algorithm is an interactive foreground extraction algorithm which is using Iterated Graph Cuts. It would be impossible to run the GrabCut algorithm before the classification models as it would take excessive time and computing power. In order to speed

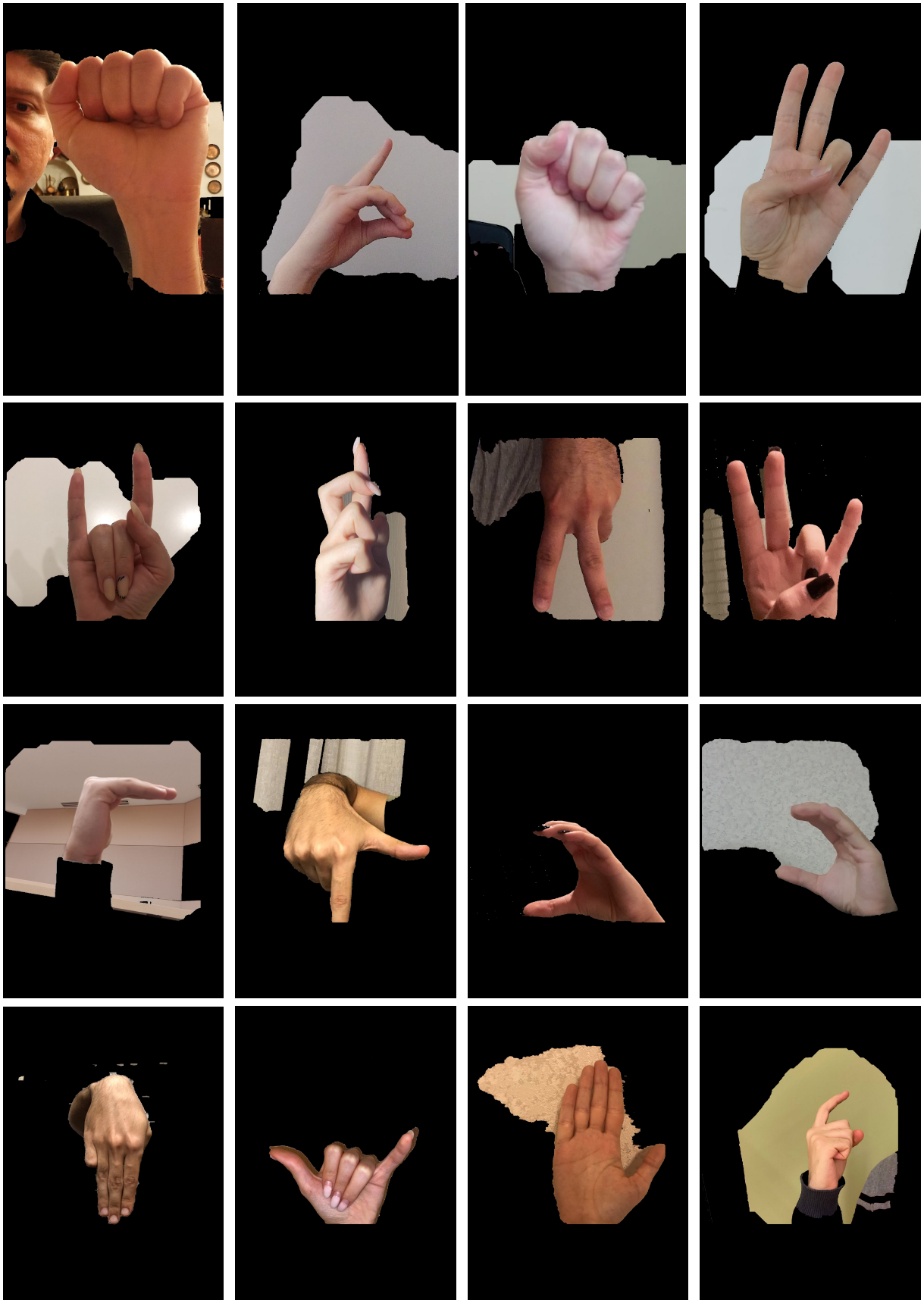

Figure 34 Sample images from the dataset of foreground extraction using the GrabCut algorithm [12]

up the process of classification on the foreground extracted images, I have used the GrabCut algorithm as a preparation tool for the dataset. The algorithm was applied on the dataset with 10 iterations in order to optimize the precision of the cut on the images.

The foreground extraction was successfully done and then the resulted dataset was fed into the MediaPipe model in order to see an improvement in the classification result. However, as we can see from the Table 5 there was no change in the model performance. The reason behind this consistency is very intuitive to grasp: the MediaPipe feature extraction model for hands was trained to detect a hand first in the image and then pull out a unique set of features regarding the position, form and many other details related to the hand. Therefore, when we run the MediaPipe model on the foreground extracted dataset, the only change in the working flow of the model is finding hands in images with more ease. Extraction of the key points related to the located hand doesn't change and it gives the same result as the default model running on the raw dataset.

Despite the fact that assigning the most probable letter to the found label makes intuitive sense, it has many flaws. First of all, as we have already mentioned above there are many labels that have been assigned with the same letters. Secondly, there are some static signs that have not been assigned to any of the found labels. In addition to these, how can we assure ourselves in finding the right letter by the most probable letter if the second most probable letter is very close to the first one? In order to address these issues properly, we need to have a visual check on all of the labels and their top three most probable letters. This will definitely give us a broader view on the similarity of the letters and will help us to improve the classification algorithm by decreasing the closeness of the images inside the same cluster and increase the betweenness among the different clusters. In order to do that, the visualisation of the top 5 predictions has been generated for different labels. The Figure 35 has 4 pie charts:

- The top two charts describe far better prediction examples. Based on the top 5 percentages we can conclude with a moderate level of confidence that the **label0** should be the letter A and the **label14** should be the letter T.
- The bottom two charts, on the other side, show a very not-deterministic clustering result. As we can see, almost all 5 classes have got the same level of importance. It means that those 5 letters have a very high similarity between themselves and the classifiers are not capable of distinguishing them easily. That's why the **label2** and **label20** cannot be assigned any letter as a result of the classification.

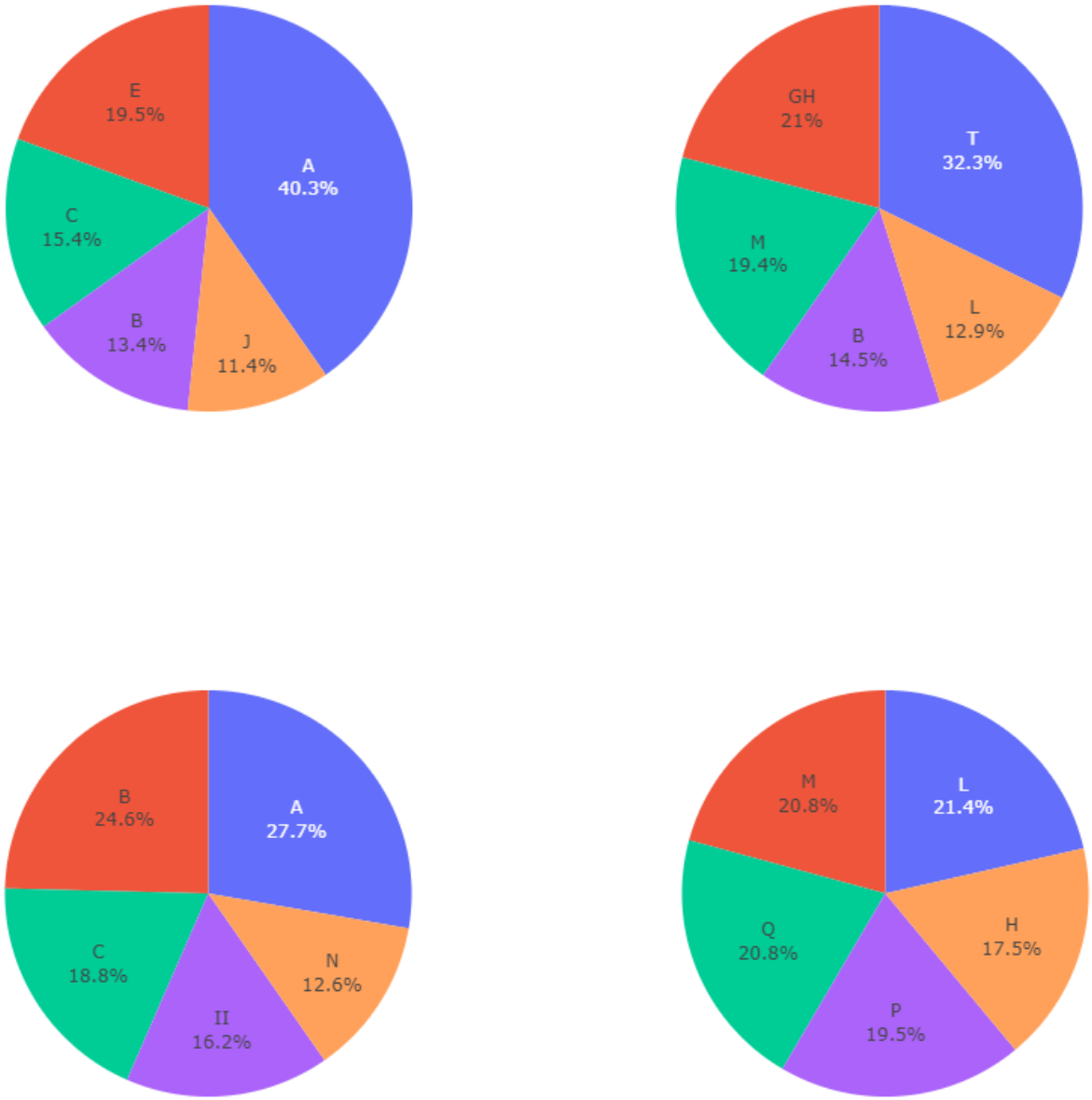

Figure 35 Visualisation of the top 5 predictions for 4 labels. Top left: **label0**; Top right: **label14**; Bottom left: **label2**; Bottom right: **label20**.

7.2 Results of the Image and Video Augmentation

The following figure shows a few samples from the result of the image augmentation. In order to maximize the size of the resulting dataset for having better performance in future models, each image in the original dataset is augmented 10 times with different set of filters and with random parameter values.

As it is obvious from the Figure 36 sometimes results of the augmentations can be uncontrollable and lead to a generation of useless and invalid images/videos. That's why there is no statistically approved way of evaluation of the job done. However, visually checking the results can give us some certainty about the validation of the techniques and we can adjust the solution accordingly. As a result of the image augmentation, we could have achieved a dataset of around 140,000 images and around 30,000 videos. Considering the huge appetite of Machine Learning models for a dataset, this resulting dataset can be used for different purposes.

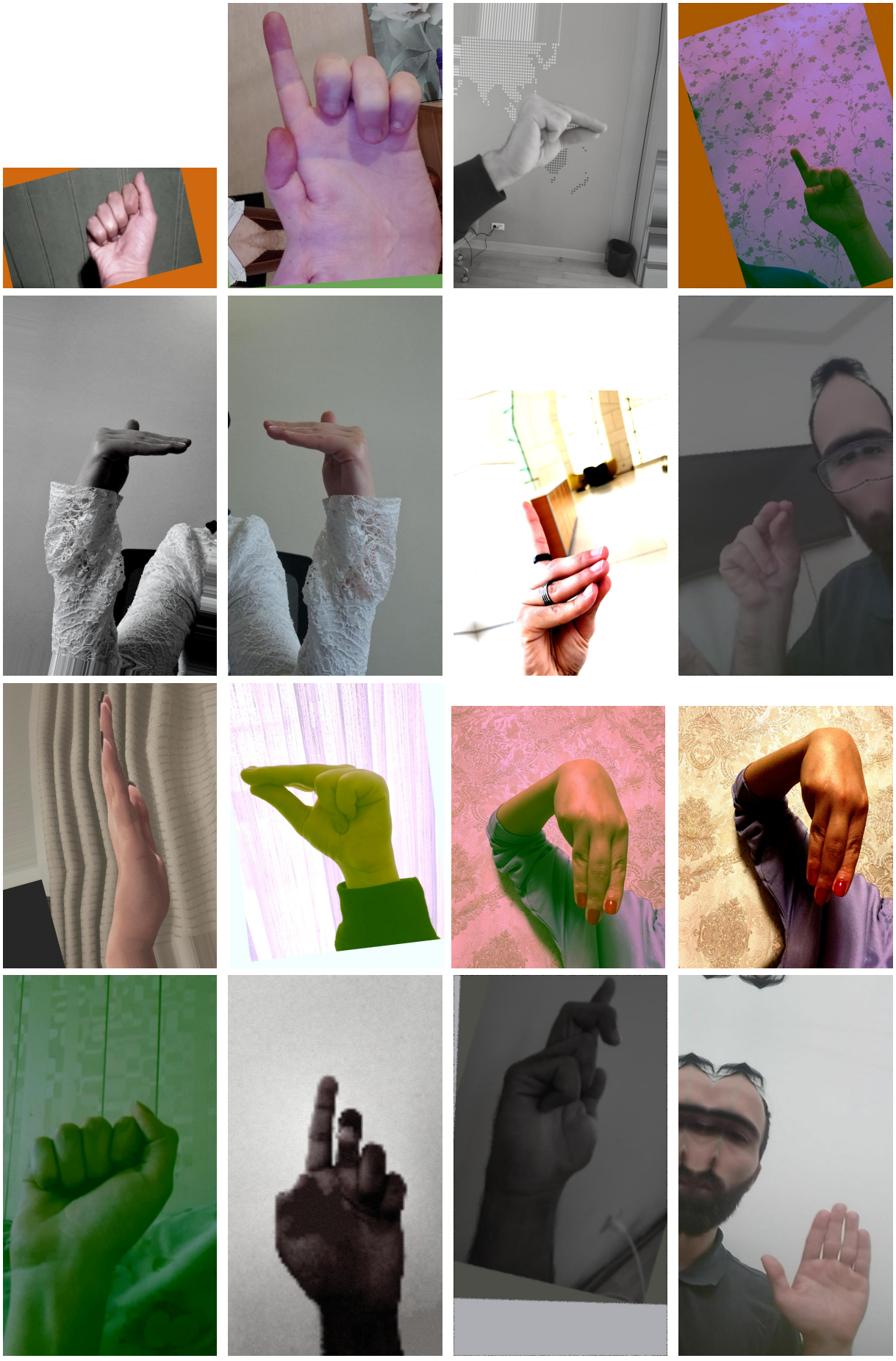

Figure 36 Sample images from the dataset of augmented images

7.3 Results of the GAN

The GAN model can take a lot of time for training as for each epoch we are training a Discriminator and a Generator model. As shown in the following figures, both of the models consists of many layers and they both are trained at the same time. As mentioned in the original Github repo [15] of StyleGAN Pytorch there are a few configurations that can be used for parameter tuning in order to see the best result generating model. After finding the best resolution and GPU size, the model starts to run and logs all the performance metrics into a json file. It automatically calculates the Frechet Inception Distance after each tick and saves the result.

The metrics we have used for both of the models were the results of the loss functions. Both of the model have been improving their results after each tick, however, the overall result were not satisfying when I used the original dataset.

It is obvious from the following figure that the fake images created by the generator are not realistic at all. Actually, it is not something unexpected considering the size of the dataset used. We can also see this in action with the help of loss curve during the training phase. It should be mentioned that as the StyleGAN model is a pre-trained network, we are fine tuning it with the help of `torch.optim.Adam` optimizer. The figures 41 and 42 are describing many things about the training process. When the training process starts, the discriminator cannot distinguish fake images from the real ones at all and that's why there is maximum loss value. However, as the training continues it updates its weights quickly in order to detect the fake ones and improves its performance rapidly. On the other hand, the generator starts the training by successfully cheating the discriminator, but fails very fast as its counterpart learns all the tricks very fast. That's why we can see a sudden rise in the loss function of the generator. But with the help of adaptive instance normalization, it starts to create more realistic looking images and make it hard to distinguish them from the real images for the discriminator.

As it is mentioned in the paper [17] the size of the dataset matters in GAN network. The most optimized StyleGAN implementation in PyTorch requires at least 25.000 original images to be used for the training of the models for the best results. In our case, it was only about 10.000. In order to see it in action, I have ran both of the models on the augmented dataset which is about 120.000. The disadvantage of this dataset is that a huge percentage of the dataset is not original and somehow looks fake and duplicate. That's why when the Discriminator validates them as a real image, the Generator will also start to generate those kinds of images with even worse features. Actually, the model result proves this right. The results were better than the first one, however, not something useful. As a result of this, we can decide that the model can generate valid, real-looking images as long as the original dataset is large enough. This implementation can be optimized in many ways for future study.

Generator	Parameters	Buffers	Output shape	Datatype
mapping.fc0	262656	-	[8, 512]	float32
mapping.fc1	262656	-	[8, 512]	float32
mapping	-	512	[8, 16, 512]	float32
synthesis.b4.conv1	2622465	32	[8, 512, 4, 4]	float32
synthesis.b4.torgb	264195	-	[8, 3, 4, 4]	float32
synthesis.b4:0	8192	16	[8, 512, 4, 4]	float32
synthesis.b4:1	-	-	[8, 512, 4, 4]	float32
synthesis.b8.conv0	2622465	80	[8, 512, 8, 8]	float32
synthesis.b8.conv1	2622465	80	[8, 512, 8, 8]	float32
synthesis.b8.torgb	264195	-	[8, 3, 8, 8]	float32
synthesis.b8:0	-	16	[8, 512, 8, 8]	float32
synthesis.b8:1	-	-	[8, 512, 8, 8]	float32
synthesis.b16.conv0	2622465	272	[8, 512, 16, 16]	float32
synthesis.b16.conv1	2622465	272	[8, 512, 16, 16]	float32
synthesis.b16.torgb	264195	-	[8, 3, 16, 16]	float32
synthesis.b16:0	-	16	[8, 512, 16, 16]	float32
synthesis.b16:1	-	-	[8, 512, 16, 16]	float32
synthesis.b32.conv0	2622465	1040	[8, 512, 32, 32]	float32
synthesis.b32.conv1	2622465	1040	[8, 512, 32, 32]	float32
synthesis.b32.torgb	264195	-	[8, 3, 32, 32]	float32
synthesis.b32:0	-	16	[8, 512, 32, 32]	float32
synthesis.b32:1	-	-	[8, 512, 32, 32]	float32
synthesis.b64.conv0	2622465	4112	[8, 512, 64, 64]	float16
synthesis.b64.conv1	2622465	4112	[8, 512, 64, 64]	float16
synthesis.b64.torgb	264195	-	[8, 3, 64, 64]	float16
synthesis.b64:0	-	16	[8, 512, 64, 64]	float16
synthesis.b64:1	-	-	[8, 512, 64, 64]	float32
synthesis.b128.conv0	1442561	16400	[8, 256, 128, 128]	float16
synthesis.b128.conv1	721409	16400	[8, 256, 128, 128]	float16
synthesis.b128.torgb	132099	-	[8, 3, 128, 128]	float16
synthesis.b128:0	-	16	[8, 256, 128, 128]	float16
synthesis.b128:1	-	-	[8, 256, 128, 128]	float32
synthesis.b256.conv0	426369	65552	[8, 128, 256, 256]	float16
synthesis.b256.conv1	213249	65552	[8, 128, 256, 256]	float16
synthesis.b256.torgb	66051	-	[8, 3, 256, 256]	float16
synthesis.b256:0	-	16	[8, 128, 256, 256]	float16
synthesis.b256:1	-	-	[8, 128, 256, 256]	float32
synthesis.b512.conv0	139457	262160	[8, 64, 512, 512]	float16
synthesis.b512.conv1	69761	262160	[8, 64, 512, 512]	float16
synthesis.b512.torgb	33027	-	[8, 3, 512, 512]	float16
synthesis.b512:0	-	16	[8, 64, 512, 512]	float16
synthesis.b512:1	-	-	[8, 64, 512, 512]	float32
---	---	---	---	---
Total	28700647	699904	-	-

Figure 37 Generator model summary

Discriminator	Parameters	Buffers	Output shape	Datatype
b512.fromrgb	256	16	[8, 64, 512, 512]	float16
b512.skip	8192	16	[8, 128, 256, 256]	float16
b512.conv0	36928	16	[8, 64, 512, 512]	float16
b512.conv1	73856	16	[8, 128, 256, 256]	float16
b512	-	16	[8, 128, 256, 256]	float16
b256.skip	32768	16	[8, 256, 128, 128]	float16
b256.conv0	147584	16	[8, 128, 256, 256]	float16
b256.conv1	295168	16	[8, 256, 128, 128]	float16
b256	-	16	[8, 256, 128, 128]	float16
b128.skip	131072	16	[8, 512, 64, 64]	float16
b128.conv0	590080	16	[8, 256, 128, 128]	float16
b128.conv1	1180160	16	[8, 512, 64, 64]	float16
b128	-	16	[8, 512, 64, 64]	float16
b64.skip	262144	16	[8, 512, 32, 32]	float16
b64.conv0	2359808	16	[8, 512, 64, 64]	float16
b64.conv1	2359808	16	[8, 512, 32, 32]	float16
b64	-	16	[8, 512, 32, 32]	float16
b32.skip	262144	16	[8, 512, 16, 16]	float32
b32.conv0	2359808	16	[8, 512, 32, 32]	float32
b32.conv1	2359808	16	[8, 512, 16, 16]	float32
b32	-	16	[8, 512, 16, 16]	float32
b16.skip	262144	16	[8, 512, 8, 8]	float32
b16.conv0	2359808	16	[8, 512, 16, 16]	float32
b16.conv1	2359808	16	[8, 512, 8, 8]	float32
b16	-	16	[8, 512, 8, 8]	float32
b8.skip	262144	16	[8, 512, 4, 4]	float32
b8.conv0	2359808	16	[8, 512, 8, 8]	float32
b8.conv1	2359808	16	[8, 512, 4, 4]	float32
b8	-	16	[8, 512, 4, 4]	float32
b4.mbstd	-	-	[8, 513, 4, 4]	float32
b4.conv	2364416	16	[8, 512, 4, 4]	float32
b4.fc	4194816	-	[8, 512]	float32
b4.out	513	-	[8, 1]	float32
---	---	---	---	---
Total	28982849	480	-	-

Figure 38 Discriminator model summary

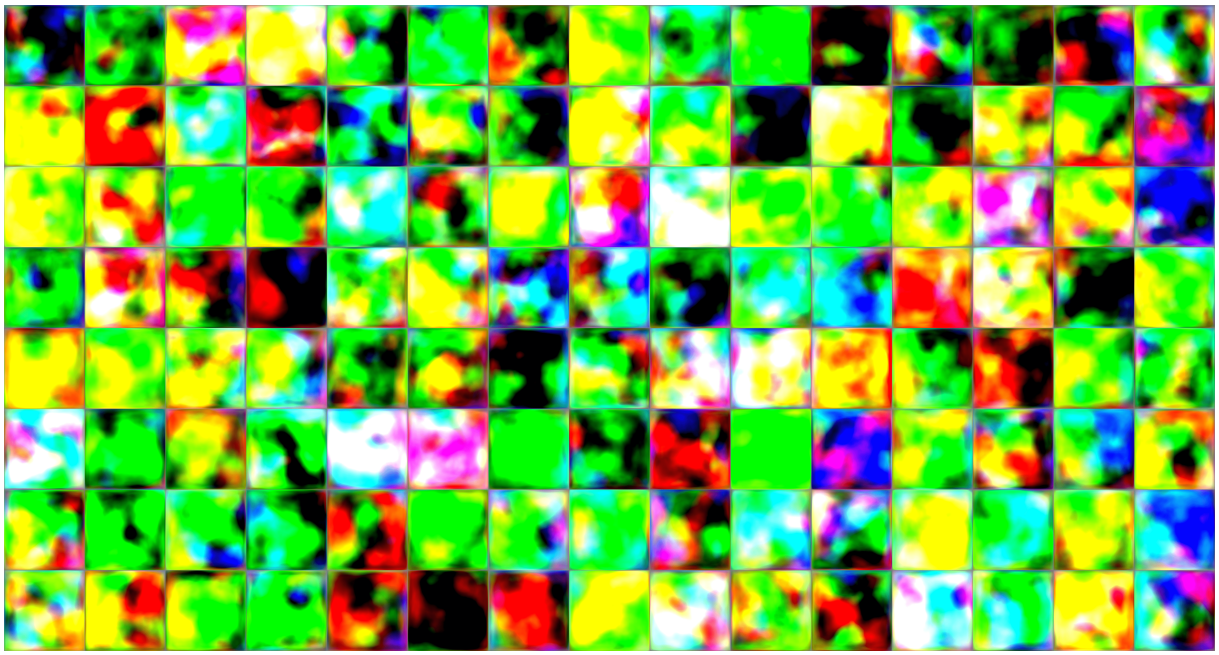

Figure 39 First fake images snapshot by the StyleGAN model on the original dataset after 4 hours of training

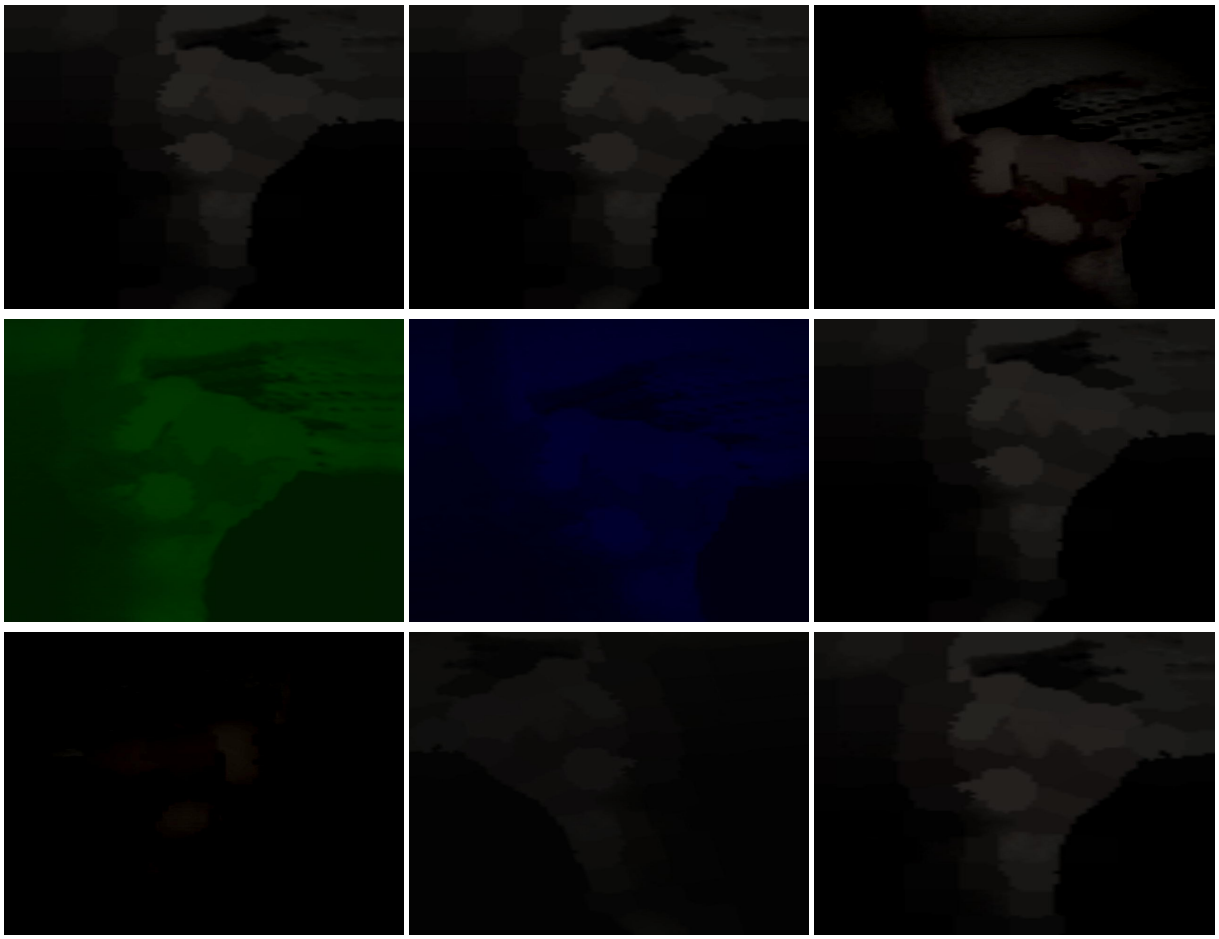

Figure 40 Resulted fake images snapshot by the StyleGAN model on the augmented dataset after 20 hours of training

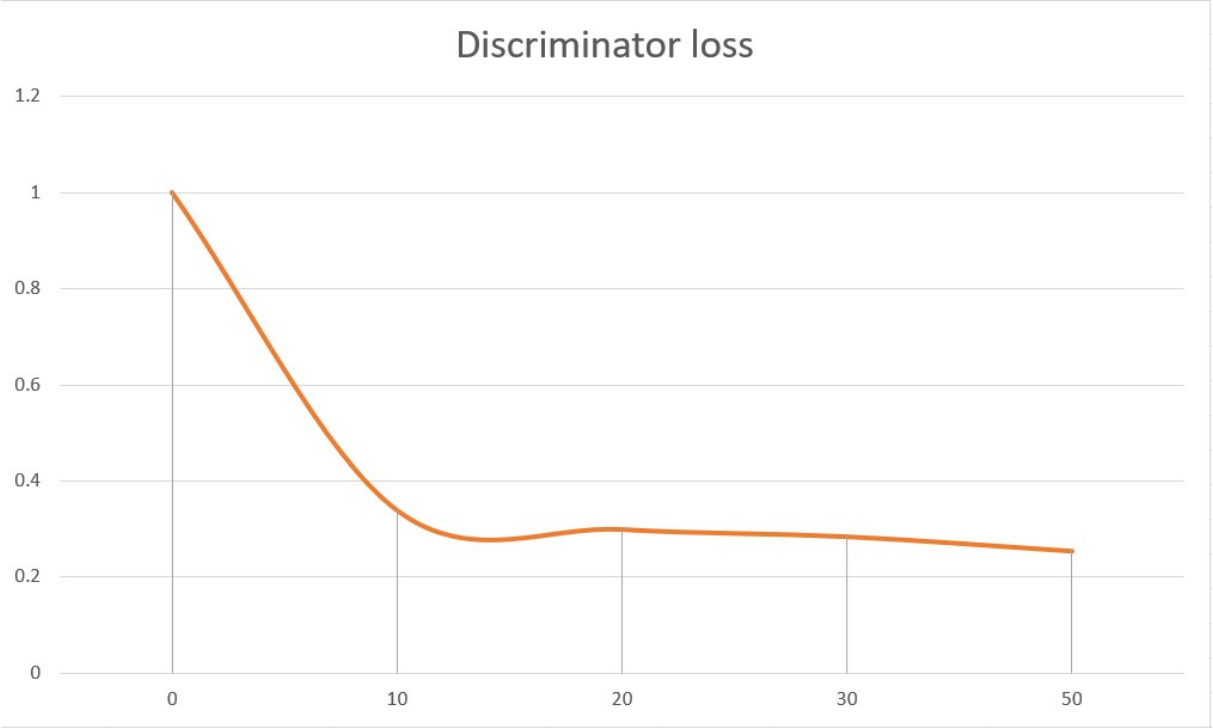

Figure 41 Discriminator loss during training process

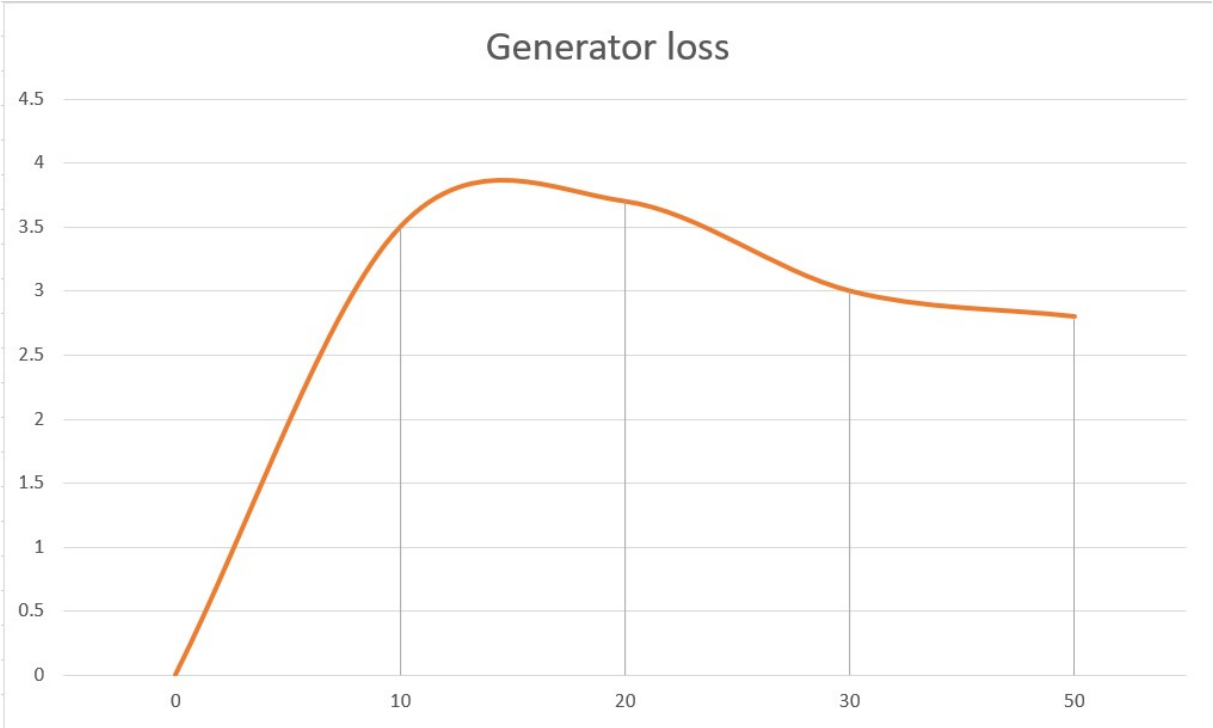

Figure 42 Generator loss during training process

Table 6 Metrics evaluated on the original and augmented datasets.

Metrics	**10K (original)**	**100K (augmented)**
FID score	~ 17.5	~ 15.4
Inception score	~ 1.9	~ 2.3

Frechet Inception Distance is a measure of similarity between the original distribution of images and the fake ones. So, in other words, the higher its value, the better the synthetic images are similar to the original images [5]. As we can see from the above table, the FID score had increased slightly in the case of the augmented images. However, in general, the FID score is considered to be acceptable for fake images when it is around 5.

Inception score has been used to validate the quality of the generated images relative to the original one before the FID score. This score was introduced in [13] for Inception V3 model and was used in comparing the generated label distribution with the original one. That's why when the generator can create a set of images with more diversity, the label distribution is going to be high. In other words, the high score is better for this score. For this score, the same pattern is present. The Inception score for the augmented dataset is a little bit higher than the original dataset, but generally the scores are not high enough to generate real looking images.

8 Discussion and Conclusions

This thesis has described all the important steps for a data collection, classification, augmentation, and generation for a Sign Language project. The previous sections discuss all the important parts during the data collection, what kind of improvements have been conducted in order to construct the dataset with equally distributed classes, which feature extraction Machine Learning models have been implemented to cluster the dataset correctly, in what stage image and video augmentation techniques were applied, and how the Generative Adversarial Network was trained with the original and the augmented dataset. As a future study, a better multi-stage approach will be experimented to cluster the images into classes with a higher success rate, and a bigger original dataset of images should be used with better configured parameters in the mentioned StyleGAN model [15] for more realistic synthetic images.

References

[1] Mohamed Aktham Ahmed et al. "A Review on Systems-Based Sensory Gloves for Sign Language Recognition State of the Art between 2007 and 2017". In: *Sensors* 18 (July 2018), p. 2208. DOI: 10.3390/s18072208.

[2] Cao Dong, Ming C. Leu, and Zhaozheng Yin. "American Sign Language Alphabet Recognition Using Microsoft Kinect". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2015.

[3] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[4] M Grif and Y Kondratenko. "Development of a software module for recognizing the fingerspelling of the Russian Sign Language based on LSTM". In: *Journal of Physics: Conference Series* 2032 (Oct. 2021), p. 012024. DOI: 10.1088/1742-6596/2032/1/012024.

[5] Martin Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: (2017). DOI: 10.48550/ARXIV.1706.08500. URL: https://arxiv.org/abs/1706.08500.

[6] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. DOI: 10.48550/ARXIV.1812.04948. URL: https://arxiv.org/abs/1812.04948.

[7] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. DOI: 10.48550/ARXIV.1812.04948. URL: https://arxiv.org/abs/1812.04948.

[8] *Mediapipe hands*. URL: https://google.github.io/mediapipe/solutions/hands (visited on 04/01/2021).

[9] Ozge Mercanoglu and Hacer Keles. *AUTSL: A Large Scale Multi-modal Turkish Sign Language Dataset and Baseline Methods*. Aug. 2020.

[10] Umberto Michelucci. *An Introduction to Autoencoders*. 2022. DOI: 10.48550/ARXIV.2201.03898. URL: https://arxiv.org/abs/2201.03898.

[11] Ilias Papastratis et al. "Artificial Intelligence Technologies for Sign Language". In: *Sensors* 21.17 (2021). ISSN: 1424-8220. DOI: 10.3390/s21175843. URL: https://www.mdpi.com/1424-8220/21/17/5843.

[12] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut -Interactive Foreground Extraction using Iterated Graph Cuts". In: *ACM Transactions on Graphics (SIGGRAPH)* (Aug. 2004). URL: https://www.microsoft.com/en-us/research/publication/grabcut-interactive-foreground-extraction-using-iterated-graph-cuts/.

[13] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[14] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. DOI: 10.48550/ARXIV.1409.1556. URL: https://arxiv.org/abs/1409.1556.

[15] *StyleGAN ADA Pytorch*. URL: https://github.com/NVlabs/stylegan2-ada-pytorch. (accessed: 20.04.2022).

[16] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. DOI: 10.48550/ARXIV.1512.00567. URL: https://arxiv.org/abs/1512.00567.

[17] Shengjing Wei et al. "A Component-Based Vocabulary-Extensible Sign Language Gesture Recognition Framework". In: *Sensors* 16.4 (2016). ISSN: 1424-8220. DOI: 10.3390/s16040556. URL: https://www.mdpi.com/1424-8220/16/4/556.

[18] Fan Zhang et al. *MediaPipe Hands: On-device Real-time Hand Tracking*. 2020. DOI: 10.48550/ARXIV.2006.10214. URL: https://arxiv.org/abs/2006.10214.

Appendices

A Kaggle notebooks

The following links provide all the models for Data Classification, Data Augmentation and GAN parts as repos in Kaggle:

- Clustering VGG16 model
- Clustering Inception V3 model
- Clustering MediaPipe model
- Clustering MediaPipe model on the Foreground Extracted dataset
- StyleGAN model