PROJECT TOPIC

Sign language hand gesture image classification on the MNIST Sign language dataset and farther applications.

GROUP MEMBERS:

Mouratidis Stathis
Georgoulis Ilias

Instractor: Mr.Barmparis

Project proposal:
Our primary goal is to do an image classification on the MNIST Sign language dataset. The dataset contains about 35k labeld images. The images represent the english alphabet. After we finish investigating the best models for the task, we will try to implement the model on a video of sign language letters representations.We will propably try to build a hand detection model or use an existing one to isolate hand images. Then will try to feed our model these images as input to make predictions on.

Abstract

With sufficiently large datasets, neural networks provide opportunity to train models that perform well and address challenges of existing object tracking/detection algorithms — varied/poor lighting, diverse viewpoints and even occlusion. The

main drawbacks to usage for real-time tracking/detection is that they can be complex, are relatively slow compared to tracking-only algorithms and it can be quite expensive to assemble a good dataset.

But things are changing with advances in fast neural networks.Furthermore, this entire area of work has been made more approachable by deep learning frameworks (such as the tensorflow object detection api) that simplify the process of training a model for custom object detection. More importantly, the advent of fast neural network models like ssd, faster r-cnn, rfcn etc make neural networks an attractive candidate for real-time detection (and tracking) applications.There are multiple applications for robust hand tracking like this across HCI areas (as an input device etc.).

So we decided to use these methods to create an automatic system for tracking and classifying Asl sign language hand gestures, a task that, in its entirety, could be very useful for people with disabilities and their families.

<u>**Dataset**</u>

our main dataset in this task is Sign language MNIST. We chose this dataset primarily for its usability, as it contains low-quality images that are well-balanced for image classification tasks.
The dataset format is patterned to match closely with the classic MNIST. Each training and test case represents a label (0-25) as a one-to-one map for each alphabetic letter A-Z (and no cases for 9=J or 25=Z because of gesture motions). The training data (27,455 cases) and test data (7172 cases) are approximately half the size of the standard MNIST but otherwise similar with a header row of label, pixel1,pixel2….pixel784 which represent a single 28x28 pixel image with grayscale values between 0-255.

The original hand gesture image data represented multiple users repeating the gesture against different backgrounds. The Sign Language MNIST data came from greatly extending the small number (1704) of the color images included as not cropped around the hand region of interest.

The dataset's main disadvantage is that the images have no backround and thus cannot be used in natural backrounds.

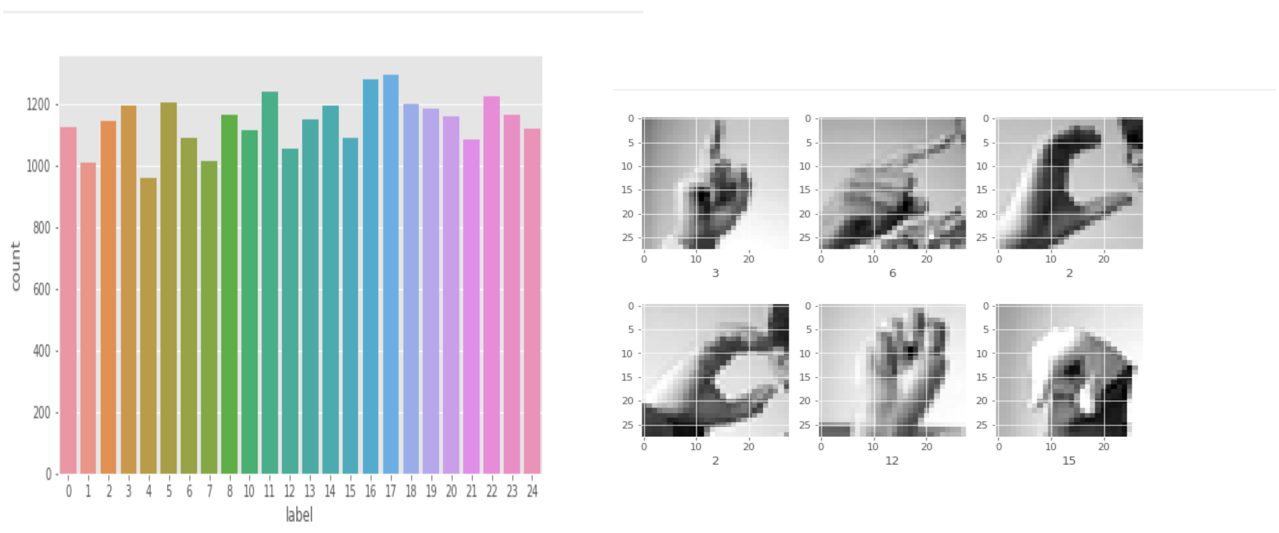
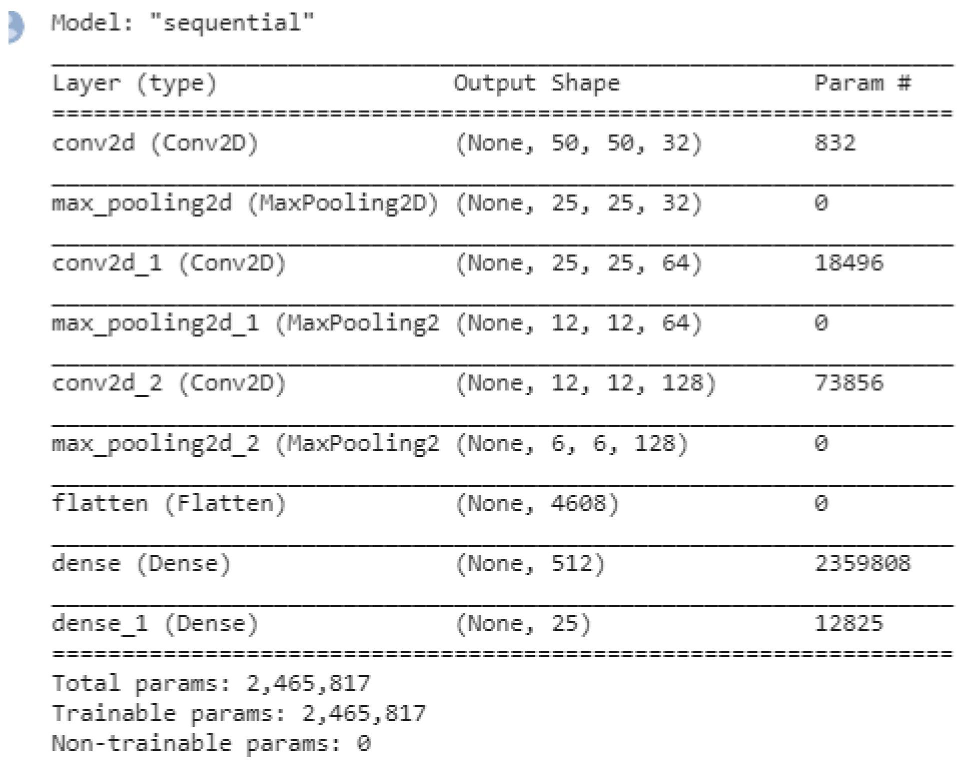


Image 1: Train data label contribution and examples.

<u>Model</u>

We experimented with and compared various models for classification.As expected, CNN was the best for the image classification compared to raw artificial networks. The 3 convolutional layer model 32,(5,5) ,64,(3,3),128,(3,3) followed by (2,2) max pooling and one simple layer aften the flatten seemed to be the best stracture.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 50, 50, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 512)               2359808
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
=================================================================
Total params: 2,465,817
Trainable params: 2,465,817
Non-trainable params: 0
```

Image 2: Model summary

We used the "same" padding and relu as activation functions, with the exception of the final convolutional layer, where we used swish, an activation function that, due to its form, appears to have good results on the sum of the models (as it has in previous models).Bigger batch sizes seemed to work better, but there were still some unexpected spikes on our learning curves. Although the accuracy of the model remained high.In table 1 are all the hyperparameters and characteristics of the model after the optimization.

Table 1:

Model 1:	
padding	Same
Activation	Relu,swish
Optimiser	Adam
Batch size	254
epochs	About 50
loss	categorical_crossentropy

Data Augumentation

We attempted to implement data augumentation during model optimization. Something that made a big difference in our models' behavior and stability. It also made the training time slower. We discovered that small values of the parameters (Table 2) were better for the new augumented data and didnt significantly increased the train time.

Table 2:

Image augmentation generators parameters	
rotation_range	15
width_shift_range	0.1
height_shift_range	0.1
shear_range	0.3
zoom_range	0.3

horizontal_flip	True

Model evaluation:

Our final model achieved 99% accuracy on the test set, which is most likely due to the similarity of the train set images as a result of the augumentation methods (more information on (2)).

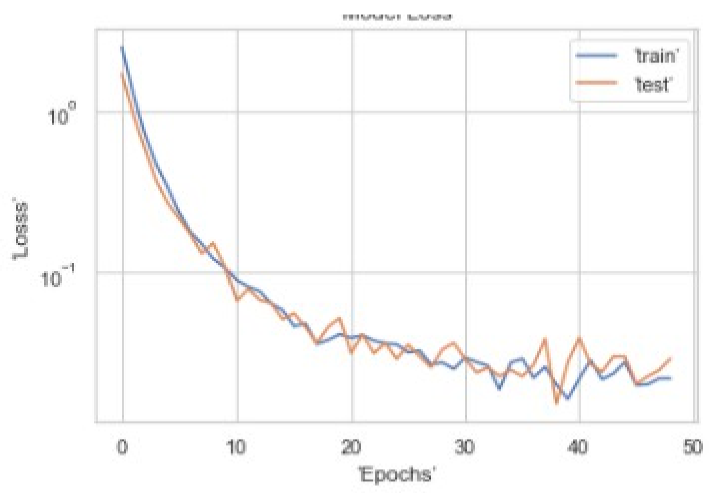


Image 3: Models learning curve

Tranfer learning:

After finishing with the model optimisation and given its very high accuracy, we thought as an evaluation method to use our model as pre trained on a similar dataset. For that, we used the Sign Language Digits Dataset consisting of about 2000, 64x64 images of sign language numbers. Using the models convolutional layers, keeping its weights untrainable and only by adding a 10 dense layer, we achieved 95% accuracy on the new dataset. Because there were more layers before the flattening () command, the model learned something after the first few epochs and then stopped to, as shown in the image.This was remedied by the removal of these dense layers.

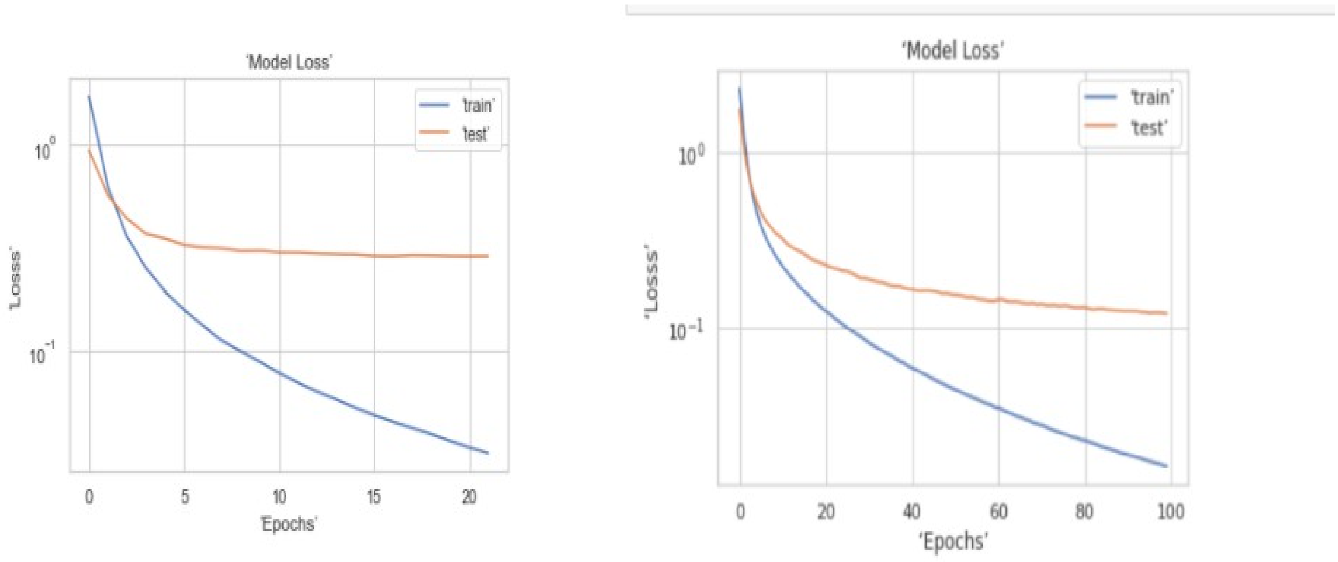


Image 4: Learning curves with and without extra dense layers.

<u>**Model use on video file**</u>

After deciding which models were best for the predictions, we decided to apply them on a live video and see if we could get live predictions of the asl letters. In order to do that, we used the open CV contribute library for python.
Before that, we randomly took some images from the video which were used for static image predictions. To accomplish this, we defined an area of interest (the roi in our jupyter notebook) and a counter.The images were taken by multiplying the counter by 150.For example, if the counter is set to 1, then the picture taken would be frame 150, if counter=2 the picture would be frame 300 and so on. We set the area of interest so that the pictures taken would be as close to the ones in the dataset as possible.
That's why some of them have cropped fingers, because if we set the height any higher, we would ruin the predictions of the other

pictures.

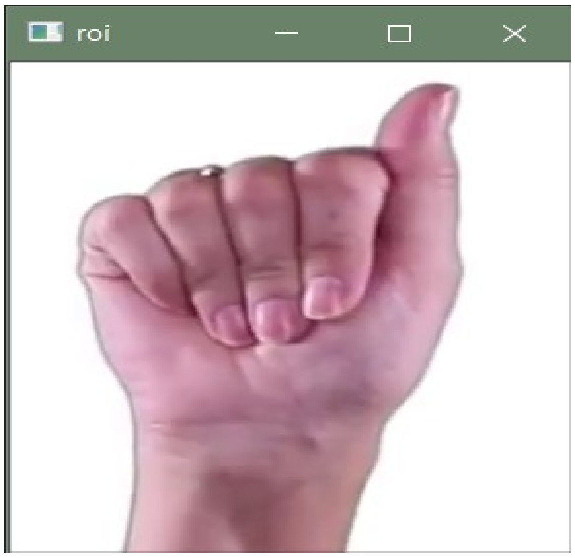


Image 5: Representation of the area of interest used to take the pictures for predictions

Regarding the live predictions, the initial idea was to create or use a pretrained object detection model which we would use for the palm recognition in the video and apply our prediction model to the detection. However, that task proved to be time-consuming, so we settled on object tracking and the cv2 library's in-built trackers.

First, we applied a mask on the video in order to remove the background and detect the movement of the hand. We achieved that using the cv2.createBackgroundSubtractorMOG2 module which subtracts the background of a stable camera feed and creates a mask that detects the movement.

The hyper-parameters history and threshold determine the history of object movement that will be taken under consideration and the sensitivity to the movement respectively.

Then we created bounding boxes that surround the moving object through cv2.findContours (mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE).

In this line, we apply the findContours on the mask, which is in black and white, black for the background and white for the moving objects, and we take the external bounding boxes and only the necessary points of the contours, meaning that we don't take lines around the moving object but the end points of the line of movement.

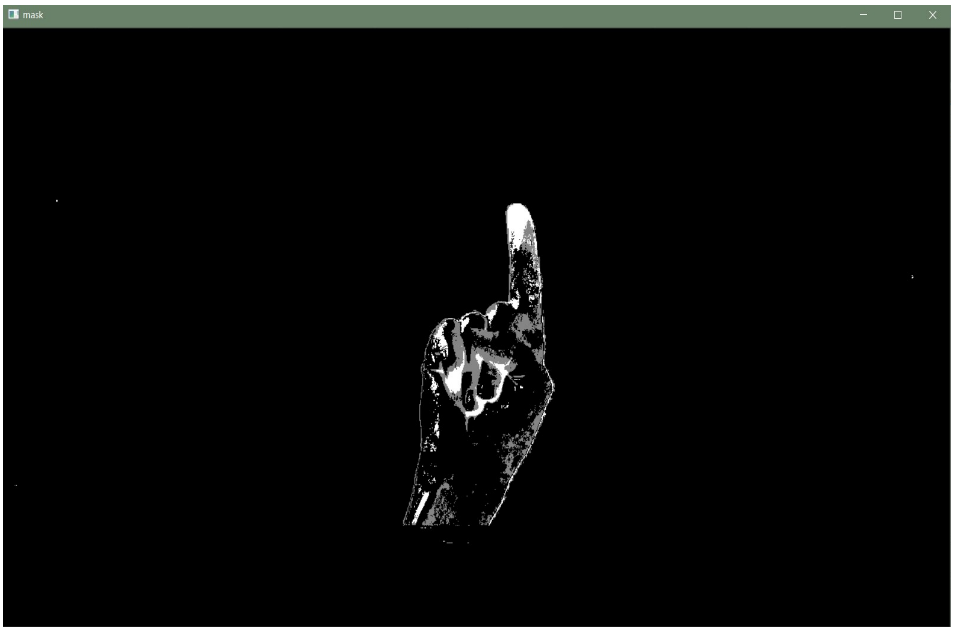


Image 6: Representation of mask used on video

Then, in order to reduce the redundant contours, since not only the hand but the fingers too move during the video, we define the area of the contour and since we want the whole hand and not the fingers, we want it to be greater than 2000.

In that area, we take the x, y coordinates and height and width of the bounding boxes, we turn the frame to a gray image and in that image we apply the coordinates above. That is the picture that we will predict with our model after making it a gray scale image and resizing it to 50x50.

At first, we applied the prediction using only the coordinates x, y, height and width in order to crop appropriately the image and take from the frame only the area surrounded by the bounding box.

However, the predictions were not good because we realized that the image was not on par with the images used for the prediction. So we tried to make the area that we cropped from the frame a little bigger and bring it as close as we could to the training images. That gave us much better results, but still we could not predict every letter.

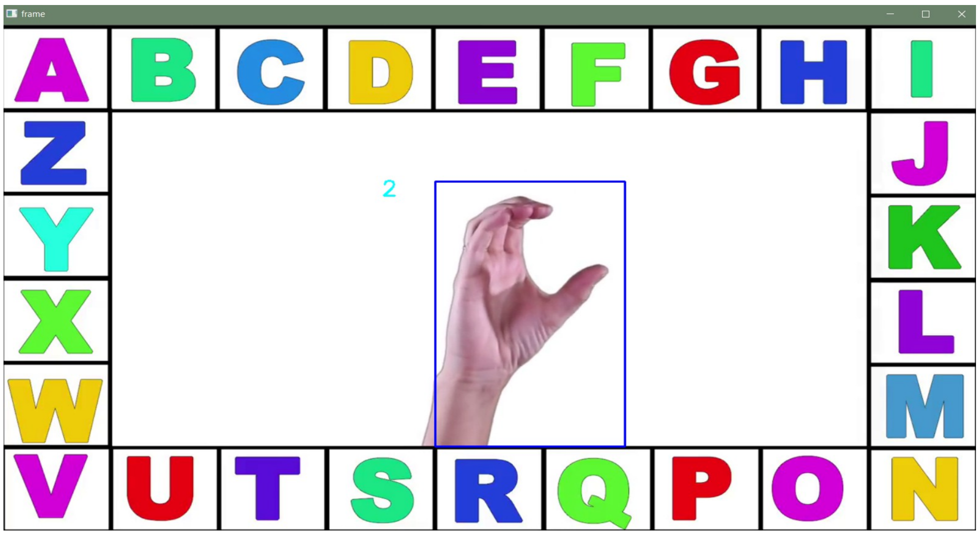
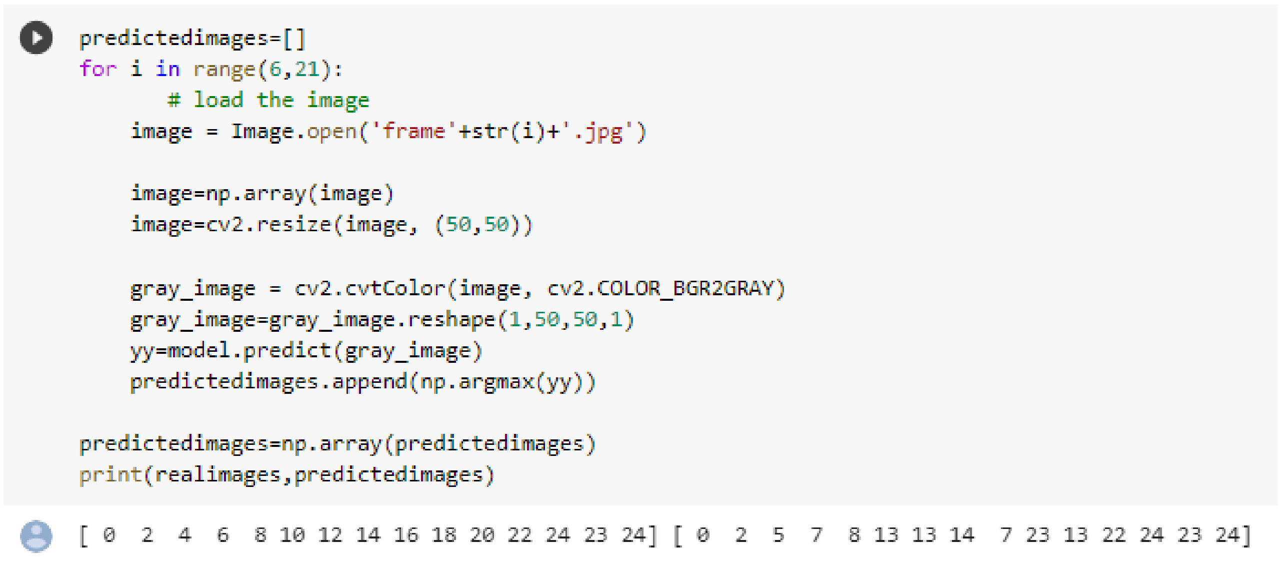


Image 7: Representation of the live predictions on the video

It is important to note that the model was trained on 50x50 images from the mnist dataset, as opposed to their original dimensions of 28x28.because when resizing the video frames to 28x28 the picture was very blurry and not good enough for accurate predictions. Resizing the dataset to 50x50 improved our predictions a lot. We tried also different seeds to see the results and found a small corellation with the predictions on the video. In some cases we got 50% accuracy and also different models appeared to make predictions in different letters better.



```
predictedimages=[]
for i in range(6,21):
      # load the image
    image = Image.open('frame'+str(i)+'.jpg')

    image=np.array(image)
    image=cv2.resize(image, (50,50))

    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_image=gray_image.reshape(1,50,50,1)
    yy=model.predict(gray_image)
    predictedimages.append(np.argmax(yy))

predictedimages=np.array(predictedimages)
print(realimages,predictedimages)
```

```
[ 0  2  4  6  8 10 12 14 16 18 20 22 24 23 24] [ 0  2  5  7  8 13 13 14  7 23 13 22 24 23 24]
```

<u>Image 8:</u> Real values and predictions of the area of intrest taken frames.

<u>Conclusions:</u>

As previously stated, the datasets we used are extremely simple (28x28 images with no backgrounds), and even after a minor resize to 50x50 pixels, the models became noticeably heavier.Real-world applications undoubtedly necessitate much larger datasets as well as capable systems.Thats why we didnt stick to our first plan of a hand detecting and tracking algorithm and moved to motion detection. Another factor that proved to be strongly related to the video results was the way boundary boxes were cropped in our attempt to match the initial dataset images.Something that also indicates the need for a more derivative dataset. Finally, given the simplicity of the material used, we consider our results satisfying.

<u>Sources:</u>

(1)https://medium.com/@victor.dibia/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflowd6bac0e4b2ce

(2)https://www.kaggle.com/datamunge/sign-language-mnist

(3)https://www.youtube.com/watch?v=O3b8lVF93jU

(4)https://www.youtube.com/watch/v=6_gXiBe9y9A&ab_channel
=LauraBergLife