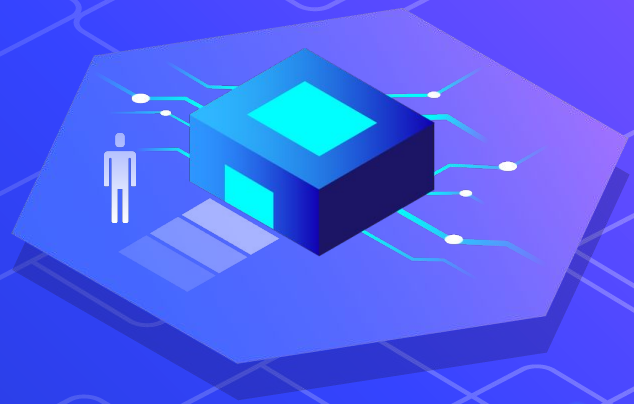
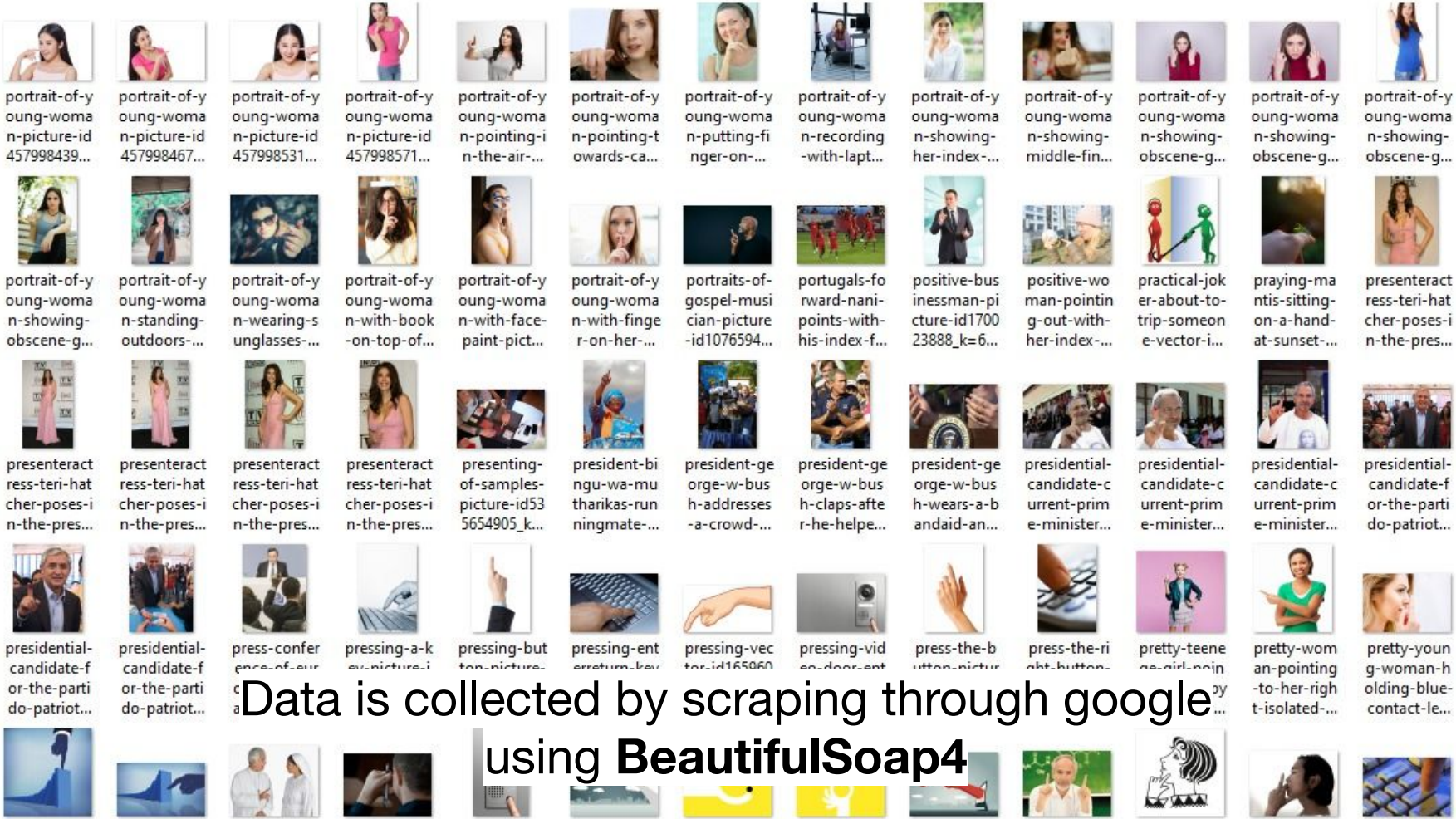
Classification of Numbers on Fingers

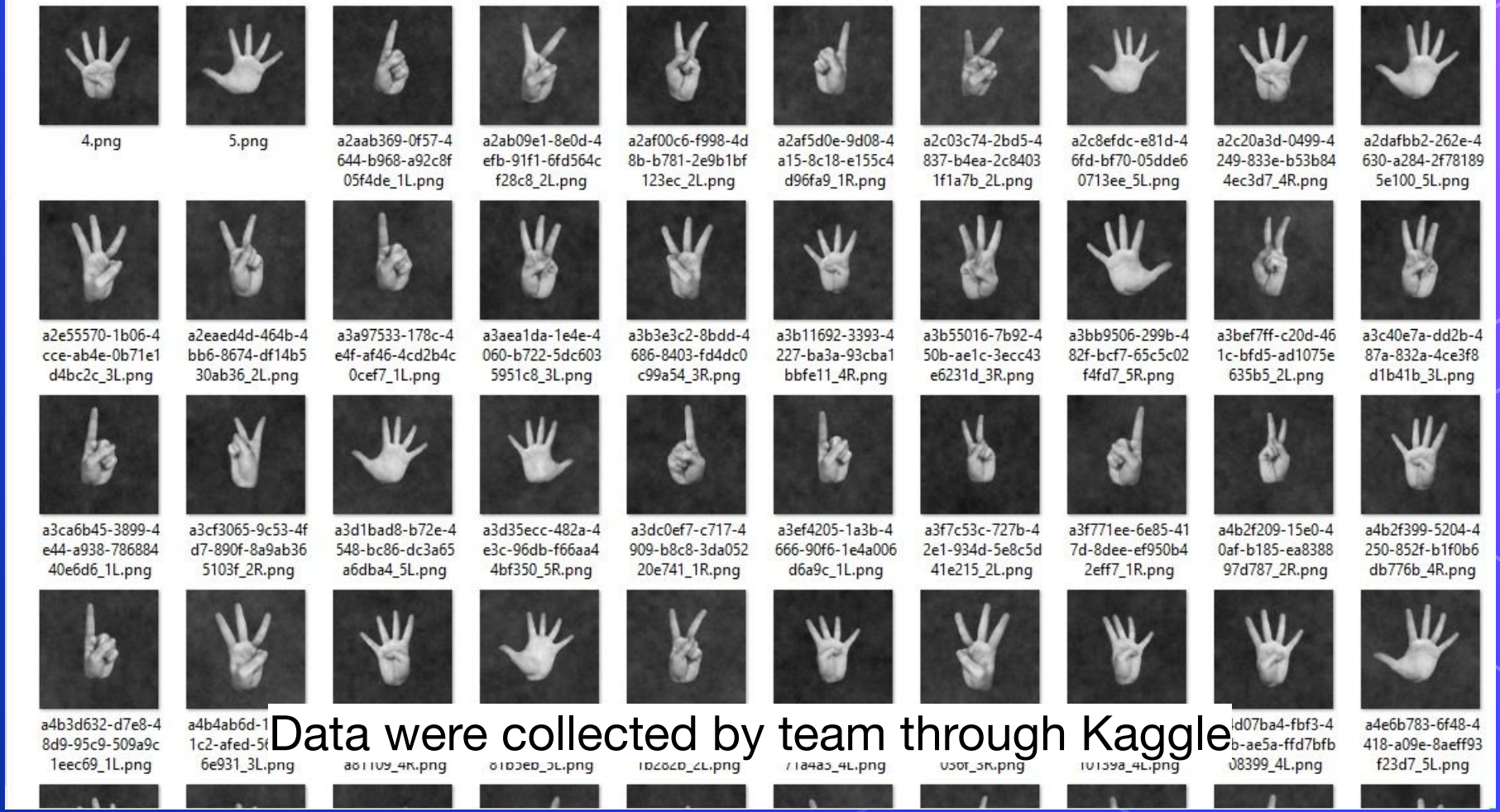
This Classification has 7 steps

1. Data gathering

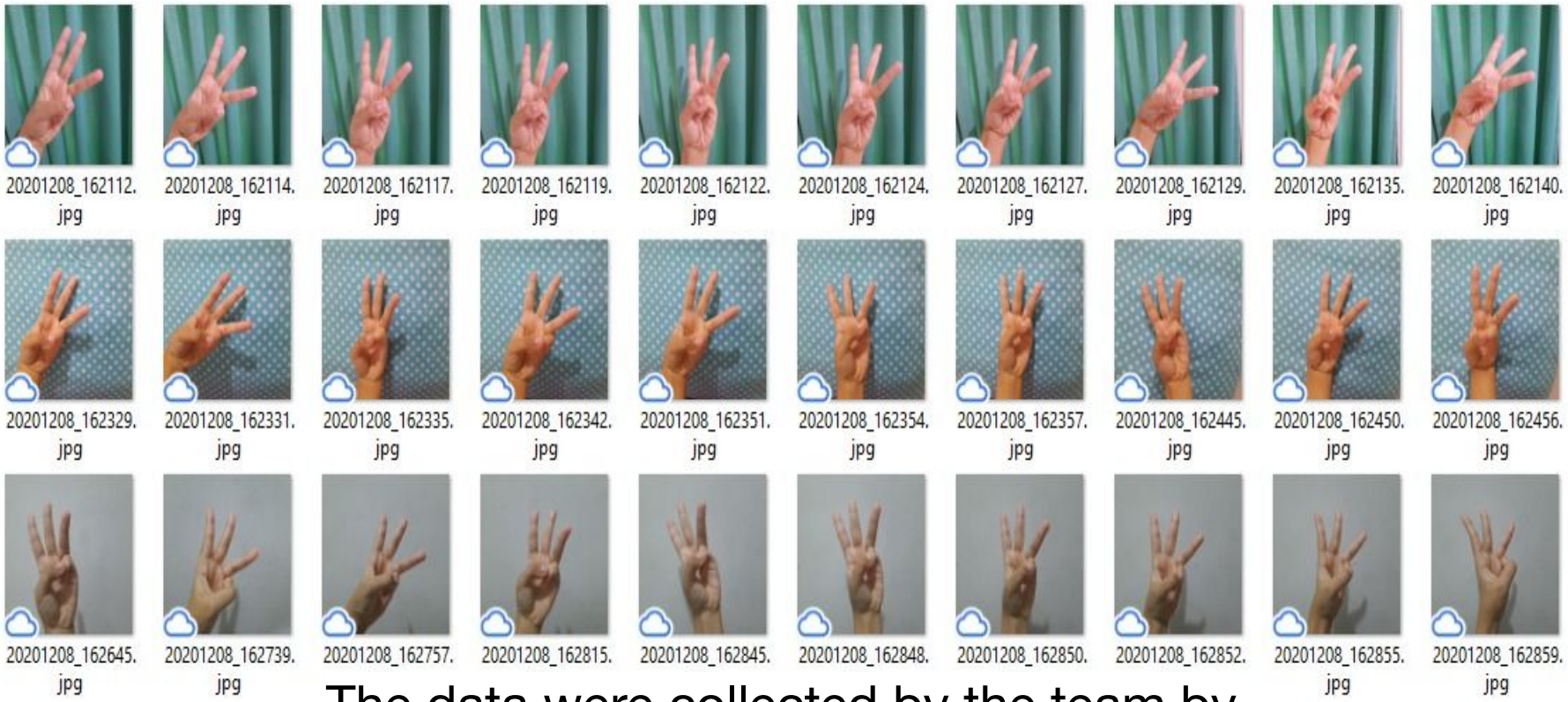




Data is collected by scraping through google using **BeautifulSoap4**

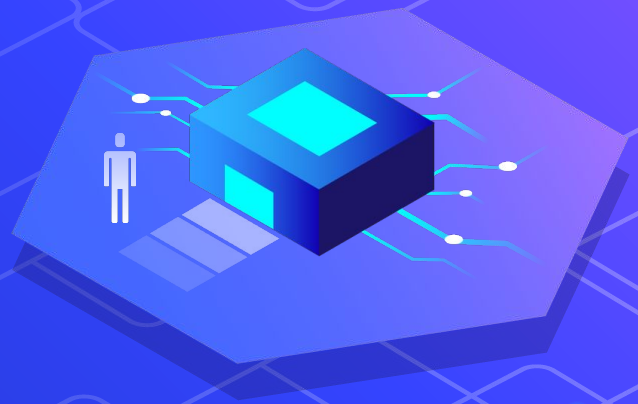


Data were collected by team through Kaggle



The data were collected by the team by capturing their own fingers

2. Data preparation



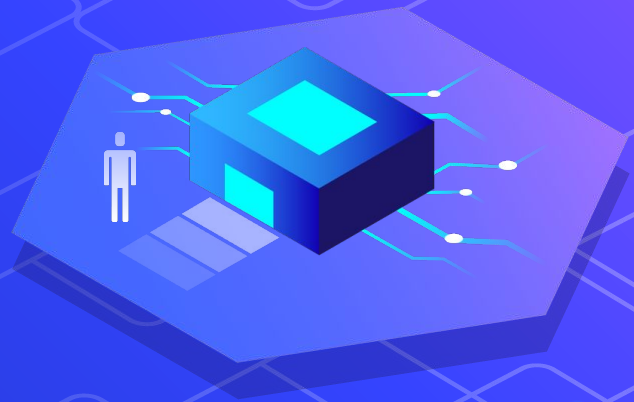
The team cleans data that is not suitable to use.

The team devides preprocessing data into the classes (1-5), then we devide it into train and test sets.

Dataset

	Class 1	Class 2	Class 3	Class 4	Class 5	Total
Train Set	240	240	240	240	240	1200
Test Set	60	60	60	60	60	300

3. Data preprocessing



Get Population mean and STD Of each channel in Image

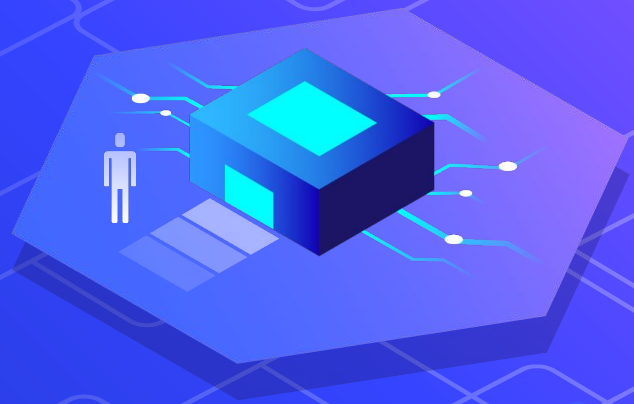
```
def online_mean_and_sd(loader):
    """Compute the mean and sd in an online fashion

        Var[x] = E[X^2] - E^2[X]
    """
    cnt = 0
    fst_moment = pt.empty(3)
    snd_moment = pt.empty(3)

    for images, _ in loader:

        b, c, h, w = images.shape
        nb_pixels = b * h * w
        sum_ = pt.sum(images, dim=[0, 2, 3])
        sum_of_square = pt.sum(images ** 2, dim=[0, 2, 3])
        fst_moment = (cnt * fst_moment + sum_) / (cnt + nb_pixels)
        snd_moment = (cnt * snd_moment + sum_of_square) / (cnt + nb_pixels)

        cnt += nb_pixels

    return fst_moment, pt.sqrt(snd_moment - fst_moment ** 2)
```

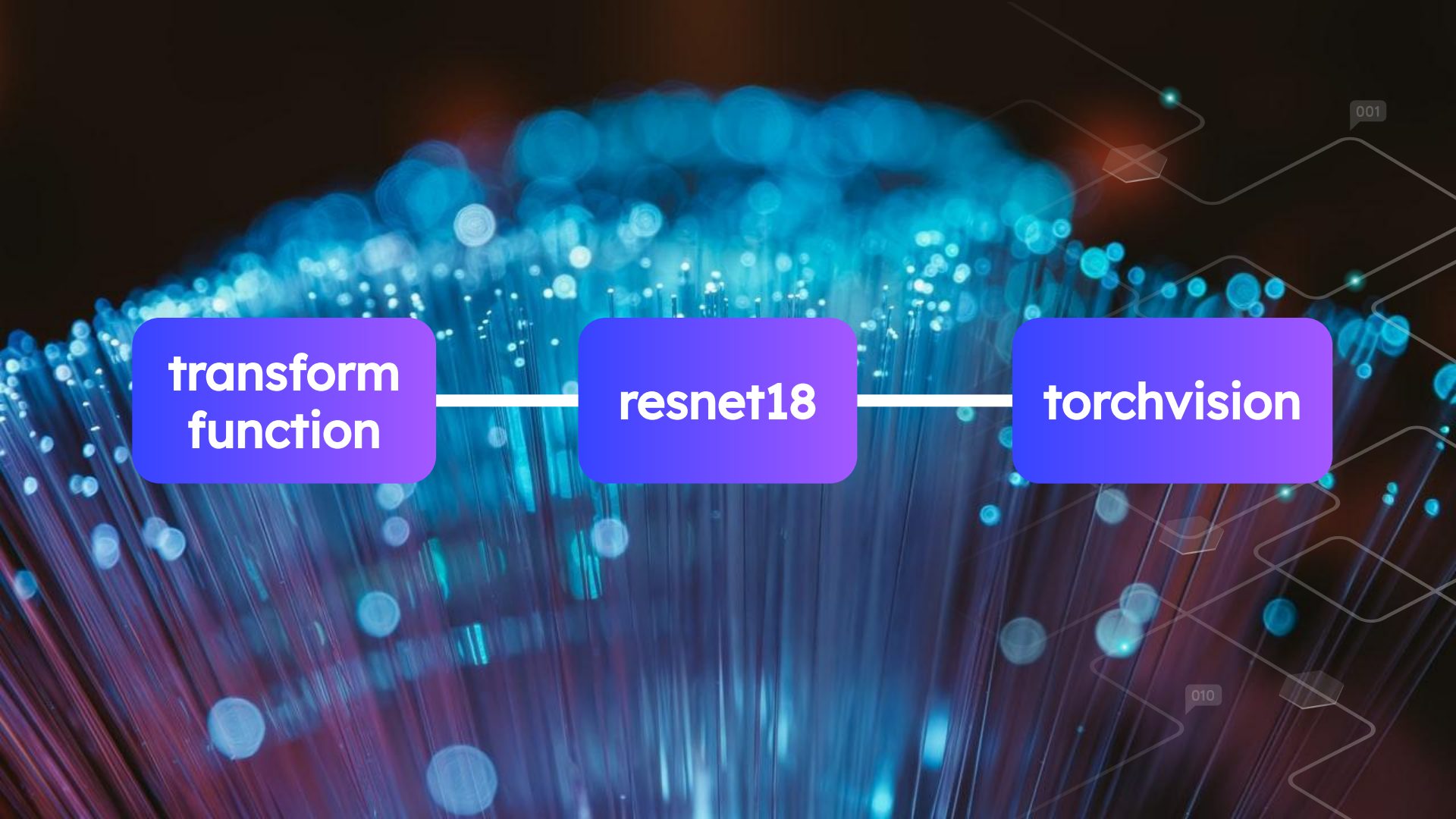
Transform the data

```
transforms.Resize((128,128)),
transforms.ToTensor(),
transforms.Normalize([0.3738, 0.3607, 0.3460],[0.2392, 0.2279, 0.2257])
```

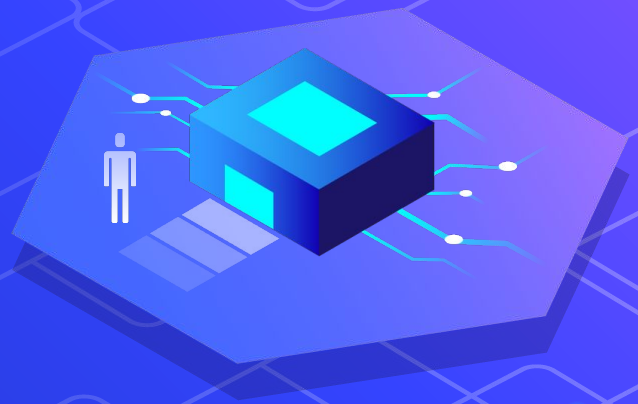
(Normalize based on mean and std
from previous step)

4. Modelling




transform function
resnet18
torchvision

5. Training

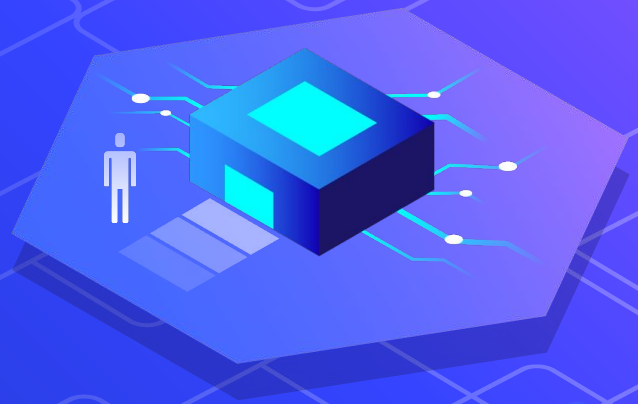


Dataset will be composed and resized into sizes (128, 128). Data is trained with RESNET 18-layer Architecture and Adam Optimizer :

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

tures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

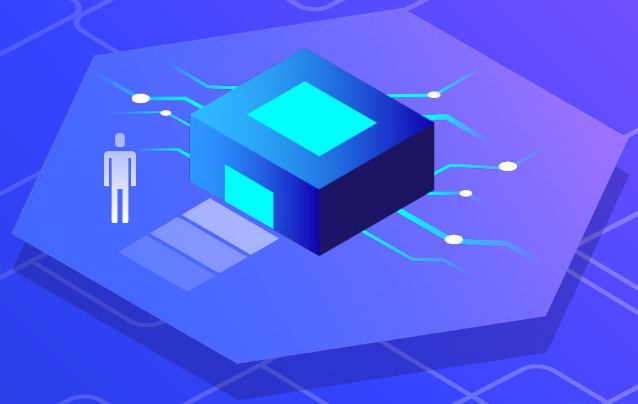
6. Testing



Model that has been saved given the test set image data as testing. At this stage,

the model can predict how many fingers appear from the given image

7. Deployment

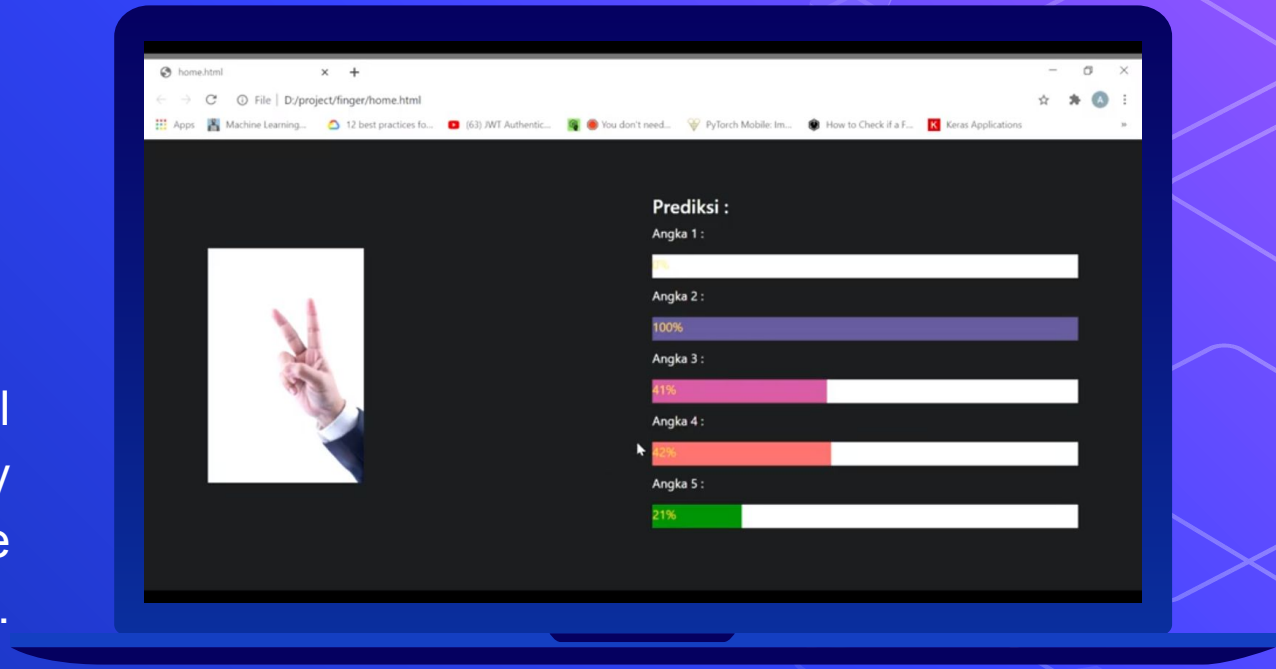


Flask

For web

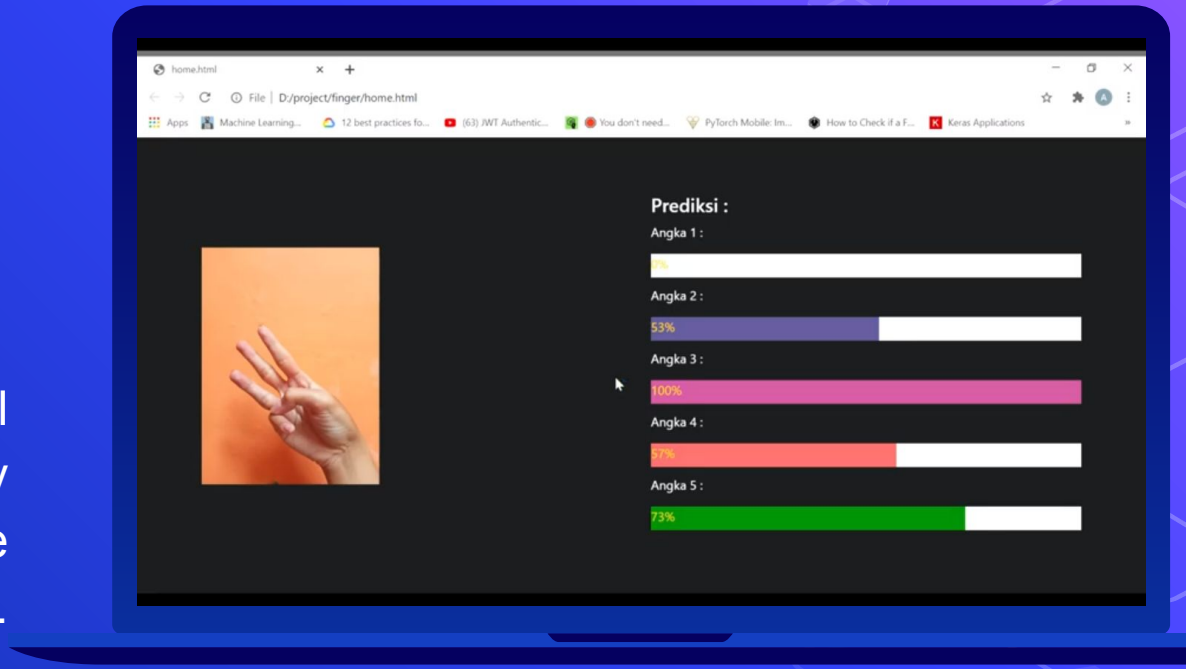
Prediction on Web

Using Flask, the model predict how many fingers appear from the given image.



Prediction on Web

Using Flask, the model predict how many fingers appear from the given image.

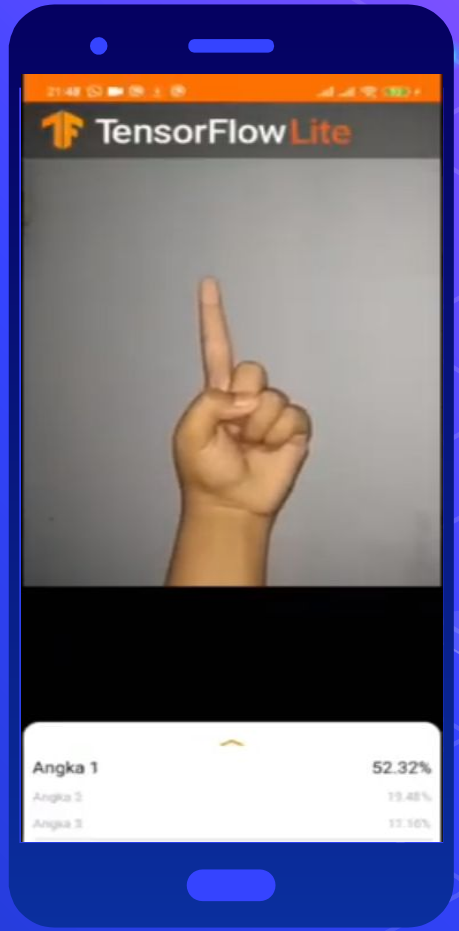


TensorFlow Lite

For android

Prediction on Android

Using TFLite, the model predict how many fingers appear from the given image.



Prediction on Android

Using TFLite, the model predict how many fingers appear from the given image.

Prediction on Android

Using TFLite, the model predict how many fingers appear from the given image.

Repository

Youtube	Classification of Numbers on Fingers
Github	Finger Classification Resnet18
Kaggle	Dataset Fingers
TFLite	Finger Application

Tim Hore

Axel Christiant	17523232
Salsabila Zahirah P.	18523066
Risca Naquitasia	18523136

Thanks!