

Classification of Numbers on Fingers

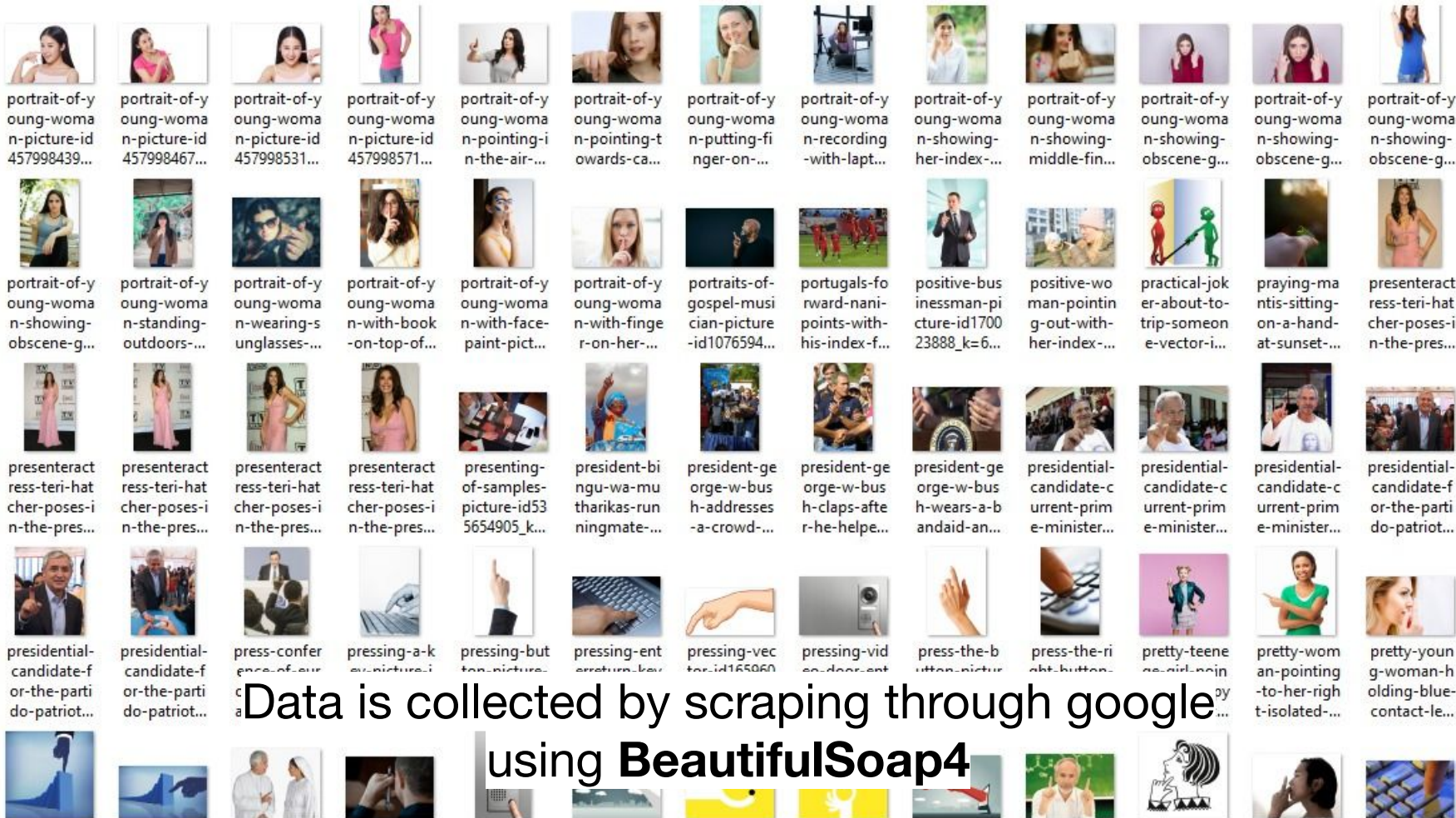


**This
Classification has
7 steps**



1. Data gathering





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



4.png



5.png



a2aab369-0f57-4644-b968-a92c8f05f4de_1L.png



a2ab09e1-8e0d-4efb-91f1-6fd564cf28c8_2L.png



a2af00c6-f998-4d8b-b781-2e9b1bf123ec_2L.png



a2af5d0e-9d08-4a15-8c18-e155c4d96fa9_1R.png



a2c03c74-2bd5-4837-b4ea-2c84031f1a7b_2L.png



a2c8efdc-e81d-46fd-bf70-05dde60713ee_5L.png



a2c20a3d-0499-4249-833e-b53b844ec3d7_4R.png



a2dafbb2-262e-4630-a284-2f781895e100_5L.png



a2e55570-1b06-4cce-ab4e-0b71e1d4bc2c_3L.png



a2eaed4d-464b-4bb6-8674-df14b530ab36_2L.png



a3a97533-178c-4e4f-af46-4cd2b4c0cef7_1L.png



a3aea1da-1e4e-4060-b722-5dc6035951c8_3L.png



a3b3e3c2-8bdd-4686-8403-fd4dc0c99a54_3R.png



a3b11692-3393-4227-ba3a-93cba1bbfe11_4R.png



a3b55016-7b92-450b-ae1c-3ecc43e6231d_3R.png



a3bb9506-299b-482f-bcf7-65c5c02f4fd7_5R.png



a3bef7ff-c20d-461c-bfd5-ad1075e635b5_2L.png



a3c40e7a-dd2b-487a-832a-4ce3f8d1b41b_3L.png



a3ca6b45-3899-4e44-a938-78688440e6d6_1L.png



a3cf3065-9c53-4fd7-890f-8a9ab365103f_2R.png



a3d1bad8-b72e-4548-bc86-dc3a65a6dba4_5L.png



a3d35ecc-482a-4e3c-96db-f66aa44bf350_5R.png



a3dc0ef7-c717-4909-b8c8-3da05220e741_1R.png



a3ef4205-1a3b-4666-90f6-1e4a006d6a9c_1L.png



a3f7c53c-727b-42e1-934d-5e8c5d41e215_2L.png



a3f771ee-6e85-417d-8dee-ef950b42eff7_1R.png



a4b2f209-15e0-40af-b185-ea838897d787_2R.png



a4b2f399-5204-4250-852f-b1f0b6db776b_4R.png



a4b3d632-d7e8-48d9-95c9-509a9c1eec69_1L.png



a4b4ab6d-11c2-afed-5f6e931_3L.png



a4b109_4R.png



a4b0eb_2L.png



a4b0eb_2L.png



a4b0eb_2L.png



a4b0eb_2L.png



a4b0eb_2L.png



a4b0eb_2L.png



a4e6b783-6f48-4418-a09e-8aeff93f23d7_5L.png

Data were collected by team through Kaggle



20201208_162112.
jpg



20201208_162114.
jpg



20201208_162117.
jpg



20201208_162119.
jpg



20201208_162122.
jpg



20201208_162124.
jpg



20201208_162127.
jpg



20201208_162129.
jpg



20201208_162135.
jpg



20201208_162140.
jpg



20201208_162329.
jpg



20201208_162331.
jpg



20201208_162335.
jpg



20201208_162342.
jpg



20201208_162351.
jpg



20201208_162354.
jpg



20201208_162357.
jpg



20201208_162445.
jpg



20201208_162450.
jpg



20201208_162456.
jpg



20201208_162645.
jpg



20201208_162739.
jpg



20201208_162757.
jpg



20201208_162815.
jpg



20201208_162845.
jpg



20201208_162848.
jpg



20201208_162850.
jpg



20201208_162852.
jpg



20201208_162855.
jpg



20201208_162859.
jpg

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 divides preprocessing data into the classes (1-5), then we divide 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  
  
    
$$\text{Var}[x] = E[X^2] - E^2[X]$$
  
    """  
    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				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	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

ures 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



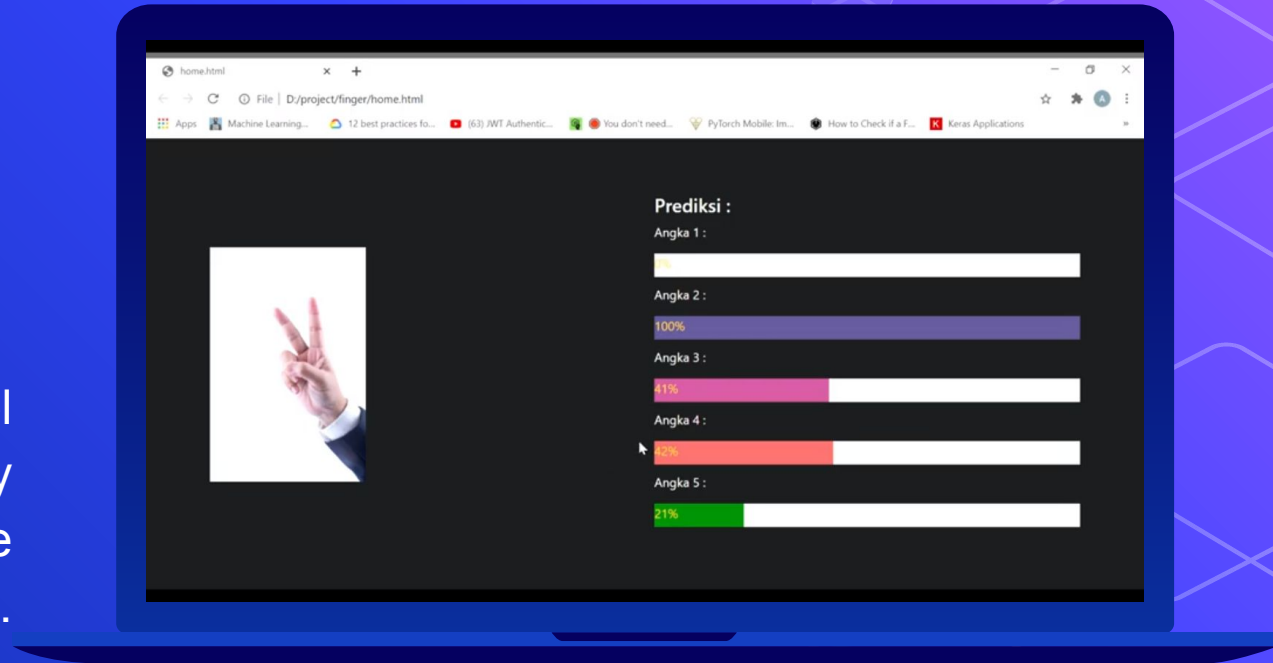
Flask

For web



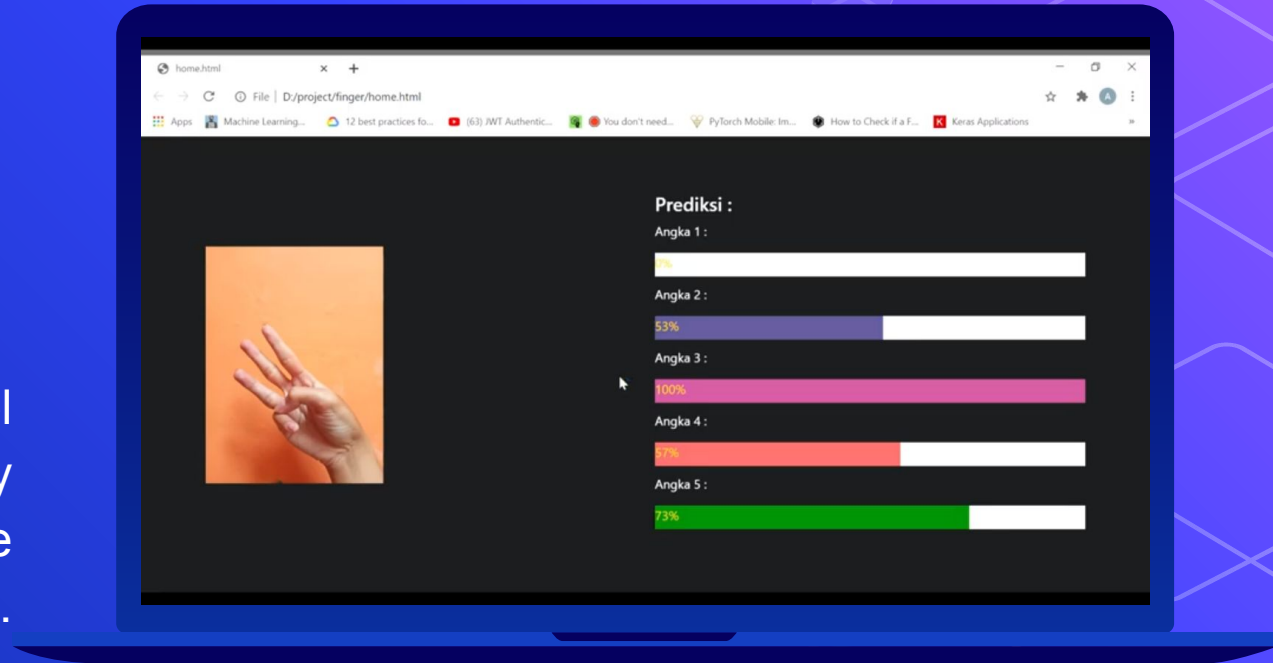
Prediction on Web

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.



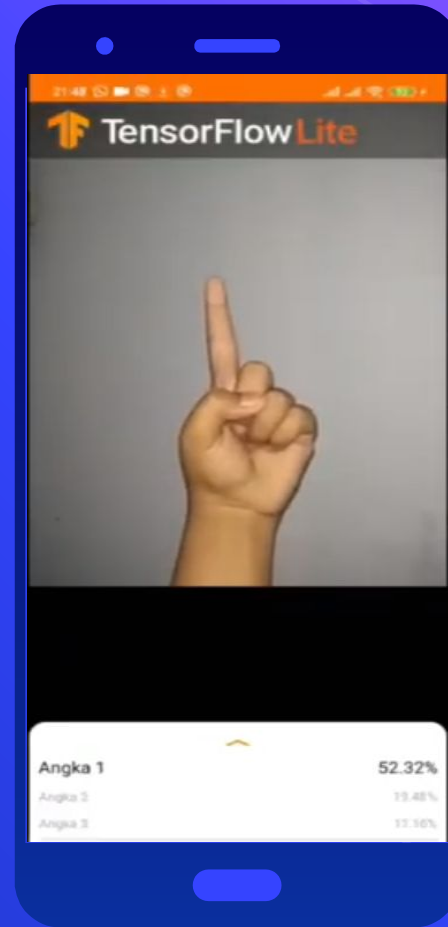
TensorFlow Lite

For android



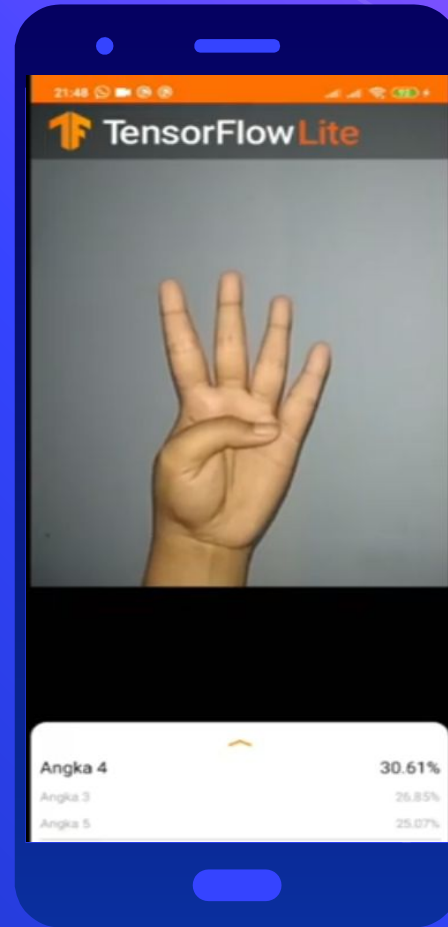
Prediction on Android

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



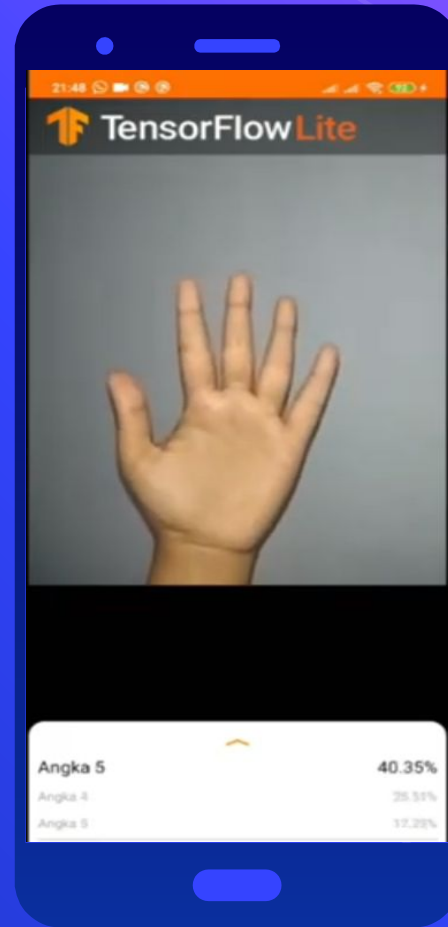
Prediction on Android

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!

