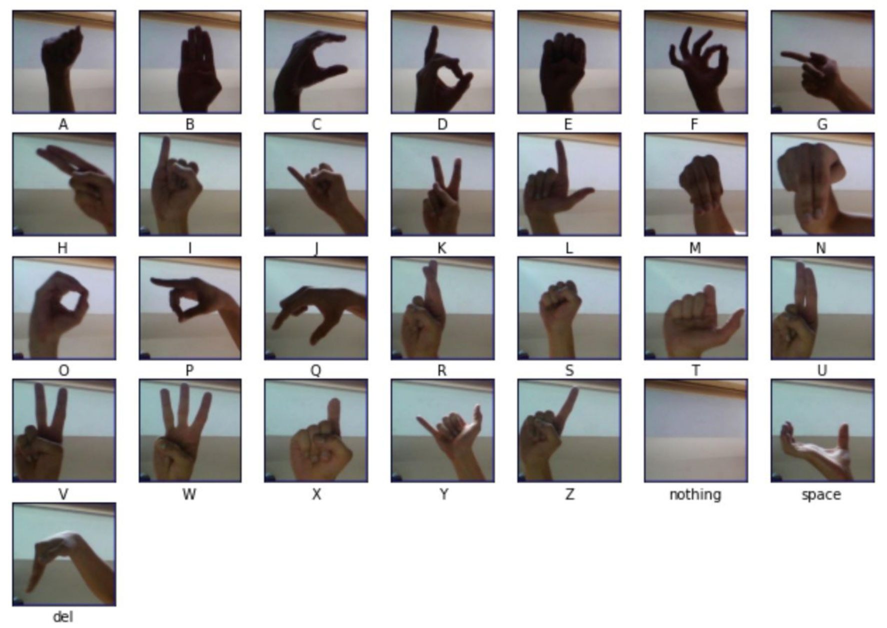
# SIGN LANGUAGE

## ASL Alphabet

# American Sign Language Dataset

- The data set is a collection of images of alphabets from the American Sign Language, separated in 29 folders which represent the various classes.



# Task objectives

## Global context

To develop computer vision system that translate sign language to spoken language in streaming video.

## Our task

As first step, towards understanding how to build a translation system, we can reduce the size of the problem by translating individual letters, instead of sentences.


SignALL is pioneering the first automated sign language translation solution, based on computer vision and natural language processing (NLP), to enable everyday communication between individuals with hearing who use spoken English and deaf or hard of hearing individuals who use ASL.

# Train data

The training data set contains **87,000** images which are **200x200 pixels**.

There are **29 classes**, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.

# Data preparation

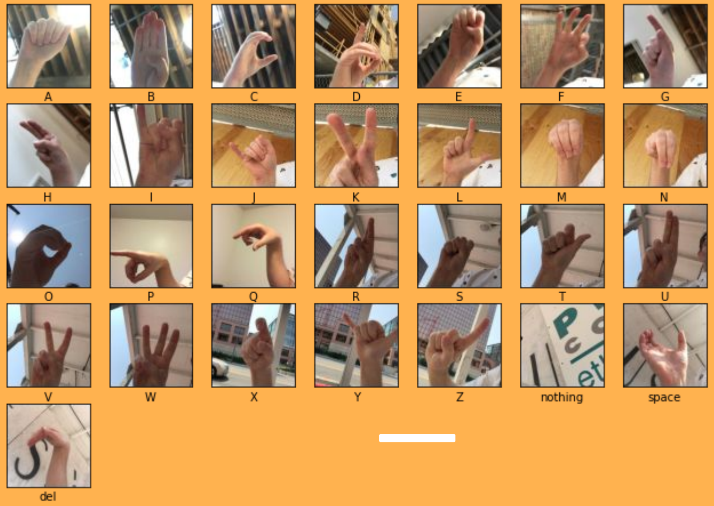
## Data frames

Creating smaller train and test data frames 2900 images for train

Training set for the model: 64x64 colour images


## Labels map

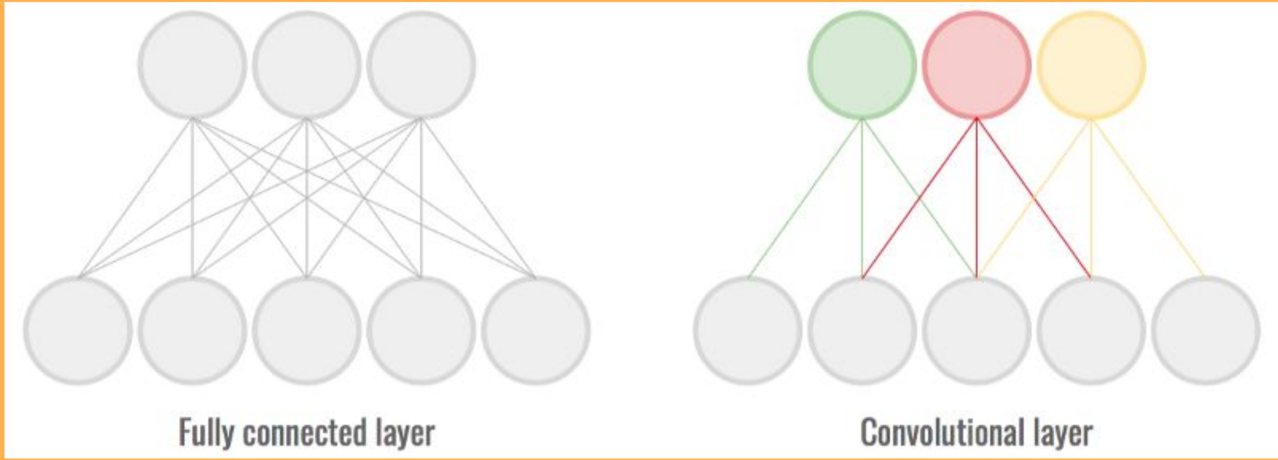
From 0 to 28 for each letter and special signs



# Neural Network for ASL Classification

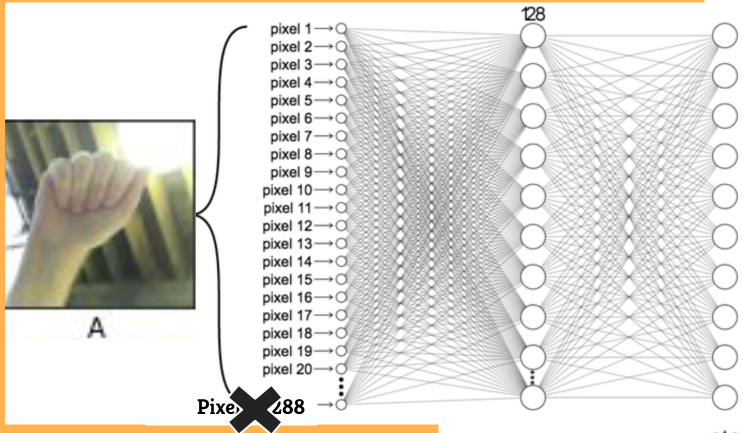
## Simple

## Convolutional



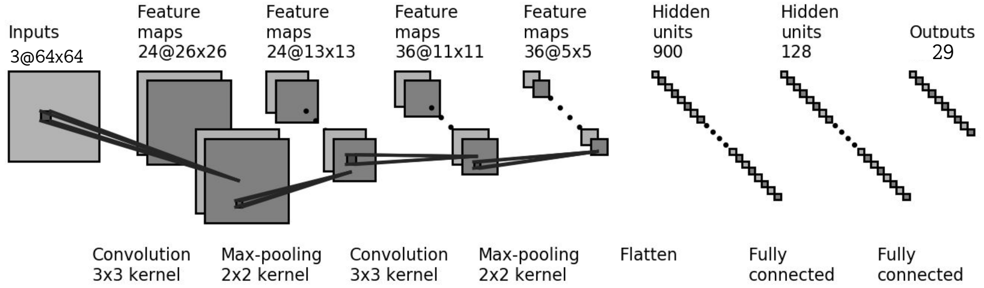
# Models Description

## (1) Fully connected

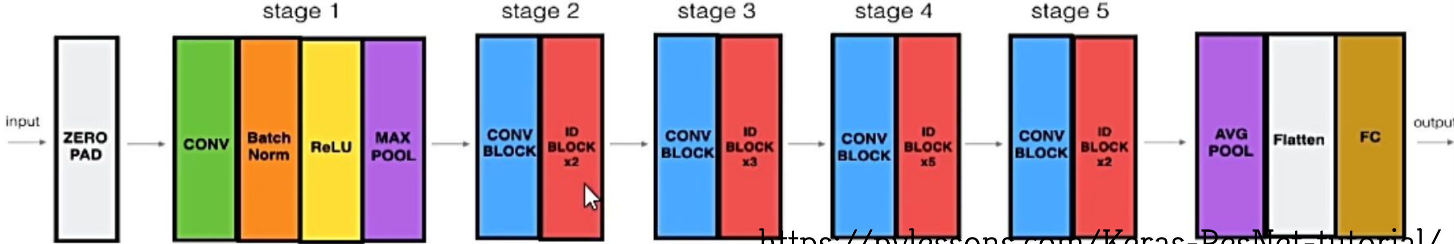



Pixel 2,352
(28x28x3)

## (2) CNN




## (3) ResNet (50 layers)




https://pylessons.com/Keras-ResNet-tutorial/

# Models Comparison

**Train**: 100 examples from each of the 29 classes => 2900 examples
**Test**: same as above, but different images
100 epochs, batch_size = 64

| | **Fully connected** | **CNN** | **ResNet** |
|---|---|---|---|
| Train accuracy | **0.01** | **1** | **1** |
| Test accuracy | **0.01** | **0.81** | **0.68** |
| | | | |
| Train loss | nevermind | 2.6236e-04 | 7.4790e-05 |
| Test loss | nevermind | 1.50 | 2.7 |
| Train duration (min) | nevermind | 18 | 300 |

# Fully connected – for sure it can do more !

| Results for fully connected network | | | |
|---|---|---|---|
| Same = data comes from the same dataset | | | |
| Other = data comes from the other dataset (the second one, with 30 examples of each sign) | | | |
| | | | |
| **image size →** | **28x28** | **28x28** | **28x28** |
| **train data →** | **train on 2970 each** | **train on 2970 each** | **train on 100 each** |
| **test data →** | **test on other 30 each** | **test on same 30 each** | **test on same 100 each** |
| **train acc** | 0.87 | 0.87 | 0.03 |
| **test acc** | 0.05 | 0.87 | 0.03 |
| **training duration** | 9.7 min | 9 min | 1 min |

# Models Comparison

**Train**: 100 examples from each of the 29 classes => 2900 examples
**Test**: same as above, but different
100 epochs, batch_size = 64

| 28x28 |
| --- |
| train on 2970 each |
| test on same 30 each |

| | Fully connected | CNN | ResNet |
| --- | --- | --- | --- |
| Train accuracy | **0.87** | **1** | **1** |
| Test accuracy | **0.87** | **0.81** | **0.68** |
| Train loss | 0.346 | 2.6236e-04 | 7.4790e-05 |
| Test loss | 0.46 | 1.50 | 2.7 |
| Train duration (min) | 9 | 18 | 300 |

# Models Comparison

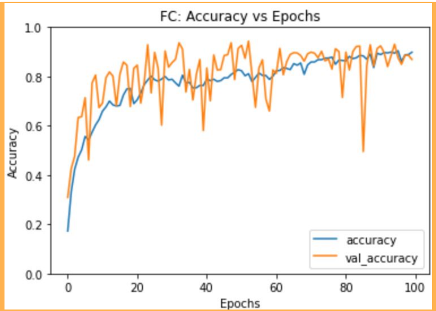
Room for improvement with more epochs

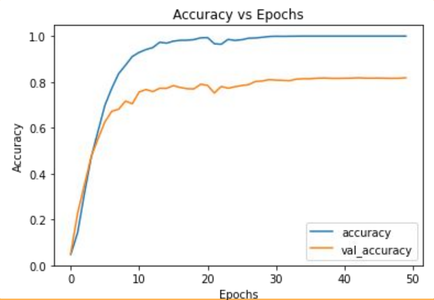
High variability in test results (between epochs)

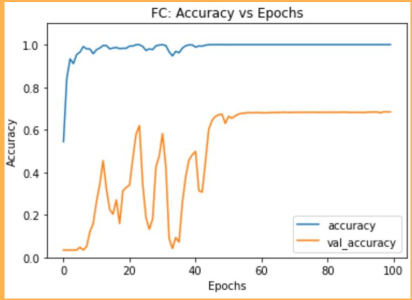
We need more variability in our train data (augmentation), because the features we're learning don't seem very general

| Fully connected | CNN | ResNet |
|---|---|---|

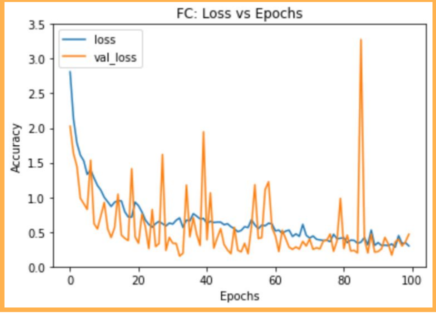
Accuracy (train and test) vs epochs

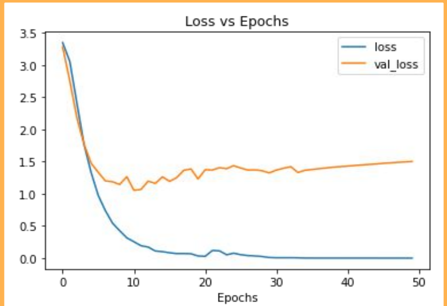


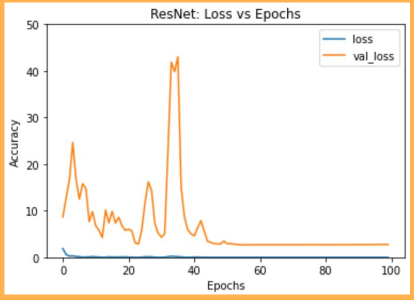




Loss (train and test) vs epochs



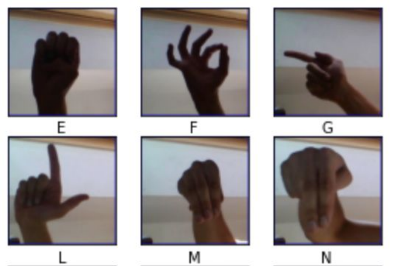




Remember: train sz = 86k, test sz = 870 | train sz = 2.9k, test sz = 2.9k | train sz = 2.9k, test sz = 2.9k

# Conclusion

- A simple fully connected network has the best test accuracy & training time for our data
    - If we increase epochs <-- 500, we get:
        - Train accuracy: 0.98
        - Test accuracy: 0.98
        - Training time: 40 min
- Our CNN and ResNet do not generalize well:
    - Test accuracy for both plateaus rather quickly
    - So maybe we don't have enough variability in train set => enrich our data with transformations (rotations & other tricks)
- It doesn't make sense to train a full ResNet for 5 hrs for these results
    - Try "transfer learning" and fine-tuning to check if decent train time & decent accuracy in test (generalization)



# YOLO (You Only Look Once)

## YOLO versions by Joseph Redmon

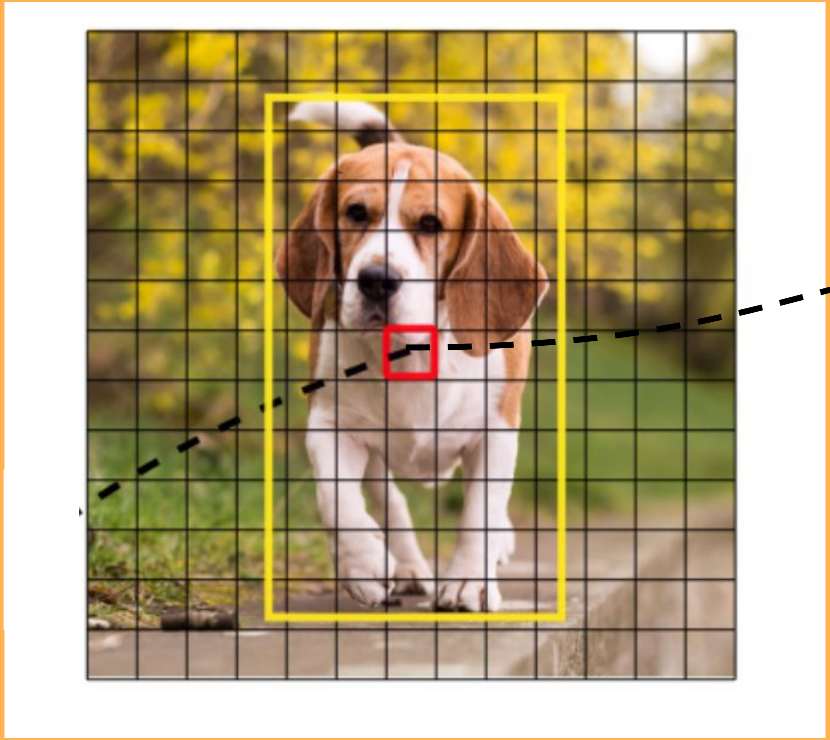
1. Version 1
   ‘*You Only Look Once: Unified, Real-Time Object Detection*’ (2016)
2. Version 2
   ‘*YOLO9000: Better, Faster, Stronger*’ (2017)
3. Version 3
   ‘*YOLOv3: An Incremental Improvement*’ (2018)

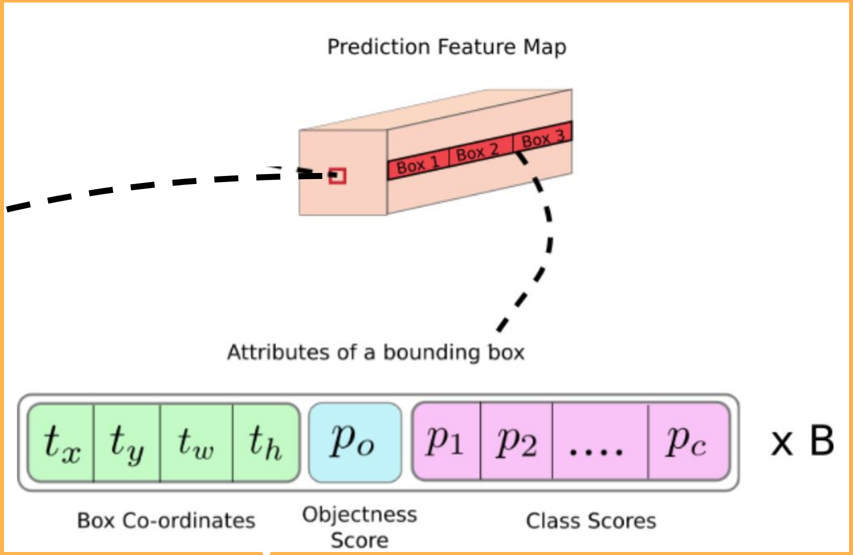
→ all above based on **Darknet**, open source neural network framework written in C and CUDA.
→ PyTorch version of YOLOv3 by Glenn Jocher

## YOLO versions by others

4. Version 4
   ‘*YOLOv4: Optimal Speed and Accuracy of Object Detection*’ by Alexey Bochkovskiy et al. (2020)
5. Version 5
   by the Glenn Jocher (2020), uses PyTorch

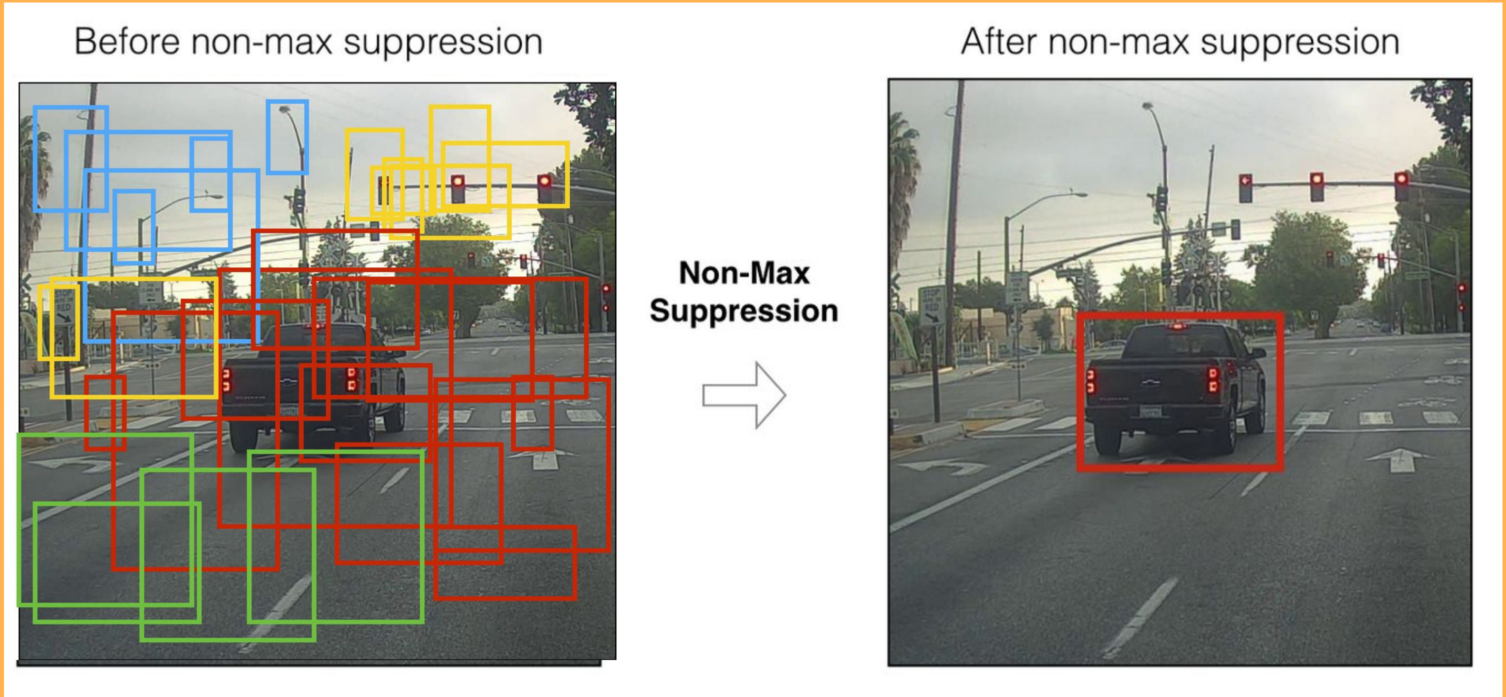
# YOLO v3 – how it works



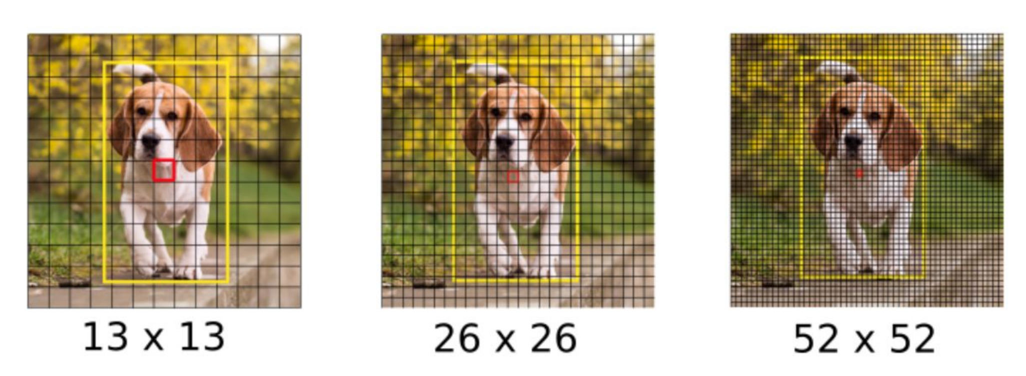



Bounding boxes and feature maps

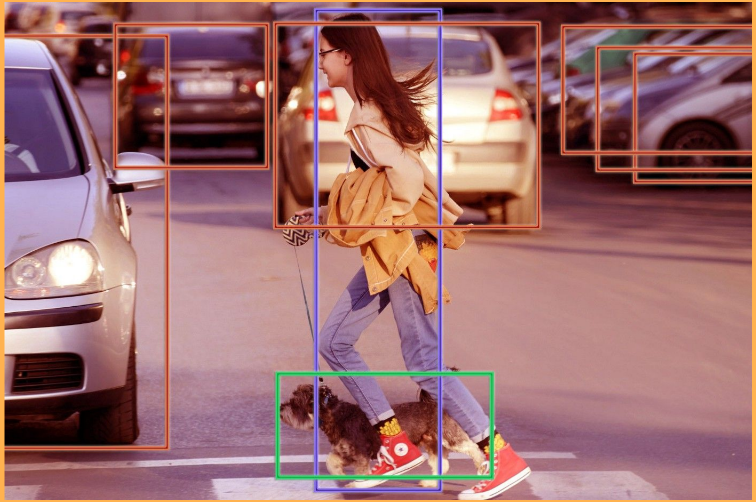
# YOLO v3 - non-max suppression



# YOLO v3 – how it works




Multiple sizes



What we get

# YOLO v3 – implementations

## Original

→ Darknet, open source neural network framework written in C and CUDA.

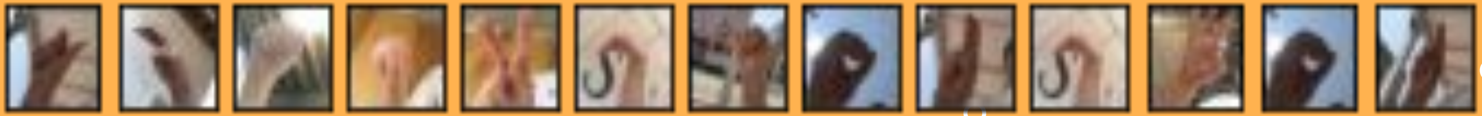
→ error building it

## Next to try:

→ TensorFlow-2.x-YOLOv3 by pythonlessons:

- GitHub here
- Code explained in this article

→ Darkflow



T H A N K del Y O U del F O R

A T T E N T I O N