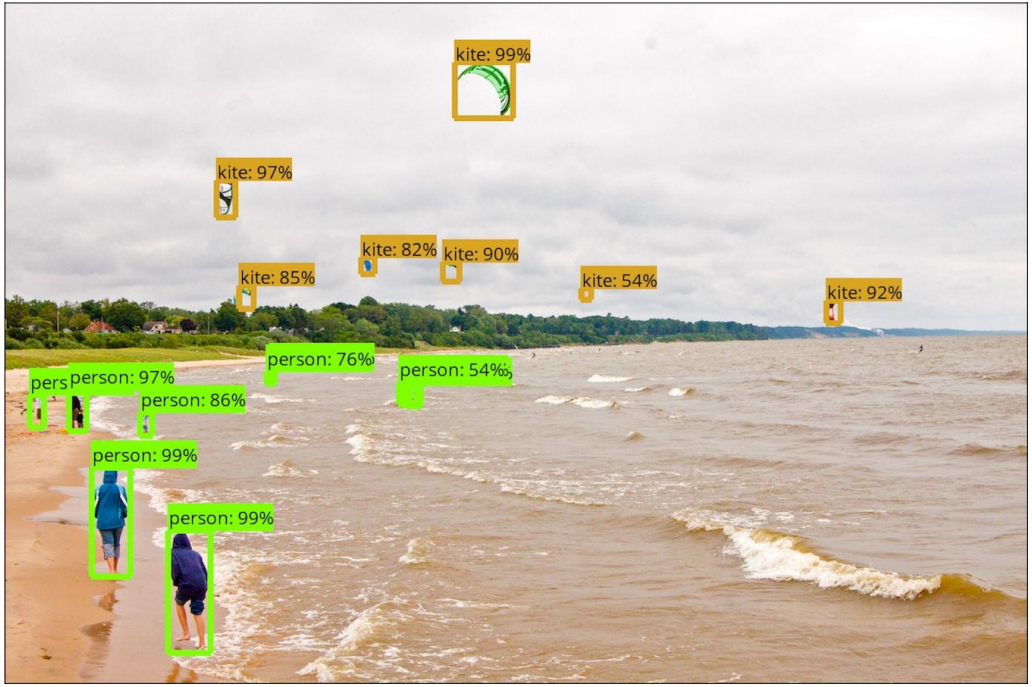
Transfer learning, Active learning

using tensorflow object detection api

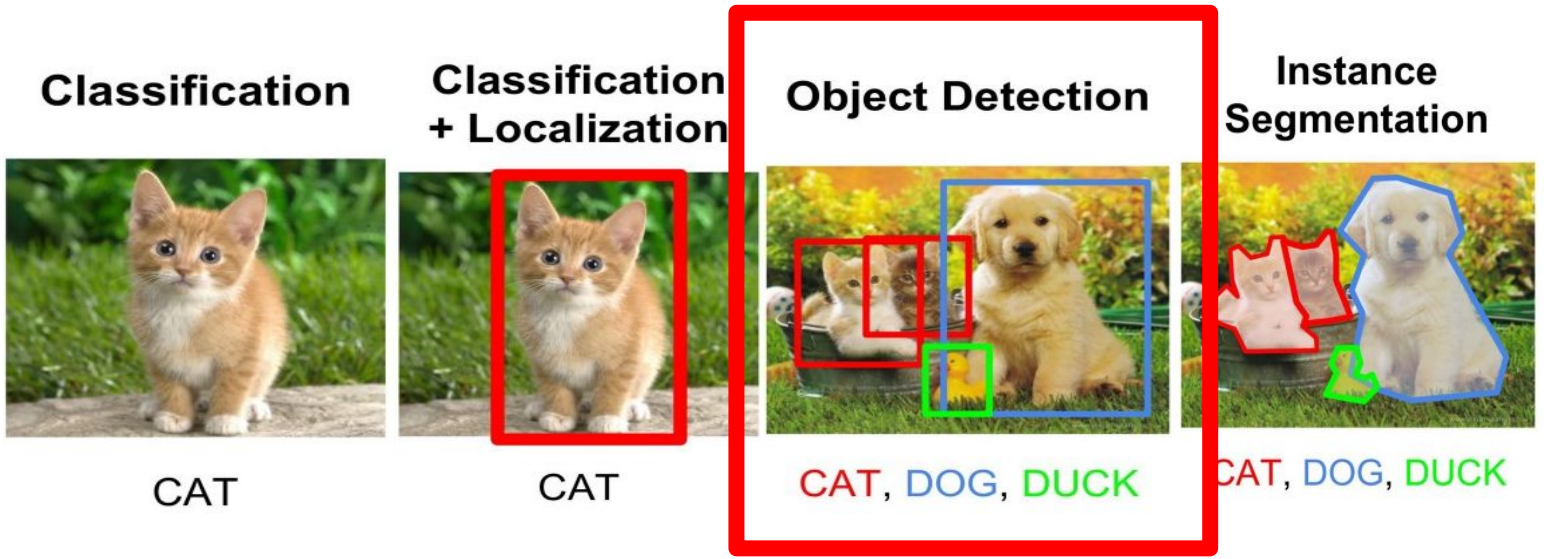
Google released the tensorflow-based object detection API in June 2017.
By offering models with various speeds and accuracy, the general public can easily detect objects.

Tensorflow Object detection api



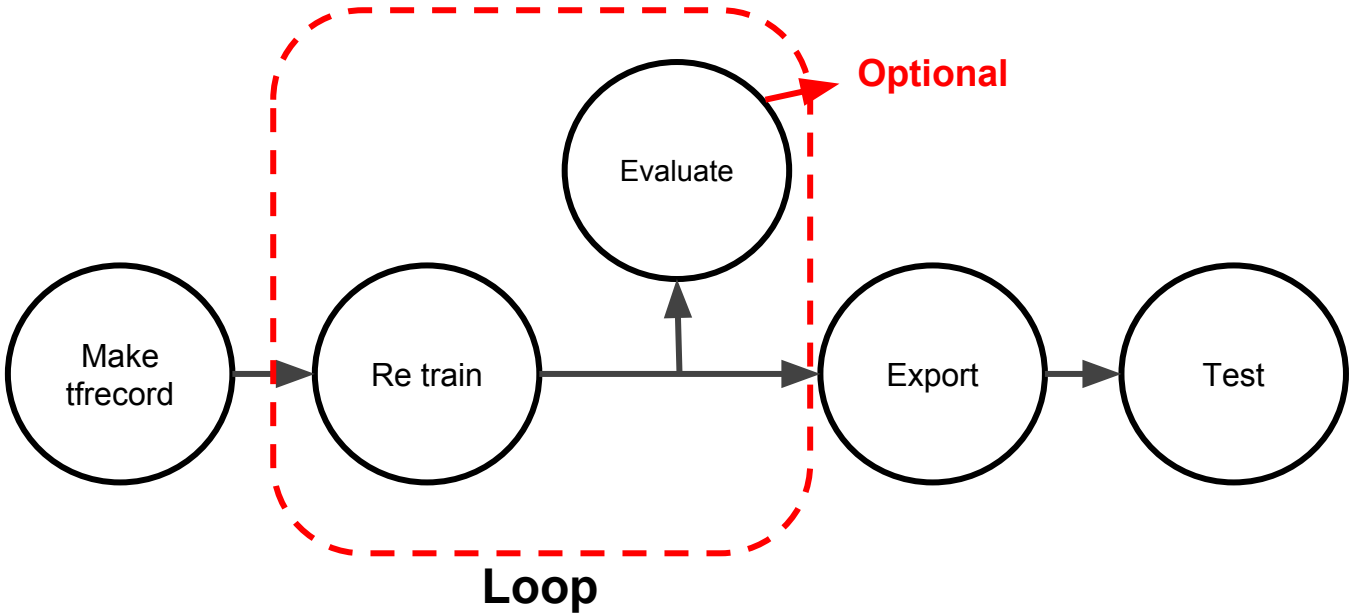
There are various types of image detection.
image classification is simply a determination of what the image is.
You can also localize from the image classification to the exact location of the object

Object detection

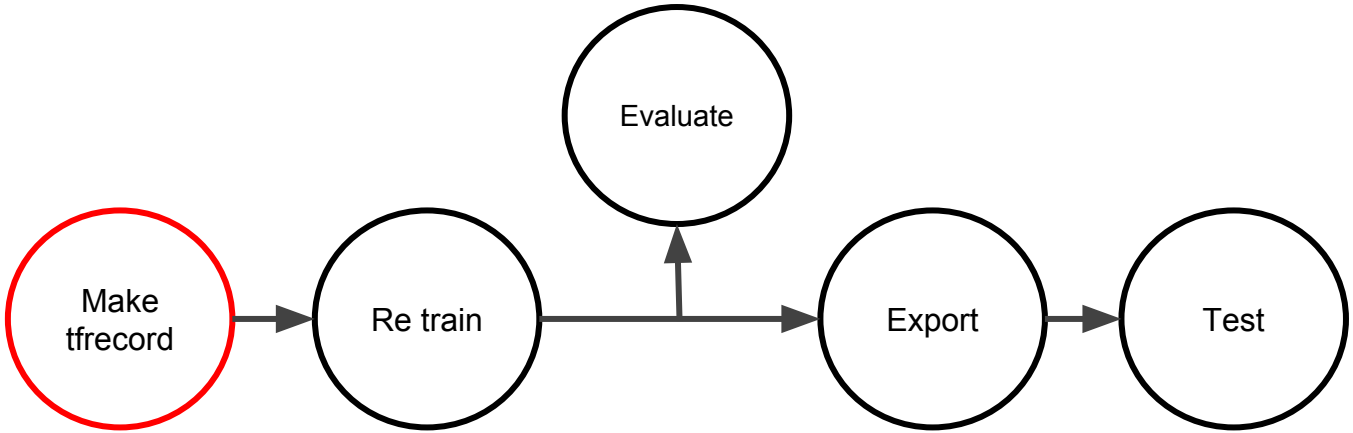


The most basic flow of the tensorflow object detection api.
All functions are provided to process the data to api, train this data, export the model to a usable form, and test this model.
You can also evaluate ongoing or completed models.

Tensorflow Object detection api



Tensorflow Object detection api



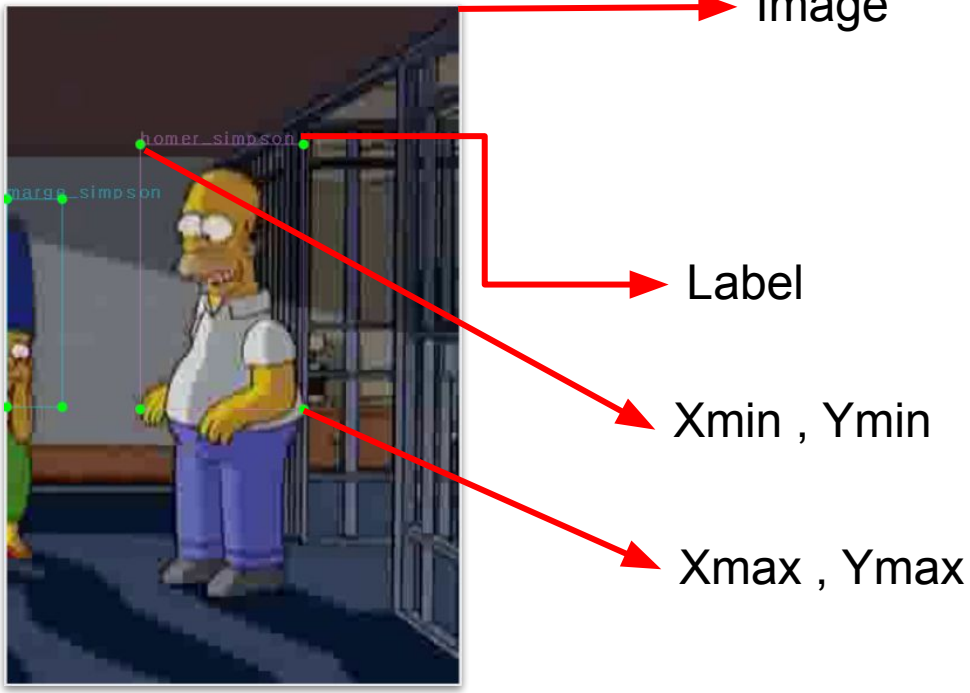
The data type for using object detection api is tfrecord.
The source for creating this tfrecord file is provided, below is how to use it.
Set the csv file containing the image information and the output directory.

Make tfrecord

```
python generate_tfRecord.py \
--csv_input=data/train.csv \
--output_path=data/train.record
```

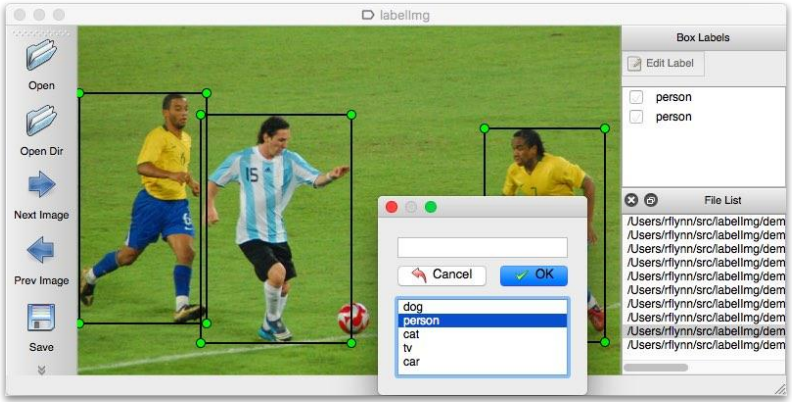
The data needed to create the tfrecord file is as follows:
First, you need the original image file.
Next, we need a label for each object in the image.

Make tfrecord



It is difficult to manually determine the location point of an object.
There are various programs that do this automatically.
I used the most widely known labelimg.

Make tfrecord



LabelImg

https://github.com/tzutalin/labelImg

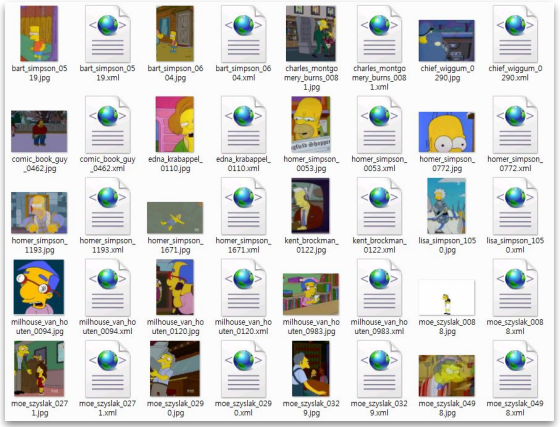
When you use labelimg, an xml file is created that contains the label of each object and the values of xmin, ymin, xmax, and ymax.

Make tfrecord


```
<annotation>
	<folder>inference_6</folder>
	<filename>homer_simpson_1070.jpg</filename>
	<path>C:₩Users₩5taku₩Desktop₩inference_6₩homer_simpson_1070.jpg
</path>
	<source>
		<database>Unknown</database>
	</source>
	<size>
		<width>288</width>
		<height>432</height>
		<depth>3</depth>
	</size>
	<segmented>0</segmented>
	<object>
		<name>homer_simpson</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>85</xmin>
			<ymin>88</ymin>
			<xmax>189</xmax>
			<ymax>257</ymax>
		</bndbox>
	</object>
	<object>
		<name>marge_simpson</name>
		<pose>Unspecified</pose>
		<truncated>1</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>1</xmin>
			<ymin>122</ymin>
			<xmax>35</xmax>
			<ymax>255</ymax>
		</bndbox>
	</object>
</annotation>
```

If we have an image to train, xml, and a labelmap that stores the id for each class, we can generate a tfrecord file.

Make tfrecord



```
item {
  id: 1
  name: 'homer_simpson'
}
item {
  id: 2
  name: 'ned_flanders'
}
item {
  id: 3
  name: 'moe_szyslak'
}
item {
  id: 4
  name: 'lisa_simpson'
}
item {
  id: 5
  name: 'bart_simpson'
}
item {
  id: 6
  name: 'marge_simpson'
}
item {
  id: 7
  name: 'krusty_the_clown'
}
item {
  id: 8
  name: 'principal_skinner'
}
item {
  id: 9
  name: 'charles_montgomery_burns'
}
item {
  id: 10
  name: 'milhouse_van_houten'
}
```

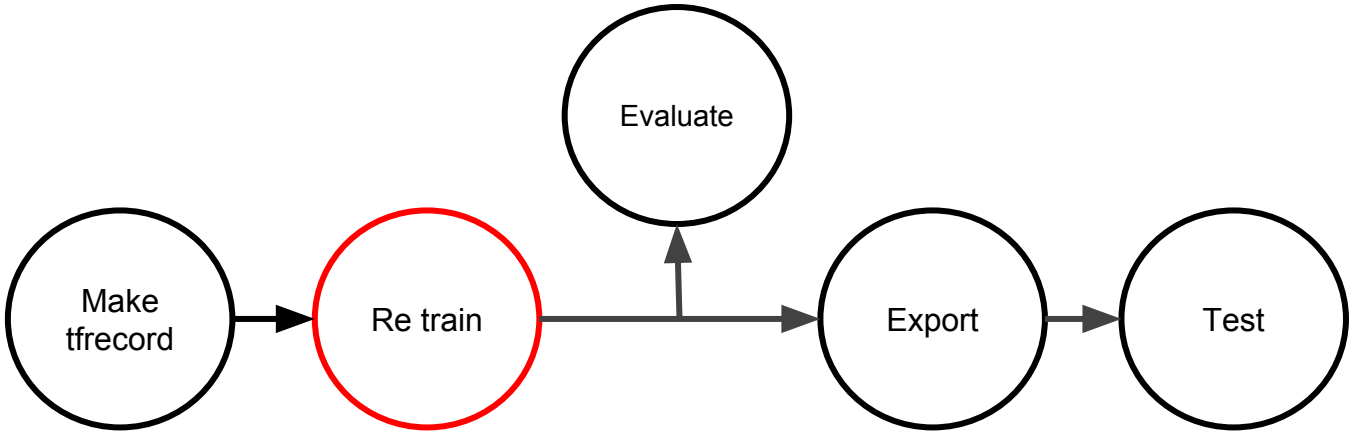
The default code is train and test.
Since this is a troublesome task, I have created a program that will do this automatically.
When executed, each class is distributed according to the ratio of train, validate, and a log is generated.

Make tfrecord tfgenerator (custom)

```
ksulki.tensorflow@instance-1:~/active_learning/Active_learning_simpson_dataset$ python tfgenerator.py -i ./train_dataset -sr 9
[INFO|tfgenerator.py:168] 2018-06-04 21:38:46,335 > TF Record Generator Start
[INFO|tfgenerator.py:21] 2018-06-04 21:39:30,188 >                TF Record Summary
[INFO|tfgenerator.py:22] 2018-06-04 21:39:30,188 >     ID           NAME          Train    Validate
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,188 >     1       homer_simpson       1863      208
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,188 >     2        ned_flanders       1253      140
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,188 >     3        moe_szyslak        1148      128
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,188 >     4        lisa_simpson       1144      128
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,188 >     5        bart_simpson       1146      128
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     6       marge_simpson       1125      126
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     7      krusty_the_clown     1032      115
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     8     principal_skinner     1012      113
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     9    charles_montgomery_burns   997       111
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     10   milhouse_van_houten    871        97
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     11       chief_wiggum       782        87
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     12   abraham_grampa_simpson   790        88
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     13       sideshow_bob       715        80
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     14   apu_nahasapeemapetilon   531        59
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     15      kent_brockman       374        42
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     16      comic_book_guy      363        41
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     17      edna_krabappel      351        39
[INFO|tfgenerator.py:24] 2018-06-04 21:39:30,189 >     18       nelson_muntz       269        30
[INFO|tfgenerator.py:201] 2018-06-04 21:44:10,289 > TF Record Generator End [ Total Generator time : 0 Hour 5 Minute 23 Second ]
```

```
968K    train.csv
411M    train.record
108K    validate.csv
46M     validate.record
```

Tensorflow Object detection api



object detection api python base code related to train.
Set up the model you want to train, and set the training output directory.
There are many other options.

Re train Transfer learning

```
python object_detection/train.py \
--logtostderr \
--pipeline_config_path=pipeline.config \
--train_dir=train
```

It provides various pre-trained models for object detection of tensorflow.
Each model has different speed and accuracy.
The first part of the model name is the algorithm, and the second part is the data set.

Re train Object detection API model zoo

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

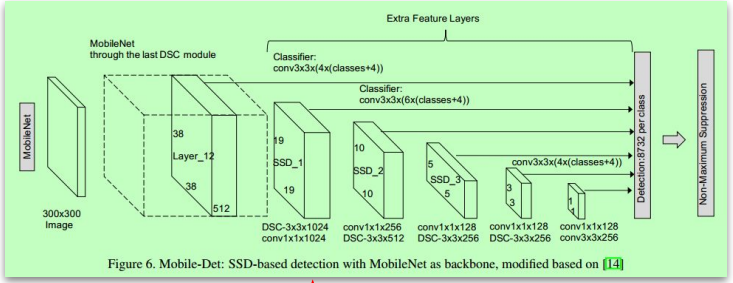
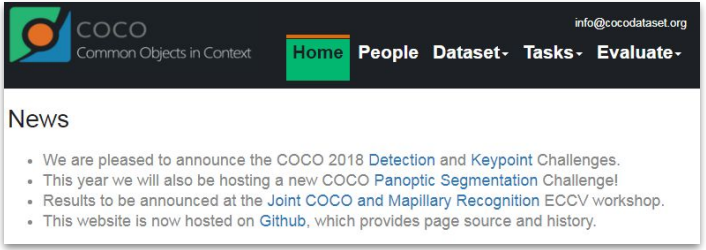


Figure 6. Mobile-Det: SSD-based detection with MobileNet as backbone, modified based on [14]

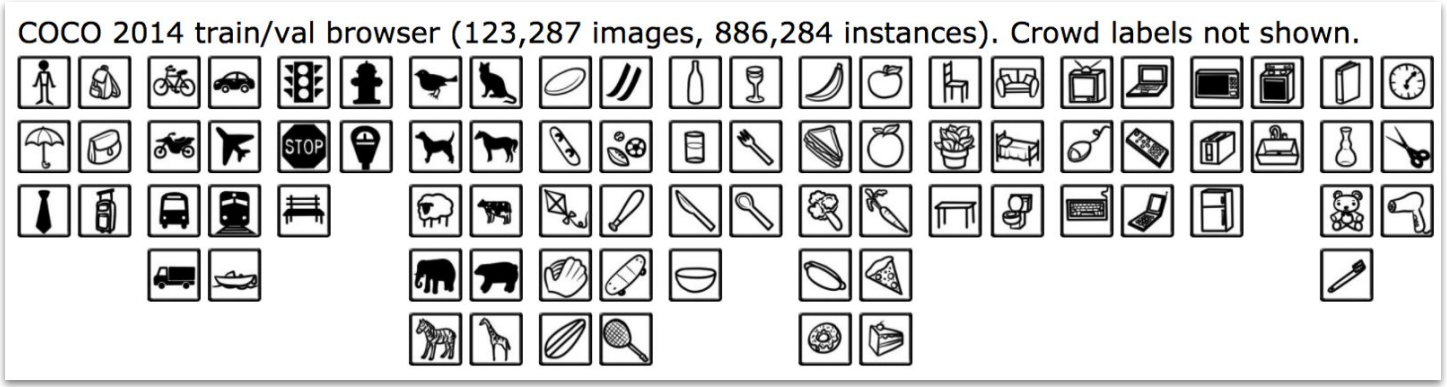
ssd_mobilenet_v1_coco



Dataset : COCO dataset, Kitti dataset, Open Images dataset.

If the class you want to detect is an instance already in coco, kitty, or open image dataset, you do not need to train.
Just use the built-in model and select it.

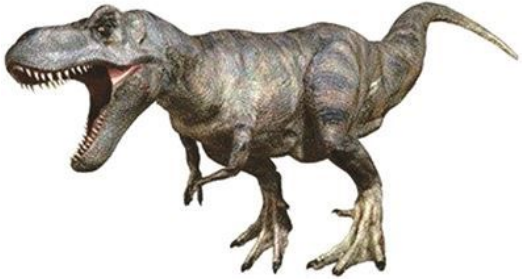
Re train What you want to detect

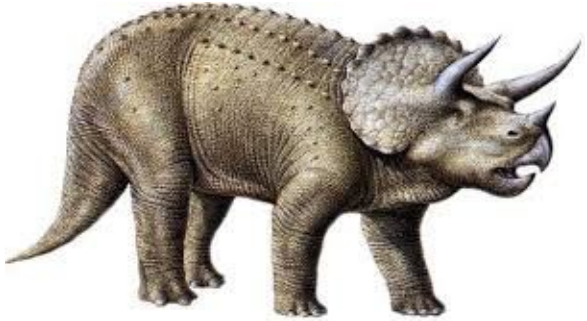


COCO dataset Instance

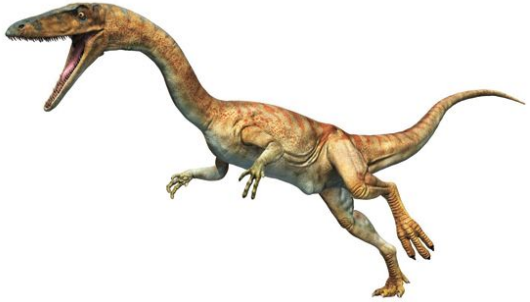
However, if you want to classify dinosaurs such as Tyrannosaurus and Triceratops as shown in the picture, we need to train on the above classes.
Here we have two choices.
Create models with completely new layers, or leverage existing models.

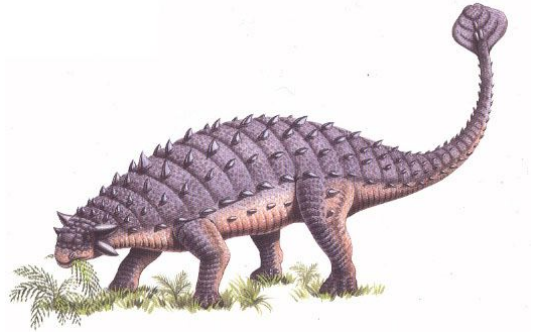
Re train What you want to detect





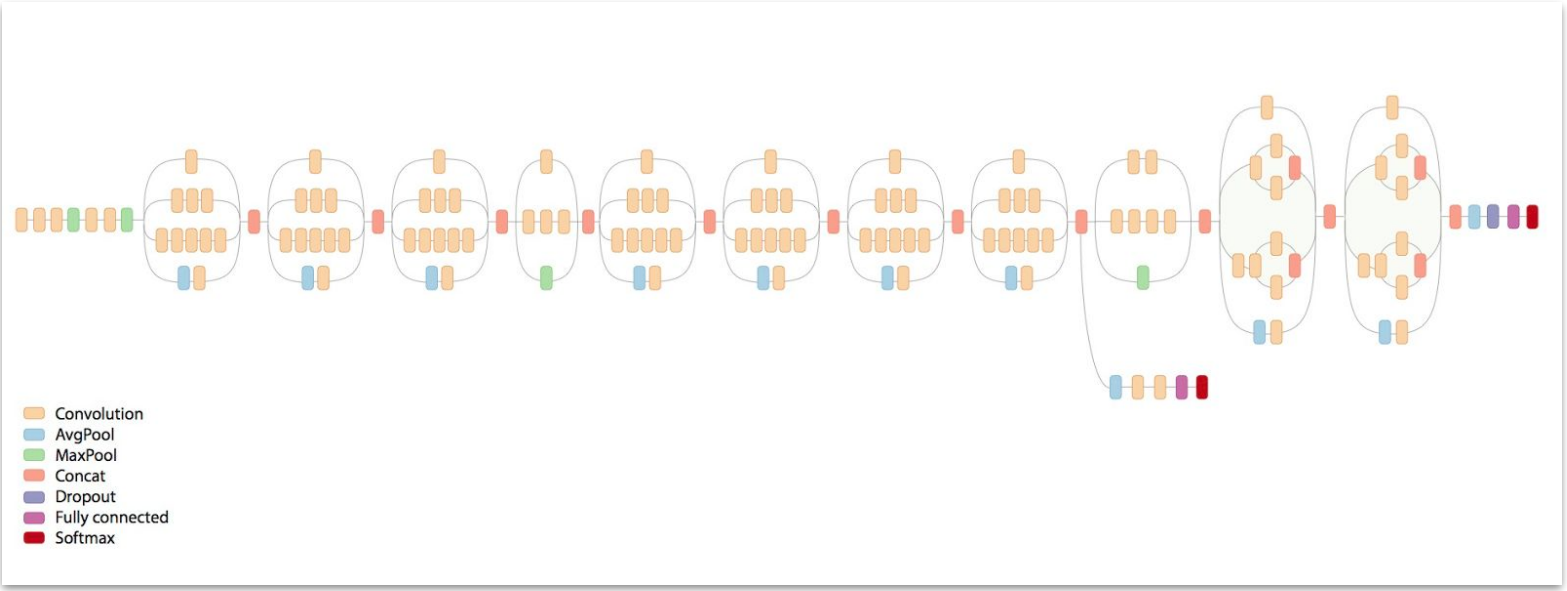
?





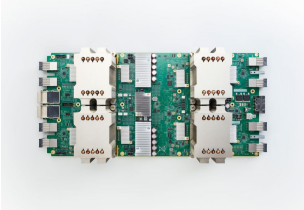
It is known that the depth of the neural network and the wider the layer, the higher the accuracy.
The picture is google inception v2 model.
Deep networks require exponential computing resources.

Re train Make own model

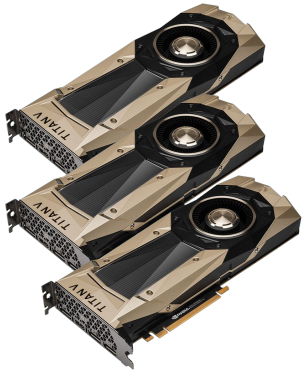


This time I released tpu 3.0 on Google i / o 2018.
To simplify the quantification of the flops power of tpu 2.0 and other graphics cards: Perhaps the maximum gpu that a non-enterprise user can have is titan.
Google uses a tpu pot that contains 64 tpu. Google may take weeks or even months to do the work for an hour.

Re train Make own model



google TPU 2.0
45 TFLOPS



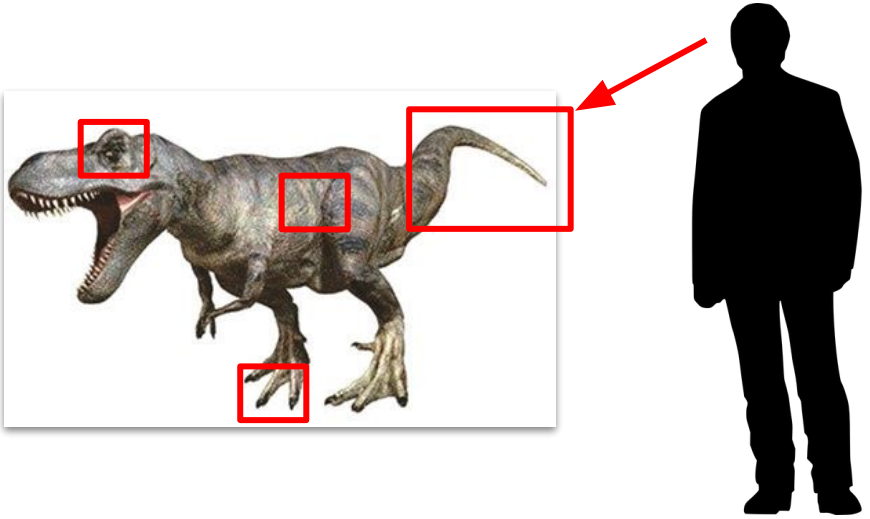
Titan v
15 TFLOPS

gtx 1080
9 TFLOPS

gtx 970
4 TFLOPS

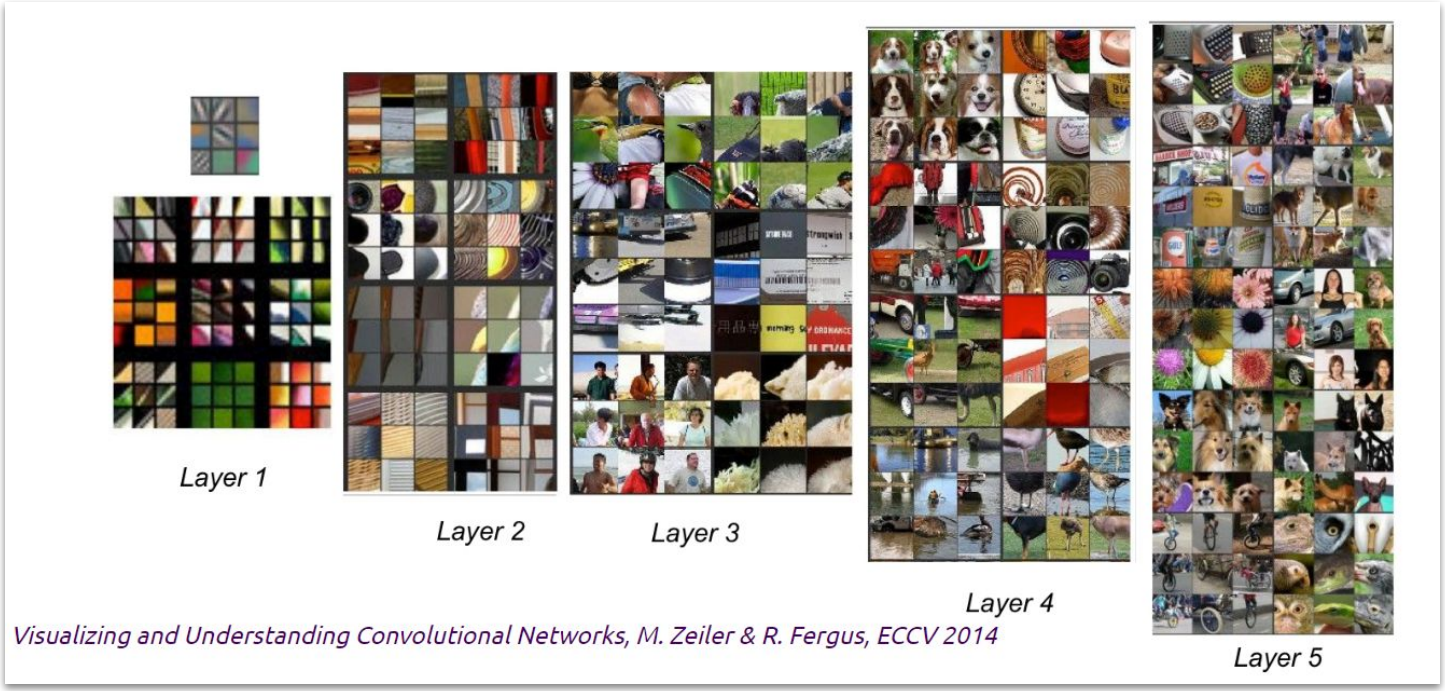
So we will use a method called transfer learning.
There are two people who do not know dinosaurs at all.
One is a very young baby and the other is an adult.

Re train Transfer learning



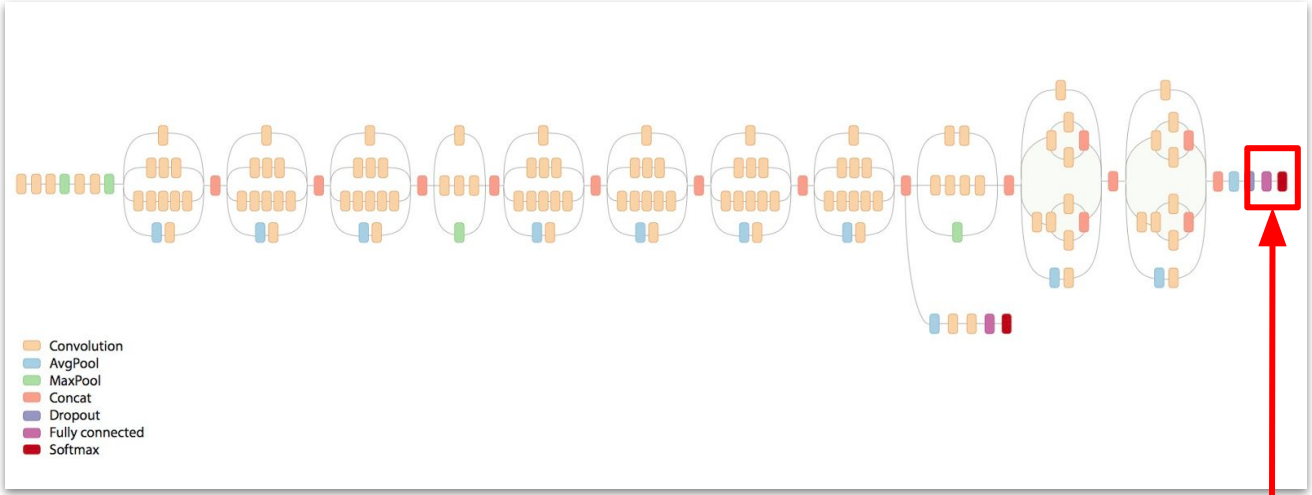
This photo is a visualization of the weights we have for each layer. Looking closer at the picture, the nearest layer to the input has a low-dimensional feature such as a line, As you go to the output, you can see that it has more detailed high-dimensional features.
Since low dimensional features have any object, transfer learning takes it as it is.

Re train Transfer learning



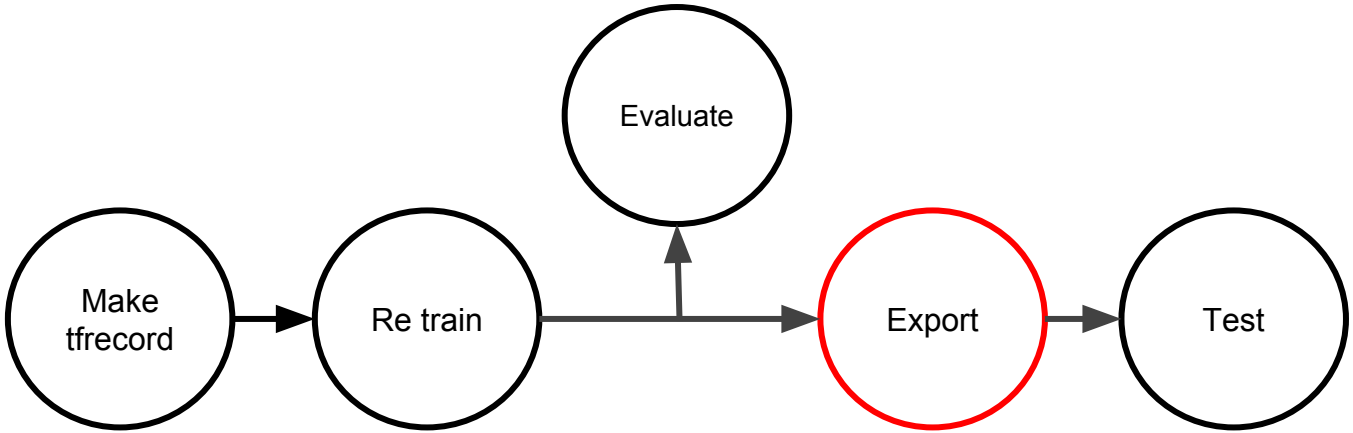
For example, inception v2 model, we take only the weights of previous layers and change only the last fully connected layer to custom labels.

Re train Transfer learning



replace custom class

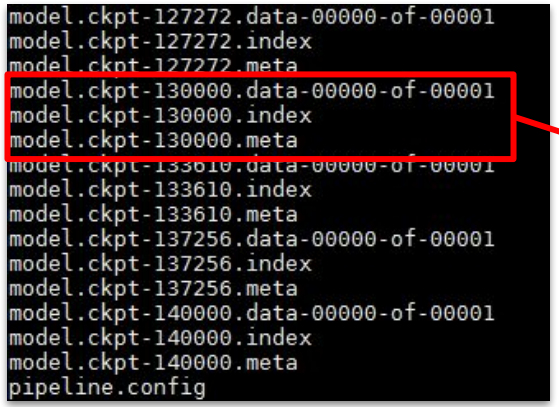
Tensorflow Object detection api



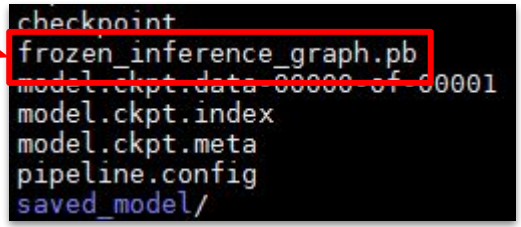
Once you train your model and reach your goal, we need to export this output to an executable file.
Below is the object detection default python export code.

Export

```
python object_detection/export_inference_graph.py \
--input_type=image_tensor \
--pipeline_config_path=pipeline.config \
--trained_checkpoint_prefix=train/model.ckpt-xxxxx \
--output_directory=output
```

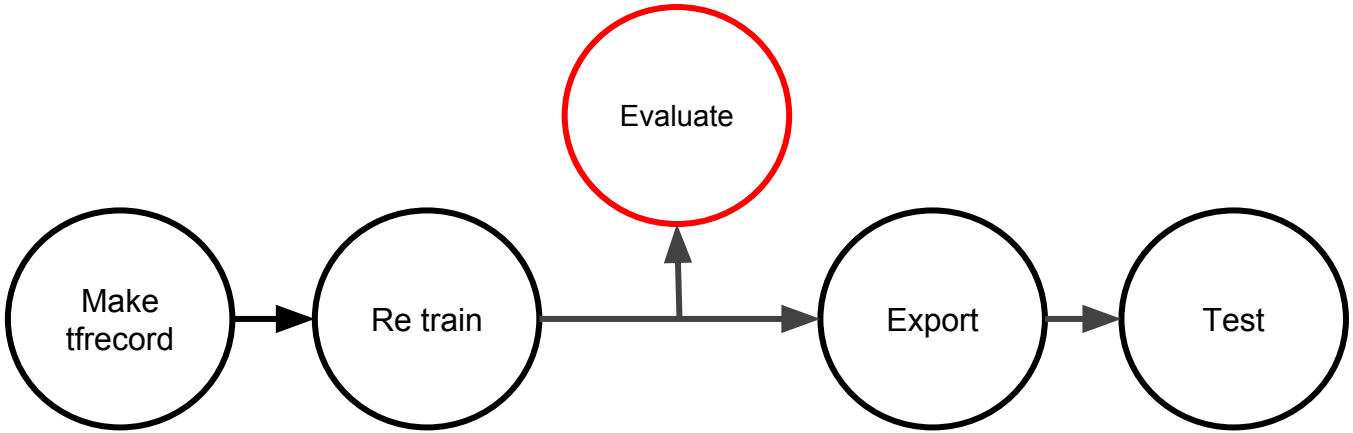


train_checkpoint_prefix



output_directory

Tensorflow Object detection api



We would like to quantitatively identify how good this model is during training, or after the train is over. The api to use is evaluate.
Below is the python evaluate code provided by the object detection api.
We can visually check the evaluate result value through the tensorboard.

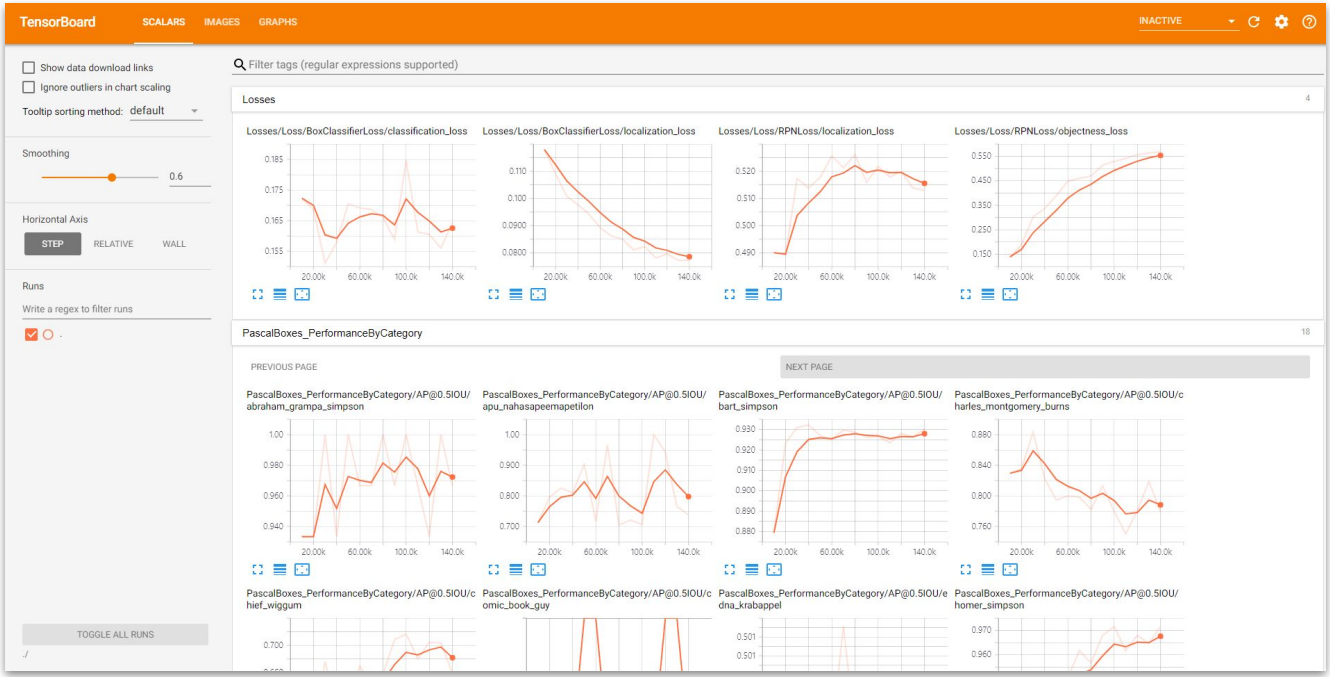
Evaluate

```
python eval.py \
    --logtostderr \
    --pipeline_config_path=training/ssd_mobilenet_v1_coco.config \
    --checkpoint_dir=training/ \
    --eval_dir=eval/
```

```
tensorboard --logdir=./
```

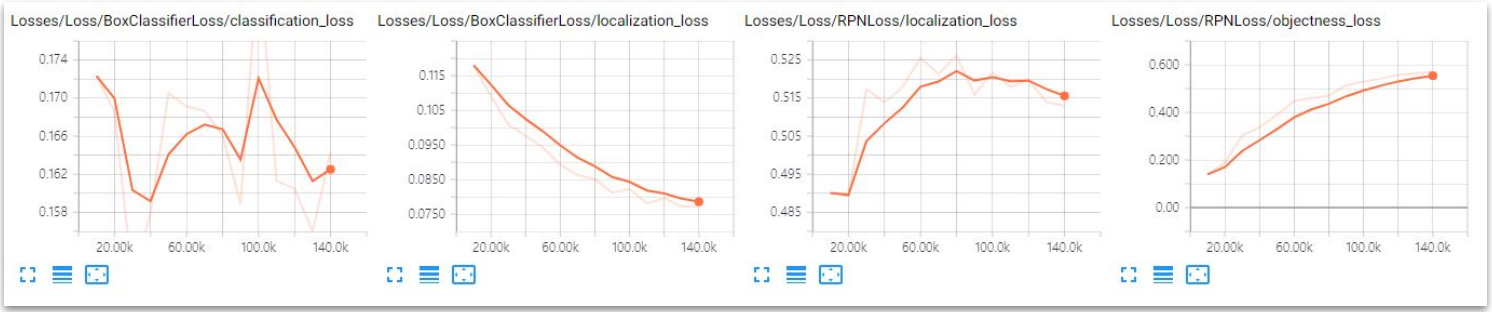
This is the main screen of the evaluated tensorboard.
Let's take a quick look at each one.

Evaluate Using tensorboard



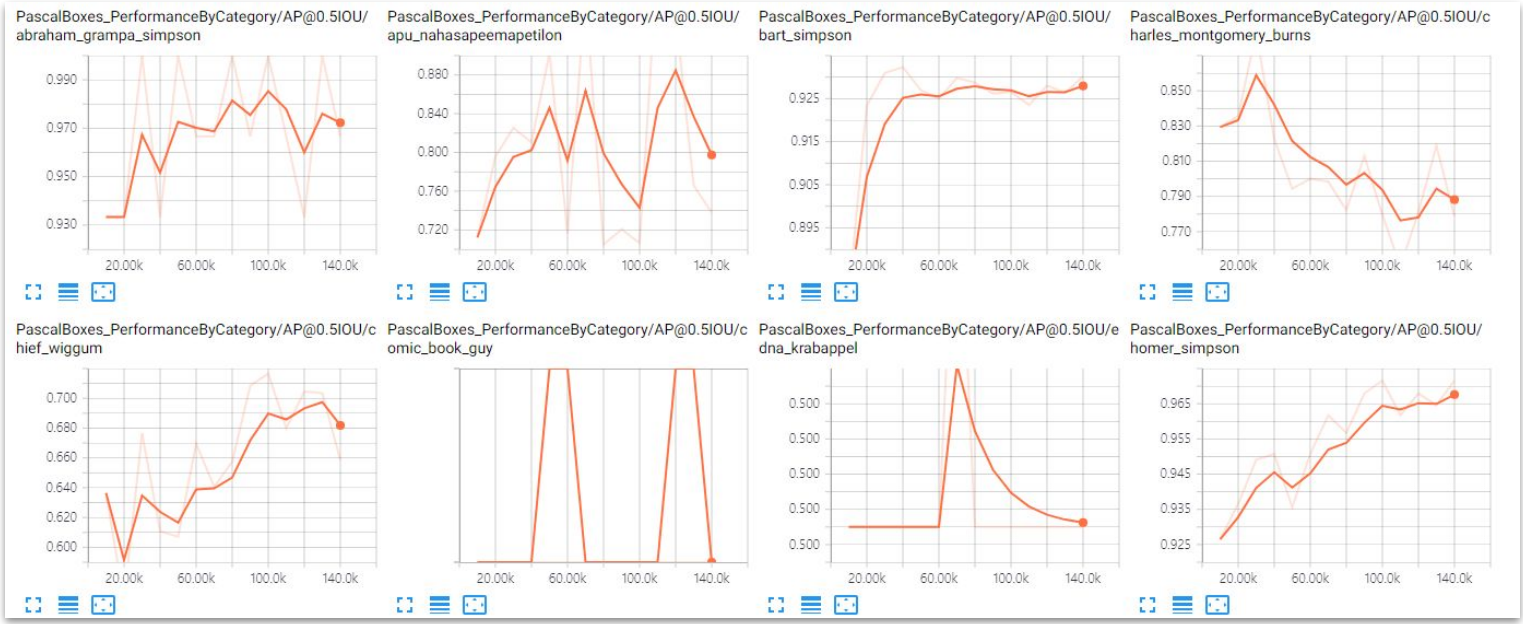
First, you can see the loss value for classification or localization.
The lower the loss value, the better.

Evaluate Using tensorboard



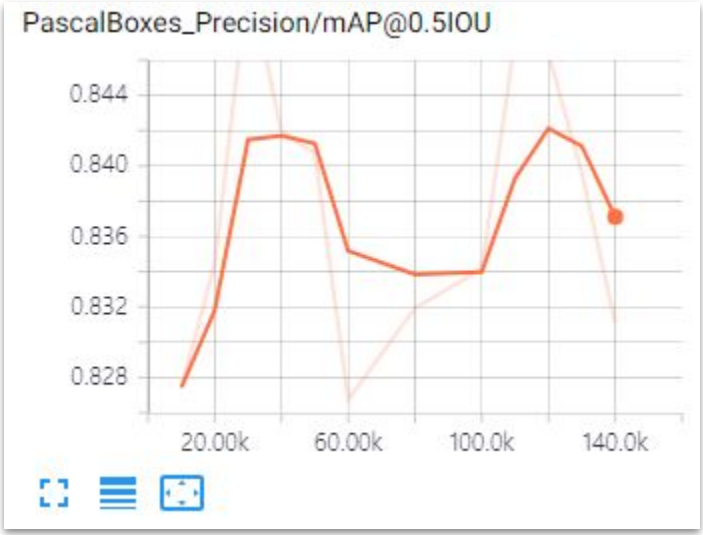
Second, you can check the accuracy of each class.
For different reasons, each character does not have the same classification accuracy.

Evaluate Using tensorboard



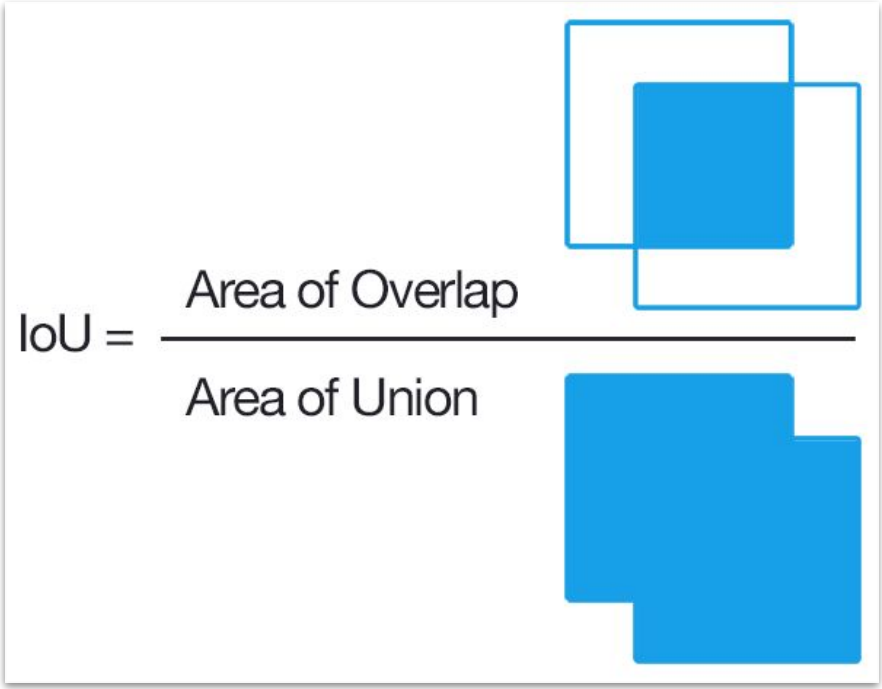
Third, you can measure the accuracy of the entire validation data.
Accuracy measurement method is 0.5IoU.

Evaluate Using tensorboard



IoU is an abbreviation of intersection of union, which is the ratio of the sum of the overlapping areas of the actual ground truth box and the area of the prediction truth box.
As you can see in the photo, the accuracy is not 100% even though the predicted value wraps around the actual value.

Evaluate Using tensorboard

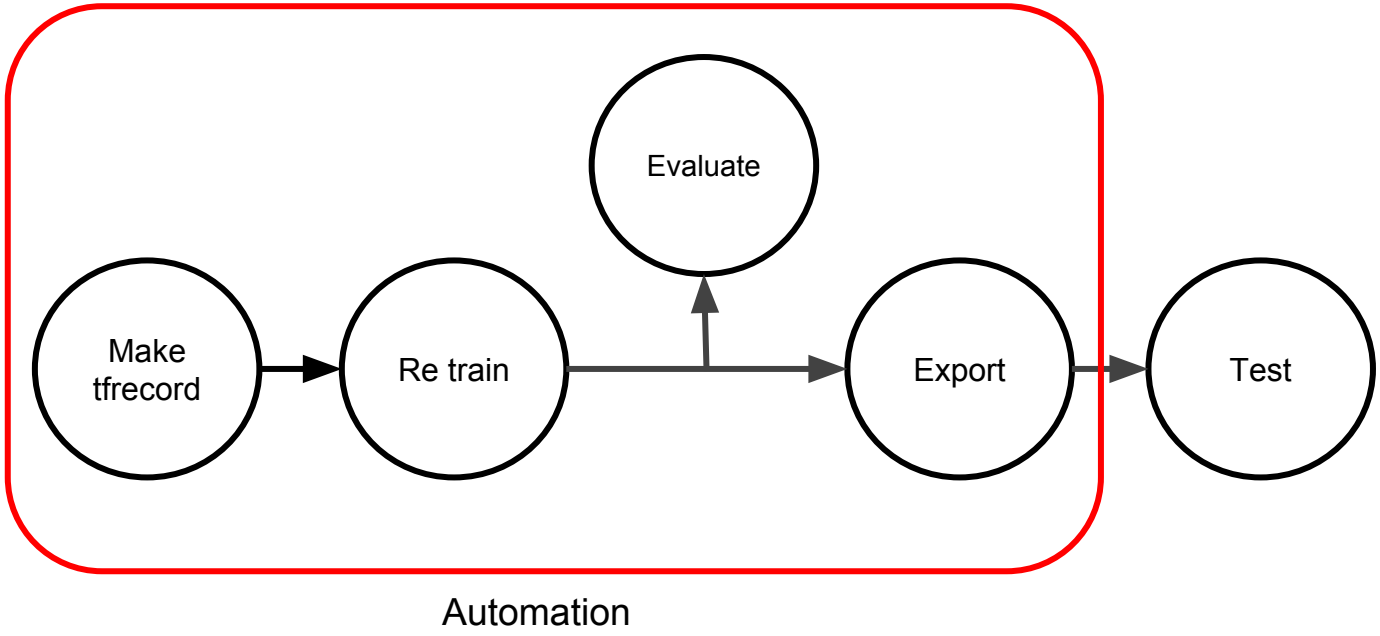


Finally, on the tensorboard, we can see how the estimate of the image changes every evaluation step.

Evaluate Using tensorboard

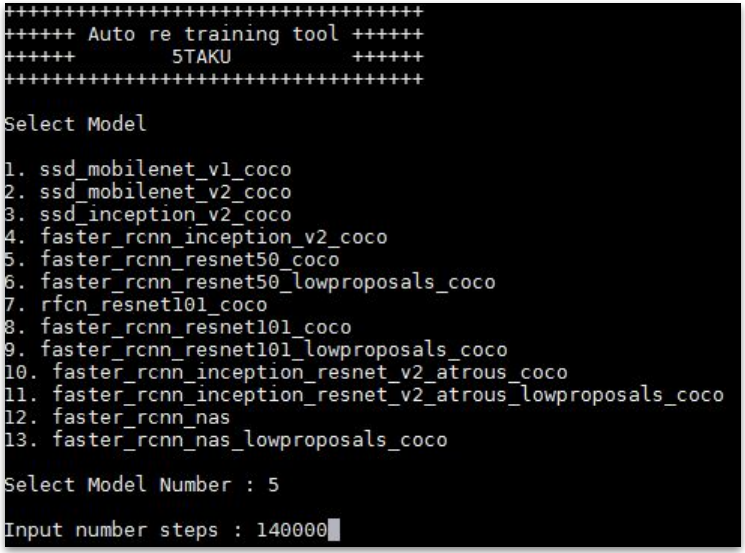
The set of data creation, train, evaluate, and export described above must be done manually by default.
I created a program that automatically feels inconvenienced.

Tensorflow object detection helper tool



Since you have set all the settings to the default settings that you can run by default, you only need to select the model and enter the training step.

Tensorflow object detection helper tool



You can check the log of the result value and the execution time of each process.

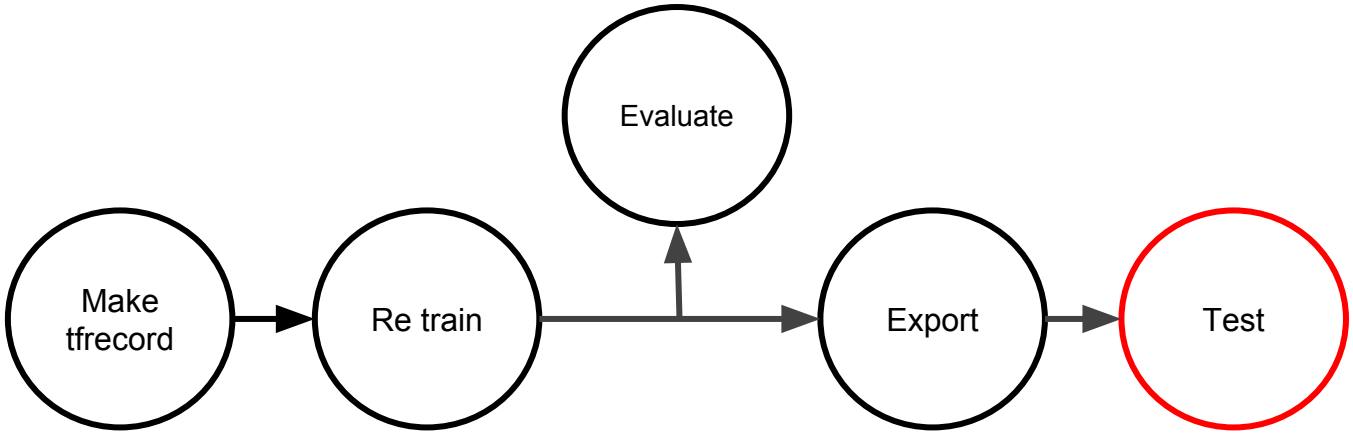
Tensorflow object detection helper tool

```
[INFO|main.py:92] 2018-06-04 06:11:22,555 > Program start [ model : faster_rcnn_resnet50_coco_2018_01_28, num steps : 140000 ]
[INFO|main.py:24] 2018-06-04 06:11:22,556 > Transfer learning start
[INFO|main.py:41] 2018-06-04 06:39:14,430 > Transfer learning Success [ Total learning time : 0 Hour 27 Minute 51 Second ]
[INFO|main.py:64] 2018-06-04 06:39:14,430 > Evaluate model start [ Step number : 10000 ]
[INFO|main.py:76] 2018-06-04 06:39:43,380 > Evaluate model Success
[INFO|main.py:24] 2018-06-04 06:39:43,381 > Transfer learning start
[INFO|main.py:41] 2018-06-04 07:07:10,456 > Transfer learning Success [ Total learning time : 0 Hour 27 Minute 27 Second ]
[INFO|main.py:64] 2018-06-04 07:07:10,457 > Evaluate model start [ Step number : 20000 ]
[INFO|main.py:76] 2018-06-04 07:07:38,888 > Evaluate model Success
[INFO|main.py:24] 2018-06-04 07:07:38,889 > Transfer learning start
[INFO|main.py:41] 2018-06-04 07:35:21,480 > Transfer learning Success [ Total learning time : 0 Hour 27 Minute 42 Second ]
[INFO|main.py:64] 2018-06-04 07:35:21,481 > Evaluate model start [ Step number : 30000 ]
[INFO|main.py:76] 2018-06-04 07:35:49,017 > Evaluate model Success
[INFO|main.py:24] 2018-06-04 07:35:49,018 > Transfer learning start
[INFO|main.py:41] 2018-06-04 08:03:31,860 > Transfer learning Success [ Total learning time : 0 Hour 27 Minute 42 Second ]
[INFO|main.py:64] 2018-06-04 08:03:31,860 > Evaluate model start [ Step number : 40000 ]
```

```
[INFO|main.py:24] 2018-06-04 12:18:01,556 > Transfer learning start
[INFO|main.py:41] 2018-06-04 12:45:53,040 > Transfer learning Success [ Total learning time : 0 Hour 27 Minute 51 Second ]
[INFO|main.py:64] 2018-06-04 12:45:53,040 > Evaluate model start [ Step number : 140000 ]
[INFO|main.py:76] 2018-06-04 12:46:20,527 > Evaluate model Success
[INFO|main.py:45] 2018-06-04 12:46:20,528 > Export model start
[INFO|main.py:60] 2018-06-04 12:46:35,302 > Export model Success
[INFO|main.py:119] 2018-06-04 12:46:35,303 > Program end [ Total time : 6 Hour 35 Minute 12 Second ]
```

https://github.com/5taku/tensorflow_object_detection_helper_tool

Tensorflow Object detection api



It is the code that loads the file generated by export and loads it into memory.
These images are detected using the loaded graph file.

Test

```
# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# Load a (frozen) Tensorflow model into memory.
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

https://github.com/tensorflow/models/blob/master/research/object_detection/object_detection_tutorial.ipynb

All the tests related to the announcement were tested on the Google cloud virtual machine.
The specifications are as follows.

Test cloud resource (google cloud compute engine)

16 vCPU

60gb Ram

1 x NVIDIA Tesla P100

ubuntu 16.0.4

python 2.7.12

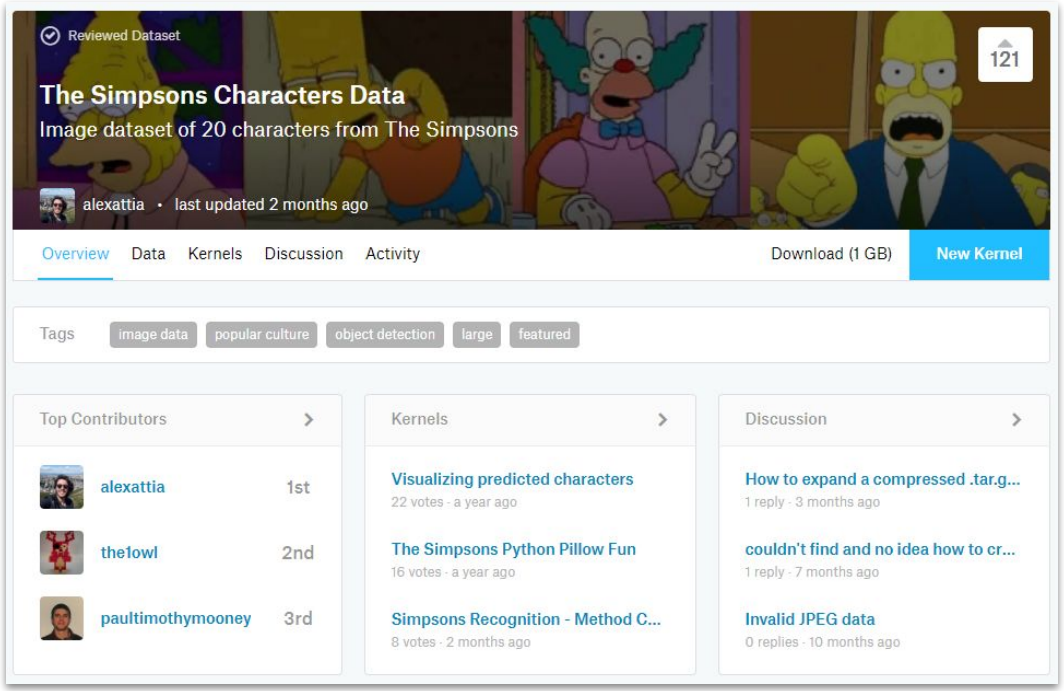
tensorflow 1.8.0

cuda 9.0

cudnn 7.1

The dataset used in the test was a simpson dataset from kaggle.

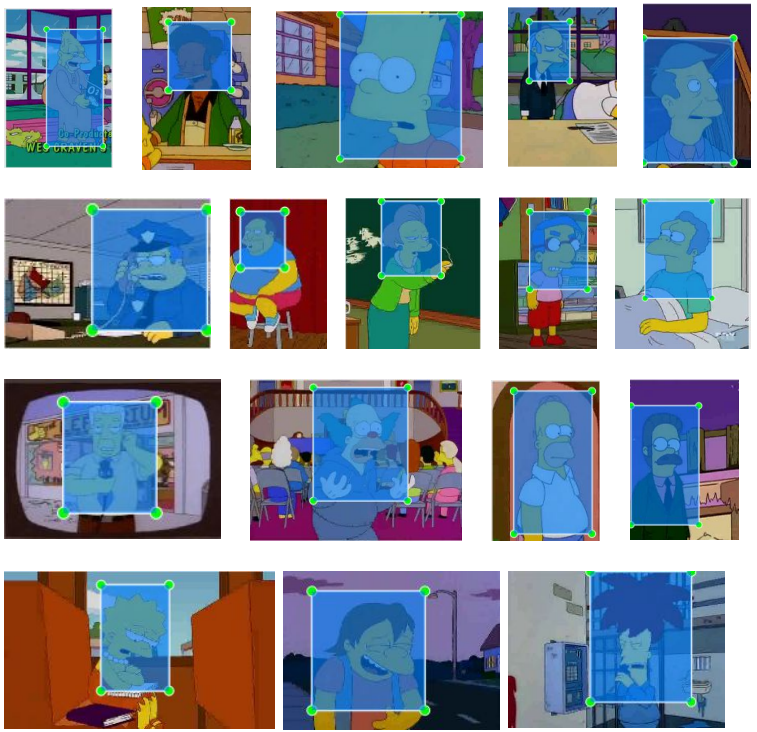
Test dataset



Each character is fully labeled. (300 to 2000)
Some of them have box data. (200 to 600)
The number of data is not constant for each character.

Test dataset

name	total	bounding_box
Homer Simpson	2246	612
Ned Flanders	1454	595
Moe Szyslak	1452	215
Lisa Simpson	1354	562
Bart Simpson	1342	554
Marge Simpson	1291	557
Krusty The Clown	1206	226
Principal Skinner	1194	506
Charles Montgomery Burns	1193	650
Milhouse Van Houten	1079	210
Chief Wiggum	986	209
Abraham Grampa Simpson	913	595
Sideshow Bob	877	203
Apu Nahasapeemapetilon	623	206
Kent Brockman	498	213
Comic Book Guy	469	208
Edna Krabappel	457	212
Nelson Muntz	358	219



I have trained 140,000 faster rcnn resnet 50 pre-train models with my tool.
Performs evaluation every 10,000 times.
Total training time is approximately 6 hours and 30 minutes.

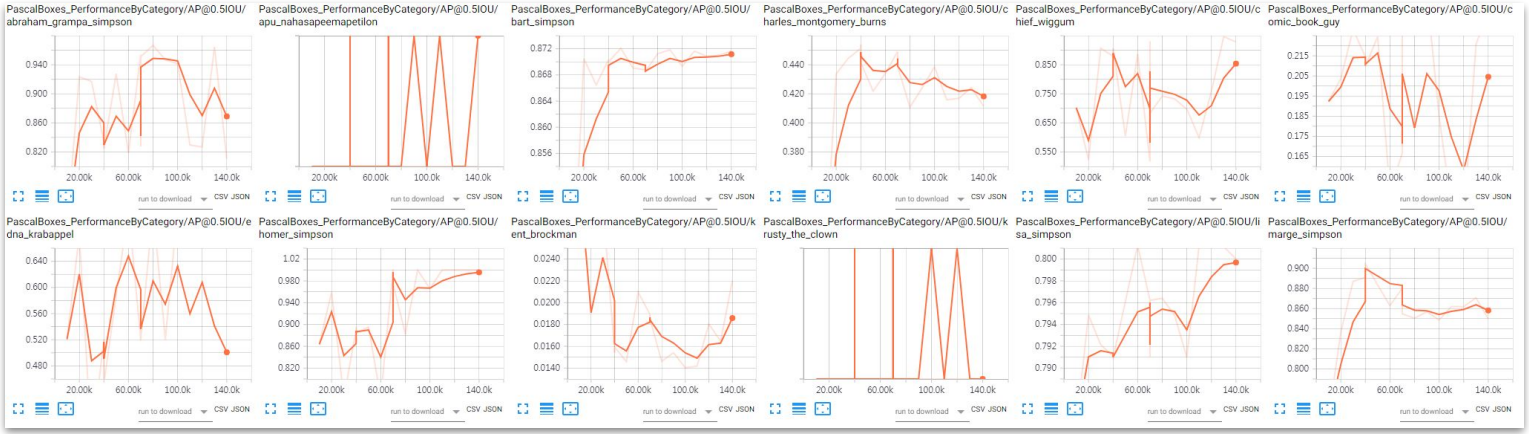
Train , Evaluate , Export

model = faster rcnn resnet 50
training step = 140,000
training : validate rate = 8 : 2

```
[INFO|main.py:24] 2018-06-04 12:18:01,556 > Transfer learning start
[INFO|main.py:41] 2018-06-04 12:45:53,040 > Transfer learning Success [ Total learning time : 0 Hour 27 Minute 51 Second ]
[INFO|main.py:64] 2018-06-04 12:45:53,040 > Evaluate model start [ Step number : 140000 ]
[INFO|main.py:76] 2018-06-04 12:46:20,527 > Evaluate model Success
[INFO|main.py:45] 2018-06-04 12:46:20,528 > Export model start
[INFO|main.py:60] 2018-06-04 12:46:35,302 > Export model Success
[INFO|main.py:119] 2018-06-04 12:46:35,303 > Program end [ Total time : 6 Hour 35 Minute 12 Second ]
```

The overall iou value was 0.67, which did not perform well.
IoU for each character is also inferior in performance.

Test tensor board result

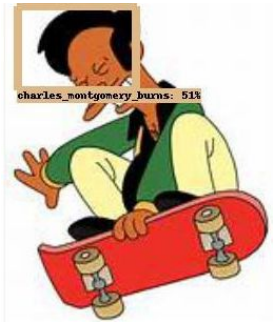


I did not touch the figures well, so I decided to check the image by inserting the actual image.
Twenty images per character were predialized. There are a few errors that I see in my eyes :(
The criteria for each T F N are as follows. (If multiple labels are detected on one object, the highest accuracy is T.)

Test handmade check

True (T)

False (F)



None (N)

NAME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
abraham_grampa_simpson	T	T	T	T	T	T	T	T	T	T	T	F	T	T	T	F	T	T	T
apu_nahasapeemapetilon	N	T	N	N	N	N	F	N	N	N	F	N	F	N	F	N	F	N	N	N
bart_simpson	T	T	T	T	T	T	T	T	F	T	T	T	T	T	T	T	T	T	T	N
charles_montgomery_burns	T	N	N	T	T	N	N	T	T	N	T	N	T	T	N	F	T	F	N	T
chief_wiggum	N	T	N	N	F	T	N	F	N	F	F	N	T	F	N	N	T	N	T	N
comic_book_guy	N	T	N	T	T	T	T	N	T	T	F	T	N	F	N	T	T	N	F	N
edna_krabappel	N	N	N	N	N	F	N	N	N	F	N	N	N	N	T	F	F	F	N	F
homer_simpson	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	T	T	T	T	T
kent_brockman	T	N	N	N	N	N	F	T	T	N	N	F	N	T	N	N	T	N	N	N
krusty_the_clown	T	T	N	T	N	T	N	N	N	N	T	T	T	N	T	T	T	T	T	T
lisa_simpson	F	N	N	N	F	N	N	T	N	T	F	N	N	T	N	T	T	N	T	T
marge_simpson	T	T	T	T	T	F	N	T	T	F	T	F	T	T	T	N	T	T	T	T
milhouse_van_houten	F	N	N	T	N	T	F	N	N	T	N	N	N	T	T	F	N	F	N	T
moe_szyslak	T	T	T	T	T	N	T	T	T	T	T	T	T	T	T	N	T	T	T	N
ned_flanders	N	N	N	T	T	F	T	T	T	T	T	T	T	T	T	F	T	T	T	T
nelson_muntz	N	T	N	F	N	T	F	F	F	F	F	T	N	T	T	F	F	N	F	T
principal_skinner	T	T	T	T	N	T	T	T	N	T	N	T	T	N	T	N	N	N	T	T
sideshow_bob	N	N	N	N	N	N	N	N	N	N	N	F	N	N	N	N	F	N	N	N

Training : 140,000
Data : 6,932

The result of training 6,000 data for 140,000 times is not satisfactory.
How can I improve my accuracy?
There are many ways to increase accuracy.

Improved accuracy

1. **Change Model**
2. **Increase data**
    a. **Data augmentation**
    b. **Labelling**
    c. **Active learning**
3. **etc….**

In choosing a model, tensorflow already offers several models.
We only need to select the model considering the accuracy, prediction speed, and training speed.
In the tests I've done, resnet50, inception v2 has guaranteed the best performance.

Improved accuracy change model

Model name	Speed (ms)	COCO mAP[^1]	Size	Training time	Outputs
ssd_mobilenet_v1_coco	30	21	86M	9m 44s	Boxes
ssd_mobilenet_v2_coco	31	22	201M	11m 12s	Boxes
ssd_inception_v2_coco	42	24	295M	8m 43s	Boxes
faster_rcnn_inception_v2_coco	58	28	167M	4m 43s	Boxes
faster_rcnn_resnet50_coco	89	30	405M	4m 28s	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		405M	4m 30s	Boxes
rfcn_resnet101_coco	92	30	685M	6m 19s	Boxes
faster_rcnn_resnet101_coco	106	32	624M	6m 13s	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		624M	6m 13s	Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	712M	18m 6s	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		712M		Boxes
faster_rcnn_nas	1833	43	1.2G	47m 49s	Boxes
faster_rcnn_nas_lowproposals_coco	540		1.2G		Boxes

Rotating, zooming in and out using a single image is a widely used method of increasing data.
However, if you change the size of the image, you have to change the box position as well, which will be quite cumbersome.

Improved accuracy Data augmentation

size up

brightness down

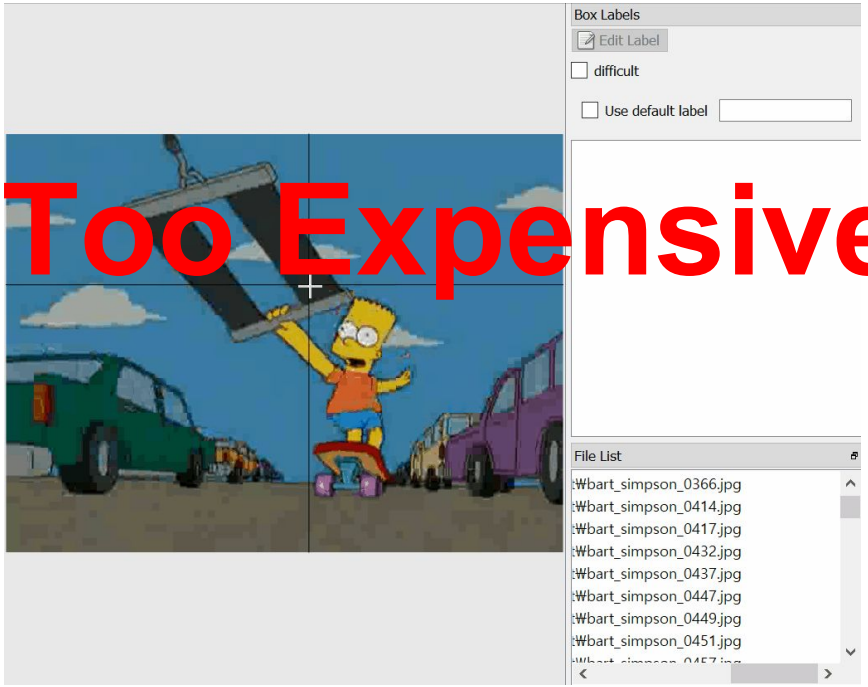
size down

original

brightness up

Another way to increase data is to manually create one object box boundary.
Manual labeling results in many labeling costs.

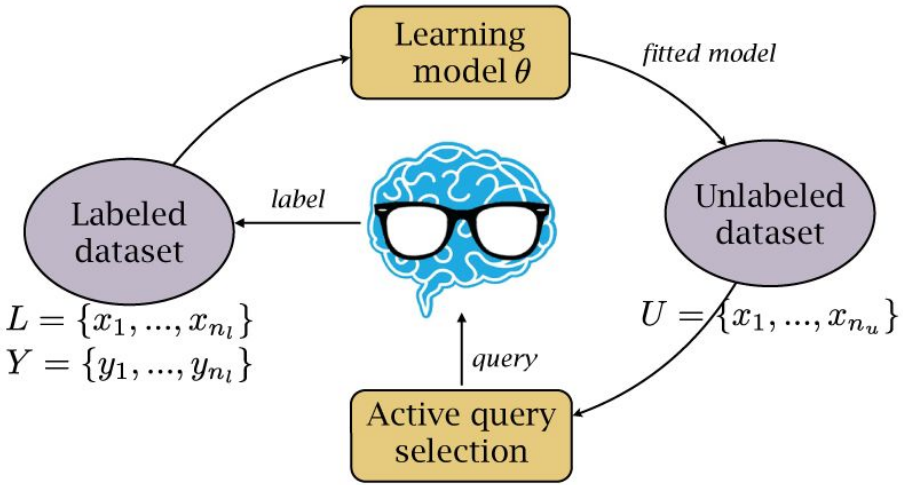
Improved accuracy labelling



Too Expensive COST!!!

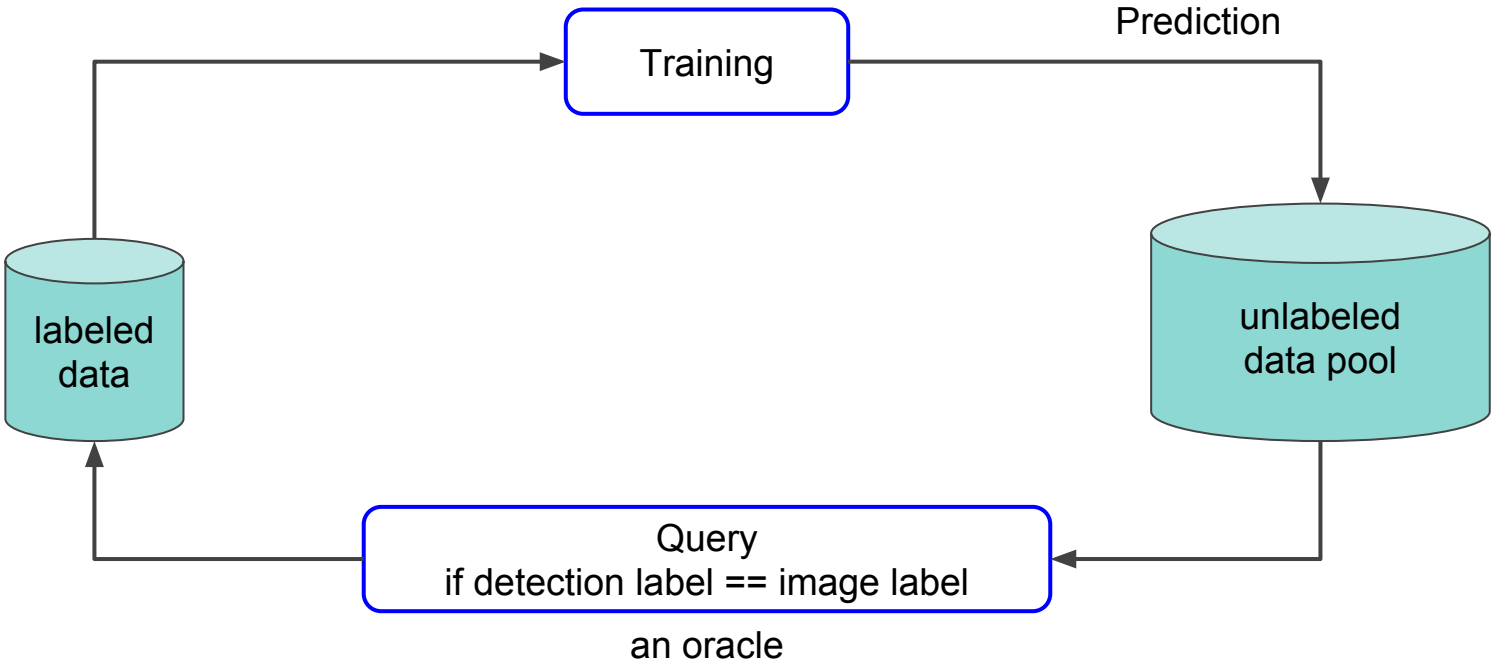
Introducing Active learning, which allows labeling of images at low cost.
The basic concept is to use the data obtained by predicting the candidate data with the model made by using a small amount of data, and to use the result as the training data again. Of course, the predicted result is verified by the person or machine named oracle.

Active learning



Simpson dataset is an ideal data set for active learning because each unboxed image is labeled.
The resnet 50 algorithm is trained 20,000 times with 6,000 box data provided initially.
Then, 12,000 unboxed images are predicted, and if the prediction result matches the image label, it is moved to the train dataset.

Active learning



The number of images adopted in the training set is as follows.
At the first prediction, two times the image was adopted, and since then the number has been reduced, but it has been adopted as a steady train set.

Active learning

ID	NAME	1 step	2 step	3 step	4 step	5 step	6 step	7 step
1	homer_simpson	626	1830	1980	2018	2050	2061	2066
2	ned_flanders	607	1214	1335	1355	1381	1387	1388
3	moe_szyslak	215	692	945	1121	1170	1238	1246
4	lisa_simpson	578	1116	1217	1234	1271	1271	1271
5	bart_simpson	571	1133	1243	1256	1256	1265	1272
6	marge_simpson	568	1229	1244	1248	1248	1250	1250
7	krusty_the_clown	239	558	1002	1139	1145	1147	1147
8	principal_skinner	519	1054	1110	1114	1116	1119	1122
9	charles_montgomery_burns	663	1058	1085	1099	1104	1105	1107
10	milhouse_van_houten	223	776	897	925	952	954	965
11	chief_wiggum	206	561	788	837	858	863	867
12	abraham_grampa_simpson	608	844	875	878	878	878	878
13	sideshow_bob	202	217	420	651	733	760	772
14	apu_nahasapeemapetilon	223	537	570	581	588	588	590
15	kent_brockman	217	250	270	324	371	415	415
16	comic_book_guy	224	351	387	395	398	403	403
17	edna_krabappel	226	227	327	370	384	386	390
18	nelson_muntz	217	284	287	288	297	297	299
Total		6932	13931	15982	16833	17200	17387	17448
Increase rate			6999	2051	851	367	187	61

The first and seventh data split ratios.
Because the number of images per character differs a lot, the number of training is uneven.
However, the number of training data in all classes increased by 30% ~ 300%.

train , evaluate , export

```
43,629 > 1 Step start
TF Record Generator Start
              TF Record Summary
   ID              NAME          Train    Validate
   1         homer_simpson        500       126
   2          ned_flanders        485       122
   3          moe_szyslak         172        43
   4          lisa_simpson        462       116
   5          bart_simpson        456       115
   6         marge_simpson        454       114
   7       krusty_the_clown       191        48
   8      principal_skinner       415       104
   9     charles_montgomery_burns    530       133
   10    milhouse_van_houten      178        45
   11        chief_wiggum         164        42
   12    abraham_grampa_simpson     486       122
   13        sideshow_bob         161        41
   14    apu_nahasapeemapetilon     178        45
   15       kent_brockman         173        44
   16       comic_book_guy        179        45
   17       edna_krabappel        180        46
   18        nelson_muntz         173        44
```

```
50,884 > 7 Step start
TF Record Generator Start
              TF Record Summary
   ID              NAME          Train    Validate
   1         homer_simpson        1652      414
   2          ned_flanders        1110      278
   3          moe_szyslak         996       250
   4          lisa_simpson        1016      255
   5          bart_simpson        1017      255
   6         marge_simpson        1000      250
   7       krusty_the_clown       917       230
   8      principal_skinner       897       225
   9     charles_montgomery_burns    885       222
   10    milhouse_van_houten      772       193
   11        chief_wiggum         693       174
   12    abraham_grampa_simpson     702       176
   13        sideshow_bob         617       155
   14    apu_nahasapeemapetilon     472       118
   15       kent_brockman         332        83
   16       comic_book_guy        322        81
   17       edna_krabappel        312        78
   18        nelson_muntz         239        60
```

Because there is a process of predicting 11,000 images, it took 21 hours for a task that took 6 hours and 30 minutes to work with active learning.

train , evaluate , export

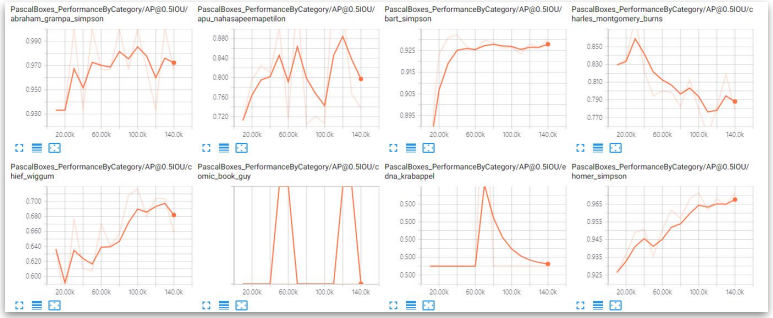
```
2018-06-02 00:13:16,747 > Transfer learning Success [ Total learning time : 0 Hour 14 Minute 3 Second ]
2018-06-02 00:13:16,747 > Evaluate model start [ Step number : 135000 ]
2018-06-02 00:13:44,805 > Evaluate model Success
2018-06-02 00:13:44,806 > Transfer learning start
2018-06-02 00:27:49,457 > Transfer learning Success [ Total learning time : 0 Hour 14 Minute 4 Second ]
2018-06-02 00:27:49,457 > Evaluate model start [ Step number : 140000 ]
2018-06-02 00:28:16,882 > Evaluate model Success
2018-06-02 00:28:16,882 > Export model start
2018-06-02 00:28:31,449 > Export model Success
2018-06-02 00:28:32,303 > Inference Process Start
2018-06-02 01:04:00,692 > Inferrence Image Prediction End [ Total Generator time : 0 Hour 35 Minute 27 Second ] Detection Count : 81/1132
2018-06-02 01:15:03,618 > Test Image Prediction End [ Total Generator time : 0 Hour 11 Minute 2 Second ]
2018-06-02 01:15:03,618 > Inferrence Process End [ Total Generator time : 0 Hour 46 Minute 31 Second ]
2018-06-02 01:15:03,917 > Program end [ Total time : 21 Hour 10 Minute 20 Second ]
```

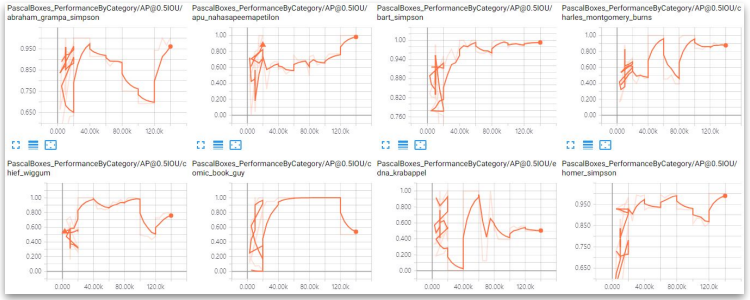
When the localization loss value was learned from the existing 6900 chapters, it decreased from 0.115 to 0.06 after applying active learning.

Test tensor board result

Accuracy for each class has also gradually decreased.
In active learning, the initial graph value is suddenly suddenly the train data is suddenly increased, but it is not certain.

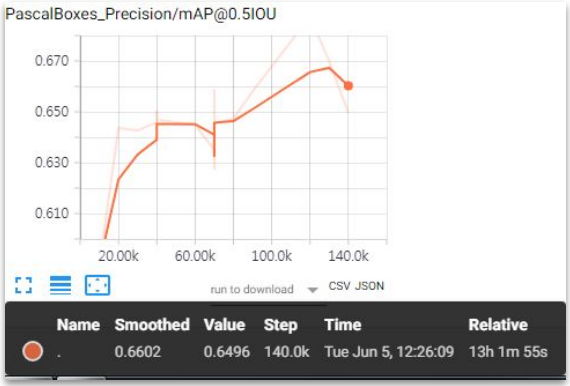
Test tensor board result



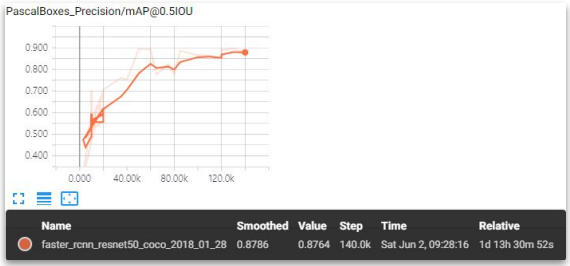


IOU accuracy for the entire class has been increased from 0.66 to 0.87.

Test tensor board result



0.66



0.87

As data increases, the accuracy of each test case increases.
Accuracy increases rapidly in the early days and continues to increase in the future.

Active learning

NAME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
abraham_grampa_simpson	T	T	T	T	T	T	T	T	T	F	F	T	F	T	T	T	N	T	T	T
apu_nahasapeemapetilon	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	N	F	N	F	F
bart_simpson	T	T	T	T	F	T	T	T	N	T	T	T	T	T	T	F	T	T	T	N
charles_montgomery_burns	T	T	N	N	T	T	T	N	T	N	T	T	T	T	T	N	T	T	N	T
chief_wiggum	F	T	N	N	N	N	N	N	N	N	N	N	N	N	F	N	N	N	N	N
comic_book_guy	F	F	N	N	N	N	N	N	F	N	N	F	N	N	N	N	F	N	N	F
edna_krabappel	F	N	N	N	N	F	F	F	F	N	N	N	N	N	F	N	N	N	N	N
homer_simpson	T	T	T	T	T	T	F	T	T	T	T	T	T	T	T	F	T	T	T	T
kent_brockman	F	F	F	F	F	F	F	N	T	N	F	N	N	N	N	F	N	N	N	N
krusty_the_clown	N	N	N	N	N	N	N	N	N	N	N	N	N	N	F	N	N	N	N	N
lisa_simpson	T	F	F	F	F	N	N	T	N	N	F	N	N	N	N	N	N	N	T	N
marge_simpson	T	T	T	T	T	N	N	T	T	N	T	N	T	T	T	N	T	T	T	T
milhouse_van_houten	F	N	F	N	F	T	F	F	N	T	F	T	F	F	F	T	F	T	T	N
moe_szyslak	N	N	N	N	T	T	T	T	T	N	T	N	T	T	T	T	N	N	T	N
ned_flanders	F	N	F	N	T	F	T	N	N	T	F	T	F	T	T	F	F	T	F	F
nelson_muntz	N	T	N	T	N	T	N	T	F	T	N	T	N	T	T	N	N	N	F	T
principal_skinner	T	F	F	F	F	F	N	N	N	F	N	N	F	N	N	N	F	T	N	F
sideshow_bob	N	N	N	N	N	N	F	F	F	N	N	N	N	F	N	N	N	N	N	N

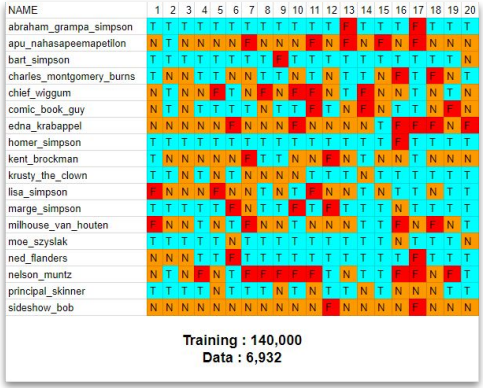
Training : 20,000
Data : 6,932

Extra page

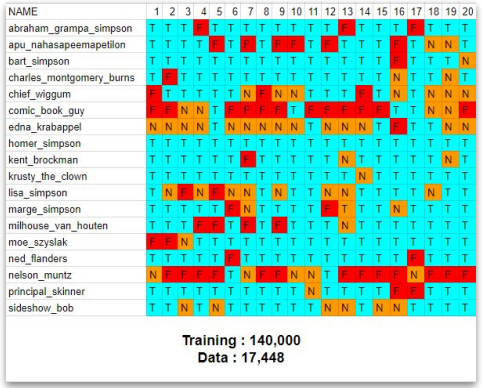
Further analysis

The first picture is the result of learning 6,932 pieces of data 140,000 times.
The second picture is the result of learning 140,000 times while adding training data to active learning every 20,000 times.
The last picture is the result of learning 140,400 data from the beginning of 17,448 data from the last result of active learning.

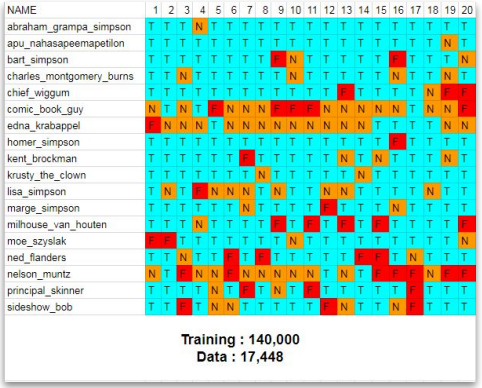
Active learning



Data 6,932
0 to 140,000



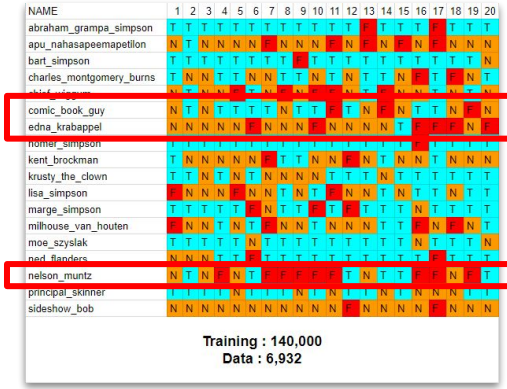
0 to 140,000
active learning every 20,000

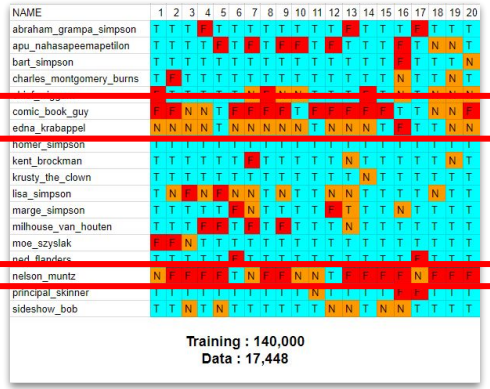


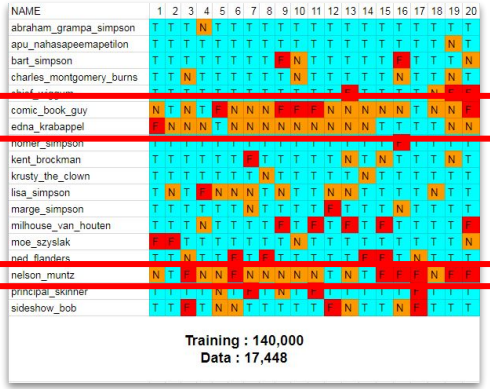
Data 17,448
0 to 140,000

Looking at the results, some characters seem to show little increase in accuracy.
This is because the distribution of the entire dataset per class is so small that even if active learning is used, the percentage of the character in the entire data set is rather lower.

Active learning







Data 6,932
0 to 140,000

comic_book_guy : 3.23 %
edna_krabappel : 3.26 %
nelson_muntz : 3.13 %

0 to 140,000
active learning every 20,000

Data 17,448
0 to 140,000

comic_book_guy : 2.30 %
edna_krabappel : 2.23 %
nelson_muntz : 1.71 %

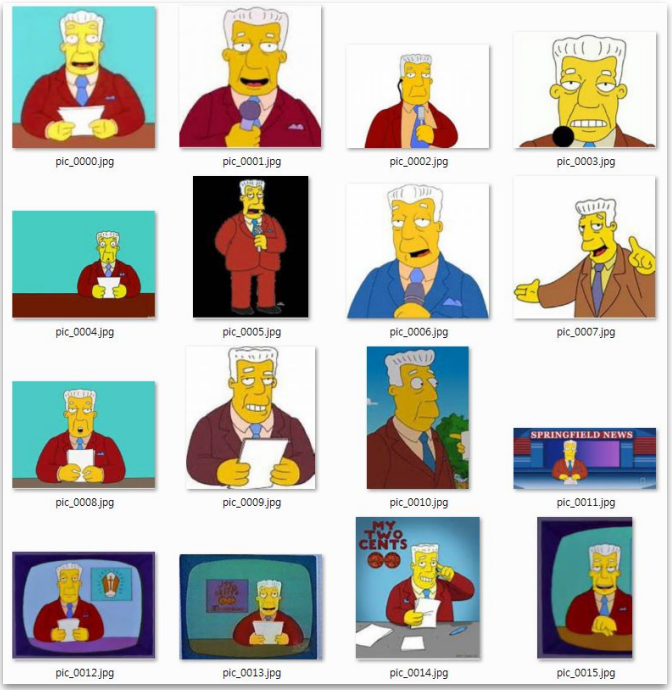
For kent brockman, the accuracy is fairly high, even though the data rate is 2.37%.
Though the test data may be luckily well-detected, I think that the characteristic white hair represents different classes and salient features.

Active learning

NAME	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
abraham_grampa_simpson	T	T	T	F	T	T	T	T	T	T	T	T	F	T	T	T	F	T	T	T
apu_nahasapeemapetilon	T	T	T	T	F	T	F	T	F	F	T	F	T	T	T	T	F	T	N	N
bart_simpson	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	F	T	T	T
charles_montgomery_burns	T	F	T	T	T	T	T	T	T	T	T	T	T	T	T	N	T	T	N	T
chief_wiggum	F	T	T	T	T	T	N	F	N	N	T	T	T	F	T	N	T	N	N	N
comic_book_guy	F	F	N	N	T	F	F	F	F	T	F	F	F	F	F	T	T	N	N	F
edna_krabappel	N	N	N	N	T	N	N	N	N	N	T	N	N	N	T	F	T	T	N	N
homer_simpson	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
kent_brockman	T	T	T	T	T	T	F	T	T	T	T	T	N	T	T	T	T	T	N	T
krusty_the_clown	T	T	T	T	T	T	T	T	T	T	T	T	T	N	T	T	T	T	T	T
lisa_simpson	T	N	F	N	F	N	N	T	N	T	T	N	N	T	T	T	T	N	T	T
marge_simpson	T	T	T	T	T	F	N	T	T	T	T	F	T	T	T	N	T	T	T	T
milhouse_van_houten	T	T	T	F	F	T	F	T	F	T	T	T	N	T	T	T	T	T	T	T
moe_szyslak	F	F	N	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
ned_flanders	T	T	T	T	T	F	T	T	T	T	T	T	T	T	T	T	F	T	T	T
nelson_muntz	N	F	F	F	F	T	N	F	F	N	N	T	F	F	F	F	N	F	F	F
principal_skinner	T	T	T	T	T	T	T	T	T	T	N	T	T	T	T	F	F	T	T	T
sideshow_bob	T	T	N	T	N	T	T	T	T	T	T	N	N	T	N	N	T	T	T	T

Training : 140,000
Data : 17,448

kent_brockman : 2.37 %



Today we looked at the entire Tensorflow object detection API.
I also had a brief introduction to the concepts of tranfer learning and active learning and the helper tool I created.
We have confirmed that the accuracy of actual data is increased by active learning with low labeling cost.

Thank you!

1. Tensorflow Object detection API
2. Transfer learning
3. Object detection API helper tool
4. Active learning (with test result)

Thank you!

Q&A