

Playing card detection using CNN



Akhil C K · [Follow](#)

6 min read · Jun 7, 2021



7



1



In this blog post we'll be creating a playing card detector — finding out which cards are present in the image(hearts of king, clubs of three etc). We will be using a pre-trained classifier with specific neural network architectures .

Before dive deep let me first briefly explain object detection and classification.

Detection Vs Classification



Fig 1.

When performing *image classification*, we present one input image to the network and obtain one class label out.(Fig 1)

Fig 1 will be recognised as King of hearts(Kd) by the system.



Fig 2.

But when performing *object detection*, we can present one input image and obtain multiple bounding boxes and class labels out.(Fig 2)

Fig 2 shows how the algorithm detect and localize not just one but multiple cards in the image.

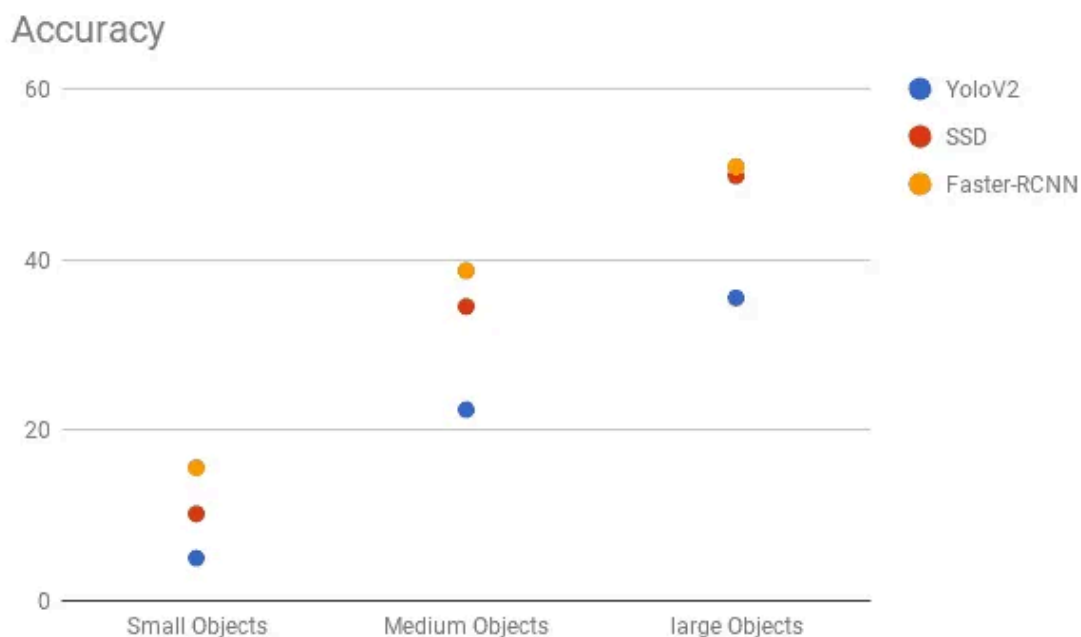
Alright, Lets get started then.

1. Download the full TensorFlow object detection repository located at <https://github.com/tensorflow/models> by clicking the “Clone or Download” button and downloading the zip file.

This working directory will contain the full TensorFlow object detection framework, as well as training images, training data, trained classifier, configuration files, and everything else needed for the card detection classifier.

2. Download the Faster-RCNN-Inception-V2-COCO model.

TensorFlow provides several object detection models (pre-trained classifiers with specific neural network architectures) in its model zoo.



Model such as the SSD-MobileNet have an architecture that allows for faster detection but with less accuracy, while models such as the Faster-RCNN model give slower detection but with more accuracy.

We will use the Faster-RCNN-Inception-V2 model. [Download the model here.](#) Open the downloaded faster_rcnn_inception_v2_coco_2018_01_28.tar.gz file and extract the faster_rcnn_inception_v2_coco_2018_01_28 folder to the object_detection folder.

3. Install necessary dependencies.

Reach out to this [link](#), under Table of contents, Setup section, click on Installation sub section. Basically the Installation section consist of list of libraries on which TensorFlow Object Detection API depends. So, install each and every dependencies before moving forward.

4. Configure PYTHONPATH

We need to point PYTHONPATH to the \models, \models\research, and \models\research\slim directories. We can do this by issuing export command

```
export
PYTHONPATH=/home/nidhin/Ck/card_detection/models:/home/nidhin/Ck/card
_detection/models/research:/home/nidhin/Ck/card_detection/models/rese
arch/slim
```

5. Compile protobufs

TensorFlow is using protobufs to configure model and training parameters. So we need to compile them next.

Download and install the 3.0 release of protoc, then unzip the file.

From tensorflow/models/research/

```
wget -O protobuf.zip  
https://github.com/google/protobuf/releases/download/v3.0.0/protoc-3.0.0-linux-x86\_64.zip  
unzip protobuf.zip
```

Run the compilation process again, but use the downloaded version of protoc

```
./bin/protoc object_detection/protos/*.proto --python_out=.
```

This creates a name_pb2.py file from every name.proto file in the \object_detection\protos folder.

Now run `setup.py` under \models\research directory:

```
python setup.py build  
python setup.py install
```

6. Gather and label images

As we have setup Tensorflow object detection API, the next step is gathering training images and labelling them. To make good detection classifier tensorflow need hundreds of images in various background, lighting condition etc.

For this card detector, we are having 53 classes(52+ joker) and I have used 40–50 images of each class. I have collected images from [here](#) and google images.

After collecting the pictures, we need to move 20% of them to the \object_detection\images\test directory, and 80% of them to the \object_detection\images\train directory.

For image labelling, we can use LabelImg. LabelImg saves a .xml file containing the label data for each image. These .xml files will be used to generate TFRecords, which are one of the inputs to the TensorFlow trainer. Once you have labeled and saved each image, there will be one .xml file for each image in the \test and \train directories.

7. Generating training data

Once the labelling completes, next step is generating the TFRecords that serve as input data to the TensorFlow training model.

First we need to convert the .xml files to csvs containing all the data for the train and test images. From the \object_detection folder, issue the following command in the Anaconda command prompt:

```
python xml_to_csv.py
```

This will create train_labels.csv and test_labels.csv files.

Next we need to add the classifiers in generate_tfrecord.py. We'll change `def class_text_to_int(row_label)`

```
def class_text_to_int(row_label):
    if row_label == '1d':
        return 1
    elif row_label == '2d':
        return 2
    elif row_label == '3d':
        return 3
    ...//all other conditions
    else:
        return None
```

Then, generate the TFRecord files by issuing these commands from the \object_detection folder:

```
python generate_tfrecord.py --csv_input=images/train_labels.csv --
image_dir=images/train --output_path=train.record
python generate_tfrecord.py --csv_input=images/test_labels.csv --
image_dir=images/test --output_path=test.record
```

These generate a train.record and a test.record file in \object_detection and will be used to train the card detection classifier.

8. Training configuration

We need to create a label map and edit the training configuration file before training.

The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. Use a text editor to create a new file and save it as labelmap.pbtxt in the \object_detection\training folder. The content should be,

```

item {
  id: 1
  name: '1d'
}

item {
  id: 2
  name: '2d'
}

// all 53 cases

```

Next, we need to define which model and what parameters will be used for training. Navigate to `object_detection\samples\configs` and copy the `faster_rcnn_inception_v2_pets.config` file into the `\object_detection\training` directory. Then, open the file with a text editor. There are several changes to make to the .config file, mainly changing the number of classes and examples, and adding the file paths to the training data.

- Change `num_classes` to 53
- Change `fine_tune_checkpoint` to: `"object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.cpkt"` path
- In the `train_input_reader` section, change `input_path` and `label_map_path` to: `input_path : "/object_detection/train.record"`, `label_map_path: "/object_detection/training/labelmap.pbtxt"`
- Change `num_examples` to the number of images you have in the `\images\test` directory.
- In the `eval_input_reader` section, change `input_path` and `label_map_path` to: `input_path : "/object_detection/test.record"`, `label_map_path: "/object_detection/training/labelmap.pbtxt"`

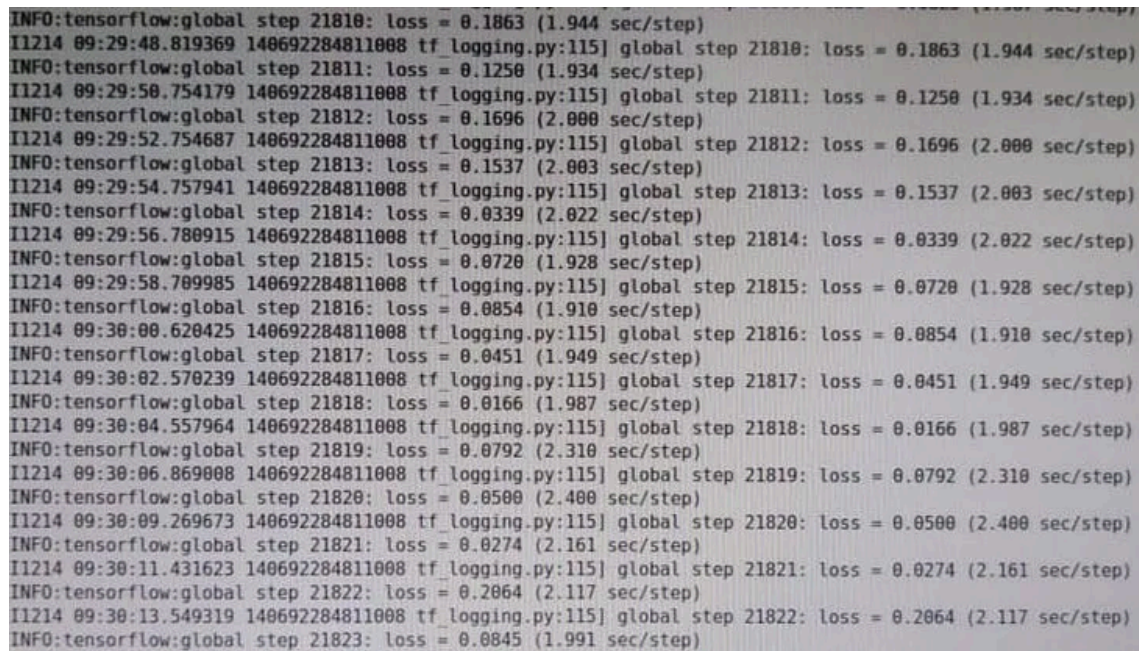
Save the file after the changes have been made. That's it! We are ready to go!

9. Run training

From the \object_detection directory, issue the following command

```
python train.py --logtostderr --train_dir=training/ --  
pipeline_config_path=training/faster_rcnn_inception_v2_pets.config
```

TensorFlow will initialize the training(Fig 3), Each step of training reports the loss. It will start high and get lower and lower as training progresses. I recommend allowing your model to train until the loss consistently drops below 0.05, which will take about 40,000 steps.



```
INFO:tensorflow:global step 21810: loss = 0.1863 (1.944 sec/step)  
I1214 09:29:48.819369 140692284811008 tf_logging.py:115] global step 21810: loss = 0.1863 (1.944 sec/step)  
INFO:tensorflow:global step 21811: loss = 0.1250 (1.934 sec/step)  
I1214 09:29:50.754179 140692284811008 tf_logging.py:115] global step 21811: loss = 0.1250 (1.934 sec/step)  
INFO:tensorflow:global step 21812: loss = 0.1696 (2.000 sec/step)  
I1214 09:29:52.754687 140692284811008 tf_logging.py:115] global step 21812: loss = 0.1696 (2.000 sec/step)  
INFO:tensorflow:global step 21813: loss = 0.1537 (2.003 sec/step)  
I1214 09:29:54.757941 140692284811008 tf_logging.py:115] global step 21813: loss = 0.1537 (2.003 sec/step)  
INFO:tensorflow:global step 21814: loss = 0.0339 (2.022 sec/step)  
I1214 09:29:56.780915 140692284811008 tf_logging.py:115] global step 21814: loss = 0.0339 (2.022 sec/step)  
INFO:tensorflow:global step 21815: loss = 0.0720 (1.928 sec/step)  
I1214 09:29:58.709985 140692284811008 tf_logging.py:115] global step 21815: loss = 0.0720 (1.928 sec/step)  
INFO:tensorflow:global step 21816: loss = 0.0854 (1.910 sec/step)  
I1214 09:30:00.620425 140692284811008 tf_logging.py:115] global step 21816: loss = 0.0854 (1.910 sec/step)  
INFO:tensorflow:global step 21817: loss = 0.0451 (1.949 sec/step)  
I1214 09:30:02.570239 140692284811008 tf_logging.py:115] global step 21817: loss = 0.0451 (1.949 sec/step)  
INFO:tensorflow:global step 21818: loss = 0.0166 (1.987 sec/step)  
I1214 09:30:04.557964 140692284811008 tf_logging.py:115] global step 21818: loss = 0.0166 (1.987 sec/step)  
INFO:tensorflow:global step 21819: loss = 0.0792 (2.310 sec/step)  
I1214 09:30:06.869008 140692284811008 tf_logging.py:115] global step 21819: loss = 0.0792 (2.310 sec/step)  
INFO:tensorflow:global step 21820: loss = 0.0500 (2.400 sec/step)  
I1214 09:30:09.269673 140692284811008 tf_logging.py:115] global step 21820: loss = 0.0500 (2.400 sec/step)  
INFO:tensorflow:global step 21821: loss = 0.0274 (2.161 sec/step)  
I1214 09:30:11.431623 140692284811008 tf_logging.py:115] global step 21821: loss = 0.0274 (2.161 sec/step)  
INFO:tensorflow:global step 21822: loss = 0.2064 (2.117 sec/step)  
I1214 09:30:13.549319 140692284811008 tf_logging.py:115] global step 21822: loss = 0.2064 (2.117 sec/step)  
INFO:tensorflow:global step 21823: loss = 0.0845 (1.991 sec/step)
```

Fig 3.

10. Time to test our card detector

Once training completes, we need to generate the frozen inference graph (.pb file). From the \object_detection folder, issue

```
python export_inference_graph.py --input_type image_tensor --  
pipeline_config_path training/faster_rcnn_inception_v2_pets.config --  
trained_checkpoint_prefix training/model.ckpt-XXXX --output_directory  
inference_graph
```

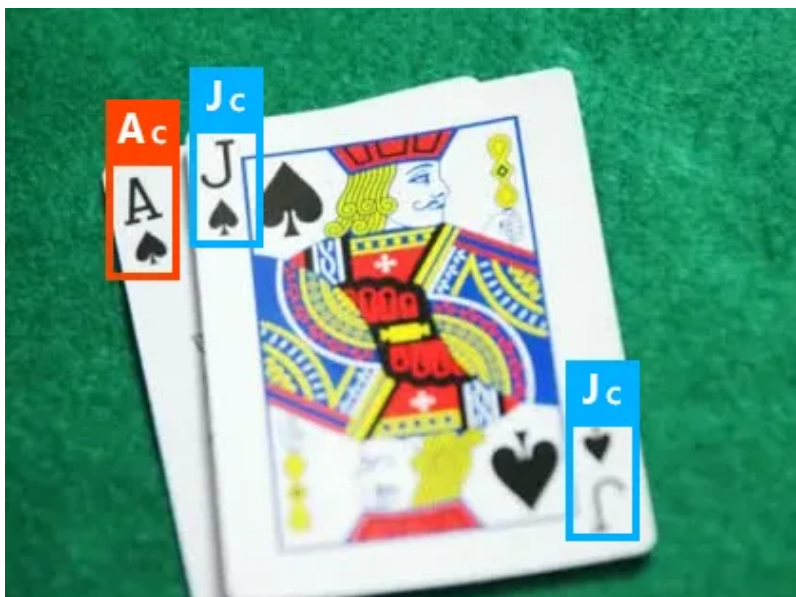
XXXX is the highest number in that folder. This creates a frozen_inference_graph.pb file in the \object_detection\inference_graph folder. The .pb file contains the object detection classifier.

Before running the Python scripts, we need to modify the NUM_CLASSES as 53 and issue the command.

```
python Object_detection_image.py
```

Thats it!

This will open the image that we have fed as input and bounding boxes will be there around the card as in Fig.



Conclusion

In this blog we have learnt to develop a card detector using Tensorflows Object detection api and pre-trained classifier Faster-RCNN-Inception-V2-COCO .



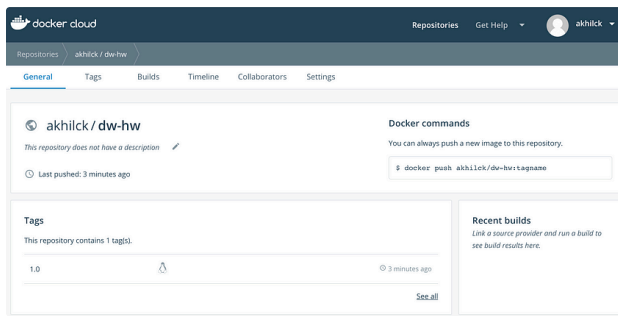
Written by Akhil C K

4 Followers · 2 Following

Programmer

Follow

More from Akhil C K



Akhil C K

Deploying dropwizard java application with Kubernetes and...

In this article we are going to learn how we can create a simple dropwizard application i...

Jul 3, 2021



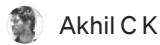
Akhil C K

Python Scripts : Start/Stop AWS EC2

May 11, 2021



```
ok-Probot akhilck make train-nlu
train -c nlu_config.yml --data data/nlu_data.md --o models --fixed_model_name nlu --project current --verbose
1 INFO rasa_nlu.utils.spacy_utils - Trying to load spacy model with name 'en'
2 INFO rasa_nlu.components - Added 'nlp_spacy' to component cache. Key 'nlp_spacy-en'.
3 INFO rasa_nlu.training_data.loading - Training data format of data/nlu_data.md is md
4 INFO rasa_nlu.training_data.training_data - Training data stats:
samples: 76 (6 distinct intents)
intents: 'name', 'affirm', 'joke', 'greet', 'thanks', 'goodbye'
entities: 19 (1 distinct entities)
entities: 'name'
5 INFO rasa_nlu.model - Starting to train component nlp_spacy
6 INFO rasa_nlu.model - Finished training component.
7 INFO rasa_nlu.model - Starting to train component tokenizer_spacy
8 INFO rasa_nlu.model - Finished training component.
9 INFO rasa_nlu.model - Starting to train component intent_featurizer_spacy
10 INFO rasa_nlu.model - Finished training component.
11 INFO rasa_nlu.model - Starting to train component intent_entity_featurizer_regex
12 INFO rasa_nlu.model - Finished training component.
13 INFO rasa_nlu.model - Starting to train component ner_crf
14 INFO rasa_nlu.model - Finished training component.
15 INFO rasa_nlu.model - Starting to train component ner_synonyms
16 INFO rasa_nlu.model - Finished training component.
17 INFO rasa_nlu.model - Starting to train component intent_classifier_sklearn
18 INFO rasa_nlu.model - Finished training component.
19 INFO Using backend SequentialBackend with 1 concurrent workers.
20 INFO Done 12 out of 12 | elapsed: 0.25 finished
21 INFO rasa_nlu.model - Finished training component.
22 INFO rasa_nlu.model - Successfully saved model into '/Users/akhilck/Documents/Akhil/Projects/rasa/movesbot/models/c
23 INFO _main_ - Finished training
```

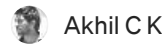


Akhil C K

Building a bot using Rasa NLU & Core

How about building your own assistant which

twitter_id1	1					
twitter_id2	2					
twitter_id3	3					
twitter_id4	4					
twitter_id5	5					
twitter_id6	6					
twitter_id7	7					
twitter_id8	8					
twitter_id9	9					
twitter_id10	10					
...	...					
twitter_id2995	2995					
twitter_id2996	2996					



Akhil C K

Build an Influencer score model using Linear Regression

Influencer is the one who can make an impact

Open in app

Sign up

Sign in

Medium



Search



Write



See all from Akhil C K