“Steps for Custom Training on object Detection” (RCNN)

1. Install TensorFlow-GPU 1.5 (skip this step if TensorFlow-GPU 1.5 is already installed)

pip install --upgrade tensorflow-gpu

command will automatically download version 1.5
Download and install CUDA v9.0
Be sure to install Anaconda with Python 3.6

2. Set up TensorFlow Directory and Anaconda Virtual Environment

The TensorFlow Object Detection API requires using the specific directory structure provided in its GitHub repository. It also requires several additional Python packages, specific additions to the PATH and PYTHONPATH variables, and a few extra setup commands to get everything set up to run or train an object detection model.

2a. Download TensorFlow Object Detection API repository from GitHub

Create a folder and name it “tensorflow”. This working directory will contain the full TensorFlow object detection framework, as well as your training images, training data, trained classifier, configuration files, and everything else needed for the object detection classifier.
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.
Open the downloaded zip file and extract the “models-master” folder directly into the “tensorflow” directory you just created. Rename “models-master” to just “models”.

2b. Download the Faster-RCNN-Inception-V2-COCO model from TensorFlow's model zoo

TensorFlow provides several object detection models (pre-trained classifiers with specific neural network architectures) in its model zoo. Some models (such as the SSD-MobileNet model) have an architecture that allows for faster detection but with less accuracy, while some models (such as the Faster-RCNN model) give slower detection but with more accuracy.
I initially started with the SSD-MobileNet-V1 model, but it didn’t do a very good job identifying the cards in my images. I re-trained my detector on the

Faster-RCNN-Inception-V2 model, and the detection worked considerably better, but with a noticeably slower speed.
You can choose which model to train your objection detection classifier on. If you are planning on using the object detector on a device with low computational power (such as a smart phone or Raspberry Pi), use the SDD-MobileNet model. If you will be running your detector on a decently powered laptop or desktop PC, use one of the RCNN models.
Download the model:
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Open the downloaded faster_rcnn_inception_v2_coco.tar.gz file with a file archive such as WinZip or 7-Zip and extract the faster_rcnn_inception_v2_coco_2018_01_28 folder to the tensorflow\models\research\object_detection folder.

2c. Download other repository from GitHub

https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10
extract all the contents directly into the tensorflow1\models\research\object_detection directory.
This repository contains the images, annotation data, .csv files, and TFRecords needed to train samples playing cards.
It also contains Python scripts that are used to generate the training data.
If you want to train your own object detector, delete the following files (do not delete the folders):

- All files in \object_detection\images\train and \object_detection\images\test
- The “test_labels.csv” and “train_labels.csv” files in \object_detection\images
- All files in \object_detection\training
- All files in \object_detection\inference_graph

2d. Set up new Anaconda virtual environment

conda create -n tensorflow1 pip python=3.5
Then, activate the environment by issuing:
activate tensorflow1
Install tensorflow-gpu in this environment by issuing:

```
(tensorflow1) > pip install --ignore-installed --upgrade tensorflow-gpu
```
Install the other necessary packages by issuing the following commands:
```
conda install -c anaconda protobuf
pip install pillow
pip install lxml
pip install Cython
pip install jupyter
pip install matplotlib
pip install pandas
pip install opencv-python
```

2e. Configure PYTHONPATH environment variable

A PYTHONPATH variable must be created that points to the \models, \models\research, and \models\research\slim directories.
```
set
PYTHONPATH=C:\tensorflow\models;C:\tensorflow1\models\research;C:\tensorflow1\models\research\slim
```

2f. Compile Protobufs and run setup.py

Next, compile the Protobuf files, which are used by TensorFlow to configure model and training parameters. Unfortunately, the short protoc compilation command posted on TensorFlow's Object Detection API
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md
```
# From tensorflow/models/research/
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.
Finally, run the following commands from the tensorflow\models\research directory:
```
python setup.py build
python setup.py install
```

2g. Test TensorFlow setup to verify it works

You can test it out and verify your installation is working by launching the object_detection_tutorial.ipynb script with Jupyter. From the \object_detection directory, issue this command:
```
(tensorflow1)tensorflow1\models\research\object_detection> jupyter notebook object_detection_tutorial.ipynb
```

3. Gather and Label Pictures

3a. Gather Pictures

TensorFlow needs hundreds of images of an object to train a good detection classifier. To train a robust classifier, the training images should have random objects in the image along with the desired objects, and should have a variety of backgrounds and lighting conditions.

Make sure the images aren't too large. They should be less than 200KB each, and their resolution shouldn't be more than 720x1280. The larger the images are, the longer it will take to train the classifier. You can use the resizer.py script in this repository to reduce the size of the images.

After you have all the pictures you need, move 20% of them to the \object_detection\images\test directory, and 80% of them to the \object_detection\images\train directory. Make sure there are a variety of pictures in both the \test and \train directories.

3b. Label Pictures

LabelImg is a great tool for labeling images.
https://github.com/tzutalin/labelImg

Download and install LabelImg, point it to your \images\train directory, and then draw a box around each object in each image. Repeat the process for all the images in the \images\test directory.

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.

Also, you can check if the size of each bounding box is correct by running sizeChecker.py

```
C:\tensorflow1\models\research\object_detection> python sizeChecker.py --move
```

4. Generate Training Data

With the images labeled, it's time to generate the TFRecords that serve as input data to the TensorFlow training model.

First, the image .xml data will be used to create .csv files containing all the data for the train and test images. From the \object_detection folder.

```
tensorflow\models\research\object_detection> python xml_to_csv.py
```

This creates a train_labels.csv and test_labels.csv file in the \object_detection\images folder.
Next, open the generate_tfrecord.py file in a text editor.Replace the label map starting at line 31 with your own label map, where each object is assigned an ID number.
EX:

```
# TO-DO replace this with label map
def class_text_to_int(row_label):

  if row_label == 'jack':
      return 1
   elif row_label == 'queen':
      return 2
   elif row_label == 'king':
      return 3
   elif row_label == 'ace':
      return 4
   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. These will be used to train the new object detection classifier.

5. Create Label Map and Configure Training

The last thing to do before training is to create a label map and edit the training configuration file.

5a. Label map

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 C:\tensorflow1\models\research\object_detection\training folder.
In the text editor, copy or type in the label map in the format below.
EX:

```
item {
  id: 1
  name: 'jack'
}

item {
  id: 2
  name: 'queen'
}

item {
  id: 3
  name: 'king'
}

item {
  id: 4
  name: 'ace'
}
```

The label map ID numbers should be the same as what is defined in the generate_tfrecord.py file.

5b. Configure training

Finally, the object detection training pipeline must be configured. It defines which model and what parameters will be used for training. This is the last step before running training!

Navigate to C:\tensorflow1\models\research\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.

Line 9. Change num_classes to the number of different objects you want the classifier to detect. For the above jack, queen,king and ace detector, it would be num_classes : 4

Line 110. Change fine_tune_checkpoint to:

fine_tune_checkpoint : "/tensorflow/models/research/object_detection/faster_rcnn_inception_v2_coco/model.ckpt"

Lines 126 and 128. In the train_input_reader section, change input_path and label_map_path to:

- input_path : "/tensorflow/models/research/object_detection/train.record"

- label_map_path:"/tensorflow/models/research/object_detection/training/labelmap.pbtxt"

Line 132. Change num_examples to the number of images you have in the \images\test directory.
Lines 140 and 142. In the eval_input_reader section, change input_path and label_map_path to:

- input_path : "/tensorflow/models/research/object_detection/test.record"

- label_map_path:"/tensorflow/models/research/object_detection/training/labelmap.pbtxt"

Save the file after the changes have been made. That's it! The training job is all configured and ready to go!

6. Run the Training

the train.py file is still available in the /object_detection/legacy folder. Simply move train.py from /object_detection/legacy into the /object_detection folder and then continue following the steps below.

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

TensorFlow will initialize the training.
Each step of training reports the loss. It will start high and get lower and lower as training progresses. For my training on the Faster-RCNN-Inception-V2 model, it started at about 3.0 and quickly dropped below 0.8. I recommend allowing your model to train until the loss consistently drops below 0.05, which will take about 40,000 steps, or about 2 hours (depending on how powerful your CPU and GPU are). Note: The loss numbers will be different if a different model is used. MobileNet-SSD starts with a loss of about 20, and should be trained until the loss is consistently under 2.
You can view the progress of the training job by using TensorBoard.
change to the C:\tensorflow1\models\research\object_detection directory, and issue the following command:

```
(tensorflow1) C:\tensorflow\models\research\object_detection>tensorboard –logdir=training
```

This will create a webpage on your local machine at YourPCName:6006 which can be viewed through a web browser.

The TensorBoard page provides information and graphs that show how the training is progressing. One important graph is the Loss graph, which shows the overall loss of the classifier over time.

7. Export Inference Graph

Now that training is complete, the last step is to generate the frozen inference graph (.pb file). From the \object_detection folder
where “XXXX” in “model.ckpt-XXXX” should be replaced with the highest-numbered .ckpt file in the training folder:

```
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
```

This creates a frozen_inference_graph.pb file in the \object_detection\inference_graph folder. The .pb file contains the object detection classifier.

8. Use Your Newly Trained Object Detection Classifier!

Before running the Python scripts, you need to modify the NUM_CLASSES variable in the script to equal the number of classes you want to detect.