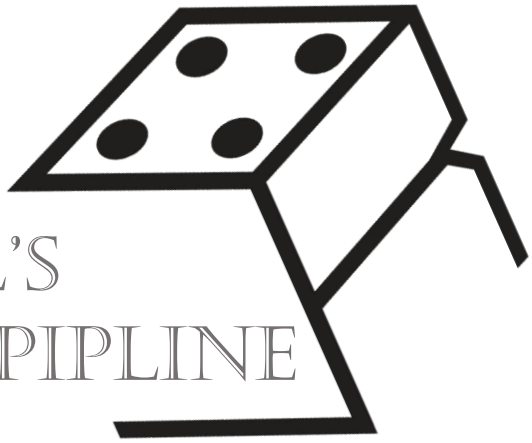
Welcome to

DIGITAL ROLL'S OBJECT DETECTION PIPLINE



Tyler Boice Caleb Johnson Tyler Malmon Brandon Measley

Table of Contents:

1) Description

This repo will walk you through the process of creating an object detection classifier for mobile applications. The repo features three main features:

1) A data collection application (ios only) that can take images and classification data for training the model

2) A workbench that trains the model and produces a TensorFlow and a CoreML model

3) An application (ios only) that can display models created from the workbench

Currently the data application app is set to detect polyhedral dice; however, it can be repurposed for any custom classifier. This tutorial we will classifying polyhedral dice of sides 4, 6, 8, 10, 12, and 20. For this tutorial we will assume you have basics of directories and have some command line interface experience. For more information about this project and our team, visit our website.

2) Data Collection Application

To train an AI classifier, you will first need to gather data. For object detection you will need to collect images of the objects you want to classify and label them. It is recommended that you have 1000 labeled objects per classifier to create a highly accurate model. There are two ways you can label this data.

1) **Mobile Data Collection Application**: If you don't have any images already taken, we have created an ios mobile application that assist in the process of collecting data. We recommend this method because it displays the orientation of your phone and adds that data along with the labelling data.

2) **Labelling Application:** if you already have the images and just need to label them you can use the labelling application. This application has been placed in Digital Roll's repo but originates from tzutalin's repo. This application allows you to label any image with custom classifiers.

Mobile Data Collection Application

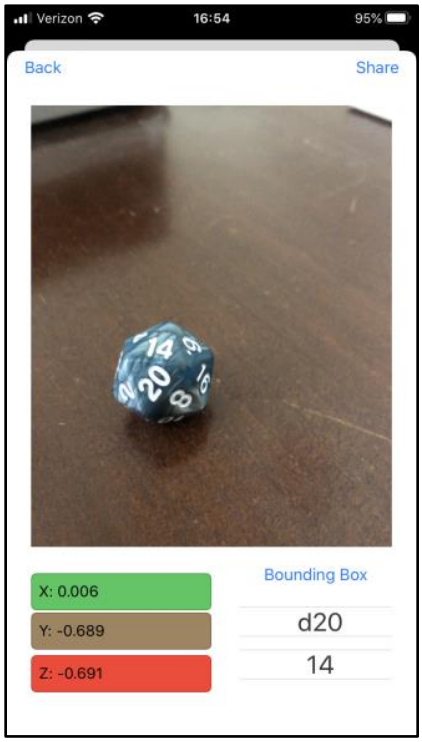


Figure 1: Mobile Data Collection Application for ios

The mobile data collection application is an ios application that streamline the data collection process. Figure 1 is an example of the application. In the bottom right is the orientation data of your phone. You want to get photos of a consistent angle to have a more

accurate model. On the bottom right is the object labels. Select the type of dice you are taking a picture of and top value of the dice. Drag your finger across the image to draw a bounding box around the dice.

Once you have taken all the pictures that you desire, use the “share” button in the top right to transfer the data to your computer for training. Each image and its data are embedded in a single *.xml* file. When you run the workbench, it will pull the image out of the file before it begins training.

Labelling Application

Once you have installed Digital Roll’s repository, the first step is to create an environment. After you have created an environment and installed all the packages needed for the application you can enter the environment and cd into the "labelImg" folder located in the root of the repository. Then run:

```
pyrcc5 -o libs/resources.py resources.qrc
python labelImg.py
```

After these commands are ran, it will open the labelling application. Within the application use the "Open Dir" button located in the left column to open the folder that contains the images you wish to label. To label each image, create a bounding box around and type the name of the classifier. To label polyhedral dice, label the whole dice and use the naming convention *type-value* (e.g. d6-2 for a six-sided dice with the value of two). After all the objects within the image have been labeled and you save the file, it will save as a sperate *.xml* that contains the name of the image it is using.

Ensure you are saving the xml files as the same name as the image. If you change the name of the image after you save the *.xml* file, it will not be able to find the image. Since the value of the image is hardcoded in the *.xml* file, make sure the image names are the ones you want before you begin this process.

Tips The labelling process can be a long and tedious process. To increase the speed and efficiency of this process you can use the following tips:

- You can modify the file *.\labelImg\data\predefined_classes* to ensure it has all the classes you are labelling. This will allow the application to give suggestions, so you don't have to enter the name of the classifier each time.

- Activate auto save mode (View->Auto) which will continuously save so you can move on to the next photo without needed to save every time. **Warning**: this may save to the .*labelImg**path*, make sure you move to the correct directory (.*Tensorflow-Workbench**images*) when done labeling.
- Use the hotkeys. For a more in-depth tutorial check out the original repo or online tutorials.

Hotkey	**Usage**
Ctrl + u	Load all of the images from a directory
Ctrl + r	Change the default annotation target dir
Ctrl + s	Save
Ctrl + d	Copy the current label and rect box
Space	Flag the current image as verified
w	Create a rect box
d	Next image
a	Previous image
del	Delete the selected rect box
Ctrl++	Zoom in
Ctrl--	Zoom out
↑→↓←	Keyboard arrows to move selected rect box

3) TensorFlow Workbench

This is a modified version of the repo. This repo is the basis for a workbench that will aid users in training and validating their own data and converting the models produced into Apple CoreML to be used on apple mobile devices. It is highly recommended that you use a GPU. Since you are training a model from scratch, the more images and classifiers you have, the longer the training process takes. Make sure your system has a valid GPU before proceeding.

Required (GPU Only)

- **CUDA 10.1** - The drivers needed to run TensorFlow GPU. Currently 10.1 is the most stable version that supports TensorFlow 2.0. Use the exe (local) installer type
- **CUDNN 7.6.4** - The additional files needed for TensorFlow GPU. You will need to create a NVIDIA Developer account. CUDNN 7.6.4 is the most stable version for CUDA 10.1.
- **Visual Studio 2019 (Community)** - Needed for TensorFlow GPU with a windows machine
- At minimum 60GB of free space on your machine

Installation (GPU Only)

1. Install the proper drivers for your graphics card
2. Install CUDA 10.1 update2

   a) Click OK on the first setup prompt

   b) Click Agree and Continue when prompted about the license agreement

   c) Select the Express installation

   d) Complete final installation steps

3. Download CUDNN 7.6.4

   a) Open the CUDA path: *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1*

   b) Open the CUDNN installer (you may need 7-zip to unzip the file). When open it will contain one folder named cuda.

c) Drag all the cuda folder from the CUDNN folder to: *C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1* like in Figure 2. Then click Replace all items if prompted.

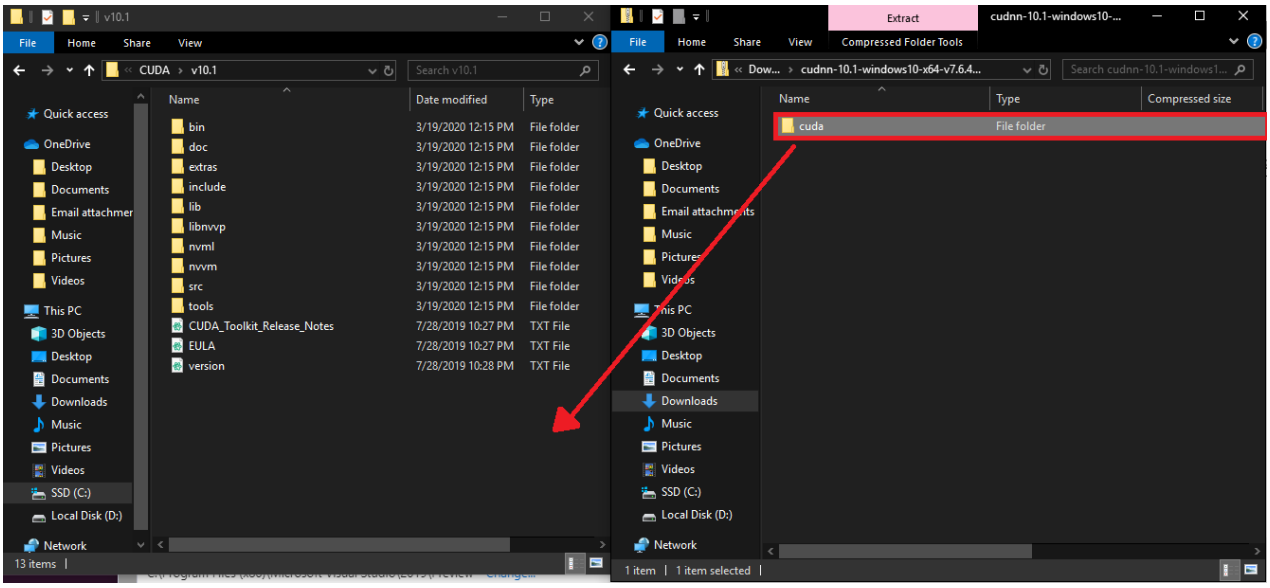


Figure 2: Drag cuda file to CUDA\10.1

4. Install Visual Studio 2019 (Community)

a) Run the installer

b) Select the C++ development workload like in Figure 3 and finish downloading visual studio

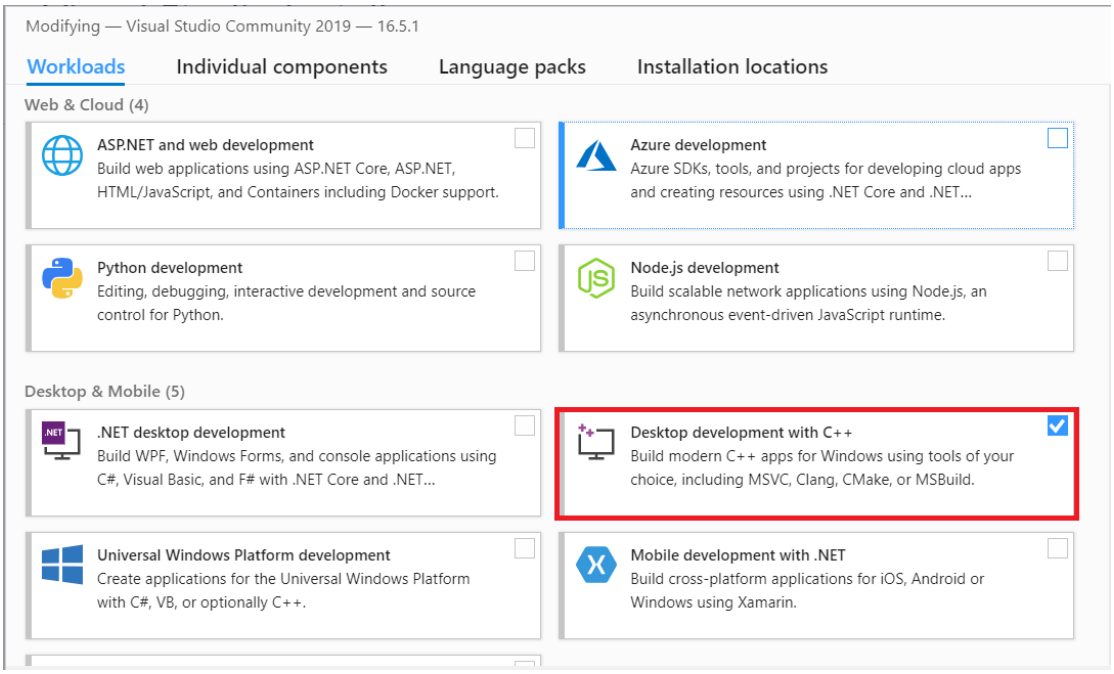


Figure 3: Visual Studio 2019 (community) C++ installation

5. Set up environment variables:

a) Click on start and type "*environment variables*". Select "*Edit the system environment variables*" like in Figure 4

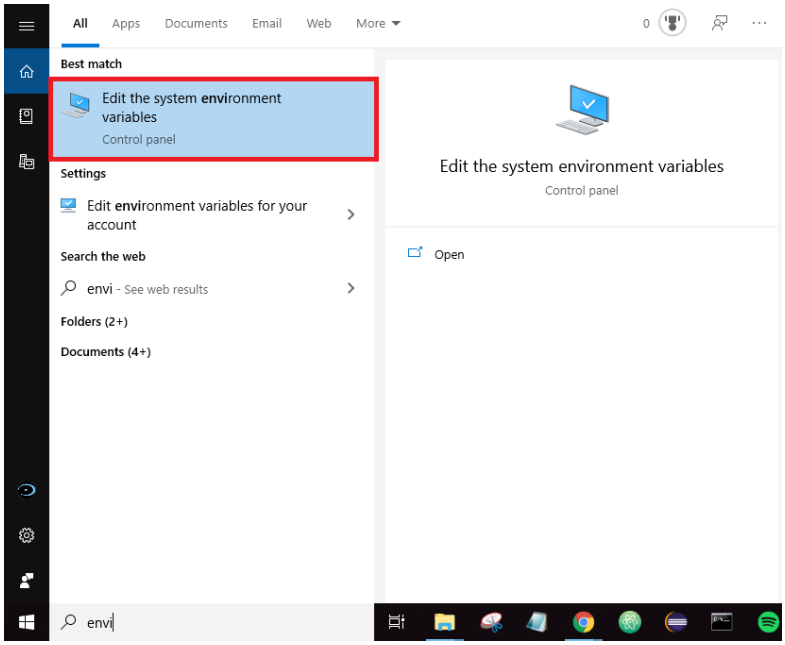


Figure 4: Click on Edit the system variables

b) Click on "environment variables", as seen in Figure 5

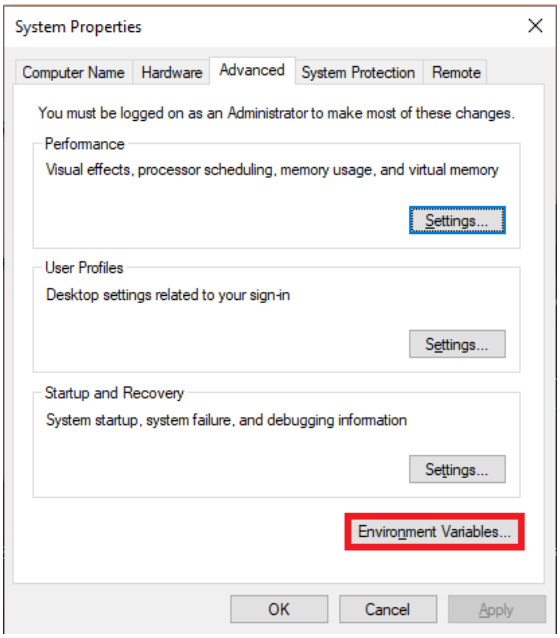


Figure 5: Click on environment Variables

c) In the system path section, double click on path, as seen in Figure 6:

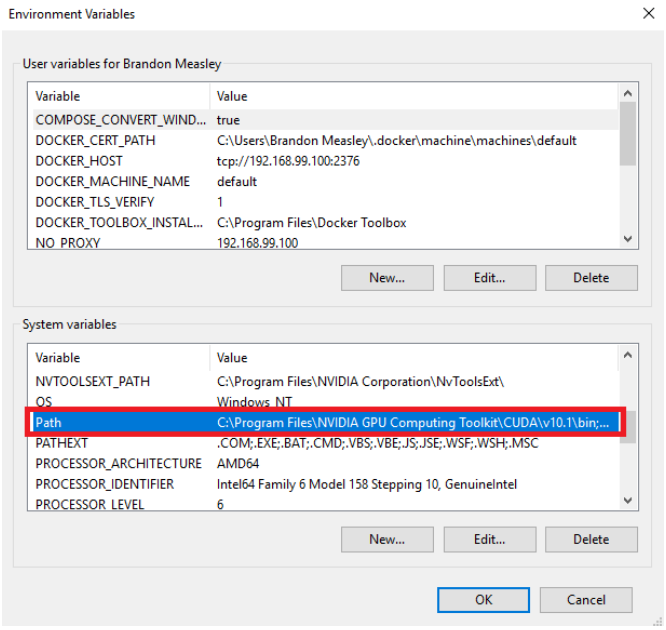


Figure 6: Double Click on Path

d) Enter variables below manually by clicking "new". **Make sure you select "ok" when done or the variables will not save**. Close the window and reopen to ensure the changes were made like in Figure 7. The order of the variables does not matter.

```
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\bin

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\libnvvp

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.1\extras\CUPTI\libx64
```

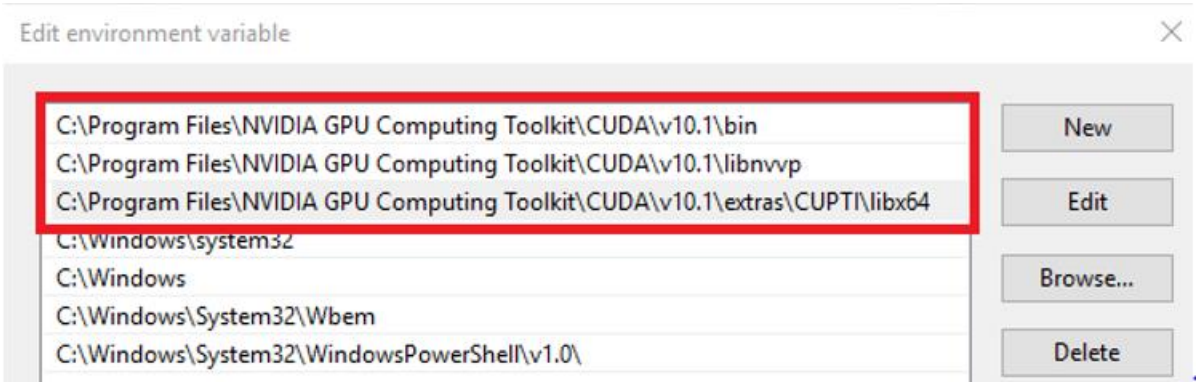
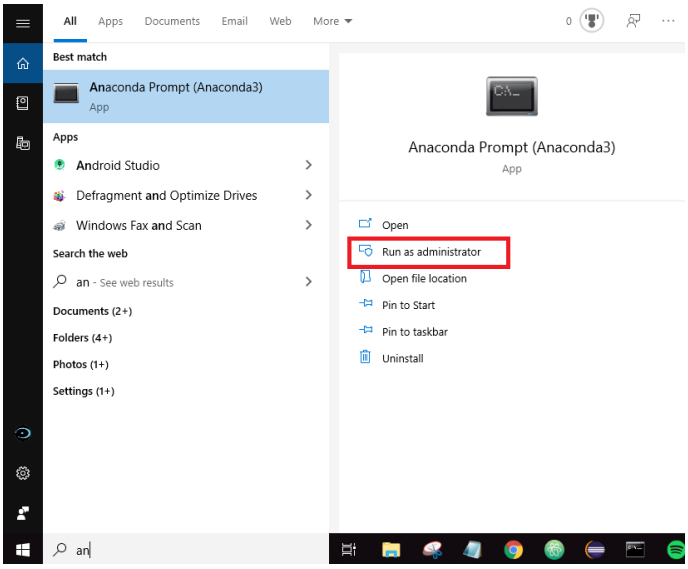


Figure 7: Edit Environment Variable

Set up

Install Anaconda

1. Download Anaconda (Python 3.7 version) and run the installer.
2. Select the "*Just Me*" option, however if the file-path contains a space, you may need to select "*All Users*" instead. Sometimes Anaconda can't have a directory with a space in it.
3. Select where you want Anaconda to be installed and click "*next*"
4. Don't Select either of the Advanced Options and click "*install*"
5. After installation, type anaconda in the search bar and open anaconda as admin



Create Anaconda Environment

cd into the "Tenorflow-Workbench" directory

For TensorFlow GPU (RECOMMENDED)

```
conda env create -f conda-gpu.yml
conda activate gpu
pip install -r requirements-gpu.txt
```

For TensorFlow CPU

```
conda env create -f conda-cpu.yml
conda activate cpu
pip install -r requirements-cpu.txt
```

Usage

Before you start the workbench, ensure the following:

- All your images are labelled
- All data collected is placed in the *./Tensorflow-Workbench/images* folder:
    - If the labelling application was used it produces both image and .xml files
    - If the mobile ios data collection app was used it will produce just .xml
- You are in correct conda environment
    - To ensure this, look at the word before your current working directory. By default, it says "(base)", it needs to say "(cpu)" or "(gpu)" depending on the enviornment
    - To enter an environment, run:

```
#gpu (recommended)
conda activate gpu

#cpu
conda activate cpu
```

Starting the Workbench:

```
# from the Tensorflow-Workbench directory
python run_workbench.py
```

Once this command is running, it sets all the preferences to default. There are many preferences that can be set for the workbench. See the preferences section of this document for a list of all the preferences and the benchmark section to see how the preferences affect the time it takes to train, and the accuracy of the models produced.

Configuring the Workbench:

Help

If you ever need help with knowing how to run the workbench, you can always use the **help(h)** command for assistance. When this command is ran, it will display all the command that the workbench takes as input.

Examples:

```
# display help menu
help
```

Modify Preferences

To set preferences, use the **modify(m)** command followed by a space, then the name of the variable you wish to change followed by another space and the value.

Examples:

```
#Modify batch size to 10
m batch_size 10

#modify output to desktop
modify output C:\Users\JohnDoe\Desktop
```

Warning: Every time you leave the workbench all preferences will be lost; please save any preferences you wish to keep and load them when running the workbench.

Load Preferences

The workbench preference can be loaded from a *.txt* file that contains the variable names followed by a semicolon and the value.

Examples from text file:

```
batch_size: 10
output : C:\Users\JohnDoe\Desktop
```

To load preferences, use the **load(l)** command followed by a space and the name of the .txt file and path that contains your preferences. If just the file was given without a path, it will check the *Tensorflow-Workbench* directory.

Examples:

```
# Loading file from Tensorflow-Workbench directory
l preferences.txt

# Loading from custom file path
load  C:\Users\JohnDoe\Desktop\pref_files\pref_1.txt
```

Save Preferences

To save preferences, use the save preferences, use the **save(s)** command. Using the command without anything followed will save it as a *preference.txt* file in the *Tensorflow-Workbench* directory. You can also use the command followed by a space and a file path and file name. If just a file path is given it will still save as a *preference.txt*

Examples:

```
# Saving file as preferences.txt to Tensorflow-Workbench directory
save

# Saving to custom file path and filename
s C:\Users\JohnDoe\Desktop\pref_files\pref_1.txt
```

Display Current Preferences

To display your current preferences, use the save preferences, use the **display(d)** command. This will display a full list of all the current preferences and their values

Examples:

```
# Display command
display
```

Get Info on preferences

There are many preferences in the workbench that you can modify. To view them you can go to the preferences section of this document or use the **info(i)** command in the workbench for a brief overview of each preference.

Examples:

```
# Display brief summary of each preference
info
```

Running the Workbench:

Start training from scratch

To run the workbench and train a model use the **run(r)**. This will count all the objects you labeled, display them, and create a *classifier.names*. Then it will organize your data into three folders within the *Tensorflow-Workbench/images* folder: Test, Train and Validate. file. The testing and training will be used to create a *.tfrecord* files, used for training the model. The *.names* and *.tfrecord* files will be stored in the *Tensorflow-Workbench/data* folder. Before the workbench begins training, will output your GPU information if you are GPU boosted. We highly recommend you use a GPU or it may take days to train. After the GPU information is displayed, the workbench will begin to train a model. The output looks like Figure 8 and the part to focus on is the *loss:* category. This displays the overall loss rate of the model being trained. Initially it will be high but as it trains it will get decrease. A truly trained model will have a loss rate below 1 but you will begin to see the model predict accurately around 2. For every epoch the workbench saves a checkpoint in the format of three files: a *.tf.index*, a *.tf.data.data-00000-of-00002*, and a *.tf.data.data-00001-of-00002*. To stop training early, use the ctl + C.

```
 1/Unknown - 14s 14s/step - loss: 2434.1301 - yolo_output_0_loss: 162.9299 - yolo_output_1_loss: 540.5997 - yolo_output_2_loss: 1718.9543
 2/Unknown - 15s 7s/step - loss: 2934.4340 - yolo_output_0_loss: 239.9029 - yolo_output_1_loss: 750.5129 - yolo_output_2_loss: 1932.3573
 3/Unknown - 15s 5s/step - loss: 2750.6104 - yolo_output_0_loss: 233.4875 - yolo_output_1_loss: 804.4748 - yolo_output_2_loss: 1700.9696
 4/Unknown - 15s 4s/step - loss: 2481.6126 - yolo_output_0_loss: 196.6682 - yolo_output_1_loss: 755.8445 - yolo_output_2_loss: 1517.4009
 5/Unknown - 15s 3s/step - loss: 2219.6410 - yolo_output_0_loss: 170.5287 - yolo_output_1_loss: 683.9294 - yolo_output_2_loss: 1353.4614
 6/Unknown - 15s 3s/step - loss: 2001.5001 - yolo_output_0_loss: 155.0971 - yolo_output_1_loss: 615.3350 - yolo_output_2_loss: 1219.3225
 7/Unknown - 15s 2s/step - loss: 1821.5730 - yolo_output_0_loss: 137.4511 - yolo_output_1_loss: 563.3542 - yolo_output_2_loss: 1108.9974
 8/Unknown - 16s 2s/step - loss: 1671.4554 - yolo_output_0_loss: 123.2357 - yolo_output_1_loss: 520.4500 - yolo_output_2_loss: 1015.9744
 9/Unknown - 16s 2s/step - loss: 1547.1955 - yolo_output_0_loss: 111.6764 - yolo_output_1_loss: 484.3461 - yolo_output_2_loss: 939.3524
10/Unknown - 16s 2s/step - loss: 1440.1364 - yolo_output_0_loss: 102.2722 - yolo_output_1_loss: 453.4458 - yolo_output_2_loss: 872.5729
```

Figure 8: Model training in Workbench

Once the training is complete, the workbench will create a *.pb* (TensorFlow model) and a *.coreml* (Apple CoreML model) and save it to your output folder. Lastly, the workbench will test your validate images against the model you trained and print the predicted dice type, value, and prediction like seen in Figure 9. All files will be saved to your output folder

Examples:

```
# running the workbench
run
```

Figure 9: Model trained against validate image

Continue training

If you stopped training early or if you wish to create a model and test a model from your output folder you can run **continue(c)** command. This will start the workbench from the moment after training and use the most recent checkpoint in your output folder.

Additionally, if you wanted to continue training from the start of a checkpoint you can run the **continue(c)** followed by a space and the folder the checkpoint is in or the checkpoint name. This will start the workbench at the training process using that checkpoint as a starting place.

Note: A checkpoint consists of the three files *.tf.index*, a *.tf.data.data-00000-of-00002*, and a *.tf.data.data-00001-of-00002*. Make sure all these filse are in the folder and use the *.tf* format to refer to a checkpoint. To make it easier just refer to the folder and the workbench will grab the most recent checkpoint

Examples of continuing:

```
# to continue with workbench from after training
continue

# to continue training from checkpoint
c C:\Users\JohnDoe\Desktop\output
# In your folder you have:
#   yolov3_train_1.tf.data-00000-of-00002
#   yolov3_train_1.tf.data-00001-of-00002
#   yolov3_train_1.tf.index
c C:\Users\JohnDoe\Desktop\output\yolov3_train_1.tf
```

Finish Workbench

If you stopped training early or if you wish to run the continue workbench after the training process with your current checkpoints use the **finish(f)** command. This will start the workbench from the moment after training and use the most recent checkpoint in your output folder.

Testing Models on images

If you wanted to train you model on more images, you could use the **test(t)** command followed by the folder or image you want to test. If ran on a folder, the workbench will test against all images within the folder. This will produce images like in Figure 9 and output it to your output folder.

Examples:

```
# to test and image
t /Users/johndoe/Desktop/images/image_test.jpg

# to test a folder of images
test /Users/johndoe/Desktop/images
```

Graph Training

If you want to display a graph that shows the training and testing loss rate, use the **graph(g)** command after training is complete.

Example:

```
# to graph recent training and test loss
graph
```

Quitting the workbench

Finally, to exit the workbench you can use the **quit(q)** command. This will return you back to the terminal. If you enter the workbench all your current preferences will be lost so be sure to save a preference file if necessary

Examples:

```
# quit the workbench
quit
```

Preferences within the Workbench:

The workbench runs on many preferences which you can modify to create the perfect model for your dataset. Use the **info(i)** command to display a brief summary of the preferences

batch_size – **Integer** – Default: 4

Number of examples that are trained at once. Batch size is depended on hardware. Better harder can support a higher batch size

classifiers – **String** – *.names* file

Number of examples that are trained at once. Batch size is depended on hardware. Better harder can support a higher batch size

dataset_test – **String** – *.tfrecords* file - Default*: ./data/test.tfrecord*

When TensorFlow runs it users two *.tfrecord* files, one for testing and one for training. This is the *.tfrecord* file that is used for testing. The workbench will create this file using the images and *.xml* files in ./images/test folder. This variable will most likely never need to be modified

dataset_train– **String** – *.names* file - Default*: ./data/train.tfrecord*

When TensorFlow runs it users two *.tfrecord* files, one for testing and one for training. This is the *.tfrecord* file that is used for training. The workbench will create this file using the images and *.xml* files in ./images/train folder. This variable will most likely never need to be modified

epochs – **Integer** – Default: 50

an epoch is one iteration through all the images and xml files given to the workbench. The more epochs the more the workbench iterates through the dataset; therefore, the more trained the model will be.

image_size - **Integer** – 224 or 256 or 416 – Default: 224

This workbench uses the yolo method of training. In summary this method divides the image into boxes to predict where classifiers will be. Image size determines how many boxes are created. While 416 will make more boxes, it will also be slower and require more processing power than 256.

max_checkpoints – **Integer** – Default: 15

after every iteration(epoch), the workbench will save a checkpoint. These checkpoints take up a lot of memory. This variable is the amount of checkpoints that the checkpoint will keep before deleting older checkpoints. Therefore if you have 5 max checkpoints then the workbench will never have more than 5 checkpoints saved. On the 6th epoch, the first checkpoint will be deleted.

max_sessions – **Integer** - Default: 5

once the model is finished all the output from the workbench is saved. If the workbench is ran again, then that output is saved to a saved_session folder. This variable determines the amount of sessions that are saved before older checkpoints are deleted. Therefore, if you have 5 max sessions then the workbench will never have more than 5 sessions saved. On the 6th session, the first session will be deleted.

Note: if the name of a saved_session folder is modified, the workbench will not delete that session

mode - **String** - must be 'fit', 'eager_fit' or 'eager_tf' - Default: 'fit'

Determines if the training ends early due to overfitting. This is when a model trains for to long and the loss starts getting worse instead of better

- **fit**: model stops training after loss rate can't go any lower (recommended)
- **eager_fit**: model stops training immediately
- **eager_tf**: model does not stop training till epochs finish (not recommended)

output - **String** - file-path - Default: *./current_session*

folder where all workbench output is placed including:

- saved checkpoints
- models created
- images tested

pref - **String** - *.txt* file – Default: None

a .txt file that contains all the variables. These can be used to as specific preferences for training the workbench, instead of having to edit every variable. Use the load(l) command to load in a preference file. Use the save(s) command to save the current preferences to a file.

sessions - **String** - file-path - Default: ./saved_sessions

once the model is finished all the output from the workbench is saved. If the workbench is ran again, then that output is saved to a saved_sessions folder. This variable should be a file-path where you want the sessions saved

tiny_weights - **Boolean** - True or False – Default: False

Uses tiny weights for a smaller model for mobile use; however, both tiny and non-tiny weights can be used for mobile. To use tiny weights, you must use the YOLOv3-tiny file

transfer - **String** - 'none', 'darknet', no_output', frozen', or 'fine_tune' – Default: 'none'

Type of model you are transferring data from

- **none**: No transfer; Training from scratch (recommended)
- **darknet**: Transferring from darknet model
- **no_output**: Transferring just the ouput
- **frozen**: using a frozen model; transfer and freeze all data
- **fine_tune**: transfer and freeze darknet model

val_img_num - **Integer** – Default: 3

The first time the workbench is ran, it divides the images and their *xml* files into three folders: test, train, and validate. The validate folder is not used in the model. Instead it is used after the model is created to test the accuracy of the model. This variable determines how many images should be removed from the workbench to test on the model

val_image_path - **String** - file-path – Default: *./images/validate*

The first time the workbench is ran, it divides the images and their xml files into three folders: test, train, and validate. The validate folder is not used in the model. Instead it is used after the model is created to test the accuracy of the model. This variable is the folder that the validate images are stored

weighted_classes – **Integer** – Default: 80

number of classifiers that was used for the weights file. If training from a weights file, this is the number of classifiers the weights file has. The default is 80 because of the YOLOv3-608 file. If you are not using a weights file then this variable has no effect on the training.

weights - Sting - *.weights* or *.tf* file

when the workbench trains a model, it uses a pre-trained model to assist it
this variable is the path to the pre-trained model. For tiny weights download the YOLOv3-tiny file. For regular weights you can use YOLOv3-608

Workbench Benchmark Tests:

Tests using the following *benchmark.txt* file:

batch_size --------- 8
classifiers ---------- ./data/classifier.names
dataset_test ------- ./data/test.tfrecord
dataset_train ------ ./data/train.tfrecord
epochs ------------ 30
image_size -------- 224
max_checkpoints - 15
max_sessions ----- 5
mode ------------- fit
output ------------ ./current_session/
sessions ---------- ./saved_sessions/
tiny_weights ------ False
transfer ---------- none
val_img_num ----- 3
val_image_path -- ./images/validate/
weights ---------- ./data/yolov3.weights

GPU	**Final Loss rate**	**Training time**	**Total Time**
Geforce GTX 1660 Ti	7.8890	*42min 51sec*	*42min 08sec*
CPU			
Intel Core i5-8400	7.1063	*9hr 32min 5sec*	*9hr 35min 39sec*
	6.9853	17hr 2min 39sec	17hr 5min 1sec

4) Errors

Conda Environment Issues:

If issues arise in your conda environment, you can remove and re-add the environment:

To remove a conda environment:

```
#example conda remove --name cpu --all -y
conda remove --name <name of env> --all -y
```

ImportError: DLL load failed: The specified module could not be found.

This issue arises when your cuda and cudnn was improperly installed. This error happens when all the files that cuda needs were not found. Most commonly this happens do to the incorrect versions being installed go back to the gpu install portion of this document and ensure you correctly went through the steps, then restart your machine.

Allocator: ran out of memory trying to allocate

This issue arises when you have run out of memory. The common issue is that your batch size is too high. Lower your batch size to solve this issue. Another issue that can arrive is your drive is full. Checkpoints take up a lot of memory, lower your max_checkpoint integer and remove any unnecessary files from your drive