Digital Factory floor visit web app

# ALGORITHM TRAINING PROCEDURES:

We have used **SDD Mobilenet V2 lite** which gives acceptable accuracy at a good speed. We have used the tensorflow object detection API which provides various algorithms like SSD Mobilenet, SSD Inception etc.. We have used the pre-trained model which has been trained on Microsoft coco dataset with 90 labels. We have fine-tuned the pre-trained model with setting the pre-trained models weights as the initialization weights and trained on our custom dataset. We have two trained models one for the physical object detection and other for the floor panorama. We have used google colab for training purposes. So we need to upload the training samples in google drive for to be trained in colab because if you leave the colab without interaction, it is possible for the local variables to get removed from the kernel. So it is safe to save the data on drive.

We have used **VOTT**(visual object tagging tool) for annotation purposes. We need to annotate the images either as standalone images or from video. Other tool available in the market is labelImg but I found VOTT to be more user friendly.

Using the VOTT tool, we shall create our dataset – We need to have positive samples and as well as negative samples. We have to export the tagged and untagged images – tagged(for positive images) and untagged (for negative samples) as PASCAL VOC format. In this format we will have three folders in the output directory which we specified – annotations, Main, jpegimages and a file called .pbtext. Inside the Main folder we will have two files train.txt and val.txt with the corresponding image names. Let say we have only one class for detection, in that case if the image has that class, then in that files we will have the file name followed by 1 with a space and if that image is a untagged one, it will have -1. NOTE: The classes should start with 1 and not zero which you can see in the .pbtext file. Then create a folder called data (any name) and create two sub folders train and test. Then place the images(present in JPEGimages folder) given in the val.txt file to the test folder. Now copy the image annotations in the annotations folder with same name as the images to the test folder. That is we need to have both annotations and their corresponding images in the train and test folders accordingly. Once it is done, compress the data folder along with the .pbtext file and upload it in drive. Follow the steps in the colab for training. The notebook contains the steps to be followed while training. The notebook also has testing part as inference at the end of training.

Below is the colab notebook link which has been used or training.

<https://github.com/sampathkumaran90/dffloor/blob/master/SSD_FINE_TUNING.ipynb>

Below are the references for the algorithm training.

<https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html>

<https://medium.com/analytics-vidhya/custom-object-detection-with-tensorflow-using-google-colab-7cbc484f83d7>

<https://github.com/rishi93/raccoon-detector>

Once you have created the model, it will be in .pb (protobuf) format which is the default format for tensorflow. We need to download it and convert it into tfjs format (json) using a python package called tensorflow\_converter.

Below is the sample code for conversion

tensorflowjs\_converter --input\_format=tf\_frozen\_model --output\_node\_names='detection\_boxes,detection\_classes,detection\_scores,num\_detections' --saved\_model\_tags=serve --output\_format=tfjs\_graph\_model **‘path to the frozen model’** **‘path to the desired output folder’**

We have used web workers to run the inference as it hangs the browser if not used. We have also used a node package called **‘workerize-loader’**. The main advantage of using this workerize loader is we shall import the node packages like tfjs inside the web workers directly. Also we have not used the default ssdcoco tfjs package. We have modified the source code of the same for our needs. It is present in worker.js and worker\_physical.js in examples folder.

# UI PROCEDURES:

We have used three.js for rendering the equirectangular video panorama with device orientation controls which uses accelerometer and gyroscope of the device used. We have used the media capture and stream capture for physical object detection and panorama place detection respectively. The code for the panorama is in panorama\_video.js.

For physical object detection, we have used the webcam / back camera of mobile. This video feed is drawn over a canvas and then the image is extracted from it and it has been sent to the web worker for inference.

For place detection inside panorama, we have screen capture / media capture and performed the same steps as the above for detection. Here we have also placed a marker at the center of the bounding box. Once the user clicks on the marker, we have loaded a video explaining about the place of interest. Once the video is over, we prompt the user for asking questions about that particular place. We have used the Google TTS(text to speech) and speech synthesizer for replying the text through speech. If the user says, negative words like no, sorry like that we will exit that conversation and go back to the panorama.

# DIALOGFLOW PROCEDURES:

For the conversational part, we have used **DialogFlow** as the agent. We have configured the intents in the console. <https://dialogflow.cloud.google.com/#/login>. Once we have configured the intents, we need to setup the google cloud account. <https://dialogflow.com/docs/reference/v2-auth-setup> - this link provides the steps to setup the dialogflow v2. We need to download the json file at the end of the procedure given in the link and save it in our local machine. Then we need to set it up as an environment variable in the name called **“GOOGLE APPLICATION CREDENTIALS”**. Then we have used express as a server which acts as a middleware. The code for the express api part is in index.js in the root folder. We use dialogflow node package in express to use its api part.