Iteration 3 (DL methods)

Problem Statement:

To find the actions of the JCB arm, i.e; (idle, dig, swing, dump)

Method 1

CNN-with-regression (doesnt work as expected)

- **Model:** 5-layerd 2d-cnn with a mean square error loss over logits. Each layer has a 2d convolution layer with relu activation followed by 2d max-pooling.
- **Output:** vector of length 6, each pair represent the location of one of the three joint points. (3*2)
- **Trained:** >=1000 epochs
- **Dataset:** Train data: 40 frames, manually labelled all three points on each image
  Test data: 20 frames (unable to create large dataset)

Cons:

- ✗ Loss was not able to converge to an acceptable range.
- ✔ Faster training time

Note: The plan was to use these joint points and train a time-series based RNN network. Since due to lack of enough training data, the loss was not able to converge.

Method 2.1

3D-CNN v1 (works, but slow)

- **Model:** 6-layerd 3d-cnn with a cross entropy loss over softmax-logits. Each layer has a 3d convolution layer (3 frames – 3 filters) with relu activation followed by 3d max-pooling.
- **Output:** one-hot vector of actions
- **Trained:** 12 epochs with k-fold training
- **Dataset:** Train data: used first 1:40 sec for training (approx 200 frames)
  Test data: tested the model on remaining part of the video

Cons:

- ✗ High training time
- ✗ More no.of epochs needed
- ✔ Higher accuracy

Method 2.2

3D-CNN v2 (Final working model !)

- **Model:** 6-layerd 3d-cnn with a cross entropy loss over softmax-logits. Each layer has a 3d convolution layer (2 frames – 1 filters) with relu activation followed by 3d max-pooling.
- **Output:** one-hot vector of actions

- **Trained:** 12 epochs with k-fold training

- **Dataset:** Train data: used first 1:40 sec for training (approx 200 frames)
  Test data: tested the model on remaining part of the video

Pros: (when compared to previous model - v1)

- ✔ Relative drop in training time
- ✔ Less no.of training epochs needed
- ✔ Faster processing rate during testing
- ✔ Near to higher accuracy

Results:

output video link : https://www.youtube.com/watch?v=ZFAAfcEaXpM
(Please play at fullscreen HD resolution, or the labels might not be visible.)
(Actions being detected are – idle, digging, swing, dumping)

Note:

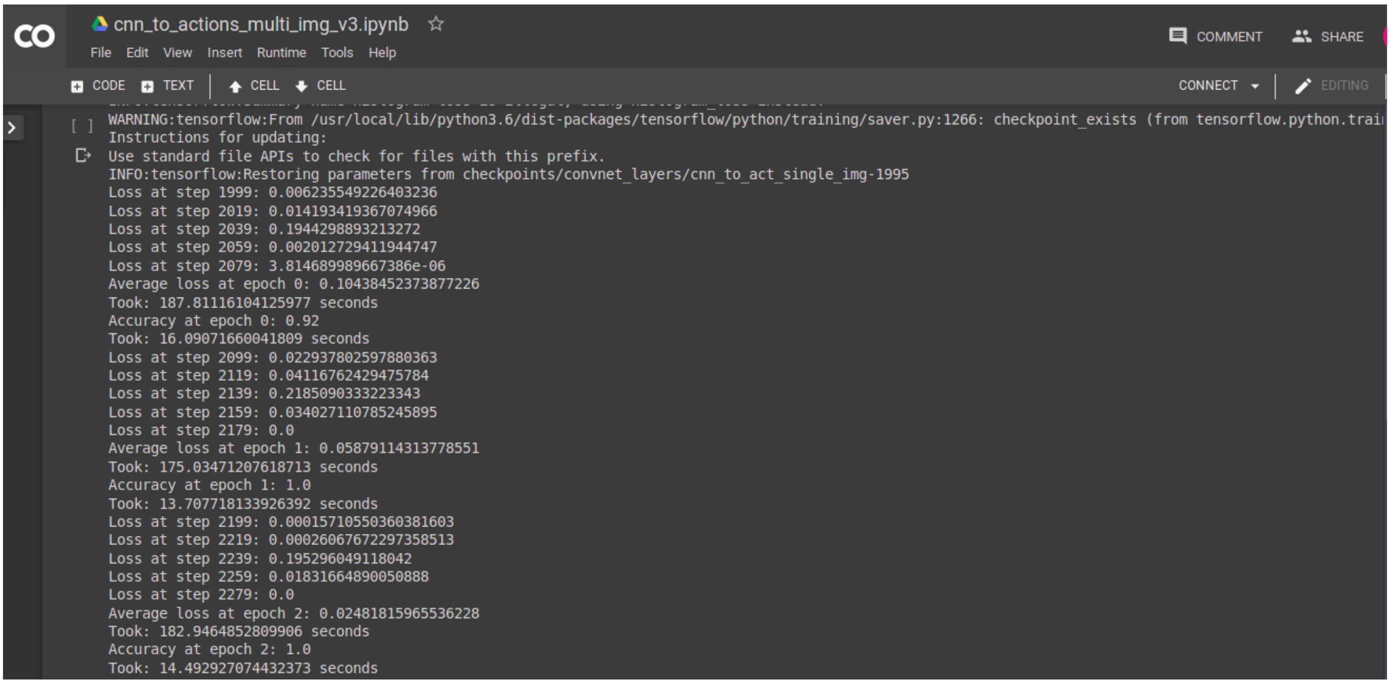
Also tested on a similar video from youtube, **without any additional training**, and it was able to detect digging and swings, for most of the time. (If trained, will perform better.)
output video link : https://www.youtube.com/watch?v=cPDQ7AMzt2k

Processing time:

Video frame rate is 24 frames per sec. The model process 12 frames at atime.
On my machine (i5, 8GB RAM, NO GPU) it takes approx 1.5 sec to process 12 frames.
On google colab (12GB GPU Memory) it takes approx 0.01 sec to process 12 frames.

Screenshots of runtime in Google colab

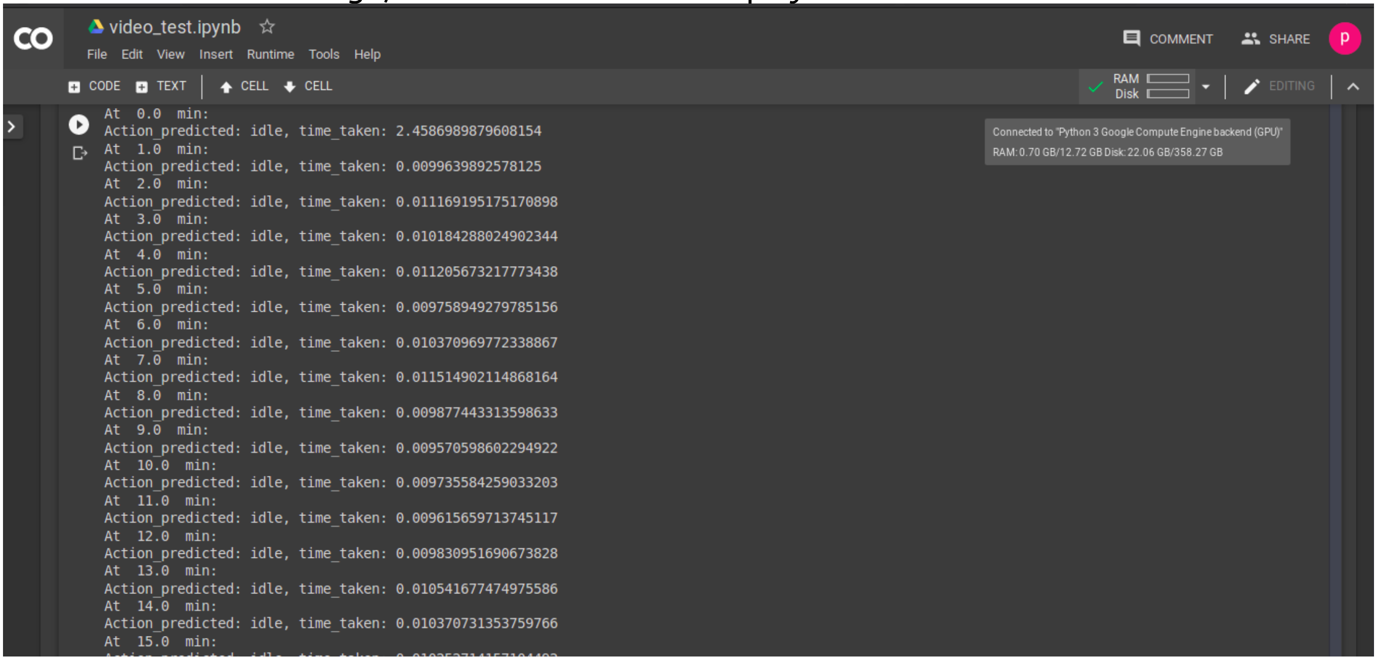
Training phase in colab:


CO
cnn_to_actions_multi_img_v3.ipynb ☆
File Edit View Insert Runtime Tools Help
COMMENT
SHARE
CODE
TEXT
CELL
CELL
CONNECT
EDITING

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/training/saver.py:1266: checkpoint_exists (from tensorflow.python.trai
Instructions for updating:
Use standard file APIs to check for files with this prefix.
INFO:tensorflow:Restoring parameters from checkpoints/convnet_layers/cnn_to_act_single_img-1995
Loss at step 1999: 0.006235549226403236
Loss at step 2019: 0.014193419367074966
Loss at step 2039: 0.1944298893213272
Loss at step 2059: 0.002012729411944747
Loss at step 2079: 3.814689989667386e-06
Average loss at epoch 0: 0.10438452373877226
Took: 187.81116104125977 seconds
Accuracy at epoch 0: 0.92
Took: 16.09071660041809 seconds
Loss at step 2099: 0.022937802597880363
Loss at step 2119: 0.04116762429475784
Loss at step 2139: 0.2185090333223343
Loss at step 2159: 0.034027110785245895
Loss at step 2179: 0.0
Average loss at epoch 1: 0.05879114313778551
Took: 175.03471207618713 seconds
Accuracy at epoch 1: 1.0
Took: 13.707718133926392 seconds
Loss at step 2199: 0.00015710550360381603
Loss at step 2219: 0.00026067672297358513
Loss at step 2239: 0.195296049118042
Loss at step 2259: 0.01831664890050888
Loss at step 2279: 0.0
Average loss at epoch 2: 0.02481815965536228
Took: 182.9464852809906 seconds
Accuracy at epoch 2: 1.0
Took: 14.492927074432373 seconds
```

Testing phase in colab: (takes around 0.01 sec to process 12 frames)

Note: Also in the image, the GPU allocation is displayed



Connected to "Python 3 Google Compute Engine backend (GPU)"
RAM: 0.70 GB/12.72 GB Disk: 22.06 GB/358.27 GB

```
At  0.0  min:
Action_predicted: idle, time_taken: 2.4586989879608154
At  1.0  min:
Action_predicted: idle, time_taken: 0.0099639892578125
At  2.0  min:
Action_predicted: idle, time_taken: 0.011169195175170898
At  3.0  min:
Action_predicted: idle, time_taken: 0.010184288024902344
At  4.0  min:
Action_predicted: idle, time_taken: 0.011205673217773438
At  5.0  min:
Action_predicted: idle, time_taken: 0.009758949279785156
At  6.0  min:
Action_predicted: idle, time_taken: 0.010370969772338867
At  7.0  min:
Action_predicted: idle, time_taken: 0.011514902114868164
At  8.0  min:
Action_predicted: idle, time_taken: 0.009877443313598633
At  9.0  min:
Action_predicted: idle, time_taken: 0.009570598602294922
At  10.0  min:
Action_predicted: idle, time_taken: 0.009735584259033203
At  11.0  min:
Action_predicted: idle, time_taken: 0.009615659713745117
At  12.0  min:
Action_predicted: idle, time_taken: 0.009830951690673828
At  13.0  min:
Action_predicted: idle, time_taken: 0.010541677474975586
At  14.0  min:
Action_predicted: idle, time_taken: 0.010370731353759766
At  15.0  min:
```

Further improvements:

- To create a generalized model, so that given any new video with different point-of-view / environment setting our algorithm must be able to detect the actions, with minimal amount of training.
- Conversion of fish-eye camera view to a flat image using intrinsic and extrinsic camera parameters for viewieng angles.

Few ideas for scope of improvement:

- unet(segmentation) + cnn encoder (feature extraction) + rnn (time series prediction)