Table of Contents

Table of Contents **2**

Train and validate a deep neural network **3**

on MNIST dataset using tensorflow **3**

Model Performance 3

Table 1: Q1 CNN Model Information 3

Fig. 1: Q1 CNN Loss 4

Fig 2: Q1 CNN Accuracy 4

Discussion 4

Fig. 3: MNIST Data Set Images 5

Fig. 4: Q1 CNN Model Architecture 5

Use the TF-Hub module to perform object **6**

detection **6**

Table 2: Inference Time by model and image 6

Figs. 5 & 6: Top to Bottom: Image of Khraungbin with FasterRCNN+InceptionResNetV2, Image of Khraungbin with ssd + mobilenetV2 7

Figs. 7 & 8: Top to Bottom: Image of two women key punching at Texas A&M with FasterRCNN+InceptionResNetV2, Image of two women key punching at Texas A&M with ssd + mobilenetV2 8

Train custom object detection model in Tensorflow **9**

Figure 9: EfficientDet Architecture, Google Research 9

Figure 10: Screenshot of EfficientDet model being trained 10

Figure 11: Inference on Salad Platter 10

Fig 12: Inference on Fruit Salad 11

Fig. 13: Inference on Cee Lo Green and The Muppets 11

Resources **12**

Q1: Train and validate a deep neural network on MNIST dataset using tensorflow

Model Performance

Model	CNN
Network Architecture	Input (Flatten), Dense, Dropout, Dense
Loss Function	Sparse Categorical Crossentropy
Training Accuracy	97.64%
Test Accuracy	97.69%

Table 1: Q1 CNN Model Information

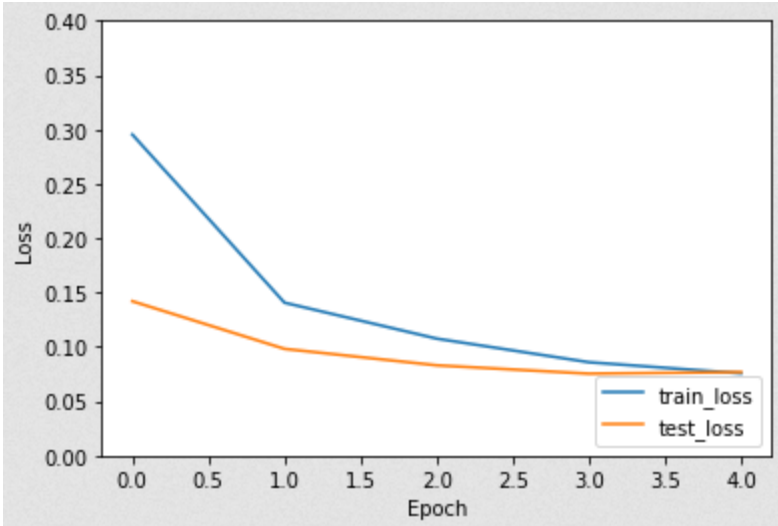


Fig. 1: Q1 CNN Loss

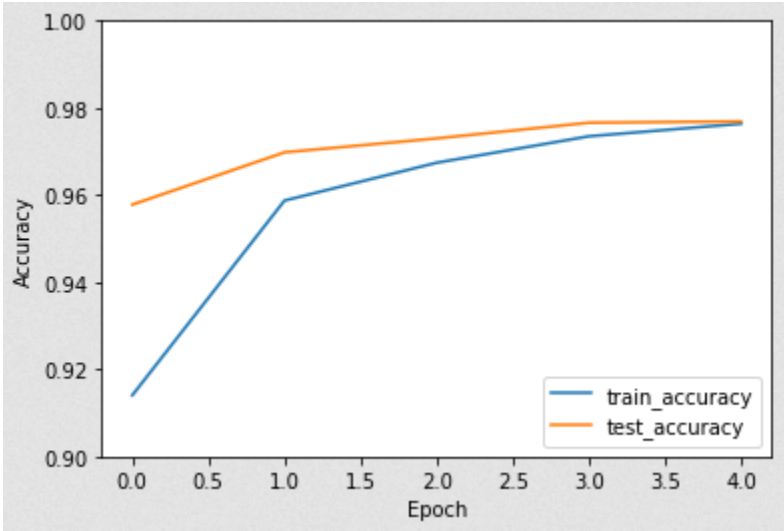


Fig 2: Q1 CNN Accuracy

Discussion

This model was chosen and implemented because of the simplicity and effectiveness of the Keras API. Although this model does perform relatively well with little training and/or hardware needed, performance could be improved in a number of ways:

- Increase batch size
- Implement Hyperparameter Tuning
- Data Augmentation
- Creating a deeper (more complex) network structure

Below are images depicting some example images from the MNIST data set, as well as the depiction of this CNN model architecture.

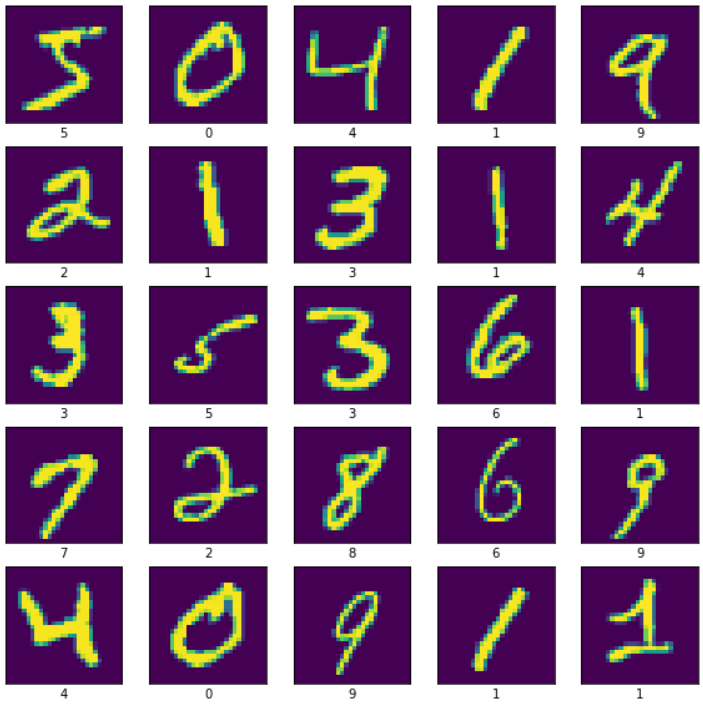


Fig. 3: MNIST Data Set Images

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_8 (Flatten)         (None, 784)               0

 dense_16 (Dense)            (None, 128)               100480

 dropout_1 (Dropout)         (None, 128)               0

 dense_17 (Dense)            (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

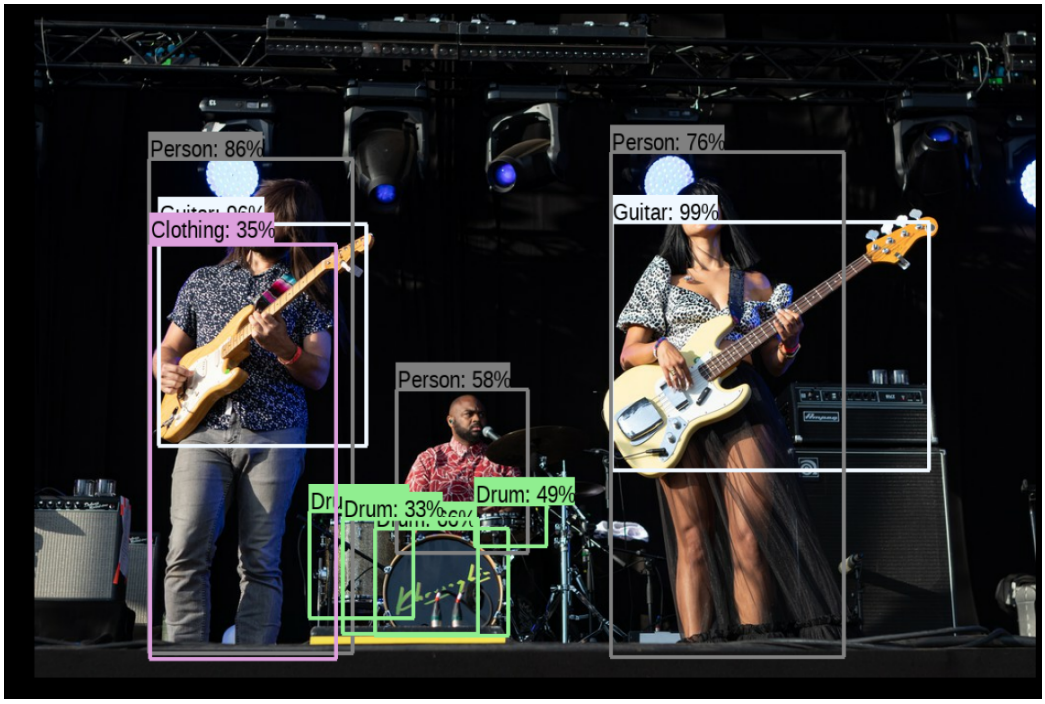
Fig. 4: Q1 CNN Model Architecture

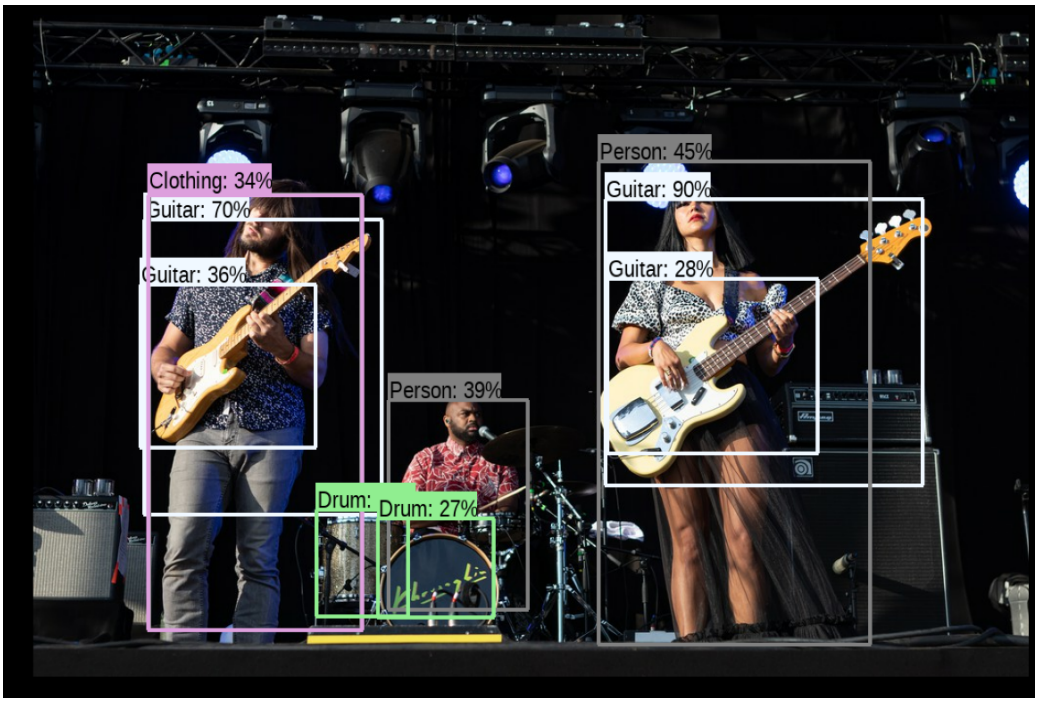
Q2: Use the TF-Hub module to perform object detection

	Key Punching	Band Performing
FasterRCNN+InceptionResNetV2	25.105 [s]	22.032 [s]
ssd + mobilenetV2	7.231 [s]	0.287 [s]

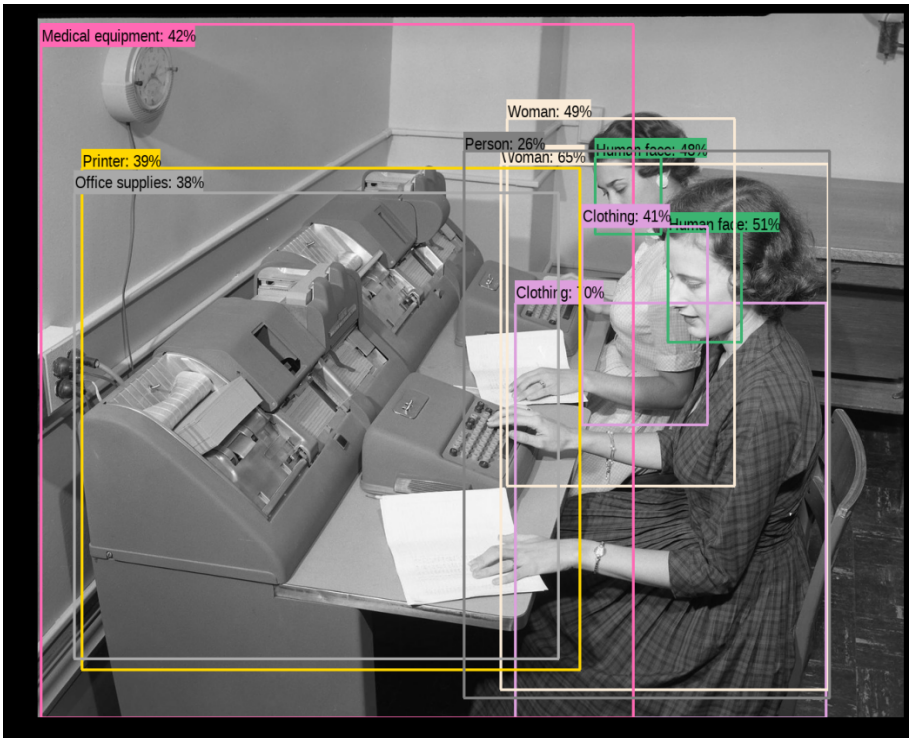
Table 2: Inference Time by model and image

Both of the models used in this example are composed of a network-based architecture, but capped with a detection network (FasterRCNN and sdd, respectively). As can be understood from the differences in computing times, ssd+mobilenetV2 is best used when a quick inference time or light computing requirements are paramount. However, there is a notable difference in performance (see figs. 5-9), notably in the % confidence in objects that are identified correctly.





Figs. 5 & 6: Top to Bottom: Image of Khraungbin with FasterRCNN+InceptionResNetV2, Image of Khraungbin with ssd + mobilenetV2



Figs. 7 & 8: Top to Bottom: Image of two women key punching at Texas A&M with FasterRCNN+InceptionResNetV2, Image of two women key punching at Texas A&M with ssd + mobilenetV2

Q3: Train custom object detection model in Tensorflow

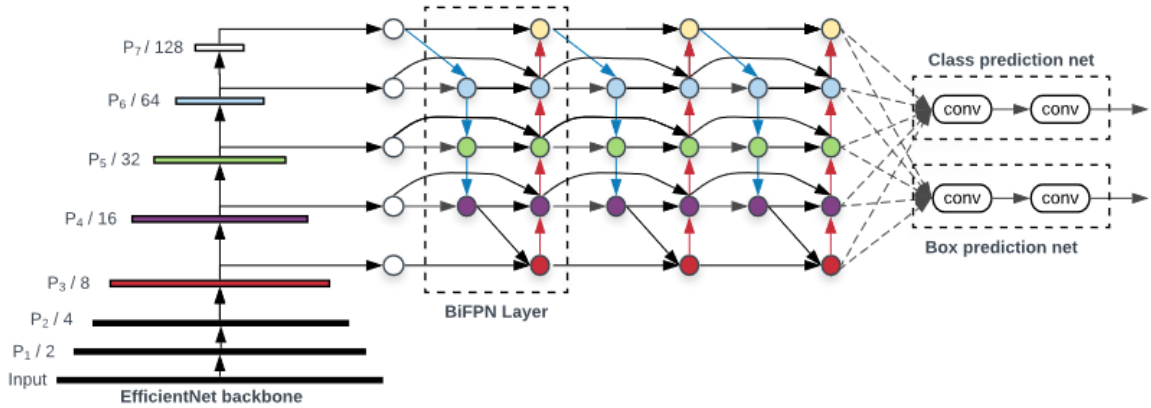


Figure 9: EfficientDet Architecture, Google Research

The architecture of this model is described as follows in the Google Research paper on methodology and development of the model:

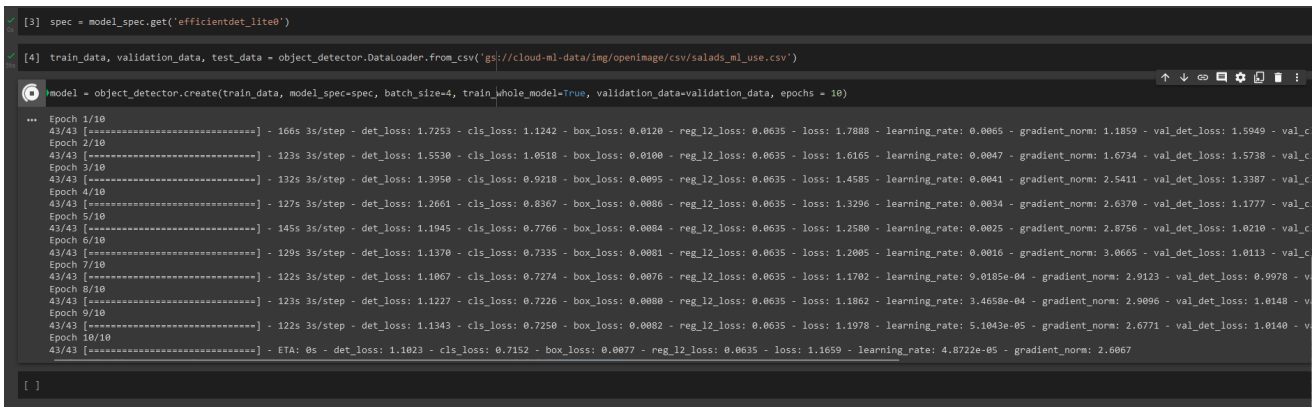
> "[Figure 9] shows the overall architecture of EfficientDet, which largely follows the one-stage detectors paradigm [27, 33, 23, 24]. We employ ImageNet-pretrained EfficientNets as the backbone network. Our proposed BiFPN serves as the feature network, which takes level 3-7 features {P3, P4, P5, P6, P7} from the backbone network and repeatedly applies top-down and bottom-up bidirectional feature fusion. These fused features are fed to a class and box network to produce object class and bounding box predictions respectively. Similar to [24], the class and box network weights are shared across all levels of features."

The largest challenges to training this model on unseen data were really involved in acquiring and mounting the data set and training the model. For the purposes of this report, 10 epochs were used with a batch size of 4 to speed up the training process. The data set used, a public google image set of salads, only labels the following objects:

- Salad
- Baked Goods
- Cheese
- Seafood

- Tomato

A detection threshold of only 20% was used for inference. As seen in Figs. 11-13, the model does not seem to perform very well even on a basic salad platter - missing baked goods and incorrectly identifying a nut and an olive as tomatoes.
Notably, we reach a 23% detection threshold on a Tomato on an actual tomato, but 41% on Cee Lo Green in a red jacket singing with The Muppets. This model with the used training parameters may not be recommended for use for other salad detection purposes. Perhaps an increased batch size and number of epochs would yield improved results.



```
[3] spec = model_spec.get('efficientdet_lite0')

[4] train_data, validation_data, test_data = object_detector.DataLoader.from_csv('gs://cloud-ml-data/img/openimage/csv/salads_ml_use.csv')

model = object_detector.create(train_data, model_spec=spec, batch_size=4, train_whole_model=True, validation_data=validation_data, epochs = 10)

Epoch 1/10
43/43 [==============================] - 166s 3s/step - det_loss: 1.7253 - cls_loss: 1.1242 - box_loss: 0.0120 - reg_l2_loss: 0.0635 - loss: 1.7888 - learning_rate: 0.0065 - gradient_norm: 1.1859 - val_det_loss: 1.5949 - val_c
Epoch 2/10
43/43 [==============================] - 123s 3s/step - det_loss: 1.5530 - cls_loss: 1.0518 - box_loss: 0.0100 - reg_l2_loss: 0.0635 - loss: 1.6165 - learning_rate: 0.0047 - gradient_norm: 1.6734 - val_det_loss: 1.5738 - val_c
Epoch 3/10
43/43 [==============================] - 132s 3s/step - det_loss: 1.3950 - cls_loss: 0.9218 - box_loss: 0.0095 - reg_l2_loss: 0.0635 - loss: 1.4585 - learning_rate: 0.0041 - gradient_norm: 2.5411 - val_det_loss: 1.3387 - val_c
Epoch 4/10
43/43 [==============================] - 127s 3s/step - det_loss: 1.2661 - cls_loss: 0.8367 - box_loss: 0.0086 - reg_l2_loss: 0.0635 - loss: 1.3296 - learning_rate: 0.0034 - gradient_norm: 2.6370 - val_det_loss: 1.1777 - val_c
Epoch 5/10
43/43 [==============================] - 145s 3s/step - det_loss: 1.1945 - cls_loss: 0.7766 - box_loss: 0.0084 - reg_l2_loss: 0.0635 - loss: 1.2580 - learning_rate: 0.0025 - gradient_norm: 2.8756 - val_det_loss: 1.0210 - val_c
Epoch 6/10
43/43 [==============================] - 129s 3s/step - det_loss: 1.1370 - cls_loss: 0.7335 - box_loss: 0.0081 - reg_l2_loss: 0.0635 - loss: 1.2005 - learning_rate: 0.0016 - gradient_norm: 3.0665 - val_det_loss: 1.0113 - val_c
Epoch 7/10
43/43 [==============================] - 122s 3s/step - det_loss: 1.1067 - cls_loss: 0.7274 - box_loss: 0.0076 - reg_l2_loss: 0.0635 - loss: 1.1702 - learning_rate: 9.0185e-04 - gradient_norm: 2.9123 - val_det_loss: 0.9978 - v
Epoch 8/10
43/43 [==============================] - 123s 3s/step - det_loss: 1.1227 - cls_loss: 0.7226 - box_loss: 0.0080 - reg_l2_loss: 0.0635 - loss: 1.1862 - learning_rate: 3.4658e-04 - gradient_norm: 2.9096 - val_det_loss: 1.0148 - v
Epoch 9/10
43/43 [==============================] - 122s 3s/step - det_loss: 1.1343 - cls_loss: 0.7250 - box_loss: 0.0082 - reg_l2_loss: 0.0635 - loss: 1.1978 - learning_rate: 5.1043e-05 - gradient_norm: 2.6771 - val_det_loss: 1.0140 - v
Epoch 10/10
43/43 [==============================] - ETA: 0s - det_loss: 1.1023 - cls_loss: 0.7152 - box_loss: 0.0077 - reg_l2_loss: 0.0635 - loss: 1.1659 - learning_rate: 4.8722e-05 - gradient_norm: 2.6067

[ ]
```

Figure 10: Screenshot of EfficientDet model being trained

Figure 11: Inference on Salad Platter

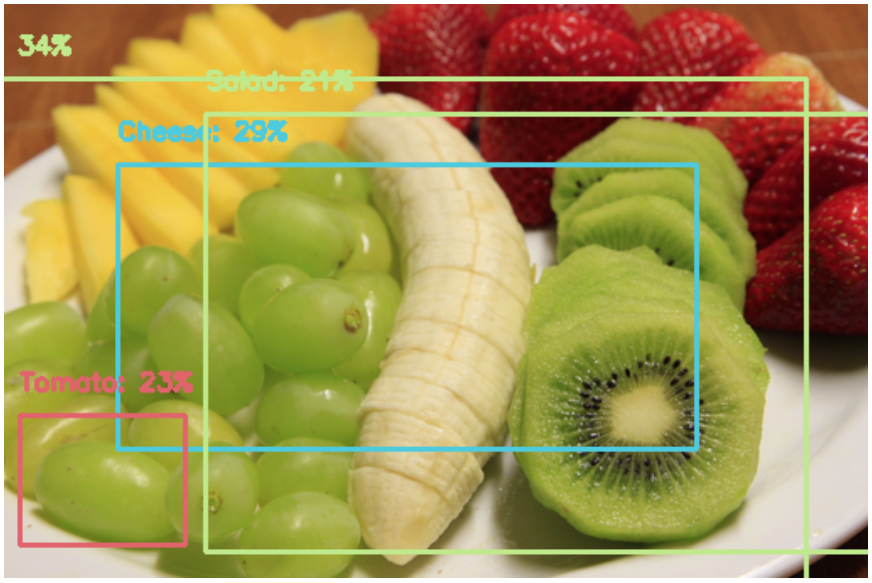


Fig 12: Inference on Fruit Salad

Fig. 13: Inference on Cee Lo Green and The Muppets

Resources

1. Train a salad detector with TFLite Model Maker
   The TensorFlow Authors, 2021
   https://colab.research.google.com/github/googlecodelabs/odml-pathways/blob/main/object-detection/codelab2/python/Train_a_salad_detector_with_TFLite_Model_Maker.ipynb
2. EfficientDet: Scalable and Efficient Object Detection
   Google Research, Brain Team 2020
   https://arxiv.org/pdf/1911.09070.pdf
3. Khraungbin Image:
   https://upload.wikimedia.org/wikipedia/commons/1/15/Khruangbin_am_Haldern_Pop_Festival_2019_-_09_-_Foto_Alexander_Kellner.jpg
4. TAMU Cushing Library Image:
   https://upload.wikimedia.org/wikipedia/commons/6/6f/Keypunching_at_Texas_A%26M.jpg
5. Salad Platter Image: https://upload.wikimedia.org/wikipedia/commons/9/94/Salad_platter.jpg
6. Fruit Salad Plate:
   https://upload.wikimedia.org/wikipedia/commons/1/1f/Fruit_salad_plate.jpg
7. CeeLo Green performing with the Muppets at the Rockefeller Center Christmas Tree Lighting 2012:
   https://upload.wikimedia.org/wikipedia/commons/5/58/CeeLo_Green_performing_with_the_Muppets_at_the_Rockefeller_Center_Christmas_Tree_Lighting_2012_%2811200418446%29.jpg