Grocery Identification for Smart Refrigerators

Apoorva Lakhmani, Neha Lalwani and Nirali Merchant
lakhmani.a@husky.neu.edu, lalwani.n@husky.neu.edu, merchant.n@husky.neu.edu

***Abstract* – Deep learning has made significant breakthrough in the past decade. Various novel applications have been developed and achieved good performance by leveraging the latest advances in deep learning. In this paper, we propose to utilize deep learning based technique, specifically, Convolutional Neural Network (CNN) and Region based Convolutional Neural Network(R-CNN) using Keras and TensorFlow Object Detection API, to develop an auto-counting system for food items stored in the refrigerator. Given a picture of the refrigerator at a particular time, the system can automatically detect the specified categories of refrigerated items and give their respective counts. This is achieved by learning about CNN and RCNN models and then applying transfer learning on a pretrained model on coco dataset which uses Region - Based Fully Convolutional Network (RFCN) using Tensorflow Object Detection API on our own dataset. The dataset is web scraped and labelled using the graphical image annotation tool- Labelimg. This is given as an input to the tweaked pre- trained model. The accuracy achieved using the resnet 101-layer model which is uses RFCN gives dependable results. Experimental results on our dataset confirm that RFCN has comparable accuracy to methods that use multiple costly per-region subnetwork hundreds of times (fast RCNN and faster RCNN) in contrast to the computation shared on the entire image.**

INTRODUCTION

Most developments in technology are for the purpose of doing more or making life easier. Many people these days find it difficult to manage their time in an organized way. We always find ourselves wondering what we have got in our fridge *after* we have already left for the store or get ourselves to reach home and remember that we forgot a vital ingredient for the recipe we decided to make for dinner. Smart refrigerator achieves this by tracking the contents of the fridge using Machine Learning and ordering from an online store when stocks are low which is an in-progress iOS app. A Smart refrigerator allows the user to remotely monitor which food items they have refrigerated. It helps to keeps an inventory list, which can be managed through a smartphone, tablet or other mobile device. Our application aims to achieve grocery identification and quantity of as many as possible refrigerated items. The idea is to integrate this external application on mobile or tablet and allow the user to get the information when requested

We decided to take a 'from the top' approach for the problem at hand which involved understanding CNN and moving forward to RCNN. We proceeded to building an Image classifier for the most likely refrigerated items to understand CNN exhaustively. Moving forward, we studied the differences between Fast RCNN, Faster RCNN and RFCN. Thereafter, we decided to use Tensorflow Object Detection API by studying three pretrained models on *coco* dataset namely Single Shot Detector(SSD), faster RCNN and RFCN purely due to its simplicity and availability of resources. Based on the results provided by each one of them, we elected to move forward with RFCN since it gave the most dependable results. The subsequent sections give a detailed assimilation on our understanding of the concepts for solving the problem, the most notable being an average accuracy of 70% of the refrigerated items.

METHODS

NEURAL NETWORKS AND ACTIVATION FUNCTIONS

Neural network is a machine learning technique which enables a computer to learn from the observational data. Neural networks are typically used to derive meaning from complex and non-linear data, detect and extract patterns which cannot be noticed by the human brain.
One of the most important application of neural network is Image classification which is falls under Convolutional Neural network[5]
Neural networks are organized on layers made up of interconnected nodes which contain an activation function. These patterns are presented to the network through the input layer which further communicates it to one or more hidden layers. The hidden layers perform all the processing and pass the outcome to the output layer.

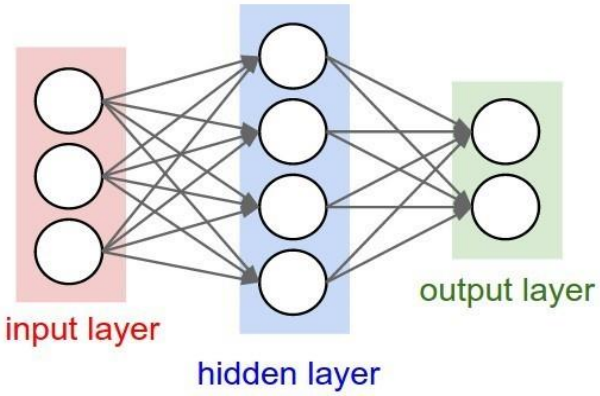


FIGURE 1
NEURAL NETWORK STRUCTURE

The activation functions used are:

1. RELU: A(x) = max (0, x) - The ReLu function is as shown above. It gives an output x if x is positive and 0 otherwise

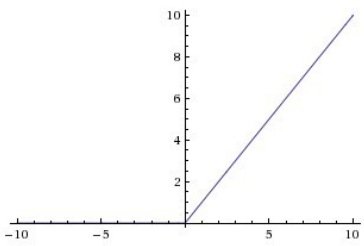


FIGURE 2
RELU ACTIVATION FUNCTION

2. Softmax - The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function.The output of the softmax function is equivalent to a categorical probability distribution, it tells you the probability that any of the classes are true.

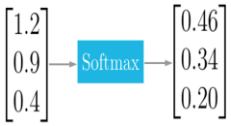


FIGURE 3
SOFTMAX ACTICATION FUNCTION

I. Image Classification Using Keras

- **Installation**: a machine with Keras, SciPy, PIL installed and a training data directory, validation data directory containing one subdirectory per image class of the dataset. [9]
- **Database**: The database used to achieve this is a Fruits dataset . We have taken a limited number of images for training purpose. Training a convnet from scratch on a small image dataset will still yield reasonable results, without the need for any custom feature engineering since convnets are the right tool for the job.
- **CNN for image classification tasks**: Each layer of the input is basically describing the locations in the original image for where certain low-level features appear. When a set of filters is applied on top of that it passes through the second convolutional layer, the output of which will be activations that represent higher level features. Types of these features are semicircles (combination of a curve and straight edge) or squares (combination of several straight edges). As we go through the network and go through more conv layers, you get activation maps that represent more and more complex features. By the end of the network, we have some filters that activate when there are different categories of fruit in the dataset. The functions of each of the layer is explained below.

  Function of the input layer:

  This is the first layer of a neural network. It is used to provide the input data or features to the network.

  Function of the hidden layer:

  This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from.

  Function of the output layer:

  The way this fully connected layer works is that it looks at the output of the previous layer (which as we remember should represent the activation maps of high level features) and determines which features most correlate to a particular class.

  Function of Max Pooling:

  Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

  Function of Flattening:

  We need to convert the output of the convolutional part of the CNN into a 1D feature vector, to be used by the ANN part of it. This operation is called flattening. It gets the output of the convolutional layers, flattens all its structure to create a single long feature vector to be used by the dense layer for the final classification.

  Function of Dropout layer:

  Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. By using a dropout layer, we randomly deactivate certain units (neurons) in a layer with a certain probability p. As a result of this the neural networks will learn the redundant representation differently. After activating the dropout layer. the training will also be faster.

  Function of the dense layer:

  A dense layer connects each neuron from the previous layer to the next layer. Once we have extracted the features, we want to make some decisions about what those features mean with respect to what we are trying to classify. A dense layer helps in examining the global picture of the features in contrast to zooming in more on the features. [7]

  Figure 4 presents the snippet of the model that we have used.

- **Process:** The input is fed to a network of stacked convolutional, pool and dense layers. The activation function used is RELU. The model is trained for 30 epochs. The output is a SoftMax layer indicating whether it identifies the test image provided
- **Data pre-processing and augmentation:** To make the most of our few training examples, we augmented them via a number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better. [7]

  Data Augmentation performed on the train and validation set is presented in Figure 5.

```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 49, 49, 32)	0
dropout_1 (Dropout)	(None, 49, 49, 32)	0
conv2d_2 (Conv2D)	(None, 47, 47, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 23, 23, 32)	0
dropout_2 (Dropout)	(None, 23, 23, 32)	0
conv2d_3 (Conv2D)	(None, 21, 21, 64)	18496
max_pooling2d_3 (MaxPooling2	(None, 10, 10, 64)	0
dropout_3 (Dropout)	(None, 10, 10, 64)	0
flatten_1 (Flatten)	(None, 6400)	0
dense_1 (Dense)	(None, 128)	819328
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 9)	1161

FIGURE 4
MODEL SUMMARY FOR KERAS

```
batch_size = 16

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory('fruits/data/train',
        target_size=(100, 100),
        batch_size=batch_size,
        class_mode='categorical')

# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory('fruits/validation',
        target_size=(100, 100),
        batch_size=batch_size,
        class_mode='categorical')
```

FIGURE 5
DATA AUGMENTATION

- **Training the model:**

We fit the created model (Figure 4) on the training data. While fitting the model, we provide training data, steps_per_epoch, epochs, validation data and validation steps as:

i. steps_per_epoch: Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default None is equal to the number of samples in your dataset divided by the batch size, or 1 if that cannot be determined.

ii. validation_steps: relevant if steps_per_epoch is specified. Total number of steps (batches of samples) to validate before stopping.

iii. epochs: Integer. Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided. Note that in conjunction with initial_epoch, epochs is to be understood as "final epoch". The model is not trained for a number of iterations given by epochs, but merely until the epoch of index epochs is reached.

iv. train generator: No of images in the training dataset

v. validation generator: Number of images in validation dataset.

```
#fitting the model on training data
def fitModel(epochs):
    model.fit_generator(train_generator,
    steps_per_epoch=4361 // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=1460 // batch_size)
```

FIGURE 6
MODEL FITTING METHOD

- **Hyper parameter tuning**

Hyperparameter tuning typically address the problem of how to choose the next hyperparameter values so that model gives higher accuracy. We briefly survey the hyperparameters for convnet

i. Number of epochs: Number of epochs is the the number of times the entire training set pass through the neural network. We should increase the number of epochs until we see a small gap between the test error and the training error

ii. Batch Size: Mini-batch is usually preferable in the learning process of convnet. A range of 16 to 128 is a good choice to test with. We should note that convnet is sensitive to batch size.

iii. Activation function: Activation function introduces non-linearity to the model. Usually, RELU works well with convnet. Other alternatives are sigmoid, tanh and other functions depending on the task

For this problem, we have used RELU, tanh and sigmoid. [16]

II. Transfer Learning using TensorFlow Object Detection API

TensorFlow's Object Detection API is a powerful tool that makes it easy to construct, train, and deploy object detection models. In most of the cases, training an entire convolutional network from scratch is time consuming and requires large datasets. This problem can be solved by using the advantage of transfer learning with a pre-trained model using the TensorFlow API. Before getting into the technical details of implementing the API, let's discuss the concept of transfer learning.

Transfer learning is a research problem in machine learning that focuses on storing the knowledge gained from solving one problem and applying it to a different but related problem. Transfer learning can be applied three major ways:

- Convolutional neural network (ConvNet) as a fixed feature extractor: In this method the last fully connected layer of a ConvNet is removed, and the rest of the ConvNet is treated as a fixed feature extractor for the new dataset.
- Fine-tuning the ConvNet: This method is similar to the previous method, but the difference is that the weights of the pre-trained network are fine-tuned by continuing backpropagation.
- Pre-trained models: Since modern ConvNets takes weeks to train from scratch, it is common to see people

release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning.

In this project, we used a pre-trained model for the transfer learning. The advantage of using a pre-trained model is that instead of building the model from scratch, a model trained for a similar problem can be used as a starting point for training the network. Many pre-trained models are available. In this project we used the COCO pre-trained model/checkpoints RFCN resnet, Faster RCNN resnet and SSD mobile net to compare the accuracy of object detection and localization. These models were used as an initialization checkpoint for training. The model was further trained with images of fruits dataset for achieving grocery identification. This fine-tuned model was used for grocery identification and counting each type detected[17].

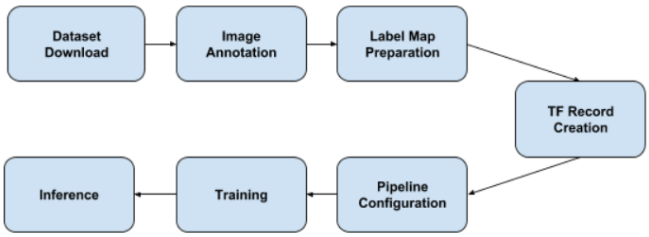


FIGURE 6
TENSORFLOW OBJECT DETECTION API WORKFLOW

- **Dataset download:**
  The dataset for fine-tuning the pre-trained model was prepared using over 100 images for each class of grocery type. We used a web scraper script [1] to download images from Google images for preparing the dataset.
- **Image annotation:**

i. Configuring the LabelImg[4] tool. Before starting with the annotation of images, the classes for labelling needs to be defined in the LabelImg/data/predefined_classes.txt file.
ii. Launch labelimg.exe and then select the dataset folder by clicking the OpenDir icon on the left pane.
iii. For each image that appears, draw a rectangular box across each grocery item by clicking the Create RectBox icon. These rectangular boxes are known as bounding boxes. Select the category as defined in predefined classes.txt from the drop-down list that appears.
iv. Repeat this process for every grocery item present in the image. Figure 7 below shows an example of a completely annotated image.

Once the annotations for an image are completed, save the image to any folder.
The corresponding eXtensible Markup Language (XML) files will be generated for each image in the specified folder. XML files contain the coordinates of the bounding boxes, filename, category, and so on for each object within the image. These annotations are the ground truth boxes for comparison. Figure below represents the XML file of the image above:

FIGURE 7
ANNOTATING IMAGE IN LABLEIMG

```
<annotation>
    <folder>test</folder>
    <filename>fridgefruit.jpg</filename>
    <path>E:/ADS/Project/images/test/fridgefruit.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>885</width>
        <height>442</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>apple</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>4</xmin>
            <ymin>170</ymin>
            <xmax>234</xmax>
            <ymax>391</ymax>
        </bndbox>
    </object>
    <object>
        <name>orange</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>176</xmin>
            <ymin>67</ymin>
            <xmax>417</xmax>
            <ymax>232</ymax>
        </bndbox>
    </object>
    <object>
        <name>lemon</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>274</xmin>
```

FIGURE 8
XML GENERATED AFTER ANNOTATING IMAGE IN FIGURE 7

- **Label map generation:**
  Each dataset requires a label map associated with it, which defines a mapping from string class names to integer class IDs. Label maps should always start from ID 1.
  We have used three classes for our model, the label map for this experiment file has the structure as shown in Figure 9.

- **TensorFlow records (TF record) generation:**
  TensorFlow accepts inputs in a standard format called a TFRecord file, which is a simple record-oriented binary format. Eighty percent of the input data is used for training and 20 percent is used for testing. The split dataset of images and ground truth boxes are converted to train and test TFRecords. Here, the XML files are

converted to csv, and then the TFRecords are created. Sample scripts for generation are available here

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

item {
  id: 2
  name: 'tomato'
}

item {
  id: 3
  name: 'yellow-bellpepper'
}
```

FIGURE 9
LABEL MAP USED FOR OUR MODEL

- **Pipeline configuration:**
  This section discusses the configuration of the hyperparameters, and the path to the model checkpoints, TF records, and label map. A detailed explanation is given in Configuring the Object Detection Training Pipeline. The following are the major settings to be changed for the experiment. At a high level, the config file is split into following parts:

i. In the model config, the major setting to be changed is the num_classes that specifies the number of classes in the dataset. We have set num_classes = 3
ii. The train config is used to provide model parameters such as batch size (set to 1), learning_rate(0.0003) and fine_tune_checkpoint. fine_tune_checkpoint field is used to provide path to the pre-existing checkpoint for pretrained RFCN model.
iii. The train_input_config and eval_input_config fields are used to provide paths to the TFRecords and the label map for both train as well as test data.

- **Training:**
  The final task is to assemble all that has been configured so far and run the training job Once setting the optimization parameters the training file is executed. By default, the training job will continue to run until the user terminates it explicitly. But we have used about 1000 epochs to train our model. The models will be saved at various checkpoints.

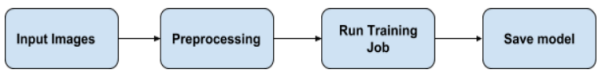


FIGURE 10
TRAINING WORKFLOW

```
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Starting Queues.
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:Recording summary at step 0.
INFO:tensorflow:global step 1: loss = 3.4343 (191.672 sec/step)
INFO:tensorflow:global step 1: loss = 3.4343 (191.672 sec/step)
INFO:tensorflow:global_step/sec: 0.00833416
INFO:tensorflow:global_step/sec: 0.00833416
INFO:tensorflow:global step 2: loss = 3.1871 (85.855 sec/step)
INFO:tensorflow:global step 2: loss = 3.1871 (85.855 sec/step)
INFO:tensorflow:Recording summary at step 2.
INFO:tensorflow:Recording summary at step 2.
INFO:tensorflow:global step 3: loss = 3.3203 (88.131 sec/step)
INFO:tensorflow:global step 3: loss = 3.3203 (88.131 sec/step)
INFO:tensorflow:global_step/sec: 0.0166692
INFO:tensorflow:global_step/sec: 0.0166692
INFO:tensorflow:Recording summary at step 3.
INFO:tensorflow:Recording summary at step 3.
```

FIGURE 11
TRAINING EPOCH LOSS RECORD

- **Inference:**
  The saved model is then used to test some input images to obtain bounding boxes on different grocery items and their respective count which can be integrated with a mobile/web application as an output to a user who wants to remotely identify the count of grocery items in the refrigerator.

CODE WITH DOCUMENTATION

LINK TO GITHUB

RESULTS

For image classification using Keras, following results are observed for Relu activation function and rmsprop as optimizer:

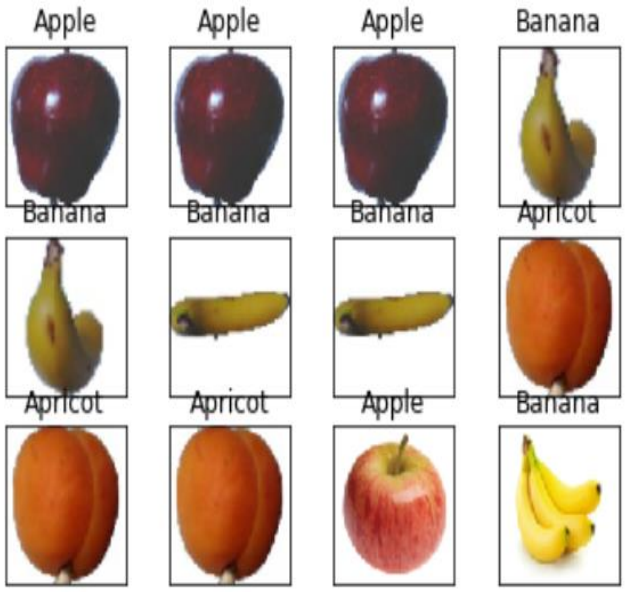
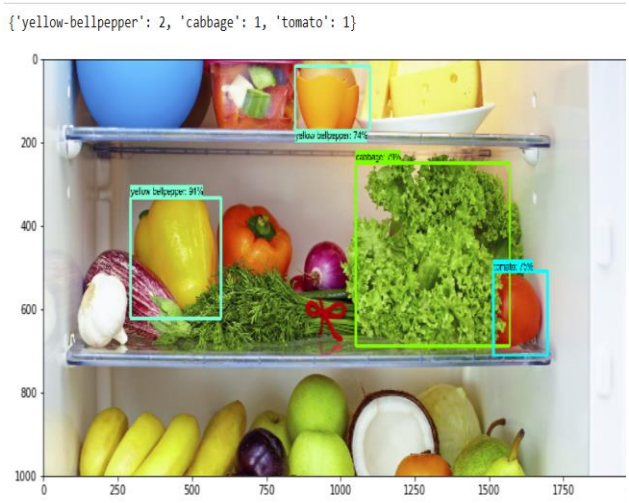
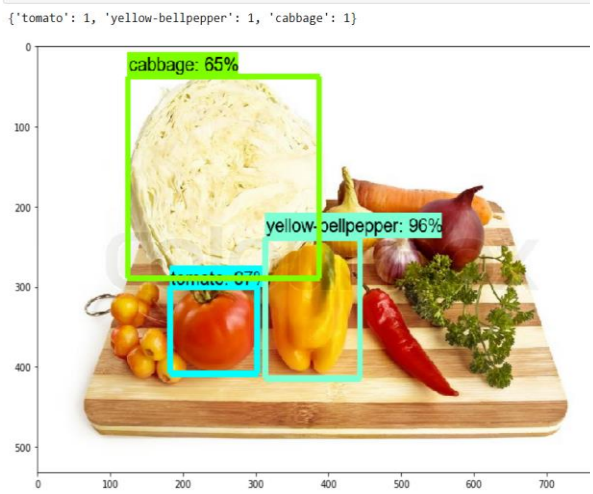


FIGURE 12
IMAGE CLASSIFICATION USING KERAS

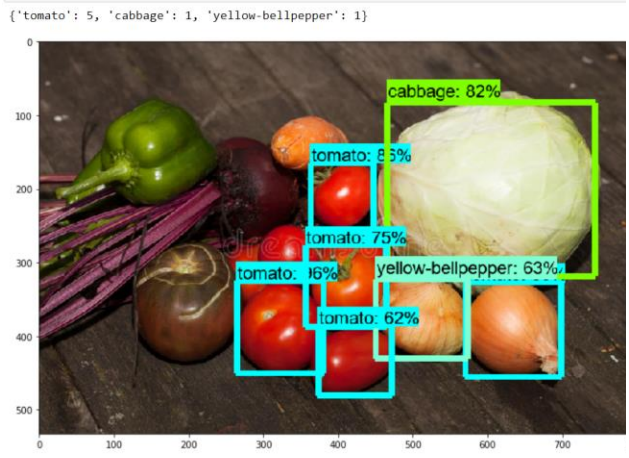
From the results, we observed that the grocery items were detected with a high level of accuracy. Following are a few test images that quantify our results.



(A)



(B)



(B)

FIGURE 13
GROCERY IDENTIFICATION RESULTS FOR OUR MODEL

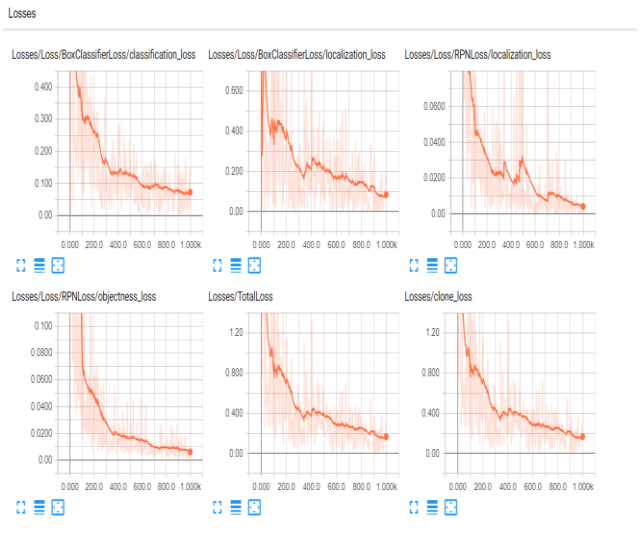


FIGURE 14
TENSORBOARD LOSS VARIATIONS WHILE MODEL TRAINING

CONCLUSION

We present our approach for an auto counting system of a smart refrigerator that can predict refrigerated items to an accuracy of 70% on our web scraped dataset. The result was achieved post an exhaustive study and understanding of the CNN and RCN models. Practical results for the CNN model were obtained by performing Image classification on the Fruits dataset using Keras which gave an accuracy of 99.93% after choosing the best parameters post hyperparameter tuning. Practical results for the RCNN model were obtained post performing Object detection on our web scraped dataset. We achieved this while maintaining above average detection time. We also demonstrated qualitative results to depict how a well-trained model on small dataset can generalize to an entirely independent environment.

In developing this system, we performed tweaking of the pretrained model on coco dataset set which is a Regional-Fully convoluted neural network. We trained this model on our own dataset which shows improvements in object detection as compared to Single shot detector (SSD) and faster RCNN model. Experimental results show that refrigerated items are detected on variety of refrigerator images when tested on this trained model.

Future work for our project involves training the model on a large variety of refrigerated items by web scraping relevant images while maintaining the same or more accuracy, less detection time and loss. We plan to use Tensorflow object detection API on android and iOS to further integrate the application on a cellular device.

ACKNOWLEDGEMENT

We would like to express our gratitude to Professor Nik Brown for guiding us during this project

REFERENCES

[1] https://github.com/hardikvasa/google-images-download
[2] https://link.springer.com/content/pdf/10.1007/978-3-319-65482-9_24.pdf
[3]https://software.intel.com/en-us/articles/traffic-light-detection-using-the-tensorflow-object-detection-api
[4] https://github.com/tzutalin/labelImg
[5]https://www.learnopencv.com/understanding-feedforward-neural-networks/
[6]https://www.learnopencv.com/deep-learning-using-keras-the-basics/
[7]https://www.learnopencv.com/neural-networks-a-30000-feet-view-for-beginners/
[8]https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/
[9]https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[10]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC501738
[12]https://github.com/tzutalin/labelImg
[13]https://medium.com/@rohitrpatil/how-to-use-tensorflow-object-detection-api-on-windows-102ec8097699
[11]https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883
[14]https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9
[15]https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions
[16]https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
[17]https://github.com/tensorflow/models/tree/master/research/object_detection
[18]https://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/
[19]https://arxiv.org/abs/1605.06409
[20]https://arxiv.org/abs/1605.06409
[21]https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721
[22]https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491