Machine Learning Engineer Nanodegree

Capstone Project

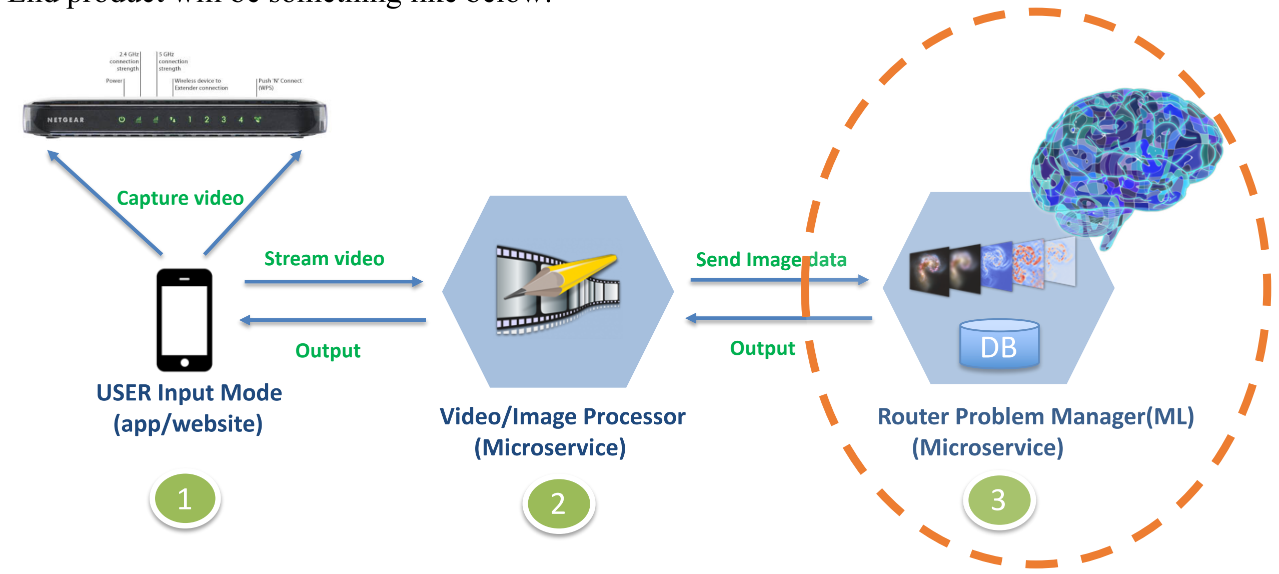
Ajay Singh
Dec 26th, 2018

Definition

Project Overview

Creating machine learning based application which can capture frames from video and do analysis of that frames picture. So, what is Video about – almost everyone has internet on their home, and we use router/modem/gateways/cable boxes for getting internet connection/wifi/tv services. And we all get sometime these problems with our home devices and call customer care for that. So, my proposal is to create software which will capture a video of device (router/modem/gateways/cable boxes) and identify which lights are on/off and what color they have. Based on that data, we can decide what issue we have with that device like ethernet cable is loose or internet is not available and notify backend system to fix it or call customer care.

End product will be something like below:


Capture video
Stream video
Output
USER Input Mode (app/website)
1
Video/Image Processor (Microservice)
2
Send Image data
Output
DB
Router Problem Manager(ML) (Microservice)
3

I am trying to work on **module -3**, working on architecture solution for identifying network device light patterns and based on that try to resolve issue automatically without customer support.

Something like below problem/solution –

https://link.springer.com/chapter/10.1007/978-3-540-74853-3_16

https://pure.qub.ac.uk/portal/files/17844756/machine.pdf

Problem Statement

Home Router issues like ethernet cable loose, no internet connection, wifi off etc. So customers call customer care which takes time and also some customer care support person to look and understand problem. Customers get frustrated most of the time with wait time/explanations and companies have to provide some support person on call even it is very small problem. With Machine learning we can achieve better User experience and save customer time and also save money/time for Company too.

Business Problem:

- ❑ Long wait times
- ❑ Unnecessary expensive dispatches
- ❑ Lack of digital Capabilities
- ❑ Unoptimized resource utilization

Metrics

As it is multi-classification problem, I am using loss metrics as **categorical_crossentropy** , Optimizer as **rmsrop** and overall performance metrics for CNN model – **metrics.**

Like example –

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

categorical_crossentropy

keras.losses.categorical_crossentropy(y_true, y_pred)

RMSprop

keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned). This optimizer is usually a good choice for recurrent neural networks.

Accuracy Score

Defined as % labels correctly classified when comparing model prediction vs actual. This metric will be computed on the training dataset and the validation set which is the population for which we have labels for the images. This is the secondary metric that will be computed and shared but the select of the optimal model will be based on minimizing log loss. It is expected that the model with the lowest log loss will also have one of the best accuracy score.

Analysis

Data Exploration And Exploratory Visualization

Datasets and Inputs

Dataset :- image of 1066x150 px of device as input.

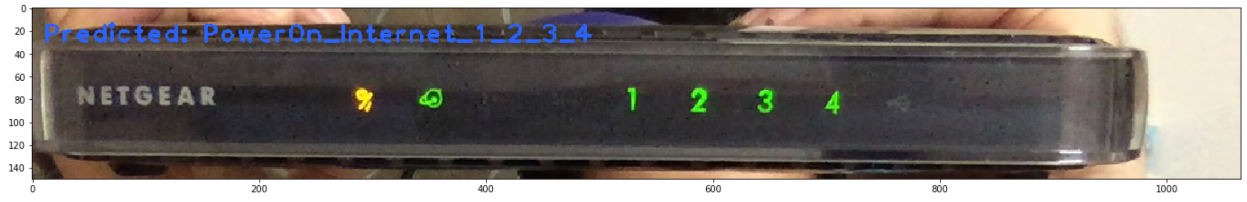
Classes images(approx.) – 30x30 px power on led
30x30 px wifi on led
30x30 px internet on led
30x30 px 1,2,3,4 on led

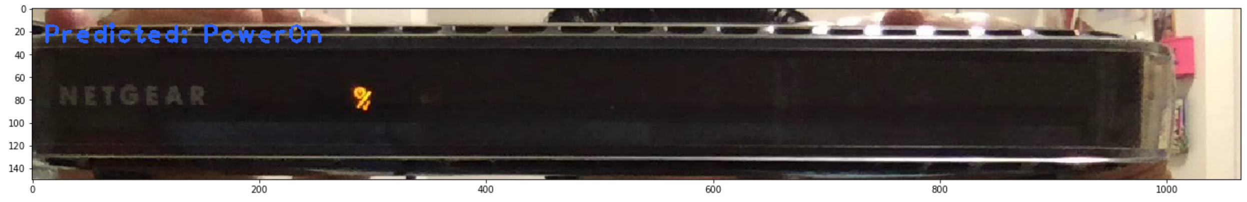
As this idea needs lot of data, I decided to try small scale proof-of-concept with one Netgear router first. Actual solutions will have several routers/gateway/cable-boxes and modems. I took many pictures with guided box as scale of 1066*150 with different lights on/off. I divided into three categories – **power_on, internet_on, wifi_on,2.4 channel, 5 channel, ethernet port(1,2,3,4).**

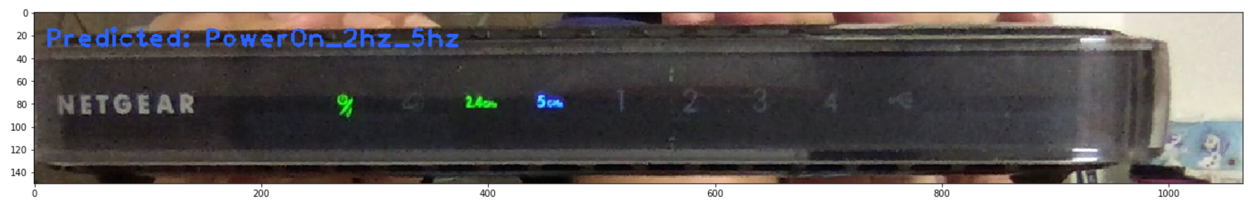
My dataset looks like below:

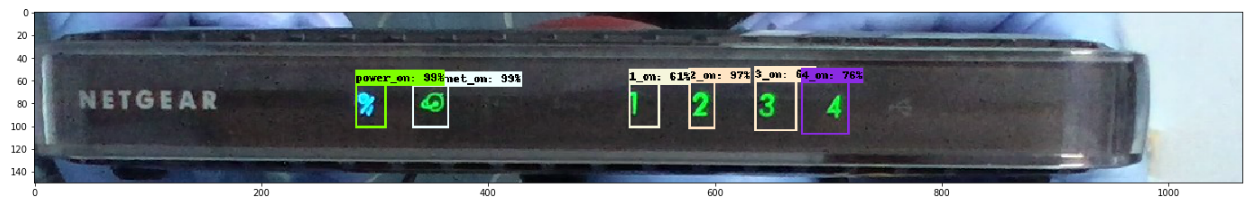
```
├── test
│   ├── PowerOn
│   ├── PowerOn_2hz_5hz
│   └── PowerOn_Internet_1_2_3_4
├── train
│   ├── PowerOn
│   ├── PowerOn_2hz_5hz
│   └── PowerOn_Internet_1_2_3_4
└── valid
    ├── PowerOn
    ├── PowerOn_2hz_5hz
    └── PowerOn_Internet_1_2_3_4
```

Some Image Examples: -







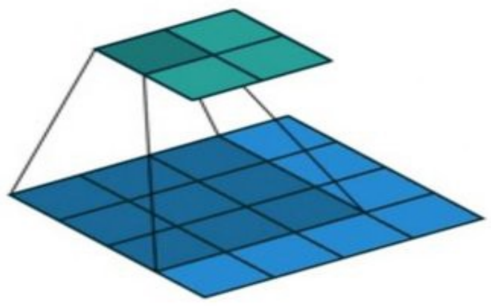


(above one with Tensorflow API)

Algorithms and Techniques

Deep Learning is becoming a very popular subset of machine learning due to its high level of performance across many types of data. A great way to use deep learning to classify images is to build a convolutional neural network (CNN). The Keras library in Python makes it pretty simple to build a CNN.

A convolution multiplies a matrix of pixels with a filter matrix or 'kernel' and sums up the multiplication values. Then the convolution slides over to the next pixel and repeats the same process until all the image pixels have been covered.



To create such CNN model, it is necessary to go through the following phases:

1. model construction
2. model training
3. model testing
4. model evaluation

As I don't have much data and it is custom data (cannot find on internet, took from myself), I tried multiple approaches to find out results and tried to evaluate results.

```
├── bottle_neck_cnn (VGG16 Bottle-Neck approach with Image Augmentation)
│   ├── RouterLightClassifier.html
│   ├── RouterLightClassifier.ipynb
│   ├── bottleneck_fc_model.h5
│   ├── bottleneck_features_train.npy
│   ├── bottleneck_features_validation.npy
│   ├── class_indices.npy
│   └── data
├── cnn_from_scratch (CNN approach with/without Image Augmentation)
│   ├── RouterLightClassifier-without-augmentation.ipynb
│   ├── RouterLightClassifier.ipynb
│   ├── data
│   ├── first_try.h5
│   └── weights.best.from_scratch1.hdf5
└── tf_api_code (Using Tensorflow API and train model with label/images)
    ├── data
    ├── models
    ├── nwd-tf
    ├── object_detection
    ├── object_detection_tutorial.ipynb
    └── test_images
```

Benchmark

I have data in my project's Jupyter Notebook. Because of less data accuracy is very high but we can see results are not 100% accurate. I used three approaches – CNN from scratch, CNN from Bottleneck (VGG16) and Tensorflow APIs. Full data is on Jupyter Notebooks.

Methodology

Data Preprocessing

I used image size as 1066x150 pixel. Train set with total 1500 images, Valid with 300 Images and Test with 150 Images.

I used many libraries and function for Image processing like –

keras.preprocessing - loads RGB image as PIL.Image.Image type

img_to_array(img) - convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)

numpy expand_dims(x, axis=0) - convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor

I also used Data Augmentation for image data to get good results, like –

```
# 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)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

I also used sklearn, numpy modules to separate out train/valid and test data for CNN from scratch.

```
def load_dataset(path):
    data = load_files(path)
    img_files = np.array(data['filenames'])
    img_targets = np_utils.to_categorical(np.array(data['target']), 8)
    return img_files, img_targets
```

```
# load train, test, and validation datasets
train_files, train_targets = load_dataset('./data/train')
valid_files, valid_targets = load_dataset('./data/valid')
test_files, test_targets = load_dataset('./data/test')

# load list of dog names
router_classification_names = [item[13:-1] for item in sorted(glob.glob("./data/train/*/"))]

# print statistics about the dataset
print('There are %d total router classification categories.' % len(router_classification_names))
print('There are %s total router classification images.\n' % len(np.hstack([train_files, valid_files, test_files])))
print('There are %d training router classification images.' % len(train_files))
print('There are %d validation router classification images.' % len(valid_files))
print('There are %d test router classification images.' % len(test_files))
```

```
There are 3 total router classification categories.
There are 1950 total router classification images.

There are 1500 training router classification images.
There are 300 validation router classification images.
There are 150 test router classification images.
```

Implementation

Model construction depends on machine learning algorithms. In this projects case, it was neural networks.

Such an algorithm looks like:

1. begin with its object: model = Sequential() - Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer.
2. then consist of layers with their types: model.add(*type_of_layer()*) - to add layers to our model.
3. after adding a sufficient number of layers the model is compiled. At this moment Keras communicates with TensorFlow for construction of the model. During model compilation it is important to write a loss function and an optimizer algorithm. It looks like: model.comile(loss= 'name_of_loss_function', optimizer= 'name_of_opimazer_alg') The loss function shows the accuracy of each prediction made by the model.
4. **Max Pooling 2D** layer is pooling operation for spatial data. Numbers 2, 2 denote the pool size, which halves the input in both spatial dimensions.
5. Activation is the activation function for the layer. The activation function we will be using for layers is the ReLU, or Rectified Linear Activation. This activation function has been proven to work well in neural networks.
6. There is also 'Flatten' layer. Flatten serves as a connection between the convolution and dense layers.
7. 'Dense' is the layer type we will use in for our output layer. Dense is a standard layer type that is used in many cases for neural networks.
8. The activation is 'softmax'. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

Before model training it is important to scale data for their further use.

After model construction it is time for **model training.** In this phase, the model is trained using training data and expected output for this data.

It's look this way: model.fit(training_data, expected_output).

Progress is visible on the console when the script runs. At the end it will report the final accuracy of the model.

Once the model has been trained it is possible to carry out **model testing.** During this phase a second set of data is loaded. This data set has never been seen by the model and therefore it's true accuracy will be verified.

After the model training is complete, and it is understood that the model shows the right result, it can be saved by: model.save("name_of_file.h5").

Finally, the saved model can be used in the real world. The name of this phase is **model evaluation**. This means that the model can be used to evaluate new data.

Image Augmentation

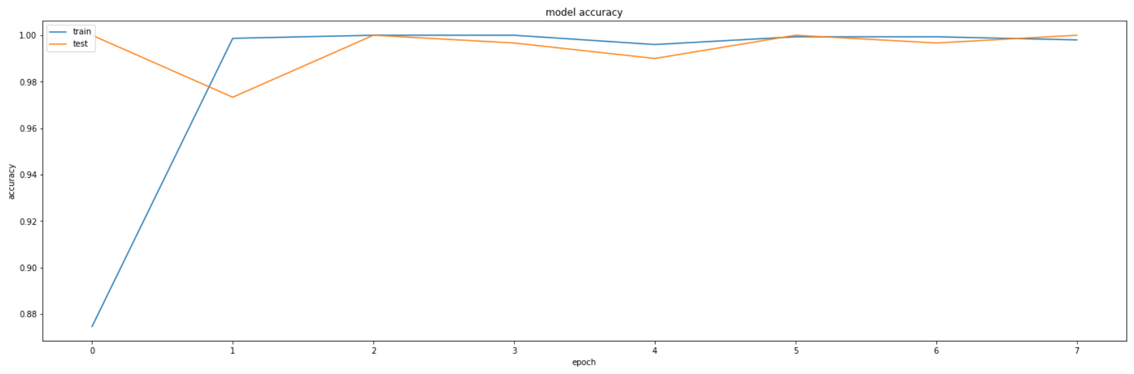
Using little data is possible when the image is preprocessing with Keras ImageDataGenerator class. This class can create a number of random transformations, which helps to increase the number of images when it is needed.

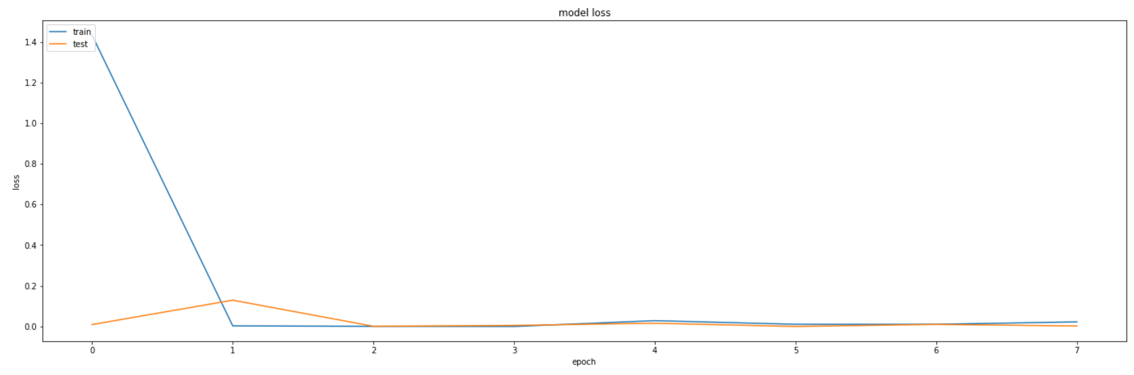
CNN From Scratch: -

Without Image Augmentation:

CNN Model: -

```
model = Sequential()

model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
                 input_shape=(1066, 150, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.7))
model.add(Dense(8, activation='softmax'))

model.summary()

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```



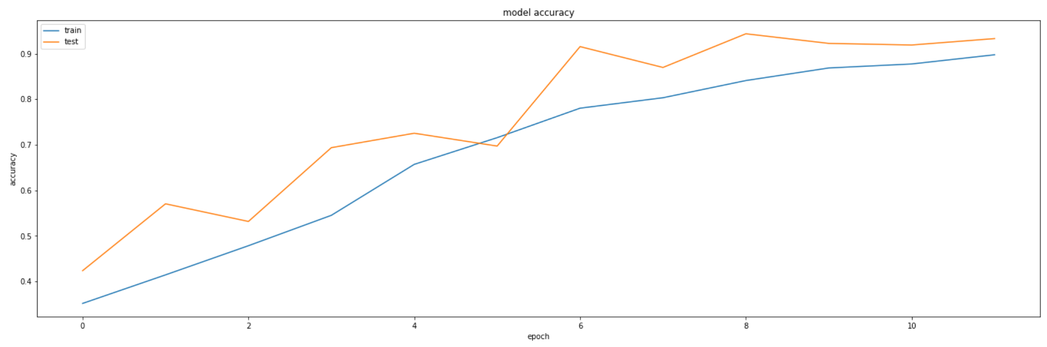


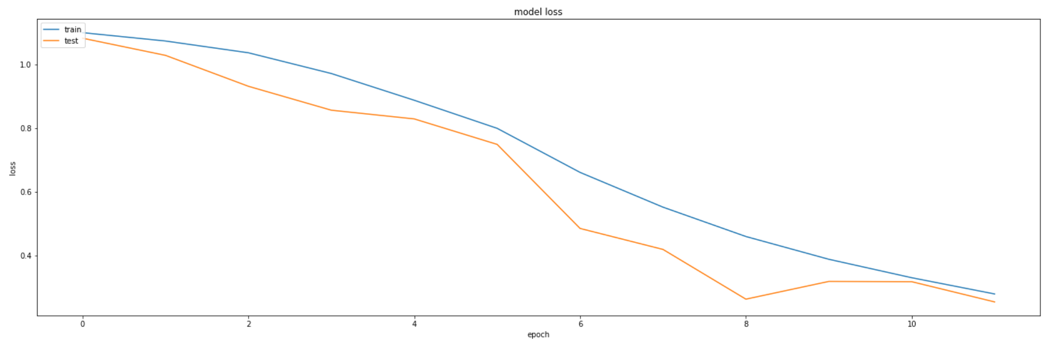
With Image Augmentation:

CNN Model: -

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(GlobalAveragePooling2D(input_shape=input_shape))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(classes_num,activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

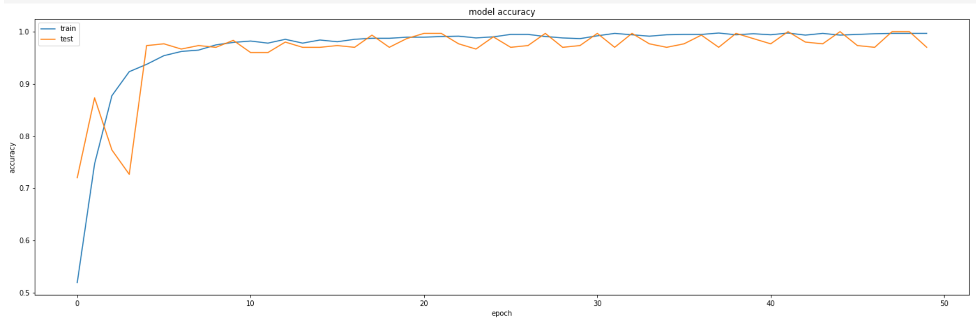
```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

global history
history = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)
```

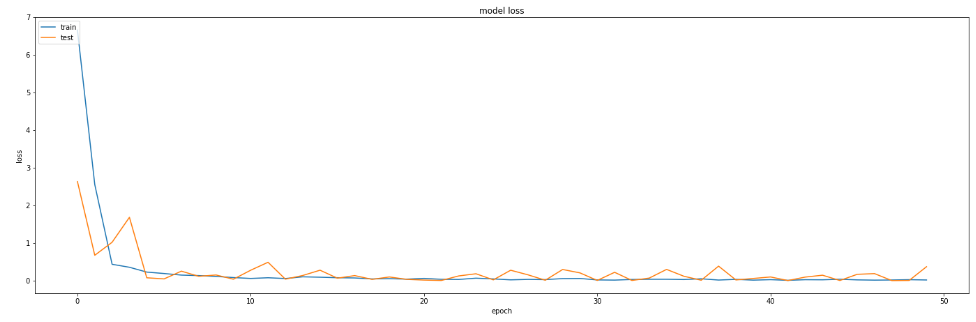




Bottle-Neck CNN From Scratch: -

```
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy', metrics=['accuracy'])
```





Tensorflow API: -

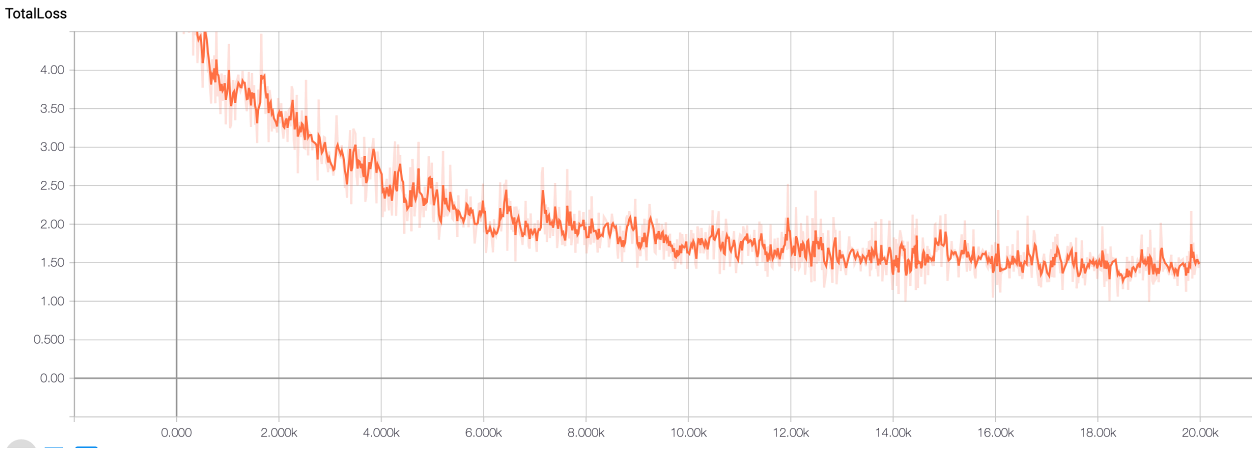
I experimented with training a custom object detector using TensorFlow's object detection API to detect three labels of router. The files for this project can be found in the under tf_api_code project folder. I followed the Raccoon detector tutorial for this project.

Training a custom object detector takes place in mostly 3 parts:

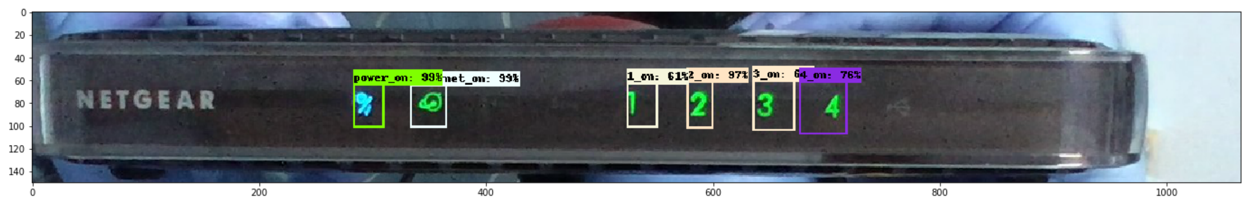
1. Creating the data set
2. Training an object detector model on your data set
3. Testing the results of your custom object detector model

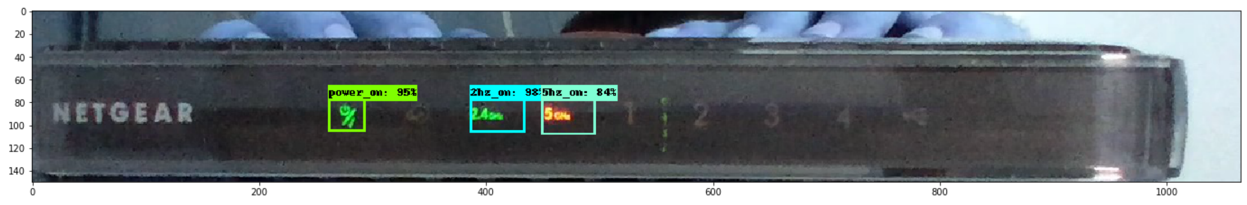
Here are the results from my training and evaluation jobs. In total, I ran it over about one hour/22k steps with a batch size of 24 but I already achieved good results in about 40mins.

This is how the total loss evolved:



Some of good Results





Refinement

Bulk of my time in this project was spent on collecting Data images but still I think I don't have sufficient amount of data. Data Augmentation helped me lot for CNN models (non Tensorflow API approach). I tried multiple combination of CNN models but because of less amount of data, they all almost provided me similar results.
My goal was to identify lights of router, so I decided to try Tensorflow API approach and even with less data it provided me good results.

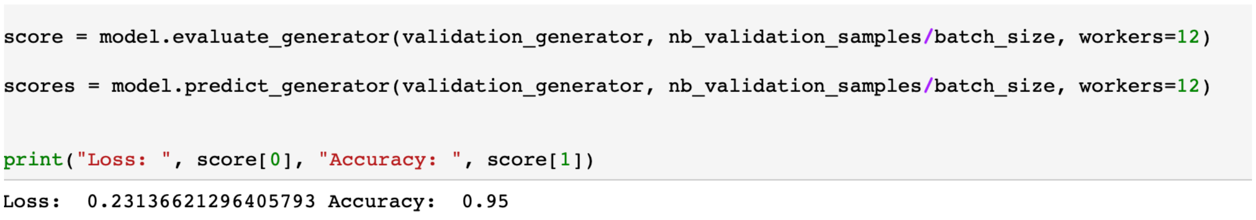
Results

Model Evaluation and Validation

Evaluation metrics explain the performance of a model. An important aspects of evaluation metrics is their capability to discriminate among model results. I used Test Accuracy and validation Accuracy as Evaluation metrics.

Though I got very high accuracy but test results was not near to that high score.
Like below –

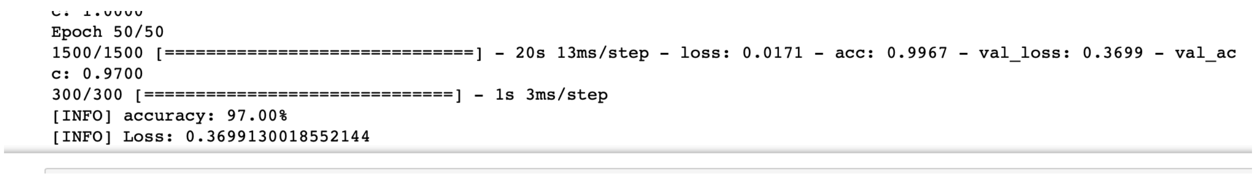
CNN from scratch



```
score = model.evaluate_generator(validation_generator, nb_validation_samples/batch_size, workers=12)

scores = model.predict_generator(validation_generator, nb_validation_samples/batch_size, workers=12)


print("Loss: ", score[0], "Accuracy: ", score[1])
```

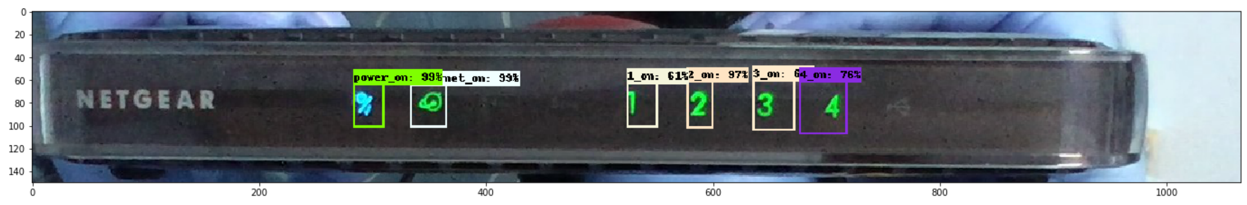
```
Loss:  0.23136621296405793 Accuracy:  0.95
```

With Bottleneck approach



```
c: 1.0000
Epoch 50/50
1500/1500 [==============================] - 20s 13ms/step - loss: 0.0171 - acc: 0.9967 - val_loss: 0.3699 - val_ac
c: 0.9700
300/300 [==============================] - 1s 3ms/step
[INFO] accuracy: 97.00%
[INFO] Loss: 0.3699130018552144
```

TensorFlow API -



Justification

I have tried many different combinations of network parameters, optimization function and regularization to tune the model to its final setting. I see little incremental benefit of continuing this optimization further and feel comfortable stopping at this point. Also less data was another problem for me.
I tried three approaches and in the end based of amount of limited data , I got good results.

Conclusion

Free-Form Visualization

Reflection

Gather images as many for different status(lights) of device. Process images and create three sets – train, validate and test. Use CNN with VGG16 model and train model with train set of images. Validate with validate set and run test cases with test sets. Measure accuracy of prediction and change CNN model designs and if necessary, try other benchmark models. Finalize model and store for future use.

Understanding of deep learning, convolution neural networks and Tensorflow through courses and books. When I started I had little to no understanding of this field. This was the pre-phase to give me the basic tools to get started.

Pre-Processing of the dataset – Learning about image processing was the next phase. Deciding what pre-processing must be done with the first iteration (resize image, flatten to rgb, converting to array data) vs what can be tried later after I get comfortable (image augmentation, colored images)

The depth of neural network materially affects accuracy. As the network gets deeper, its ability to learn finer features improves. This is intuitive. However, I found that increasing the number of neurons in each layer didn’t always have a positive effect on accuracy and sometimes decreased accuracy. The best setting for me was a deep network with small number of neurons in each layer

Improvement

I believe that if I collect more devices light combination images and increase image data size, I can achieve very good CNN model which we can rely to accurately find which light combination are ON on network device. I also think that collecting different LED lights separately like

1,2,3,4, internet, power etc. and then train separately and use RNN or Tensorflow API, we can achieve good results.
Use of other more modern techniques that can help reduce overfitting, Better input pre-processing to help the model**,** Use of external data (In my case, I did not find any)