42028: Deep Learning and Convolutional Neural Network

Assignment -2

Date of Submission: 10th June 2020

Student Name: Nazmul Kaonine

Student ID: 13300912

<u>Contents:</u>

Using CNN in

I) Customizing Baseline for image classification

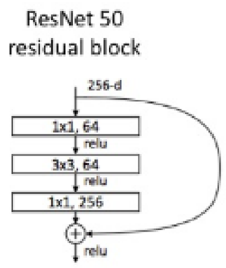
II) Implementing Faster-RCNN & SDD for object detection

III) Analysis of above

1. Introduction:

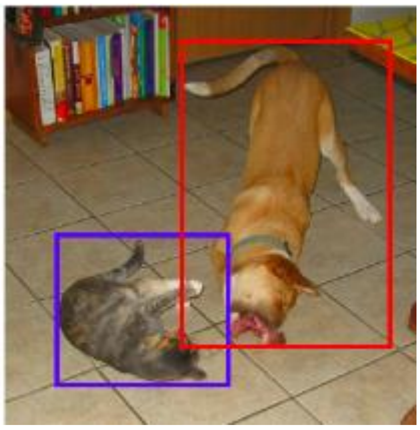
This report focuses on the Fruits-360 database of fruits and vegetables and the implementation of three different architectures. The development of a baseline CNN architecture, an optimized model of the same for image classification and an object detection architecture is presented. To achieve this, ResNet-50, a model that was used in 2015 to win the Imagenet competition have been implemented for image classification. For object detection, Faster R-CNN and SSD (Single Shot Detector) was used. To observe the combination with the highest accuracy, different parameters have been tweaked and the outlining reasons behind high and low accuracy scores have been investigated.

ResNet50



This model was the winner ofImageNet challenge in 2015

Object Detection:



Faster R-CNN:

Presented in 2015, it is one of the most famous object detection algorithms.

SSD:

An object detection algorithm mentioned in ECCV paper 2016 with more than 2000 citations because of its performance and accuracy.

2. Dataset:

The Fruit-360 dataset was made by using a Logitech C920 webcam to create short movies on fruits. This was one of the best webcams available at the time. The images were customized accordingly for use. Different varities of the same fruit fell into separate classes that was distributed in separate folders with the respective class name. The dataset used has been provided inside the github link: https://github.com/Horea94/Fruit-Images-Dataset. The multi-fruits set has been disregarded as per instructions.

The deconstruction of the dataset is as follows:

Total images: 90483

Training set: 67,692

Testing set: 22,688 testing divided into 27107 for testing & 13554 for validation (60% and 40%)

Classes: 131

Images size: 100x100 pixels

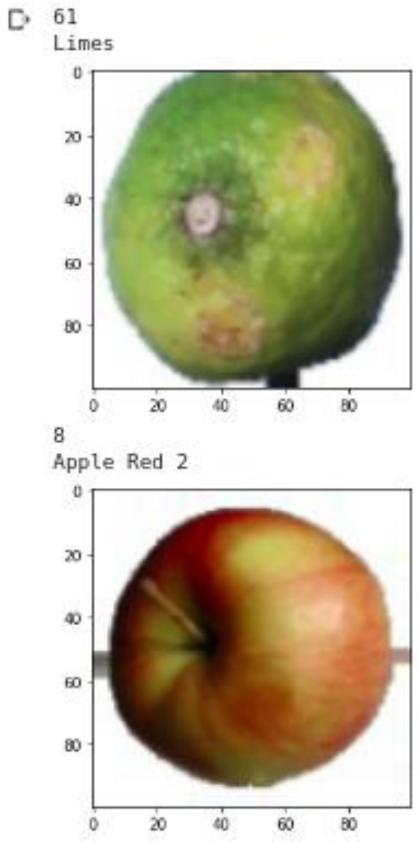
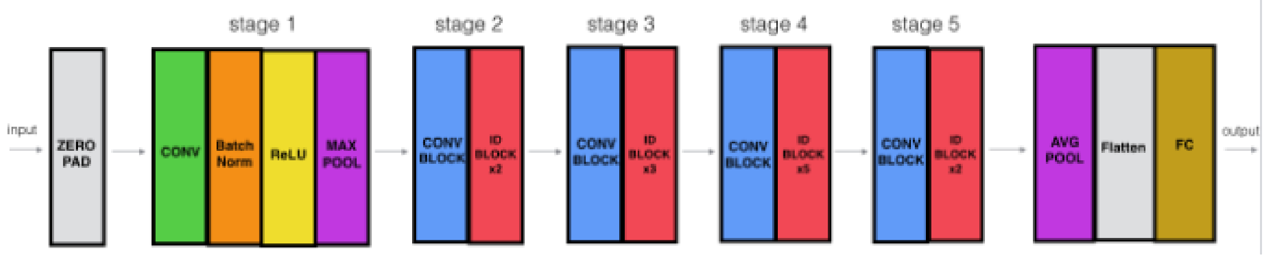


Fig:1 Sample images of a 'limes' and 'apple red 2'

3. Proposed CNN Architecture for Image Classification:

a. Baseline architecture used

The baseline architecture used here is ResNet50. Other architectures attempted are ResNet50v2, ResNet101, VGG19 & VGG16 but ResNet50 was the fastest to train. ResNets utilize skip connection features that allows it to jump over some layers.



ResNet-50 Model

ResNet50 is a deep residual network that is simply 50 layers deep. This model consists of 5 stages and each stage contains a convolution and identity block. Each convolution and identity block has 3 convolution layers each.

```
model2 = models.Sequential()
model2.add(conv_base)
model2.add(layers.Dense(131, activation='softmax'))
```

Transfer Learning was used with include_top=False to remove the fully connected layers and final pooling layer.

Additionally, the baseline architecture used contains an output layer of activation “softmax” that represents 131 output classes. The optimizer used is “Rmsprop” as this is an adaptive optimizer unlike SGD that will decay over time from manual parameters (learning rate, momentum, etc.).

b. Customized architecture

The customized architecture adds 6 more layers to the baseline architecture and has proven to improve the accuracy and performance of the previous architecture.

```
model = models.Sequential()
model.add(conv_base)
model.add(GlobalAveragePooling2D())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(.25))
model.add(layers.BatchNormalization())
model.add(layers.Dense(131, activation='softmax'))
```

Although transfer learning allows us

to start off with a model that was trained previously on millions of images, fine-tuning that model on the basis of the existing smaller dataset is necessary. To achieve this, re-training the model on the last few layers while keeping the base layers intact is key.

Convolution base (ResNet50) was kept non-trainable. GlobalAveragePooling2D was used first and then a Dense layer followed by a dropout layer. Batch Normalization was performed before getting the classified output.

Activation functions used is "relu" for each layer while the output layer is "softmax".

c. Assumption/intuitions

The graph was showing to be under-fitting. To optimize that to a certain extent, layers were added along with dropouts to prevent too much fitting as well. For faster computation of classification, an initial GlobalAveragePooling2D layer was added. This will reduce trainable parameters and over-fitting by taking average output of each feature map from the previous layer by a factor of 2. It has no trainable parameters itself, and so enables faster computation.

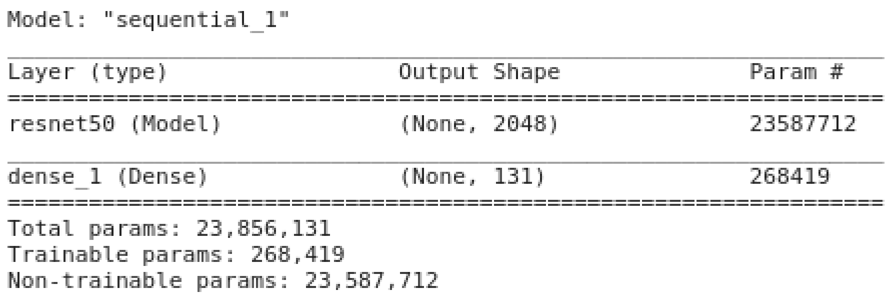
One dropout layers is added after the dense layer to avoid over-fitting. The dropout layer basically sets the inputs to 0 at a rate mentioned in brackets during each training step. These layers are not affected by trainable=True/ False.

The model was shown to be under-fitting and therefore one dense layers with 256 neurons was added. The reason behind the number of neurons is due to the fact that although many inputs were fed to the network, the outputs of current layer is decreasing with depth.

To stabilize the inputs, Batch Normalization was used which will increase the computation and performance even further.

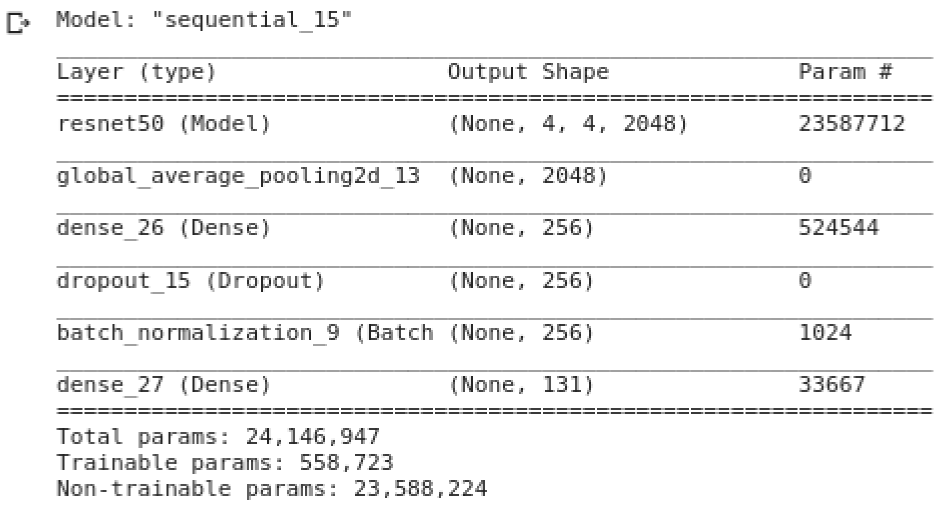
The activation function "Relu" was used to avoid back-propagation errors. "Softmax" was used in the output classifier as this has a probabilistic output.

d. Model Summary



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
resnet50 (Model)             (None, 2048)              23587712  
_________________________________________________________________
dense_1 (Dense)              (None, 131)               268419    
=================================================================
Total params: 23,856,131
Trainable params: 268,419
Non-trainable params: 23,587,712
```

model.summary() of baseline



```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
resnet50 (Model)             (None, 4, 4, 2048)        23587712  
_________________________________________________________________
global_average_pooling2d_13  (None, 2048)              0         
_________________________________________________________________
dense_26 (Dense)             (None, 256)               524544    
_________________________________________________________________
dropout_15 (Dropout)         (None, 256)               0         
_________________________________________________________________
batch_normalization_9 (Batch (None, 256)               1024      
_________________________________________________________________
dense_27 (Dense)             (None, 131)               33667     
=================================================================
Total params: 24,146,947
Trainable params: 558,723
Non-trainable params: 23,588,224
```

model.summary() of customized Architecture

4. CNN Architecture for Object Detection:

a. Faster RCNN

This object detection system basically contains a deep convolutional network module that proposes regions and passes it to a Fast-RCNN network that generates feature maps. The region proposals are made by RPN (Region Proposal Networks). The predicted object proposals are passed through a RoI pooling layer that performs max pooling on the inputs. As a result the image is classified and bounding box offset values are predicted.

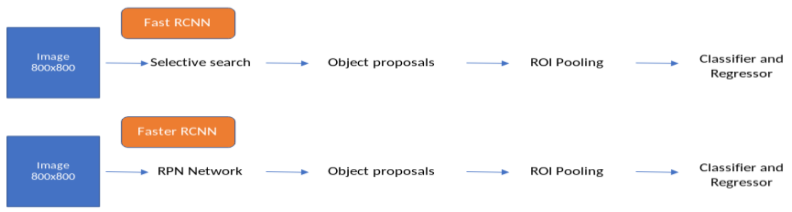


Fig: Differential architecure of Fast RCNN and Faster RCNN

b. SSD (Single Shot detector)

This architecture consists of a base network, some feature layers and prediction layers. The model depends on default boxes (bounding boxes) to perform its task. It contains 8732 default boxes and the main task is to select which boxes to use for the unknown image and choose offsets. Thus it leads to its prediction.

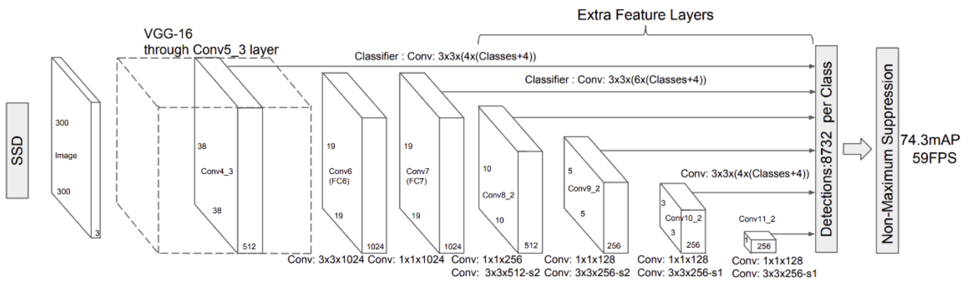


Fig: The SSD pipieline for VGG16

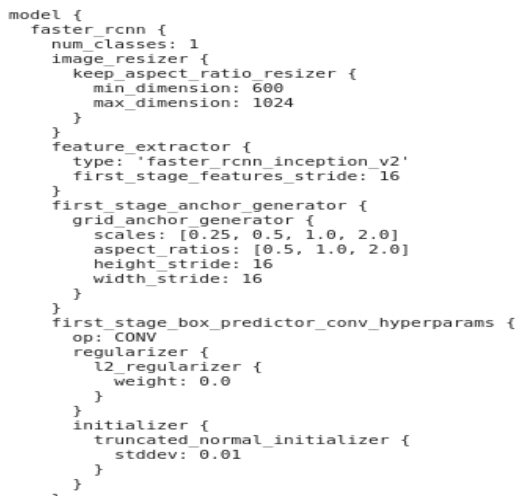
c. Assumption/intuitions

Faster R-CNN has a better feature extractor than most others due to its inception base. To achieve a good performance the number of proposals can be decreased while keeping the change in accuracy to a minimum.

SSD is known for conducting object detection on larger objects. The given dataset has blood cells that does not have tiny objects so no data augmentation was necessary. SSD chooses its default boxes very carefully depending on the size, aspect ratios and other image attributes. It uses most of its computation time on the base layer which in this case is **inception v2**. It also assists in giving great accuracy and speed.

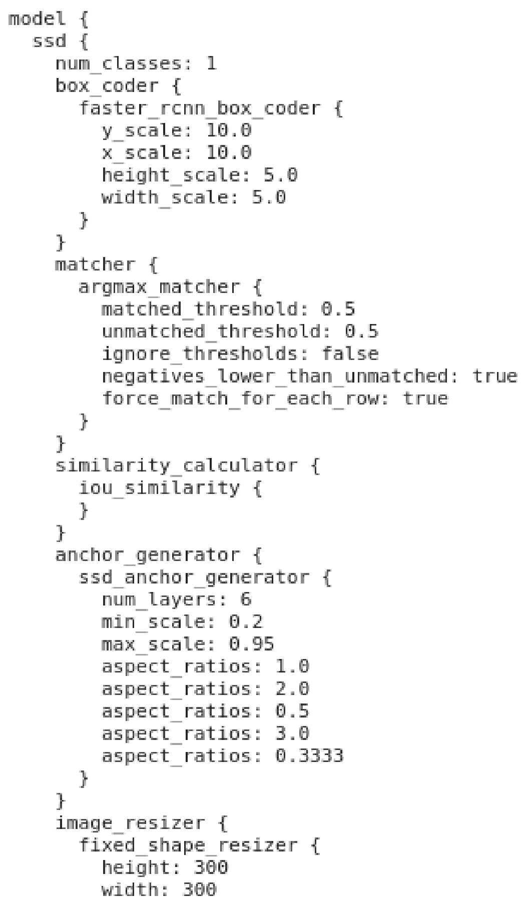
d. Model Summary

Faster R-CNN:



```
model {
  faster_rcnn {
    num_classes: 1
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
```

SSD:



```
model {
  ssd {
    num_classes: 1
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
      }
    }
    similarity_calculator {
      iou_similarity {
      }
    }
    anchor_generator {
      ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
      }
    }
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
```

5.Experimental results and discussion:

a. Experimental settings:

I. Image Classification:

Hyper-Parameters:
batch_size: 32
Optimizer: “Rmsprop”
loss='categorical_crossentropy',

Number of hidden layers: 1 layer with 256 neurons
Dropout: 1 layer with rate=0.25
Activation functions: “relu” & “softmax”

Data Augmentation: rescale=1. / 255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True

Transfer Learning: Yes
Pretrained model: ResNet50 from Keras with
include_top=False

Other settings: Early Stopping & CheckPointing.

ii. Object Detection:

Faster R-CNN:

Transfer Learning: Yes, faster_rcnn_inception_v2'
batch_size: 12
optimizer: momentum_optimizer
learning_rate: manual_step_learning_rate initial_learning_rate: 0.0002
step_900000 learning_rate: .00002
regularizer: l2_regularizer weight: 0.0
Data Augmentation: random horizontal flip

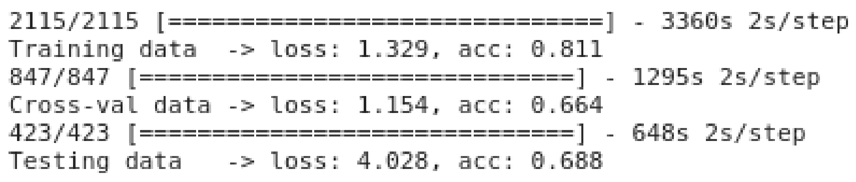
SSD:

Transfer Learning: Yes, 'ssd_mobilenet_v2'
min_depth: 16
activation: RELU_6
regularizer: l2_regularizer weight: 0.00004
batch_size: 12
optimizer: RmsProp
learning rate=0.004
Data Augmentation: ssd random crop

b. Experimental results:

I. Image classification:

Performance on baseline/standard architecture

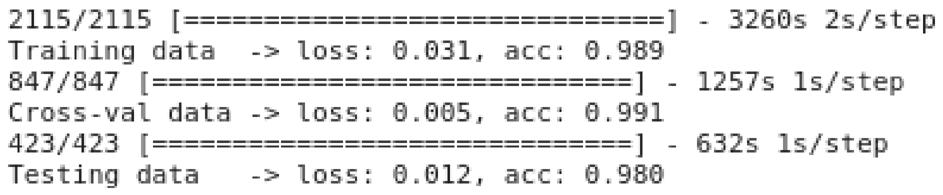


```
2115/2115 [==============================] - 3360s 2s/step
Training data  -> loss: 1.329, acc: 0.811
847/847 [==============================] - 1295s 2s/step
Cross-val data -> loss: 1.154, acc: 0.664
423/423 [==============================] - 648s 2s/step
Testing data   -> loss: 4.028, acc: 0.688
```

Performance on baseline was average as shown by the accuracy and loss of train, validation and test below:

Type	**Accuracy**	**Loss**
Train	0.811	1.329
Validation	0.664	1.154
Test	0.688	4.028

Performance on customized architecture



```
2115/2115 [==============================] - 3260s 2s/step
Training data  -> loss: 0.031, acc: 0.989
847/847 [==============================] - 1257s 1s/step
Cross-val data -> loss: 0.005, acc: 0.991
423/423 [==============================] - 632s 1s/step
Testing data   -> loss: 0.012, acc: 0.980
```

Performance on customized architecture was close to maximum as shown by the accuracy and loss of train, validation and test below:

Type	**Accuracy**	**Loss**
Train	0.989	0.031
Validation	0.991	0.005
Test	0.980	0.012

ii. Object Detection:

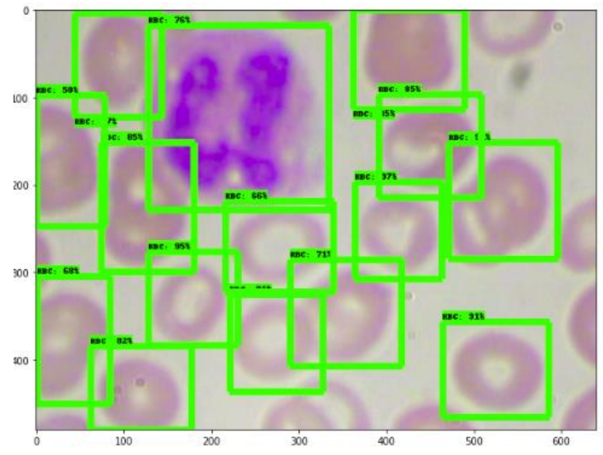


Fig: After detection using SSD

Performance on Faster-RCNN

DetectionBoxes_Precision/mAP	0.62391454
DetectionBoxes_Precision/mAP (large) 0.6559187	0.6559187
DetectionBoxes_Precision/mAP (medium)	0.56038404
DetectionBoxes_Precision/mAP (small)	-1.0
DetectionBoxes_Precision/mAP@.50IOU	0.9048928
DetectionBoxes_Precision/mAP@.75IOU	0.8035592
DetectionBoxes_Recall/AR@1	0.07411168
DetectionBoxes_Recall/AR@10	0.57563454
DetectionBoxes_Recall/AR@100	0.72944164
DetectionBoxes_Recall/AR@100 (large)	0.74852943
DetectionBoxes_Recall/AR@100 (medium)	0.68688524
DetectionBoxes_Recall/AR@100 (small)	-1.0,

Other settings after **Global step 1000:**
Loss/BoxClassifierLoss/classification_loss = 0.21763116,
Loss/BoxClassifierLoss/localization_loss = 0.20062903 Loss/RPNLoss/localization_loss = 0.23809999 Loss/RPNLoss/objectness_loss = 0.15345734
Loss/total_loss = 0.80981755
learning_rate = 0.0002
loss = 0.80981755

Analysis: Computation is slow but precision is high.

Performance on SSD or any object detector

DetectionBoxes_Precision/mAP	0.35251582
DetectionBoxes_Precision/mAP (large)	0.40314654
DetectionBoxes_Precision/mAP (medium)	0.31496614
DetectionBoxes_Precision/mAP (small)	-1.0
DetectionBoxes_Precision/mAP@.50IOU	0.6860004
DetectionBoxes_Precision/mAP@.75IOU	0.35604975
DetectionBoxes_Recall/AR@1	0.038071066,
DetectionBoxes_Recall/AR@10	0.37817258
DetectionBoxes_Recall/AR@100	0.55888325

DetectionBoxes_Recall/AR@100 (large)	0.6455882
DetectionBoxes_Recall/AR@100 (medium)	0.36557376
DetectionBoxes_Recall/AR@100 (small)	-1.0

Other settings after **Global step 1000:**

Loss/classification_loss = 4.882548
Loss/localization_loss = 0.7998091
Loss/regularization_loss = 0.24674812
Loss/total_loss = 5.929105
learning_rate = 0.004

Analysis: Computation is very fast (100 steps-40secs)
but precision is lower than Faster R-CNN.

III. Discussion:

Reasons behind image classifier accuracy:

ResNet Baseline model is pretrained on imagenet dataset that has a different number of classes. Fine-tuning the model to the current dataset allowed the pre-trained weights to be used in collaboration with the deep resnet layers. Also, it consists of 50 layers and contains residual blocks that feeds into the next layers. Not only this, a ResNet architecture is so deep that it generally contains 23 million trainable parameters. The performance of ResNet is better because of spontaneous feature reduction.

Reasons behind Object detection performance:

Then RPN is key for Faster R-CNN as it basically proposes the important regions based on the blood cell objects. Since two convolutional networks is working alongside each other, the time taken is quite long.

Whereas SSD has default bounding boxes that work great for larger objects. Some blood cells inside the dataset fell into the smaller category and therefore the accuracy was low. But due to its existing bounding boxes, SSD gave faster performance.

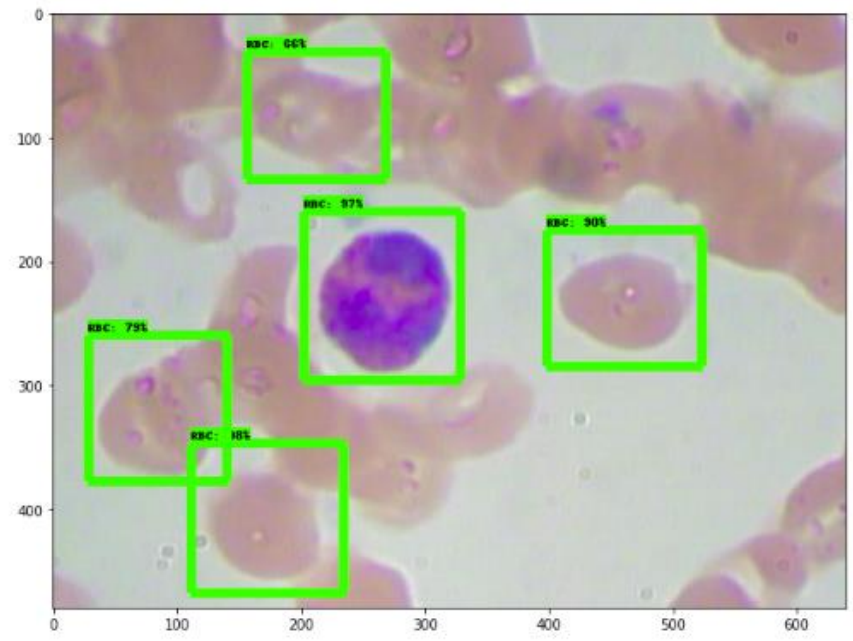


Fig: Poor SSD precision on overlapping cells

6. Conclusion

The observation from the experiments is that both the tasks of image classification and object detection have been simplified using transfer learning. Although 6 models have been tested, ResNet50 with a few layers in addition had a drastical change in accuracy. For object detection Faster R-CNN with an inception base performed slower than SSD with mobilenet. It is fascinating to see how these architectures are changing the world in terms of automation and simplifying human workloads.