**OBJECT DETECTION**

**GROUP 14**

**GROUP MEMBERS :**

Tirth Patel – AU1940137

Nipun Patel – AU1940033

Neel Shah – AU1940055

Kuldip Gohil – AU1940159

Anmol Pokharna – AU1843052

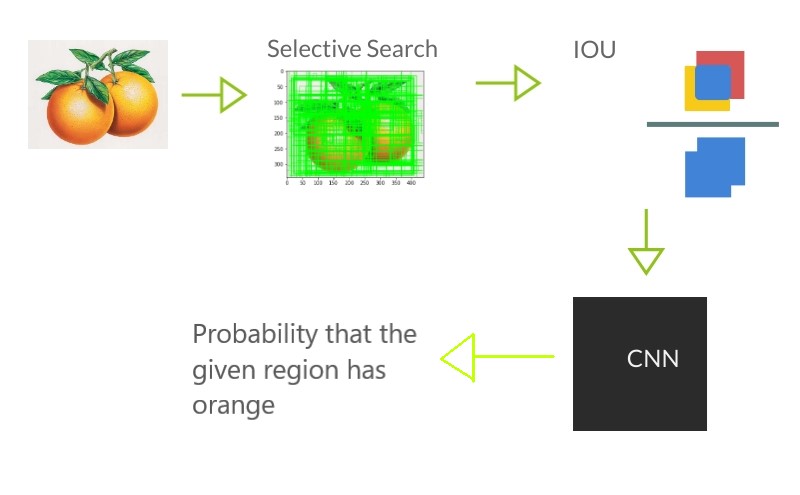
**INTRODUCTION:**

**Background:** Object detection is one of the challenges in the field of computer vision that helps to identify objects and differentiate between them. The basic idea behind object detection is that no two objects are identical and each object can be differentiated from the other object by unique properties that an object possess. Similarly, because of this unique property we can group all similar objects using object recognition. Object recognition involves training computer how to identify objects, how to differentiate one object from the other thus, making jobs of people easy and apart from that, object recognition is used for many different purposes like image retrieval, security, machine inspection and many more.

**Motivation and Overview:** For object recognition, the method used is Region Proposals with Convolution Neural Network(R-CNN) which is different from other neural networks in many respects. This neural network takes help of selective search algorithm to extract only limited number of regions to classify, thus making it easier to process. These regions are then buckled into Convolutional Neural Network(CNN) that gives feature map as an output which ultimately detects the object. CNN is faster as compared to other neural networks as it takes a smaller number of regions but one can still consider it slow as we still have to train quite a few numbers of regions of a given image and because of that, we can not use it in real time image processing.

**Approach:**

In this project, we are using R-CNN method to detect object from an image. R-CNN is a two-stage detection algorithm, the first stage of the algorithm identifies a subset of regions in an image that might have an object in it and the second stage of the algorithm detects object in each selected region through CNN. The flow of the approach is showed below



In our [dataset](https://www.kaggle.com/mbkinaci/fruit-images-for-object-detection) we have images of orange fruits with ground truth boxes in csv files. The very first step in the algorithm is to perform selective search on the image to get the region proposals.

Selective search firstly performs segmentation on the image using Felzenswalb and Huttenlocher algorithm and then starts with the random point in the image and groups the neighbours w.r.t similarities in colour, texture, size and shape (1). Selective search returns region proposals in sorted order. The number of region proposals generated by selective search is not fixed. In our approach we use first 2000 region proposals only. [[4]](#_References)

The 2000 region proposals generated still is s huge number of region proposals for a single image to train. We use Intersection Over Union (IOU) method to even cut down the number of region proposals. In IOU we calculate the intersection and union area of the ground truth box and the region proposal and output Intersection/Union. In our project we have putted the criteria of that we only consider 30 region proposals having IOU greater than 0.7 and 30 region proposals having IOU less than 0.3. We do this so that the CNN we train will not be biased as we have equal number of positive and negative samples.

Finally, we convert the rest of the regions from RGB to grayscale and split the regions in two parts train and test. We than train the CNN by passing the training regions in it.

The CNN will have convolution layer, max pooling layer and at last SoftMax layer for classification. A particular region will be sent to CNN for forward propagation and output will be saved. Cross entropy loss will be calculated with the output from forward propagation and actual label. Through the loss we will perform back propagation in the neural network and update the weights. We do it for all the regions in the training set.

After training CNN, we test it for the data which we separated earlier. And implement predict method which takes image as input and generates detection boxes.

**LA Concepts Used:**

**Linear Transformation**

**Forward Propagation:**

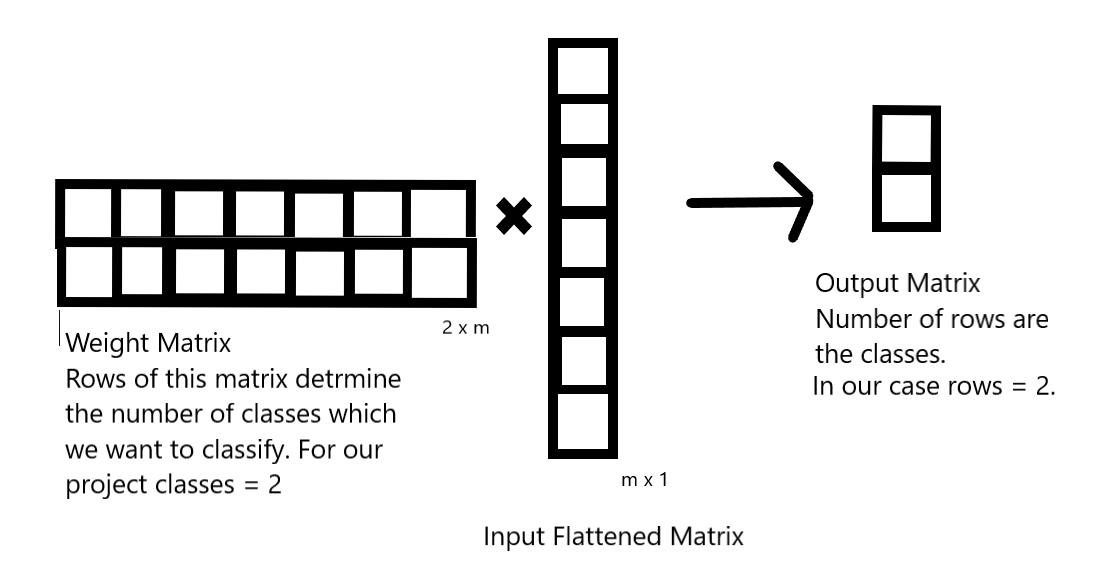
Almost every Neural Networks uses linear transformation to generate feature maps of the given input. The very last layer of the almost any CNN is used for classification which is mostly SoftMax layer. The dese layer + SoftMax layer takes the input of feature maps of form axbxc and flattens them to a vector of size (a\*b\*c)x1 shape. We generate weights matrix of the size mx(a\*b\*c) where m is the number of classes we want to classify by the formula.

Z = W\*X + B

Here

W = weights vector of the size M x (a\*b\*c), M = no. of classes for classification

X = Input vector of the size (a\*b\*c)x1



Values in the output matrix are represent the strength of that class. But the value is still not the probability. In order to get the probabilities of the classes we apply SoftMax activation function.

**Backward Propagation:**

Backward Propogation uses element wise matrix multiplication to generate the matrix of gradients which will be used to update the values of the parameters.

**Coding and Simulation:**

We have training data set in the folder train\_zip which has image file and csv file. Firstly, we began by visualizing the data set. Throughout the code we have used cv2.imread() to read the image and pd.read\_csv() to read the csv file and plt.imshow() to show the image and cv2.rectange() to add the rectangle to the image.

Then we tested the selective search on the image to see the output boxes.

**Generating Region Proposals:**

get\_iou()

Input: Box1, Box2

Output: IOU of the Two boxes

Implementation: we find the coordinates for intersection part. then we calculate the area of intersection part. then to find the area of union area we use this formula union\_area=(area\_of\_first\_region+area\_of\_second\_region-Inter\_section\_area). Then we return the division of intersection area and union area.

For generating the region proposals, we enumerate through all the files and perform selective search on them. For performing selective search, we use cv2 library functions.

ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()

ss.setBaseImage(image)

ss.switchToSelectiveSearchFast()

ssresults = ss.process()

ssresults will have the region proposals made from selective search. We will use the first 2000 entries of the ssresults to calculate the IOU between the ground truth box and the proposed region. For each image we limit 30 positive samples with IOU greater than 0.7 and 30 negative samples with IOU less than 0.3.

**Processing the region proposals:**

The region proposals we get will be of arbitrary shape and size but we cannot define a CNN model which takes any input. There for in our case we convert the proposed regions from arbitrary size to 64x64 size matrix. Moreover, we also convert input data of the color scale RGB to grayscale. At last we split out proposed regions into train data and test data with the help of the sklearn library train\_test\_split() method.

**Preparing CNN Layers:**

Convolutional neural networks have three main layers or operations namely Convolution, Max-pooling, SoftMax. So, we have defined classes for each of the three operations. In the respective classes we wrote the methods of forward and backward propagation.

**Class conv\_op**

Members: num\_filters, filter\_size, conv\_filter

* filter\_size (s) – Filter Size. Taken as input
* num\_filters (n) – Number of Filters. Taken as input
* conv\_filter – We generate n filters of size s with random values by using np.random.randn() function.

**Methods**:

* image\_region()

Input: image

Process: It generates the regions of the images which are going to be multiplied to the filter.

Output: yield regions of the image.

Implementation: We can generate the regions of the same size as filter by running two for loops.

* forward\_prop()

Input: image

Process: Performs Fast Fourier Transform or generates matrix of features after performing convolution on the image by conv\_filter.

Output: num\_filters(n) feature matrix

Implementation: We Call the image\_region() method to generate the regions and then multiply the generated region with the filters from conv\_filter and sum them to get the value.

* back\_prop()

Input: dl\_out – output of the gradient from the maxpool layer

learning\_rate – learning rate

Process: Updating the new parameters by learning the loss.

Output: dl/dconv or the derivative of loss w.r.t convolution operation

Implementation: Write Here

**Class Max\_Pool:**

Members: filter\_size

* filter\_size (s) – Filter Size. Taken as input
* image – down sampled image we generate in image\_region() method.

**Methods**:

* image\_region()

Input: image

Process: It generates the regions of the image of the size of filter.

Output: yield regions of the image.

Implementation: We can generate the regions of the same size as filter by running two for loops.

* forward\_prop()

Input: image

Process: Down sampling the given image by the factor of filter\_size.

Output: Down sampled num\_filters(n) feature matrix

Implementation: We Call the image\_region() method to generate the regions and then find the maximum value in that region by np.amax() and keep that value while we discard other values.

* back\_prop()

Input: dl\_dout – output of the gradient from the softmax layer

Process: Increase the size of the input from the softmax layer by factor num\_filters.

Output: dl/dmaxpool or the derivative of loss w.r.t maxpool operation

Implementation: Here as there are no extra parameters applied during front propagation, we only need to pass the derivates from the softmax layer by changing the dimensions by a factor x which were reduced earlier. [[2]](#_References)

**Class SoftMax:**

Members: weight, bias

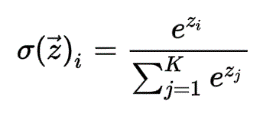
* weight – we randomly generate weight vector of size //Write Here
* bias – Bias vector we initialize to zero

**Methods**:

* forward\_prop()

Input: image

Process: We flatten the image matrix to a vector V. Then perform V\*weight + Bias and store it in x. Then performing

 to calculate the class probabilities.

Output: Probabilities of the image region to be orange or not.

Implementation: To flatten the image we use flatten() method of NumPy. To do the matrix multiplication of flattened image and weights we use np.dot() method of NumPy. Then after adding the bias in order to calculate the SoftMax output we use np.exp().

* back\_prop()

Input: dl\_dout – output of the gradient from the cross-entropy loss

learning\_rate – learning rate

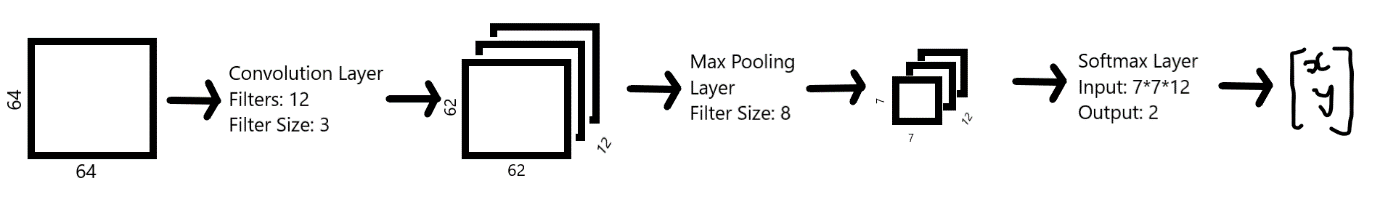
Process: Calculate the gradients of weights, bias w.r.t loss and ultimately calculate the gradient of SoftMax layer w.r.t loss.

Output: dl/SoftMax

Implementation: We do not need to do much in order to compute the gradient of We firstly calculate the gradient of softmax activation function and use it to calculate the gradient of weights matrix.

Modelling CNN:

We have chosen a small CNN architecture in our project. It can be shown by



For this we created objects from the classes which we defined earlier.

The input to convolution layer is 64x64 matrix and we convolve it with 12 filters of size 3x3. Since we have not applied any padding or stride the output dimension of the matrix will reduced to 64(input) – 3(filter size) + 1 = 62. Therefore, our convolution operation will output 12 feature maps of size 62x62.

The output of the convolution layer will go as an input to the max pool layer which will do down sampling and produce the output of the size 62//8 = 7. Therefore, the max pool layer will output 12 feature maps of the size 7x7.

The 12 7x7 matrix will go in the soft max layer. For our case since we provided the output of 2 units the soft max will generate weights in such manner that the output will be 2 values (Probability that region is orange, Probability that the region is background). [[1]](#_References)

**Forward Propagation in the CNN:**

cnn\_forward\_prop()

Input: image, label

Output: out\_p – probabilities from the SoftMax function

cross\_ent\_loss – cross entropy loss

accuracy\_eval – accuracy

Implementation: We firstly normalize the image matrix and pass the image to the forward\_prop() method of convolution object then pass the output of this method to the forward\_prop() method of maxpool object then pass the output of this method to the forward\_prop() method of SoftMax layer. The SoftMax layer forward\_prop() will give the probability or out\_p. We calculate the accuracy as 1 if the maximum probability by SoftMax is equal to the label else zero. We calculate the cross-entropy loss with the help of np.log().

**Training and testing the CNN:**

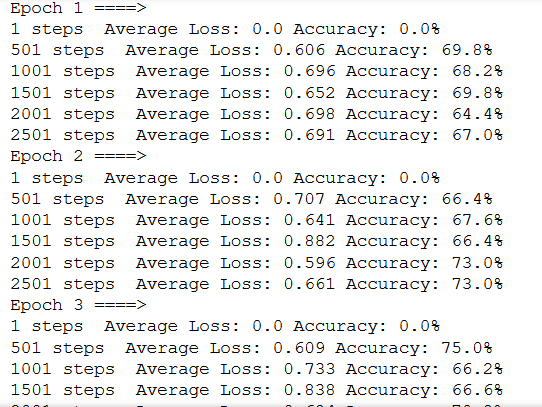
training\_cnn()

Input: image, label , learning rate

Output: loss, accuracy

Implementation: We firstly call the cnn\_forward\_prop() function to do the forward propagation in the CNN model. We will initialize the gradient and pass it to the back\_prop() method of SoftMax, maxpool and convolution layer to update the weights and bias.

For training the images we initialize the epochs to 5 and implemented the for loop which calls training\_cnn() and passes the images and labels one by one. We print the average loss and average accuracy per 500 steps in the model.



We test the model using the for loop and calling cnn\_forward\_prop(im, label) and just calculating the loss and accuracy and they are average 0.6 and 70% . [[5]](#_References)

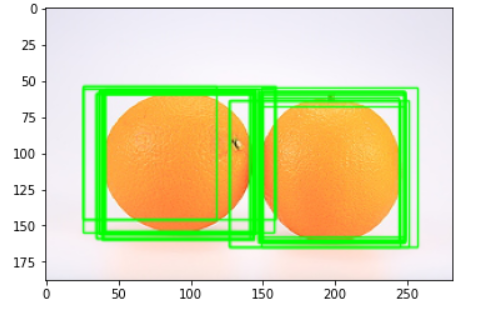
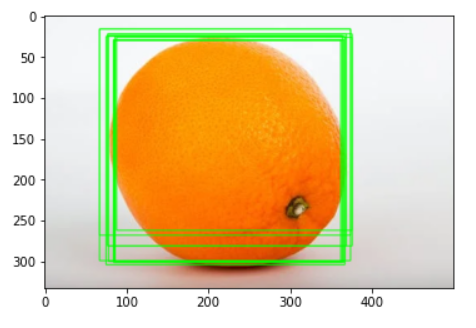
**Testing the RCNN**:

predict(image) – performs forward propagation in CNN and outputs the probability

We have two sample images in orange\_test folder which we are going to use to test the whole code at once. Firstly, we perform selective search on the input image and then for first 2000 proposals we send them to predict() function which outputs the probabilities. We output all the boxes which have probability of more than 0.6 for orange. [[3]](#_References)

**Inference:**

We tested our model on few images. Our model detects the orange in the image and draws a bounding box on it. Overall algorithm is very slow. It takes up to 2-3 minutes on our device to perform detection. Some of the results of detection are shown below.



Our implementation has some flaws. Our model doesn’t work well when there are multiple oranges in the image as shown in the above image. Also, sometimes many bounding boxes are generated foe a single object which should not be the case. We output the boxes which has high probabilities however if the co-ordinates of box differ by one-unit multiple boxes may appear for one region. It is because each box will have no idea that they are indicating the same region of the image.

**CONCLUSION:**

Hence, we implemented RCNN one of the very basic and old object detection algorithm. When it comes to performance there are many object detection algorithms which clearly surpasses RCNN. Still RCNN is one of the pioneer algorithms in the field of object detection. We successfully implemented RCNN algorithm and performed localization with the accuracy of more than 70%. We learned how the features are extracted from the image and are manipulated in order to detect some specific object. We learned the math behind the Convolution Neural Network and saw how the concepts of linear algebra are used in Machine Learning Domain.

Individual Contribution

Tirth Patel – Designed and prepared CNN and helped in other parts of project.

Neel Shah – Wrote report assisted in other parts of project.

Nipun Patel – Implemented IOU and selective search for the region proposals and helped in other parts of code.

Kuldeep Gohil – Founded the dataset and prepared presentation.

# References

1. *cs231n*. (n.d.). Retrieved from convolutional-networks: https://cs231n.github.io/convolutional-networks/
2. Gandhi, R. (2018, 7 10). *towardsdatascience*. Retrieved from r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e?gi=edf8177f8439#:~:text=The%20reason%20%E2%80%9CFast%20R%2DCNN,map%20is%20generated%20from%20it.
3. Girshick, R. (n.d.). *cv-foundation.org*. Retrieved from Girshick\_Rich\_Feature\_Hierarchies\_2014\_CVPR\_paper.pdf: https://www.cv-foundation.org/openaccess/content\_cvpr\_2014/papers/Girshick\_Rich\_Feature\_Hierarchies\_2014\_CVPR\_paper.pdf
4. Weng, L. (2017, 10 29). *Lil'log*. Retrieved from object-recognition-for-dummies-part1: https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html
5. Weng, L. (2017, 12 31). *Lil'log*. Retrieved from object-recognition-for-dummies-part-3: https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html