# Object Detection with Region-Based Convolution Neural Network based on VGG Backbone

**Samay Pashine**
Tufts University
samay.pashine@tufts.edu

**Lan Nguyen**
Tufts University
lan.nguyen@tufts.edu

**Yuyao Guo**
Tufts University
yuyao.guo@tufts.edu

**Zhangcheng Sun**
Tufts University
zhangcheng.sun@tufts.edu

## Abstract

Object detection is a computer vision technique that allows us to identify and locate objects in an image or video. We utilize the R-CNN, which uses bounding boxes across the object regions to evaluate convolutional networks independently on all the region proposals and in turn classify multiple image regions into the proposed class. We further extend R-CNN by leveraging self-attention mechanism to enable the model to focus on discriminative parts on region proposals. For the scope of this project, we selected the open-source airplane dataset from Kaggle. Our trained R-CNN model shows a respectable matching of approximately 98% accuracy.

## 1 Introduction

The object detection is a computer vision technique that allows us to identify and locate objects of certain classes (for example, cars, humans, animals, or buildings) in digital images such as photos or video frames. Since the most fundamental information needed by computer vision applications is "What objects are where?", the goal of object detection is to figure out this problem by developing computational models. Thus, it establishes the basis of many other computer vision tasks, for example, image captioning, object tracking, instance segmentation, etc. Specific object detection applications include number-plate recognition, face detection, text detection, pedestrian detection, pose detection, person counting and so on. [7] [8] Both rapid development of hardware and software in recent years greatly accelerated the momentum of object detection. Past decades have seen great progress of Imaging technology. Cameras are smaller, cheaper, and of higher quality than ever before. Meanwhile, computing power has significantly been strengthened and became much more efficient. Object detection can be performed using either traditional image processing techniques or modern deep learning method. Image processing techniques generally don't require data for training and are unsupervised in nature. Thus, those tasks do not require annotated images, where humans labeled data manually (for supervised training). These techniques are limited by a variety of factors, such as complex scenes (no monochromatic background), occlusion (partially hidden objects), lighting and shadows, and clutter effects. Deep Learning methods generally depend on training data. The performance is limited by the computation power of GPUs that is rapidly increasing year by year. Deep learning object detection is significantly more reliable for occlusion, complex scenes and challenging lighting. In the development of deep learning for object recognition, the proposal of Region-based Neural Network (R-CNN) is one of the most important milestones. In this project, we are going to study R-CNN architecture, and build a model to detect airplane in images using python with the help of some deep learning libraries like Tensorflow.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

### 1.1 Previous works and the evolution of R-CNN

To talk about R-CNN, we need to start with Neural Network (NN) and Convolutional Neural Network (CNN). Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network. Convolutional Neural Networks are one type of NN and are often utilized for classification and computer vision tasks. A CNN has three main types of layers, which are convolutional layer, pooling layer, fully-connected (FC) layer. The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity to identify greater portions of the image. Earlier layers focus on simple pattern, such as colors and edges. As the image feature progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object. Kunihiko Fukushima and Yann LeCun laid the foundation of research around convolutional neural networks in their work in 1980 and 1989 respectively. Notably, Yann LeCun successfully applied backpropagation to train neural networks to identify and recognize patterns within a series of handwritten zip codes. He would continue his research with his team throughout the 1990s, culminating with "LeNet-5", which applied the same principles of prior research to document recognition. Since then, a number of variant CNN architectures have emerged with the introduction of new datasets, such as MNIST and CIFAR-10, and competitions, like ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Some of these other architectures are AlexNet, VGGNet, GoogLeNet, ResNet, ZFNet, etc. However, LeNet-5 is regarded as the classic CNN architecture. Region-based Neural Network is first proposed by Ross Girshick et al in 2000. It is a type of machine learning model that is used for computer vision tasks, specifically for object detection, generated to bypass the problem of selecting a huge number of regions in the process of object detection.

## 2 Methodology

Region-Based Convolution Neural Network (R-CNN) is developed based on Convolutional Neural Network (CNN), which is a type of artificial neural network used in image recognition that is optimized to process pixel data. CNN layers enable the designed neural network to identify and recognize the object of interest in an image. They use selective search to extract just 2000 regions from the image, which is called region proposals[8]. These 2000 region proposals are first generated using the following selective search algorithm:

1. Generate initial sub-segmentation to formulate many candidate regions.
2. Use greedy algorithm to recursively combine similar regions into larger ones.
3. Use the generated regions to produce the final candidate region proposals. After that, they will be warped into a square and fed into a convolutional neural network that produces a 4096-dimensional feature vector as output. Here, the CNN acts as a feature extractor.

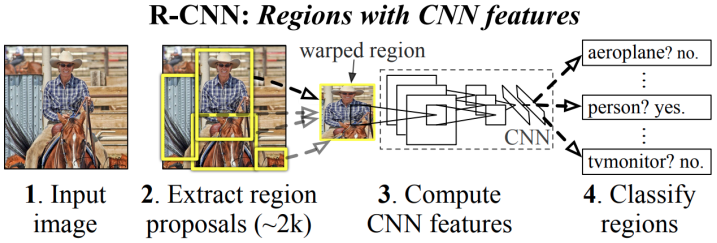



Figure 1: R-CNN by Rich feature hierarchies for accurate object detection and semantic segmentation, CVPR14

### 2.1 Region Proposal Generation

First, we use selective search to generate region proposals. Greedy algorithm:

1. From set of regions, choose two that are most similar.
2. Combine them into a single, larger region.
3. Repeat until only one region remains.


Figure 2: Example of selective search showing the necessity of different scales as the girl is contained by the TV

### 2.2 Bounding Box Unifications (IoU)

In order to improve localization performance, we can use bounding-box regression step to correct the predicted object (airplane) location. For this step, we have used the OpenCV segment detector tool to get the bounding boxes. Once we have all the bounding boxes, we find Intersection of Union (IoU) to get one prime box encapsulating the object.

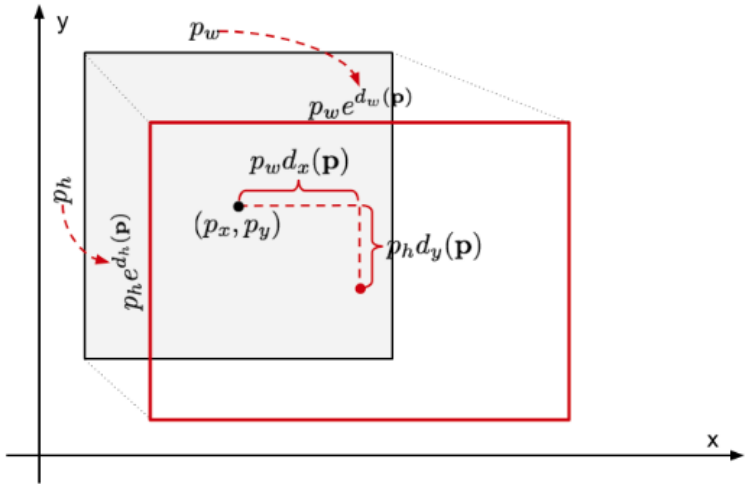



Figure 3: Image showing the how the bounding boxes are combined using IoU.

### 2.3 Feature extraction from Region Proposals

The first step of the R-CNN pipeline is the generation of 'region proposals' in an image that could belong to a particular object. At the end of this step in the pipeline, we can generate a 4096

dimensional feature vector from each of the 2000 region proposals for every image using a CNN. To compute the final output, the final classification layer of a pretrained CNN network is removed and a 4096 dimensional feature vector is obtained from the penultimate layer of the CNN for each of the 2000 region proposals.

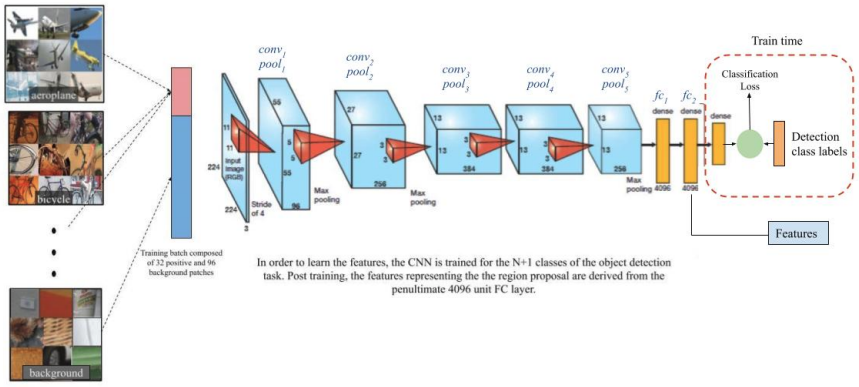



Figure 4: The pipeline of the network from image to the localization and classification of the object in the image.

### 2.4 VGG for object classification

VGG stands for Visual geometry group which is a standard models deep Convolutional Neural Network (CNN) architecture with multiple layers. The "deep" refers to the number of layers with VGG-16 or VGG-19 consisting of 16 and 19 convolutional layers. The VGG architecture is the basis of ground-breaking object recognition models. It was proposed by Karen Simonyan and Andrew Zisserman of Visual Geometry Group (VGG), Oxford University. VGG-16 is one of the most popular models which is able to classify 1000 different categories with 92.7% accuracy. For this part we are only working with airplane object (i.e. binary classification) but we can also use this with COCO dataset.

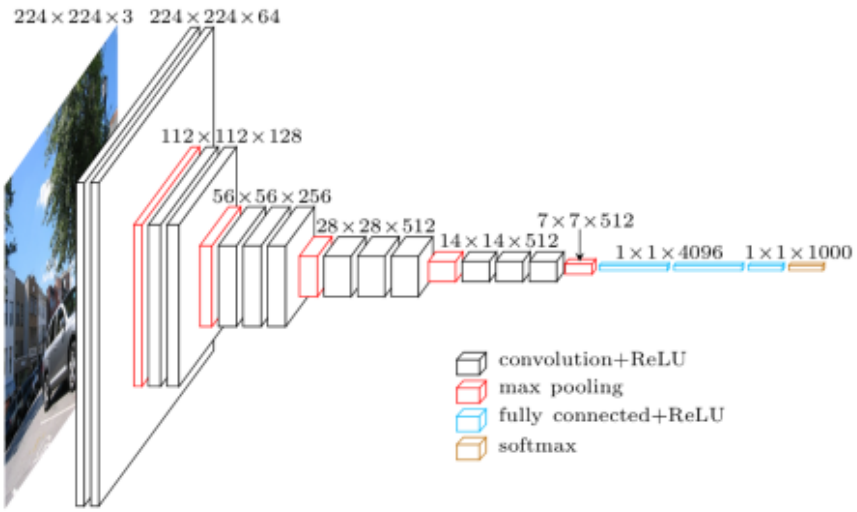



Figure 5: VGG-16 model architecture with the parametres.

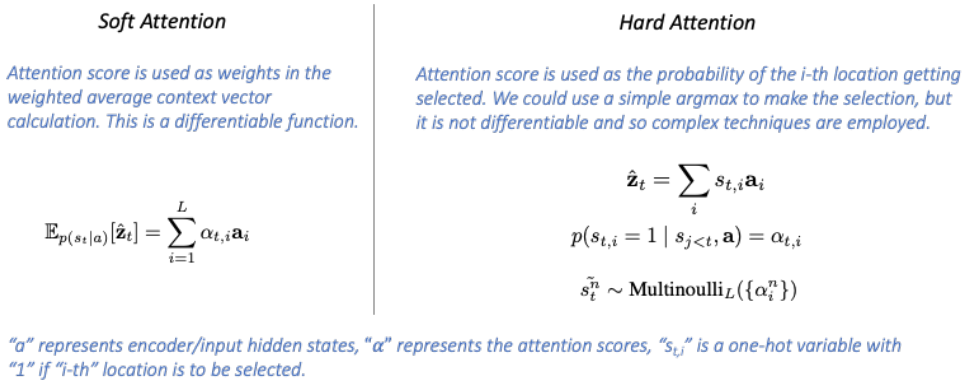



Figure 6: Difference between Soft attention and Hard attention.

# 3 Experiments

## 3.1 Our experiment with R-CNN and VGG16

### 3.1.1 Region proposal generation: Selective Search

For feature extraction: VGG-16 backbone contains 13 convolutional layers with 5 max pooling and 3 fully connected layers for prediction. Here we use an ImageNet pretrained VGG16, freeze the first 15 layers and fine-tune the remaining for our downstream task of airplane recognition.

### 3.1.2 Attention Mechanism Experiment for Classification

Attention mechanisms are inspired by the mechanisms of selective attention in the visual human cortex. These methods divert attention to the most important regions of an image and neglecting the other irrelevant parts [6]. After the implementation of R-CNN, the image classification and target detection has made significant progress. The image acts as an input into a neural network model, then the model completes the extraction of their deep features, the outputs go through the full connection layers as well as the features computation for a region proposal before achieving the purpose of classification [4]. By applying the bottom-up visual attention computing models to the specific tasks of computer vision, the interference of irrelevant information can be ignored, so the generalization performance of network can be improved.[5] As for the attention mechanism, it is usually divided into soft attention mechanism [1, 3, 2] and hard attention mechanism [9]. In this project, we use soft attention mechanism which is also a popular one in object detection. Please see figure 6 for more details.

In this project, we leverage self-attention layer to improve the model prediction. Specifically, self-attention layer computes a weighted sum of features vectors in feature map for each feature vector to focus on the discriminative locations in an image crop (self-attention module from keras tensorflow).

## 3.2 Accuracy

### 3.2.1 R-CNN model

**R-CNN model with VGG16 backbone analysis:** According to figure 7, we can see that the loss line significantly decreased from 0 to epoch 1. Then it steadily decreased to epoch 6. After that the loss line fluctuated until epoch 9. We assumed that this fluctuation occurred because the learning rate was too big and was not small enough to converge. In this case, we can decrease the learning rate for further decrease in loss line. In overall, the result was promising with 97.61%. Please see the training result table below in figure 8.

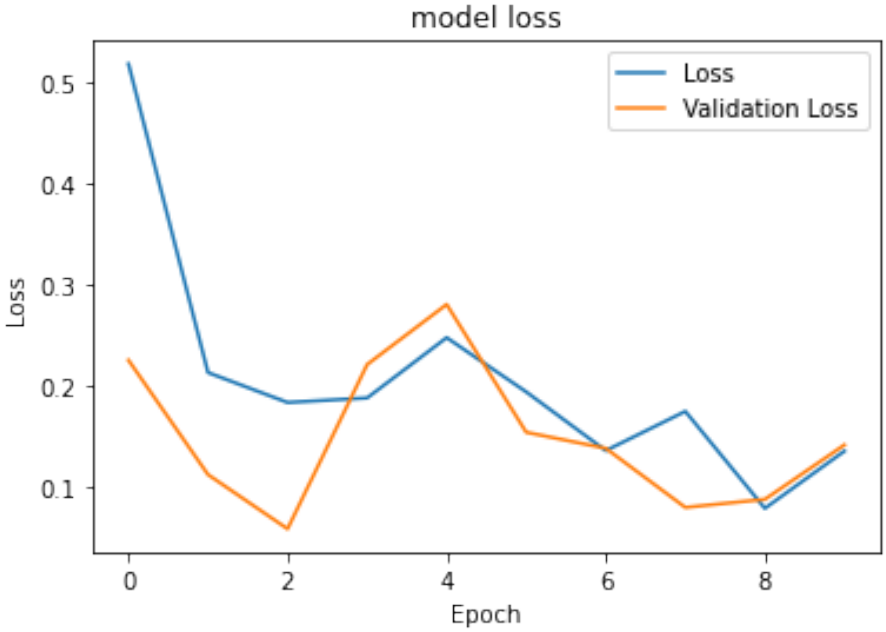



Figure 7: Model loss graph without attention layer.

```
Epoch 3: val_loss improved from 0.11203 to 0.05846, saving model to ieeercnn_vgg16_1.h5
10/10 [==============================] - 216s 22s/step - loss: 0.1832 - accuracy: 0.9283 - val_loss: 0.0585 - val_accuracy: 1.0000
Epoch 4/10
10/10 [==============================] - ETA: 0s - loss: 0.1877 - accuracy: 0.9469
Epoch 4: val_loss did not improve from 0.05846
10/10 [==============================] - 226s 22s/step - loss: 0.1877 - accuracy: 0.9469 - val_loss: 0.2207 - val_accuracy: 0.9375
Epoch 5/10
10/10 [==============================] - ETA: 0s - loss: 0.2473 - accuracy: 0.9156
Epoch 5: val_loss did not improve from 0.05846
10/10 [==============================] - 220s 22s/step - loss: 0.2473 - accuracy: 0.9156 - val_loss: 0.2800 - val_accuracy: 0.8438
Epoch 6/10
10/10 [==============================] - ETA: 0s - loss: 0.1937 - accuracy: 0.9219
Epoch 6: val_loss did not improve from 0.05846
10/10 [==============================] - 222s 22s/step - loss: 0.1937 - accuracy: 0.9219 - val_loss: 0.1538 - val_accuracy: 0.9688
Epoch 7/10
10/10 [==============================] - ETA: 0s - loss: 0.1357 - accuracy: 0.9500
Epoch 7: val_loss did not improve from 0.05846
10/10 [==============================] - 222s 23s/step - loss: 0.1357 - accuracy: 0.9500 - val_loss: 0.1380 - val_accuracy: 0.9688
Epoch 8/10
10/10 [==============================] - ETA: 0s - loss: 0.1745 - accuracy: 0.9531
Epoch 8: val_loss did not improve from 0.05846
10/10 [==============================] - 219s 22s/step - loss: 0.1745 - accuracy: 0.9531 - val_loss: 0.0796 - val_accuracy: 0.9688
Epoch 9/10
10/10 [==============================] - ETA: 0s - loss: 0.0787 - accuracy: 0.9761
Epoch 9: val_loss did not improve from 0.05846
10/10 [==============================] - 204s 21s/step - loss: 0.0787 - accuracy: 0.9761 - val_loss: 0.0876 - val_accuracy: 0.9844
Epoch 10/10
10/10 [==============================] - ETA: 0s - loss: 0.1353 - accuracy: 0.9625
Epoch 10: val_loss did not improve from 0.05846
10/10 [==============================] - 219s 22s/step - loss: 0.1353 - accuracy: 0.9625 - val_loss: 0.1413 - val_accuracy: 0.9688
```

Figure 8: Training data of the model with accuracy details

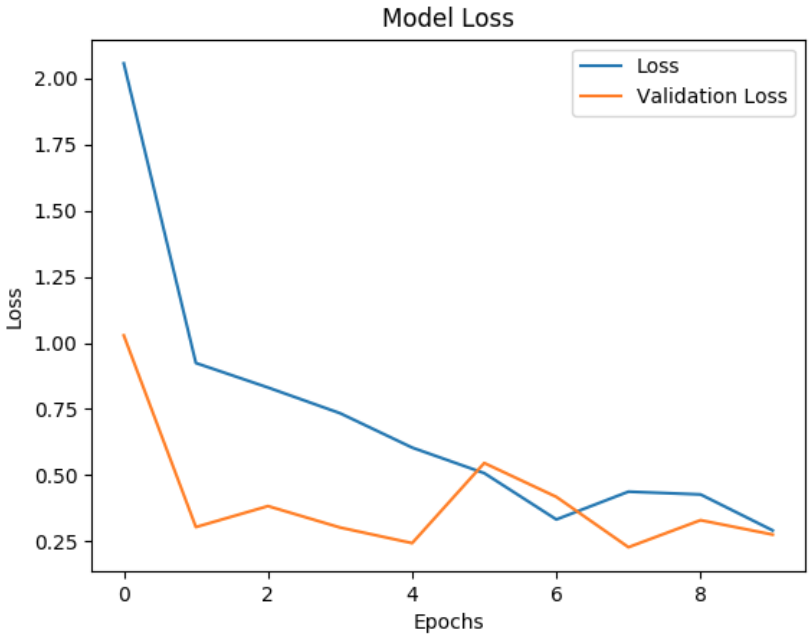
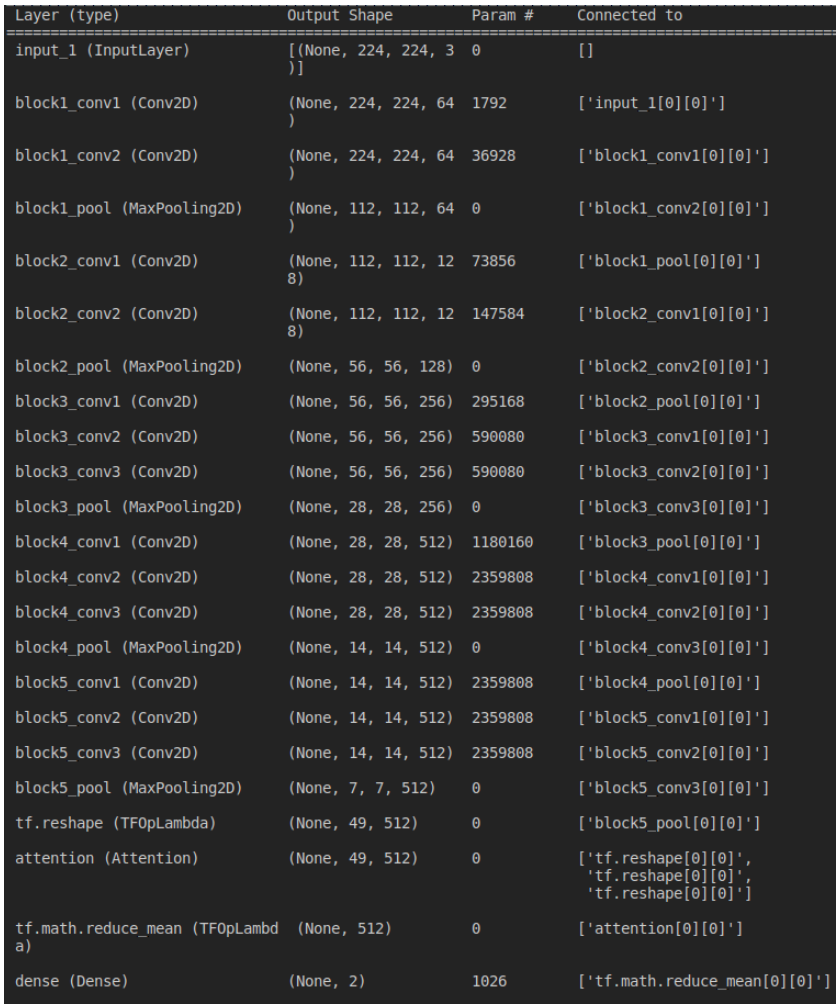



Figure 9: Model Loss graph with attention layer

```
Total params: 14,715,714
Trainable params: 7,080,450
Non-trainable params: 7,635,264
_________________________________________________________________
[INFO]. Starting the Training.
Epoch 1/10
10/10 [==============================] - ETA: 0s - loss: 2.0572 - accuracy: 0.5414
Epoch 1: val_loss improved from inf to 1.02860, saving model to ./model_weights.h5
10/10 [==============================] - 75s 7s/step - loss: 2.0572 - accuracy: 0.5414 - val_loss: 1.0286 - val_accuracy: 0.7872
Epoch 2/10
10/10 [==============================] - ETA: 0s - loss: 0.9243 - accuracy: 0.7906
Epoch 2: val_loss improved from 1.02860 to 0.30398, saving model to ./model_weights.h5
10/10 [==============================] - 75s 8s/step - loss: 0.9243 - accuracy: 0.7906 - val_loss: 0.3040 - val_accuracy: 0.8511
Epoch 3/10
10/10 [==============================] - ETA: 0s - loss: 0.8316 - accuracy: 0.8276
Epoch 3: val_loss did not improve from 0.30398
10/10 [==============================] - 68s 7s/step - loss: 0.8316 - accuracy: 0.8276 - val_loss: 0.3833 - val_accuracy: 0.8936
Epoch 4/10
10/10 [==============================] - ETA: 0s - loss: 0.7344 - accuracy: 0.8594
Epoch 4: val_loss improved from 0.30398 to 0.30229, saving model to ./model_weights.h5
10/10 [==============================] - 74s 7s/step - loss: 0.7344 - accuracy: 0.8594 - val_loss: 0.3023 - val_accuracy: 0.9149
Epoch 5/10
10/10 [==============================] - ETA: 0s - loss: 0.6044 - accuracy: 0.8621
Epoch 5: val_loss improved from 0.30229 to 0.24330, saving model to ./model_weights.h5
10/10 [==============================] - 68s 7s/step - loss: 0.6044 - accuracy: 0.8621 - val_loss: 0.2433 - val_accuracy: 0.8936
Epoch 6/10
10/10 [==============================] - ETA: 0s - loss: 0.5078 - accuracy: 0.8500
Epoch 6: val_loss did not improve from 0.24330
10/10 [==============================] - 74s 7s/step - loss: 0.5078 - accuracy: 0.8500 - val_loss: 0.5461 - val_accuracy: 0.8085
Epoch 7/10
10/10 [==============================] - ETA: 0s - loss: 0.3327 - accuracy: 0.8813
Epoch 7: val_loss did not improve from 0.24330
10/10 [==============================] - 74s 7s/step - loss: 0.3327 - accuracy: 0.8813 - val_loss: 0.4181 - val_accuracy: 0.8511
Epoch 8/10
10/10 [==============================] - ETA: 0s - loss: 0.4379 - accuracy: 0.8862
Epoch 8: val_loss improved from 0.24330 to 0.22781, saving model to ./model_weights.h5
10/10 [==============================] - 66s 7s/step - loss: 0.4379 - accuracy: 0.8862 - val_loss: 0.2278 - val_accuracy: 0.9362
Epoch 9/10
10/10 [==============================] - ETA: 0s - loss: 0.4273 - accuracy: 0.8621
Epoch 9: val_loss did not improve from 0.22781
10/10 [==============================] - 70s 8s/step - loss: 0.4273 - accuracy: 0.8621 - val_loss: 0.3300 - val_accuracy: 0.9149
Epoch 10/10
10/10 [==============================] - ETA: 0s - loss: 0.2914 - accuracy: 0.9241
Epoch 10: val_loss did not improve from 0.22781
10/10 [==============================] - 72s 7s/step - loss: 0.2914 - accuracy: 0.9241 - val_loss: 0.2756 - val_accuracy: 0.9574
[INFO]. Plotting the loss graph.
[INFO]. Saving the loss graph at current directory.
```

Figure 10: Training data of the model with attention layer including accuracy rate.



| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 | [] |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 | ['input_1[0][0]'] |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 | ['block1_conv1[0][0]'] |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 | ['block1_conv2[0][0]'] |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 | ['block1_pool[0][0]'] |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 | ['block2_conv1[0][0]'] |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 | ['block2_conv2[0][0]'] |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 | ['block2_pool[0][0]'] |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 | ['block3_conv1[0][0]'] |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 | ['block3_conv2[0][0]'] |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 | ['block3_conv3[0][0]'] |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 | ['block3_pool[0][0]'] |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 | ['block4_conv1[0][0]'] |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 | ['block4_conv2[0][0]'] |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 | ['block4_conv3[0][0]'] |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 | ['block4_pool[0][0]'] |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 | ['block5_conv1[0][0]'] |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 | ['block5_conv2[0][0]'] |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 | ['block5_conv3[0][0]'] |
| tf.reshape (TFOpLambda) | (None, 49, 512) | 0 | ['block5_pool[0][0]'] |
| attention (Attention) | (None, 49, 512) | 0 | ['tf.reshape[0][0]', 'tf.reshape[0][0]', 'tf.reshape[0][0]'] |
| tf.math.reduce_mean (TFOpLambda) | (None, 512) | 0 | ['attention[0][0]'] |
| dense (Dense) | (None, 2) | 1026 | ['tf.math.reduce_mean[0][0]'] |

Figure 11: Attention Mechanism Architecture in our experiment.

### 3.2.2 R-CNN Model with Attention layer

**R-CNN model with VGG16 backbone and attention layer analysis:** We used tf.keras.layers.Attention in Tensorflow. The goal was to help the model focusing on the discriminative part of each region proposal. We put this layer after the feature extraction step and before the classification step. The accuracy rate for this implementation reached 92.4% which was lower than the original model above. However, by looking the loss line in figure 9, although it was going downward but it was not as good as the figure 7. For this lower accuracy rate, we assumed that when classifying the airplane, it would be better if the mechanism could see all the components but the attention layer only focused on one small part and ignored the other important parts. As the result, this could be the reason that brought the accuracy rate down. Please see the training result table in figure 10 and its architecture in figure 11.

## 4 Conclusion

This report is an analysis of our project building R-CNN model based VGG backbone using python and deep learning libraries like tensorflow, scikit-learn with airplane dataset. We studied about the OpenCV functionalities to develop the major parts of the project. In addition, experiments on this model with self-attention layer to improve the performance, however, released a result in a decreasing performance by 5.21%. Although, self-attention layer showed little improvement in term of accuracy. Further advanced techniques such as fast R-CNN, faster R-CNN or Mask R-CNN will definitely

improve the results. We also explored the tensorflow, keras library and more advanced algorithm for object detection like YOLO v4 and more.

## References


[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[2] D. Huynh and E. Elhamifar. "A Shared Multi-Attention Framework for Multi-Label Zero-Shot Learning". In: *IEEE Conference on Computer Vision and Pattern Recognition* (2020).

[3] D. Huynh and E. Elhamifar. "Compositional Zero-Shot Learning via Fine-Grained Dense Feature Composition". In: *Conference on Neural Information Processing Systems* (2020).

[4] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[5] Saptarsi Goswami. *How to use a pre-trained model (VGG) for image classification.* `https://towardsdatascience.com/how-to-use-a-pre-trained-model-vgg-for-image-classification-8dd7c4a4a517`. [Online; accessed 01-May-2022].

[6] Meng-Hao Guo et al. "Attention mechanisms in computer vision: A survey". In: *Computational Visual Media* (2022), pp. 1–38.

[7] Elisha Odemakinde. *Mask R-CNN: A Beginner's Guide.* `https://viso.ai/deep-learning/mask-r-cnn/`. [Online; accessed 01-May-2022].

[8] Rohit Thakur. *step by step r-cnn implementation from scratch in python.* `https://towardsdatascience.com/step-by-step-r-cnn-implementation-from-scratch-in-python-e97101ccde55`. [Online; accessed 01-May-2022]. 2019.

[9] Kelvin Xu et al. "Show, attend and tell: Neural image caption generation with visual attention". In: *International conference on machine learning*. PMLR. 2015, pp. 2048–2057.