|  |  |
| --- | --- |
|  |  |

Image Classification and Object Detection

# Deep Learning

#### Student\_Name: Aaron Cherian

#### Student\_ID: S398713

#### Department: Applied Artificial Intelligence

#### Date: 3/2/2023

|  |  |  |
| --- | --- | --- |
|  |  | Image Classification and Object Detection |

# Assessment Questions

**Task 1 – Image Classification on the MNIST handwritten digits recognition dataset.**

**Task 1a – Train two different models Fully Connected and CNN Network and discuss the architecture and hyperparameters chosen**

**FCN (Fully Connected Network) Architecture**

The first Fully Connected Architecture was the Perceptron which was a simple algorithm that states that for a given input vector x of m values (x1, x2, x3…xm) which are known as features, output will be either a 1(‘yes) or 0(‘no’)

Where w is a vector of weights, w.x is the dot product ∑j=1 to m wjxj, and b is the bias.(Amita Kapoor, n.d.)

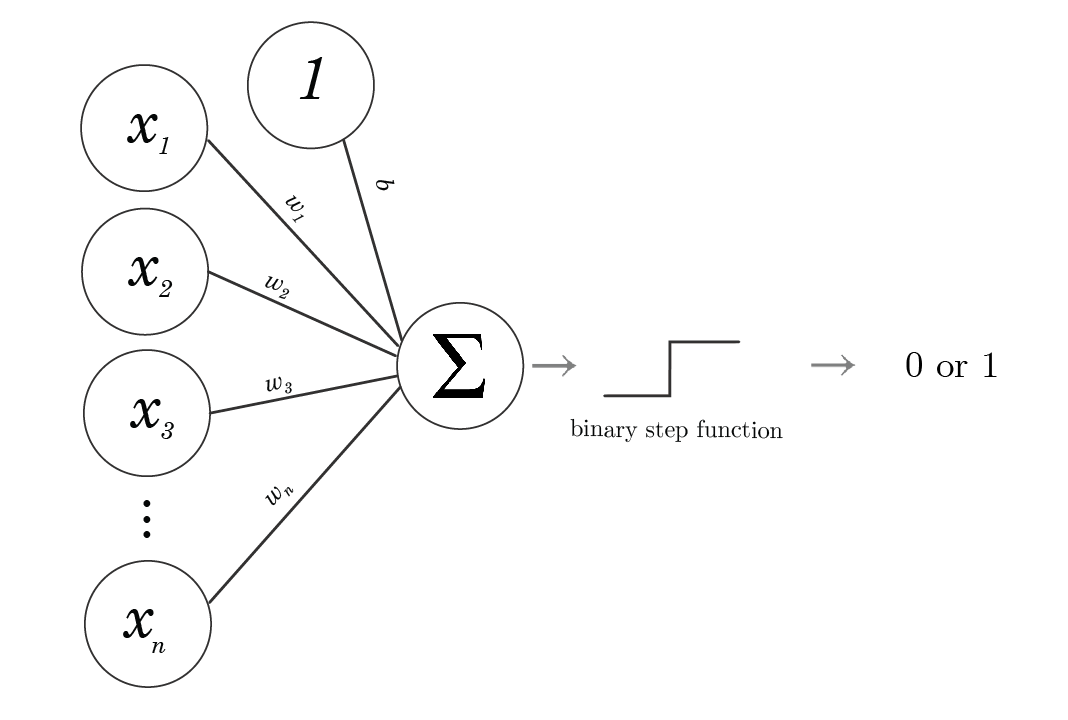


Figure 1: Perceptron(Adam Dhalla, n.d.)

After the Perceptron came the concept of Multiple Layer Perceptron which consisted of multiple layers unlike the Perceptron. In this mode, the input and the output layers are visible from the outside while the layers in the middle are hidden thus making it hidden layers. In this the first hidden layer receives an input and fires (0,1) according to the values of the associated linear function. Then the output of the first hidden layer is passed to the second layer where another linear function is applied. Then the results are passed to the final layer which can vary depending on the classification you perform.(Amita Kapoor, n.d.)

Multiple Layer Perceptron learns the training data through a process called back-propagation. At the beginning the weights associated with the neurons have some random assignment. As soon as the input is fed the neurons in the input layer are activated and then the values are propagated forward to the next layer. As we know the true observed value of the training set, error made during prediction can be calculated. The idea behind backpropagation is that the error is propagated back in the network using optimizer algorithms to adjust the weights of the neural network with the goal to reduce the error .

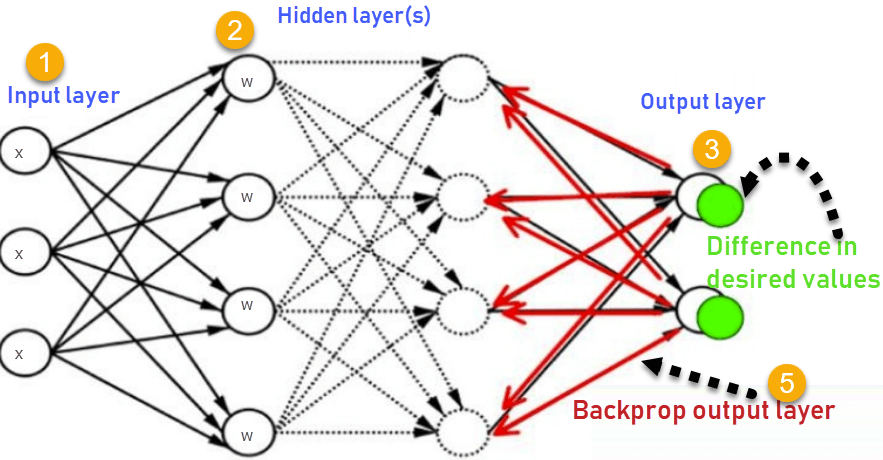


Figure 2 :Backpropagation(Https://Www.Niser.Ac.in/~smishra/Teach/Cs460/2020/Lectures/Lec19\_1/, n.d.)

**CNN (Convolution Neural Network) Architecture**

In fully connected network we were feeding the network with an input in our case the image of the MNIST dataset which had 28\*28 pixels which is a total of 784 input neurons. But this network doesn’t leverage the spatial structure and relationship between each image. In fully connected layer this spatial information is removed which causes important information to be lost making it less reliable while performing complex image classification tasks. CNN networks leverage this spatial information as well making it more reliable. These networks use the ad hoc architecture which has been inspired by physiological experiments performed on the visual cortex.(MK Gurucharan, n.d.)

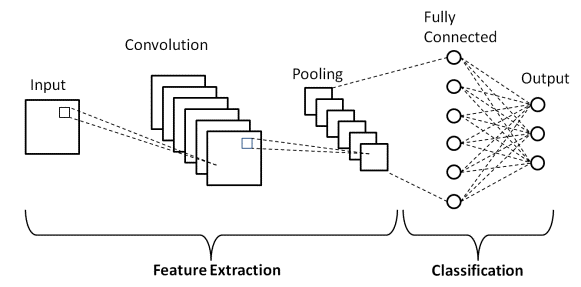


Figure 3: CNN Architecture(MK Gurucharan, n.d.)

A typical CNN Architecture includes the following components.

1. **Convolution Layers**: These layers apply a convolution operation on the input data, typically using a set of learnable filters, to extract meaningful features from the input images. These filters have kernels and the strides attributes that are used by these filters to extract meaningful information.(Michele Cavaioni, n.d.)

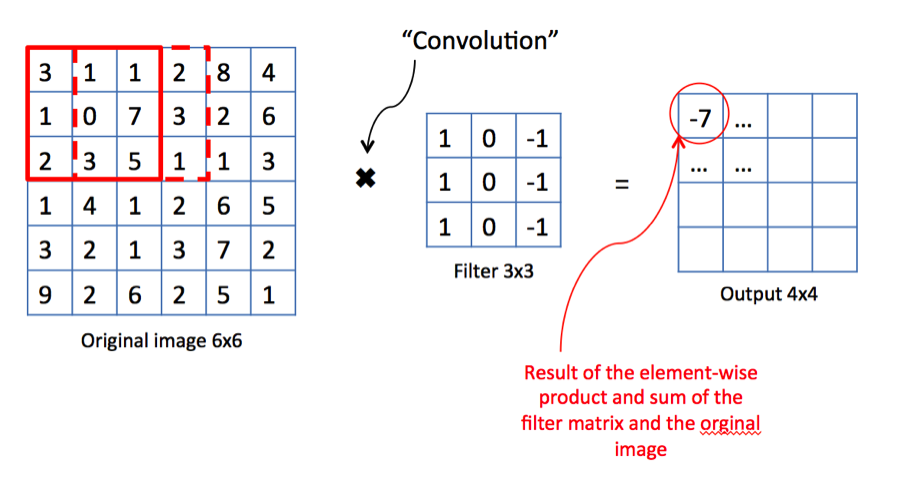


Figure 4: Filter operation on an input image(Michele Cavaioni, n.d.)

1. **Pooling Layers**: These layers down-sample the output of the convolution layers by reducing the spatial dimensions of the data and retaining only the most important information.

In this image we see what the pooling layer does is that it takes the maximum value from each and every filter and gives a down-sampled output

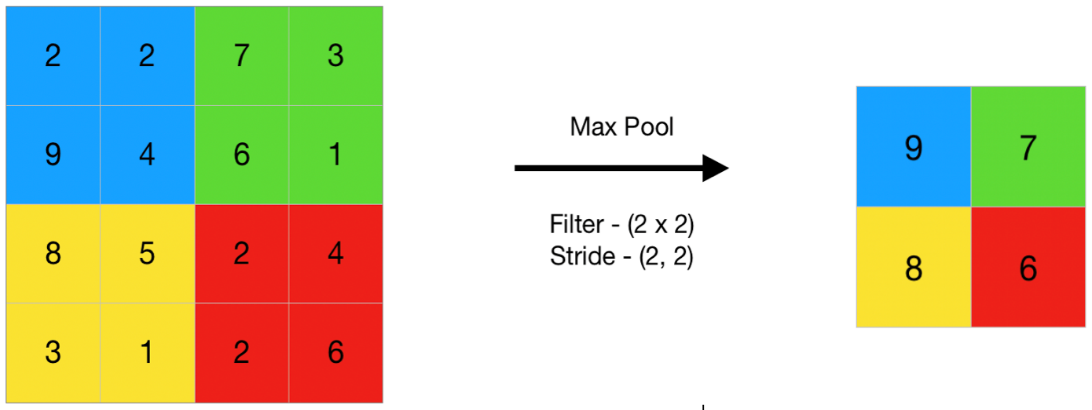


Figure 5: Pooling operation(Https://Www.Geeksforgeeks.Org/Cnn-Introduction-to-Pooling-Layer/, n.d.)

1. **Activation Functions**: These functions include non-linearities into the network, allowing it to learn complex relationships between the input and output.
2. **Fully Connected Layers**: These layers connect every neuron in one layer to every neuron in the next layer, allowing the network to make a final prediction based on the features learned by the convolution and pooling layers.
3. **Softmax Output Layer:** The layer is used to produce probabilities for each class, allowing the network to make a final classification decision.

This is the basic architecture of the Fully Connected Network and Convolution Neural Network. Let us discuss the hyperparameters chosen for these two models.

**Hyperparameters for Fully Connected Network and CNN Architecture**

Ideally we would want the computer to choose the best weights and bias so that the errors produced at the output are minimized. For example, if we feed an image to the neuron we would like the neuron to adjust its weights and bias in such a way that fewer images are recognized as wrong. Perceptron fails to portray a little-by-little learning behaviour as it gives an output of either 0 or 1 which is a big jump.

For this phenomena, activation functions are used to help the neuron to learn little-by-little.

1. **Activation Functions**
2. **Sigmoid Activation Function**

The sigmoid function is a function that maps any input to a value between 0 and 1. It has an S- shaped curve and it helps to tackle the issue perceptron had where the changes are more gradual. (Amita Kapoor, n.d.)

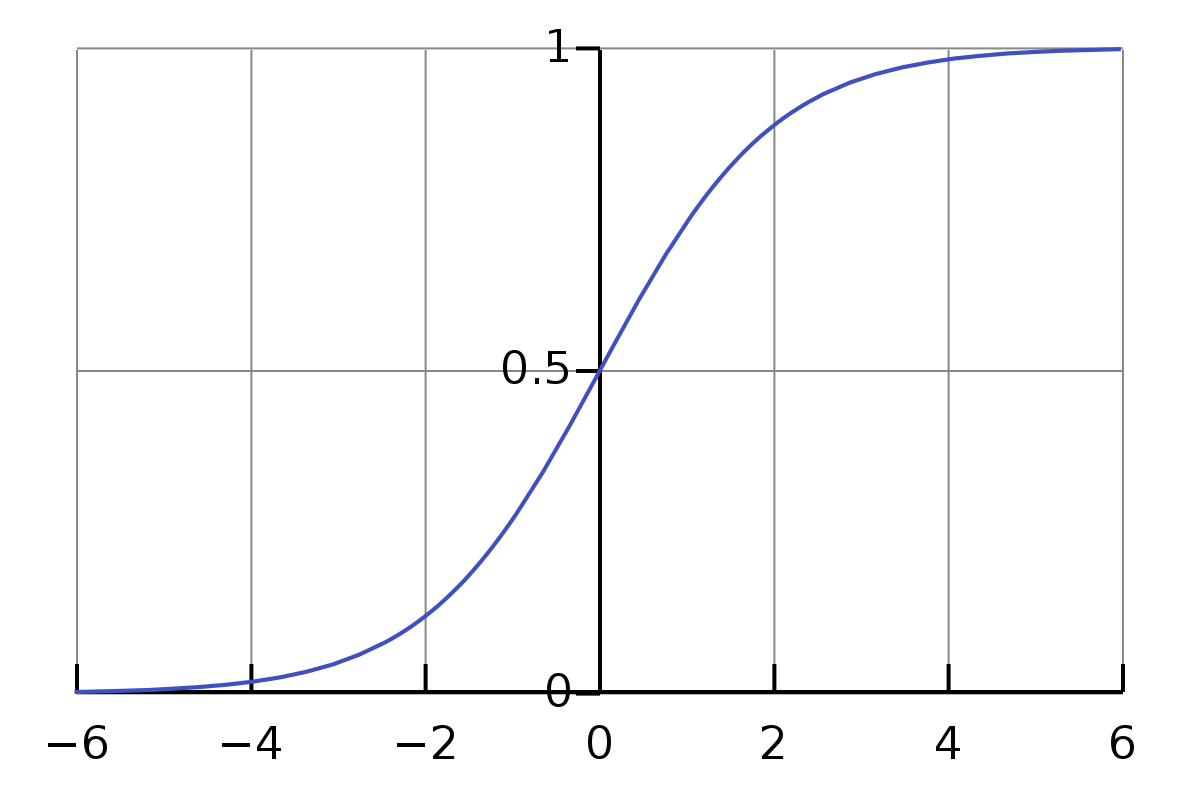


Figure 6: Sigmoid Activation Function(Wikipedia, n.d.)

One of the main drawbacks of the sigmoid function is that it can cause the vanishing gradient problem, where the gradient becomes very small and the learning slows down or stops completely. This happens because the sigmoid function is not centred around 0, so the gradients near the extremes (0 and 1) become very small and slow down the optimization process. This can lead to less efficient and less stable convergence of optimization algorithms. (Amita Kapoor, n.d.)

1. **ReLU Activation Function**

ReLU is a very simple activation function that has been developed recently and helps to address the issues of vanishing gradient observed in Sigmoid Activation Function. It is defined as

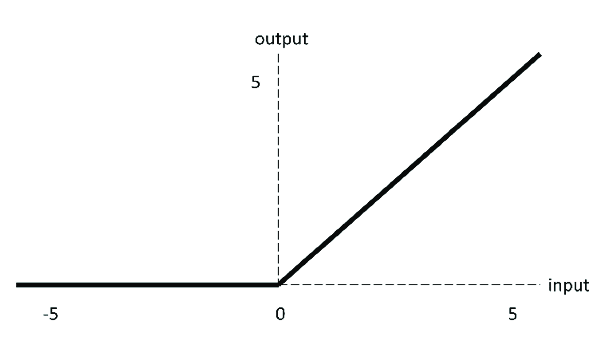


Figure 7: ReLU Activation Function(Sultan et al., 2019)

It solves the problem of vanishing gradient by allowing the function to have non-zero gradients for positive values, while having gradients of zero for negative values. The ReLU function outputs the input if it is positive and 0 otherwise. As a result, the gradient is either 0 or 1 and can avoid the vanishing gradient problem. This also makes the optimization process more efficient and stable, as gradients are larger and more consistent as compared to sigmoid function.(Amita Kapoor, n.d.)

Hence, ReLU Activation function was used in our model and softmax which is a generalization of the sigmoid activation function is used for classification as softmax squashes a K-dimensional vector of arbitrary values into a K-dimensional vector of real values in the range (0,1), so they all add up to 1. In the MNIST dataset it aggregates ten answers provided by the previous layer with ten neurons as there are ten labels in our case.(Amita Kapoor, n.d.)

1. **Optimizer**

An optimizer is a specific algorithm which is used to update the weights of the neural network during the training process. Various optimizers such as SGD(Stochastic Gradient Descent), RMSProp, Adam were used. SGD has the acceleration component that it uses to reach convergence. It was observed in RMSProp and Adam an additional component of momentum is included in addition with the acceleration component. And it was seen that the addition of this additional component helped in achieving better results and also helped in reducing oscillations.(Amita Kapoor, n.d.)

In our case, I used the Adam dataset which gave the best result among the other optimizers.

1. **Objective Functions**

The objective function is used by the optimizer to calculate and navigate the space of weights. Objective Function is also called as the Loss Function and Loss minimization is the process of optimization.

In our model we have used categorical\_crossentropy as our loss function. This is because it defines the multiclass logarithmic loss. In this function it compares the distribution of predictions with that of the true distribution with the probability set to 1 for the true class and 0 for other classes. (Amita Kapoor, n.d.)

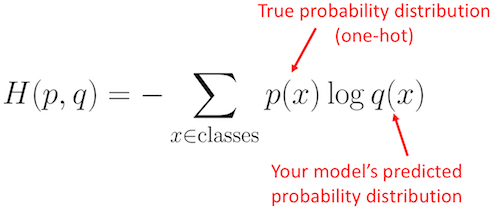


Figure 8:Categorical\_Crossentropy(Https://Stackoverflow.Com/Questions/41990250/What-Is-Cross-Entropy, n.d.)

In our dataset we had converted the Y\_train and Y\_test to a one-hot encoded vector. So, what categorical\_crossentropy does is that it compares the true class vector to the Y\_train vector and closer it is to the output of the Y\_train output the less is the loss.

There are a lot of loss functions such as binary\_crossentropy, mse but categorical\_crossentropy was the best option in our case.

1. **Increasing the Hidden Layers**

This method helps in improving the performance of the neural network. Because these additional neurons help in analysing the complex patterns present in the training dataset. These additional layers add some additional parameters which helps the model in memorizing complex patterns. These hidden layers are called hidden as they are neither connected with the input layer or the output layer.

1. **Dropout**

The idea behind Dropout is to randomly drop some of the values propagated in the internal network of hidden layers during training. It is known as a form of regularization and helps in improving the accuracy of the model. The theory behind that is randomly dropping values in a network helps the network in identifying redundant patters which help in generalization. This can be seen that networks with dropout introduced in the hidden layers helps in better generalization on unseen examples which are in the testing dataset. The neurons become less dependant on the neighbouring neurons which causes it to store information in a more redundant way. This also helps in avoiding overfitting of data.(Amita Kapoor, n.d.)

1. **Number of epochs**

Epochs is defined as the number of times the model is exposed to the training set. Every time the model is exposed to the training set after setting the number of epochs, at each iteration it is trying to adjust its weights so as to minimize the cost function and achieving the best optimization. Increasing the epochs doesn’t mean better performance of the model. It is important that the learning that is adopted is smart and not only focussed on the time spent in computations.(Amita Kapoor, n.d.)

**Task 1b – Discuss differences between FC and CNN Classifier and demonstrate that you have monitored the training of neural network by monitoring the loss function, the accuracy and confusion matrix. Demonstrate the performance of these models with the help of graphs.**

Difference between FC and CNN Classifier:

|  |  |  |
| --- | --- | --- |
| **Parameters** | **FC Classifier** | **CNN Classifier** |
| Input data | Fully connected networks accept a flat vector of inputs. | Convolution Neural Networks expect a grid-structured data such as an image. |
| Architecture | Fully connected networks have a flat, dense architecture with no spatial relationships between the neurons | CNN’s have a hierarchical, multi-stage structure that considers the spatial structure of the input data. |
| Parameters | Fully Connected networks have a lot more parameters than CNN’s making them more prone to overfitting on small datasets. | CNN’s have lot fewer parameters due to their use of shared weights and spatial pooling. |
| Invariance | Fully Connected networks are not invariant to translations or rotations of the input data. | CNN’s are designed to have some translational and rotational invariance through the use of convolution and pooling operations. |
| Computational Efficiency | Fully Connected networks are computationally expensive, particularly when working with high dimensional inputs. | CNN’s are designed to be computationally efficient even when working with high-dimensional inputs, due to the use of convolution and pooling operations. |

**Performance of the Fully Connected Neural Network**

**Model Loss Model Accuracy**

Chart, line chart

Description automatically generated A picture containing shape

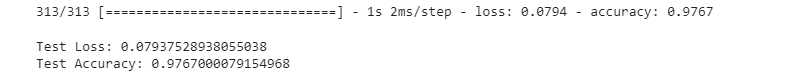
Description automatically generated

**Training and Validation Data Set Accuracy and Loss**

Table

Description automatically generated

**Test Loss and Accuracy**



**Performance of the Convolution Neural Network**

**Model Loss Model Accuracy**

A picture containing chart

Description automatically generated Shape

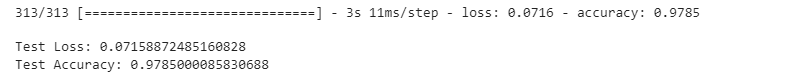
Description automatically generated with medium confidence

**Training and Validation Data Set Accuracy and Loss**

A screenshot of a computer

Description automatically generated with medium confidence

**Test Loss and Accuracy**



In loss vs epoch graph we can se that that the loss on validation is less than the loss of the training dataset. As soon as validation loss shows a slight increase after an initial drop it suggests that the model is overfit.

In fully connected layer it achieved an accuracy of 97.67% over 10 epochs whereas Convolution Neural Network was able to achieve 97.85% on the testing dataset.

**Loss Function Monitoring**

Monitoring the loss over epochs is important because it provides insight into how well the model is learning from the training data. The loss is a measure of the error between the predicted output and the actual target output and by tracking its value over each epoch one can observe if the model is overfitting, underfitting or has reached optimal performance.

A decrease in loss over epochs indicates that the model is improving, while a plateau or increase in loss suggests that the model is not learning effectively from the training dataset and may require further fine-tuning.

**Loss Function Monitoring FCN Loss Function Monitoring CNN**

Chart, line chart

Description automatically generated with medium confidence Shape

Description automatically generated

**Accuracy Monitoring**

Monitoring accuracy over epochs helps evaluate the performance of the training and validation data. This can provide insight when the model is underfitting which can be used to adjust the model accordingly.

Minoring accurcy over epochs helps to determine the best point at which to stop the model overfitting or underfitting.

In CNN structure, I had observed that when I had eochs of 20, the loss vs epoch graph showed the phenomena of overfitting. The validation loss started to increase after intitial decrease. In the graph it was observed that at the 12th epoch it started to display the overfitting phenomena. So, it helped me in giving me an idea as to how many epochs is necessary for the model to improve its accuracy.

**Confusion Matrix**

**Confusion matrix of FCN**

A screenshot of a computer

Description automatically generated with low confidence

**Confusion matrix of CNN**

A picture containing text, black, keyboard

Description automatically generated

A confusion matrix is a table used to evaluate the performance of a classification model. The matrix displays the number of instances that were correctly and incorrectly predicted. The rows represent the actual true values and the columns show the predicted values. For example, in the CNN model it was able to figure 0 , 971 times correctly and it predicted the number 2 - 4 times. This can be used to calculate recall, precision and other metrics. It can also be used to compare the performance of the model with different models.

**Task 1c – Apply pretrained network to the network and select one layer to visualize the activation of the learned feature map. Design a state-of-the-art classifier based on the MNIST dataset that will be able to beat LeNet 1998 model (98.7% accuracy on the test set). Demonstrate that the model doesn’t overfit the data.**

I chose the VGG-19 architecture to implement transfer learning on the MNIST dataset. I chose this model as I found the AlexNet and ResNet Models architecture to be complex and difficult to implement. Whereas in the VGG-19 architecture the convolution layers and Max Pooling layers were able to follow a fixed pattern which makes it easy to memorize and understand. In VGG- architecture every convolution layer will have the filter value – (3\*3), stride = 1 and padding = ‘same’. And every MaxPooling layer will have the filter size of (2\*2) with stride = 2. VGG-19 has more parameters in its architecture helping it to capture more complex patterns in the data potentially leading to better performance of given task. VGG-19 has been trained on a large image classification dataset, so it has already learned useful features that can be easily transferred to a new task with smaller dataset, speeding up the learning process.

The VGG-19 architecture consists of

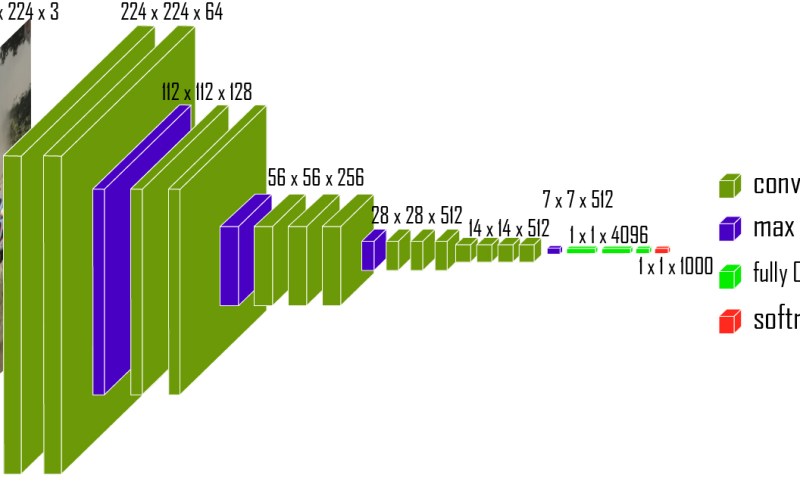


Figure 9: VGG-19 Architecture(geekycodesco, n.d.)

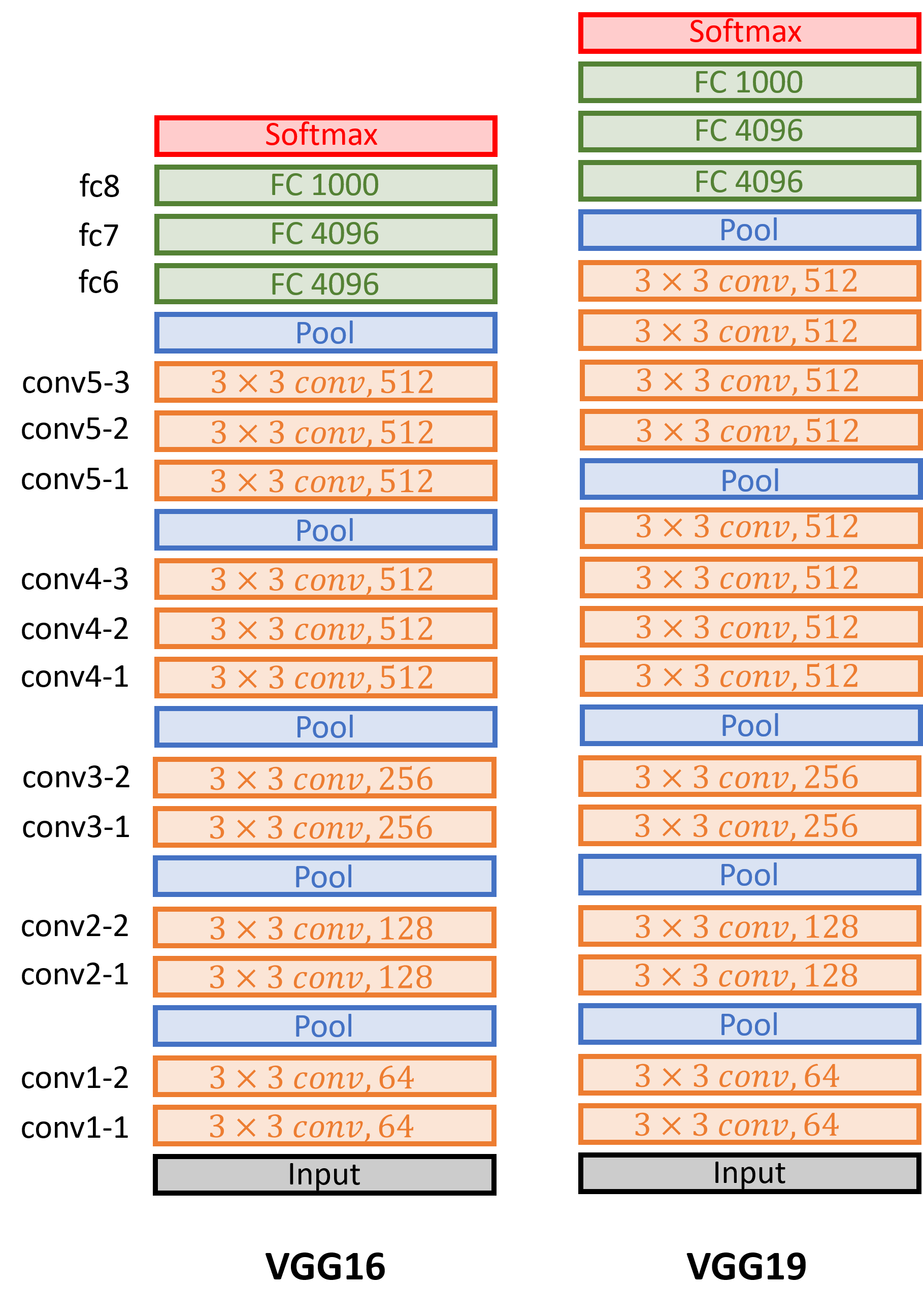


Figure 10: Comparison between VGG-16 and VGG-19 layers(Https://Datahacker.Rs/Deep-Learning-Vgg-16-vs-Vgg-19/, n.d.)

It is important to note that the image size of the VGG-19 model is 224\*224 pixels with 3 channels showing that it’s a RGB channel. In the Max Pooling we see that the next convolution layer receives a value lesser than the input image. Basically, it’s down sampling the image for the next convolution layer. It is calculated with a help of a basic formula – ((n+2p-f)/s) + 1 where n is input, p is the padding, f is the filter and s is the stride.

So, for the next convolution layer to get 112 we see that n = 224, p=0 as padding = ‘same’ , f= 2 and s= 2.

Therefore, ((224+0-2)/2)+1 = 112

**Activation of the learned feature map**

Activation of the learned feature map in the VGG-19 architecture represents the final output of a layer in the network after the non-linear activation function has been applied. And is used as input for the next layer in the network.

****

Figure 11:Learned Feature Map

**Design a state-of-the-art classifier based on the MNIST dataset that will be able to beat LeNet 1998 model (98.7% accuracy on the test set). Demonstrate that the model doesn’t overfit the data.**

I designed the classifier based on the VGG-19 model which uses three dense layers and a last classification layer activated by the softmax activation function. The training of VGG-19 model was paused and the weights were taken from ‘**imagenet’.**

I was able to achieve a 99.54% accuracy in training dataset and 97.10% on validation set and 96.70% on testing set.

**Model Loss Model Accuracy**

**Chart

Description automatically generated A picture containing shape

Description automatically generated**

**Confusion Matrix**

**A screenshot of a computer

Description automatically generated with low confidence**

**Other Advanced Architectures**

These architectures like VGG, ResNet, AlexNet are just relying on depth scaling but new architectures such as Efficient Net has been developed which focus not only on depth scaling but also on width and resolution. The idea behind Efficient Net is to up scale the resolution of the image so that the algorithm will learn complex features and increasing the depth of the layer by adding more number of layers. In addition, it will also increase the number of feature maps as we have up scaled the image to improve its resolution, this step may become useless if it doesn’t capture the image completely. But steps must be taken to ensure that the right number of layers for depth and feature maps are added as it might degrade the algorithm’s training speed.

So, in this model it is necessary to ensure that the perfect balance of the dimensions of the network is added to tackle the issue of vanishing gradient which may arise for bigger models.(Tan & Le, 2019)

**Task 2 – UAV Object Detection using Transfer Learning**

**Task 2a – Start by selecting 12 samples for both training and testing data and plot in 3x4 subplot format indicating:**

1. **Plot the labels and verify the label format: [x y width height].**

It is a good machine learning practice to first visualize the data given to us before doing any operations on it. This gives the programmer a clear idea as to what kind of data one is dealing with and whether he will have to do some data pre-processing or correction before implementing various algorithms.

After plotting the images in a 3\*4 subplot format we could deduce that the bounding box coordinates were correct and ready to implement.

**Graphical user interface, website

Description automatically generated**

Figure 12: 3\*4 subplot of the labels and images

The format given to us was in the form of (xmin, ymin, xmax, ymax)

**2. Pre-process the data set and split the data set in training/validation/testing set, explain your data split strategy.**

To implement You Only Look Once (YOLO) algorithm he label annotations has to be in the form of (class, x\_center, y\_center, width, height) where class is the class label of the object, x\_center and y\_center are the coordinates of the center of the bounding box relative to the entire image. The width and height are the dimensions of the bounding box. Converting the annotations to the correct format ensures that the data is compatible with the algorithm and can be used effectively for training and detection.

After converting the data to the YOLO format, the next step was to split the data in training, validation and testing set. In a machine learning model or an object detection model it is necessary to have separate tests for training and validation because it allows the mode to be evaluated on data that it has not seen during training. This helps in determining the true accuracy and generalization capability of the model, and helps in preventing overfitting.

For this problem I didn’t create a test folder as it is not necessary to create a test set as I had a validation folder which gave me the predicted image and label output. Although having a separate test set provides more robust evaluation of the model and can provide more insight into its generalization capabilities.

We had 314 images and labels. I put around 20% of 314 which is around 63 images and labels into a validation folder. As the normal methodology that machine engineers follow is to have a 70%-20%-10% split on your training, validation and testing set.

**Task 2a – Your aim is to train an object detector which achieves a good performance on the dataset and detects the position of UAVs in the picture. You need to apply and compare two different object detection networks, and one of it should be YOLO networks (no requirement to the specific version). You don’t have constrains about computational time (for inference) so you can opt also for two-stage detectors.**

**YOLO Architecture**

YOLO Architecture is same as that of GoogleNet. It has 24 convolution layers, 4 Max Pooling Layers and two fully connected layers.(https://Www.Datacamp.Com/Blog/Yolo-Object-Detection-Explained, n.d.)

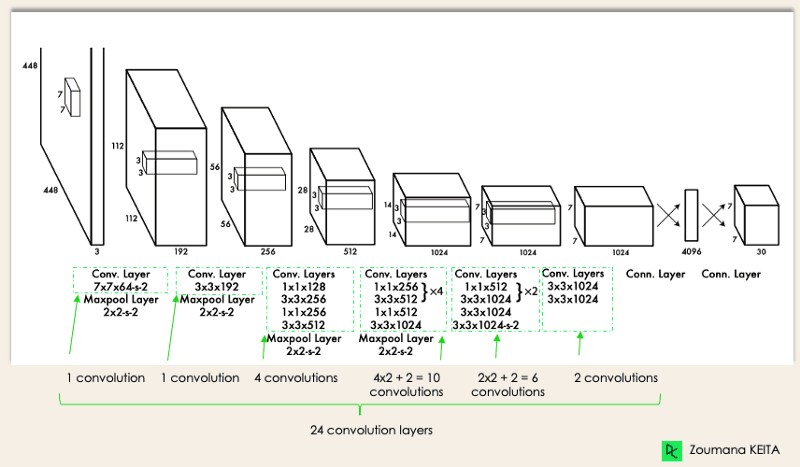


Figure 13: YOLO Architecture(Https://Www.Datacamp.Com/Blog/Yolo-Object-Detection-Explained, n.d.)

The YOLO version implemented on this project is the YOLOv5 version. As it is a new version many improvements have been done on its performance as compared to the previous YOLO versions. YOLOv5 is a simple and lightweight algorithm making it easy to implement on a limited GPU computation capacity. It has proven to have high accuracy as compared to the other models.(*Https://Www.Datacamp.Com/Blog/Yolo-Object-Detection-Explained*, n.d.)

Chart, line chart

Description automatically generated

Figure 14: Comparison of various YOLOv5 versions (Https://Www.Datacamp.Com/Blog/Yolo-Object-Detection-Explained, n.d.)

I chose the YOLOv5s version as I found that the dataset given to us wasn’t as diverse and complicated as compared to a real-world dataset. It was comparatively lighter as it a small model and its designed to be more computationally efficient. As the GPU resources in my Google Colab was less this was the best option to use. It also gave a better trade-off between accuracy and computational efficiency making it well suited for deployment.

**Steps taken to implement YOLOv5:**

* First I divided my training images into two folders **‘train’** and **‘val’** so it as divided into this format. A folder named **‘train\_data’** which consisted of two folders **‘images’** and **‘labels’**. Inside the images and labels were two folders **‘train’** and **‘val’**. These labels were converted in the YOLO format.
* After that the GitHub repository was cloned to reflect the program files of the YOLO v5 using the !git clone https://github.com/ultralytics/yolov5 command.
* After cloning the repository, changes were made in the coco.yaml file to implement changes for a single class and name of the class along with the path of the training and validation folders.
* Then training was performed and results were stored in the **runs/train/exp** file.

**Results of YOLOv5:**

**F1\_curve**

The F1 Confidence Curve in YOLO is a graphical representation of the F1 score as a function of the confidence threshold. F1 score is a mean between precision and recall metrics. The confidence threshold is a value that determines whether an object prediction is considered a true positive or not. If the score is below the threshold it is a false positive and with a sore above the threshold it is a true positive. The Curve shows F1 score with respect to changes in the confidence threshold. A high F1 score indicates that model has good balance of precision and recall. Low score shows that it is underpredicting objects.

Chart

Description automatically generated

Figure 15: F1 Confidence Curve

**PR Curve:**

Precision-recall curve is a common evaluation metric used in object detection to evaluate the performance of the model. The PR-Curve is used to evaluate the performance of the model and also provides insights that which threshold value balances the precision and recall.

Graphical user interface, chart

Description automatically generated with medium confidence

Figure 16: Precision-Recall Curve

**Confusion Matrix**

Chart, treemap chart

Description automatically generated

Figure 17: Confusion Matrix of YOLO

From the confusion matrix we can see that the model predicted UAV 90% of the time and gave a 10% error while detection of UAV. Which is a very good performance considering the model was run for only 10 epochs.

**Results:**

We can see that the losses are reducing per epoch and the precision is increasing which tells us that our model is performing good.

Graphical user interface, chart, line chart

Description automatically generated

Figure 18: Results of the YOLO Model

**Training Batches:**

Graphical user interface, application

Description automatically generated Graphical user interface, application

Description automatically generated

Graphical user interface, application, PowerPoint

Description automatically generated

**Predictions:**

A picture containing text, electronics, display, screenshot

Description automatically generated

A picture containing diagram

Description automatically generated

Graphical user interface, website

Description automatically generated

Figure 19: Predictions

**Mask RCNN Architecture**

It is a deep learning algorithm for object detection and instance segmentation. It is an extension of Faster-RCNN which is a two-stage object detection architecture by adding a parallel branch for predicting an object mask in addition to object classification and bounding box regression.

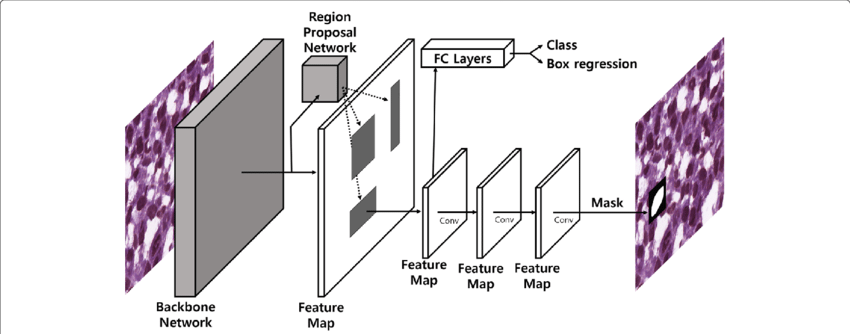


Figure 20:Mask RCNN Architecture(Https://Developers.Arcgis.Com/Python/Guide/How-Maskrcnn- Works/#:~:Text=Mask%20R%2DCNN%20architecture,Label%20with%20a%20confidence%20score., n.d.)

**Feature Extraction:** Input image is passed through a convolutional Neural Network(CNN) to extract a feature map.

**Region Proposal Network:** An object proposal is prepared that is generated by the RPN after processing the feature map. These object proposals are regions in the image that may contain objects.

**ROI Align:** These proposals are fed into a ROI Align layer which crops and aligns regions of the feature map to a fixed size to prepare them for processing by the following layers.

**Classification and Bounding Box Regression:** The ROI aligned feature maps are processed by two fully connected layers for object classification and bounding box regression. The output of these layers is a class label and bounding box coordinates for each proposal.

**Mask Prediction:** In addition to classification and bounding box regression, Mask R-CNN also has a parallel branch for predicting the object mask. The ROI aligned feature maps are pre-processed by several Convolutional and Up-Sampling layers to a binary mask for each object instance.(*Https://Developers.Arcgis.Com/Python/Guide/How-Maskrcnn-Works/#:~:Text=Mask%20R%2DCNN%20architecture,Label%20with%20a%20confidence%20score.*, n.d.)

**Prediction of RCNN**

Timeline

Description automatically generated with low confidence

Predictions were poor as compared to YOLO Architecture. (Also, epochs were less due to GPU Constraints. Training would be better with more epochs.)

**Comparison with YOLO object Detection**

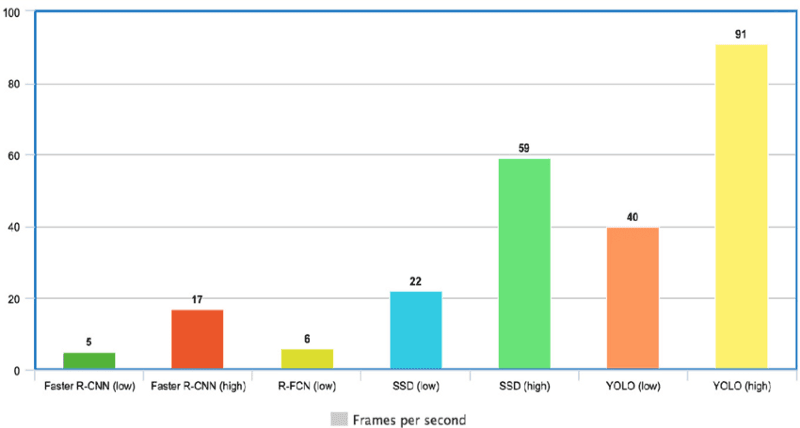


Figure 21: Comparison with other models(Https://Www.Datacamp.Com/Blog/Yolo-Object-Detection-Explained, n.d.)

Because it doesn't deal with complicated pipelines, YOLO is incredibly quick. At 45 frames per second, it can process images (FPS). YOLO is a fantastic choice for real-time processing since it can achieve more than twice the mean Average Precision (mAP) compared to other real-time systems.

With 91 FPS, YOLO is clearly superior to the other object detectors, as shown in the graph above.(*Https://Www.Datacamp.Com/Blog/Yolo-Object-Detection-Explained*, n.d.)

**Why YOLO is Better**

With a very low amount of background mistakes, YOLO's accuracy greatly outpaces that of other cutting-edge models.

YOLO is constantly evolving by releasing newer versions. With those improvements, YOLO went a little further and offered improved generalisation for new domains, making it ideal for applications that require quick and reliable object identification.

# References

Adam Dhalla. (n.d.). *https://ai.plainenglish.io/the-rise-and-fall-of-the-perceptron-c04ae53ea465*.

Amita Kapoor, A. G. S. P. (n.d.). *Deep Learning with Tensorflow and Keras* (Third Edition).

geekycodesco. (n.d.). [*https://geekycodes.in/what-is-the-vgg-19-neural-network/*](https://geekycodes.in/what-is-the-vgg-19-neural-network/).

*https://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/*. (n.d.).

*https://developers.arcgis.com/python/guide/how-maskrcnn-works/#:~:text=Mask%20R%2DCNN%20architecture,label%20with%20a%20confidence%20score.* (n.d.).

*https://stackoverflow.com/questions/41990250/what-is-cross-entropy*. (n.d.).

*https://www.datacamp.com/blog/yolo-object-detection-explained*. (n.d.).

*https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/*. (n.d.).

*https://www.niser.ac.in/~smishra/teach/cs460/2020/lectures/lec19\_1/*. (n.d.).

Michele Cavaioni. (n.d.). [*https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524*](https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524).

MK Gurucharan. (n.d.). [*https://www.upgrad.com/blog/basic-cnn-architecture/*](https://www.upgrad.com/blog/basic-cnn-architecture/).

Sultan, H. H., Salem, N. M., & Al-Atabany, W. (2019). Multi-Classification of Brain Tumor Images Using Deep Neural Network. *IEEE Access*, *7*, 69215–69225.

<https://doi.org/10.1109/ACCESS.2019.2919122>

Tan, M., & Le, Q. v. (2019). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. <http://arxiv.org/abs/1905.11946>

Wikipedia. (n.d.). [*https://en.wikipedia.org/wiki/Sigmoid\_function#cite\_note-Han-Morag\_1995-1*](https://en.wikipedia.org/wiki/Sigmoid_function#cite_note-Han-Morag_1995-1).

Python Script

(Appendix)

FCN Code

Graphical user interface, text, application, Teams

Description automatically generated

A picture containing text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

A picture containing application

Description automatically generated

A picture containing background pattern

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

A picture containing graphical user interface

Description automatically generated

CNN Code

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application

Description automatically generated

A picture containing application

Description automatically generated

Chart, timeline

Description automatically generated

VGG19 Code

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface, text

Description automatically generated



Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Timeline

Description automatically generated

Task 2a and YOLO Implementation

Graphical user interface

Description automatically generated with low confidence

Graphical user interface, application, PowerPoint

Description automatically generated

Text

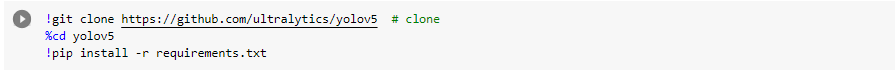
Description automatically generated with medium confidence

Text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated

R-CNN Implementation was done using the anaconda console so couldn’t include the code for that.