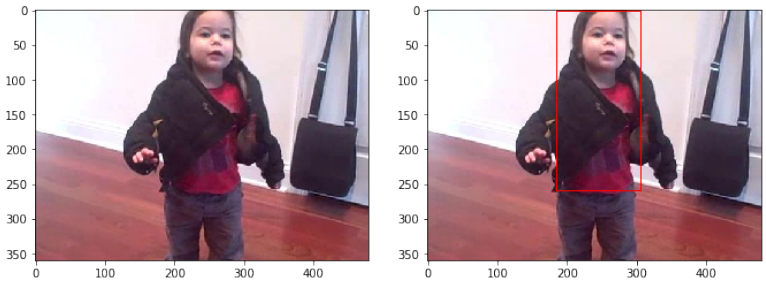
1 Introduction

While technology has improved, the machines take over an important role in our lives. Machines are everywhere and they make our lives easier. We can use the machines in almost all areas and they can do better jobs than people. In almost all areas, we should work with images and videos especially for the real-time systems. To improve the accuracy of these kind of systems, we may track objects. The systems, which track objects, are more robust to changes in pose. So to make the machines act like a human brain, see the around like an human eyes and perceive the objects, object tracker methods can be used while studying with videos, images etc. In last decades, there are several studies on this topic so several methods are improved but still this is a complex topic.



(a) Original Image (b) Object Tracked Image

Figure 1: Simple Example of Object Tracking

In this study, we will make a basic single object tracker. To achieve this, we will use two-frame architecture that means we will us features of two frames at the same time. After we build our tracker using the train and validation videos, we will test the tracker using the test videos. But during these steps, we should take into account the bounding box annotations.

Our goal is predicting the position of the object in the current frame using the position of the object in the previous frame. So our assumption is the wanted object cannot move so fast from frame to frame, so the position of the object in the current frame should be close the position of the object in the previous frame.

To predict the position of the wanted object, we will train a network using the

videos in validation and train dataset. The output of this network will give us the position of the object in the current frame.

To make this task, we will use 263 training, 25 validation and 25 test videos. Each video has frames as images and in the corresponding annotation file, we can access the position of the bounding box. In corresponding annotation file, there is ID of the frame, x1, y1, x2, y2 - which are the coordinates of top - left and bottom - right respectively in each line.

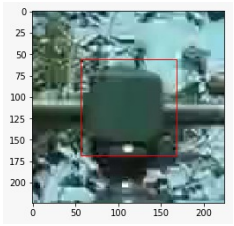
The rest of the paper as follows, The method I followed and details of the solution in Section 2, Experimental results for different parameters and comments on them in Section 3, the weakness of my implementation and the results in Section 4.

2 Implementation Details

To make the object tracker, I firstly read the annotation file and put the information of the IDs and coordinates of the bounding boxes into a dictionary according to the name of the files. After that because of I should take the frames pairwise like (frame0, frame1), (frame1, frame2), (frame2, frame3) etc, I firstly create 2 files with csv extension for train and validation dataset. In this file, I put the path of the first frame, the path of the second frame and the bounding box respectively in each line.

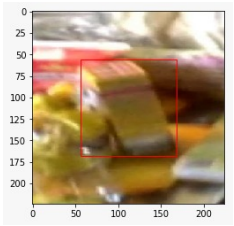
After I read the files that I create as csv file, I convert them to DataFrames and send this DataFrames to custom dataloader that I implement. In this custom dataloader, I read the images, whose paths are specified in the csv extension file, and take the bounding box location information. After that I cropped the all frames according to 2 times enlarged version of the bounding box. Then I found the relative position of the bounding boxes according to the cropped images.

(a) Position of the bounding box on original image



(b) Relative position of the bounding box

(c) Position of the bounding box on original image



(d) Relative position of the bounding box

Figure 2: Position of the bounding box

After that I cropped the images and convert them tensor then return from custom dataloader. After I load the datasets which are train and validation, I extract features. To extract features, I used VGG-16 model, which is pretrained on imgeNet dataset, as a feature extractor. I delete the fully connected layers from the model and change the avgpool layer of the model to the AvgPool2d. At the end, I come up with [1x512] vectors for each frame of the pairs and then I concatenate these features so I come up with [1x1024] vectors. After I extract features and save them in the files, I construct a fully connceted network.

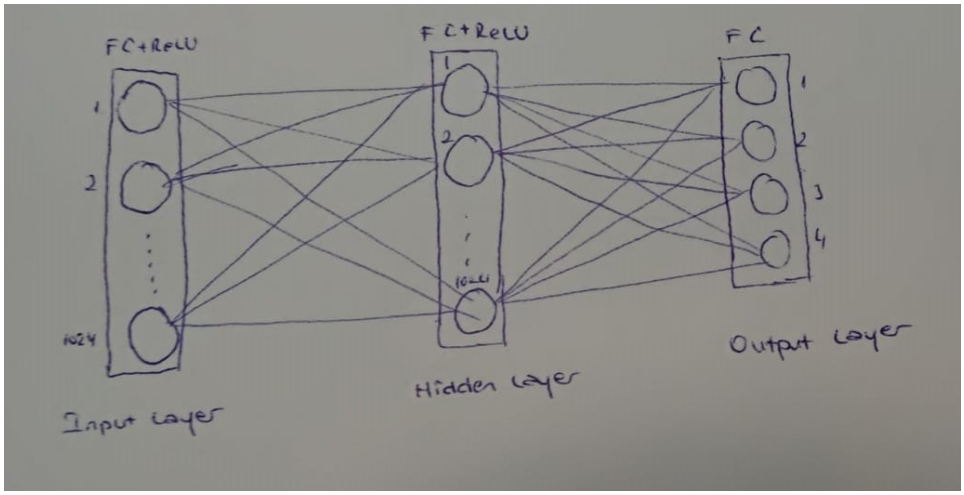


Figure 3: Basic representation of the network

In the figure 3, the basic representation of the fully connected layer that we used in the task is showed. As in the figure 3, the network is constructed as follows;

- FC(1024,1024)
- ReLU
- FC(1024,1024)
- ReLU
- FC(1024,4)

This network maps to bounding box of the second frame, frame1, relative to the search region. The size of the output is [1x4] vector which gives the top left and bottom right corners of the bounding box. This network is trained from scratch. In addition these, I will use mean squared error as loss function and Adam as optimizer.

Not: One of the most important points of this task is to be careful when cropping. Because if we do not take the relative position and if we do not consider the position of the box according to resize and crop operations, the network(see fig. 3 cannot learn.)

In the test stage, I firstly take the images like in train and validation dataset but this time, I just initialize the ground-truth bounding box of the first frame

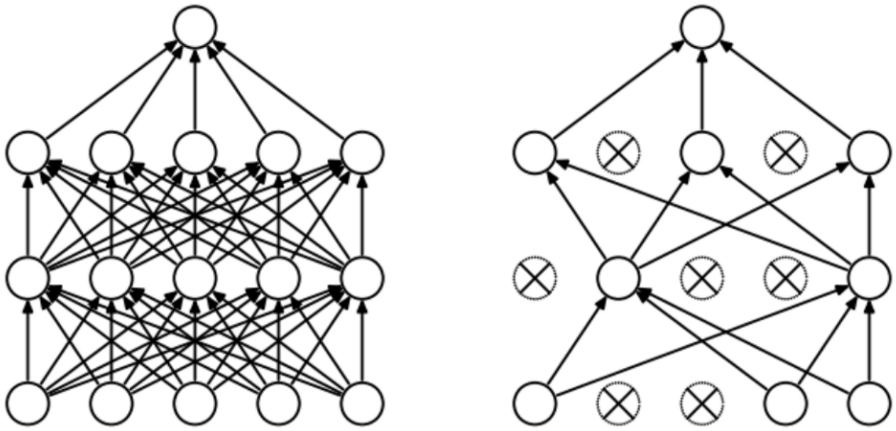
for each video in the test set. I n addition these, I do not shuffle.

After I extract the features, I send them to fully connected network (see fig. 3) whose output gives the bounding box. I use this predicted bounding box for the other frame pairs. This way makes the system track along the frames.

3 Experimental Results

I made a lot of experiments. I want to mention from them briefly.

- I firstly try to change the loss function to the cross entropy loss function. But I learned that cross entropy loss function does not expect a vector as target value. It is for classification which is not appropriate for this situation.
- I give several learning rates and I observe that when I give the high learning rate, the loss increase instead of decrease. When I give small learning rate, the network converge slowly.
- I tried several number of epochs.
- Because of a pretrained model on imageNet dataset is used to extract features without retrain, I thought that this may affect the result because the images in imageNet may not be appropriate for our own dataset. So I retrain the VGG-16 model on our own dataset. The results becomes better but then I thought that this is not the situation that we should do.
- I try SGD instead of Adam optimizer and it slightly affect the result. Optimizer algorithms are used to minimize or maximize the loss. SGD is a variantof gradient descent. It just perform on a small samples instead of all dataset. Onthe other hand, Adam is a gradient-based optimization algorithm and combines the2 SGD extensions: RMSProp and AdaGrad.
- I observe the effect of the dropout layers. This layer cause the ignoring some of the nodes that are selected randomly during training. The main idea of using the dropout layers are preventing the overfitting.



(a) Standard Neural Net (b) After applying dropout.

Figure 4: Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

- I add run the network using several weight decay values. It cause after each epoch, the weights are multiplied by a factor which is smaller than 1.
- I construct the network again and again using several number of layers.
- I used batch normalization which is a method that is used to normalize the inputs of each layer.
- Instead of ReLU, I used some other activation funcitons.

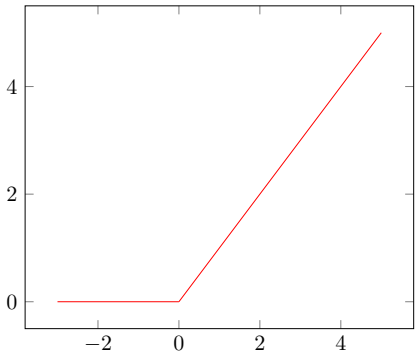


Figure 5: The graph of the ReLU

$$ReLU(x) = max(0, x) \tag{1}$$

In figure 5 shows the plot of the ReLU and the equation 1 is the formula of ReLU. ReLU is a most widely used activation function.

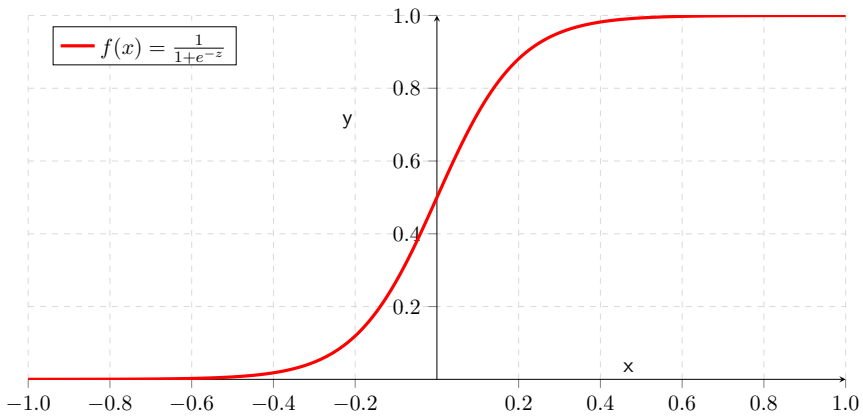


Figure 6: The plot of the sigmoid function

$$h_\theta(x) = \frac{1}{1 + e^{-z}} \tag{2}$$

In the plot 6, the plot of the sigmoid funciton is shown and in the equation 2, the formula of the sigmoid is given. Sigmoid is also an activation fuction.

When I compare the ReLU and Sigmoid(see eq. 1 and 2), I see that the ReLU gives better result. So I did a small search the reasons o it and what is the difference between ReLU and Sigmoid.

- ReLU does not vanish the gradients.
- Sigmoid cannot blow up activation.
- The cost of the ReLU is less than the cost of Sigmoid.
- In the models, The ReLU perform better convergence than the Sigmoid.

- I used a learning rate scheduler instead of a constant learning rate. Adaptive learning rate whose other name is learning rate scheduler can increase

the performance and decrease the training time. We usually use the adaptive learning rate to reduce the learning rate overtime and this provide the model learn quicker.

Because of there are a lot of experiments, I just give the results which gives the worst result, average result and the best result.

In my first try, I train the basic structure.(see fig. 3) This network is just created using the below layers respectively:

- FC(1024,1024)
- ReLU
- FC(1024,1024)
- ReLU
- FC(1024,4)

As I mentioned before, I use Adam optimizer and mean squared error as loss function. (see the eq.3)

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}\Big(y_i - p_i\Big)^2 \tag{3}$$

I found the results as below:

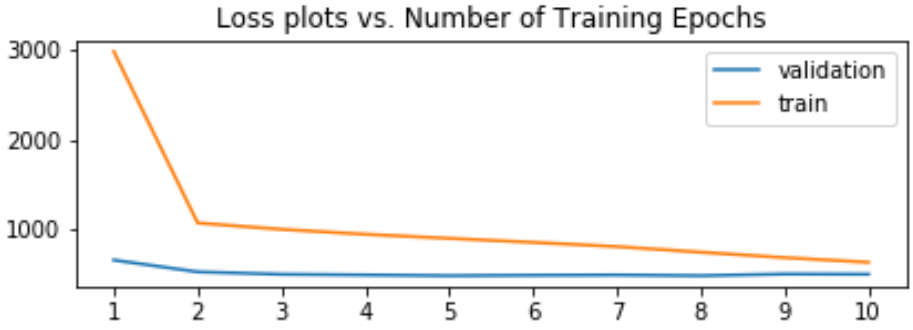


Figure 7: Validation and Train Loss of the Network

Epoch	Learning Rate	Avg. Test Loss
10	0.01	12000

Table 1: The Worst Case

When we look at the loss graph (see fig. 7) we see that although the train loss decrease, the validation loss does not decrease and there is some oscillations. As we mentioned above if we cannot do cropping well, the network cannot learn. So I check it and I am sure that the cropping operation is truly made. To check this I wrote a small piece of code and I take the display some images. And I become to sure that I made the cropping truly so I think that there are some other reasons that cause this situation.

- The weights of the fully connected network (see fig. 3) are initialized randomly. It means the network is trained from scratch. So the time for learning takes more.
- The epoch size is so small for a network which is trained from scratch. (see table 1)

- Learning rate may not be appropriate for this model. Other learning rates should be tried. (see table 1)
- The network is not complex. It just has one input layer, one hidden layer and one output layer as shown in fig. 3. To get more accurate results, the network should become more complex form.
- There is not any regularization operation in the network such as adding dropout layers and weight decay.

So I take the test results of this experiment as worst case and make a 3 video from 3 different class, putting the predicted box on top of the frames and some frames are as below:

(a) First Frame From The Video

(b) Second Frame From The Video

(c) Third Frame From The Video

(d) Fourth Frame From The Video

Figure 8: Some frames from test video in the worst case

To get the average case, I add some extra layers to the network as below:

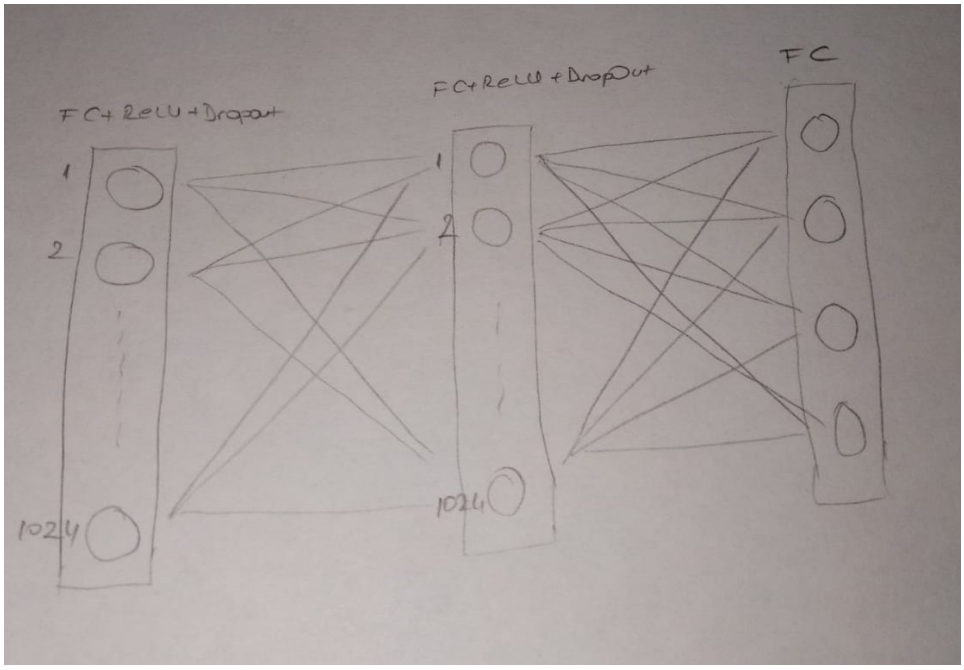


Figure 9: Basic representation of the average case network

To get more accurate result, I tried some regularization techniques and I get the best result when I add dropout layers after each fully connected layer. So the structure of the network become like in figure 9.

I tried different loss functions and the most appropriate is the mean squared error loss function. (see eq. 3)

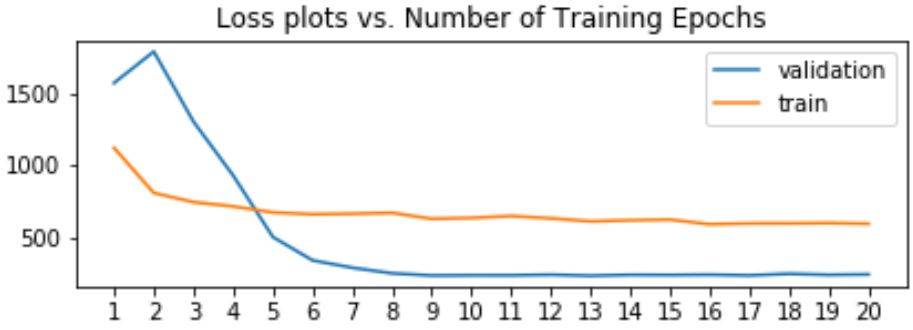


Figure 10: Validation and Train loss plot of the average case network

In the figure 10 shows the validation and train loss plot of the network. To get this result I also play with parameters such as learning rate etc. When I increase the epoch size, I see that model start to memorize so I face with an overfitting problem in this case.

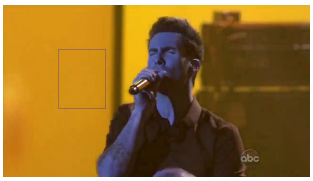
Epoch	Learning Rates	Avg. Test Loss
20	0.001	5000

Table 2: The Average Case

The above table(see table 2 shows the parameters that I used during training.

(a) First Frame From The Video

(b) Second Frame From The Video



(c) Third Frame From The Video

(d) Fourth Frame From The Video

Figure 11: Some frames from test video in the average case

To see the results, I display some of the frames from a video. Although the results are near to worst case for this image, I get a better result for another video. So of course that the result may change from video to video.

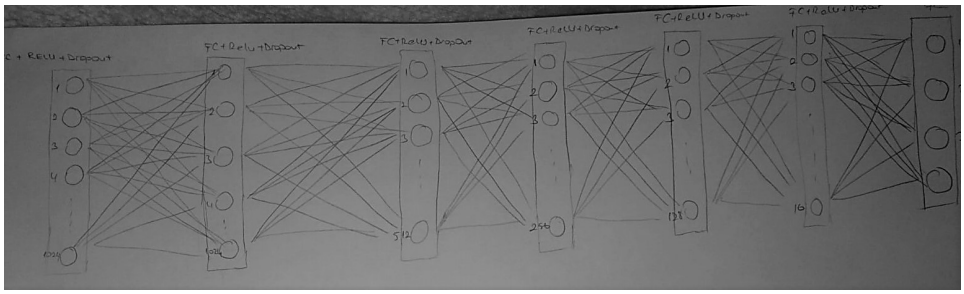


Figure 12: Basic representation of the best case network

To get more accuracy, I made the network more complex so to make it more complex, I add more fully connected, ReLU and Dropout layers. The basic structure of the new network is shown in the figure 12. As we see in this figure, I add 4 more fully connected, ReLU and Dropout layers. In addition these, instead of using a constant learning rate, I used learning rate scheduler and in addition these I slightly increase the number of epoch. So I get more similar results to accurate results at the end.

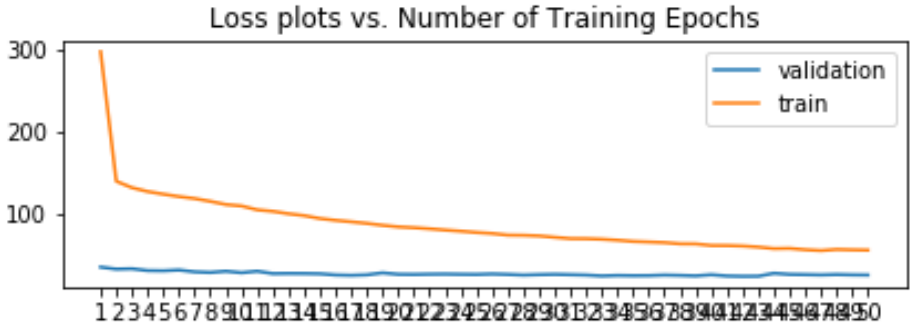


Figure 13: Validation and Train loss plot of the best case network

Figure 13 shows the validation and loss plots of the best case. When we compare with the other graphs we see that there is less oscillations in the plot. And because of the structure of the network is completely different than the other and more complex, we have more proper plot.

Epoch	Learning Rate	Avg. Test Loss
50	Learning rate scheduler	2000

Table 3: The Best Case Results

In addition the complex network, I also increase the number of epochs and I use learning rate scheduler instead of a constant learning rate as the above. (see table 3) And to see the results, I just display the 4 frames of a video.

(a) First Frame From The Video

(b) Second Frame From The Video

(c) Third Frame From The Video

(d) Fourth Frame From The Video

Figure 14: Some frames from test video in the average case

4 Conclusion

In the early stages, cropping the image and taking the relative bounding box comes a bit hard because I actually did not understand how should I compute the position of the relative bounding box. But after I understand the main topic behind it, I could easily construct the function.

The second point that becomes difficult to me is that taking the frames as pairs. At the first, I consider it as it is so hard to take the images as pairs because there is no source code about it but after I search properly and after several tries, I achieve to take the images as pairs and I recognize that it is not hard as much as I thought.

The weakness is that the network that we crate to train which is fully connected network is trained from scratch and the weights of the network are initialized randomly. So according to these weights, the result may change in each iteration. Another weakness is that, the steps takes long time to complete.

As a future work, to get more accurate results, we can do:

- We may fine tune the VGG-16 model which we extract features. If we retrain the convolution layers of it, we may get more accurate features. In addition these, we may add more layers to fully connected network. It helps us to get more accuracy.

- We may use other convolutional eural network models to extraxt features such as ResNet18, ResNet34 etc.

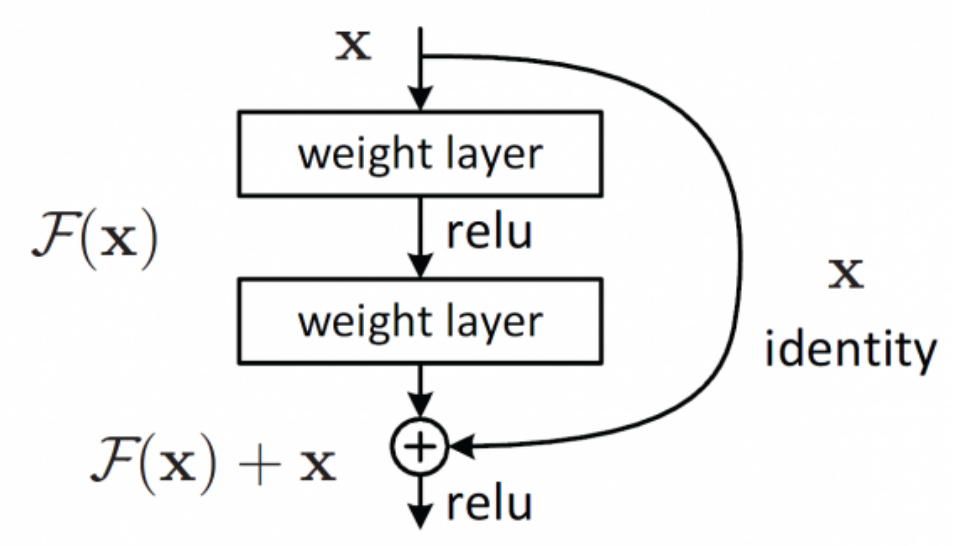


Figure 15: The basic representation of the ResNet model
https://neurohive.io/en/popular-networks/resnet/

ResNet is a deeper model than VGG-16 and when the model is huge, the gradients become too small so when the model doesn't know how to update weights anymore but this is not the case in ResNet model because there are residual layers as shown in figure 15. Residual layers provide gradient flows through layers without vanishing during backpropagation so we may use the ResNet model instead of VGG-16.

You can access the videos from the link: https://drive.google.com/open?id=1q6K$_b Ej-zHG0YEh65Hc1RnpzVDWhjOQX$