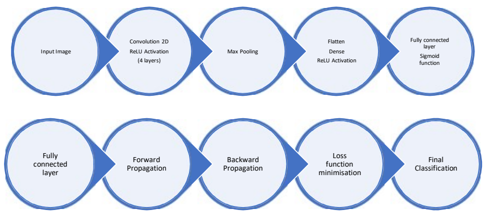
Detection of Aircraft using CNN

Ujjwal Kumar
Guide: Prof. J Indu
Department of Civil Engineering, IIT Bombay

Abstract

To address the issue of poor accuracy and slow execution time in detection of aircraft encountered in traditional methods, this project is based on using a CNN structure for training and detecting objects (aircraft in this case). The main challenge for detecting aircraft images is the background which is similar to aircraft and the altitude from which images are captured which makes aircrafts look very small. So, to cater to these challenges we use multifeatured fusion convolutional neural network. In this the shallow and deep layer features are fused at same scale after sampling to overcome the problems of low dimensionality in deep layers and the inadequate details of small objects. To increase the speed of training and detection for validating, we preprocess data with mean normalization ($x_i = \frac{x_i - \mu}{range\ of\ x_i}, \mu\ is\ mean\ of\ x_i$) and feature scaling($x_i = \frac{x_i}{range\ of\ x_i}$). Additionally, the training of network was carried out on data sets with different spatial resolutions and also images captured from different altitude.

Methodology




```
# Instantiate the model
image_size=images[0].shape
print(image_size)
model = cnn(image_size,N_LAYERS)
```

(20, 20, 3)

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 18, 18, 20)	560
activation_1 (Activation)	(None, 18, 18, 20)	0
conv2d_2 (Conv2D)	(None, 16, 16, 44)	7964
activation_2 (Activation)	(None, 16, 16, 44)	0
conv2d_3 (Conv2D)	(None, 14, 14, 68)	26996
activation_3 (Activation)	(None, 14, 14, 68)	0
conv2d_4 (Conv2D)	(None, 12, 12, 92)	56396
activation_4 (Activation)	(None, 12, 12, 92)	0
max_pooling2d_1 (MaxPooling2	(None, 6, 6, 92)	0
flatten_1 (Flatten)	(None, 3312)	0
dense_1 (Dense)	(None, 120)	397560
activation_5 (Activation)	(None, 120)	0
dense_2 (Dense)	(None, 1)	121
activation_6 (Activation)	(None, 1)	0

Total params: 489,597
Trainable params: 489,597
Non-trainable params: 0

Pseudo code for backpropagation:

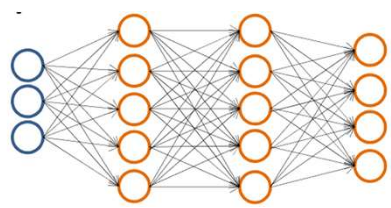
initialize network weights (with some random small values)
do
for Each training example named ex
prediction = neural network output (network, ex)
actual = known output (ex)
compute error (prediction - actual) at the output units
compute {Delta w_{h}} for all weights from hidden layer to output layer //backward pass
compute {Delta w_{i}} for all weights from input layer to hidden layer //backward pass
update network weights // except the input layer
until all examples classified correctly or any custom defined stopping criteria
return the network

The Loss function *J* is taken as Euclidean distance function and accordingly the weight parameters are updated. In this Adams optimisation algorithm was used because rest other optimisation algorithm contains a hyperparameter (similar to learning rate) and finding the optimised value of that hyperparameter again either becomes another optimisation problem or using hit and trial method becomes computationally expensive.

$J(\theta_0, \theta_1, \ldots \theta_n) = \frac{1}{2m}\sum_{i=1}^{m}(h(x)_i - y_i)^2$ where $h(x)_i$ is predicted & y_i is known value

Layer wise computation:

-



Assuming each layer is named $a^{(1)}, a^{(2)}, a^{(3)}, a^{(4)}$ where $a^{(1)}$ is input layer $a^{(4)}$ is output layer and rest are hidden layers connected with each other; θ^n connects layer n to n+1

Forward propagation equations:

$a^{(1)} = x$ where x is input layer

$z^{(2)} = \theta^{(1)} * a^{(1)} \Rightarrow a^{(2)} = g(z^{(2)})\ \left[add\ a_0^{(2)}\ to\ a^{(2)}\ as\ bias\ unit\right] \rightarrow g(x) - sigmoid\ of\ x$

$z^{(3)} = \theta^{(2)} * a^{(2)} \Rightarrow a^{(3)} = g(z^{(3)})\ \left[add\ a_0^{(3)}\ to\ a^{(3)}\ as\ bias\ unit\right]$

$z^{(4)} = \theta^{(3)} * a^{(3)} \Rightarrow a^{(4)} = g(z^{(4)}) = h_\theta(x)$

Backpropagation equations:

$\delta_j^{(l)} \rightarrow error\ of\ node\ j\ in\ layer\ l$

$\delta^{(4)} = a^{(4)} - y$

$\delta^{(3)} = (\theta^{(3)})^T * \delta^{(4)} .* \ g'(z^{(3)}) \quad .*\ is\ element\ wise\ multiplication$

$\delta^{(2)} = (\theta^{(2)})^T * \delta^{(3)} .* \ g'(z^{(2)}) \quad g'(z^{(n)}) = a^{(n)} .* (1 - a^{(n)})$

No $\delta^{(1)}$ term as input layer remains the same

Updating parameter equation:

$for\ i = 1{:}m \ \leftarrow (x^i, y^i)$

$a^{(1)} = x^i$

$forward\ computation\ to\ compute\ a^l\ for\ l = 2,3,4 \ldots L$

$using\ y^i\ compute\ \delta^{(L)} = a^{(L)} - y^i$

$compute\ \delta^{(L-1)}, \delta^{(L-2)}, \ldots \delta^{(2)}$

$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

end

$D_{ij}^{(l)} = \frac{1}{m} D_{ij}^{(l)} + \frac{\lambda}{m} \theta_{ij}^{(l)} \quad if\ j \neq 0$

$D_{ij}^{(l)} = \frac{1}{m} D_{ij}^{(l)} \quad if\ j = 0$

Finally, minimization of loss function $J(\theta)$ was done using 'Adams' optimization algorithm and loss was calculated in terms of binary cross entropy loss.

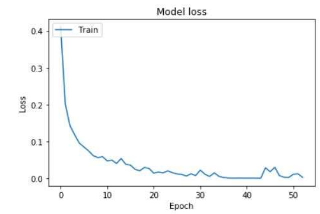
$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^i * log(h_\theta(x^i)) + (1 - y^i) * \log(1 - (h_\theta(x^i))]$

where $(h_\theta(x^i))$ is predicted value of i_{th} dataset and y^i is the known value of i_{th} dataset

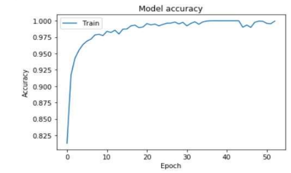
Results

The overall accuracy on the validation data set obtained was around 98.5% while the accuracy on training data set was around 75.32%.

```
# Plot training & validation loss values
plt.plot(history.history['loss'])
#plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
# Plot training & validation accuracy values
plt.plot(history.history['acc'])
#plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Limitations & Future Improvements:

- Major limitation is due to shallow layer which has led to overfitting of equation on dataset
- Shifting architecture similar to VGG net would increase parameter from 0.5M to 146M, thus a significantly higher accuracy can be obtained
- Training dataset contains a total of 32000 images with around 20% of aircraft images, thus due to low percentage model has poor generalisation
- Augmentation applied on dataset might help model to generalise better
- Resolution of dataset is 20*20*3, thus higher resolution data might provide spatial depth
- Bounding box problem can be extended using sliding window technique applied on this model

Conclusion

CNN architecture does provide a more accurate solution for identification of objects in images. Although training the model does take some time but it's time can be decreased by using GPU based computation and. Post training, the prediction takes very little time. This method is definitely quite accurate, fast and the model once trained its weight can be saved and a predicting software GUI can be made out of it.

Bibliography

1. X. Li, S. Wang, B. Jiang and X. Chan, "Airplane detection using convolutional neural networks in a coarse-to-fine manner," 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, 2017, pp. 235-239. doi:10.1109/ITNEC.2017.8284943
2. Xu, Yuelei et al. "Rapid Airplane Detection in Remote Sensing Images Based on Multilayer Feature Fusion in Fully Convolutional Neural Networks." Sensors (Basel, Switzerland) vol. 18,7 2335. 18 Jul. 2018, doi:10.3390/s18072335
3. Chen, Xueyun, Shiming Xiang, Cheng-Lin Liu and Chunhong Pan. "Aircraft Detection by Deep Convolutional Neural Networks." IPSJ Trans. Computer Vision and Applications 7 (2014): 10-17.
4. Dataset obtained from Kaggle - https://www.kaggle.com/rhammell/planesnet/data