GOING DEEPER WITH CONVOLUTIONS

Shruti Sharma, Lakshya Garg, Sidisha Shyam Barik

Overview

- A deep convolutional neural network architecture
- Classification and detection for ILSVRC14
- Improved utilization of the computing resources inside the network while increasing size, both depth and width
- Significantly more accurate than state of the art
- 22 layers deep when counting only layers with parameters and 27 layers deep if counting the pooling layers.
- The overall number of layers (independent building blocks) used for the construction of the network is about 100

Objective

To create better deep learning models with improved performance of classification and detection

One solution:

increase the size of network in both direction but problems:

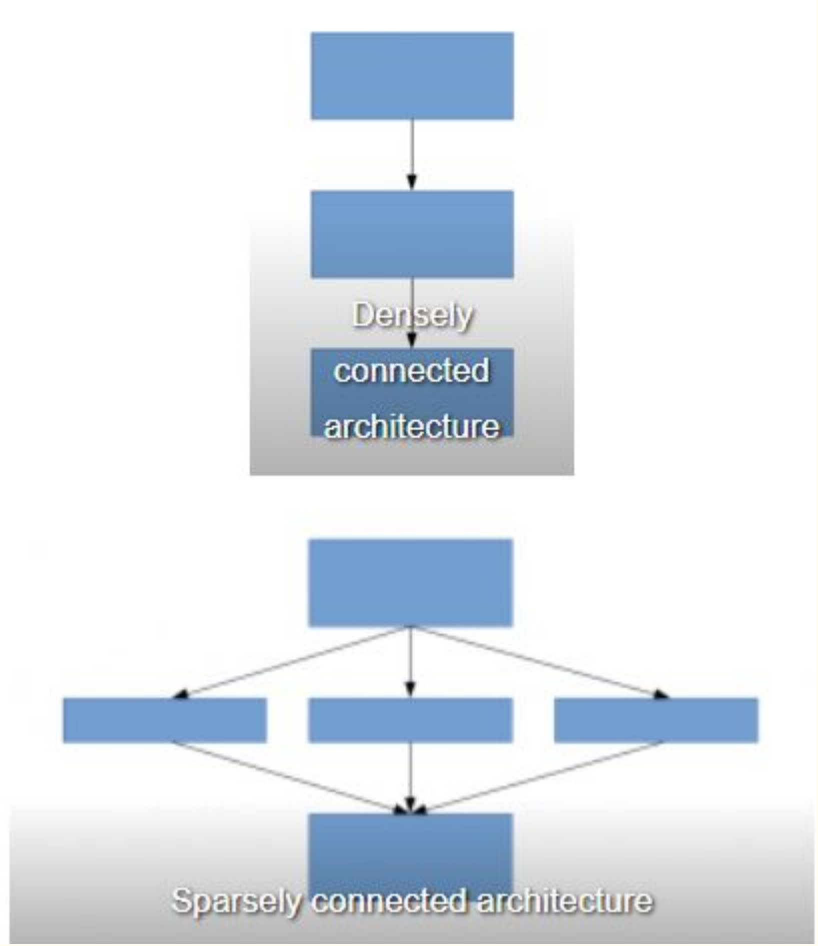
- Bigger the model, more prone it is to overfitting.
- Increasing the number of parameters (increasing existing computational resources)

Solution/Methodology

Sparsely connected network architectures which will replace fully connected network architectures

Helps maintain the “computational budget”, while increasing the depth and width of the network

Auxiliary training- auxiliary classifiers (applied softmax to the outputs of two of the inception modules, and computed an auxiliary loss over the same labels) auxiliary loss.



How to decide the number of convolution and pooling layers

To calculate receptive field, the formula is as follows,

$$\mathtt{OutputWidth} = \left(\frac{\mathtt{W} - \mathtt{Fw} + 2\mathtt{P}}{\mathtt{Sw}}\right) + 1$$

$$\mathtt{OutputHeight} = \left(\frac{\mathtt{H} - \mathtt{Fh} + 2\mathtt{P}}{\mathtt{Sh}}\right) + 1$$

To calculate pooling layer, the formula is as follows,

$$\mathtt{OM} = \left(\frac{\mathtt{IM} + 2\mathtt{P} - \mathtt{F}}{\mathtt{S}}\right) + 1$$

- OM $\rightarrow$ Output Matrix
- IM $\rightarrow$ Input Matrix
- P $\rightarrow$ Padding
- F $\rightarrow$ Filter
- S $\rightarrow$ Stride

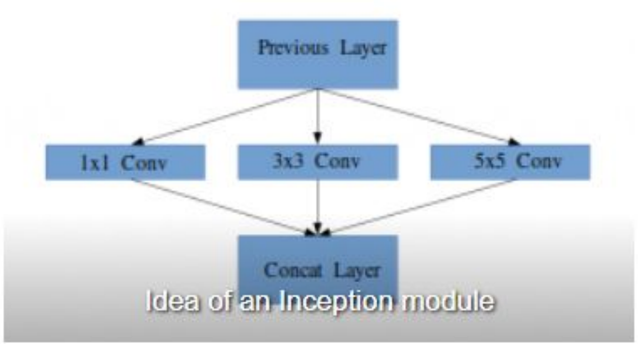
Proposed Architecture

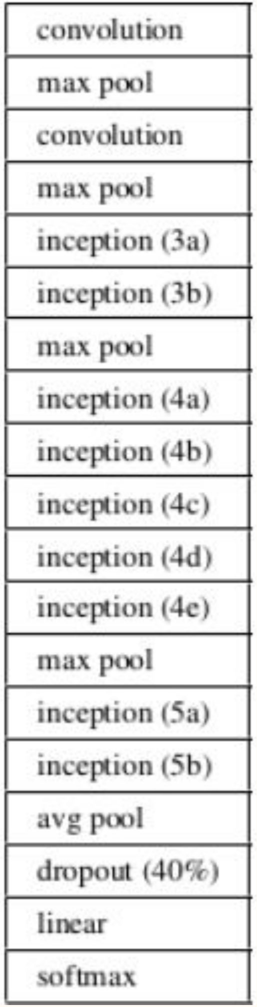
New architecture proposed – GoogLeNet or Inception v1

convolutional neural network (CNN) which is 27 layers deep

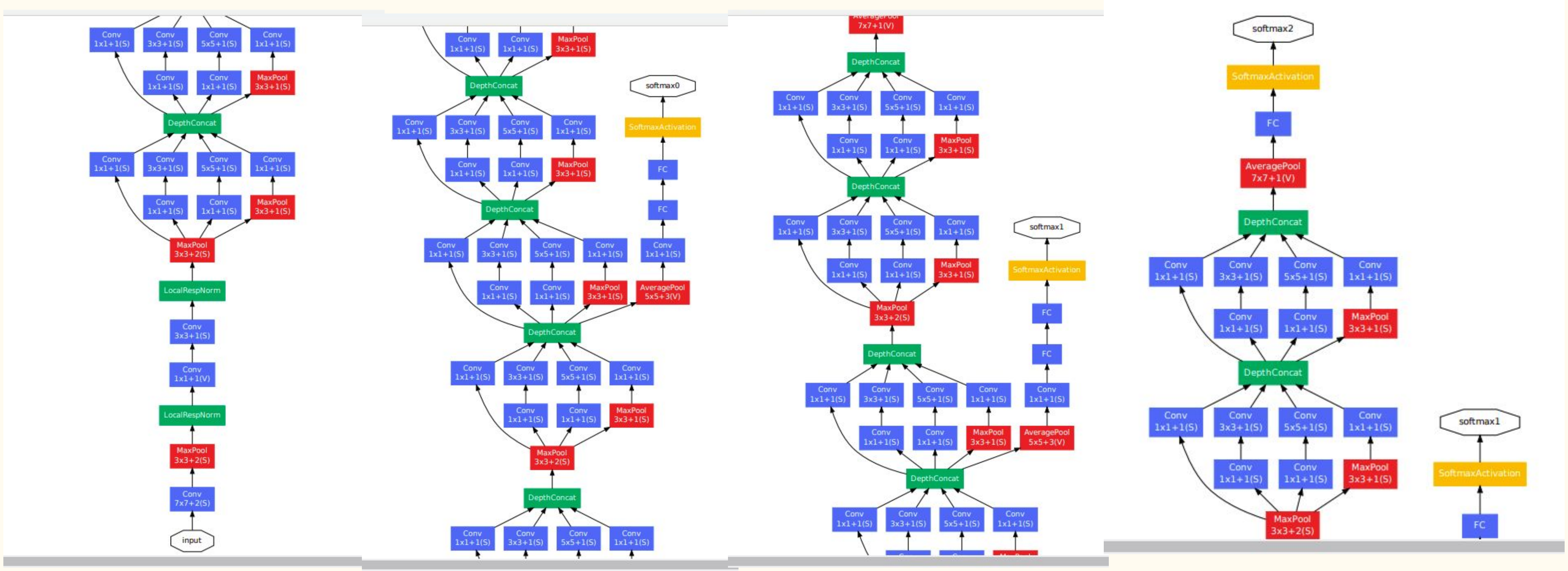
Basic idea - Inception layer

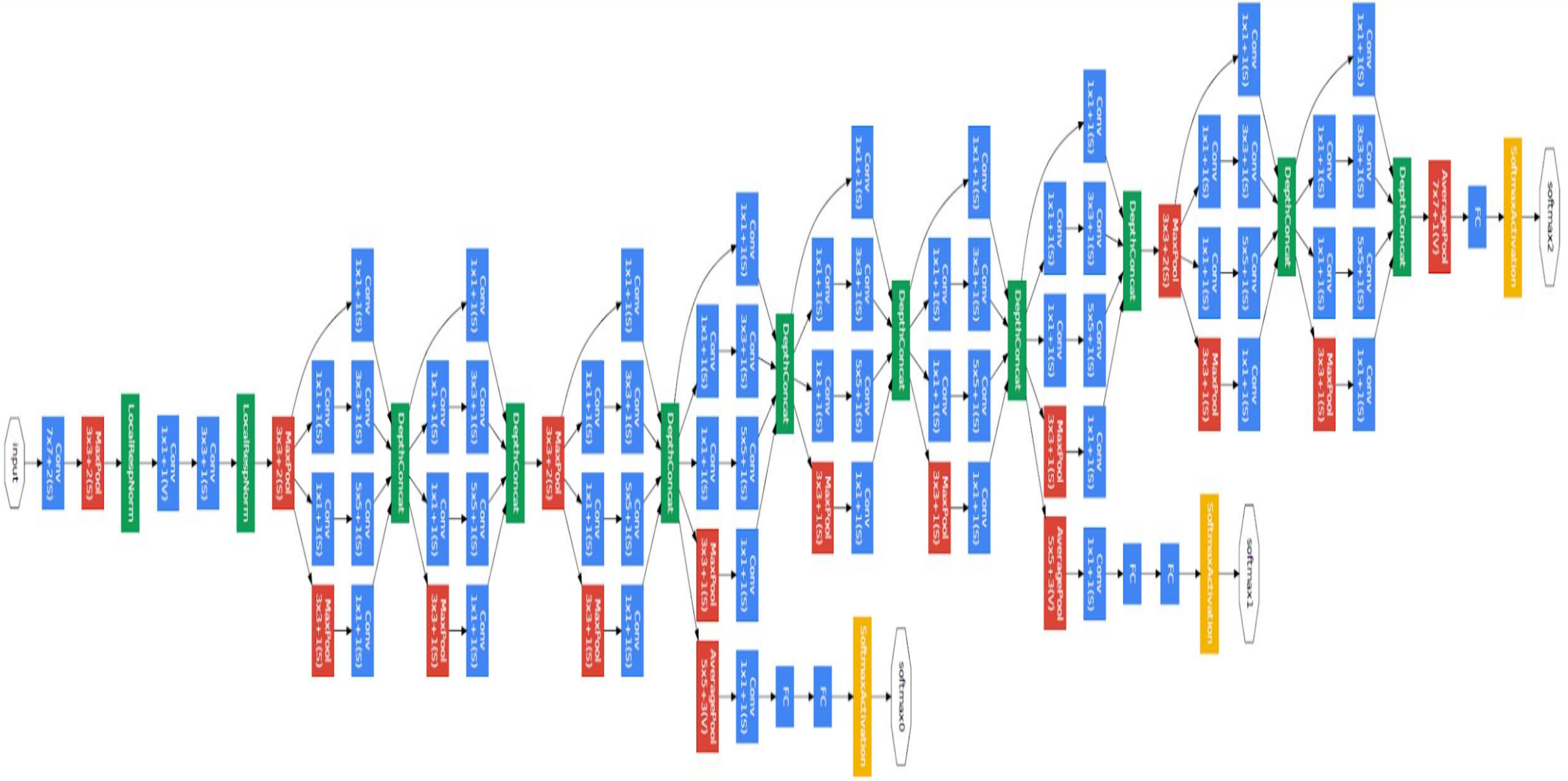
"(Inception Layer) is a combination of all those layers (namely, 1×1 Convolutional layer, 3×3 Convolutional layer, 5×5 Convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage."





The Architecture -



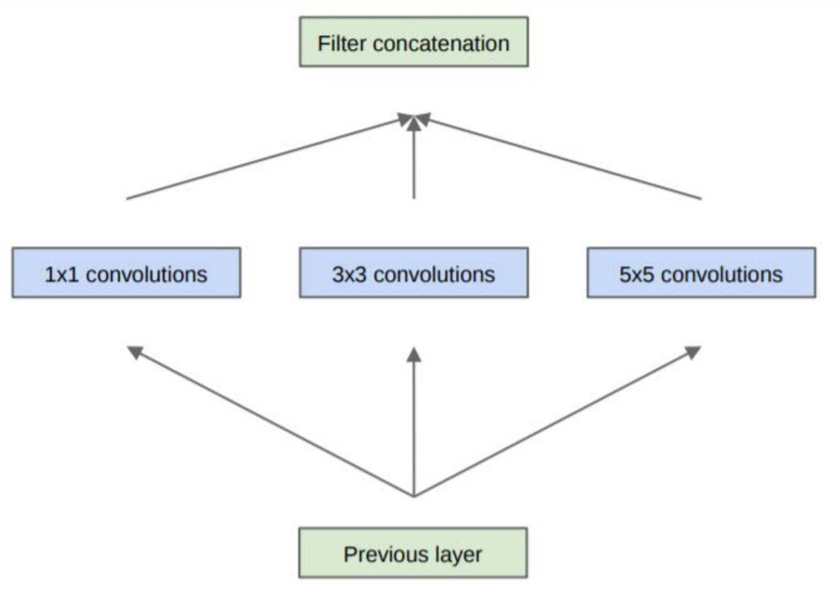

input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
FC
FC
SoftmaxActivation
softmax0
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
FC
FC
SoftmaxActivation
softmax1
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
DepthConcat
AveragePool 7x7+1(V)
FC
SoftmaxActivation
softmax2

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

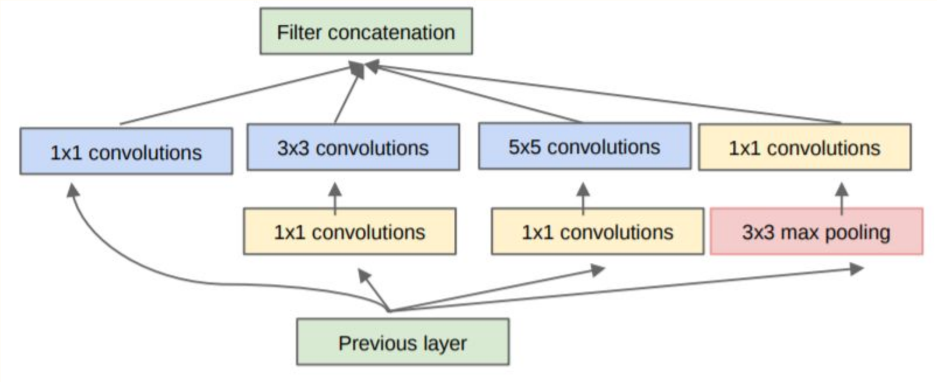
Table 1: GoogLeNet incarnation of the Inception architecture

Inception Module

Initial Idea



Modified (Inception Module)



Add-ons

-1×1 Convolutional layer before applying another layer

-A parallel Max Pooling layer

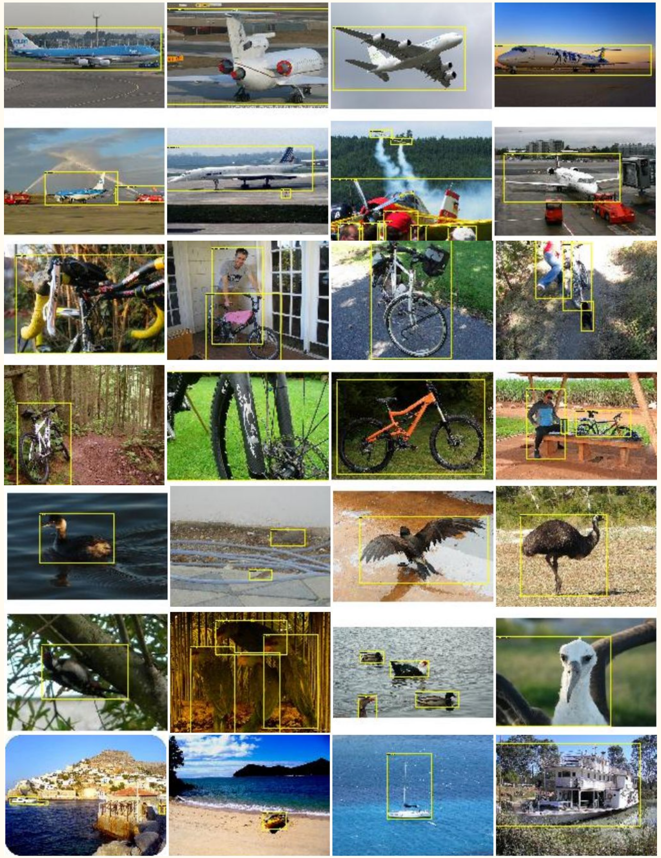
Dataset used

Cifar 10

60,000 32x32 color images in 10 different classes

cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks

6,000 images of each class



Results Obtained

```
Epoch 5/5

Epoch 00005: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.4801 - output_loss: 1.5500 - auxilliary_output_1_loss: 1.5560 - auxilliary_output_2_loss: 1.
Epoch 1/5

Epoch 00001: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.4753 - output_loss: 1.5415 - auxilliary_output_1_loss: 1.5642 - auxilliary_output_2_loss: 1.
Epoch 2/5

Epoch 00002: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.4810 - output_loss: 1.5475 - auxilliary_output_1_loss: 1.5723 - auxilliary_output_2_loss: 1.
Epoch 3/5

Epoch 00003: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.4105 - output_loss: 1.5016 - auxilliary_output_1_loss: 1.5340 - auxilliary_output_2_loss: 1.
Epoch 4/5

Epoch 00004: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.3776 - output_loss: 1.4728 - auxilliary_output_1_loss: 1.5282 - auxilliary_output_2_loss: 1.
Epoch 5/5

Epoch 00005: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.3410 - output_loss: 1.4562 - auxilliary_output_1_loss: 1.4890 - auxilliary_output_2_loss: 1.
Epoch 1/5

Epoch 00001: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.3785 - output_loss: 1.4792 - auxilliary_output_1_loss: 1.4961 - auxilliary_output_2_loss: 1.
Epoch 2/5

Epoch 00002: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.3079 - output_loss: 1.4296 - auxilliary_output_1_loss: 1.4693 - auxilliary_output_2_loss: 1.
Epoch 3/5

Epoch 00003: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.2879 - output_loss: 1.4306 - auxilliary_output_1_loss: 1.4350 - auxilliary_output_2_loss: 1.
Epoch 4/5

Epoch 00004: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.2603 - output_loss: 1.4090 - auxilliary_output_1_loss: 1.4333 - auxilliary_output_2_loss: 1.
Epoch 5/5

Epoch 00005: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 18s 4ms/step - loss: 2.1977 - output_loss: 1.3600 - auxilliary_output_1_loss: 1.4207 - auxilliary_output_2_loss: 1.
```

```
)0 - auxilliary_output_1_loss: 1.5560 - auxilliary_output_2_loss: 1.5444 - output_acc: 0.4198 - auxilliary_output_1_acc: 0.4220 - auxilliary_output_2_acc: 0.4364

.5 - auxilliary_output_1_loss: 1.5642 - auxilliary_output_2_loss: 1.5485 - output_acc: 0.4308 - auxilliary_output_1_acc: 0.4316 - auxilliary_output_2_acc: 0.4308

'5 - auxilliary_output_1_loss: 1.5723 - auxilliary_output_2_loss: 1.5394 - output_acc: 0.4336 - auxilliary_output_1_acc: 0.4270 - auxilliary_output_2_acc: 0.4352

.6 - auxilliary_output_1_loss: 1.5340 - auxilliary_output_2_loss: 1.4956 - output_acc: 0.4384 - auxilliary_output_1_acc: 0.4346 - auxilliary_output_2_acc: 0.4454

!8 - auxilliary_output_1_loss: 1.5282 - auxilliary_output_2_loss: 1.4879 - output_acc: 0.4512 - auxilliary_output_1_acc: 0.4490 - auxilliary_output_2_acc: 0.4530

;2 - auxilliary_output_1_loss: 1.4890 - auxilliary_output_2_loss: 1.4604 - output_acc: 0.4546 - auxilliary_output_1_acc: 0.4580 - auxilliary_output_2_acc: 0.4618

)2 - auxilliary_output_1_loss: 1.4961 - auxilliary_output_2_loss: 1.5017 - output_acc: 0.4544 - auxilliary_output_1_acc: 0.4496 - auxilliary_output_2_acc: 0.4450

)6 - auxilliary_output_1_loss: 1.4693 - auxilliary_output_2_loss: 1.4586 - output_acc: 0.4672 - auxilliary_output_1_acc: 0.4576 - auxilliary_output_2_acc: 0.4480

)6 - auxilliary_output_1_loss: 1.4350 - auxilliary_output_2_loss: 1.4227 - output_acc: 0.4680 - auxilliary_output_1_acc: 0.4676 - auxilliary_output_2_acc: 0.4720

)0 - auxilliary_output_1_loss: 1.4333 - auxilliary_output_2_loss: 1.4042 - output_acc: 0.4776 - auxilliary_output_1_acc: 0.4776 - auxilliary_output_2_acc: 0.4770

)0 - auxilliary_output_1_loss: 1.4207 - auxilliary_output_2_loss: 1.3717 - output_acc: 0.4960 - auxilliary_output_1_acc: 0.4850 - auxilliary_output_2_acc: 0.4906
```

New obtained accuracy

```
980
Epoch 6/8

Epoch 00006: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 19s 4ms/step - loss: 0.7110 - output_loss: 0.3359 - auxilliary_output_1_loss: 0.
7099 - auxilliary_output_2_loss: 0.5402 - output_acc: 0.8840 - auxilliary_output_1_acc: 0.7398 - auxilliary_output_2_acc: 0.8
098
Epoch 7/8

Epoch 00007: LearningRateScheduler setting learning rate to 0.01.
5000/5000 [==============================] - 19s 4ms/step - loss: 0.8204 - output_loss: 0.4266 - auxilliary_output_1_loss: 0.
7226 - auxilliary_output_2_loss: 0.5901 - output_acc: 0.8542 - auxilliary_output_1_acc: 0.7422 - auxilliary_output_2_acc: 0.7
908
Epoch 8/8

Epoch 00008: LearningRateScheduler setting learning rate to 0.0096.
5000/5000 [==============================] - 19s 4ms/step - loss: 0.6166 - output_loss: 0.2646 - auxilliary_output_1_loss: 0.
6850 - auxilliary_output_2_loss: 0.4884 - output_acc: 0.9110 - auxilliary_output_1_acc: 0.7492 - auxilliary_output_2_acc: 0.8
270
```

Conclusion

Due to unavailability of a good computing system, we ran our code on Google Colaboratory.

Since the RAM alloted is 12 GB only, we took samples of 5000 train images and 500 test images and iterated them 10 times to cover the whole dataset.

-Our model gave 49.55 % accuracy. We are working on increasing our accuracy.

-After re-running, the new accuracy obtained is 91%.

References

1. https://arxiv.org/pdf/1409.4842.pdf
2. https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/
3. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/#data