GoogLeNet - Inception Architecture

Deep Learning - SS022

Dr.-Ing. Mehdi Maboudi, Dr. Pedro Diaz

Table of Contents

Introduction

Intro

GoogLeNet or Inception V1 was introduced by Google in 2014.

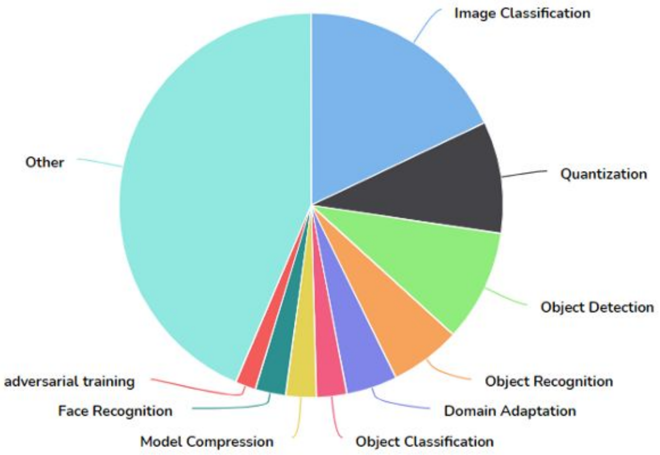
Winner of ILSVRC 2014 image classification challenge.

It is a 22-layer deep CNN based on the paper "***Going Deeper With Convolutions***"[1].

Used for computer vision tasks - image classification and object detection.

It uses ***Inception*** modules to improve efficiency.

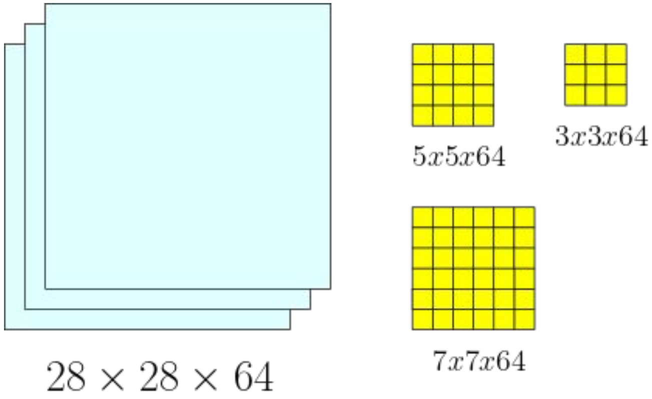
People use GoogLeNet for



TASK	PAPERS	SHARE
Image Classification	21	17.95%
Quantization	11	9.40%
Object Detection	11	9.40%
Object Recognition	7	5.98%
Domain Adaptation	5	4.27%
Object Classification	3	2.56%
Model Compression	3	2.56%
Face Recognition	3	2.56%
adversarial training	2	1.71%

Source

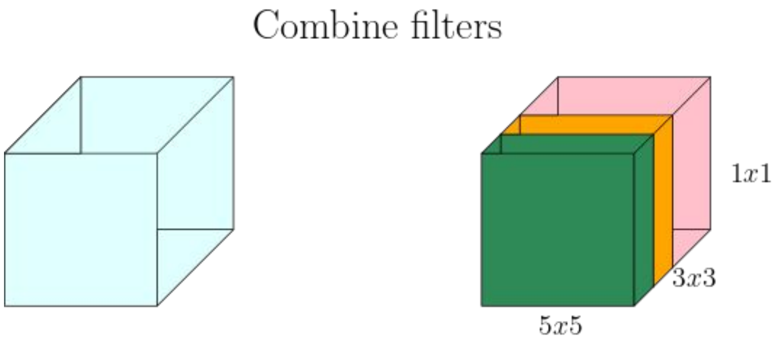
Motivation



When designing our convolutional layers:

- Should we choose filter that is 3x3, 5x5 or 7x7?
- What about pooling? How often should we apply it?

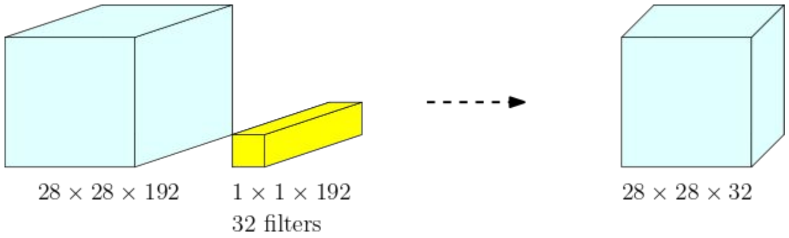
Motivation



Idea: Let’s use them all!

Problem: large filters cause too many flops.

Motivation



Idea: Let's use them all!

Problem: large filters cause too many flops.

Use 1x1 convolutions to reduce features.

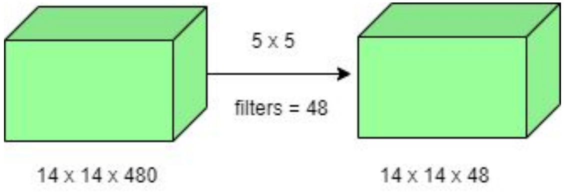
And then apply larger filters!

Features & Inception module

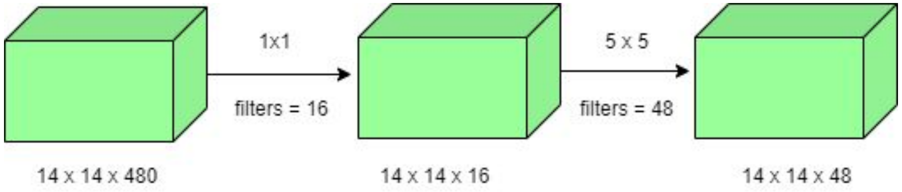
Features of GoogleNet

- **<u>1x1 convolution:</u>** These help in decreasing the number of parameters(w,b). Hence depth can be increased.

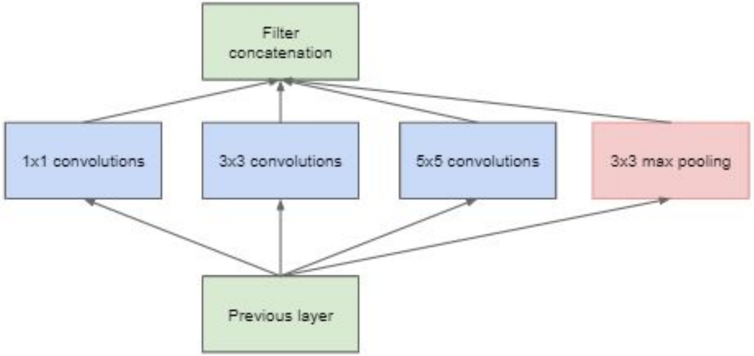
5x5 Conv - (14*14*48) x (5*5*480) = 112.9M



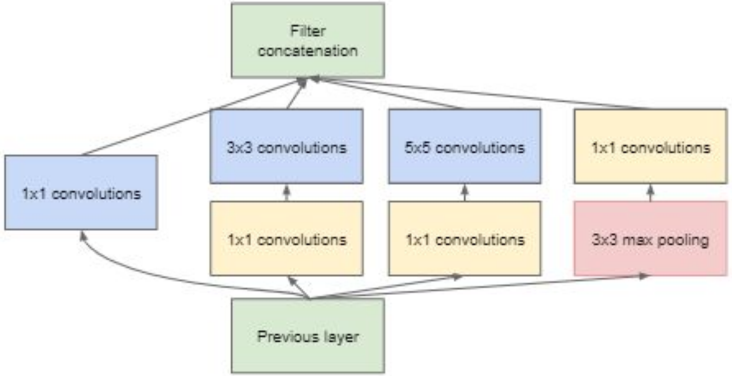
1x1 Conv - (14*14*16) x (1*1*480) + (14*14*48) x (5*5*16) = 5.3M



[Source]



(a) Inception module, naïve version



(b) Inception module with dimension reductions

Features of GoogleNet

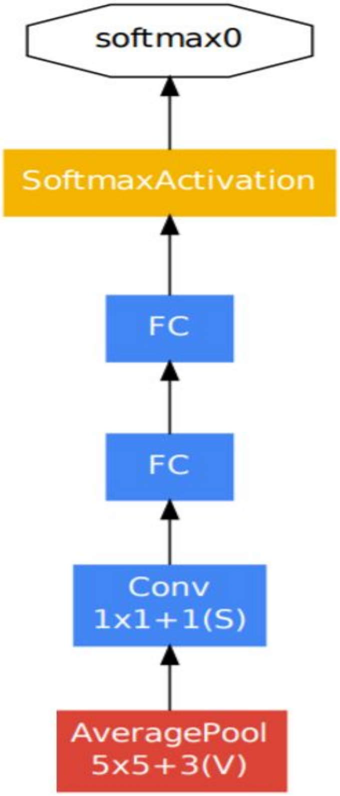
- **Global Average Pooling:** Moving from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%.[2]

  Use of dropout still remains essential even after removing the fully connected layers.

- **Inception Module:**
  - Different from previous previous architectures such as AlexNet.
  - Uses different filters for multiple feature detection.
  - Reduced the cost by dimensionality reduction.
  - 1x1, 3x3, 5x5 and 3x3 max pooling layered.

Features of GoogleNet

- **Auxiliary classifier:**
  - Intermediate classifier branches for training.
  - Consists of an average pool layer, a conv layer, two fully connected layers, a dropout layer(70%), and a final linear layer with softmax activation function.



Inception versions

Versions

- **Inception v1:**
  - Original 2014 version.
  - Produced lowest error for image classification.
- Problem of v1: 5x5 conv causes input dimensions to decrease a lot -> Accuracy decrease.

- **Inception V2:**
  - 5X5 conv replaced by two 3x3 conv nets.
  - Low computational time and high speeds.
  - nxn factorization to 1xn and nx1 factorization.

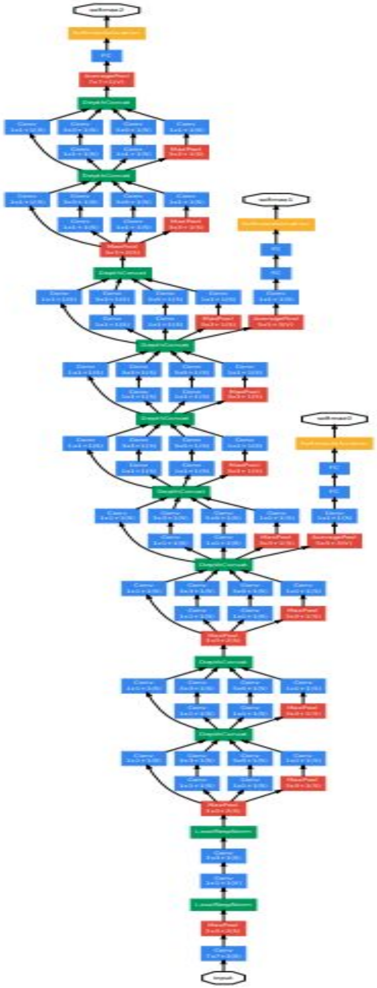
Versions

- **Inception v3:**
  - Everything from v2 plus:
  - RMSprop optimizer
  - use of 7x7 factorized convolutions.
  - Label smoothing regularization(estimates the label-dropout during training)
  - Batch normalization is used in the Auxiliary classifier.
- **Inception v4:**
  - Deeper network
  - number of inception modules increased
  - Uniform filters in inception modules.
  - Changes in the stem part.

Architecture

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								

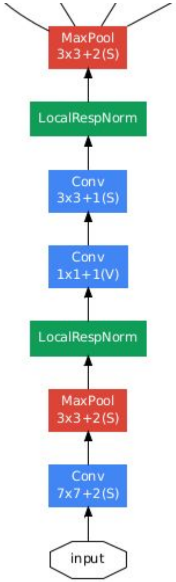
Image is also from the article.



Bit large to fit in a single slide :(

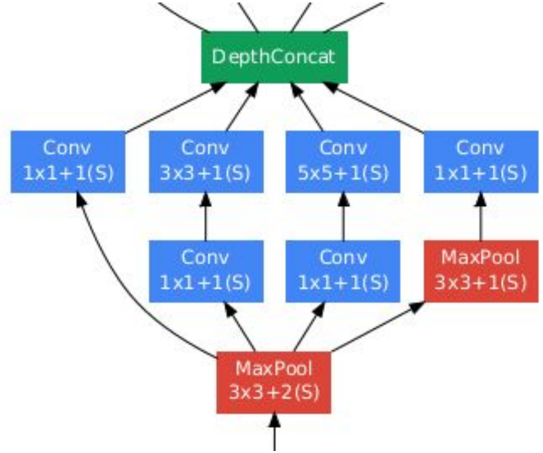
Image is also from the article.

Start

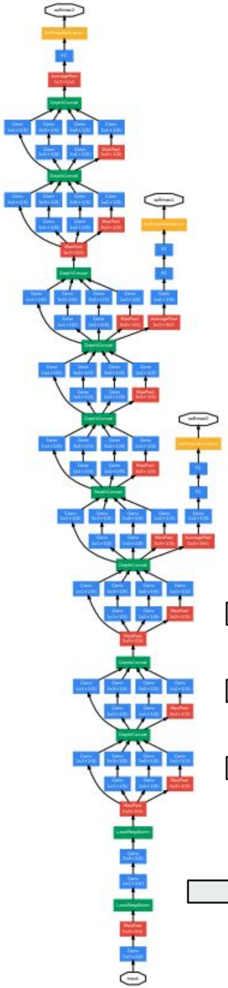


Start with a large filter size with stride = 2 to reduce dimension quickly!

Inception blocks



Use inception blocks to utilize different filters with efficient computations.



?

Inception

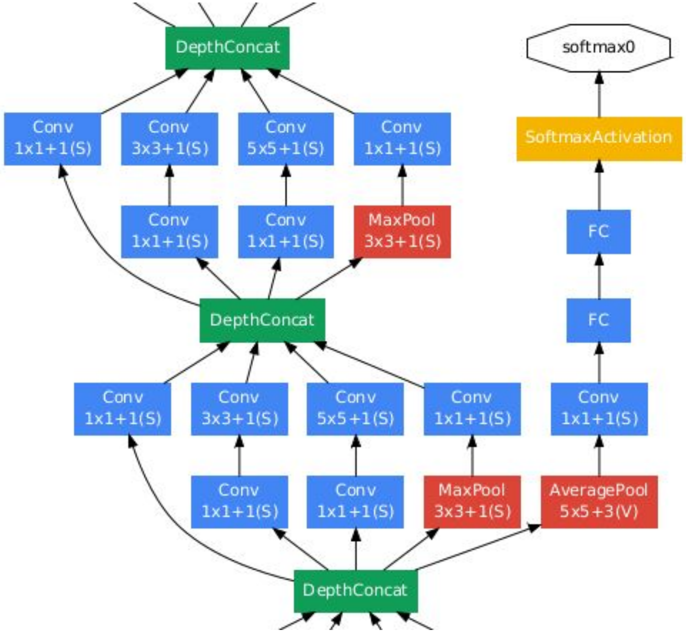
Inception

Inception

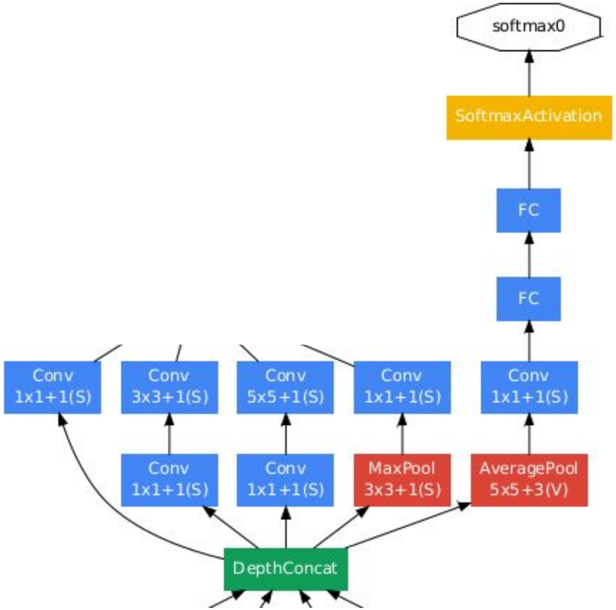
Start

Image is also from the article.

Something different?

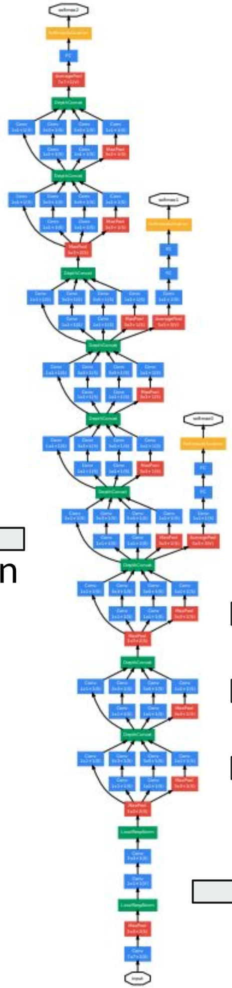


Something different?



Softmax but not at the final layer. It is aimed that even features that are in the first layers should be something meaningful so loss is not only back propagated through the last layer but also from the middle layers.

Has a regularization effect.



Inception + output

Inception

Inception

Inception

Inception

Start

Rest is pretty repetitive.

Image is also from the article.

Implementation in Keras

Implementation

We used Keras for implementation

We used inception and auxiliary modules to build the GoogleNet architecture.

The model was customised to have an input parameter of 256x256 basis our data.

```
...
Total params: 8,222,063
Trainable params: 8,221,551
Non-trainable params: 512
```

Results on UCM land use data

UCM land use data

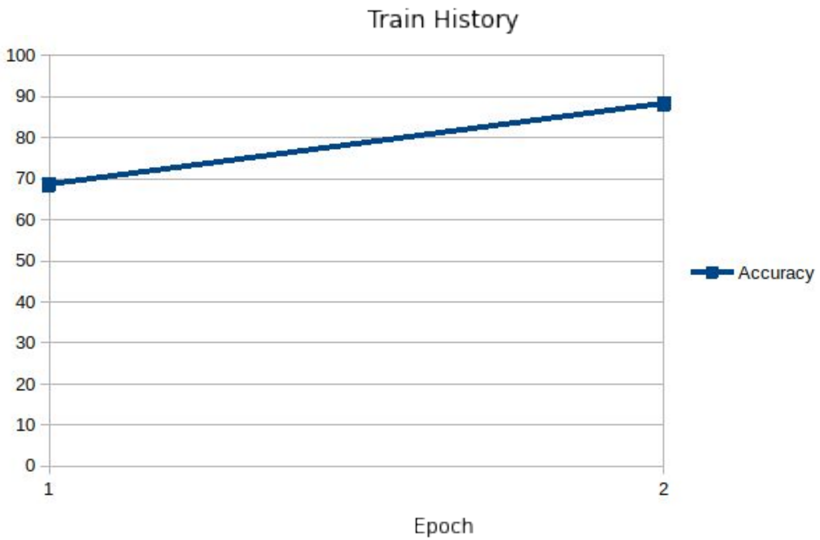
UC merced land use data has images of 256x256 resolution

We have in total 21 classes

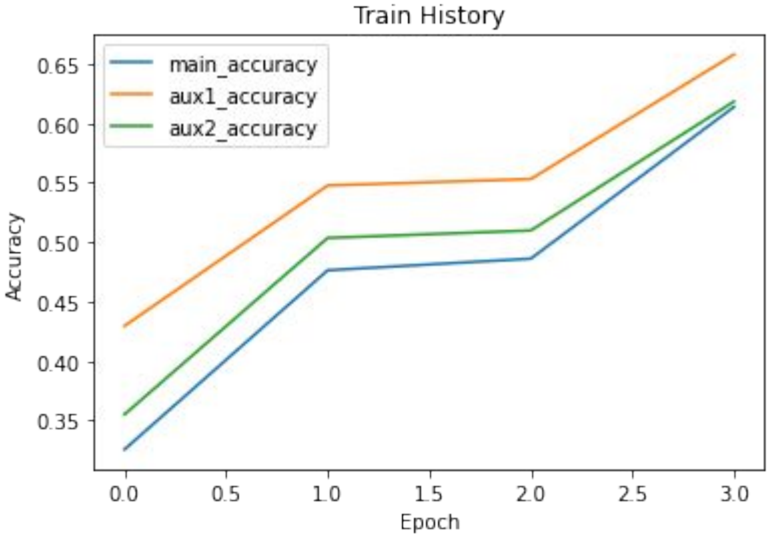
Classes correspond to satellite images of land use coverage like airplane, golf clubs, buildings etc.

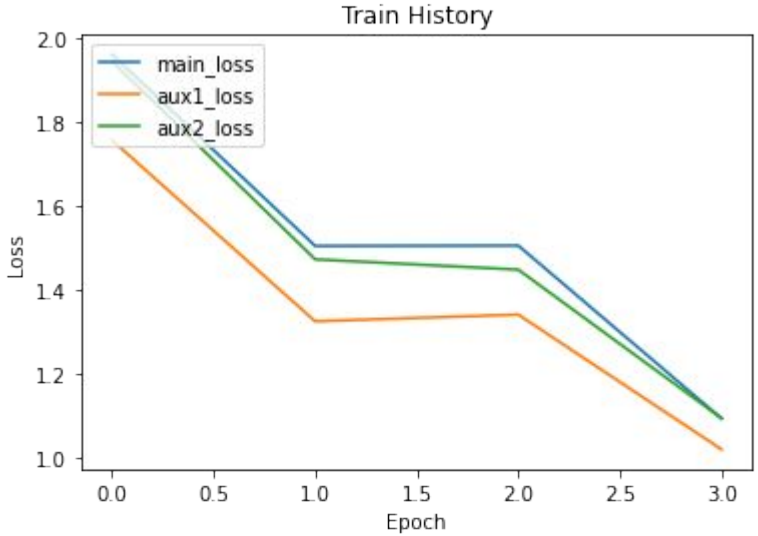
Our results:

Training(pretrained)	Test(pretrained)	Training	Test
88%	70%	65.78%	45.05%


Train History
100
90
80
70
60
50
40
30
20
10
0
1
2
Epoch
Accuracy


Train History
1.2
1
0.8
0.6
0.4
0.2
0
Loss
1
2
Epoch
Loss


Train History
main_accuracy
aux1_accuracy
aux2_accuracy
Accuracy
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Epoch


Train History
main_loss
aux1_loss
aux2_loss
Loss
2.0
1.8
1.6
1.4
1.2
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Epoch

GoogleNet is computationally efficient

“moving to sparser architectures is feasible and useful idea”

- Authors

Conclusion

GoogLeNet proposes a way to utilize different shaped filters with efficient computation. Even though, the architecture is simple to understand, it was strong enough to win a ImageNet competition. Use of multiple layered outputs has a nice regularization effect that prevents overfitting. It is not offering any new blocks to use in computer vision but rather it offers a nice way to combine existing building blocks.

Thanks!

Any questions?

References

1. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).
2. https://www.geeksforgeeks.org/understanding-googlenet-model-cnn-architecture/
3. https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765
4. https://paperswithcode.com/method/googlenet