



Assignment on Googlenet

MTech in Applied AI
Natural Language Processing

by

Sanjeev Kumar Pandey*
Faculty: Dr Saugata Sinha

Submitted April 13, 2025

* Student ID: 31050, Enrolment No:MT23AAI001, Email: pandey.sanjeev@yahoo.com

Table of Contents

1	Problem Statement	2
1.1	Python code - Methods for computation load	2
1.2	Output - Computation Load sample usage output	4
1.3	Output - Googlenet layer by layer calculation output	8
1.4	Archticture: Googlenet layer by layer	11

1 Problem Statement

1. **Write your own function in Python for computing the computation load**, i.e., number of products and summations for each layer, dimensions of the output of a layer, and number of trainable parameters.

The function should accept the following arguments:

- (a) Type of layer
- (b) For a convolution layer: height of filter kernel, width of filter kernel, stride, padding, number of filter kernels.

For a dense layer: number of nodes.

- (c) Dimension of the input

The function should return the following parameters:

- (a) Dimension of the output
- (b) Number of trainable parameters
- (c) Number of dot products
- (d) Total number of products and sums

2. **Using the function**, find out the dimensions of the output, number of trainable parameters, and total number of sums and products for each layer in the **GoogleNet architecture**.

1.1 Python code - Methods for computation load

```

1 class ComputationLoad:
2
3
4     def layer_computation(layer_type, layer_params, input_dim):
5
6
7         """
8         Compute architectural details for a neural network layer.
9
10        Args:
11            layer_type: str - 'conv' for convolutional, 'dense' for fully connected
12            layer_params: dict - parameters for the layer
13                For conv: {'h': height, 'w': width, 'stride': stride,
14                           'padding': padding, 'num_filters': number of filters}
15                For dense: {'nodes': number of nodes}
16            input_dim: tuple - (channels, height, width) for conv,
17                           (nodes,) for dense
18
19        Returns:
20            dict: {
21                'output_dim': output dimensions,
22                'trainable_params': number of trainable parameters,
23                'dot_products': number of dot products,
24                'total_ops': total number of operations (products + sums)
25            }
26        """
27        result = {}
28
29        if layer_type == 'conv':
30            # Unpack parameters
31            C_in = input_dim[0]
32            H_in = input_dim[1]

```

```

33     W_in = input_dim[2]
34     F_h = layer_params['h']
35     F_w = layer_params['w']
36     stride = layer_params['stride']
37     padding = layer_params['padding']
38     num_filters = layer_params['num_filters']
39
40     print(f"->Input Dimension : {input_dim}")
41     print(f"->Filter Parameters : {layer_params}")
42
43     # Calculate output dimensions
44     H_out = ((H_in - F_h + 2 * padding) // stride) + 1
45     W_out = ((W_in - F_w + 2 * padding) // stride) + 1
46     result['output_dim'] = (num_filters, H_out, W_out)
47
48     # Number of trainable parameters
49     # (weights + bias) for each filter
50     result['trainable_params'] = num_filters * (C_in * F_h * F_w + 1)
51
52     # Number of dot products (one per output pixel per filter)
53     result['dot_products'] = num_filters * H_out * W_out
54
55     # Total operations: each dot product is F_h*F_w*C_in multiplications
56     # and (F_h*F_w*C_in - 1) additions
57     ops_per_dot = F_h * F_w * C_in * 2 - 1
58     result['total_ops'] = result['dot_products'] * ops_per_dot
59
60     elif layer_type == 'dense':
61         # Unpack parameters
62         nodes_in = input_dim[0]
63         nodes_out = layer_params['nodes']
64
65         print(f"->Input Dimension : {input_dim}")
66
67         # Output dimension
68         result['output_dim'] = (nodes_out,)
69
70         # Number of trainable parameters (weights + bias)
71         result['trainable_params'] = nodes_in * nodes_out + nodes_out
72
73         # Number of dot products (one per output node)
74         result['dot_products'] = nodes_out
75
76         # Total operations: each dot product is nodes_in multiplications
77         # and (nodes_in - 1) additions
78         ops_per_dot = nodes_in * 2 - 1
79         result['total_ops'] = result['dot_products'] * ops_per_dot
80
81     else:
82         raise ValueError(f"Unknown layer type: {layer_type}")
83
84     return result

```

Listing 1: Code

```

1 # usage_example.py
2
3 from computation_load import ComputationLoad
4
5 def main():
6     # Example 1: Convolutional Layer
7     conv_params = {
8         'h': 3,
9         'w': 3,
10        'stride': 1,

```

```

11         'padding': 1,
12         'num_filters': 64
13     }
14     conv_input_dim = (3, 224, 224)
15     conv_result = ComputationLoad.layer_computation('conv', conv_params, conv_input_dim)
16     print(f"    Convolutional Layer Result")
17     print(f"    Output dimension: {conv_result['output_dim']}")
18     print(f"    Trainable parameters: {conv_result['trainable_params']:,}")
19     print(f"    Total operations: {conv_result['total_ops']:,}")
20
21     print("="*80)
22     # Example 2: Dense (Fully Connected) Layer
23     dense_params = {'nodes': 1000}
24     dense_input_dim = (4096,)
25     dense_result = ComputationLoad.layer_computation('dense', dense_params, dense_input_dim)
26     print(f"    Dense Layer Result:")
27     print(f"    Output dimension: {conv_result['output_dim']}")
28     print(f"    Trainable parameters: {conv_result['trainable_params']:,}")
29     print(f"    Total operations: {conv_result['total_ops']:,}")
30
31 if __name__ == "__main__":
32     main()

```

Listing 2: Sample usage of computation method

1.2 Output - Computation Load sample usage output

```

1 C:\Users\urssa\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Sanjeev\VNIT_CLASSES\VNIT-AAI-SEM3\sub
2 ->Input Dimension : (3, 224, 224)
3 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}
4     Convolutional Layer Result
5     Output dimension: (64, 224, 224)
6     Trainable parameters: 1,792
7     Total operations: 170,196,992
8     =====
9 ->Input Dimension : (4096,)
10    Dense Layer Result:
11    Output dimension: (64, 224, 224)
12    Trainable parameters: 1,792
13    Total operations: 170,196,992
14
15 Process finished with exit code 0

```

Listing 3: Output - Sample usage of computation method

```

1 from computation_load import ComputationLoad
2 def inception_module(input_dim, module_params):
3     """
4     Compute architectural details for an Inception module.
5
6     Args:
7         input_dim: tuple - (channels, height, width)
8         module_params: dict - parameters for each branch
9         {
10             'branches': [
11                 {'type': 'conv', 'params': {...}}, # 1x1 branch
12                 {'type': 'conv', 'params': {...}}, # 3x3 branch
13                 {'type': 'conv', 'params': {...}}, # 5x5 branch
14                 {'type': 'pool', 'params': {...}} # pooling branch
15             ]
16         }
17
18     Returns:
19         dict: same structure as layer_computation
20     """

```

```

21 branch_results = []
22
23 for branch in module_params['branches']:
24     if branch['type'] == 'pool':
25         # For pooling branch, we need to add a 1x1 conv after pooling
26         # First do pooling (no trainable params)
27         pool_params = branch['params']
28         H_in = input_dim[1]
29         W_in = input_dim[2]
30         F_h = pool_params['h']
31         F_w = pool_params['w']
32         stride = pool_params['stride']
33         padding = pool_params['padding']
34
35         H_out = ((H_in - F_h + 2 * padding) // stride) + 1
36         W_out = ((W_in - F_w + 2 * padding) // stride) + 1
37         pool_output_dim = (input_dim[0], H_out, W_out)
38
39         # Then do 1x1 conv
40         conv_params = {
41             'h': 1, 'w': 1, 'stride': 1,
42             'padding': 0, 'num_filters': pool_params['num_filters']
43         }
44         branch_result = ComputationLoad.layer_computation('conv', conv_params, pool_output_dim)
45     else:
46         branch_result = ComputationLoad.layer_computation(branch['type'], branch['params'], input_dim)
47
48     branch_results.append(branch_result)
49
50 # Concatenate all branches along the channel dimension
51 total_filters = sum(br['output_dim'][0] for br in branch_results)
52 output_dim = (total_filters,) + branch_results[0]['output_dim'][1:]
53
54 # Sum all operations and parameters
55 total_params = sum(br['trainable_params'] for br in branch_results)
56 total_ops = sum(br['total_ops'] for br in branch_results)
57
58 return {
59     'output_dim': output_dim,
60     'trainable_params': total_params,
61     'dot_products': sum(br['dot_products'] for br in branch_results),
62     'total_ops': total_ops
63 }
64
65 def analyze_googlenet():
66     current_dim = (3, 224, 224) # Input RGB image
67     total_params = 0
68     total_ops = 0
69
70     def pool_output(dim, k, s, p):
71         print(f"-->Input Dimension : {dim}")
72         print(f"-->Filter Parameters : {{'h': {k}, 'w': {k}, 'stride': {s}, 'padding': {p}}}")
73
74         c, h, w = dim
75         h_out = ((h - k + 2 * p) // s) + 1
76         w_out = ((w - k + 2 * p) // s) + 1
77         return (c, h_out, w_out)
78
79     layers = [
80         {'type': 'conv', 'params': {'h': 7, 'w': 7, 'stride': 2, 'padding': 3, 'num_filters': 64}}, # 1
81         {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}}, # 2 - M
82         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}}, # 3
83         {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}}, # 4
84         {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}}, # 5 - M
85

```

```

86     # Inception modules
87     {'type': 'inception', 'params': { # 6 - 3a
88         'branches': [
89             {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}},
90             {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}},
91             {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}},
92             {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}},
93             {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 32}},
94             {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 32}}
95         ]
96     },
97     {'type': 'inception', 'params': { # 7 - 3b
98         'branches': [
99             {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}},
100            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}},
101            {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}},
102            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
103            {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 96}},
104            {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
105        ]
106    },
107    {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}}, # 8 - MaxPool
108
109    {'type': 'inception', 'params': { # 9 - 4a
110        'branches': [
111            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}},
112            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}},
113            {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 208}},
114            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}},
115            {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 48}},
116            {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
117        ]
118    },
119    {'type': 'inception', 'params': { # 10 - 4b
120        'branches': [
121            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}},
122            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}},
123            {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 224}},
124            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}},
125            {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}},
126            {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
127        ]
128    },
129    {'type': 'inception', 'params': { # 11 - 4c
130        'branches': [
131            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}},
132            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}},
133            {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 256}},
134            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}},
135            {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}},
136            {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
137        ]
138    },
139    {'type': 'inception', 'params': { # 12 - 4d
140        'branches': [
141            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}},
142            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 144}},
143            {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 288}},
144            {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
145            {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}},
146            {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
147        ]
148    },
149    {'type': 'inception', 'params': { # 13 - 4e
150        'branches': [

```

```

151         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}}
152         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}}
153         {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}}
154         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
155         {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}},
156         {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}}
157     ]
158 },
159 {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}}, # 14 - MaxPool
160
161 {'type': 'inception', 'params': { # 15 - 5a
162     'branches': [
163         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}}
164         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}}
165         {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}}
166         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
167         {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}},
168         {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}}
169     ]
170 },
171 {'type': 'inception', 'params': { # 16 - 5b
172     'branches': [
173         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 384}}
174         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}}
175         {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 384}}
176         {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 48}},
177         {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}},
178         {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}}
179     ]
180 },
181 {'type': 'pool', 'params': {'h': 7, 'w': 7, 'stride': 1, 'padding': 0}}, # 17 - AvgPool
182 {'type': 'dense', 'params': {'nodes': 1000}} # 18 - Fully connected classifier
183 ]
184
185 print("GoogleNet Architecture Analysis (All 22 Layers):")
186 print("=" * 70)
187 print(f"Input dimension: {current_dim}")
188 print("-" * 70)
189
190 for i, layer in enumerate(layers):
191     if layer['type'] == 'inception':
192         print(f"Layer {i + 1} (Inception module):")
193         result = inception_module(current_dim, layer['params'])
194
195     elif layer['type'] == 'pool':
196         print(f"Layer {i + 1} (Pooling layer):")
197         result = {'output_dim': pool_output(current_dim, layer['params']['h'], layer['params']['stride']),
198                 'trainable_params': 0, 'dot_products': 0, 'total_ops': 0}
199
200     else:
201         print(f"Layer {i + 1} ({layer['type']}):")
202         result = ComputationLoad.layer_computation(layer['type'], layer['params'], current_dim)
203
204     print(f"  Output dimension: {result['output_dim']}")
205     print(f"  Trainable parameters: {result['trainable_params']:,}")
206     print(f"  Total operations: {result['total_ops']:,}")
207     print("-" * 50)
208
209     total_params += result['trainable_params']
210     total_ops += result['total_ops']
211     current_dim = result['output_dim']
212
213 print("=" * 70)
214 print(f"Total trainable parameters: {total_params:,}")
215 print(f"Total operations: {total_ops:,}")

```



```

216
217 # Run the analysis
218 analyze_googlenet()

```

Listing 4: Code - Googlenet layer by layer computation

1.3 Output - Googlenet layer by layer calculation output

```

1 C:\Users\urssa\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Sanjeev\VNIT_CLASSES\VNIT-AAI-SEM3\sub
2 GoogleNet Architecture Analysis (All 22 Layers):
3 =====
4 Input dimension: (3, 224, 224)
5 -----
6 Layer 1 (conv):
7 ->Input Dimension : (3, 224, 224)
8 ->Filter Paramaters : {'h': 7, 'w': 7, 'stride': 2, 'padding': 3, 'num_filters': 64}
9   Output dimension: (64, 112, 112)
10   Trainable parameters: 9,472
11   Total operations: 235,225,088
12 -----
13 Layer 2 (Pooling layer):
14 ->Input Dimension : (64, 112, 112)
15 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
16   Output dimension: (64, 56, 56)
17   Trainable parameters: 0
18   Total operations: 0
19 -----
20 Layer 3 (conv):
21 ->Input Dimension : (64, 56, 56)
22 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
23   Output dimension: (64, 56, 56)
24   Trainable parameters: 4,160
25   Total operations: 25,489,408
26 -----
27 Layer 4 (conv):
28 ->Input Dimension : (64, 56, 56)
29 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}
30   Output dimension: (192, 56, 56)
31   Trainable parameters: 110,784
32   Total operations: 693,030,912
33 -----
34 Layer 5 (Pooling layer):
35 ->Input Dimension : (192, 56, 56)
36 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
37   Output dimension: (192, 28, 28)
38   Trainable parameters: 0
39   Total operations: 0
40 -----
41 Layer 6 (Inception module):
42 ->Input Dimension : (192, 28, 28)
43 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
44 ->Input Dimension : (192, 28, 28)
45 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}
46 ->Input Dimension : (192, 28, 28)
47 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}
48 ->Input Dimension : (192, 28, 28)
49 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}
50 ->Input Dimension : (192, 28, 28)
51 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 32}
52 ->Input Dimension : (192, 28, 28)
53 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
54   Output dimension: (368, 28, 28)
55   Trainable parameters: 415,088
56   Total operations: 649,992,448
57 -----

```

```
58 Layer 7 (Inception module):
59 ->Input Dimension : (368, 28, 28)
60 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
61 ->Input Dimension : (368, 28, 28)
62 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
63 ->Input Dimension : (368, 28, 28)
64 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}
65 ->Input Dimension : (368, 28, 28)
66 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
67 ->Input Dimension : (368, 28, 28)
68 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 96}
69 ->Input Dimension : (368, 28, 28)
70 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
71   Output dimension: (640, 28, 28)
72   Trainable parameters: 1,649,280
73   Total operations: 2,584,565,760
74 -----
75 Layer 8 (Pooling layer):
76 ->Input Dimension : (640, 28, 28)
77 ->Filter Parameters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
78   Output dimension: (640, 14, 14)
79   Trainable parameters: 0
80   Total operations: 0
81 -----
82 Layer 9 (Inception module):
83 ->Input Dimension : (640, 14, 14)
84 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}
85 ->Input Dimension : (640, 14, 14)
86 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}
87 ->Input Dimension : (640, 14, 14)
88 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 208}
89 ->Input Dimension : (640, 14, 14)
90 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}
91 ->Input Dimension : (640, 14, 14)
92 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 48}
93 ->Input Dimension : (640, 14, 14)
94 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
95   Output dimension: (624, 14, 14)
96   Trainable parameters: 2,202,224
97   Total operations: 862,904,896
98 -----
99 Layer 10 (Inception module):
100 ->Input Dimension : (624, 14, 14)
101 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}
102 ->Input Dimension : (624, 14, 14)
103 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}
104 ->Input Dimension : (624, 14, 14)
105 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 224}
106 ->Input Dimension : (624, 14, 14)
107 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}
108 ->Input Dimension : (624, 14, 14)
109 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}
110 ->Input Dimension : (624, 14, 14)
111 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
112   Output dimension: (648, 14, 14)
113   Trainable parameters: 2,481,672
114   Total operations: 972,434,400
115 -----
116 Layer 11 (Inception module):
117 ->Input Dimension : (648, 14, 14)
118 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
119 ->Input Dimension : (648, 14, 14)
120 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
121 ->Input Dimension : (648, 14, 14)
122 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 256}
```

```
123 ->Input Dimension : (648, 14, 14)
124 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}
125 ->Input Dimension : (648, 14, 14)
126 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}
127 ->Input Dimension : (648, 14, 14)
128 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
129   Output dimension: (664, 14, 14)
130   Trainable parameters: 2,753,368
131   Total operations: 1,078,929,824
132 -----
133 Layer 12 (Inception module):
134 ->Input Dimension : (664, 14, 14)
135 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}
136 ->Input Dimension : (664, 14, 14)
137 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 144}
138 ->Input Dimension : (664, 14, 14)
139 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 288}
140 ->Input Dimension : (664, 14, 14)
141 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
142 ->Input Dimension : (664, 14, 14)
143 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}
144 ->Input Dimension : (664, 14, 14)
145 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
146   Output dimension: (704, 14, 14)
147   Trainable parameters: 3,017,920
148   Total operations: 1,182,610,688
149 -----
150 Layer 13 (Inception module):
151 ->Input Dimension : (704, 14, 14)
152 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}
153 ->Input Dimension : (704, 14, 14)
154 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}
155 ->Input Dimension : (704, 14, 14)
156 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}
157 ->Input Dimension : (704, 14, 14)
158 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
159 ->Input Dimension : (704, 14, 14)
160 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}
161 ->Input Dimension : (704, 14, 14)
162 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
163   Output dimension: (1024, 14, 14)
164   Trainable parameters: 4,686,848
165   Total operations: 1,836,642,304
166 -----
167 Layer 14 (Pooling layer):
168 ->Input Dimension : (1024, 14, 14)
169 ->Filter Parameters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
170   Output dimension: (1024, 7, 7)
171   Trainable parameters: 0
172   Total operations: 0
173 -----
174 Layer 15 (Inception module):
175 ->Input Dimension : (1024, 7, 7)
176 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}
177 ->Input Dimension : (1024, 7, 7)
178 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}
179 ->Input Dimension : (1024, 7, 7)
180 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}
181 ->Input Dimension : (1024, 7, 7)
182 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
183 ->Input Dimension : (1024, 7, 7)
184 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}
185 ->Input Dimension : (1024, 7, 7)
186 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
187   Output dimension: (1024, 7, 7)
```

```

188   Trainable parameters: 6,816,768
189   Total operations: 667,892,736
190 -----
191 Layer 16 (Inception module):
192 ->Input Dimension : (1024, 7, 7)
193 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 384}
194 ->Input Dimension : (1024, 7, 7)
195 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}
196 ->Input Dimension : (1024, 7, 7)
197 ->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 384}
198 ->Input Dimension : (1024, 7, 7)
199 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 48}
200 ->Input Dimension : (1024, 7, 7)
201 ->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}
202 ->Input Dimension : (1024, 7, 7)
203 ->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
204   Output dimension: (1264, 7, 7)
205   Trainable parameters: 7,587,056
206   Total operations: 743,345,680
207 -----
208 Layer 17 (Pooling layer):
209 ->Input Dimension : (1264, 7, 7)
210 ->Filter Parameters : {'h': 7, 'w': 7, 'stride': 1, 'padding': 0}
211   Output dimension: (1264, 1, 1)
212   Trainable parameters: 0
213   Total operations: 0
214 -----
215 Layer 18 (dense):
216 ->Input Dimension : (1264, 1, 1)
217   Output dimension: (1000,)
218   Trainable parameters: 1,265,000
219   Total operations: 2,527,000
220 -----
221 =====
222 Total trainable parameters: 32,999,640
223 Total operations: 11,535,591,144
224
225 Process finished with exit code 0

```

Listing 5: Output - Googlenet layer by layer computation

1.4 Archticture: Googlenet layer by layer

Layer No.	Type	Description
1	Conv	7×7 convolution, stride 2, 64 filters
2	Pool	MaxPool 3×3, stride 2
3	Conv	1×1 convolution, 64 filters
4	Conv	3×3 convolution, 192 filters
5	Pool	MaxPool 3×3, stride 2
6	Inception (3a)	Branches: <ul style="list-style-type: none"> – 1×1 Conv (64 filters) – 1×1 → 3×3 Conv (96 → 128) – 1×1 → 5×5 Conv (16 → 32) – 3×3 Pool → 1×1 Conv (32)
7	Inception (3b)	Similar structure to 3a: <ul style="list-style-type: none"> – 128, 192, 96 filters, etc.
8	Pool	MaxPool 3×3, stride 2
9	Inception (4a)	Input: 640 channels <ul style="list-style-type: none"> Includes: 192→208 (3×3), 96→208 (3×3), etc.

10	Inception (4b)	Moderate complexity Filters: 160, 112, 224, etc.
11	Inception (4c)	Filters: 128, 128, 256, etc.
12	Inception (4d)	Filters: 112, 144, 288, etc.
13	Inception (4e)	Largest block so far Output: 1024 channels
14	Pool	MaxPool 3×3, stride 2
15	Inception (5a)	Filters: 256, 160, 320, 128 (5×5), 128 (pool)
16	Inception (5b)	Filters: 384, 192, 384, 128 (5×5), 128 (pool)
17	Pool	AvgPool 7×7, stride 1
18	Dense	Fully connected, 1000 output nodes

1.4.1 Source Reference

Title: *Going Deeper with Convolutions*

Authors: Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

Conference: CVPR 2015 (IEEE Conference on Computer Vision and Pattern Recognition)

arXiv link: <https://arxiv.org/abs/1409.4842>