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

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

```

class ComputationLoad:

    def layer_computation(layer_type, layer_params, input_dim):


        """
        Compute architectural details for a neural network layer.

        Args:
            layer_type: str - 'conv' for convolutional, 'dense' for fully connected
            layer_params: dict - parameters for the layer
                For conv: {'h': height, 'w': width, 'stride': stride,
                          'padding': padding, 'num_filters': number of filters}
                For dense: {'nodes': number of nodes}
            input_dim: tuple - (channels, height, width) for conv,
                              (nodes,) for dense

        Returns:
            dict: {
                'output_dim': output dimensions,
                'trainable_params': number of trainable parameters,
                'dot_products': number of dot products,
                'total_ops': total number of operations (products + sums)
            }
        """
        result = {}

        if layer_type == 'conv':
            # Unpack parameters
            C_in = input_dim[0]
            H_in = input_dim[1]
```

```
            W_in = input_dim[2]
            F_h = layer_params['h']
            F_w = layer_params['w']
            stride = layer_params['stride']
            padding = layer_params['padding']
            num_filters = layer_params['num_filters']

            print(f"->Input Dimension : {input_dim}")
            print(f"->Filter Paramaters : {layer_params}")

            # Calculate output dimensions
            H_out = ((H_in - F_h + 2 * padding) // stride) + 1
            W_out = ((W_in - F_w + 2 * padding) // stride) + 1
            result['output_dim'] = (num_filters, H_out, W_out)

            # Number of trainable parameters
            # (weights + bias) for each filter
            result['trainable_params'] = num_filters * (C_in * F_h * F_w + 1)

            # Number of dot products (one per output pixel per filter)
            result['dot_products'] = num_filters * H_out * W_out

            # Total operations: each dot product is F_h*F_w*C_in multiplications
            # and (F_h*F_w*C_in - 1) additions
            ops_per_dot = F_h * F_w * C_in * 2 - 1
            result['total_ops'] = result['dot_products'] * ops_per_dot

        elif layer_type == 'dense':
            # Unpack parameters
            nodes_in = input_dim[0]
            nodes_out = layer_params['nodes']

            print(f"->Input Dimension : {input_dim}")

            # Output dimension
            result['output_dim'] = (nodes_out,)

            # Number of trainable parameters (weights + bias)
            result['trainable_params'] = nodes_in * nodes_out + nodes_out

            # Number of dot products (one per output node)
            result['dot_products'] = nodes_out

            # Total operations: each dot product is nodes_in multiplications
            # and (nodes_in - 1) additions
            ops_per_dot = nodes_in * 2 - 1
            result['total_ops'] = result['dot_products'] * ops_per_dot

        else:
            raise ValueError(f"Unknown layer type: {layer_type}")

        return result
```

Listing 1: Code

```
# usage_example.py

from computation_load import ComputationLoad

def main():
    # Example 1: Convolutional Layer
    conv_params = {
        'h': 3,
        'w': 3,
        'stride': 1,
```

```
        'padding': 1,
        'num_filters': 64
    }
    conv_input_dim = (3, 224, 224)
    conv_result = ComputationLoad.layer_computation('conv', conv_params, conv_input_dim)
    print(f"  Convolutional Layer Result")
    print(f"  Output dimension: {conv_result['output_dim']}")
    print(f"  Trainable parameters: {conv_result['trainable_params']:,}")
    print(f"  Total operations: {conv_result['total_ops']:,}")

    print("="*80)
    # Example 2: Dense (Fully Connected) Layer
    dense_params = {'nodes': 1000}
    dense_input_dim = (4096,)
    dense_result = ComputationLoad.layer_computation('dense', dense_params, dense_input_dim)
    print(f"  Dense Layer Result:")
    print(f"  Output dimension: {conv_result['output_dim']}")
    print(f"  Trainable parameters: {conv_result['trainable_params']:,}")
    print(f"  Total operations: {conv_result['total_ops']:,}")

if __name__ == "__main__":
    main()
```

Listing 2: Sample usage of computation method

1.2 Output - Computation Load sample usage output

```
C:\Users\urssa\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Sanjeev\VNIT_CLASSES\VNIT-AAI-SEM3\sub
->Input Dimension : (3, 224, 224)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}
  Convolutional Layer Result
  Output dimension: (64, 224, 224)
  Trainable parameters: 1,792
  Total operations: 170,196,992
================================================================================
->Input Dimension : (4096,)
  Dense Layer Result:
  Output dimension: (64, 224, 224)
  Trainable parameters: 1,792
  Total operations: 170,196,992

Process finished with exit code 0
```

Listing 3: Output - Sample usage of computation method

```
from computation_load import ComputationLoad
def inception_module(input_dim, module_params):
    """
    Compute architectural details for an Inception module.

    Args:
        input_dim: tuple - (channels, height, width)
        module_params: dict - parameters for each branch
            {
                'branches': [
                    {'type': 'conv', 'params': {...}},  # 1x1 branch
                    {'type': 'conv', 'params': {...}},  # 3x3 branch
                    {'type': 'conv', 'params': {...}},  # 5x5 branch
                    {'type': 'pool', 'params': {...}}   # pooling branch
                ]
            }

    Returns:
        dict: same structure as layer_computation
    """
```

```
    branch_results = []

    for branch in module_params['branches']:
        if branch['type'] == 'pool':
            # For pooling branch, we need to add a 1x1 conv after pooling
            # First do pooling (no trainable params)
            pool_params = branch['params']
            H_in = input_dim[1]
            W_in = input_dim[2]
            F_h = pool_params['h']
            F_w = pool_params['w']
            stride = pool_params['stride']
            padding = pool_params['padding']

            H_out = ((H_in - F_h + 2 * padding) // stride) + 1
            W_out = ((W_in - F_w + 2 * padding) // stride) + 1
            pool_output_dim = (input_dim[0], H_out, W_out)

            # Then do 1x1 conv
            conv_params = {
                'h': 1, 'w': 1, 'stride': 1,
                'padding': 0, 'num_filters': pool_params['num_filters']
            }
            branch_result = ComputationLoad.layer_computation('conv', conv_params, pool_output_dim)
        else:
            branch_result = ComputationLoad.layer_computation(branch['type'], branch['params'], input_dim)

        branch_results.append(branch_result)

    # Concatenate all branches along the channel dimension
    total_filters = sum(br['output_dim'][0] for br in branch_results)
    output_dim = (total_filters,) + branch_results[0]['output_dim'][1:]

    # Sum all operations and parameters
    total_params = sum(br['trainable_params'] for br in branch_results)
    total_ops = sum(br['total_ops'] for br in branch_results)

    return {
        'output_dim': output_dim,
        'trainable_params': total_params,
        'dot_products': sum(br['dot_products'] for br in branch_results),
        'total_ops': total_ops
    }

def analyze_googlenet():
    current_dim = (3, 224, 224)  # Input RGB image
    total_params = 0
    total_ops = 0

    def pool_output(dim, k, s, p):
        print(f"->Input Dimension   : {dim}")
        print(f"->Filter Parameters : {{'h': {k}, 'w': {k}, 'stride': {s}, 'padding': {p}}}")

        c, h, w = dim
        h_out = ((h - k + 2 * p) // s) + 1
        w_out = ((w - k + 2 * p) // s) + 1
        return (c, h_out, w_out)

    layers = [
        {'type': 'conv', 'params': {'h': 7, 'w': 7, 'stride': 2, 'padding': 3, 'num_filters': 64}},  # 1
        {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}},                    # 2 - M
        {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}}, # 3
        {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}},# 4
        {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}},                    # 5 - M

```

```
        # Inception modules
        {'type': 'inception', 'params': {  # 6 - 3a
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}},
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}},
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 32}},
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 32}}
            ]
        }},
        {'type': 'inception', 'params': {  # 7 - 3b
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 96}},
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
            ]
        }},
        {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}},  # 8 - MaxPool

        {'type': 'inception', 'params': {  # 9 - 4a
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}},
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 208}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 48}},
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
            ]
        }},
        {'type': 'inception', 'params': {  # 10 - 4b
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 224}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}},
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
            ]
        }},
        {'type': 'inception', 'params': {  # 11 - 4c
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 256}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}},
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
            ]
        }},
        {'type': 'inception', 'params': {  # 12 - 4d
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 144}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 288}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}},
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 64}}
            ]
        }},
        {'type': 'inception', 'params': {  # 13 - 4e
            'branches': [
```

```
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}}
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}
            ]
        }},
        {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}},  # 14 - MaxPool

        {'type': 'inception', 'params': {  # 15 - 5a
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}}
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}
            ]
        }},
        {'type': 'inception', 'params': {  # 16 - 5b
            'branches': [
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 384}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}}
                {'type': 'conv', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 384}}
                {'type': 'conv', 'params': {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 48}},
                {'type': 'conv', 'params': {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}}
                {'type': 'pool', 'params': {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}
            ]
        }},
        {'type': 'pool', 'params': {'h': 7, 'w': 7, 'stride': 1, 'padding': 0}},  # 17 - AvgPool
        {'type': 'dense', 'params': {'nodes': 1000}}  # 18 - Fully connected classifier
    ]

    print("GoogleNet Architecture Analysis (All 22 Layers):")
    print("=" * 70)
    print(f"Input dimension: {current_dim}")
    print("-" * 70)

    for i, layer in enumerate(layers):
        if layer['type'] == 'inception':
            print(f"Layer {i + 1} (Inception module):")
            result = inception_module(current_dim, layer['params'])

        elif layer['type'] == 'pool':
            print(f"Layer {i + 1} (Pooling layer):")
            result = {'output_dim': pool_output(current_dim, layer['params']['h'], layer['params']['stride'
                      'trainable_params': 0, 'dot_products': 0, 'total_ops': 0}
        else:
            print(f"Layer {i + 1} ({layer['type']}):")
            result = ComputationLoad.layer_computation(layer['type'], layer['params'], current_dim)

        print(f"  Output dimension: {result['output_dim']}")
        print(f"  Trainable parameters: {result['trainable_params']:,}")
        print(f"  Total operations: {result['total_ops']:,}")
        print("-" * 50)

        total_params += result['trainable_params']
        total_ops += result['total_ops']
        current_dim = result['output_dim']

    print("=" * 70)
    print(f"Total trainable parameters: {total_params:,}")
    print(f"Total operations: {total_ops:,}")
```

```

# Run the analysis
analyze_googlenet()
```

Listing 4: Code - Googlenet layer by layer computation

1.3 Output - Googlenet layer by layer calculation output

```
C:\Users\urssa\AppData\Local\Microsoft\WindowsApps\python3.11.exe C:\Sanjeev\VNIT_CLASSES\VNIT-AAI-SEM3\sub
GoogleNet Architecture Analysis (All 22 Layers):
======================================================================
Input dimension: (3, 224, 224)
----------------------------------------------------------------------
Layer 1 (conv):
->Input Dimension : (3, 224, 224)
->Filter Paramaters : {'h': 7, 'w': 7, 'stride': 2, 'padding': 3, 'num_filters': 64}
  Output dimension: (64, 112, 112)
  Trainable parameters: 9,472
  Total operations: 235,225,088
--------------------------------------------------
Layer 2 (Pooling layer):
->Input Dimension   : (64, 112, 112)
->Filter Parameters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
  Output dimension: (64, 56, 56)
  Trainable parameters: 0
  Total operations: 0
--------------------------------------------------
Layer 3 (conv):
->Input Dimension : (64, 56, 56)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
  Output dimension: (64, 56, 56)
  Trainable parameters: 4,160
  Total operations: 25,489,408
--------------------------------------------------
Layer 4 (conv):
->Input Dimension : (64, 56, 56)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}
  Output dimension: (192, 56, 56)
  Trainable parameters: 110,784
  Total operations: 693,030,912
--------------------------------------------------
Layer 5 (Pooling layer):
->Input Dimension   : (192, 56, 56)
->Filter Parameters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
  Output dimension: (192, 28, 28)
  Trainable parameters: 0
  Total operations: 0
--------------------------------------------------
Layer 6 (Inception module):
->Input Dimension : (192, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
->Input Dimension : (192, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}
->Input Dimension : (192, 28, 28)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 128}
->Input Dimension : (192, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}
->Input Dimension : (192, 28, 28)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 32}
->Input Dimension : (192, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
  Output dimension: (368, 28, 28)
  Trainable parameters: 415,088
  Total operations: 649,992,448
--------------------------------------------------
```

```
Layer 7 (Inception module):
->Input Dimension : (368, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
->Input Dimension : (368, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
->Input Dimension : (368, 28, 28)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 192}
->Input Dimension : (368, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
->Input Dimension : (368, 28, 28)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 96}
->Input Dimension : (368, 28, 28)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
  Output dimension: (640, 28, 28)
  Trainable parameters: 1,649,280
  Total operations: 2,584,565,760
------------------------------------------------
Layer 8 (Pooling layer):
->Input Dimension   : (640, 28, 28)
->Filter Parameters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
  Output dimension: (640, 14, 14)
  Trainable parameters: 0
  Total operations: 0
------------------------------------------------
Layer 9 (Inception module):
->Input Dimension : (640, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}
->Input Dimension : (640, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 96}
->Input Dimension : (640, 14, 14)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 208}
->Input Dimension : (640, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 16}
->Input Dimension : (640, 14, 14)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 48}
->Input Dimension : (640, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
  Output dimension: (624, 14, 14)
  Trainable parameters: 2,202,224
  Total operations: 862,904,896
------------------------------------------------
Layer 10 (Inception module):
->Input Dimension : (624, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}
->Input Dimension : (624, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}
->Input Dimension : (624, 14, 14)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 224}
->Input Dimension : (624, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}
->Input Dimension : (624, 14, 14)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}
->Input Dimension : (624, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
  Output dimension: (648, 14, 14)
  Trainable parameters: 2,481,672
  Total operations: 972,434,400
------------------------------------------------
Layer 11 (Inception module):
->Input Dimension : (648, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
->Input Dimension : (648, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
->Input Dimension : (648, 14, 14)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 256}
```

```
->Input Dimension : (648, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 24}
->Input Dimension : (648, 14, 14)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}
->Input Dimension : (648, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
  Output dimension: (664, 14, 14)
  Trainable parameters: 2,753,368
  Total operations: 1,078,929,824
--------------------------------------------------
Layer 12 (Inception module):
->Input Dimension : (664, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 112}
->Input Dimension : (664, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 144}
->Input Dimension : (664, 14, 14)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 288}
->Input Dimension : (664, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
->Input Dimension : (664, 14, 14)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 64}
->Input Dimension : (664, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 64}
  Output dimension: (704, 14, 14)
  Trainable parameters: 3,017,920
  Total operations: 1,182,610,688
--------------------------------------------------
Layer 13 (Inception module):
->Input Dimension : (704, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}
->Input Dimension : (704, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}
->Input Dimension : (704, 14, 14)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}
->Input Dimension : (704, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
->Input Dimension : (704, 14, 14)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}
->Input Dimension : (704, 14, 14)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
  Output dimension: (1024, 14, 14)
  Trainable parameters: 4,686,848
  Total operations: 1,836,642,304
--------------------------------------------------
Layer 14 (Pooling layer):
->Input Dimension   : (1024, 14, 14)
->Filter Parameters : {'h': 3, 'w': 3, 'stride': 2, 'padding': 1}
  Output dimension: (1024, 7, 7)
  Trainable parameters: 0
  Total operations: 0
--------------------------------------------------
Layer 15 (Inception module):
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 256}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 160}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 320}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 32}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
  Output dimension: (1024, 7, 7)
```

```
  Trainable parameters: 6,816,768
  Total operations: 667,892,736
--------------------------------------------------
Layer 16 (Inception module):
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 384}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 192}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 3, 'w': 3, 'stride': 1, 'padding': 1, 'num_filters': 384}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 48}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 5, 'w': 5, 'stride': 1, 'padding': 2, 'num_filters': 128}
->Input Dimension : (1024, 7, 7)
->Filter Paramaters : {'h': 1, 'w': 1, 'stride': 1, 'padding': 0, 'num_filters': 128}
  Output dimension: (1264, 7, 7)
  Trainable parameters: 7,587,056
  Total operations: 743,345,680
--------------------------------------------------
Layer 17 (Pooling layer):
->Input Dimension   : (1264, 7, 7)
->Filter Parameters : {'h': 7, 'w': 7, 'stride': 1, 'padding': 0}
  Output dimension: (1264, 1, 1)
  Trainable parameters: 0
  Total operations: 0
--------------------------------------------------
Layer 18 (dense):
->Input Dimension : (1264, 1, 1)
  Output dimension: (1000,)
  Trainable parameters: 1,265,000
  Total operations: 2,527,000
--------------------------------------------------
==================================================================
Total trainable parameters: 32,999,640
Total operations: 11,535,591,144

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:
		– 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:
		– 128, 192, 96 filters, etc.
8	Pool	MaxPool 3×3, stride 2
9	Inception (4a)	Input: 640 channels
		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