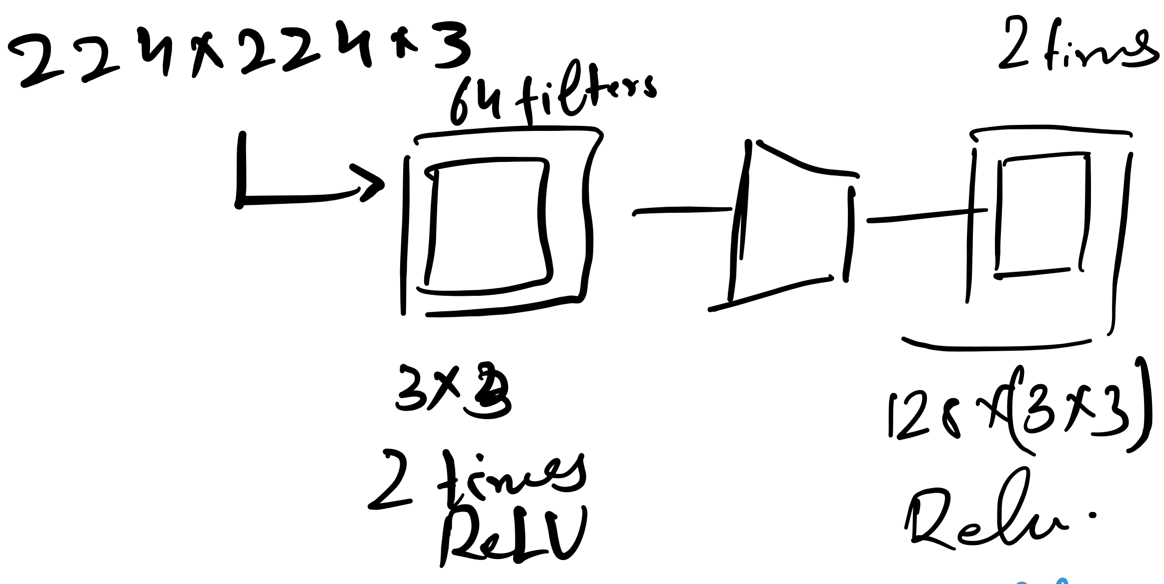
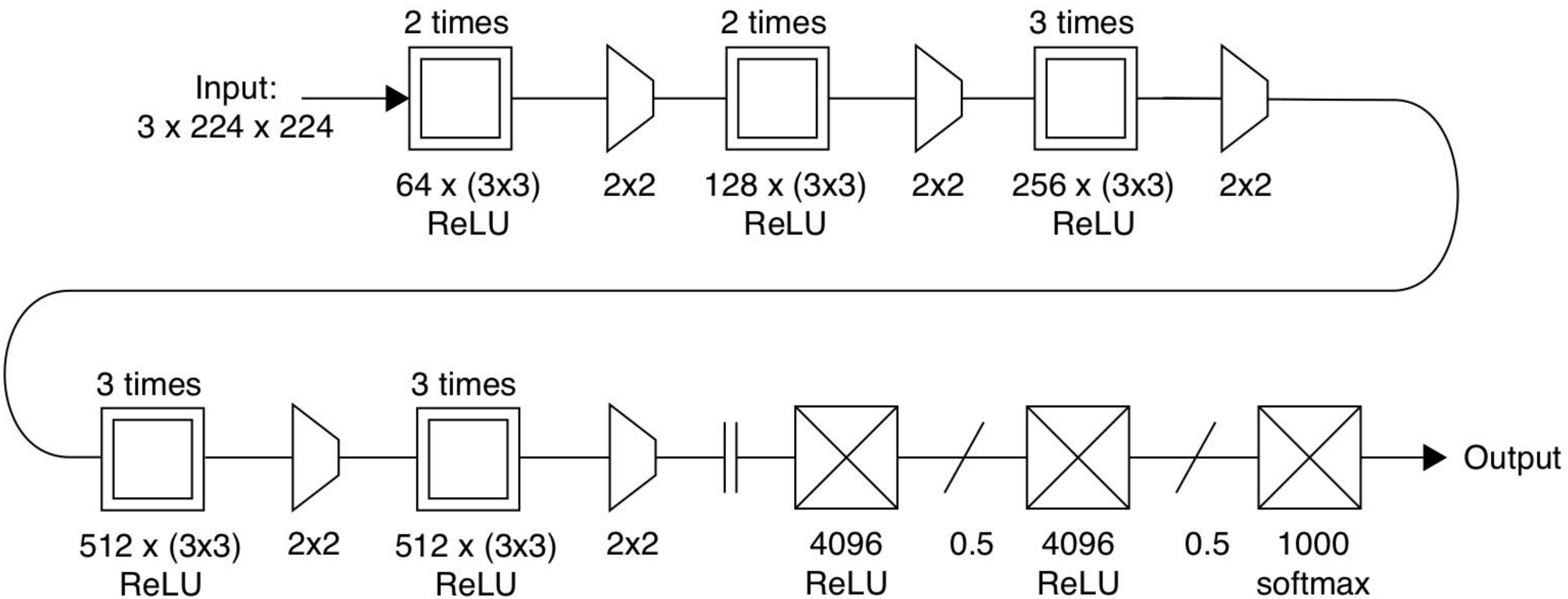
Deep learning for Computer Vision



padding same.

padding same.

VGG16 architecture



Classical Architectures

Architecture	Year	Layers	Key Innovations	Parameters	Researchers
AlexNet	2012	8	CNN Architecture	62 million	Alex Krizhevsky et al.
VGGNet	2014	16-19	3x3 convolution filters, Deep architecture	138-144 million	Karen Simonyan and Andrew Zisserman
Next?					

Classical Architectures

Architecture	Year	Layers	Key Innovations	Parameters	Researchers
AlexNet	2012	8	CNN Architecture	62 million	Alex Krizhevsky et al.
VGGNet	2014	16-19	3x3 convolution filters, Deep architecture	138-144 million	Karen Simonyan and Andrew Zisserman
Inception Net	2014	22-42		**?**	Szegedy et al.

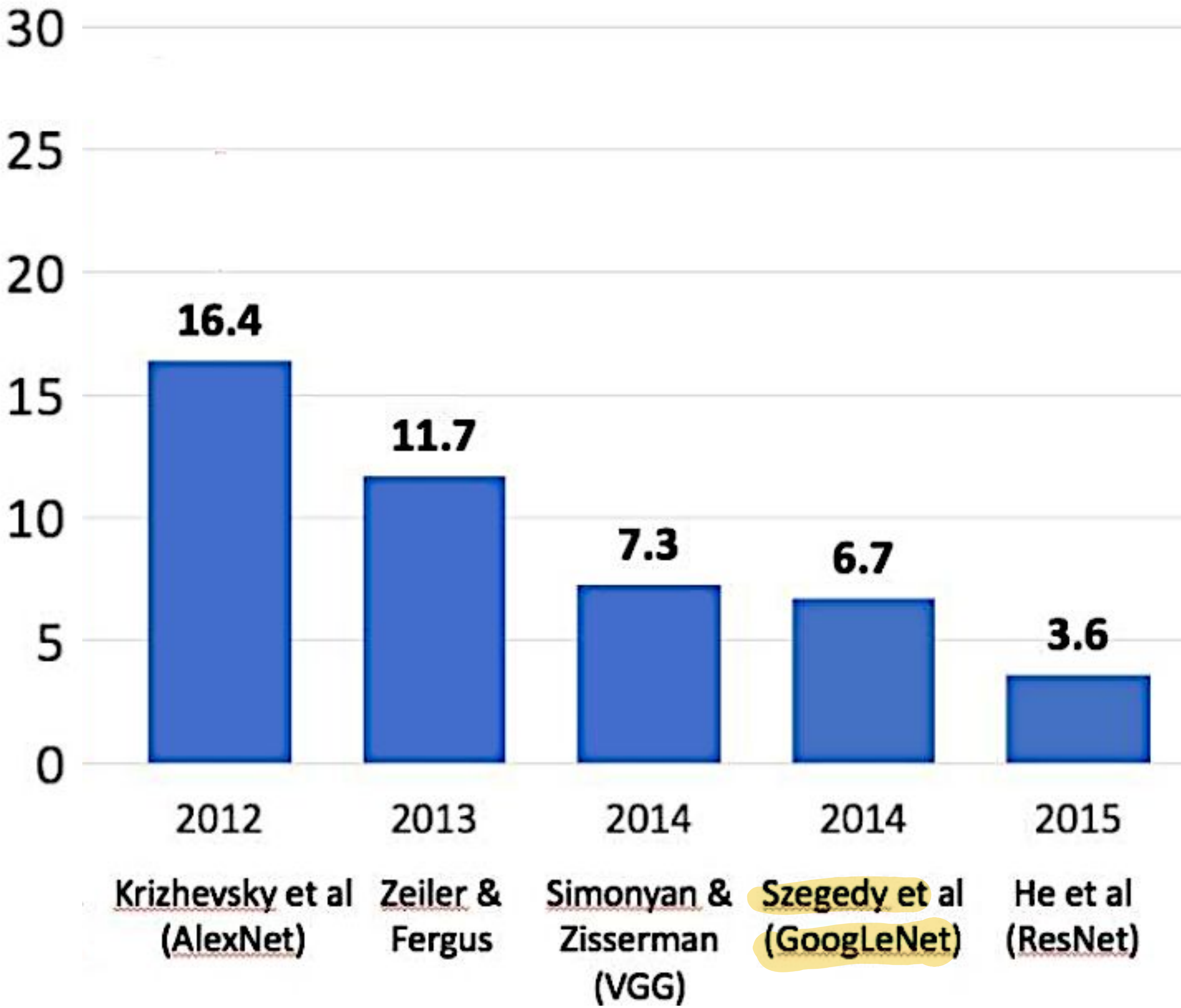
Classical Architectures

Architecture	Year	Layers	Key Innovations	Parameters	Researchers
AlexNet	2012	8	CNN Architecture	62 million	Alex Krizhevsky et al.
VGGNet	2014	16-19	3x3 convolution filters, Deep architecture	138-144 million	Karen Simonyan and Andrew Zisserman
Inception Net	2014	22-42		4-12 million	Szegedy et al.

Going Deeper with Convolutions

Inception Net

Inception Net



Fine-grained visual categories

(a) Siberian husky

(b) Eskimo dog

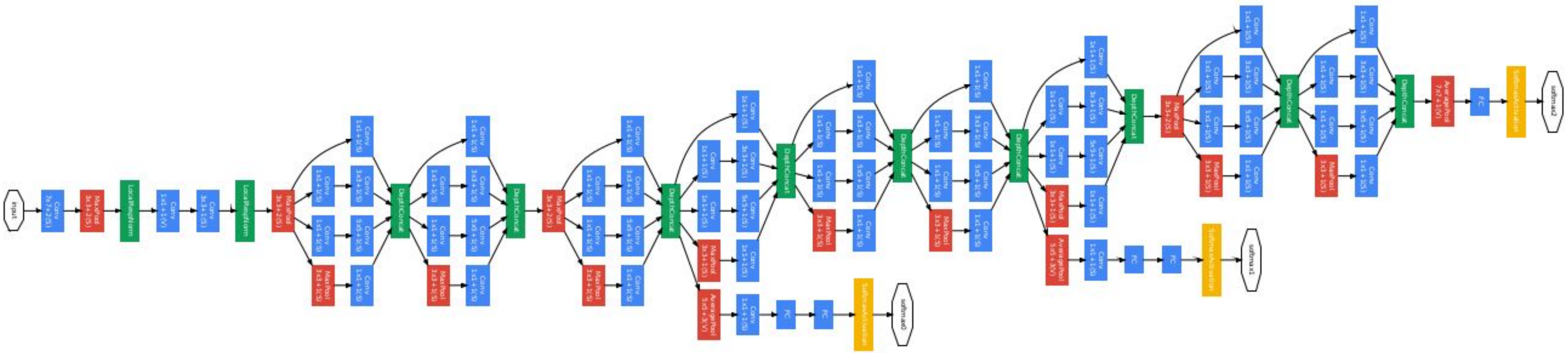
Figure 1: Two distinct classes from the 1000 classes of the ILSVRC 2014 classification challenge.

Difficult to identify for Uniformly increased networks

Inception Net

- Inception Net is a deep convolutional neural network architecture developed by Google researchers in 2014.
- Inception Net won the 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with a top-5 error rate of 6.67%.

Inception Net



Inception Net (Key Idea)

multibranch

multipathway Inception Layer

Auxilary Classifier

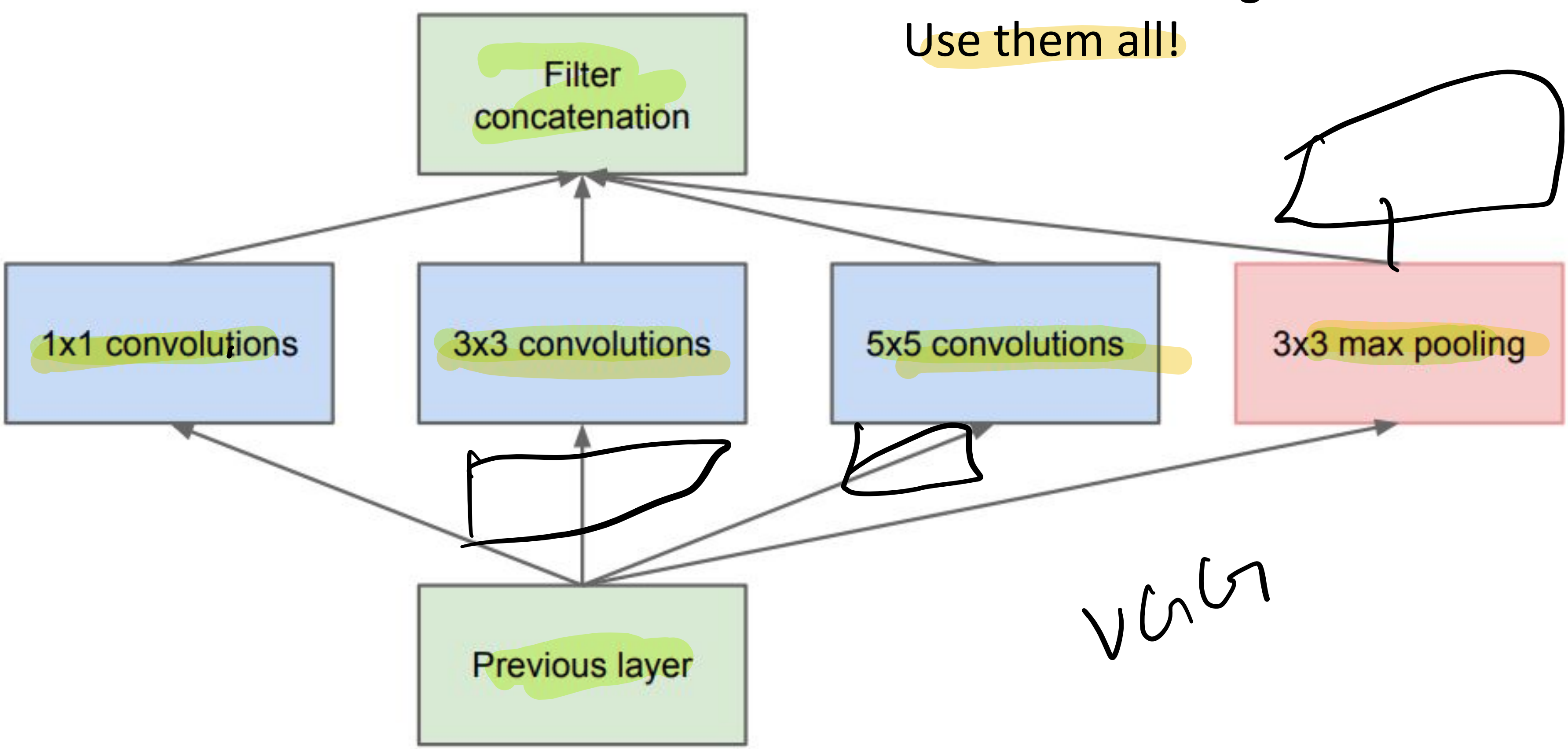
intermediate features

- **Inception Layer**: The multi-pathway convolutional blocks that enable the network to learn complex features using fewer parameters.
- **Auxiliary classifiers**: At intermediate layers of the network to encourage intermediate feature learning.
- Inception Net uses a multi-branch architecture that allows it to learn features at multiple scales and resolutions.

Inception Layer

Inception Layer

Tired of choosing filter sizes?
Use them all!

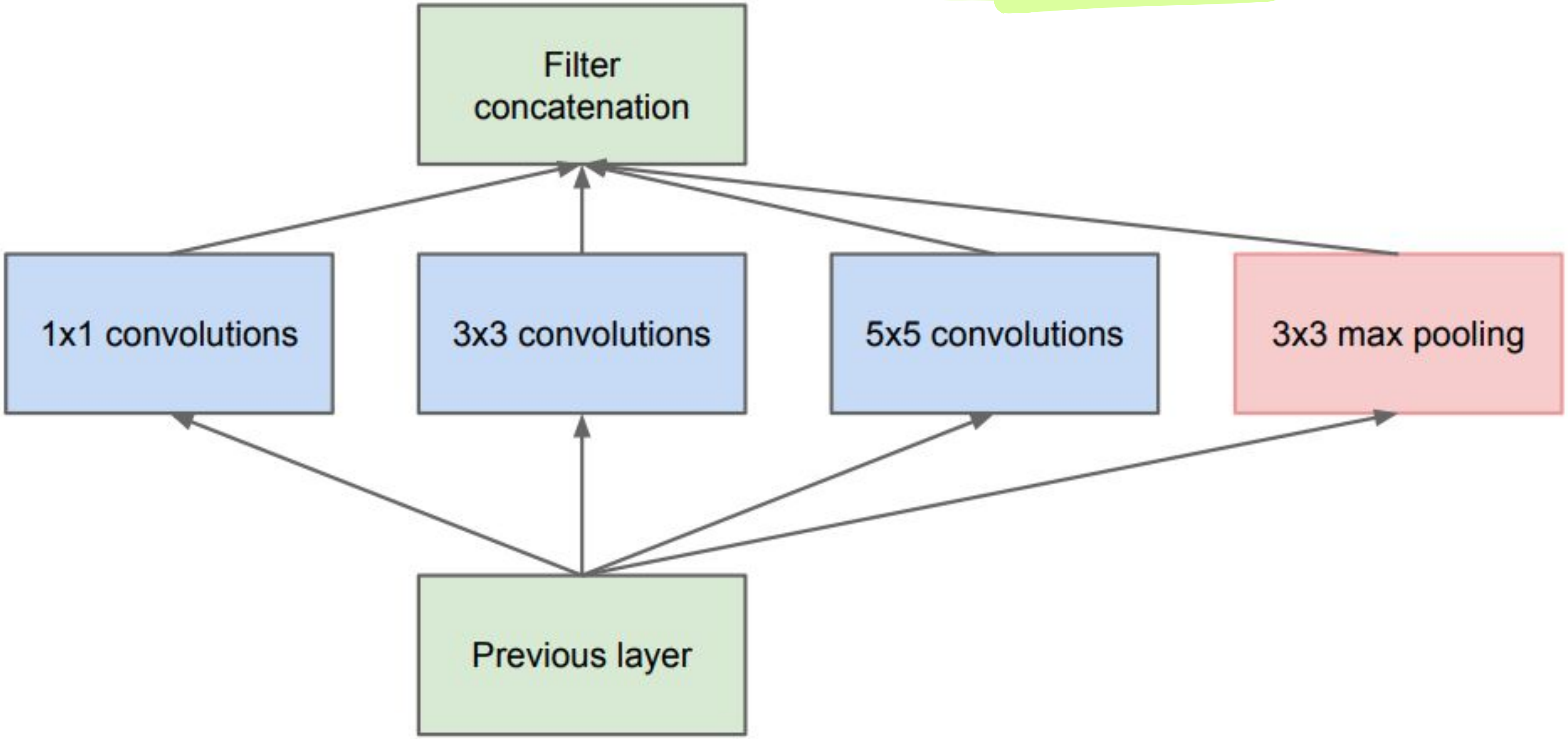


VGG

(a) Inception module, naïve version

Inception Layer

Use padding = Same

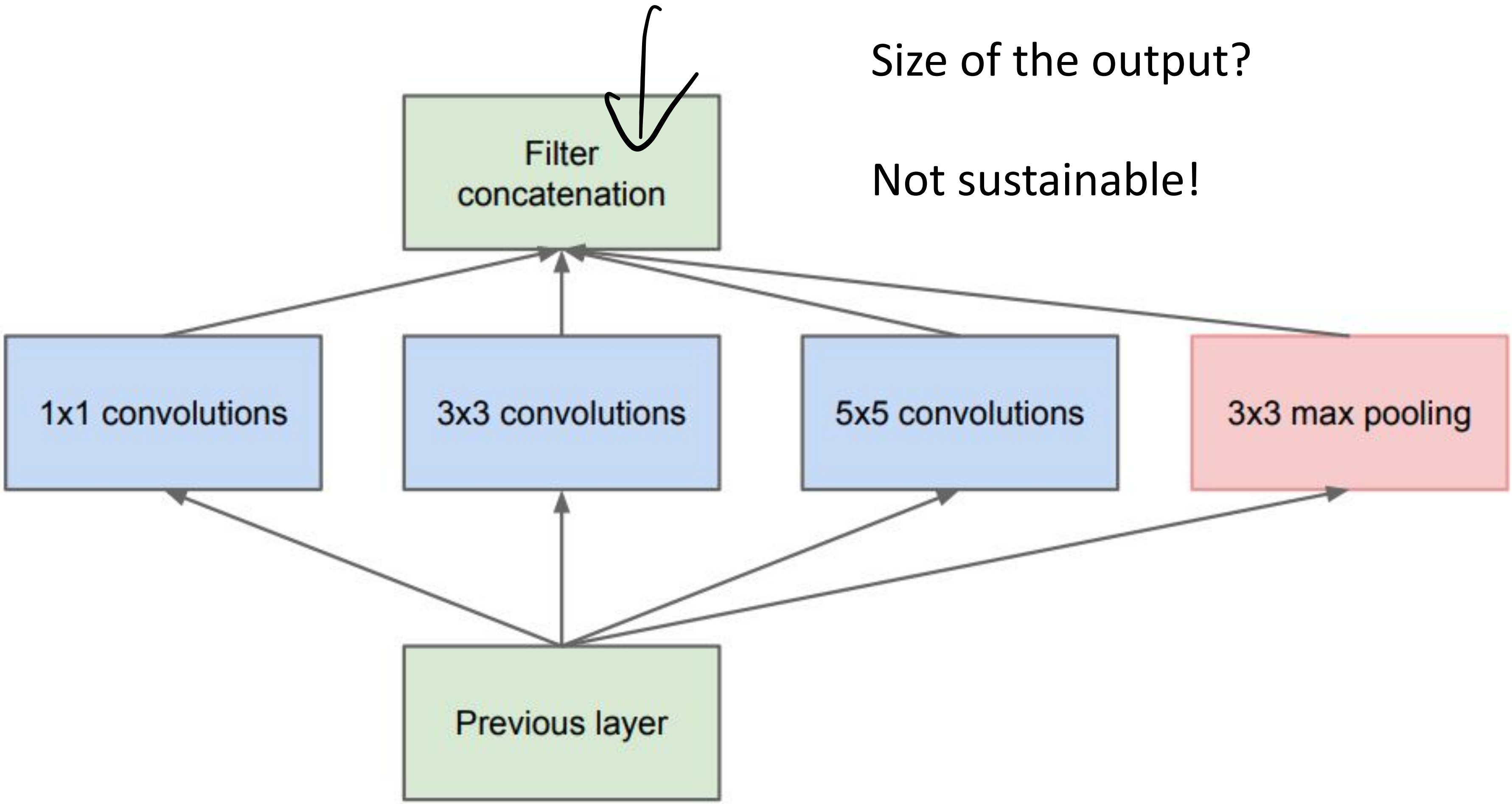


(a) Inception module, naïve version

Inception Layer

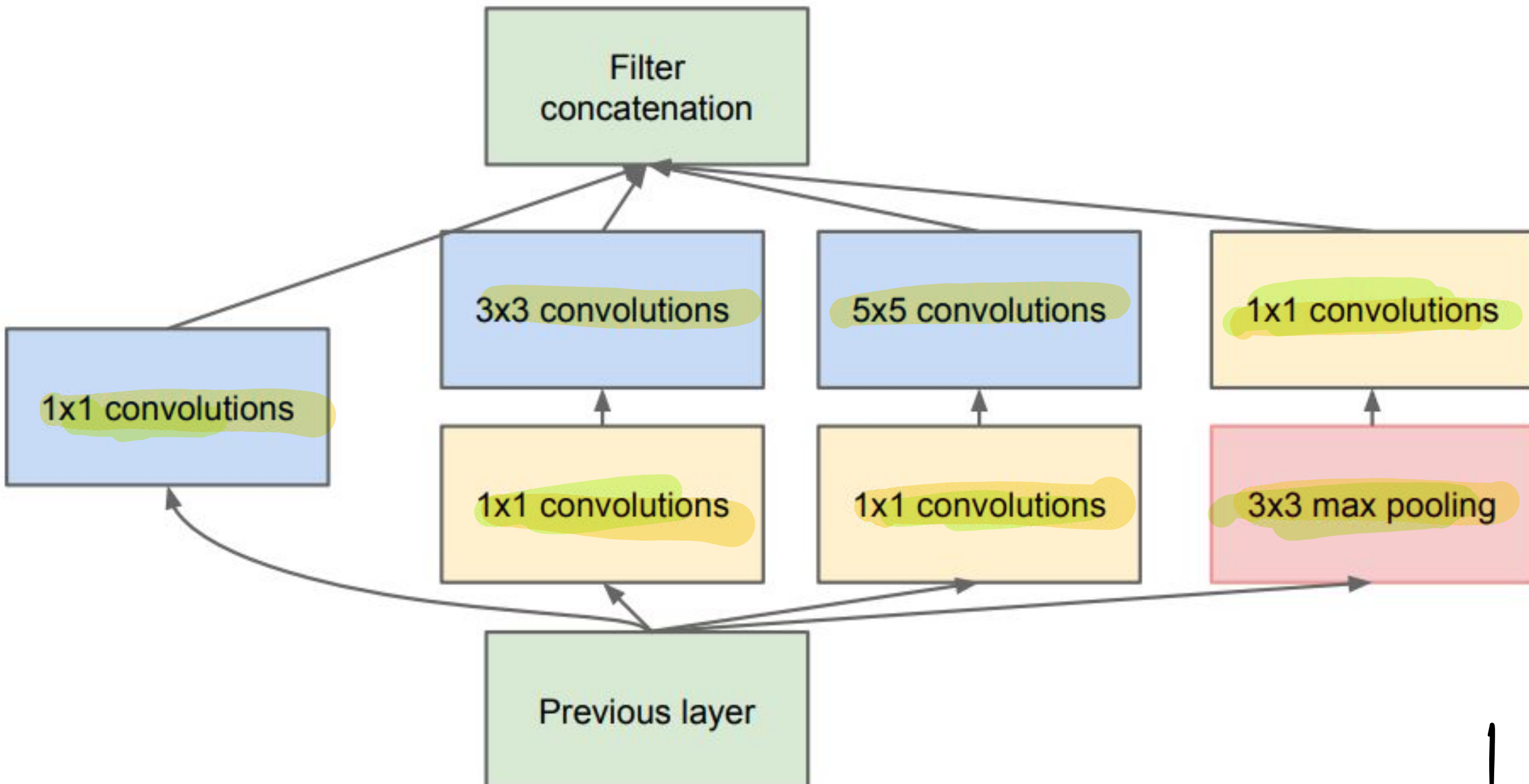
Size of the output?

Not sustainable!



(a) Inception module, naïve version

Inception Layer (key idea)



1 x 1.
Max pool

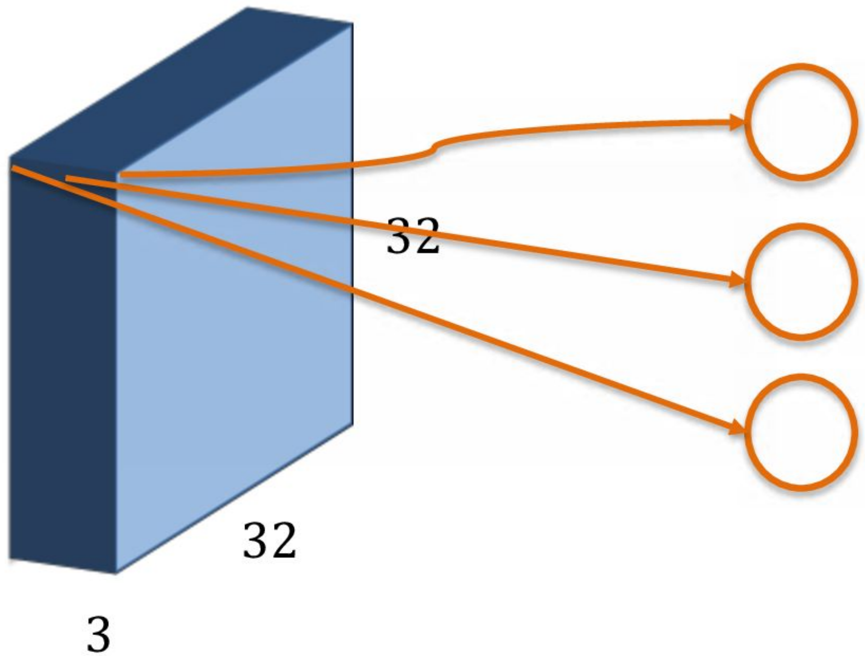
1x1

3 x 3
1x1

5 x 5
1 x 1

1x1 Convolutions

1x1 Convolution



1 output

1x1 Convolution

Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

-10				

Kernel 1x1

2

$$-5 * 2 = -10$$

1x1 Convolution

Image 5x5

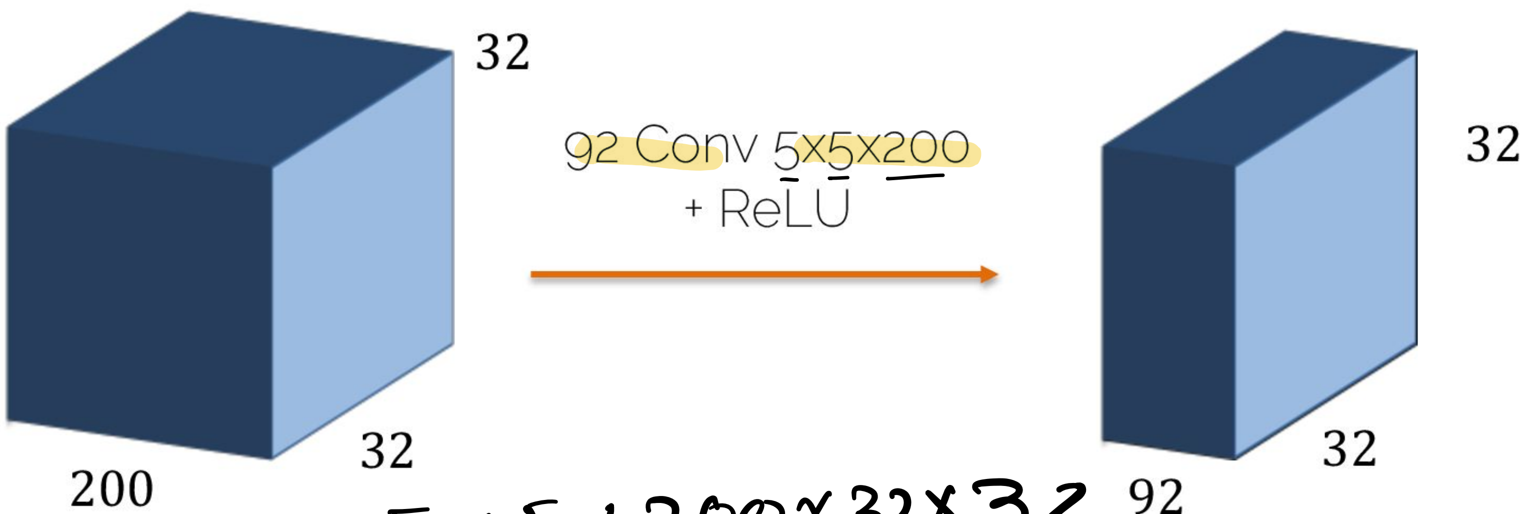
-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

-10	6	4	-10	6
8	6	4	2	-6
2	0	6	6	10
-4	0	2	8	8
10	12	14	18	-2

1x1 kernel keeps the dimensions and scales input!

Inception Layer: Computational Cost

total multiplications = $5 \times 5 \times 200 \times 32 \times 32 \times 92$.

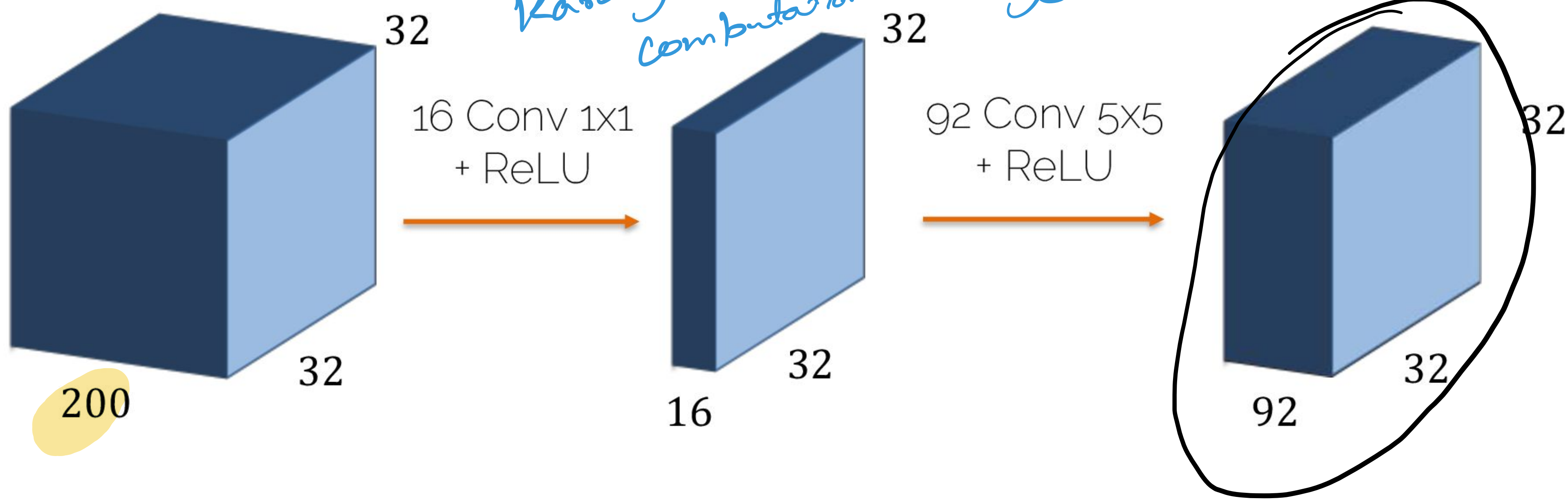


$5 \times 5 \times 200 \times 32 \times 32$

Multiplications: 5x5x200 x 32x32x92 ~ 470 million

Inception Layer: Computational Cost

200 kam karke 16
Kardiye Jiske karan kam computation hogyi

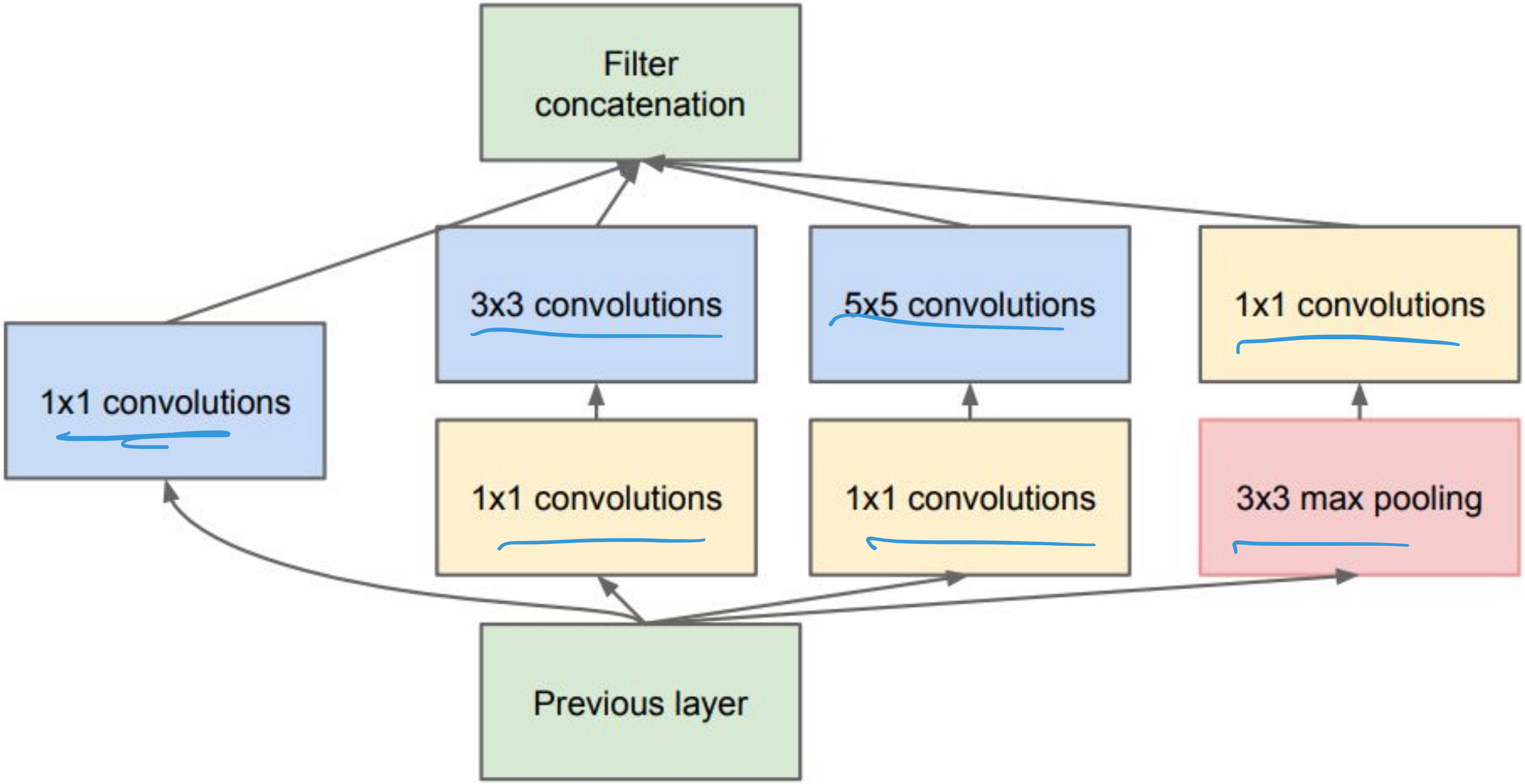


Multiplications: 1x1x200x32x32x16

5x5x16x32x32x92 ~ 40 million

Reduction of multiplications by 1/10

Inception Layer



(b) Inception module with dimensionality reduction

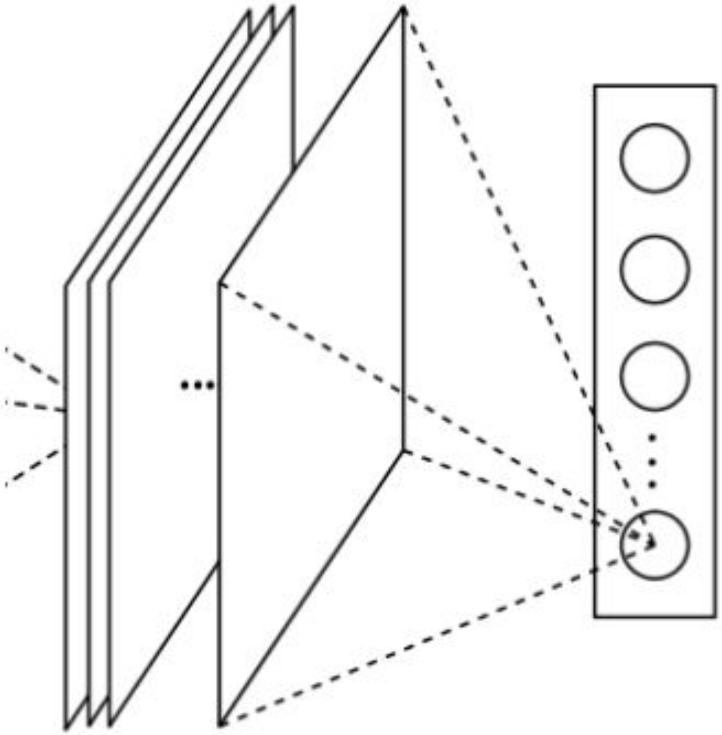
```python
def inception_module(x, filters):
    """
    Inception module of the InceptionNet
    """
    tower_1 = Conv2D(filters[0], (1, 1), padding='same', activation='relu')(x)
    tower_1 = Conv2D(filters[1], (3, 3), padding='same', activation='relu')(tower_1)

    tower_2 = Conv2D(filters[2], (1, 1), padding='same', activation='relu')(x)
    tower_2 = Conv2D(filters[3], (5, 5), padding='same', activation='relu')(tower_2)

    tower_3 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
    tower_3 = Conv2D(filters[4], (1, 1), padding='same', activation='relu')(tower_3)

    output = Concatenate(axis=-1)([tower_1, tower_2, tower_3])
    return output
```

InceptionNet

```
Input
  |
Conv2D -> ReLU -> MaxPooling2D ...
  |
Inception module
  |
  ...
  |
  ...
  |
Inception module
  |
GlobalAveragePooling -> Dense -> Softmax
```

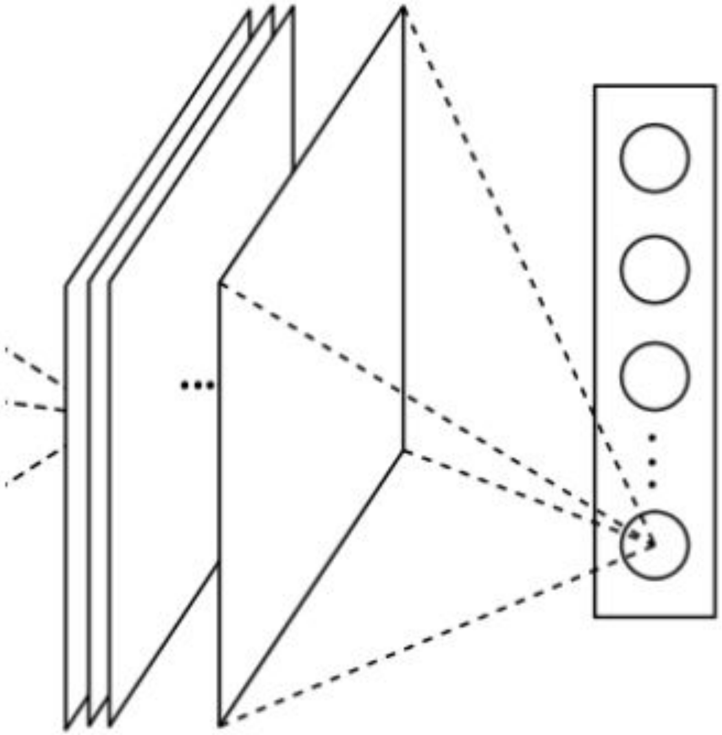
GlobalAveragePooling

- Global Average Pooling replace fully connected layers in classical CNNs.



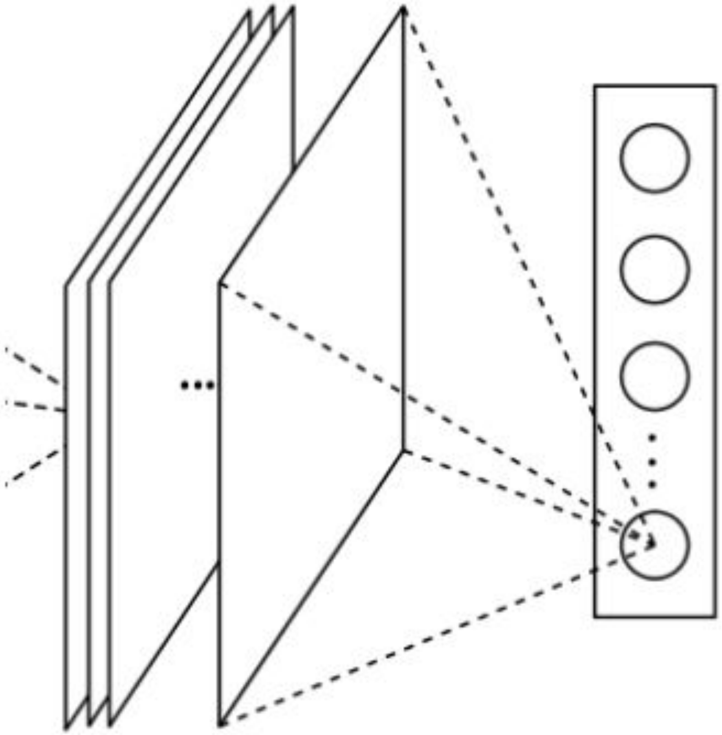
GlobalAveragePooling

- Global Average Pooling replace fully connected layers in classical CNNs.
- In this layer, the average value of each feature map is computed, resulting in a single output value for each feature map.

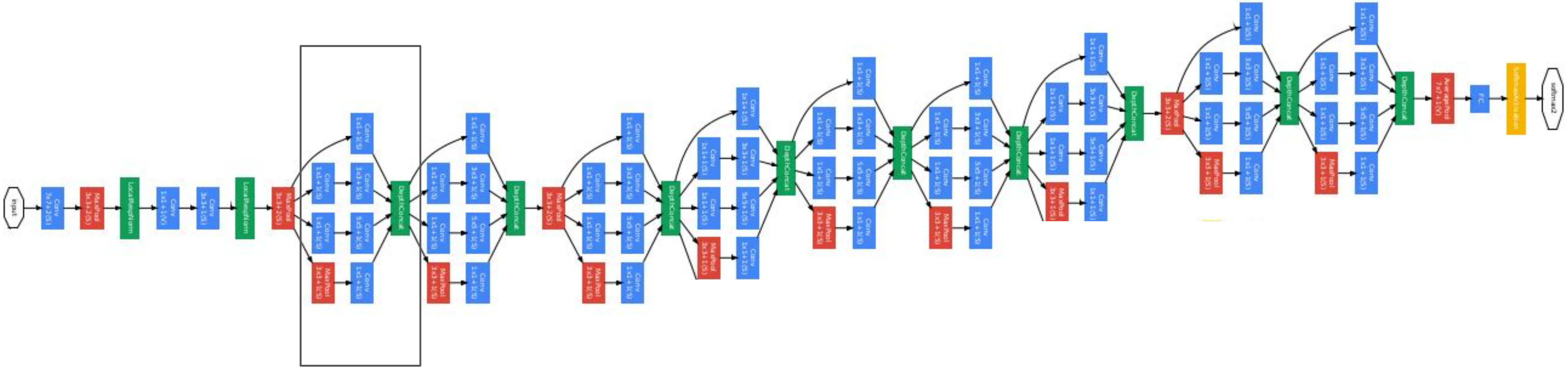


GlobalAveragePooling

- Global Average Pooling replace fully connected layers in classical CNNs.
- In this layer, the average value of each feature map is computed, resulting in a single output value for each feature map.
- Global Average Pooling helps reduce the number of parameters in the network.



Inception Layer in InceptionNet



```python
def InceptionNet(input_shape, num_classes):
    """
    InceptionNet architecture using functional API.
    """
    input_tensor = Input(shape=input_shape)

    x = Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu')(input_tensor)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = Conv2D(64, (1, 1), padding='same', activation='relu')(x)
    x = Conv2D(192, (3, 3), padding='same', activation='relu')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = inception_module(x, [64, 128, 32, 32, 64])
    x = inception_module(x, [128, 192, 96, 64, 128])

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
```

```python
    x = inception_module(x, [192, 208, 48, 64, 96])
    x = inception_module(x, [160, 224, 64, 64, 112])
    x = inception_module(x, [128, 256, 64, 64, 128])
    x = inception_module(x, [112, 288, 64, 64, 144])
    x = inception_module(x, [256, 320, 128, 128, 160])

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = inception_module(x, [256, 320, 128, 128, 160])
    x = inception_module(x, [384, 384, 128, 128, 128])

    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.4)(x)
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=input_tensor, outputs=x, name='InceptionNet')
    return model
```

Auxiliary classifiers

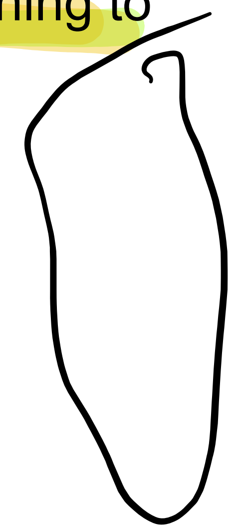
Auxiliary classifiers

- The auxiliary classifiers in InceptionNet are additional output branches that are inserted into the network at intermediate stages.

Auxiliary classifiers

- The auxiliary classifiers in InceptionNet are additional output branches that are inserted into the network at intermediate stages.
- Auxiliary classifiers provide additional supervision signals during training to improve the overall performance of the network.

additional supervision
signals.



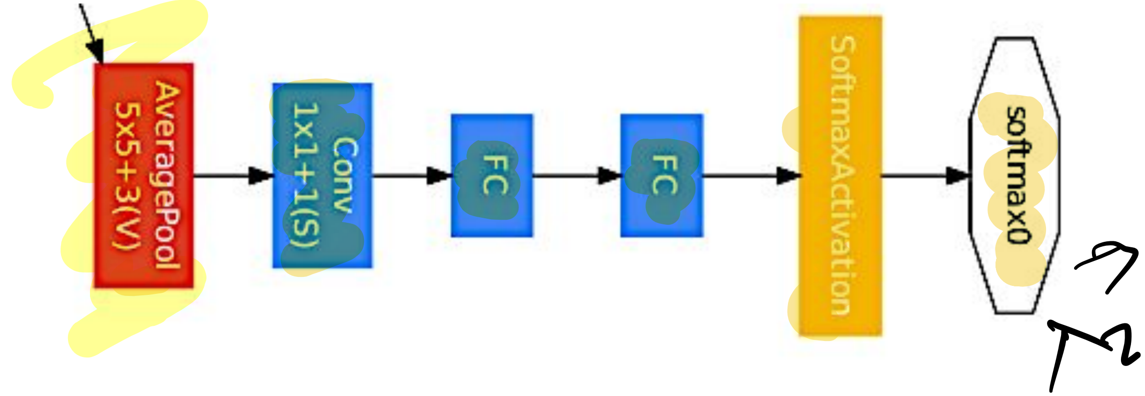
50

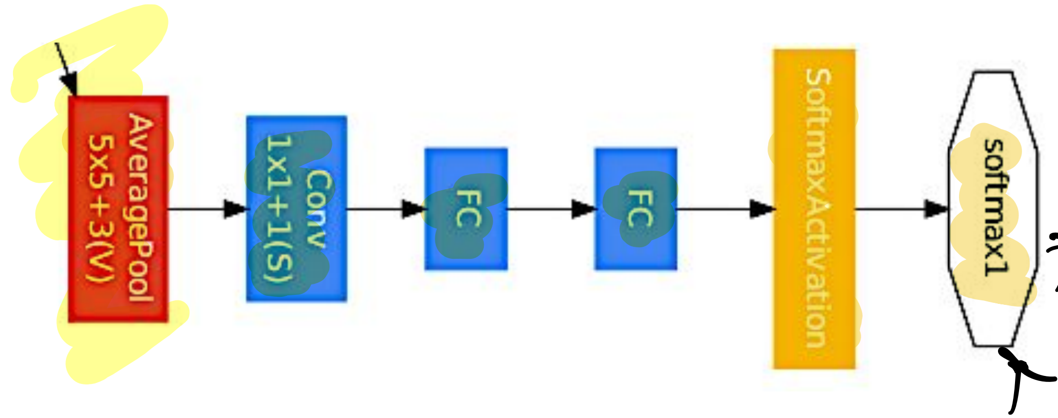
1

Auxiliary classifiers → Additional supervision.

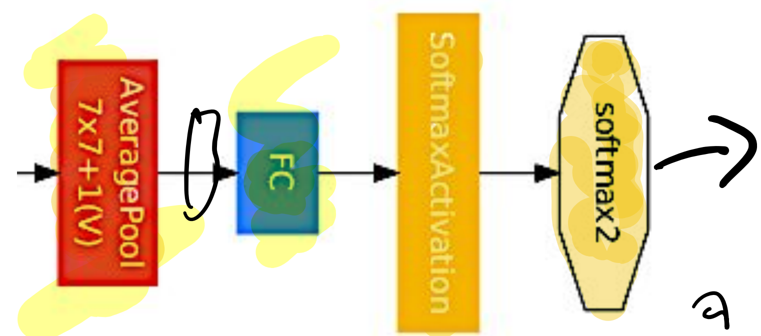
- The auxiliary classifiers in InceptionNet are additional output branches that are inserted into the network at intermediate stages.
- Auxiliary classifiers provide additional supervision signals during training to improve the overall performance of the network.
- The use of auxiliary classifiers is not limited to InceptionNet and can be applied to other deep learning architectures as well.

Auxiliary classifiers





Main classifier



Auxiliary classifiers

- **During training**, the loss from the auxiliary classifiers is added to the overall loss (main classifier) of the network with a weight factor (usually 0.3).

Auxiliary classifiers

- **During training**, the loss from the auxiliary classifiers is added to the overall loss (main classifier) of the network with a weight factor (usually 0.3).
- **During inference**, the outputs of the auxiliary classifiers are discarded, and only the output of the main classifier is used to make predictions.

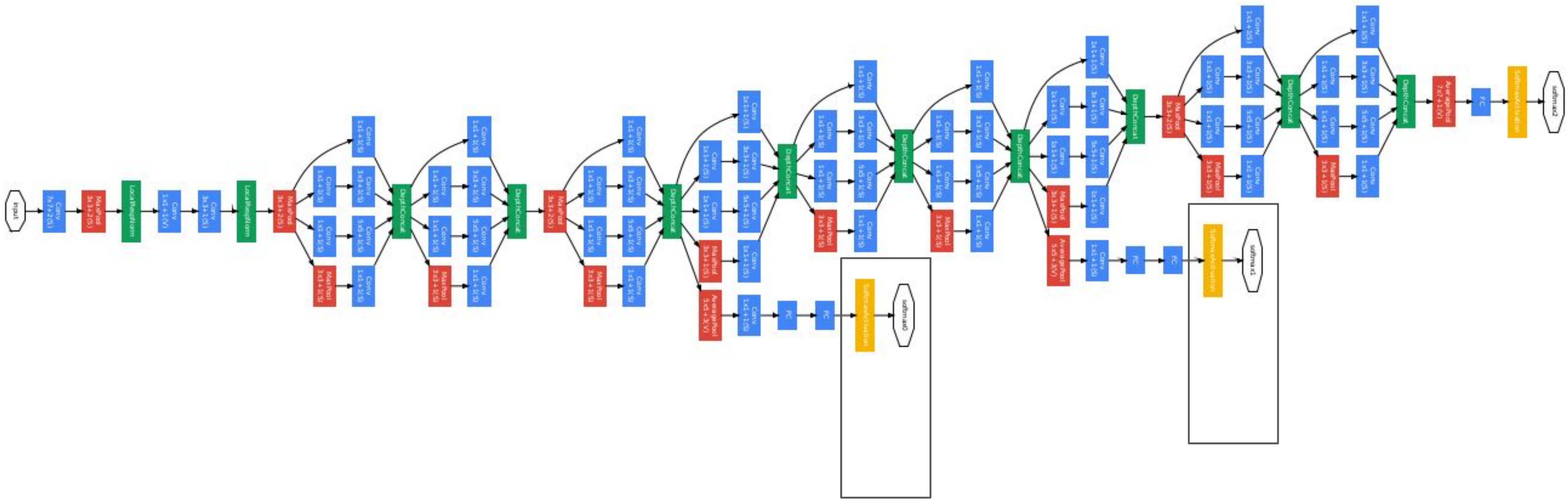
Auxiliary classifiers

- **During training**, the loss from the auxiliary classifiers is added to the overall loss (main classifier) of the network with a weight factor (usually 0.3).
- **During inference**, the outputs of the auxiliary classifiers are discarded, and only the output of the main classifier is used to make predictions.
- The number and placement of the auxiliary classifiers in InceptionNet can vary depending on the specific architecture and task.

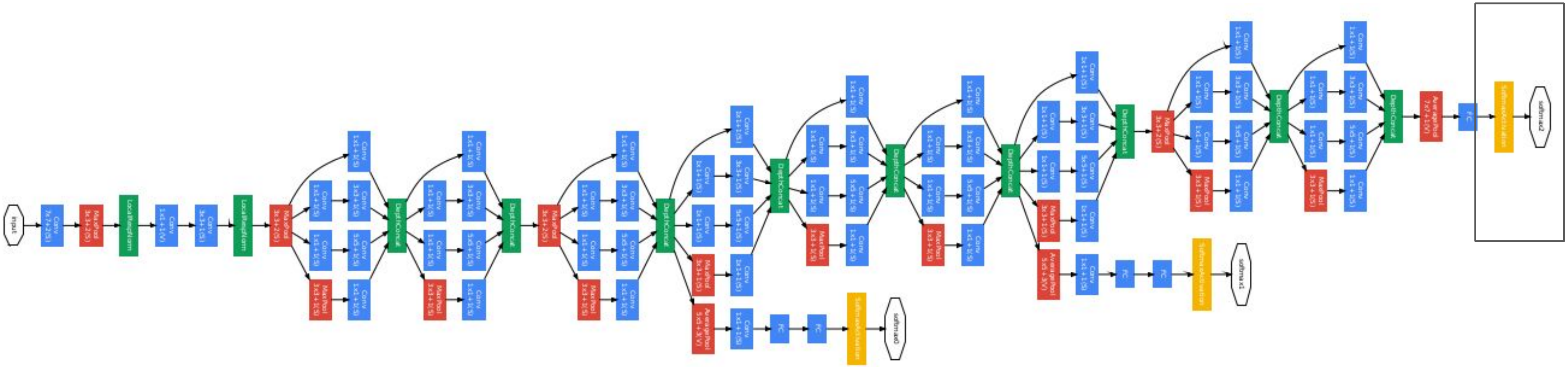
Auxiliary classifiers

- The auxiliary classifiers in InceptionNet are additional output branches that are inserted into the network at intermediate stages.
- Auxiliary classifiers provide additional supervision signals during training to improve the overall performance of the network.
- The use of auxiliary classifiers is not limited to InceptionNet and can be applied to other deep learning architectures as well.
- Auxiliary classifiers push useful gradients to the lower layers to make them immediately useful and improve the convergence during training by combating the vanishing gradient.

Inception Net

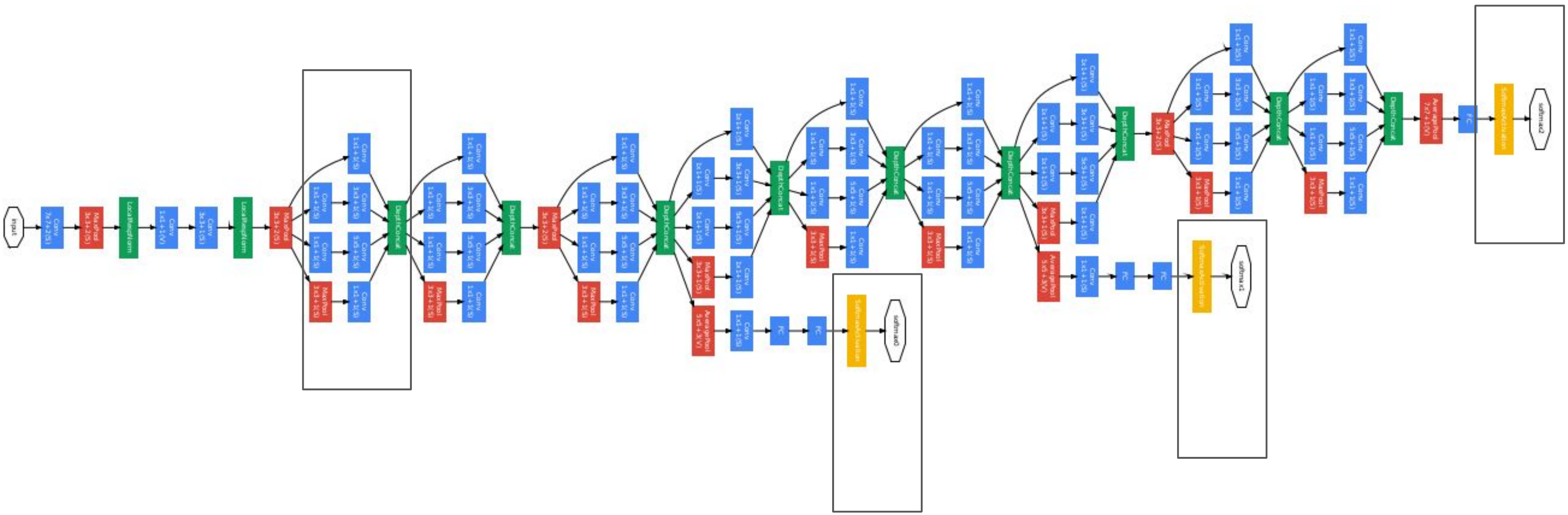


Inception Net



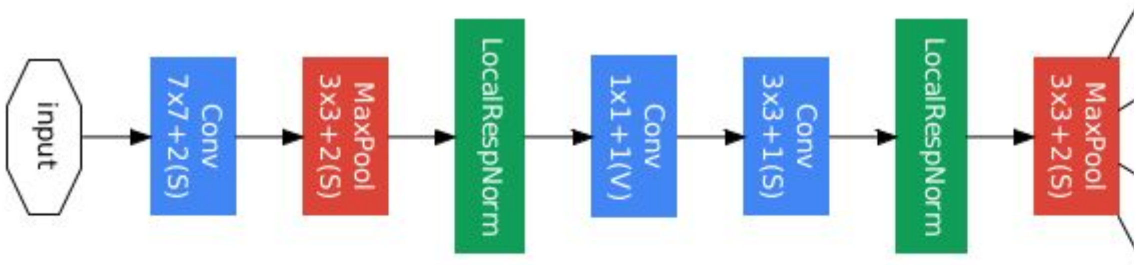
Inception Net

Inception Net (Main Components)



Inception Net

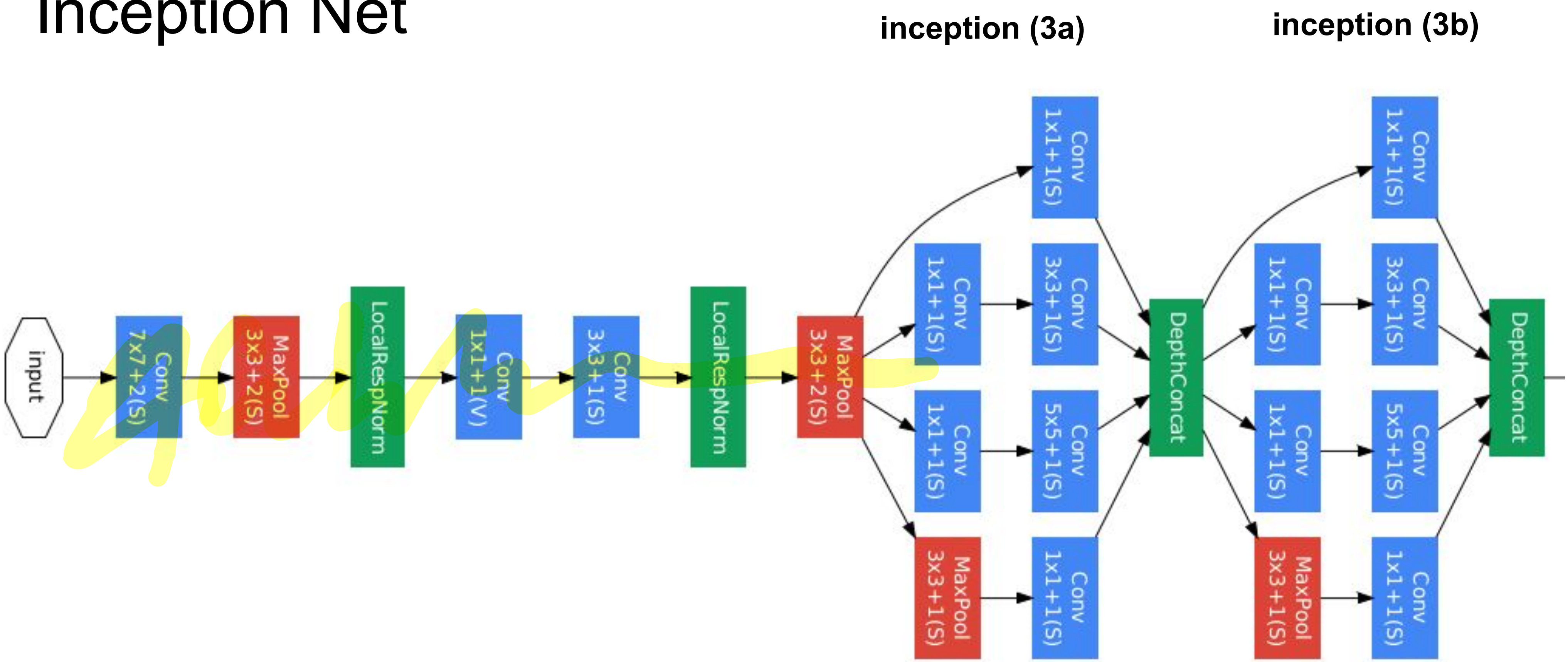
inception (3a) inception (3b)



The input to the network is a 224x224x3 RGB image.

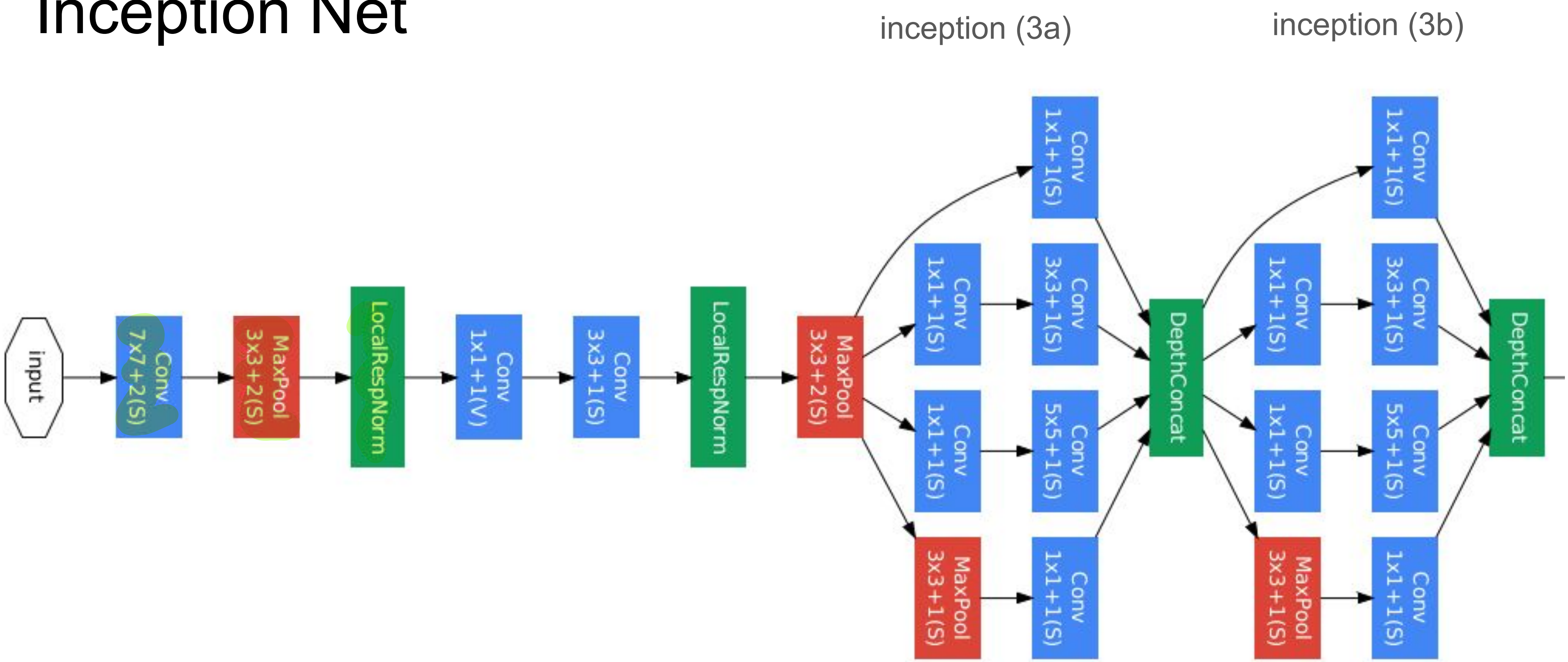
The network begins with a series of convolutional and pooling layers to extract low-level features from the image.

Inception Net



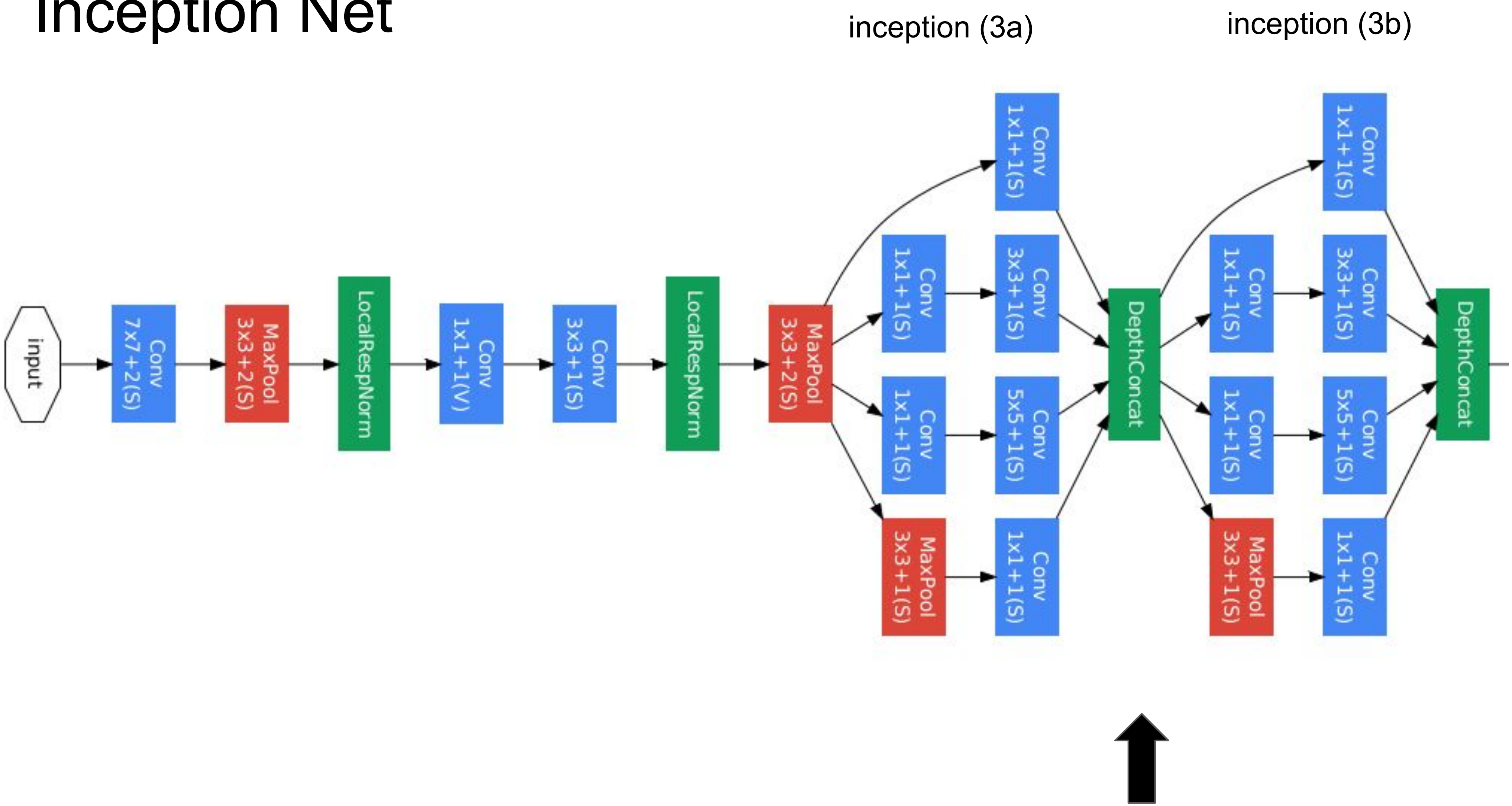
The Inception module contains multiple parallel convolutional paths of different filter sizes, including 1x1, 3x3, and 5x5 convolutions.

Inception Net



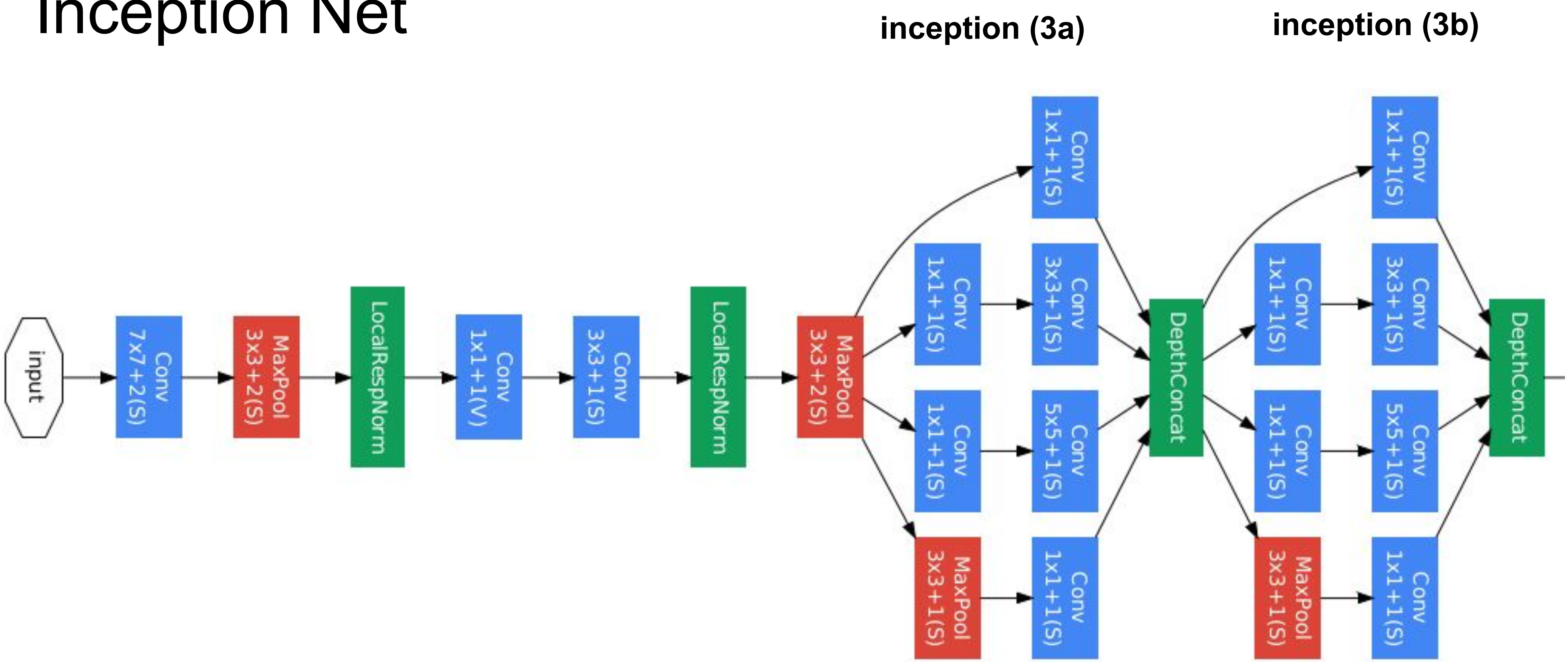
The pooling operations and 1x1 convolutions in Inception modules to reduce the dimensionality of the input.

Inception Net



The outputs of each path are concatenated together along the channel axis and fed into the next layer.

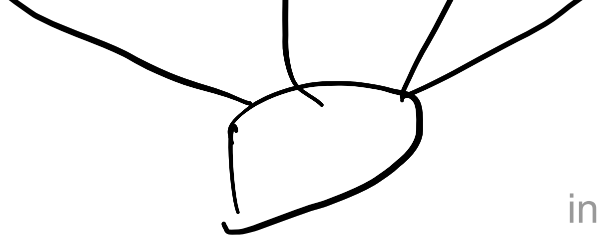
Inception Net



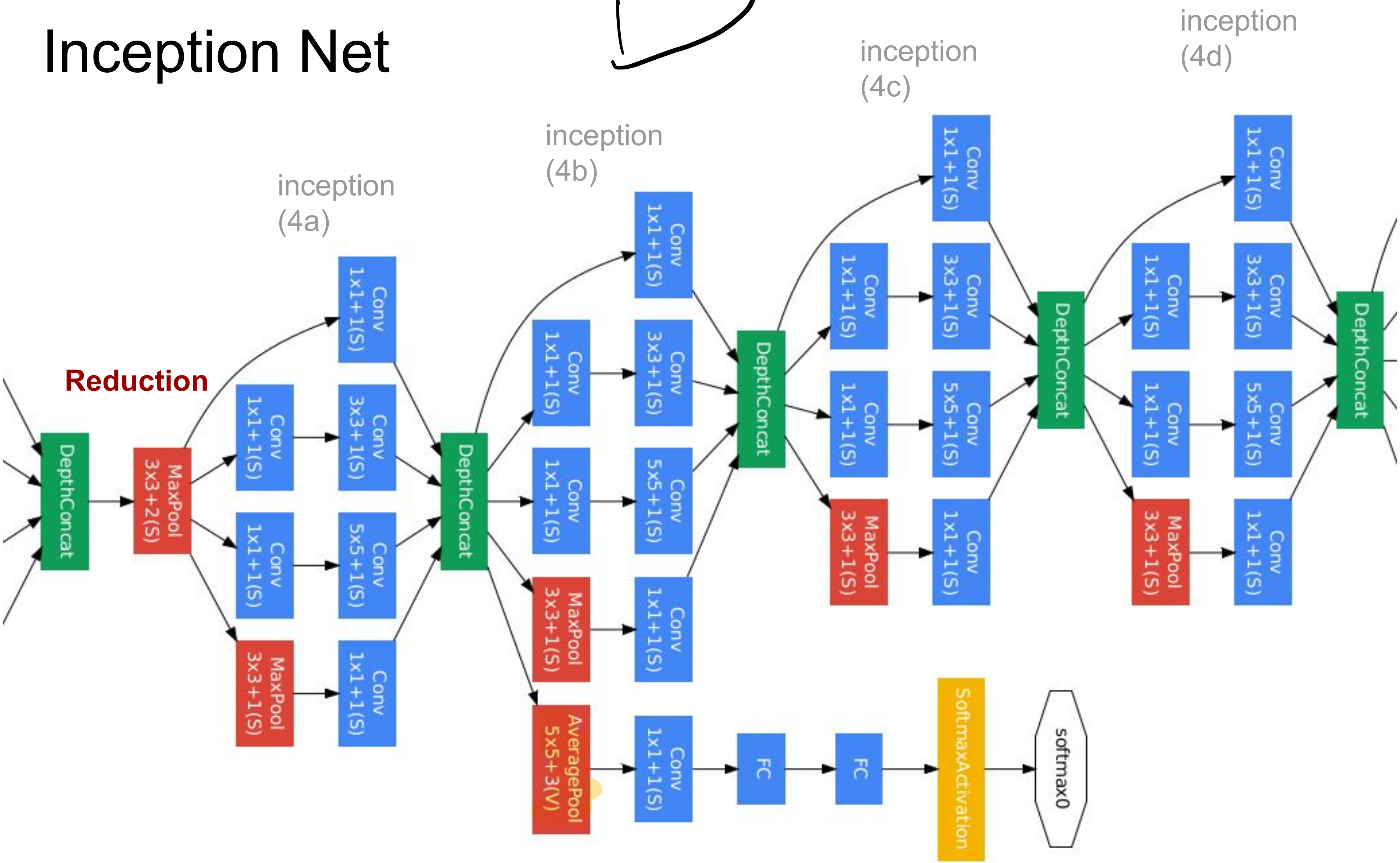
Inception modules are stacked on top of each other to form the "stem" of the network.

3

The stem is followed by a series of "Inception-A" and "Inception-B" modules.

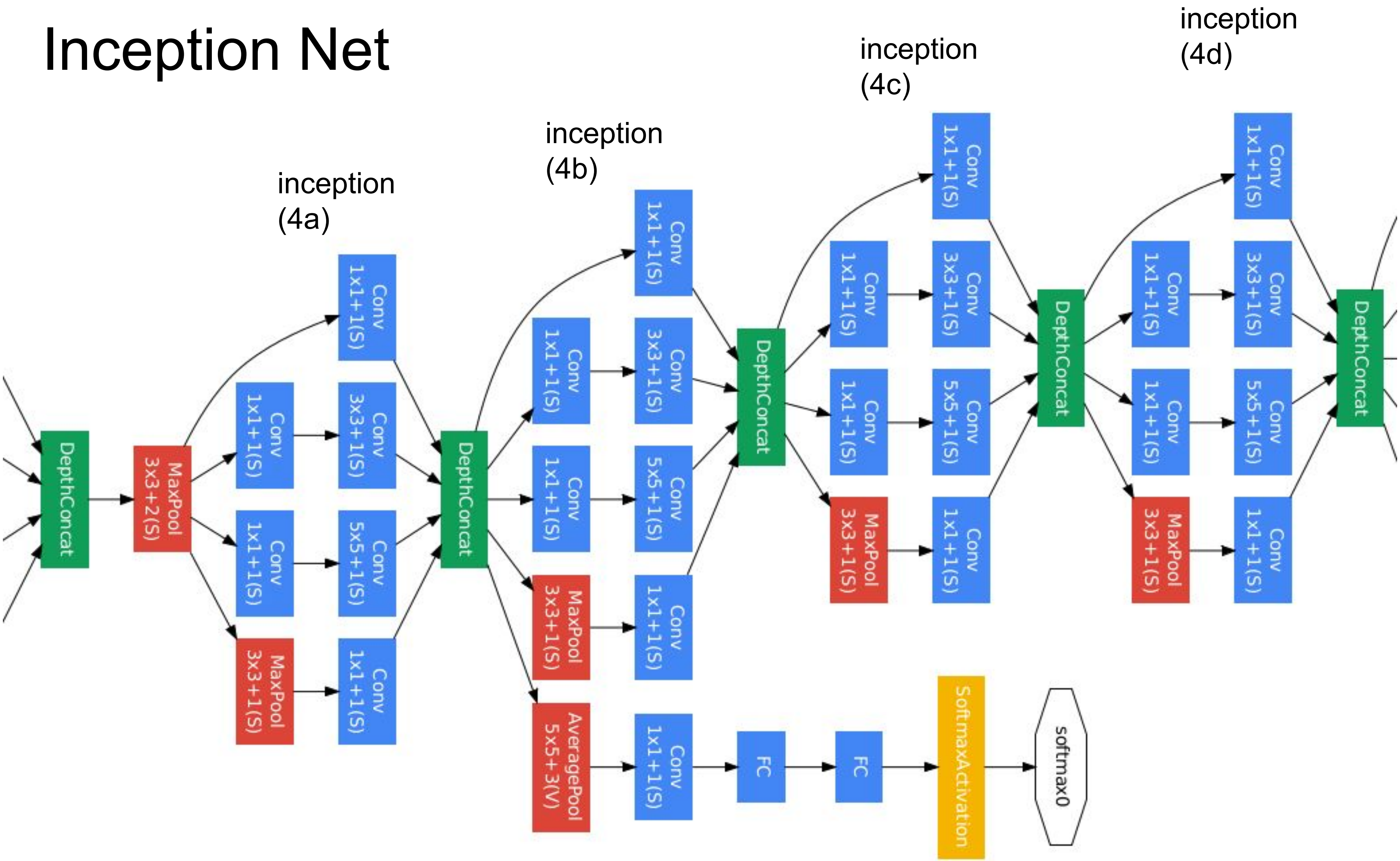


Inception Net



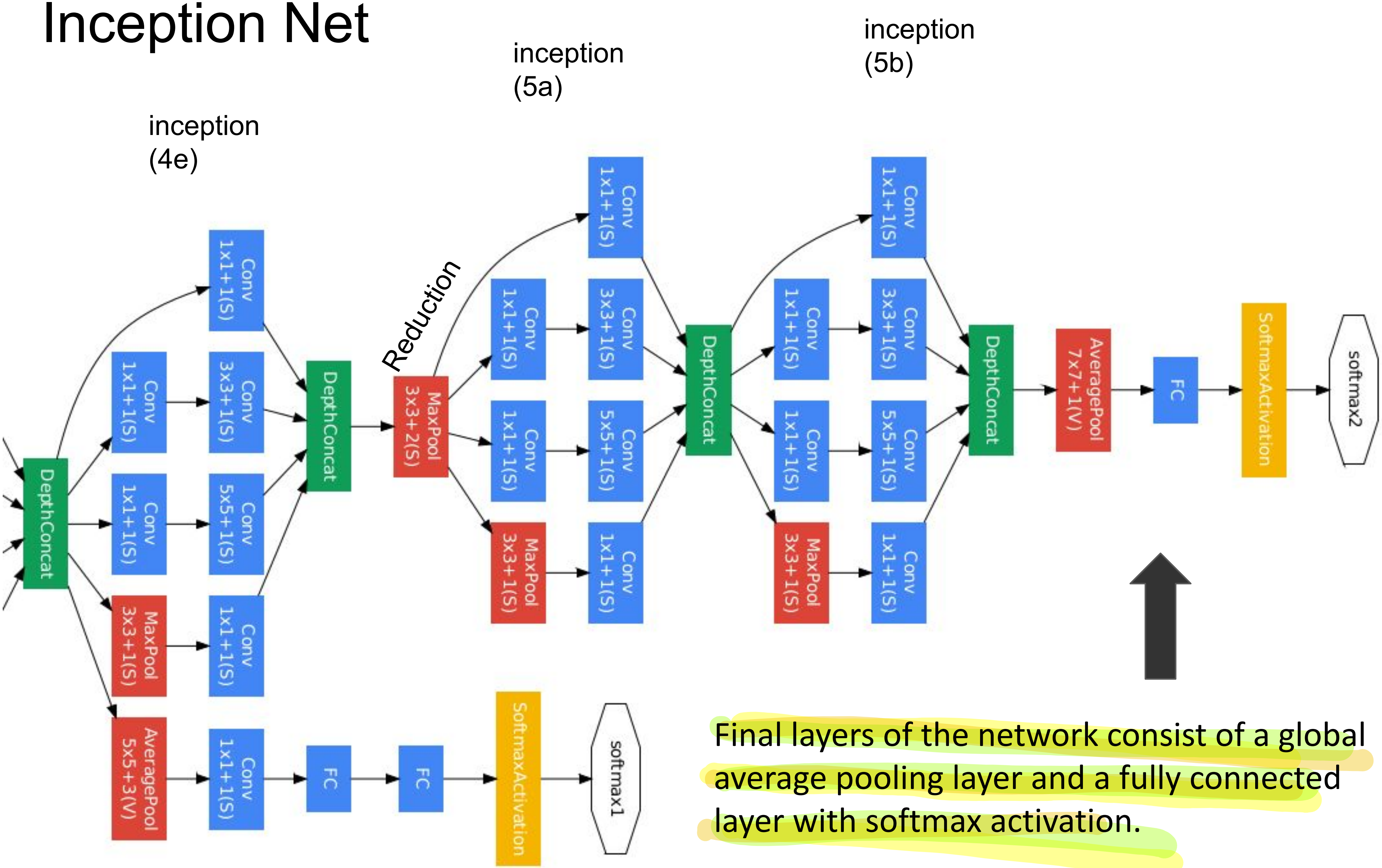
The network also includes several "Reduction" modules, which are used to reduce the spatial dimensions of the feature maps.

Inception Net



In addition to the main classification, the network also includes two "Auxiliary classifiers" at intermediate layers.

Inception Net



Final layers of the network consist of a global average pooling layer and a fully connected layer with softmax activation.

```python
import tensorflow as tf
from tensorflow.keras.applications.inception_v3 import InceptionV3,
preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np

# Load the InceptionV3 model
model = InceptionV3(weights='imagenet')

# Load the image you want to classify
img_path = 'tiger_shark.jpeg'
img = image.load_img(img_path, target_size=(299, 299))

# Convert the image to an array
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

# Use the model to predict the class of the image
preds = model.predict(x)

# Print the top 5 predictions
print('Predicted:', decode_predictions(preds, top=5)[0])
```

```python
def InceptionNet(input_shape, num_classes):
    """
    InceptionNet architecture using functional API.
    """
    input_tensor = Input(shape=input_shape)

    x = Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu')(input_tensor)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = Conv2D(64, (1, 1), padding='same', activation='relu')(x)
    x = Conv2D(192, (3, 3), padding='same', activation='relu')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = inception_module(x, [64, 96, 128, 16, 32])
    x = inception_module(x, [128, 128, 192, 32, 96])
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = inception_module(x, [192, 96, 208, 16, 48])
```

```python
# Auxiliary Classifier 1
aux_output_1 = AveragePooling2D((5, 5), strides=(3, 3))(x)
aux_output_1 = Conv2D(128, (1, 1), padding='same', activation='relu')(aux_output_1)
aux_output_1 = Flatten()(aux_output_1)
aux_output_1 = Dense(1024, activation='relu')(aux_output_1)
aux_output_1 = Dropout(0.7)(aux_output_1)
aux_output_1 = Dense(num_classes, activation='softmax')(aux_output_1)

x = inception_module(x, [160, 112, 224, 24, 64])
x = inception_module(x, [128, 128, 256, 24, 64])
x = inception_module(x, [112, 144, 288, 32, 64])
```

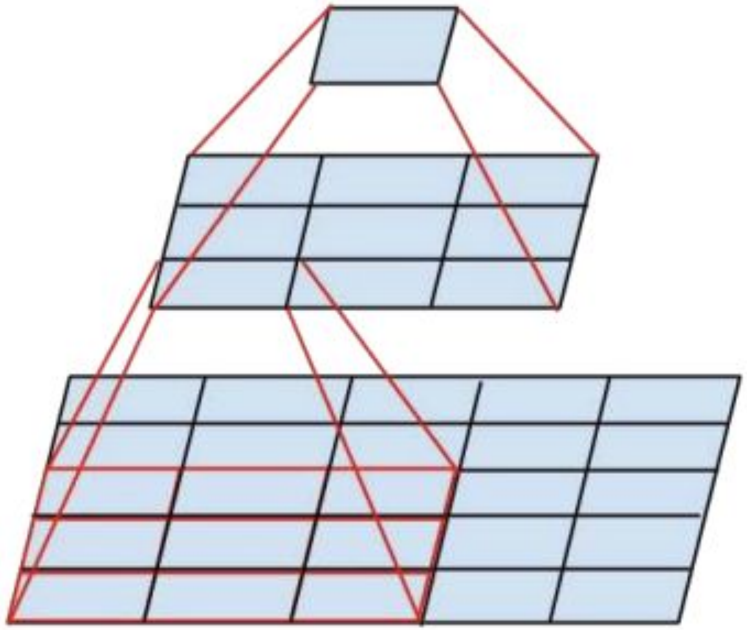
```python
# Auxiliary Classifier 2
aux_output_2 = AveragePooling2D((5, 5), strides=(3, 3))(x)
aux_output_2 = Conv2D(128, (1, 1), padding='same', activation='relu')(aux_output_2)
aux_output_2 = Flatten()(aux_output_2)
aux_output_2 = Dense(1024, activation='relu')(aux_output_2)
aux_output_2 = Dropout(0.7)(aux_output_2)
aux_output_2 = Dense(num_classes, activation='softmax')(aux_output_2)
```

```python
# Auxiliary Classifier 2
aux_output_2 = AveragePooling2D((5, 5), strides=(3, 3))(x)
aux_output_2 = Conv2D(128, (1, 1), padding='same', activation='relu')(aux_output_2)
aux_output_2 = Flatten()(aux_output_2)
aux_output_2 = Dense(1024, activation='relu')(aux_output_2)
aux_output_2 = Dropout(0.7)(aux_output_2)
aux_output_2 = Dense(num_classes, activation='softmax')(aux_output_2)

x = inception_module(x, [256, 160, 320, 32, 128])
x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

x = inception_module(x, [256, 160, 320, 32, 128])
x = inception_module(x, [384, 192, 384, 48, 128])

x = GlobalAveragePooling2D((7, 7))(x)
x = Dropout(0.4)(x)

output = Dense(num_classes, activation='softmax')(x)
```

InceptionNet variants

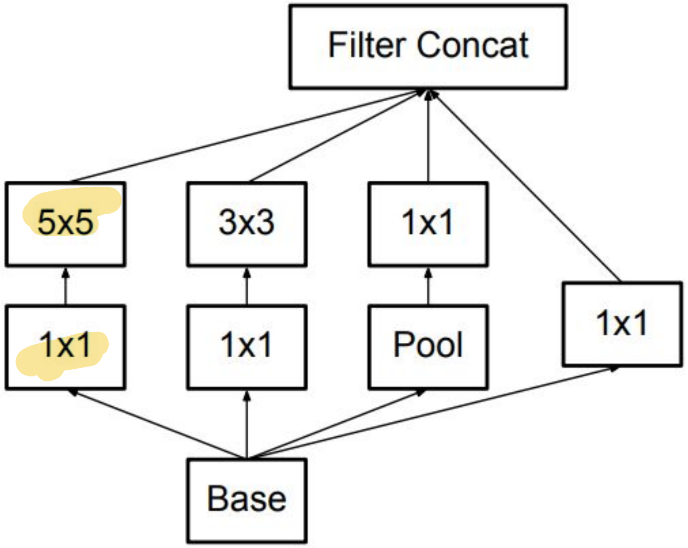
- Inception Net has been refined and optimized, leading to several smaller and faster variants such as Inception-v2, Inception-v3, and Inception-ResNet.
- Inception-ResNet incorporates residual connections into the Inception modules to further improve training stability and performance.

Rethinking the Inception Architecture



Mini-network replacing the 5×5 convolutions.

Rethinking the Inception Architecture



4. Original Inception module as described in [20].

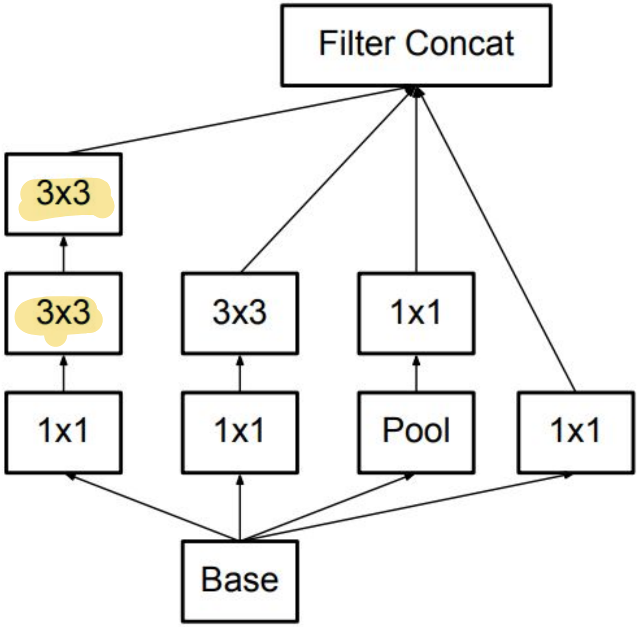


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution.

Rethinking the Inception Architecture

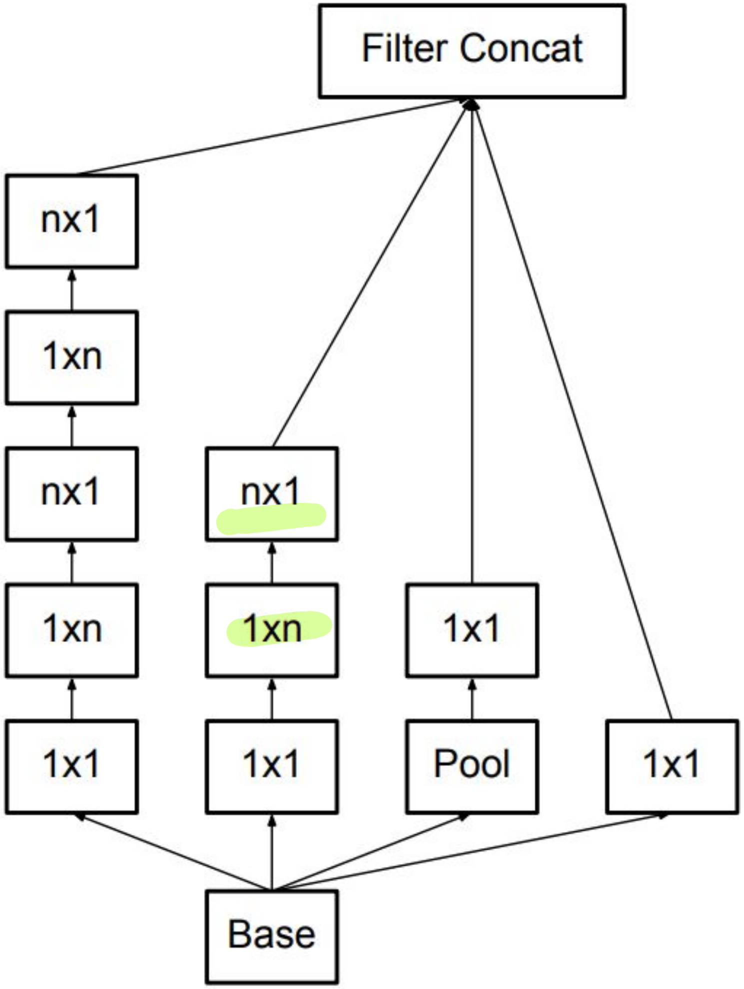


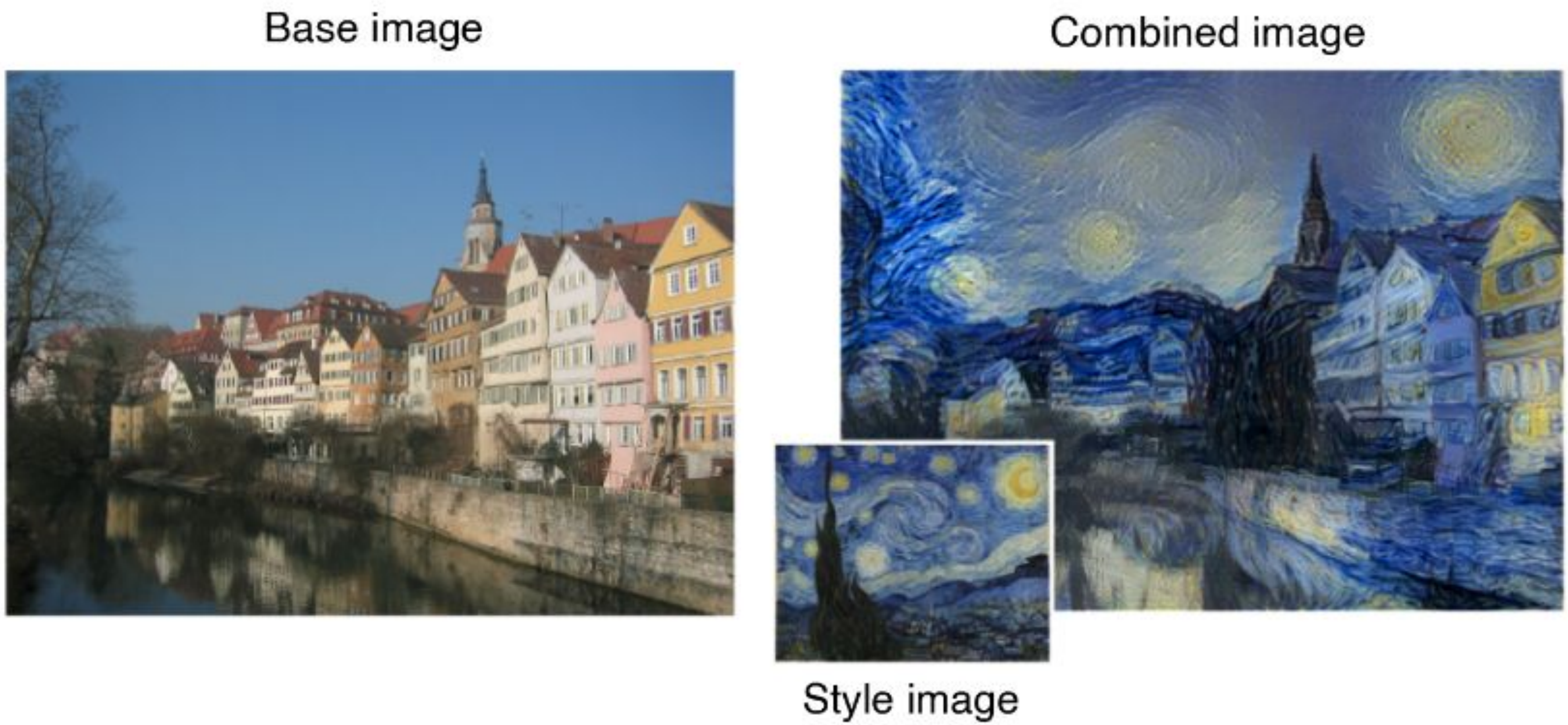
Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

InceptionNet Applications

- Image classification, Object Detection (fine-grained), semantic segmentation, and features extraction
- Image Quality Assessment for Inception Score.

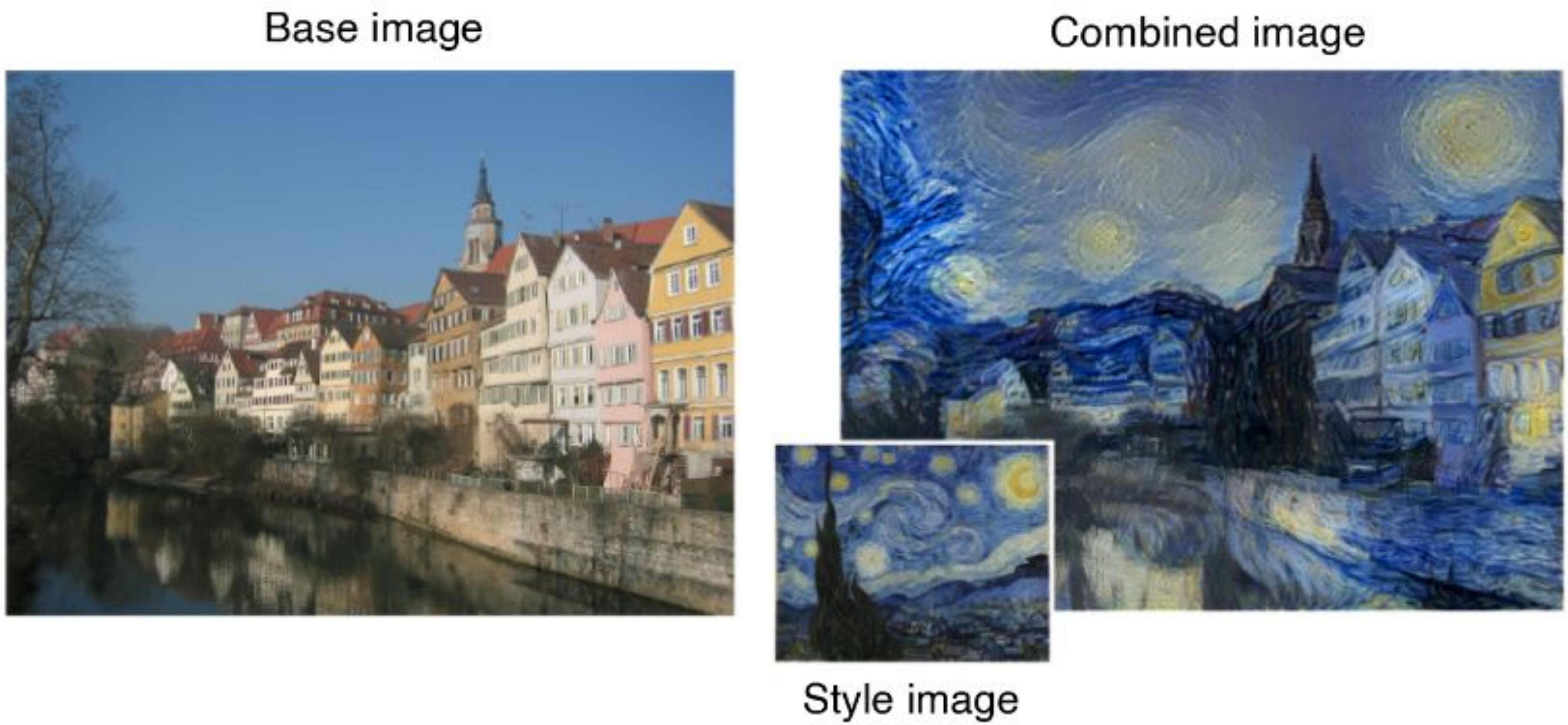
Neural Style Transfer

VGG for style transfer.



VGG v/s InceptionNet

Neural Style Transfer



- **<u>Puzzle</u>**: VGG is better feature extractor then InceptionNet for Style Transfer. The stylization performance degrades using InceptionNet instead of VGG.

Neural Style Transfer

Wang, Pei, Yijun Li, and Nuno Vasconcelos. "Rethinking and improving the robustness of image style transfer." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021.

- **Puzzle**: VGG is better feature extractor then InceptionNet for Style Transfer. The stylization performance degrades using InceptionNet instead of VGG.