Introduction to Computer Vision: Neural Networks, Image Classification, Semantic Segmentation

Navasardyan Shant

November 14, 2019

Overview

Overview

1. Image Classification Network Architectures

2. Semantic Segmentation

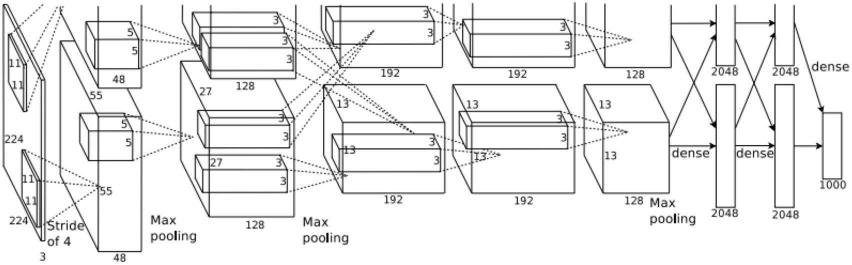


Figure: In the AlexNet architecture[1] after each convolution we have the **ReLU** activation, and after each of the first two convolutions we have **ReLU** activation, **local response normalization**. All Max-Pooling layers have the kernel size 3×3 and are done with strides 2×2. After each fully connected layer we have the **ReLU** activation and **dropout**.

[1] Imagenet classification with deep convolutional neural networks A Krizhevsky, I Sutskever, GE Hinton - Advances in neural information processing systems, 2012

Vgg

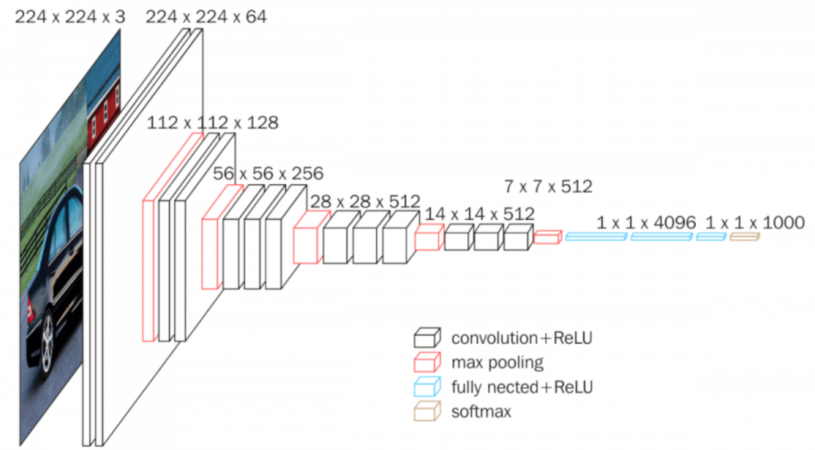


Figure: Vgg-16[2]

[2]Very deep convolutional networks for large-scale image recognition K Simonyan, A Zisserman - arXiv preprint arXiv:1409.1556, 2014

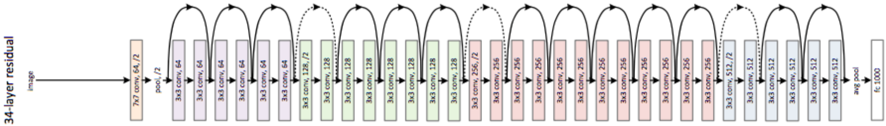


Figure: The ResNet architecture[3], dotted lines means decreasing-size residual blocks. There are two kinds of residual blocks: with and without 2-strided convolution block in the residual branch

[3] He, Kaiming et al. "Deep Residual Learning for Image Recognition." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 770-778.

ResNet

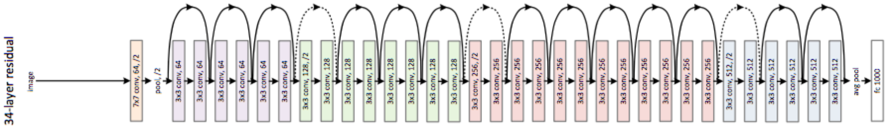


Figure: The ResNet architecture[3], dotted lines means decreasing-size residual blocks. There are two kinds of residual blocks: with and without 2-strided convolution block in the residual branch

Question

How to pass the output of a convolutional layer as the input of a dense layer? What about the cases when we want our neural network to deal with images of arbitrary size?

[3] He, Kaiming et al. "Deep Residual Learning for Image Recognition." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 770-778.

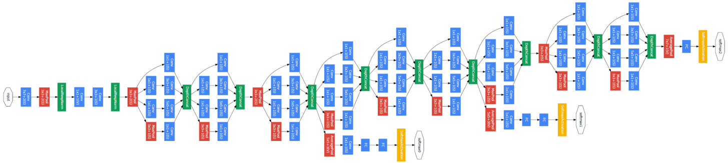


Figure: The GoogleNet architecture[4] and the Inception Module (below) used in GoogleNet

[4]Szegedy, Christian et al. "Going deeper with convolutions." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014): 1-9.

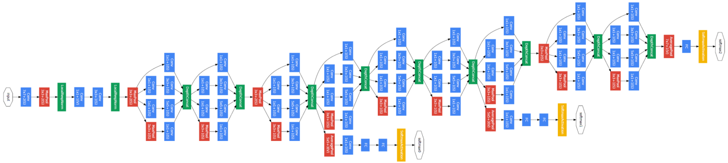
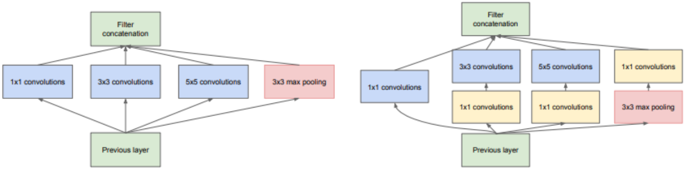


Figure: The GoogleNet architecture[4] and the Inception Module (below) used in GoogleNet



(a) Inception module, naïve version (b) Inception module with dimension reductions

[4] Szegedy, Christian et al. "Going deeper with convolutions." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014): 1-9.

Inception v2 and v3

The Inception v2[5] architecture simply is a slightly modified version of GoogleNet with **batch normalization** layers before each activation layer. In the Inception v3[6] architecture we are getting familiar with the idea of **factorizing** convolutions.

[5] Ioffe, Sergey and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." ArXiv abs/1502.03167 (2015): n. pag.

[6] Szegedy, Christian et al. "Rethinking the Inception Architecture for Computer Vision." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 2818-2826.

Inception v2 and v3

The Inception v2[5] architecture simply is a slightly modified version of GoogleNet with **batch normalization** layers before each activation layer. In the Inception v3[6] architecture we are getting familiar with the idea of **factorizing** convolutions. Before talking about the concept of convolution factorization, let's discuss a fundamental concept in deep learning, namely **receptive field of a network**.

[5] Ioffe, Sergey and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." ArXiv abs/1502.03167 (2015): n. pag.

[6] Szegedy, Christian et al. "Rethinking the Inception Architecture for Computer Vision." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 2818-2826.

Receptive Field

Let we have a convolutional neural network with consequent layers $F_0, F_1, \ldots, F_D$ (F_0, F_D are inputs and outputs of the network respectively).

Receptive Field

The **Receptive Field** of a layer F_k *with respect to* the layer F_m $(m < k)$ is the maximal region in the layer F_m each element of which is contributed in forming *one* of pixels in F_D, **considering only convolutional layers as contribution**.

Receptive Field

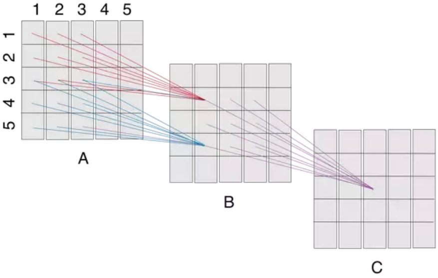
Let we have a convolutional neural network with consequent layers $F_0, F_1, \ldots, F_D$ (F_0, F_D are inputs and outputs of the network respectively).

Receptive Field

The **Receptive Field** of a layer F_k *with respect to* the layer F_m $(m < k)$ is the maximal region in the layer F_m each element of which is contributed in forming *one* of pixels in F_D, **considering only convolutional layers as contribution**. By **receptive field** of a network we mean the receptive field of its output with respect to its input.

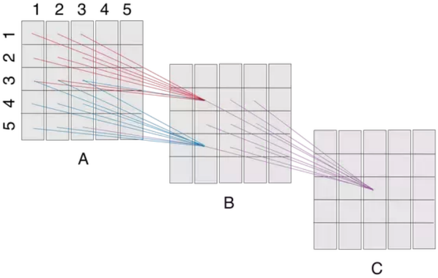
Receptive Field

Here is the illustration of the receptive field of the layer C with respect to the layer A. Here you can see that the whole layer A is this receptive field.



Receptive Field

Here is the illustration of the receptive field of the layer C with respect to the layer A. Here you can see that the whole layer A is this receptive field.



Note

Although the receptive field can be referred as the *"area under vision of the net"*, not all pixels in receptive field are *"equally contributed"* in formation of an output pixel. So there is a concept of **effective receptive field**[a].

[a] Yu, Fisher and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions." CoRR abs/1511.07122 (2015): n. pag.

Inception-V3 Revisited

So the natural need arises to enlarge the size of the receptive field of the network, or keep the size of receptive field the same but decrease the number of computations. For the latter purpose the approach of *convolution factorization* is appropriate.

Inception-V3 Revisited

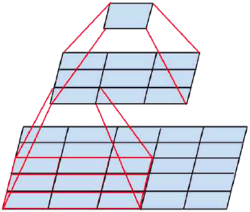
So the natural need arises to enlarge the size of the receptive field of the network, or keep the size of receptive field the same but decrease the number of computations. For the latter purpose the approach of *convolution factorization* is appropriate.

The concept of convolution factorization was introduced in the *Inception-V3* architecture. The idea is the following: in some places

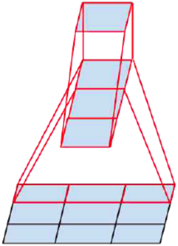
- replace 5×5 convolution layers with two 3×3 convolution layers
- replace $n \times n$ convolution layers with asymmetric consequent convolution layers of sizes $n \times 1$ and $1 \times n$.

Below you can see visualizations of these convolution factorizations.

Inception-V3: Convolution Factorization

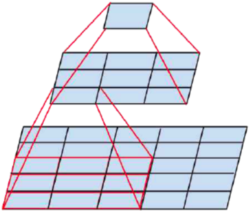


Two 3×3 convolutions replacing one 5×5 convolution

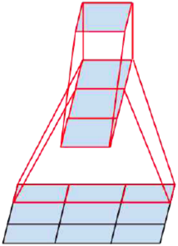


One 3×1 convolution followed by one 1×3 convolution replaces one 3×3 convolution

Inception-V3: Convolution Factorization



Two 3×3 convolutions replacing one 5×5 convolution



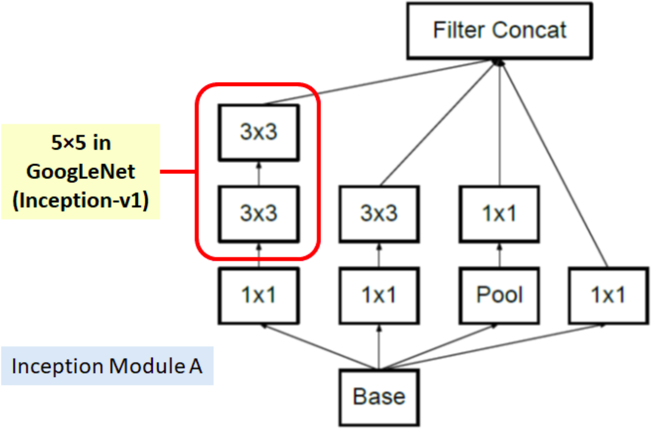
One 3×1 convolution followed by one 1×3 convolution replaces one 3×3 convolution

You can see that the receptive fields of replaced blocks are the same as before replacement.

Inception-V3: Convolution Factorization

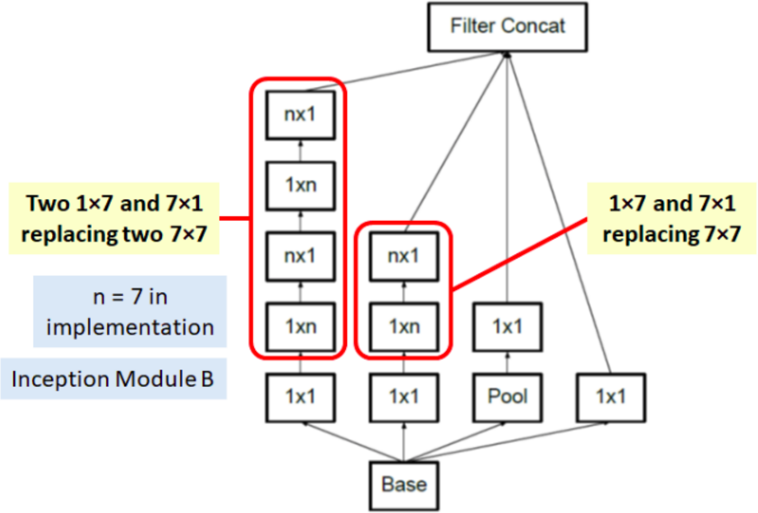
So the following types of inception modules are introduced in the architecture of Inception-V3.

So the following types of inception modules are introduced in the architecture of Inception-V3.



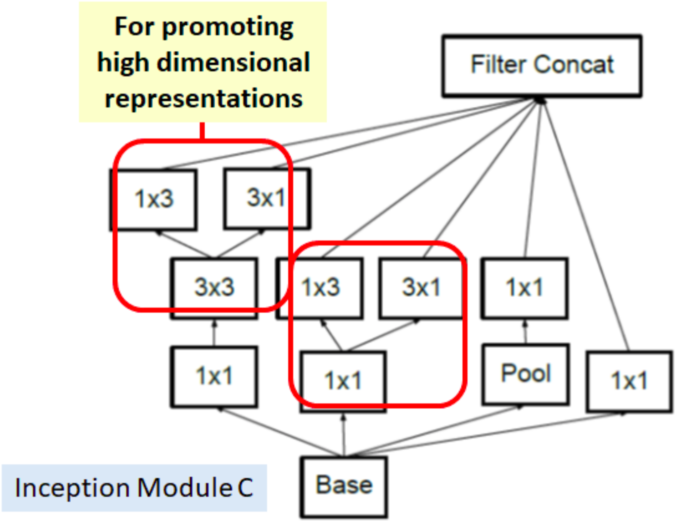
Inception Module A using factorization

Inception-V3: Convolution Factorization



Inception Module B using asymmetric factorization

Inception-V3: Convolution Factorization



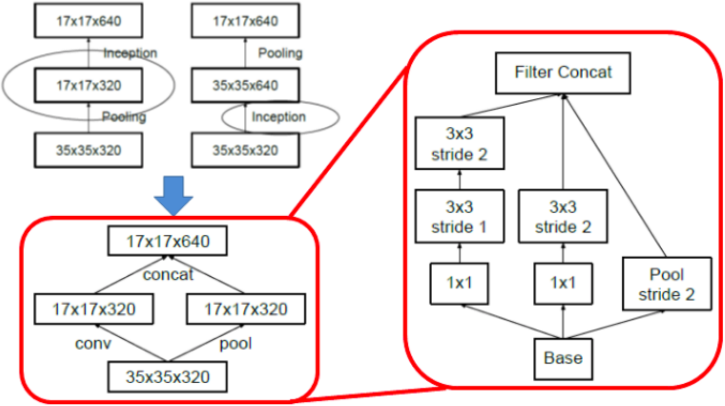
Inception Module C using asymmetric factorization

Inception-V3: Grid Size Reduction

The architecture of Inception-V3 also introduces a new size reduction method, directly in the inception block.

Inception-V3: Grid Size Reduction

The architecture of Inception-V3 also introduces a new size reduction method, directly in the inception block.



Conventional downsizing (Top Left), Efficient Grid Size Reduction (Bottom Left), Detailed Architecture of Efficient Grid Size Reduction (Right)

Inception-V3

So the whole architecture of the network Inception-V3 is the following:

So the whole architecture of the network Inception-V3 is the following:

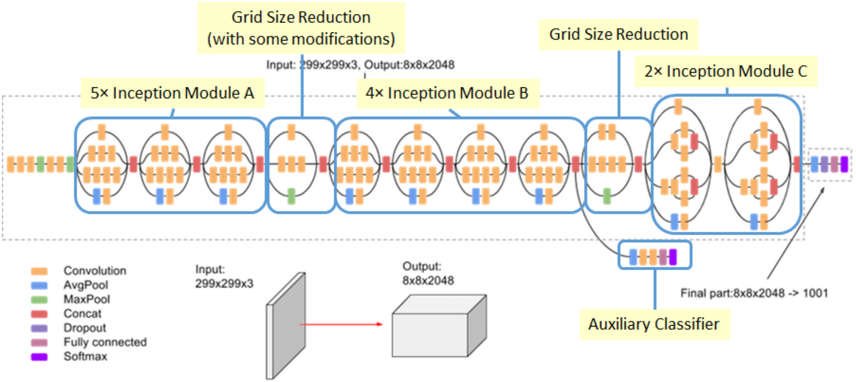


Figure: After each convolution layer we use BatchNorm and ReLU

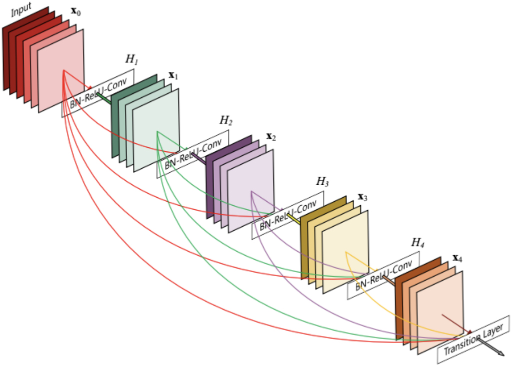
DenseNet

Another architecture similar to ResNet architecture is introdused as *DenseNet*[7]. Here you can see the main block of this network, called *dense block*.

[7] Huang, Gao et al. "Densely Connected Convolutional Networks." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 2261-2269.

DenseNet

Another architecture similar to ResNet architecture is introdused as *DenseNet*[7]. Here you can see the main block of this network, called *dense block*.



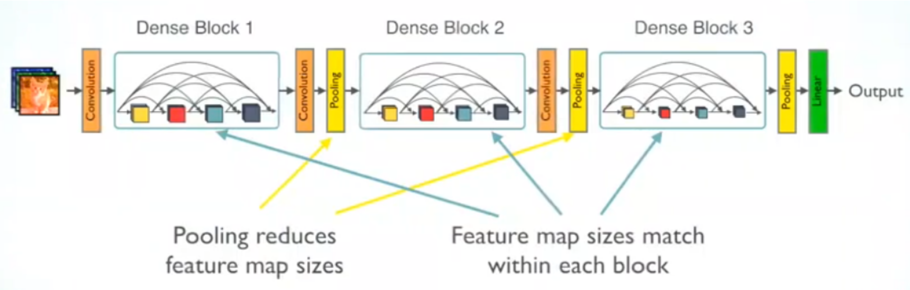
[7] Huang, Gao et al. "Densely Connected Convolutional Networks." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 2261-2269.

DenseNet

The subsampling in this network is done in so called *transition* layer, which is convolution, followed by average pooling.

DenseNet

The subsampling in this network is done in so called *transition* layer, which is convolution, followed by average pooling.



SqueezeNet

The model SqueezeNet[8] aims to reduce the model size and computation complexity while preserving the high accuracy. As described in the paper, there are several strategies to achieve a high accuracy with a small model:

[8] Iandola, Forrest N. et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡1MB model size." ArXiv abs/1602.07360 (2017): n. pag.

SqueezeNet

The model SqueezeNet[8] aims to reduce the model size and computation complexity while preserving the high accuracy. As described in the paper, there are several strategies to achieve a high accuracy with a small model:

- use more 1×1 convolutions intead of 3×3 convolutions

[8] Iandola, Forrest N. et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡1MB model size.” ArXiv abs/1602.07360 (2017): n. pag.

SqueezeNet

The model SqueezeNet[8] aims to reduce the model size and computation complexity while preserving the high accuracy. As described in the paper, there are several strategies to achieve a high accuracy with a small model:

- use more 1×1 convolutions intead of 3×3 convolutions
- in case of 3×3 convolutions, decrease the number of its input channels

[8] Iandola, Forrest N. et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡1MB model size." ArXiv abs/1602.07360 (2017): n. pag.

SqueezeNet

The model SqueezeNet[8] aims to reduce the model size and computation complexity while preserving the high accuracy. As described in the paper, there are several strategies to achieve a high accuracy with a small model:

- use more 1×1 convolutions intead of 3×3 convolutions
- in case of 3×3 convolutions, decrease the number of its input channels
- downsample in deeper features instead of earlier features.

[8] Iandola, Forrest N. et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡1MB model size." ArXiv abs/1602.07360 (2017): n. pag.

SqueezeNet

So the SqueezeNet architecture introduces the module called **fire module**:

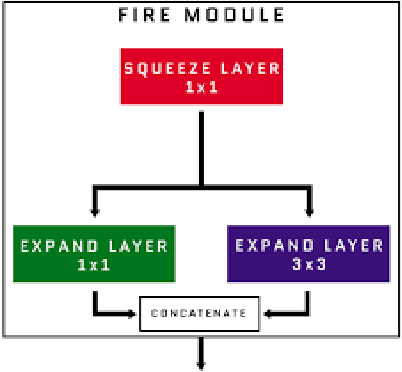
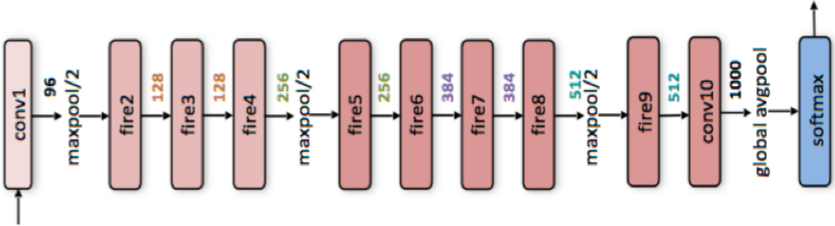


Figure: This *fire module* has three parameters. $s_{1\times1}$ is the number of the filters of the 1×1 convolution in the *"Squeeze Layer"*, $e_{1\times1}$ and $e_{3\times3}$ are the numbers of the filters of the 1×1 and 3×3 convolutions in the *"Expand Layers"*. Each layer consists of a convolution followed by *ReLU* activation.

SqueezeNet

So the SqueezeNet architecture is the following:



MobileNet

Another small-sized network architecture is described in *MobileNet*[9]. This is a bunch of neural networks similar to each other, so we refer to all of them as *MobileNet*.

[9] Howard, Andrew G. et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." ArXiv abs/1704.04861 (2017): n. pag.

MobileNet

Another small-sized network architecture is described in *MobileNet*[9]. This is a bunch of neural networks similar to each other, so we refer to all of them as *MobileNet*.

Depthwise Separable Convolutions

In the *MobileNet* architecture the core idea is the concept of **Depthwise Separable Convolution** block. This block consists of two convolutional blocks: **depthwise** block and **pointwise** block.

[9] Howard, Andrew G. et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." ArXiv abs/1704.04861 (2017): n. pag.

MobileNet

Another small-sized network architecture is described in *MobileNet*[9]. This is a bunch of neural networks similar to each other, so we refer to all of them as *MobileNet*.

Depthwise Separable Convolutions

In the *MobileNet* architecture the core idea is the concept of **Depthwise Separable Convolution** block. This block consists of two convolutional blocks: **depthwise** block and **pointwise** block. The **depthwise** convolution applies a *single* filter to each input channel, so the output of depthwise convolution has the same number of channels as the input. The **depthwise** block consists of a depthwise confolution followed by *Batch Normalization* and *ReLU* activation.

[9] Howard, Andrew G. et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." ArXiv abs/1704.04861 (2017): n. pag.

MobileNet

Another small-sized network architecture is described in *MobileNet*[9]. This is a bunch of neural networks similar to each other, so we refer to all of them as *MobileNet*.

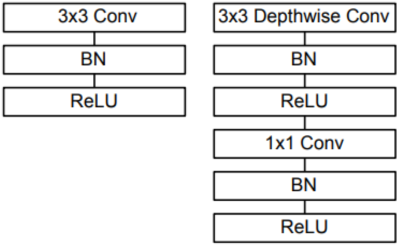
Depthwise Separable Convolutions

In the *MobileNet* architecture the core idea is the concept of **Depthwise Separable Convolution** block. This block consists of two convolutional blocks: **depthwise** block and **pointwise** block. The **depthwise** convolution applies a *single* filter to each input channel, so the output of depthwise convolution has the same number of channels as the input. The **depthwise** block consists of a depthwise confolution followed by *Batch Normalization* and *ReLU* activation. The **pointwise** convolution is just a 1×1 convolution. The **pointwise** block consists of a pointwise convolution followed by *Batch Normalization* and *ReLU* activation.

[9] Howard, Andrew G. et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications." ArXiv abs/1704.04861 (2017): n. pag.

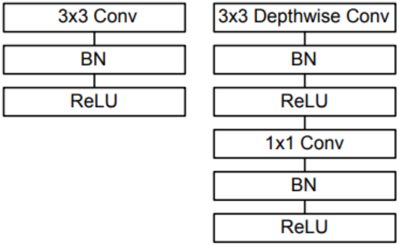
MobileNet

Left: standard convolutional block. Right: depthwise separable convolutional block



MobileNet

Left: standard convolutional block. Right: depthwise separable convolutional block



Width Multiplier

The number of filters of the pointwise 1×1 convolutions can be controlled with a hyperparameter called **width multiplier**, which is denoted by α. If in the baseline *MobileNet* architecture we have 1×1 convolution layers with numbers of filters $C_1, C_2, C_3, \ldots$, then in the $Mobilenet - \alpha$ architecture we have these numbers multiplied by α, i.e. the the numbers of filters of 1×1 convolutions are $\alpha C_1, \alpha C_2, \alpha C_3, \ldots$.

Overview

Semantic Segmentation

Semantic Segmentation

As previously, we refer to the image segmentation problem as a partitioning the image into regions (not necessarily connected).

Semantic Segmentation

Semantic Segmentation

As previously, we refer to the image segmentation problem as a partitioning the image into regions (not necessarily connected). The task of **semantic image segmentation** is in addition of partitioning the image also label the formed regions. The word *semantic* means that we desire to label these regions *semantically*, i.e. in such a way, that we, humans, understand objects we see.

Semantic Segmentation

Semantic Segmentation

As previously, we refer to the image segmentation problem as a partitioning the image into regions (not necessarily connected). The task of **semantic image segmentation** is in addition of partitioning the image also label the formed regions. The word *semantic* means that we desire to label these regions *semantically*, i.e. in such a way, that we, humans, understand objects we see. So, as the task of semantic segmentation is to get some semantic partition of the image and label the formed regions, we can combine these two steps into **classification of each pixel in the image semantically**.

Semantic Segmentation

Semantic Segmentation

As previously, we refer to the image segmentation problem as a partitioning the image into regions (not necessarily connected). The task of **semantic image segmentation** is in addition of partitioning the image also label the formed regions. The word *semantic* means that we desire to label these regions *semantically*, i.e. in such a way, that we, humans, understand objects we see. So, as the task of semantic segmentation is to get some semantic partition of the image and label the formed regions, we can combine these two steps into **classification of each pixel in the image semantically**.

Therefore, **semantic image segmentation in essence is a classification of each pixel in the image**.

Semantic Segmentation

Note

As the semantic segmentation task is a classification task, the main objective we want to minimize remains **categorical cross-entropy** loss, averaged among all pixels.

Semantic Segmentation

Note

As the semantic segmentation task is a classification task, the main objective we want to minimize remains **categorical cross-entropy** loss, averaged among all pixels. I.e. if we have an image $I \in \mathbb{R}^{H \times W \times 3}$ and a prediction $P \in \mathbb{R}^{H \times W \times K}$ (here K is the number of classes), then the loss is computed as

$$L(I, P) = -\frac{1}{H \cdot W} \sum_{i,j} \sum_{k=1}^{K} GT_{i,j,k} \log(P_{i,j,k}),$$

where $GT \in \{0, 1\}^{H \times W \times K}$ is one-hot encoded tensor of *ground truth* pixel classes of the image I.

Semantic Segmentation

Note

As the semantic segmentation task is a classification task, the main objective we want to minimize remains **categorical cross-entropy** loss, averaged among all pixels. I.e. if we have an image $I \in \mathbb{R}^{H \times W \times 3}$ and a prediction $P \in \mathbb{R}^{H \times W \times K}$ (here K is the number of classes), then the loss is computed as

$$L(I, P) = -\frac{1}{H \cdot W} \sum_{i,j} \sum_{k=1}^{K} GT_{i,j,k} \log(P_{i,j,k}),$$

where $GT \in \{0, 1\}^{H \times W \times K}$ is one-hot encoded tensor of *ground truth* pixel classes of the image I.

Note

There are also other losses for semantic segmentation about which we will discuss later in this course.

The Fully Convolutional Network

We start the investigation of semantic segmentation algorithms with **FCNs - Fully Convolutional Networks**[10]

[10] Shelhamer, Evan et al. "Fully Convolutional Networks for Semantic Segmentation." IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2014): 640-651.

The Fully Convolutional Network

We start the investigation of semantic segmentation algorithms with **FCNs - Fully Convolutional Networks**[10]

FCN

Fully Convolutional Networks can take inputs of **arbitrary** size.

At first we adapt a classifier network to the fully convolutional one. For this we can refer to each dense layer as a convolution layer with kernel size as the size of the preceding layer. Or we can just throw away the part after the last convolutional layer.

[10] Shelhamer, Evan et al. "Fully Convolutional Networks for Semantic Segmentation." IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2014): 640-651.

The Fully Convolutional Network

We start the investigation of semantic segmentation algorithms with **FCNs - Fully Convolutional Networks**[10]

FCN

Fully Convolutional Networks can take inputs of **arbitrary** size.

At first we adapt a classifier network to the fully convolutional one. For this we can refer to each dense layer as a convolution layer with kernel size as the size of the preceding layer. Or we can just throw away the part after the last convolutional layer. In both cases the question arises how to **upsample** the resulting tensor to the input image size. This can be done, for example, with bilinear upsampling. Also a method called **deconvolution or transposed convolution** is used. The latter enables us also to train these upsampling parts of the network. We will talk about transposed convolutions later.

[10] Shelhamer, Evan et al. "Fully Convolutional Networks for Semantic Segmentation." IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2014): 640-651.

The Fully Convolutional Network

As just upsampling the last convolution layer's output gives unsatisfying coarse results, we can make so called **skip-connections** from the "*finer*" layers to the "*coarser*" layers. These skip-connections are illustrated in the image below

The Fully Convolutional Network

As just upsampling the last convolution layer's output gives unsatisfying coarse results, we can make so called **skip-connections** from the "*finer*" layers to the "*coarser*" layers. These skip-connections are illustrated in the image below

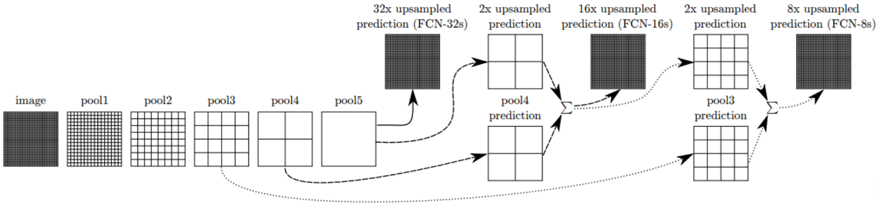


Figure: Pools are the pooling layers of some fully convolutional network, adapted from a classification network

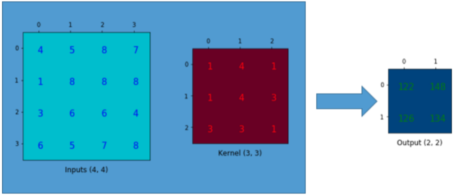
Transposed Convolutions: The Convolutional Matrix

Any convolution operation can be refered as a fully-connected layer with a (sparse) weight matrix.
Question: How?

Transposed Convolutions: The Convolutional Matrix

Any convolution operation can be refered as a fully-connected layer with a (sparse) weight matrix.
Question: How? *Answer:* By using the concept of the *convolution matrix*, which is described below.

Let's investigate a simple case, when we have a tensor X of shape 4×1 and we want to convolve it by the filter ϕ of shape $3 \times 3 \times 1$ (without padding and strides), the output, let's denote it by X', will be of shape $2 \times 2 \times 1$ as shown in the figure below:



Transposed Convolutions: The Convolution Matrix

Flattening

Let $A = (a_{ij})^{H \times W}$ be a real-valued matrix. We call the vector

$$A^f = (a_{11}\ a_{12}\ \ldots\ a_{1W}\ a_{21}\ a_{22}\ \ldots\ a_{2W}\ \ldots\ a_{HW})$$

the flattened vector of A. The operation $A \mapsto A^f$ we call the flattening operation.

Flattening

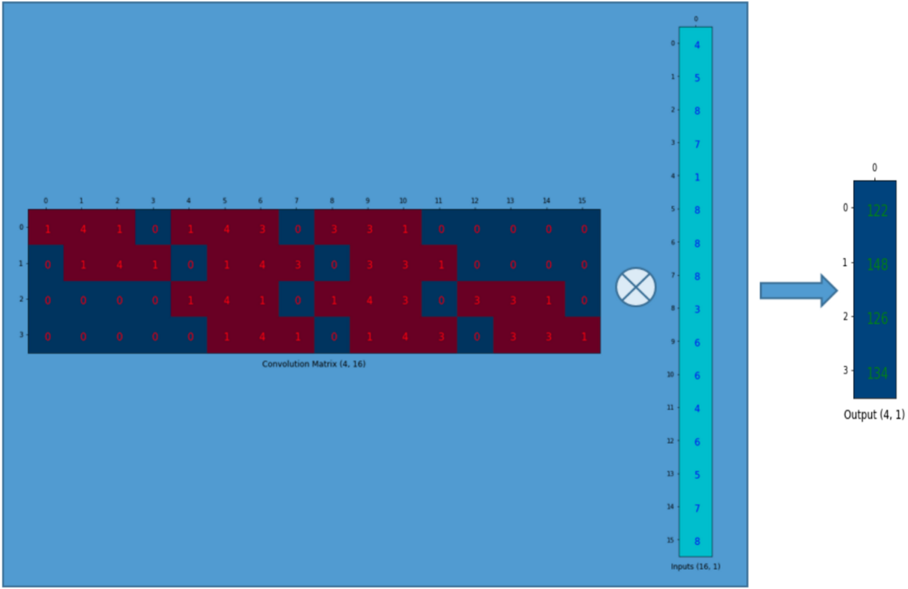
Let $A = (a_{ij})^{H \times W}$ be a real-valued matrix. We call the vector

$$A^f = (a_{11}\ a_{12}\ \dots\ a_{1W}\ a_{21}\ a_{22}\ \dots\ a_{2W}\ \dots\ a_{HW})$$

the flattened vector of A. The operation $A \mapsto A^f$ we call the flattening operation.

So in the above considered case we can obtain the matrix X' firstly by obtaining its flattened version X'^f then reshape it to the shape $2 \times 2 \times 1$. We can obtain the vector X'^f by the following way:

Transposed Convolutions: The Convolution Matrix



Transposed Convolutions: The Convolution Matrix

As you can see, the red part of this matrix is composed of the elements of the convolution kernel. We call this matrix **the convolution matrix** of the convolution operation. Let's denote the convolution matri by $Conv(X, \phi)$.

Transposed Convolutions: The Convolution Matrix

As you can see, the red part of this matrix is composed of the elements of the convolution kernel. We call this matrix **the convolution matrix** of the convolution operation. Let's denote the convolution matri by $Conv(X, \phi)$.

Exercise

Formalize the concept of the convolution matrix in case of arbitrary sizes of input, kernel and the convolution type (padding, strides).

Transposed Convolutions: The Convolution Matrix

Note

We have consider the case when input tensor is one-channeled. If it has C channels, we can obtain the result of the convolution by applying per-channel convolutions, then sum the obtained C matrices (and repeat this process C' times, when C' is the number of output filters of the convolution operation)

Transposed Convolutions: The Convolution Matrix

Note

We have consider the case when input tensor is one-channeled. If it has C channels, we can obtain the result of the convolution by applying per-channel convolutions, then sum the obtained C matrices (and repeat this process C' times, when C' is the number of output filters of the convolution operation)

Exercise

Define the concept of the convolution matrix in the general case.

Transposed Convolution

Sometimes we need to do the backward operation of the convolution operation. For this we introduce the concept of transposed convolution.

Transposed Convolution

Sometimes we need to do the backward operation of the convolution operation. For this we introduce the concept of transposed convolution. Let $X \in \mathbb{R}^{H \times W \times 1}$ be a tensor, and after convolving it by a kernel $\phi \in \mathbb{R}^{h \times w \times 1}$, we have got the tensor $X' \in \mathbb{R}^{H' \times W' \times 1}$. Then, we have

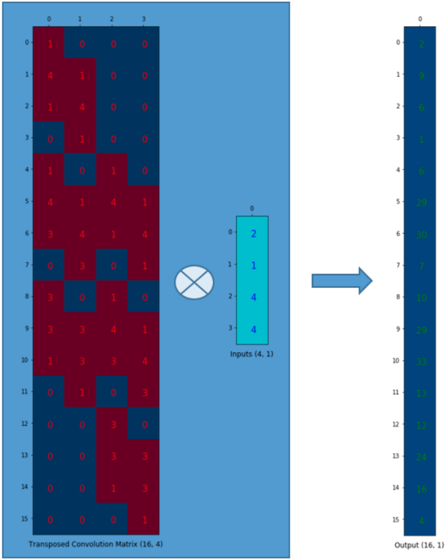
$$Conv(X, \phi)X^f = X'^f.$$

Hence we get

$$X^f = Conv(X, \phi)^T X'^f.$$

In the image below you can see the illustration of this for our example:

Transposed Convolution



Transposed Convolution

Definition (Transposed Convolution)

The mapping

$$TrConv : \mathbb{R}^{H' \times W' \times 1} \to \mathbb{R}^{H \times W \times 1} \qquad X' \mapsto Reshape(Conv(X, \phi)^T X'^f),$$

is called a **transposed convolution** with the kernel ϕ.

Transposed Convolution

Definition (Transposed Convolution)

The mapping

$$TrConv : \mathbb{R}^{H' \times W' \times 1} \to \mathbb{R}^{H \times W \times 1} \qquad X' \mapsto Reshape(Conv(X, \phi)^T X'^f),$$

is called a **transposed convolution** with the kernel ϕ.

Note

It seems, that the matrix $Conv(X, \phi)$ depends on the tensor X, but, actually, it only depends on ϕ, the shape of X, the padding and the stride of the convolution. So, the transposed convolution is **well defined** if the kernel, padding and stride of the convolution is given (the shape of X can be obtained from the padding, the strides and the shape of X').

Transposed Convolution

Note

As in case of convolutions, if the input of a transposed convolution operation has C channels, we apply the operation per-channel then sum the resulting 1-channeled outputs.

Transposed Convolution

Note

As in case of convolutions, if the input of a transposed convolution operation has C channels, we apply the operation per-channel then sum the resulting 1-channeled outputs.

Exercise

Determine the output shape of transposed convolution with kernel size k, padding p (from each size) and strides (s, s).

Dilated Convolutions

RefiNet

PSPNet

UNET

Introduction to Computer Vision: Neural Networks, Image Classification, Semantic Segmentation

Navasardyan Shant

November 14, 2019