



CMP9137M Advanced Machine Learning


Lecture 5: Regularization in CNNs and CNN based downstream tasks

<https://attendance.lincoln.ac.uk>

Access Code: 408773

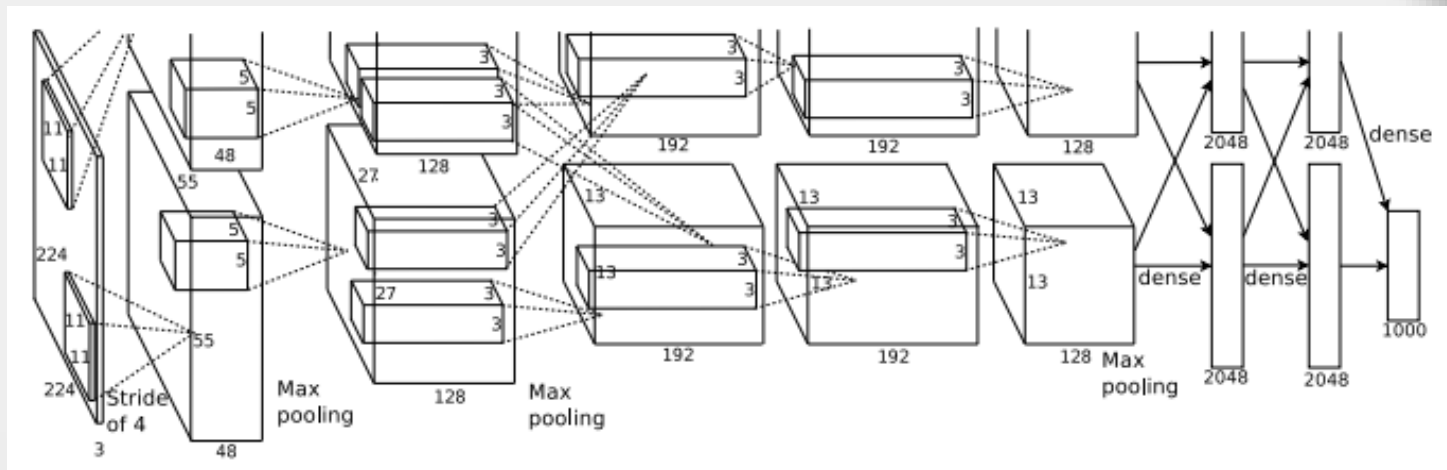
School of Computer Science

Laboratory of Vision Engineering

 Dr. Lei Zhang

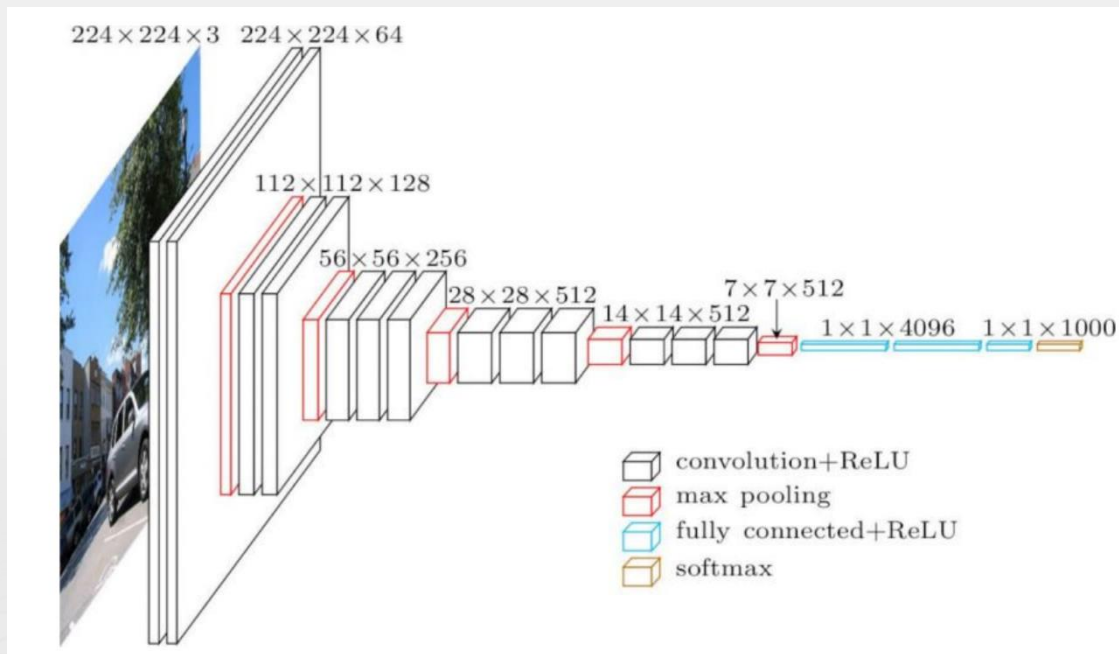
 lzhang@lincoln.ac.uk

CNN architecture-AlexNet in 2012- Recap



- It won the ImageNet challenge in 2012.
- AlexNet is considered to be the first CNN architecture which rose the interest in CNNs.
- The net contains eight layers with weights; the first five are convolutional and the remaining three are fullyconnected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.

CNN architecture-VGG16 2014- Recap



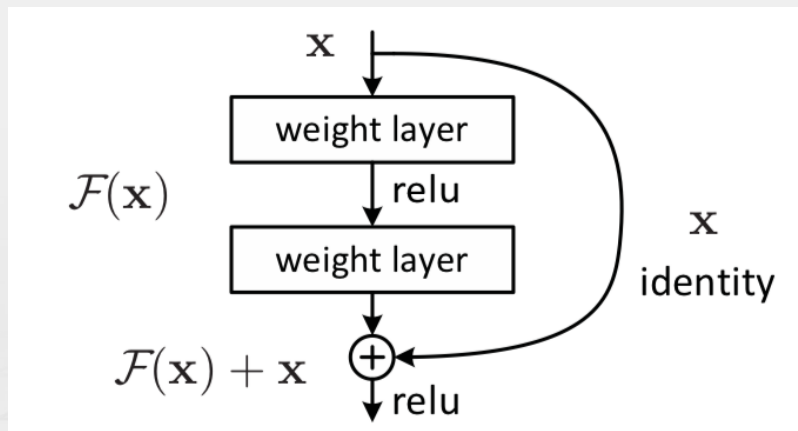
The main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to **16–19** weight layers

CNN architecture-ResNet in 2015



In this paper, they address the degradation problem by introducing a **deep residual learning framework**.

Instead of hoping every few stacked layers directly fit a desired underlying mapping, they explicitly let these layers fit a **residual mapping**.



The shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers (Fig. 1).

Fig1. Residual learning: a building block

CNN architecture-VGG19 and ResNet34- Recap

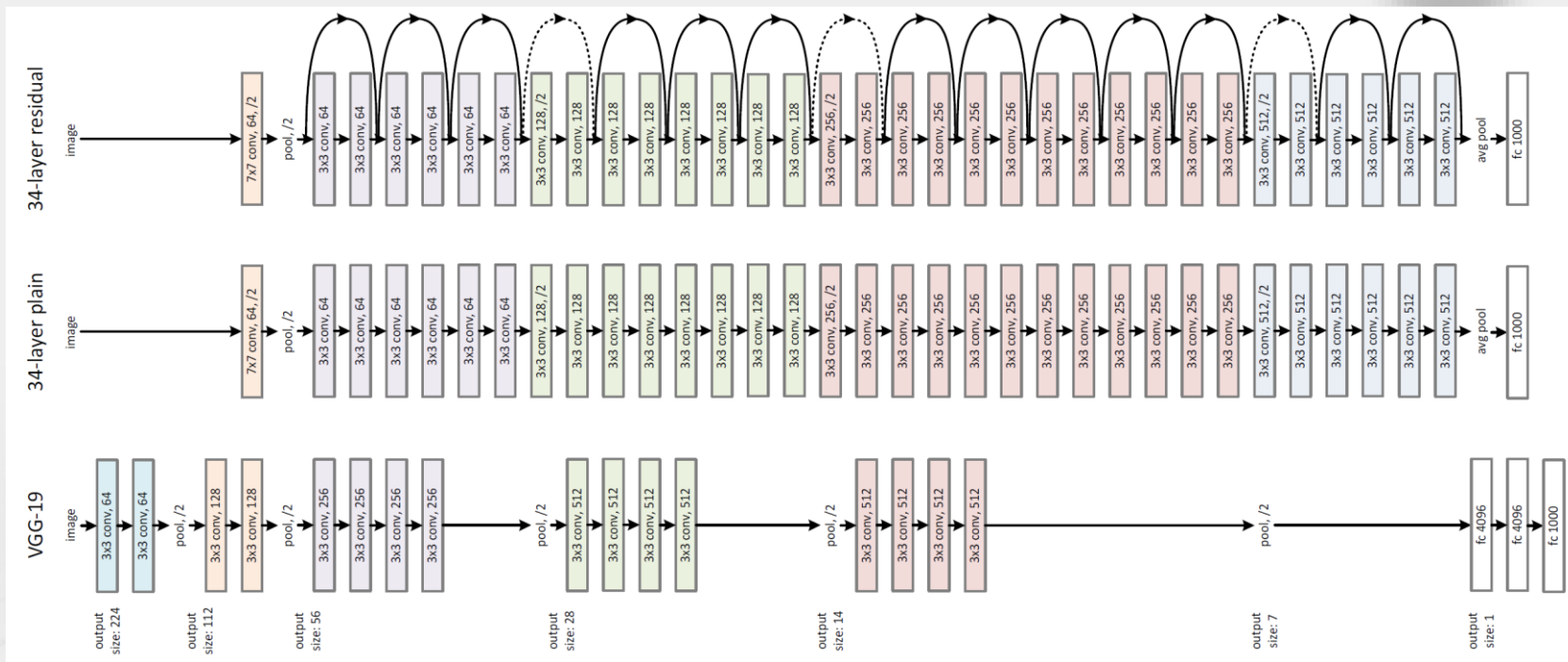


Figure 1. Example network architectures for ImageNet. Left: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions.



Objective

To understand the overfitting and underfitting issue in the ML

To know the techniques to mitigate the learning problem and accelerate the training

To know CNN based Classic downstream tasks/applications



Outline

Covering:

- Overfitting and Underfitting in Deep Learning Neural Networks
(Overfitting, Underfitting, Bias, variance)
- Regularization and transfer learning in deep CNNs
(L1 and L2 Regularization, dropout, batch normalization, transfer learning)
- Classic downstream tasks using CNN
(object detection: R-CNN, SPP-net, fast R-CNN, faster R-CNN, YOLO, SSD. Semantic segmentation: FCN, encoder-decoder architectures)



Outline

Covering:

- **Overfitting and Underfitting in Deep Learning Neural Networks**
(Overfitting, Underfitting, Bias, variance)
- Regularization and transfer learning in deep CNNs
(L1 and L2 Regularization, dropout, batch normalization, transfer learning)
- Classic downstream tasks using CNN
(object detection: R-CNN, SPP-net, fast R-CNN, faster R-CNN, YOLO, SSD. Semantic segmentation: FCN, encoder-decoder architectures)

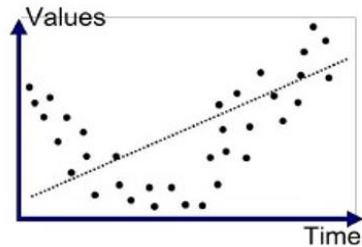
Overfitting and Underfitting in Deep Learning Neural Networks

“

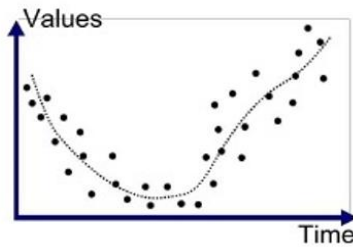
The central challenge in machine learning is that we must perform well on new, previously unseen inputs — not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called generalization.

”

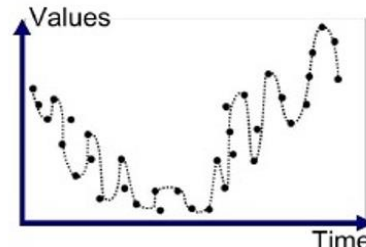
- Goodfellow, I. et al, (2016) Deep Learning, The MIT Press, page 110.



Underfitted

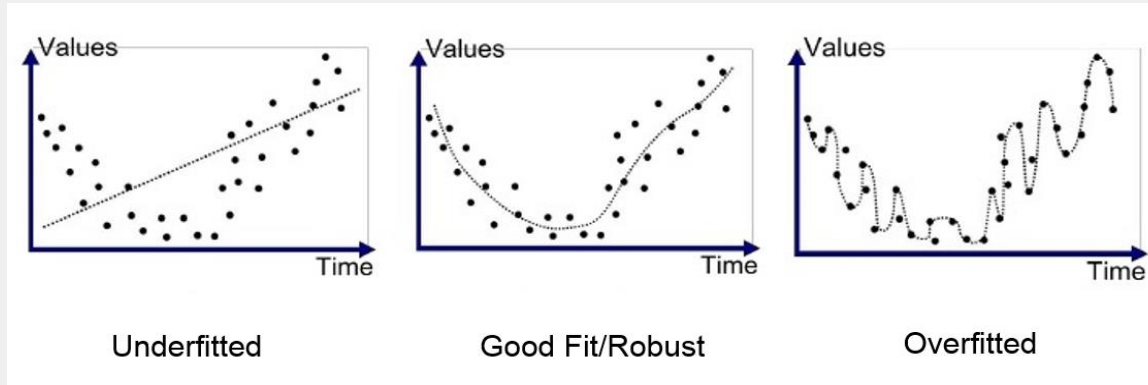


Good Fit/Robust



Overfitted

Overfitting and Underfitting in Deep Learning Neural Networks

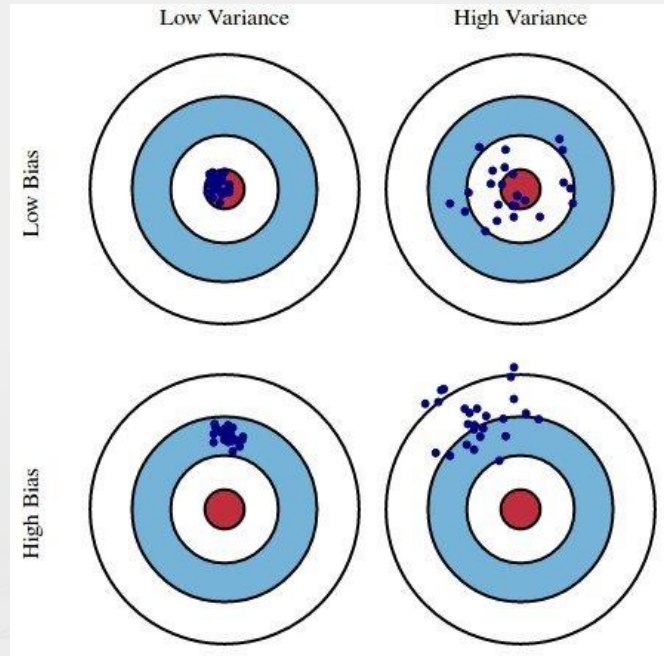


Underfitted model: A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on testing samples.

Good fit model: A model that suitably learns the training dataset and generalizes well to the test dataset.

Overfit Model: A model that learns the training dataset too well, performing well on the training dataset but does not perform well on a hold out sample.

Overfitting and Underfitting: Bias and Variance

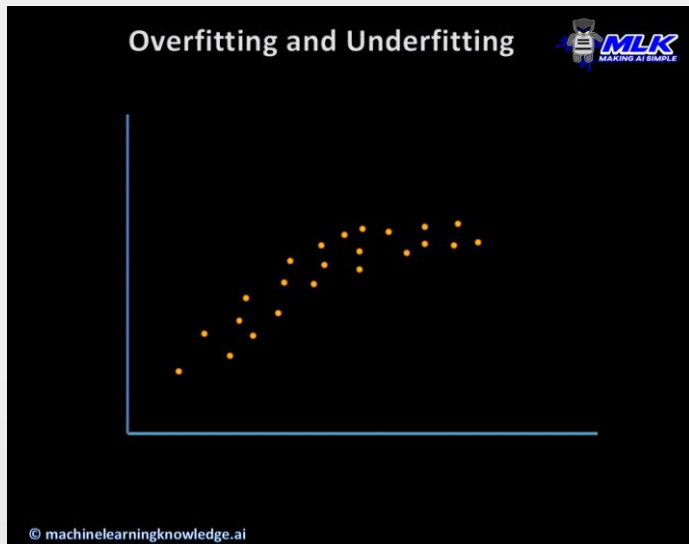


Two major sources of error in machine learning: bias and variance.

- The algorithm's error rate on the training set is informally defined as **Bias**
- **Variance** indicates how much worse the algorithm does on the test (validation) set than the training set.

Developing good intuition about Bias and Variance will help you choose effective changes for your algorithm.

Overfitting and Underfitting: Examples of Bias and Variance



Consider our cat classification task.

Suppose your algorithm performs as follows:

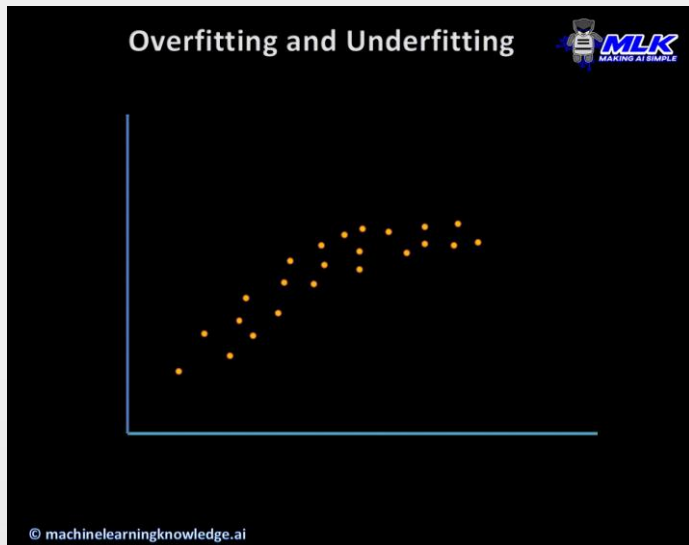
- Training error = 1%
- Validation error = 11%

What are Bias and Variance in this case?

- The **bias** is 1%
- The **variance** is 10% ($=11\%-1\%$)

It has **high variance**. The classifier has very low training error, but it is failing to generalize to the validation set. This is also called **overfitting**.

Overfitting and Underfitting: Examples of Bias and Variance



Consider our cat classification task.

Suppose your algorithm performs as follows:

- Training error = 15%
- Validation error = 16%

What are Bias and Variance in this case?

- The **bias** is 15%
- The **variance** is 1% ($=16\%-15\%$)

This classifier therefore has **high bias** , but low variance. We say that this algorithm is **underfitting** .

Overfitting and Underfitting: Examples of Bias and Variance



Consider our cat classification task.

Suppose your algorithm performs as follows:

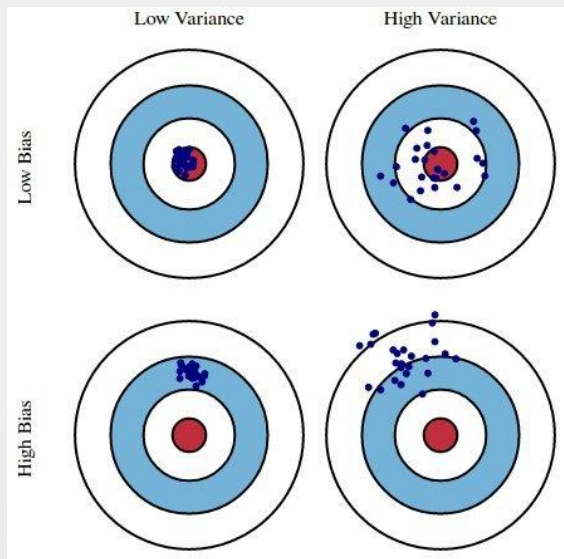
- Training error = 15%
- Validation error = 30%

What are Bias and Variance in this case?

- The **bias** is 15%
- The **variance** is 15% ($=30\%-15\%$)

The overfitting/underfitting terminology is hard to apply here since the classifier is simultaneously overfitting and underfitting.

Addressing the Overfitting and Underfitting in deep CNNs



- **Add more training data**
This is the simplest and most reliable way to address the variance
- **Modify model architecture**
(such as neural network architecture) so that it is more suitable for your problem: This technique can affect both bias and variance.
- **Increase the model size**
(such as number of neurons/layers)
- **Reduce or eliminate regularization**
(L2 regularization, L1 regularization, dropout)

Increasing the model size generally reduces bias, but it might also increase variance and the risk of overfitting. However, this overfitting problem usually arises only when you are not using **regularization**.



Outline

Covering:

- Overfitting and Underfitting in Deep Learning Neural Networks
(Overfitting, Underfitting, Bias, variance)
- **Regularization and transfer learning in deep CNNs**
(**L1 and L2 Regularization, dropout, batch normalization, transfer learning**)
- Classic downstream tasks using CNN
(object detection: R-CNN, SPP-net, fast R-CNN, faster R-CNN, YOLO, SSD. Semantic segmentation: FCN, encoder-decoder architectures)

Regularization in deep CNNs- L1 and L2 Regularization

$$Loss = Error(y, \hat{y})$$

Regularisation is a process of introducing additional information in order to prevent overfitting.

L1 and L2 regularisation owes its name to L1 and L2 norm

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N |w_i|$$

A linear regression model that implements L1 norm for regularisation is called **lasso regression**, and one that implements (squared) L2 norm for regularisation is called **ridge regression**.

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N w_i^2$$

Regularization in deep CNNs- L1 and L2 Regularization

$$Loss = Error(y, \hat{y})$$

Loss function with no regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N w_i^2$$

Loss function with L2 regularisation

A linear regression model that implements L1 norm for regularisation is called **lasso regression**, and one that implements (squared) L2 norm for regularisation is called **ridge regression**.

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “*absolute value of magnitude*” of coefficient as penalty term to the loss function.

Ridge regression adds “*squared magnitude*” of coefficient as penalty term to the loss function.

Regularization in deep CNNs- L1 and L2 Regularization



How is overfitting prevented with L1 and L2 Regularization?

Regularization in deep CNNs- L1 and L2 Regularization

Define a simple linear regression model

$$\hat{y} = wx + b$$

Loss function with no regularisation

$$\begin{aligned} L &= (\hat{y} - y)^2 \\ &= (wx + b - y)^2 \end{aligned}$$

Loss function with L1 regularisation

$$L_1 = (wx + b - y)^2 + \lambda|w|$$

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$$

Loss function with L2 regularisation

$$L_2 = (wx + b - y)^2 + \lambda w^2$$

Gradient Descent

Recall that updating the parameter w in gradient descent is as follows:

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w}$$

Loss function with no regularisation

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y)] \end{aligned}$$

Loss function with L1 regularisation

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_1}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y) + \lambda \frac{d|w|}{dw}] \\ &= \begin{cases} w - \eta \cdot [2x(wx + b - y) + \lambda] & w > 0 \\ w - \eta \cdot [2x(wx + b - y) - \lambda] & w < 0 \end{cases} \end{aligned}$$

Loss function with L2 regularisation

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_2}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y) + 2\lambda w] \end{aligned}$$

Regularization in deep CNNs- L1 and L2 Regularization

Gradient Descent

Recall that updating the parameter w in gradient descent is as follows:

$$w_{\text{new}} = w - \eta \frac{\partial L}{\partial w}$$

Loss function with no regularisation

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y)] \end{aligned}$$

Loss function with L1 regularisation

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_1}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y) + \lambda \frac{d|w|}{dw}] \\ &= \begin{cases} w - \eta \cdot [2x(wx + b - y) + \lambda] & w > 0 \\ w - \eta \cdot [2x(wx + b - y) - \lambda] & w < 0 \end{cases} \end{aligned}$$

Loss function with L2 regularisation

$$\begin{aligned} w_{\text{new}} &= w - \eta \frac{\partial L_2}{\partial w} \\ &= w - \eta \cdot [2x(wx + b - y) + 2\lambda w] \end{aligned}$$

perform the following substitutions on the equations

$$\bullet \eta = 1,$$

$$\bullet H = 2x(wx + b - y)$$

$$w_{\text{new}} = w - H \quad \text{--- (0)}$$

Let's say with Equation 0, calculating $w - H$ gives us a w value that leads to overfitting.

$$w_{\text{new}} = \begin{cases} (w - H) - \lambda, & w > 0 & \text{--- (1.1)} \\ (w - H) + \lambda, & w < 0 & \text{--- (1.2)} \end{cases}$$

λ will reduce the chances of overfitting because introducing λ makes us shift *away* from the very w

$$w_{\text{new}} = (w - H) - 2\lambda w \quad \text{--- (2)}$$

Regularization in deep CNNs- Dropout

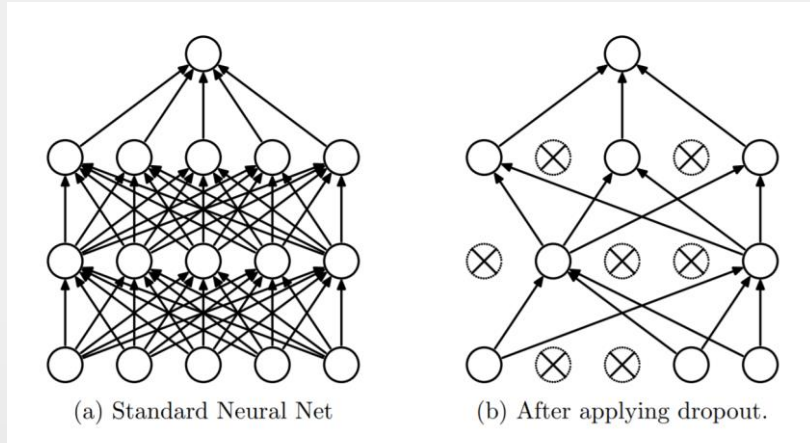


Fig. Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

- Dropouts are the regularization technique that is used to prevent overfitting in the model.
- Dropouts are added to randomly switching some percentage of neurons of the network.
- They are mostly used after the dense layers of the network. It is always good to only switch off the neurons to 50%.

Regularization in deep CNNs- Batch Normalization

$$z^N = \left(\frac{z - m_z}{s_z} \right)$$

m_z the mean of the neurons' output

s_z the standard deviation of the neurons' output.

- **Batch normalization is used to normalize the output of the previous layers.**

It is a layer that allows every layer of the network to do learning more independently and makes the network more stable during training.

- **The layer is added to the sequential model to standardize the input or the outputs.**
- **The Batch Norm has a regularization effect.**
Because it is computed over mini-batches and not the entire data set, the model's data distribution sees each time has some noise. This can act as a regularizer, which can help overcome overfitting and help learn better.

Regularization in deep CNNs- Dropout and BN

It is believed that the regularization effect of dropout in convolutional layers is mainly obtained from the robustness to noisy inputs.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1 (January 2014), 1929–1958.

The noise added using BN is quite small. Thus, it may not be enough to properly regularize on BN its own and is normally used along with **Dropout**.

“Removing Dropout from Modified *BN-Inception* speeds up training, without increasing overfitting.”

Sergey Ioffe, Christian Szegedy, (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift PMLR 37:448-456,

"While the CNN that does not use dropout achieved 83.16% accuracy, when dropout is applied to the output of every convolutional layer except the last conv4_3 layer with ratio of 0.1, the network achieved 87.78% accuracy."

Park S., Kwak N. (2017) Analysis on the Dropout Effect in Convolutional Neural Networks. In: Lai SH., Lepetit V., Nishino K., Sato Y. (eds) Computer Vision – ACCV 2016. ACCV 2016. Lecture Notes in Computer Science, vol 10112. Springer, Cham. https://doi.org/10.1007/978-3-319-54184-6_12

Batch normalization also sometimes reduces generalization error and allows dropout to be omitted, due to the noise in the estimate of the statistics used to normalize each variable.

Goodfellow, I. et al, (2016) Deep Learning, The MIT Press, page 425.

Convolutional Neural Networks -Transfer learning

- Accelerate the training with transfer learning
- Transfer learning should enable us to utilize knowledge from previously learned tasks and apply them to newer, related ones.
- Transfer learning is not a new concept, we introduce it specifically to deep learning.

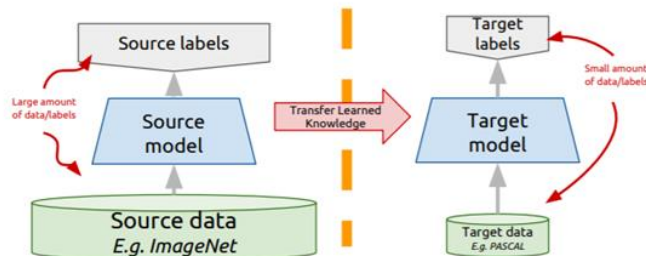
Transfer learning: idea

Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different **source task**
- Adapt it for your domain and your **target task**

Variations:

- Same domain, different task
- Different domain, same task



Convolutional Neural Networks -Transfer learning

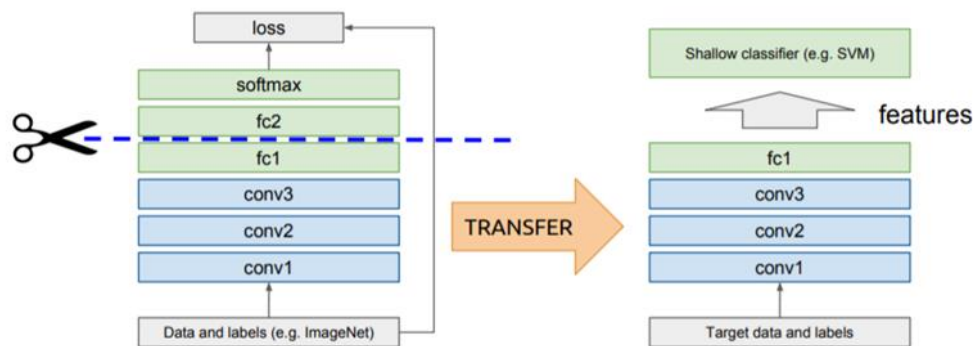
What to transfer and how?

Convolutional Neural Networks -Transfer learning

- Pre-trained Models as Feature Extractors
- we freeze layers in the network to use them as feature extractors

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



Convolutional Neural Networks -Transfer learning

- Fine Tuning Pre-trained Models
- We utilize the knowledge in terms of the overall architecture of the network
- Use its states as the starting point for our retraining step.
- This, in turn, helps us achieve better performance with less training time.

Fine-tuning: supervised domain adaptation

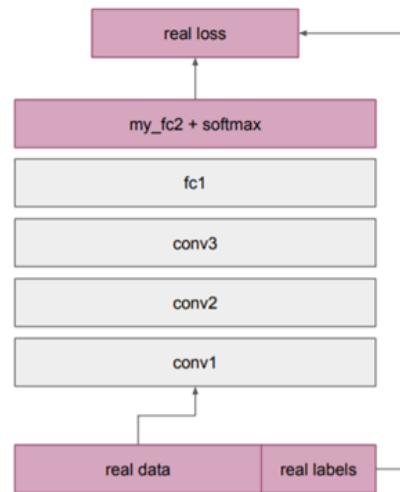
Train deep net on “nearby” task for which it is easy to get labels using standard backprop

- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

Cut off top layer(s) of network and replace with supervised objective for target domain

Fine-tune network using backprop with labels for target domain until validation loss starts to increase

Aligns D_S with D_T



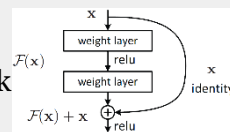


Wound image classification- Transfer learning in practice

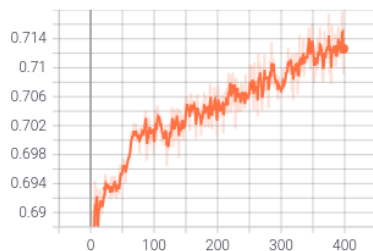


More training epoch, more layers but increased the computational complexity

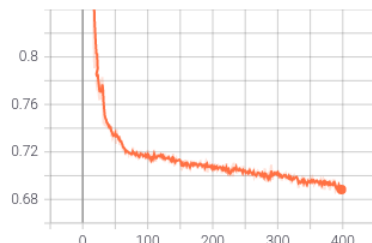
Residual Block



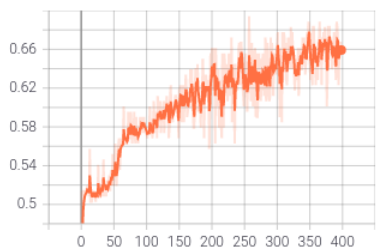
Acc1
tag: Train/Acc1



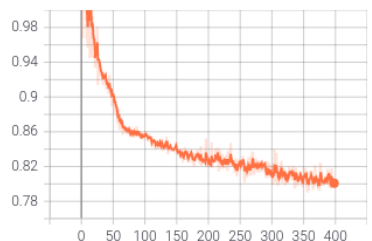
Loss
tag: Train/Loss



Acc1
tag: Val/Acc1



Loss
tag: Val/Loss

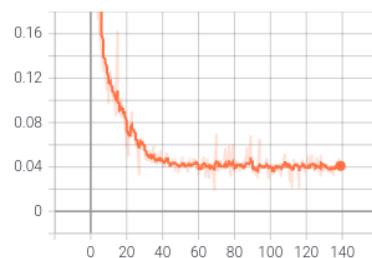


ResNet50 with random initialization (66.7%)

Acc1
tag: Train/Acc1



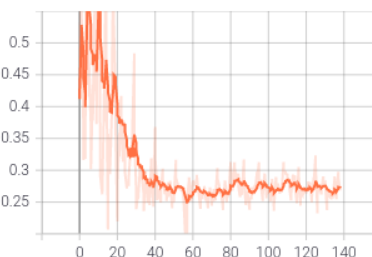
Loss
tag: Train/Loss



Acc1
tag: Val/Acc1



Loss
tag: Val/Loss



ResNet50 with pre-trained model (91.37%)

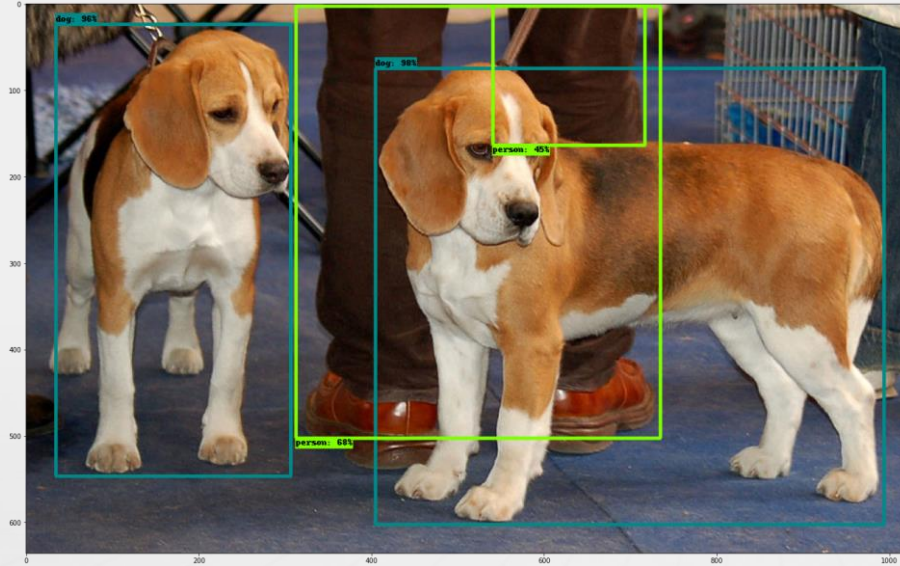


Outline

Covering:

- Overfitting and Underfitting in Deep Learning Neural Networks
(Overfitting, Underfitting, Bias, variance)
- Regularization and transfer learning in deep CNNs
(L1 and L2 Regularization, dropout, batch normalization, transfer learning)
- **Classic downstream tasks using CNN**
(object detection: R-CNN, SPP-net, fast R-CNN, faster R-CNN, YOLO, SSD. Semantic segmentation: FCN, encoder-decoder architectures)

CNN based downstream tasks-Object detection



- Predicting the location of the object along with the class is called object detection.
- The most robust object classification and detection algorithms use deep learning architectures (CNNs).

CNN based downstream tasks-Object detection

R-CNN: Region-based Convolutional Neural Networks-2014

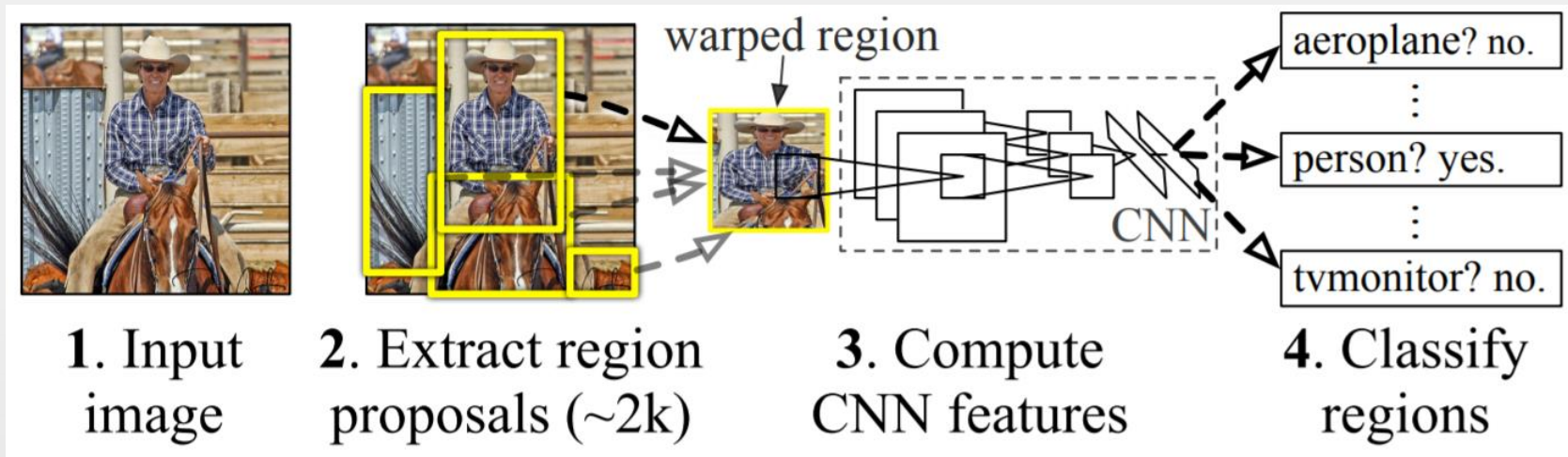
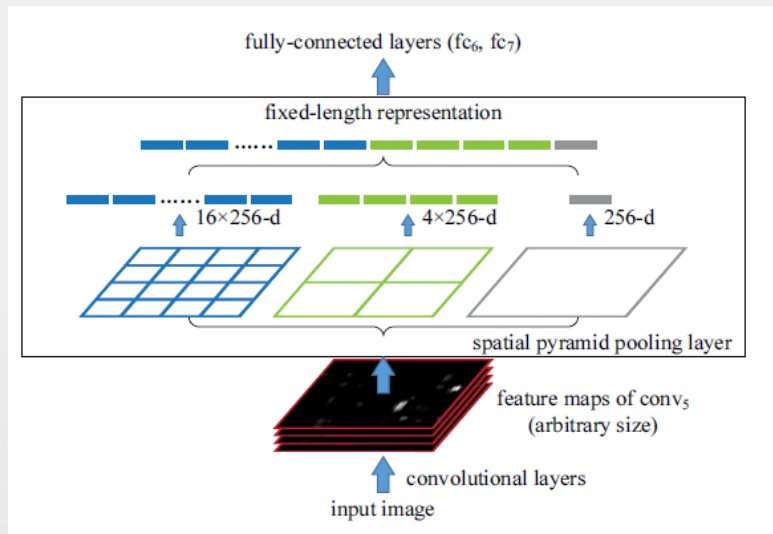


Fig. The flowchart of R-CNN, which consists of 3 stages: (1) extracts bottom-up region proposals with Selective Search, (2) computes features for each proposal using a CNN, and then (3) classifies each region with class-specific linear SVMs.

CNN based downstream tasks-Object detection

SPP-net: Spatial Pyramid Pooling network-2015



- With SPP-net, we calculate the CNN representation for entire image only once and can use that to calculate the CNN representation for each patch generated by Selective Search.
- SPP-net reuses feature maps of the 5-th conv layer (conv₅) to project region proposals of arbitrary sizes to fixed-length feature vectors.

The architecture of SPP-net for object detection

K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 9, pp. 1904–1916, 2015.

CNN based downstream tasks-Object detection

Fast R-CNN-2015

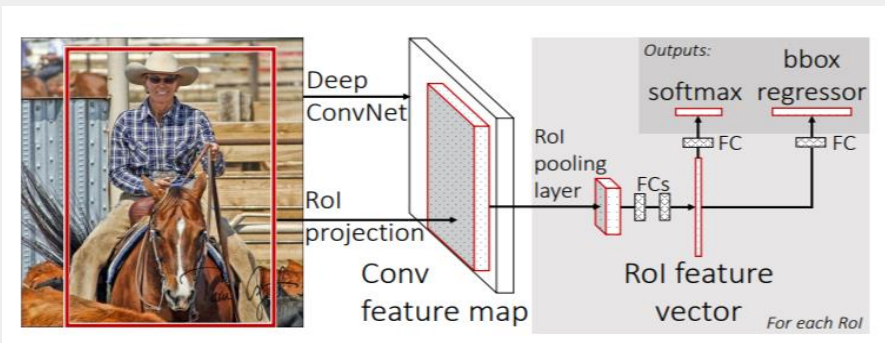


Fig. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

- Fast RCNN uses the ideas from SPP-net and RCNN and fixes the key problem in SPP-net i.e. they made it possible to **train end-to-end**.

Multi-task loss.

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v).$$

softmax
probabilities

bounding-box
regression offsets

CNN based downstream tasks-Object detection

Faster R-CNN-2015

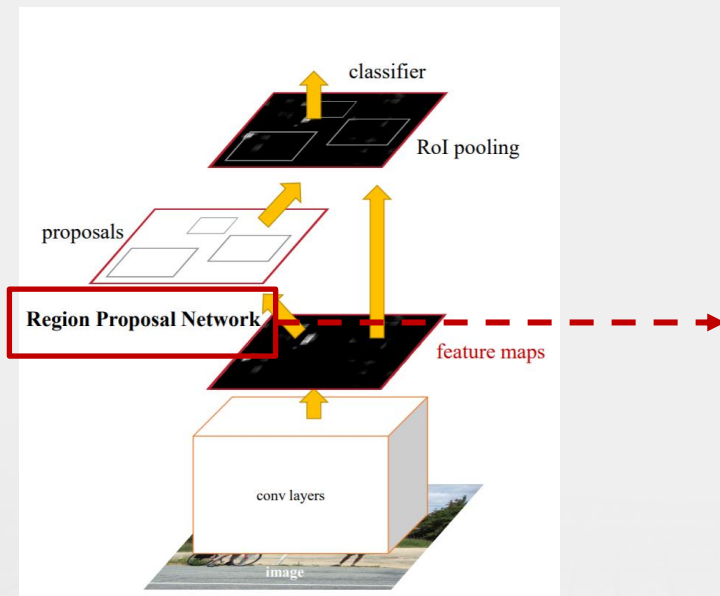


Fig. 1 Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

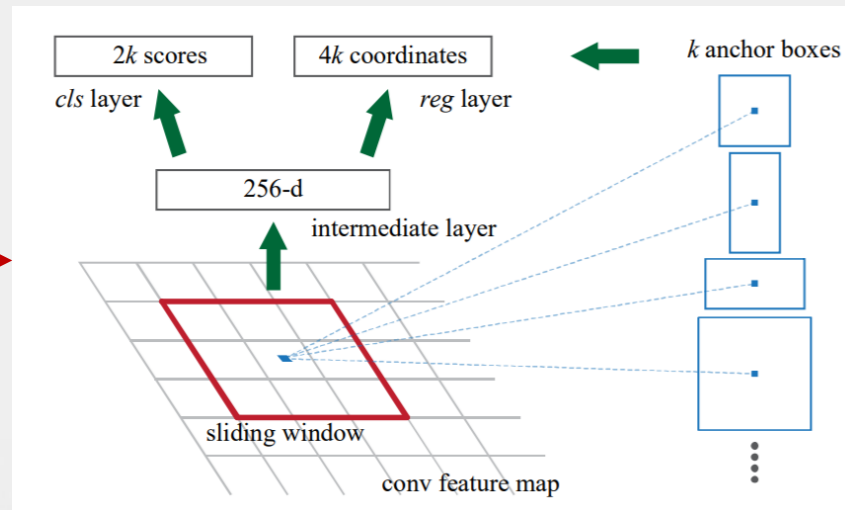


Fig.2 the RPN in Faster R-CNN [18]. K predefined anchor boxes are convoluted with each sliding window to produce fixed-length vectors which are taken by cls and reg layer to obtain corresponding outputs.

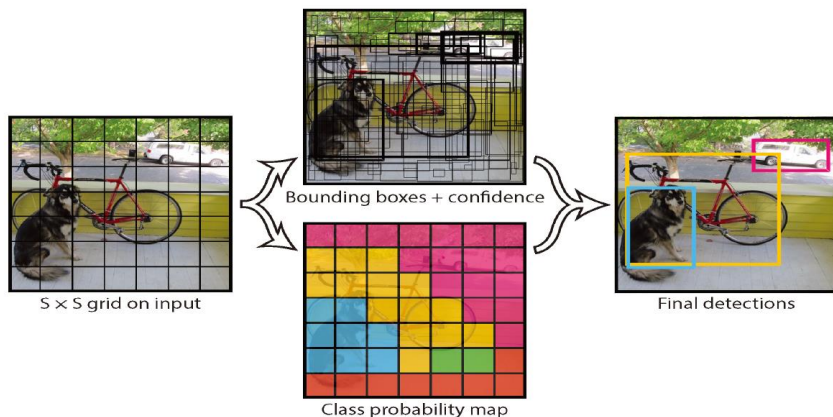
CNN based downstream tasks-Object detection

R-CNN, Fast R-CNN, and Faster R-CNN

	R-CNN	Fast R-CNN	Faster R-CNN
Test Time per Image	50 Seconds	2 Seconds	0.2 Seconds
Speed Up	1x	25x	250x

CNN based downstream tasks-Object detection

YOLO(You only Look Once)-2016

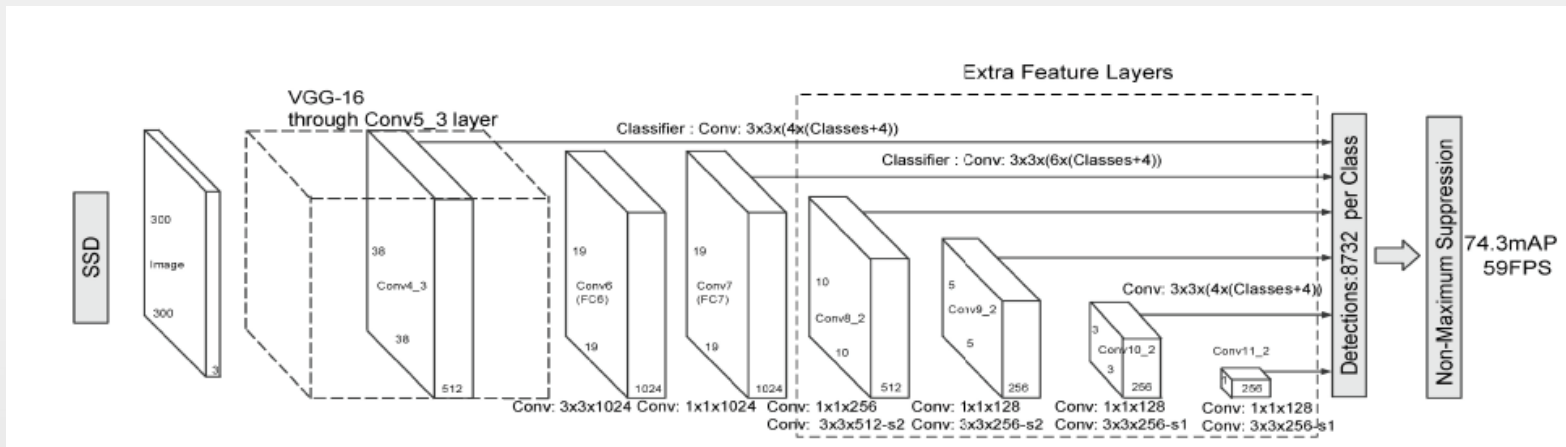


- The YOLO runs the input image on CNN only once, Hence, YOLO is super fast and can be run real time.
- It makes use of the whole topmost feature map to predict both confidences for multiple categories and bounding boxes.
- One limitation for YOLO is that it only predicts 1 type of class in one grid hence, it struggles with very small objects. It struggles to generalize to objects in new/unusual aspect ratios

Fig. YOLO divides the input image into an $S \times S$ grid and each grid cell is responsible for predicting the object centered in that grid cell. Each grid cell predicts B bounding boxes and their corresponding confidence scores. The confidence reflects the accuracy of the bounding box and whether the bounding box actually contains an object. YOLO also predicts the classification score for each box for every class in training.

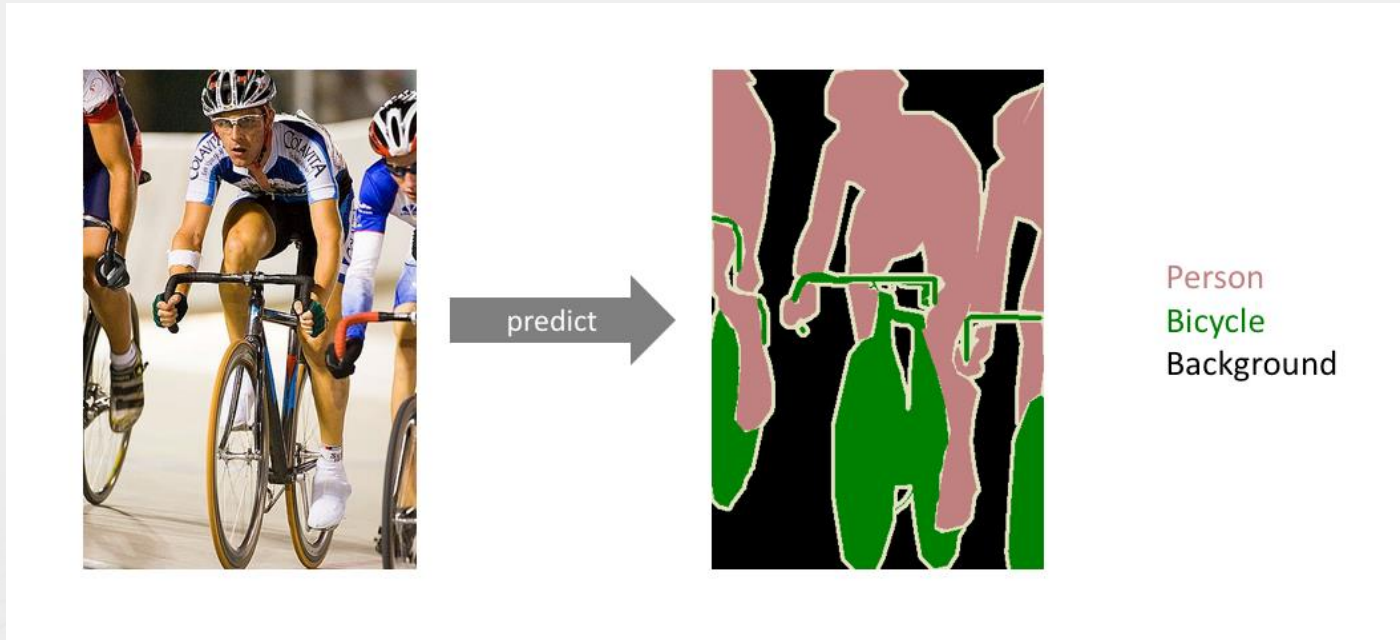
CNN based downstream tasks-Object detection

SSD(Single shot multibox detector)-2016



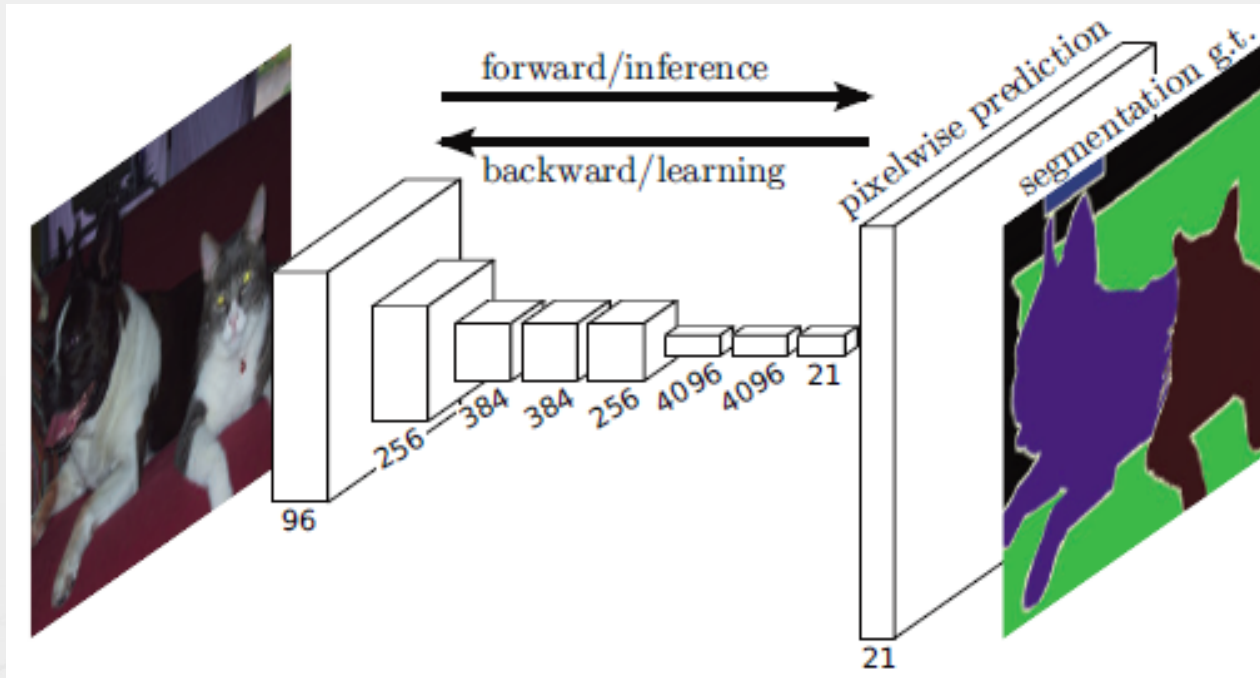
- Given a specific feature map, instead of fixed grids adopted in YOLO, the SSD takes advantage of a set of default anchor boxes with different aspect ratios and scales to discretize the output space of bounding boxes.
- To handle objects with various sizes, the network fuses predictions from multiple feature maps with different resolutions .

CNN based downstream tasks-Image segmentation



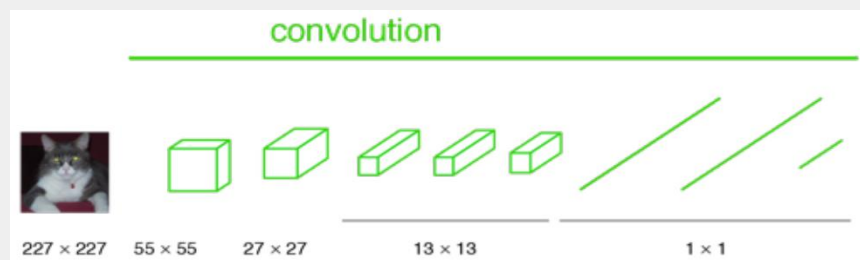
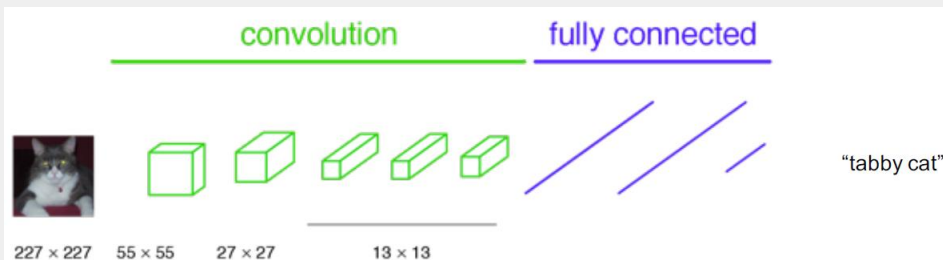
An example of semantic segmentation, where the goal is to predict class labels for each pixel in the image

CNN based downstream tasks-Image segmentation

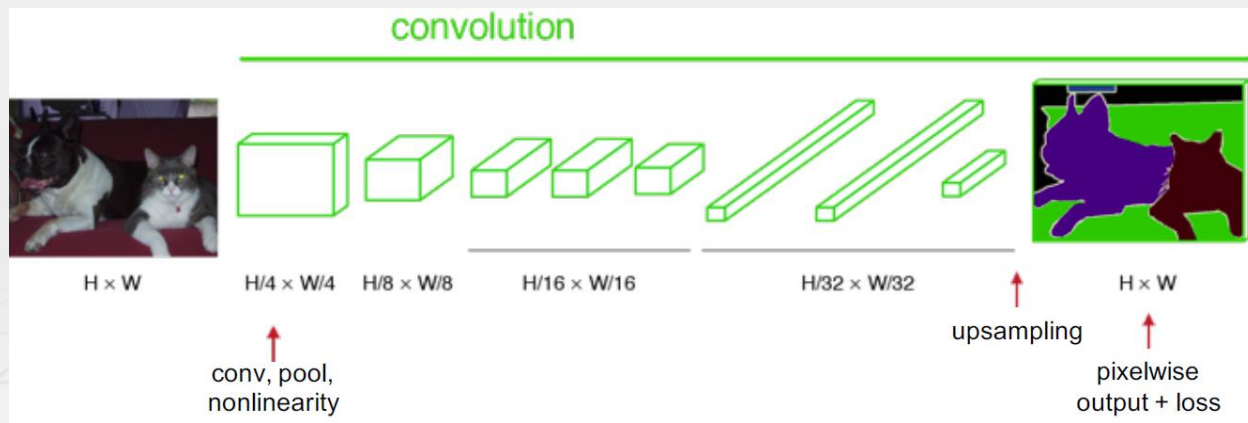


Fully convolutional networks for semantic segmentation

CNN based downstream tasks-Image segmentation



Imagine we turn the FC layers into 1×1 convolutional layers



Convolutional Encoder-Decoder network-Upsampling Techniques

Nearest Neighbor

1	2
3	4



1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

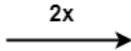
Input: 2 x 2

Output: 4 x 4

- **Nearest Neighbors:** In Nearest Neighbors, as the name suggests we take an input pixel value and copy it to the K-Nearest Neighbors where K depends on the expected output.

10	20
30	40

2x2

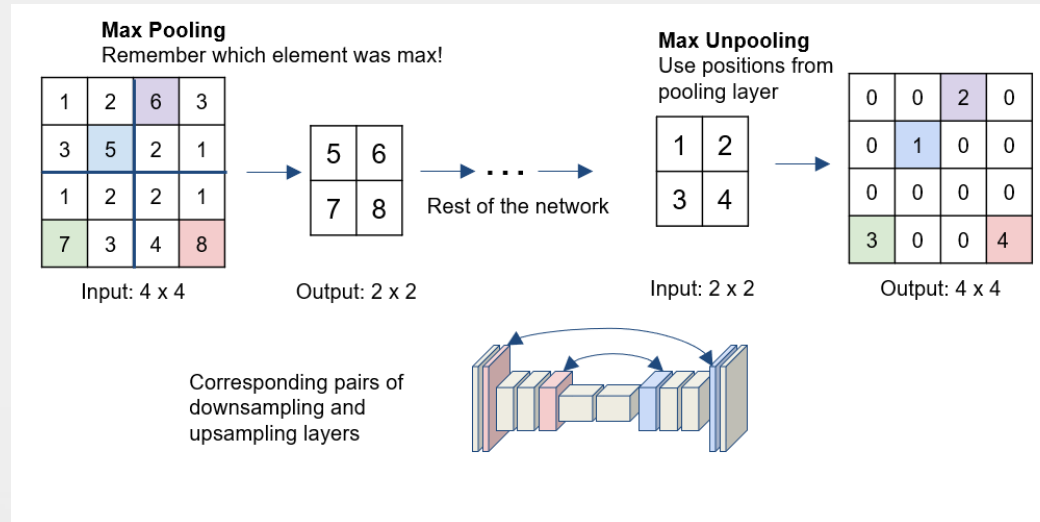


10	12	17	20
15	17	22	25
25	27	32	35
30	32	37	40

4x4

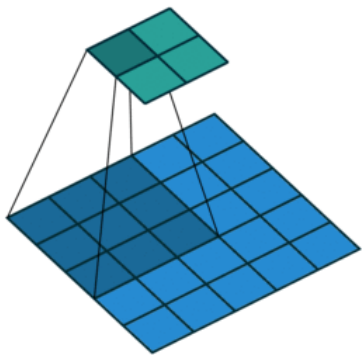
- **Bi-Linear Interpolation:** In Bi-Linear Interpolation, we take the 4 nearest pixel value of the input pixel and perform a weighted average based on the distance of the four nearest cells smoothing the output.

Convolutional Encoder-Decoder network-Upsampling Techniques

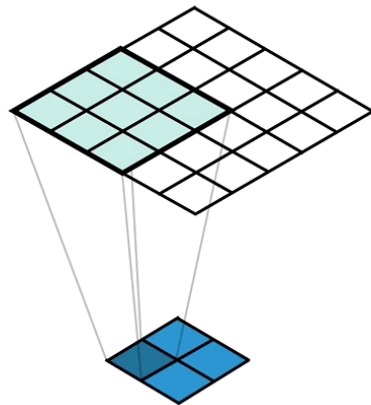


Max-Unpooling: The Max-Pooling layer in CNN takes the maximum among all the values in the kernel. To perform max-unpooling, first, the index of the maximum value is saved for every max-pooling layer during the encoding step. The saved index is then used during the Decoding step where the input pixel is mapped to the saved index, filling zeros everywhere else.

Convolutional Encoder-Decoder network-Upsampling Techniques



2D convolution with no padding, stride of 2 and kernel of 3 applied to a 5x5 input to give a 2x2 output.

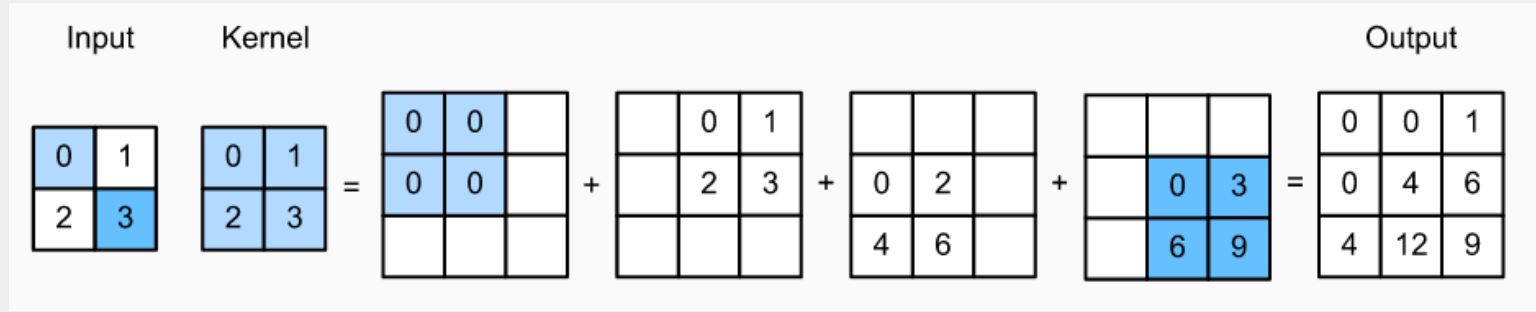


A **Conv2DTranspose** with kernel of 3 and stride of 2 applied to a 2x2 input to give a 5x5 output.

Transposed Convolutions: are used to upsample the input feature map to a desired output feature map using some learnable parameters.

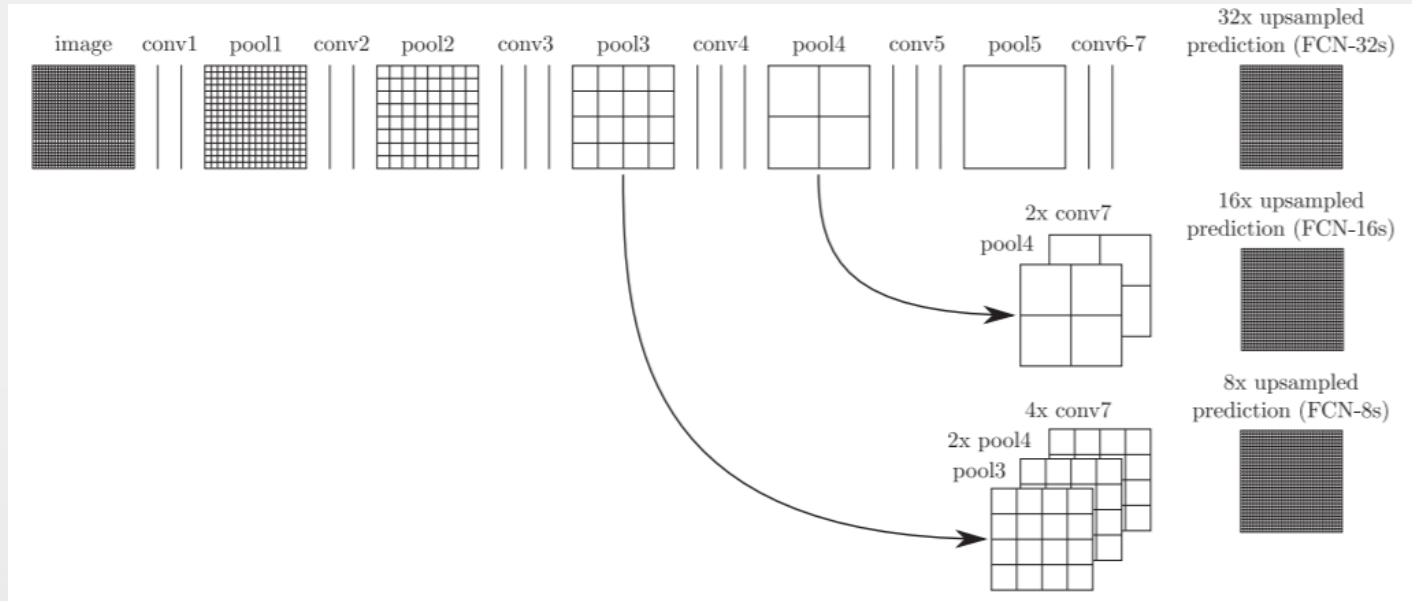
Transposed Convolutions still performing a normal convolution operation.

Convolutional Encoder-Decoder network-Upsampling Techniques



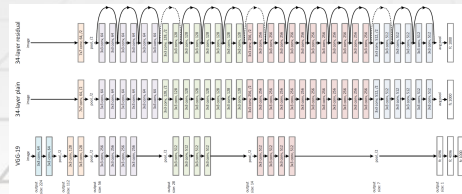
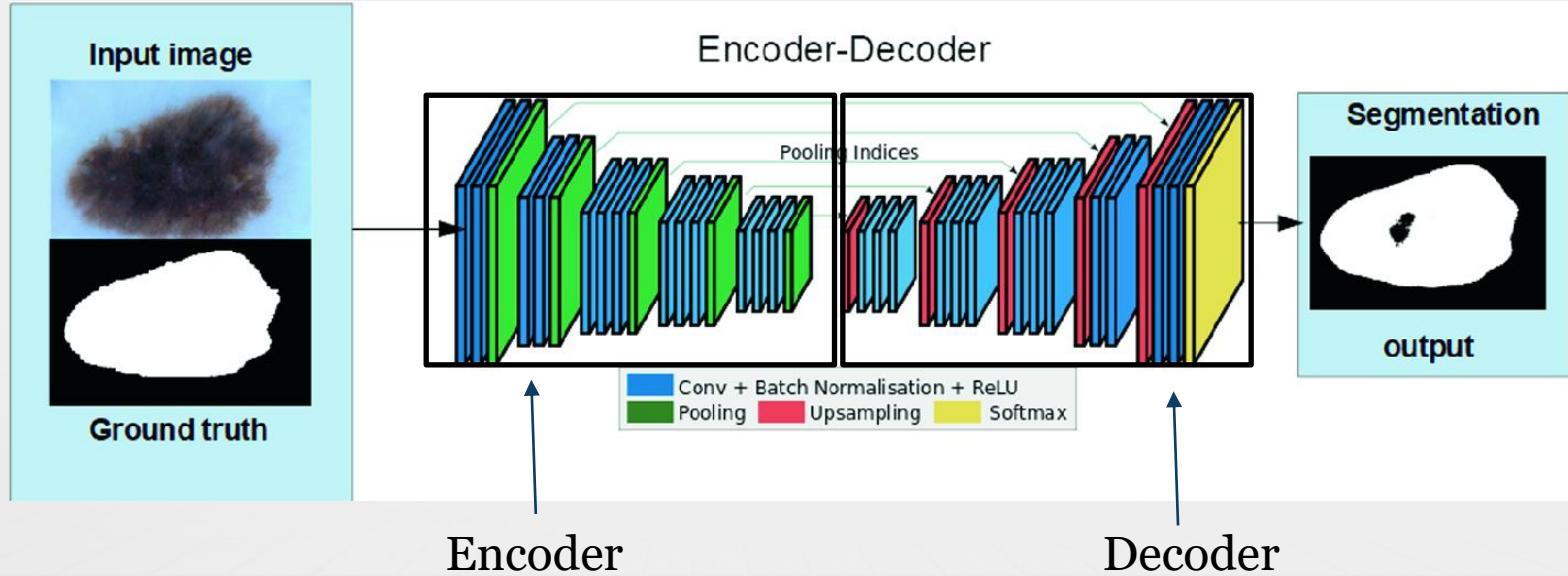
Transposed Convolutions: Transposed Convolutions are used to upsample the input feature map to a desired output feature map using some learnable parameters.

CNN based downstream tasks-Image segmentation




Deep features can be obtained when going deeper, spatial location information is also lost when going deeper. That means output from shallower layers have more location information. If we combine both, we can enhance the result. To combine, we fuse the output (by element-wise addition):

Convolutional Encoder-Decoder network- image segmentation





Summary

- Overfitting and Underfitting in Deep Learning Neural Networks
(Overfitting, Underfitting, Bias, variance)
 - Regularization and transfer learning in deep CNNs
(L1 and L2 Regularization, dropout, batch normalization, transfer learning)
 - Classic downstream tasks using CNN
(object detection: R-CNN, SPP-net, fast R-CNN, faster R-CNN, YOLO, SSD. Semantic segmentation: FCN, encoder-decoder architectures)
- 



Thank you!

Reference

- Raimi Karim, Intuitions on L1 and L2 Regularisation, 2018, <https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261>
- Baeldung, (2020), Batch Normalization in Convolutional Neural Networks, <https://www.baeldung.com/cs/batch-normalization-cnn>
- Sergey Ioffe, Christian Szegedy, (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift PMLR 37:448-456,
- J. Long, E. Shelhamer and T. Darrell, "Fully convolutional networks for semantic segmentation," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 3431-3440, doi: 10.1109/CVPR.2015.7298965.
- W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *ECCV*, 2016.
- J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.
- S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards realtime object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.
- R. Girshick, "Fast r-cnn," in *ICCV*, 2015
- K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, 2015.
- R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.
- Sergey Ioffe, Christian Szegedy, (2015) Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift PMLR 37:448-456,
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (January 2014), 1929–1958.
- Andrew Ng, (2018), Machine Learning Yearning,
- K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *NIPS*, pp. 1106–1114, 2012.