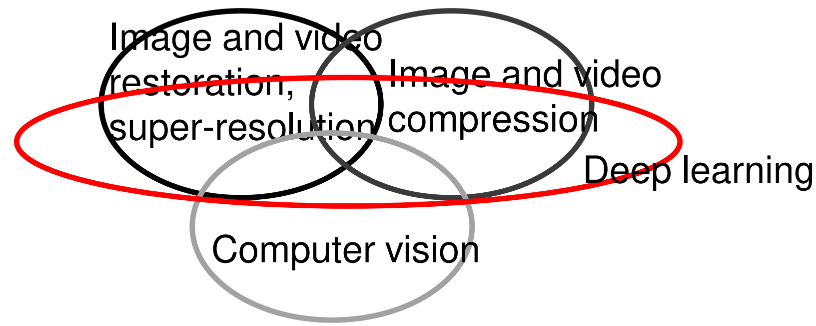
DEEP LEARNING for Image and Video Processing

A. Murat TEKALP
Ogün Kırmemiş
Koç University, İstanbul, Turkey

EUSIPCO 2018 Rome, Italy

Contents

Bridging Communities



- Computer Vision, AI
  - Image classification
  - Video object segmentation and tracking
  - Activity modeling/detection/recognition
  - Video understanding
- Signal Processing
  - Non-linear signal processing
  - Learned image restoration, super-resolution
  - Learned image/video compression


Wiener deconvolution


DNN deconvolution

Analysis of Artefacts

- Degradation Model

$$r(n_1,n_2) = h(n_1,n_2) ** s(n_1,n_2) + v(n_1,n_2)$$

$$R(u_1,u_2) = H(u_1,u_2)S(u_1,u_2) + V(u_1,u_2)$$

- Linear-shift invariant regularized restoration

$$\hat{S}(u_1,u_2) = \mathbf{\Phi}(u_1,u_2)R(u_1,u_2)$$

$$= \mathbf{\Phi}(u_1,u_2)\{H(u_1,u_2)S(u_1,u_2) + V(u_1,u_2)\}$$

Add and substract $S(u_1,u_2)$ to the right hand side:

$$= S(u_1,u_2) + \{\mathbf{\Phi}(u_1,u_2)H(u_1,u_2) - 1\}S(u_1,u_2) + \mathbf{\Phi}(u_1,u_2)\,V(u_1,u_2)$$

Deviation from the inverse filter

Enhanced noise

Signal-dependent ringing artefacts

A.M. Tekalp and M. I. Sezan, ``Quantitative analysis of artifacts in linear space-invariant image restoration,'' Multidimensional Systems and Sign. Proc., vol. 1, pp. 143-177, June 1990.

PART 1

THEORETICAL FOUNDATIONS

Adaptive Signal Processing

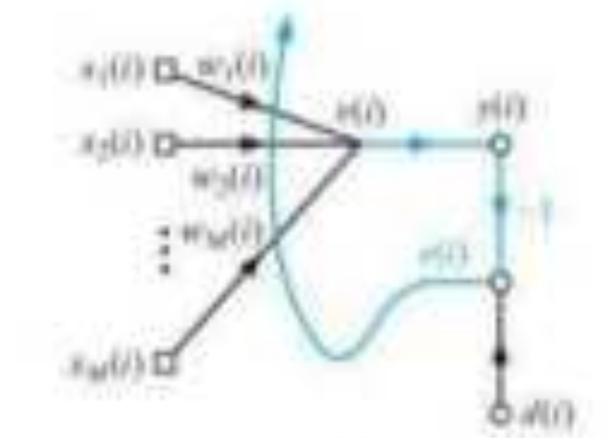
- An adaptive linear filter with time-varying weights **w**(i), input vector **x**(i), and desired output d(i), adjusts the weights to minimize the output error.

Filter output:

$$y(i) = \sum_{k=1}^{M} w_k(i)\, x_k(i) \quad i = 1, \cdots, N$$

Output error:

$$e(i) = d(i) - y(i)$$



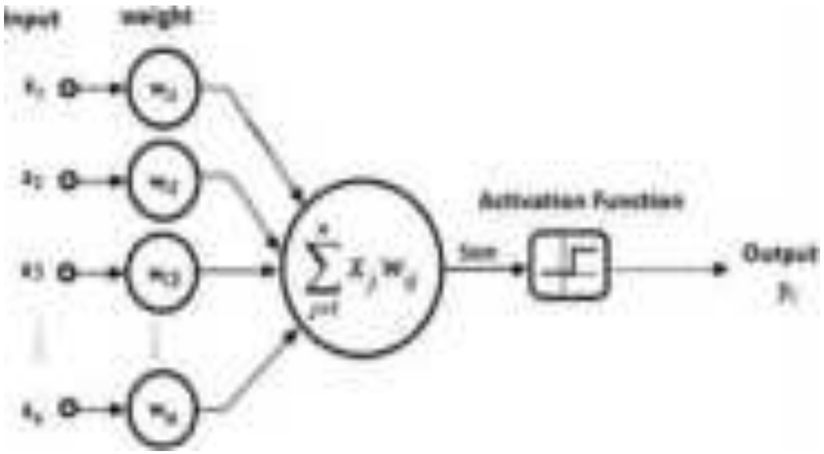
- Minimizing the mean square error

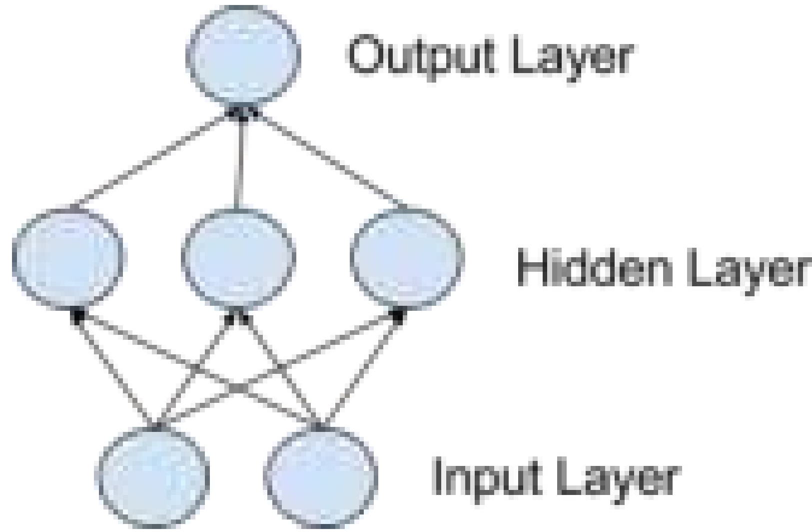
$$E\{[e(i)]^2\} = \sum_{i=1}^{N} e^2(i)$$

gives the Wiener solution in optimal prediction/filtering

- The LMS algorithm (Widrow-Hoff, 1960) minimizes $e^2(i)$ with respect to **w**(i) at each time step.

Neural Networks 101





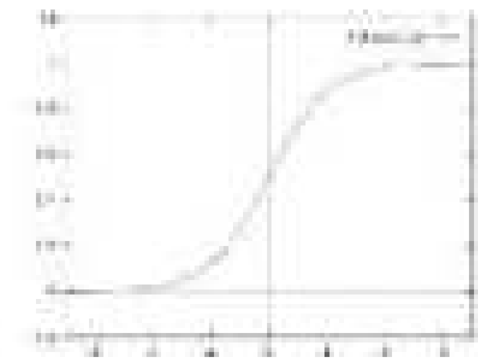
Single neuron Simple network (1 hidden layer)

- Activation Functions: A Neural Network without an activation function would simply be a **linear regression model**, which has limited capability.

Activation Functions

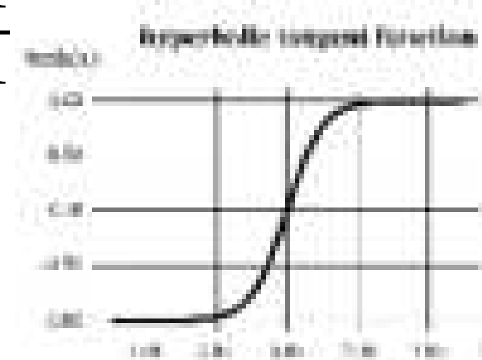
Activation function should be differentiable so as to perform to compute gradient of output error (loss function) with respect to unknown weights.

$$y(x) = \frac{1}{1+e^{-x}}$$



- Logistic function (sigmoid)
  - Vanishing gradients
  - Sigmoids saturate and kill gradients.
  - Output is between 0 and 1, not 0-centered

$$y(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

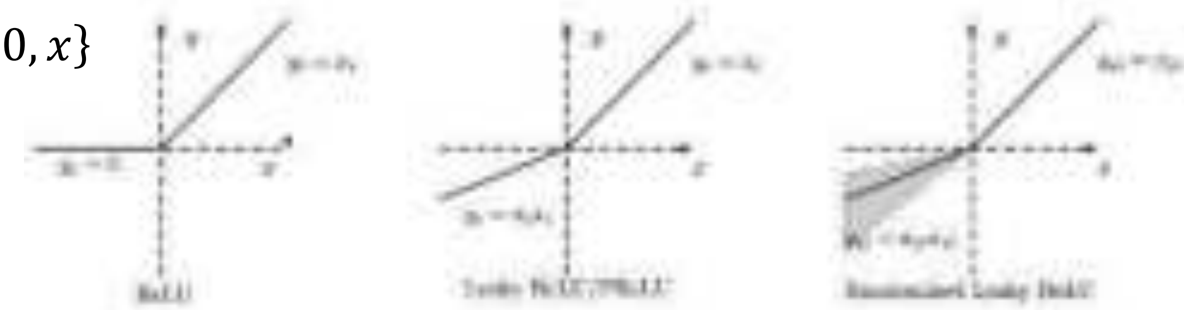


- Hyperbolic tangent
  - Output is 0-centered in between -1 to 1
  - Vanishing gradients

Activation Functions (cont'd)

- Rectified linear unit (RELU)

  $y(x) = \max\{0, x\}$

  - Avoids vanishing gradients problem
  - Only used in hidden layers – in the output layer use softmax for classification problems and linear layer for regression problems
  - Dead neuron problem – use Leaky RELU

- Scaled Exponential Linear Unit (SELU)

$$y(x) = \lambda \begin{cases} x & x > 0 \\ \alpha(e^{x} - 1) & x < 0 \end{cases}$$

  - The parameter $\lambda > 1$.

G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, Self-normalizing neural networks, 2017

Universal Approximation Theorem

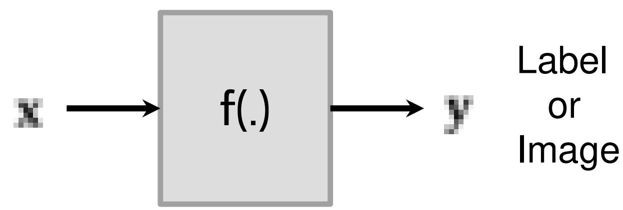
- ***Universal Approximation Theorem***: Given any continuous function $f(\mathbf{x})$, we can find a single- or multi-layer NN whose output $g(\mathbf{x})$ satisfies
$$|g(\mathbf{x}) - f(\mathbf{x})| < \epsilon$$
for all inputs $\mathbf{x}$ and some desired accuracy $\epsilon > 0$.
- Shallow and fat networks
  - Difficult to train for complex tasks
- Deep networks
  - model and learn very complex functions by a nested composition $f(\mathbf{x}) = f_1(f_2(\dots(f_n(\mathbf{x}))))$ of many simpler functions (each function representing a layer)
- The theoretical foundation of learning end-to-end image/video processing systems using NN rests on *UAT*.

KOÇ ÜNİVERSİTESİ

Deep Image/Video Processing

- Image processing systems are functions

$$\mathbf{y} = f(\mathbf{x})$$



  - Classification and regression problems
  - Deep convolutional networks to learn f(.)

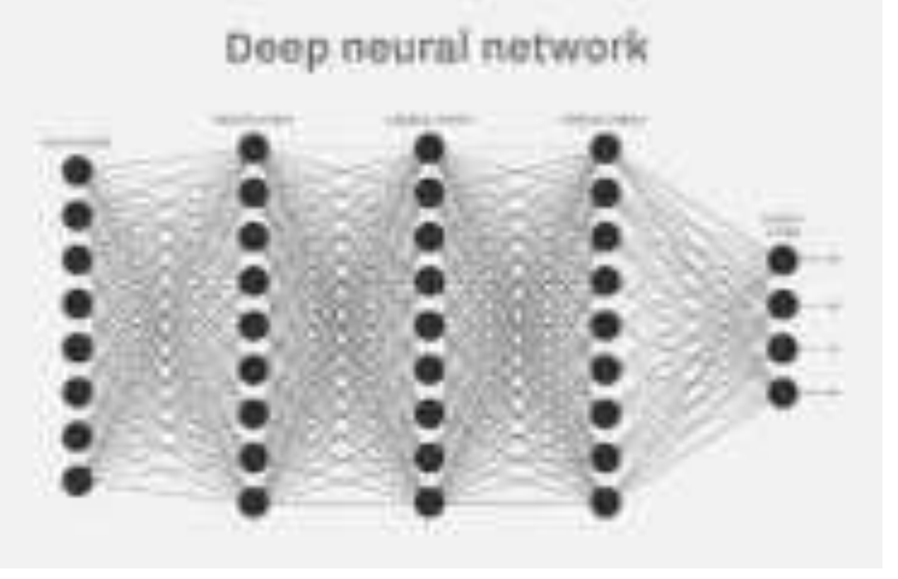
- Video processing systems are time-varying

$$\mathbf{y}(\mathbf{t}) = f(\mathbf{x}(\mathbf{t}), \mathbf{t})$$

  - Deep recurrent networks to learn f(.,t)

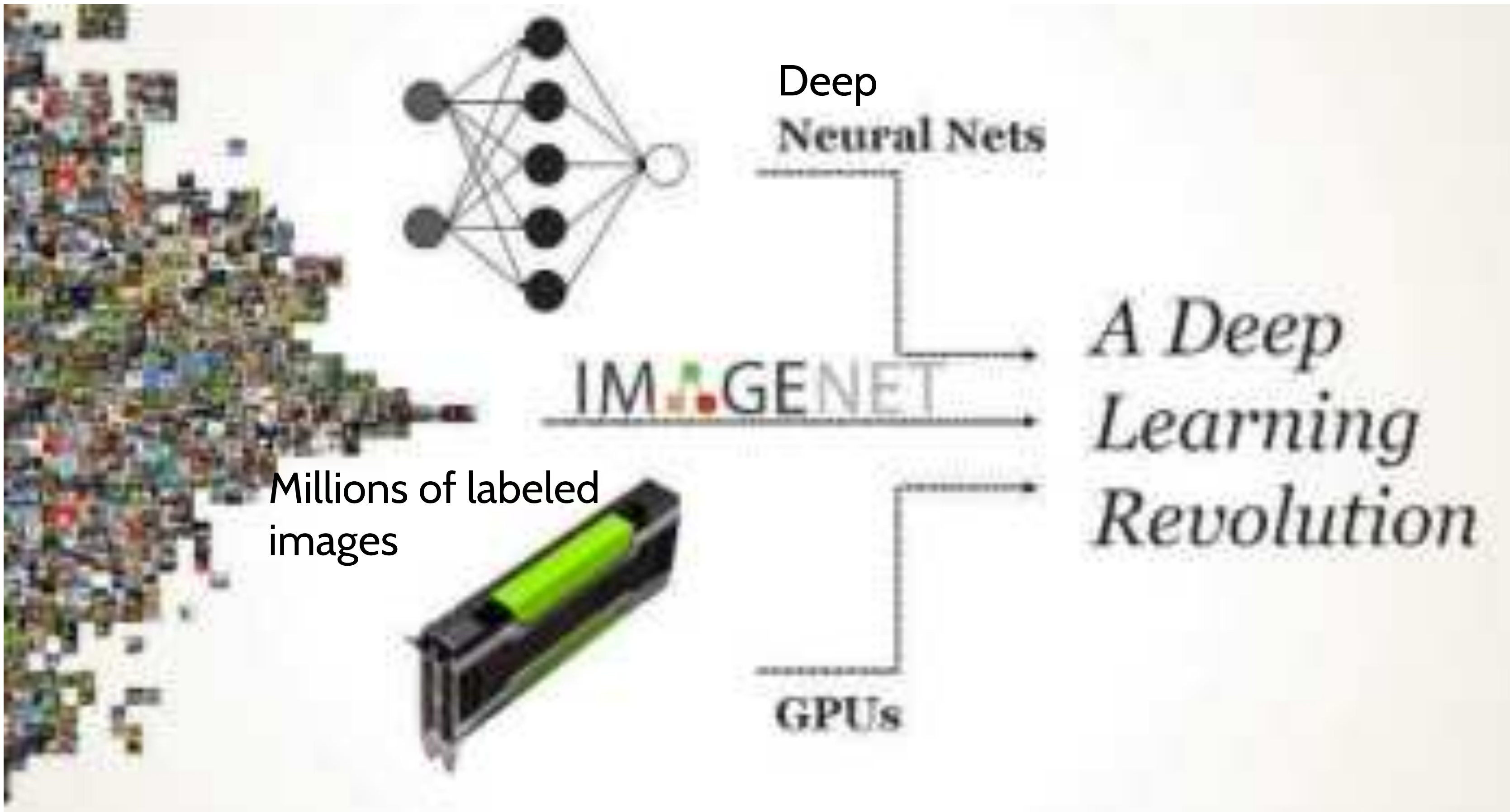
Deep Neural Networks

- Multi-layer perceptron (fully-connected)



- Convolutional networks
- Fat vs. Deep networks

Big Data and Deep Learning



Applications of Deep Learning

- **Classification problems**
  - DNN learns a mapping between images and labels (cats vs. dogs)
  - ImageNet: 1,5 million images, 1,000 class labels
  - Probability estimation – Softmax layer

$$p(x_i) = \frac{e^{-x_i}}{\sum_{j=1}^{K} e^{-x_j}}, \qquad i = 1, \cdots, K$$

- **Regression problems**
  - DNN learns a mapping between input and output images
  - Image restoration, super-resolution, in-painting
- **Image generation**
  - DNN learns a generative model $p(x|z)$, where z is a latent variable. New images are generated by sampling from the pdf $p(x|z)$

KOÇ ÜNİVERSİTESİ

Supervised Training

- Training data set
  - Given input, output pairs $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right), i = 1, \cdots, N$
- Optimization problem
  - Loss functions
  - Find $\mathbf{w}$ to minimize

$$\sum_{i=1}^{N} \left(\mathbf{y}^{(i)} - \widehat{\boldsymbol{y}}(\mathbf{w}, \mathbf{x}^{(i)})\right)^2$$

- Solution
  - Non-convex optimization by gradient descent
  - Computation of gradients – back propagation

Back-propagation

- Network is initiated with random weights.
- Forward pass: Given input, the output error is computed.
- Backward pass: The gradient of the output error function with respect to weights is computed to update previous weights
- Chain rule of differentiation
- Different gradient descent procedures exist

D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representation by back-propagating errors, Nature, 323, pp. 533-536, 9 Oct. 1986.

Koç Üniversitesi

Back-prop Example

- Forward pass:

$$net_{h1} = w_1\, i_1 + w_2\, i_2 + b_1$$

$$out_{h1} = \frac{1}{1 + e^{-net_{h1}}}$$

$$net_{h2} = w_3\, i_1 + w_4\, i_2 + b_1$$

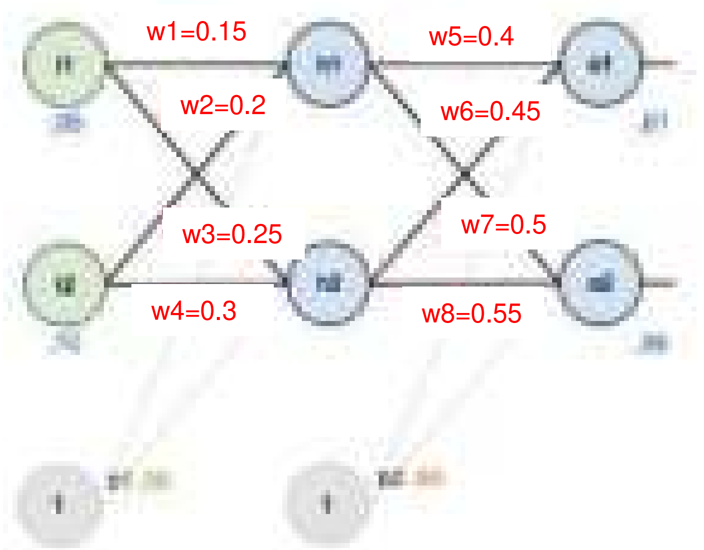
$$out_{h2} = \frac{1}{1 + e^{-net_{h2}}}$$

$$net_{o1} = w_5\, out_{h1} + w_6\, out_{h2} + b_2$$

$$out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

$$net_{o2} = w_7\, out_{h1} + w_8\, out_{h2} + b_2$$

$$out_{o2} = \frac{1}{1 + e^{-net_{o2}}}$$



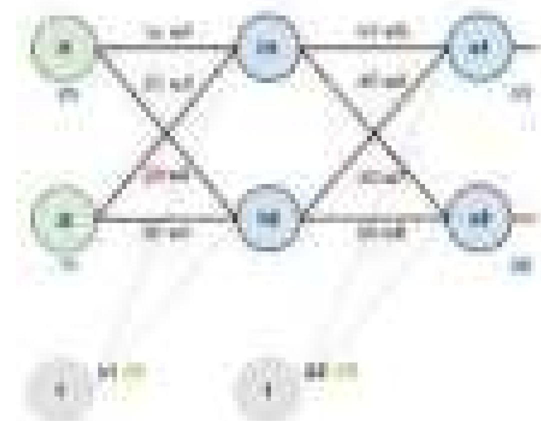
$$net_{h1} = 0.3775$$
$$out_{h1} = 0.593269992$$
$$net_{h2} = 0.3925$$
$$out_{h2} = 0.596884378$$

$$net_{o1} = 1.105905967$$
$$out_{o1} = 0.75136507$$
$$net_{o2} = 1.2249207$$
$$out_{o2} = 0.772928465$$

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Back-prop Example (cont'd)

- Backward pass:
  - Output Layer



$$E_{total} = \frac{1}{2}(0.01 - out_{o1})^2 + \frac{1}{2}(0.99 - out_{o2})^2$$

$$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta out_{o1}} \frac{\delta out_{o1}}{\delta net_{o1}} \frac{\delta net_{o1}}{\delta w_5}$$

$$\frac{\delta E_{total}}{\delta out_{o1}} = -(0.01 - out_{o1})$$

$$\frac{\delta out_{o1}}{\delta net_{o1}} = \frac{e^{-net_{o1}}}{(1+e^{-net_{o1}})^2} = \frac{1}{1+e^{-net_{o1}}} \frac{e^{-net_{o1}}}{1+e^{-net_{o1}}} = out_{o1}(1\text{- } out_{o1})$$

$$\frac{\delta net_{o1}}{\delta w_5} = out_{h1}$$

$$\frac{\delta E_{total}}{\delta w_5} = (0.74136)\ (0.18681)(0.59326)$$
$$= 0.08216$$

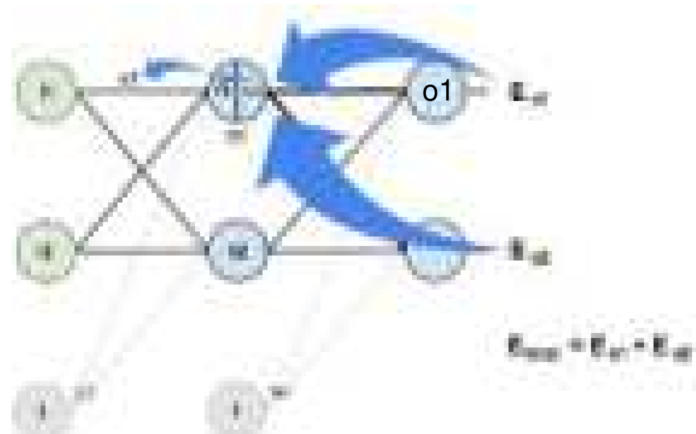
Compute $\frac{\delta E_{total}}{\delta w_6}, \frac{\delta E_{total}}{\delta w_7}, \frac{\delta E_{total}}{\delta w_8}$ similarly

  - Gradient-descent update

$$w_5 = w_5 + \alpha \frac{\delta E_{total}}{\delta w_5} \quad w_6 = w_6 + \alpha \frac{\delta E_{total}}{\delta w_6} \quad w_7 = w_7 + \alpha \frac{\delta E_{total}}{\delta w_7} \quad w_8 = w_8 + \alpha \frac{\delta E_{total}}{\delta w_8}$$

Back-prop Example (cont'd)

- Backward pass:
  - Hidden Layer



$$\frac{\delta E_{total}}{\delta w_1} = \frac{\delta E_{total}}{\delta out_{h1}} \frac{\delta out_{h1}}{\delta net_{h1}} \frac{\delta net_{h1}}{\delta w_1}$$

$$\frac{\delta net_{h1}}{\delta w_1} = i_1$$

$$\frac{\delta out_{h1}}{\delta net_{h1}} = \frac{1}{1+e^{-net_{h1}}} \frac{e^{-net_{h1}}}{1+e^{-net_{h1}}} = out_{h1}(1\text{-}\, out_{h1})$$

$$\frac{\delta E_{total}}{\delta out_{h1}} = \frac{\delta E_{o1}}{\delta out_{h1}} + \frac{\delta E_{o2}}{\delta out_{h1}}$$

$$\frac{\delta E_{o1}}{\delta out_{h1}} = \frac{\delta E_{o1}}{\delta net_{o1}} \frac{\delta net_{o1}}{\delta out_{h1}}$$

$$\frac{\delta E_{o1}}{\delta net_{o1}} = \frac{\delta E_{o1}}{\delta out_{o1}} \frac{\delta out_{o1}}{\delta net_{o1}}, \qquad \frac{\delta net_{o1}}{\delta out_{h1}} = w_5$$

$$E_{o1} = \frac{1}{2}(0.01 - out_{o1})^2$$

$$E_{o2} = \frac{1}{2}(0.99 - out_{o2})^2$$

$$\frac{\delta E_{total}}{\delta w_1} = (0.036)(0.241)(0.05) = 0.000438$$

Compute

$$\frac{\delta E_{total}}{\delta w_2}, \frac{\delta E_{total}}{\delta w_3}, \frac{\delta E_{total}}{\delta w_4}$$

similarly

  - Gradient-descent update

$$w_1 = w_1 + \alpha \frac{\delta E_{total}}{\delta w_1} \quad w_2 = w_2 + \alpha \frac{\delta E_{total}}{\delta w_2} \quad w_3 = w_3 + \alpha \frac{\delta E_{total}}{\delta w_3} \quad w_4 = w_4 + \alpha \frac{\delta E_{total}}{\delta w_4}$$

KOÇ ÜNİVERSİTESİ

Optimization Methods

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Resource: Andrew Ng video https://www.youtube.com/watch?v=UfNU3Vhv5CA

Batch Gradient Descent

- Given M pairs of training samples $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$
- Compute the gradient of the cost function

$$E_{train}(\mathbf{w}) = \frac{1}{2M}\sum_{i=1}^{M}\left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}\left(\mathbf{x}^{(i)}, \mathbf{w}\right)\right)^2$$

w.r.t. to N weights for the entire training set to perform just one update.

$$\text{Repeat for } j = 0, \cdots, N-1 \qquad \{$$

$$w_j := w_j - \alpha\frac{1}{M}\sum_{i=1}^{M}\left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}\left(\mathbf{x}^{(i)}, \mathbf{w}\right)\right)\frac{\partial\hat{\mathbf{y}}\left(\mathbf{x}^{(i)}, \mathbf{w}\right)}{\partial w_j}$$

$$\}$$

- Batch gradient descent
  - can be very slow when M is large and is intractable for datasets that do not fit in memory.
  - does not allow on-line model updates, i.e. with new samples on-the-fly.
  - is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

Stochastic gradient descent (SGD)

- SGD performs a parameter update for each training sample pair $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$

$$E_{train}(\mathbf{w}) = \frac{1}{2M}\sum_{i=1}^{M}\left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}\left(\mathbf{x}^{(i)}, \mathbf{w}\right)\right)^2$$

For each epoch:

1. Randomly shuffle training data set
2. Repeat for $i = 0, \cdots, M-1$ {

    for $j = 0, \cdots, N-1$ {

$$w_j := w_j - \alpha\left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}\left(\mathbf{x}^{(i)}, \mathbf{w}\right)\right)\frac{\partial \hat{\mathbf{y}}\left(\mathbf{x}^{(i)}, \mathbf{w}\right)}{\partial w_j}$$

} }

- SGD is much faster than batch gradient and can be used to learn online.
- SGD performs frequent updates with a high variance that cause the objective function to fluctuate, which enables it to jump to new and potentially better local minima.

KOÇ ÜNİVERSİTESİ

Mini-batch gradient descent

- Mini-batch gradient descent takes the best of both worlds and performs an update for every mini-batch of K training examples.
- Common mini-batch sizes range between $K = 50$ and $K = 256$, but can vary for different applications.
- It reduces the variance of the parameter updates, which can lead to more stable convergence; and
- It enables use of highly optimized matrix optimizations common to state-of-the-art deep learning frameworks that make computing the gradient w.r.t. a mini-batch very efficient.
- Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used.

Challenges

- SGD maintains a single learning rate (alpha) for all weight updates and alpha does not change during training.
- A learning rate (step size) that is too small causes slow convergence, while a learning rate that is too large causes the loss function to fluctuate around the minimum or even to diverge.
- *Adaptive optimizers* adjust the learning rate for each weight during training, e.g., reduce the learning rate according to a pre-defined schedule or when the change in the objective between epochs falls below a threshold.
- We must avoid getting trapped in suboptimal local minima and saddle points, i.e. points where one dimension slopes up and another slopes down. The saddle points are usually surrounded by a plateau of the same error, which makes it hard for SGD to escape, as the gradient is close to zero in all dimensions.

Adaptive Optimizers

- Momentum
- Nesterov Accelarated Gradient (NAG)
- AdaGrad
- Adadelta
- RMSProp
- Adam
- AdaMax
- Nadam
- AMSGrad

Algorithm 1 Generic Adaptive Method Setup

Input: $x_1 \in \mathcal{F}$, step size $\{\alpha_t > 0\}_{t=1}^{T}$, sequence of functions $\{\phi_t, \psi_t\}_{t=1}^{T}$
for $t = 1$ **to** T **do**
  $g_t = \nabla f_t(x_t)$
  $m_t = \phi_t(g_1, \dots, g_t)$ and $V_t = \psi_t(g_1, \dots, g_t)$
  $\hat{x}_{t+1} = x_t - \alpha_t m_t / \sqrt{V_t}$
  $x_{t+1} = \Pi_{\mathcal{F}, \sqrt{V_t}}(\hat{x}_{t+1})$
end for

$\phi(.)$ and $\varphi(.)$ averaging functions
Π projection
t is the counter of mini-batches

Ex: Stochastic Grad Descent (SGD)

$$\phi_t(g_1, \dots, g_t) = g_t \text{ and } \psi_t(g_1, \dots, g_t) = \mathbb{I}.$$

S. Ruder, An overview of gradient descent optimization algorithms, arXiv, 15 June 2017.

Adaptive Optimizers (cont'd)

- Adaptive Moment Estimation (Adam)
  - computes adaptive learning rates for each parameter
  - uses an exponentially decaying average of past gradients (first moment), like Momentum and AdaGrad
  - Also uses an exponentially decaying average of past squared gradients (second moment), like Adadelta and RMSprop.
- AMSGrad

Algorithm 2 AMSGRAD

Input: $x_1 \in \mathcal{F}$, step size $\{\alpha_t\}_{t=1}^{T}$, $\{\beta_{1t}\}_{t=1}^{T}$, β_2

Set $m_0 = 0$, $v_0 = 0$ and $\hat{v}_0 = 0$

for $t = 1$ to T **do**

$g_t = \nabla f_t(x_t)$

$m_t = \beta_{1t} m_{t-1} + (1 - \beta_{1t}) g_t$

$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$\boxed{\hat{v}_t = \max(\hat{v}_{t-1}, v_t)}$ and $\hat{V}_t = \mathrm{diag}(\hat{v}_t)$

$x_{t+1} = \Pi_{\mathcal{F}, \sqrt{\hat{V}_t}}(x_t - \alpha_t m_t / \sqrt{\hat{v}_t})$

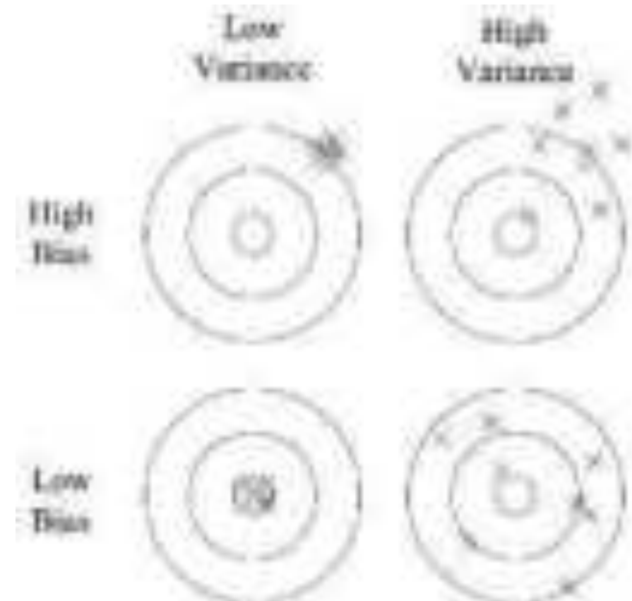
end for

Recommended
$\beta_1 = 0.9$ and $\beta_2 = 0.999$

S.J. Reddi, S. Kale & S. Kumar, On the convergence of Adam and beyond, ICLR 2018.

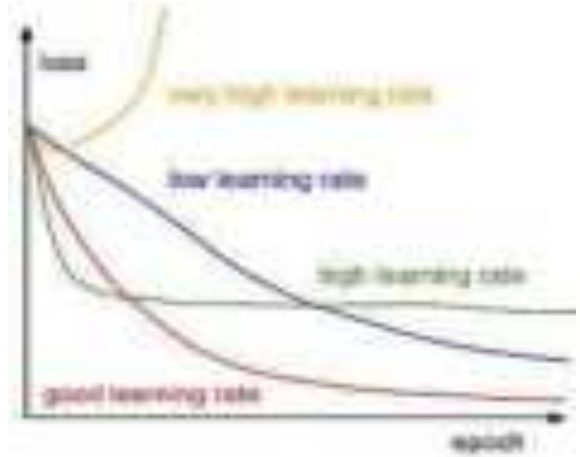
Bias vs. Variance

- Small models typically have high bias (underfitting)
- As the number of parameters in a model increases, the complexity of the model rises and variance (overfitting) becomes the primary concern while bias falls steadily.

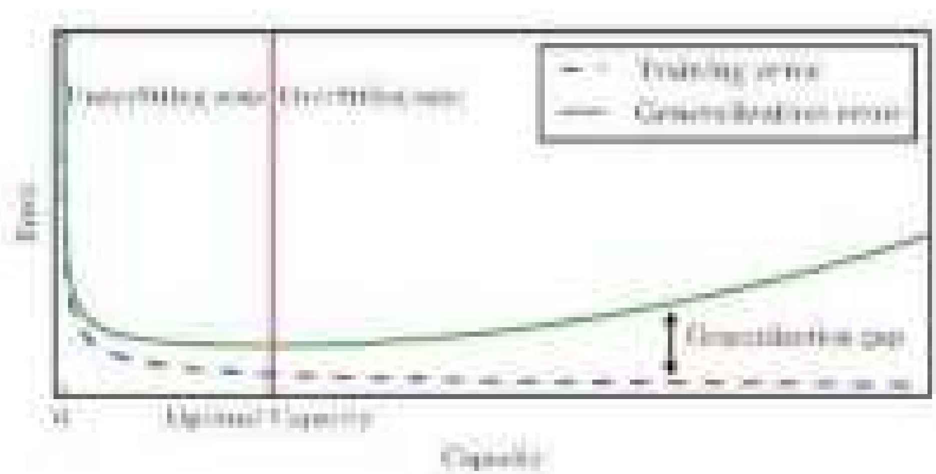


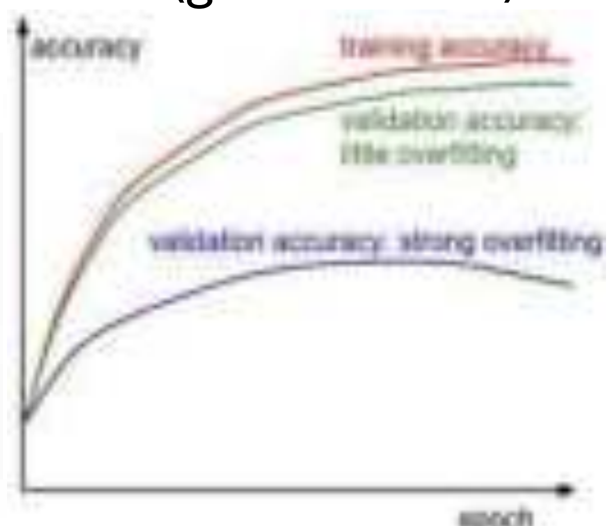
Learning Capacity and Rate

- Convergence
  - Learning rate



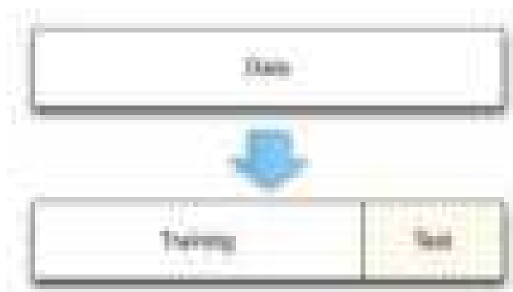
- Overfitting
  - Network size
  - Amount of data
  - Gap between training and test performance (generalization)





Generalization Error

- "resampling based measures such as cross-validation should be preferred over theoretical measures such as Akaike's Information Criteria"



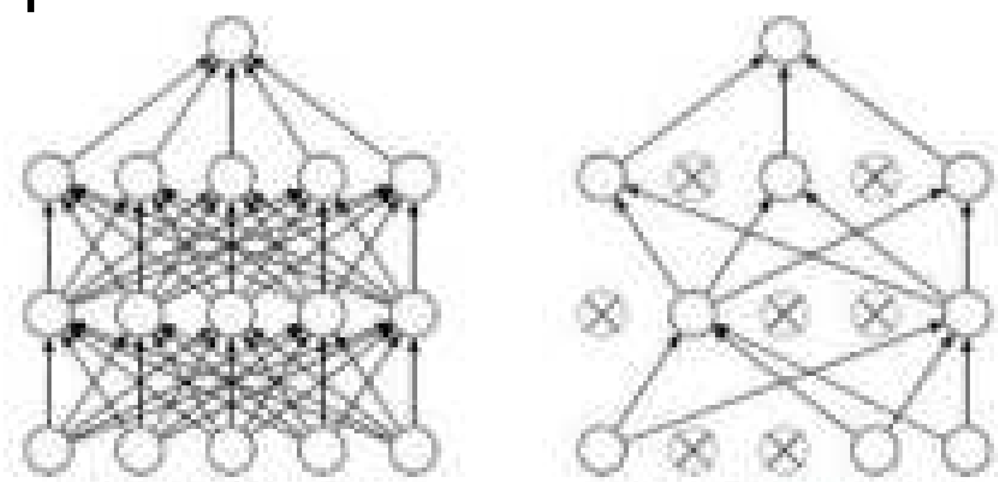
- Hold-out data split

- 5-fold cross-validation data split

Scott Fortmann-Roe, Accurately Measuring Model Prediction Error, 2012
http://scott.fortmann-roe.com/docs/MeasuringError.html

Regularization

- To prevent over fitting
  - Weight-decay (L1 Decay, L2 decay)
  - Drop out



(a) Standard Neural Net (b) After applying dropout.

Amount of Data

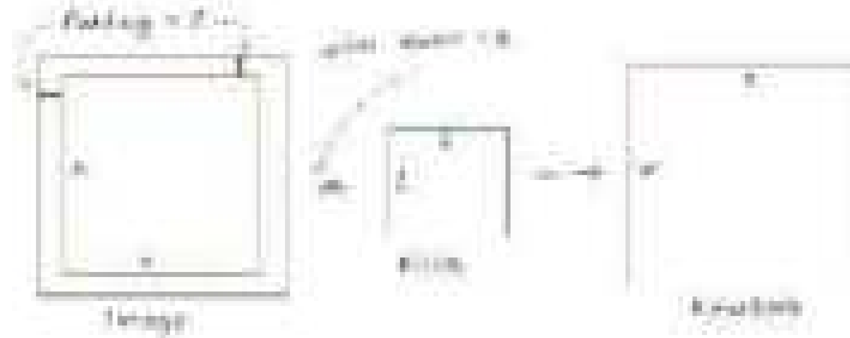
When there is not enough specific training data

- Pre-training
  - on other generic datasets
- Data augmentation
  - Random crop
  - Horizontal, vertical flip
  - Rotations
  - Create synthetic data using the degradation (noise, blur, etc.) model

KOÇ ÜNİVERSİTESİ

Basics of Convolutional Layers

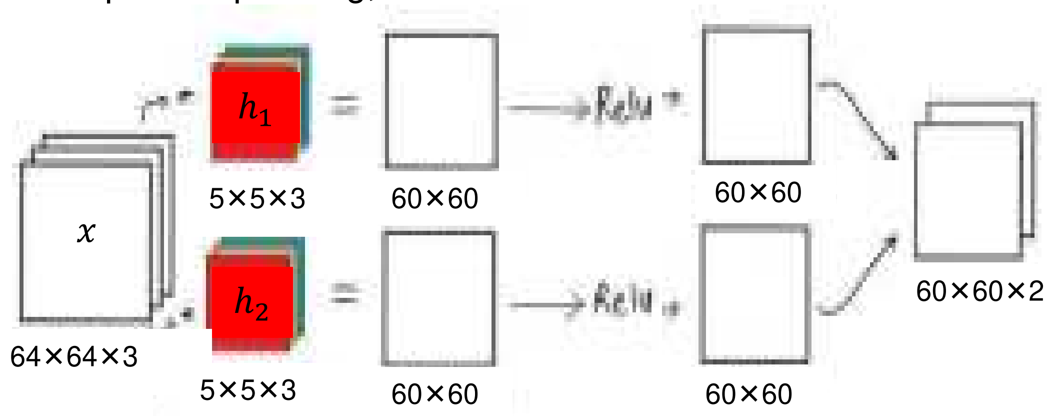
- Convolution
- Padding
- Stride (subsampling)
  - Jump pixels



$$r = \frac{n + 2p - f}{s} + 1$$

$$p = \frac{f-1}{2} \rightarrow \ r = n$$

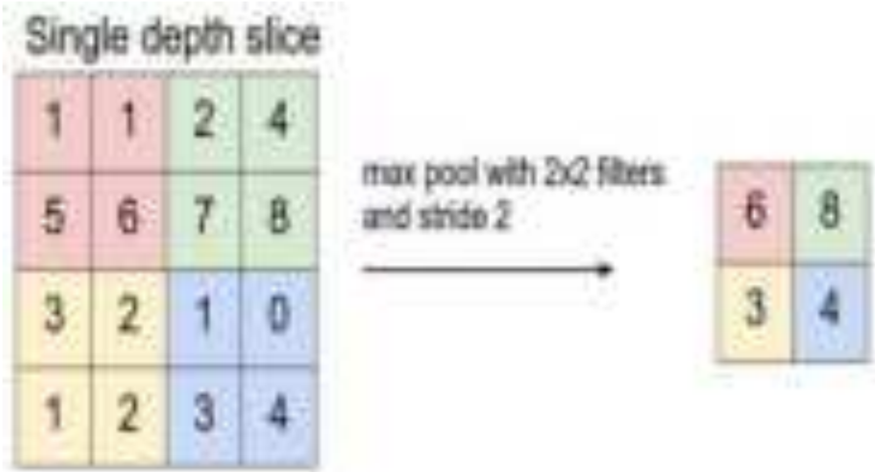
Example: No padding, stride=1



Given an image x with dimensions $N_1 \times N_2 \times \#\mathrm{Channels}$, and two filters h_1 and h_2 with dimensions $H_1 \ \times H_2 \times \#\mathrm{Channels}$. (Channels: R, G, B)

Pooling Layer

- Subsampling
- Max pooling



Normalization Layer

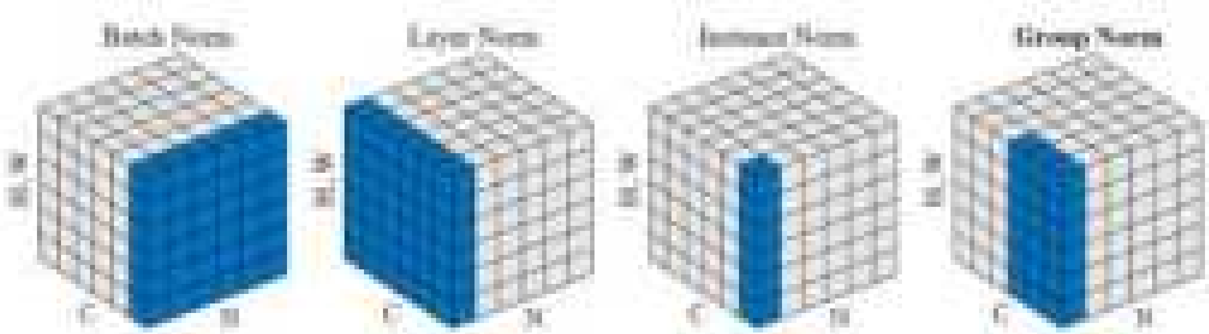
- Input Normalization: Normalizing inputs to mean zero and variance 1 speeds up learning.
- Weight Normalization proposes normalizing the filter weights.
- Normalization of Output of Hidden Layers: Normalizing all features in the hidden layers to mean zero and variance 1 also speeds up learning.

- Smoothing effect: More stable behavior of the gradients.
- *Regularization effect*: Output of hidden layers are scaled by the mean and variance computed on each mini-batch rather than mean and variance using the entire data set. Similar to drop out, this has the effect of adding some small noise to each hidden layer's activation.

Reference video:
https://www.coursera.org/learn/deep-neural-network/lecture/81oTm/why-does-batch-norm-work

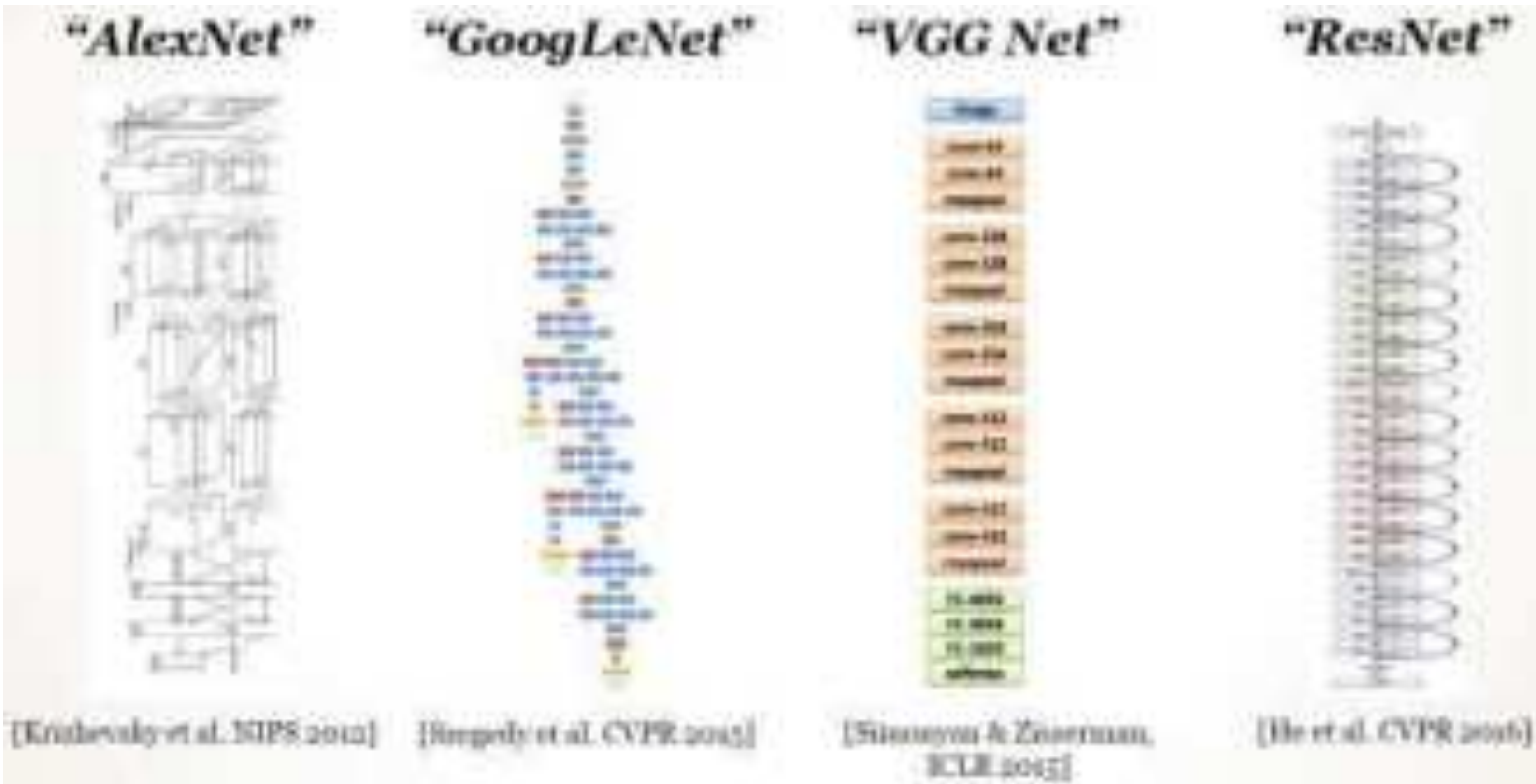
Types of Output Normalization

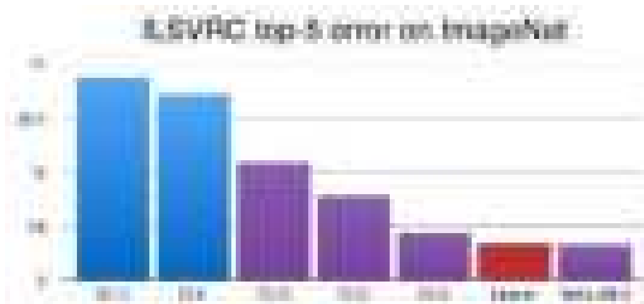
- BatchNorm
- LayerNorm (when the notion of a batch is problematic, e.g., RNN)
- InstanceNorm (normalizes across the height and width)
- GroupNorm (across a subset of the batch, e.g., in case of variable batch size)
- SwitchNorm learns different normalization operations for different normalization layers in a DNN in an end-to-end manner.



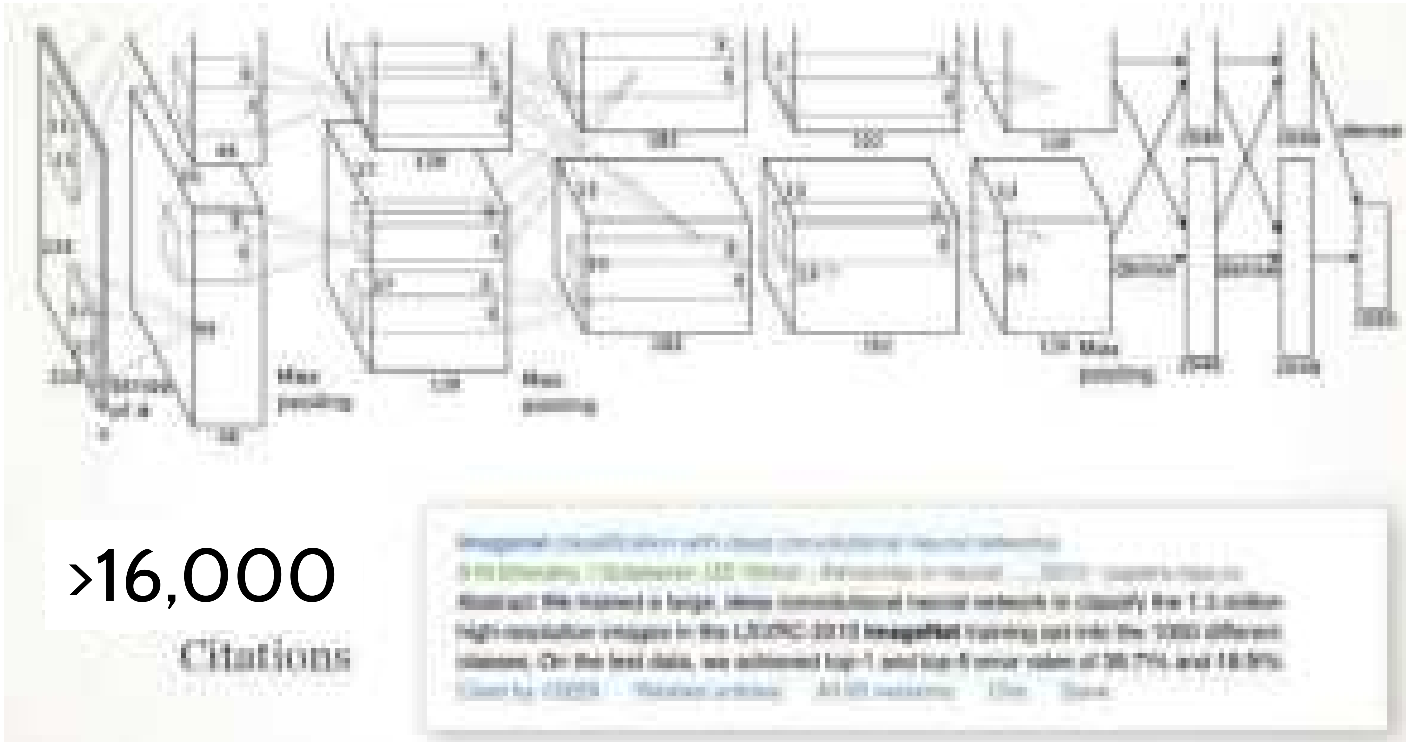
Y. Wu and K. He, Group normalization, arXiv, 2018.
P. Luo, J. Ren and Z. Peng, Differentiable learning-to-normalize via switchable normalization, arXiv, 2018.
S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, How does batch normalization help optimization? arXiv, 2018.

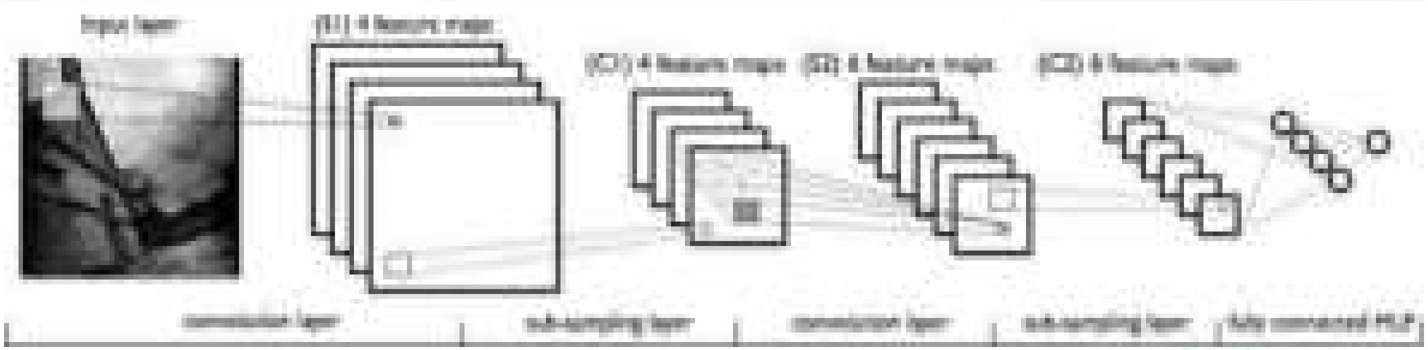
Popular Convolutional Networks





AlexNet (2012)



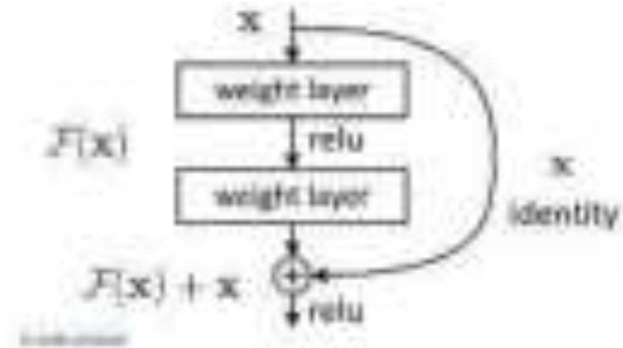


VGG Net (2015)

- Proposed by Karen Simonyan and Andrew Zisserman of the University of Oxford (England). (Very deep convolutional networks for large-scale image recognition, ICLR 2015)
- 19 layer CNN that uses only 3x3 filters (as opposed to AlexNet's 11x11 filters in the first layer) with stride and pad of 1, along with 2x2 maxpooling layers with stride 2.
- A cascade of two 3x3 conv layers has an effective receptive field of 5x5. 3 conv layers back to back have an effective receptive field of 7x7. This simulates a larger filter while keeping the benefits of smaller filters with less number of parameters. In addition, with two (three) conv layers, we're able to use two (three) ReLU layers instead of one.
- The number of filters doubles after each maxpool layer. This reinforces shrinking spatial dimensions, but growing the depth of volume.
- Trained on 4 Nvidia Titan Black GPUs using Stochastic Gradient Descent for **two to three weeks**.

Residual Networks (ResNet 2016)

- The information in the errors is lost due to underflow after about 20 layers. He et al. (Microsoft Research, Asia) realized that this problem could be solved by adding a shortcut path from the input to the output of layers, so each layer can be modeled $h_i(x) = f_i(x) + x$



- ResNet with 152 layers broke the record for ILSVRC challenge and reduced error rate to 3.57% from the previous 6.7% set by GoogleNet.
- Note that after only the *first 2* layers, the spatial size is reduced from an input volume of 224x224 to 56x56.
- Authors claim that a naïve increase of layers in plain nets result in higher training and test error.
- Trained on an 8 GPU machine for **two to three weeks**.

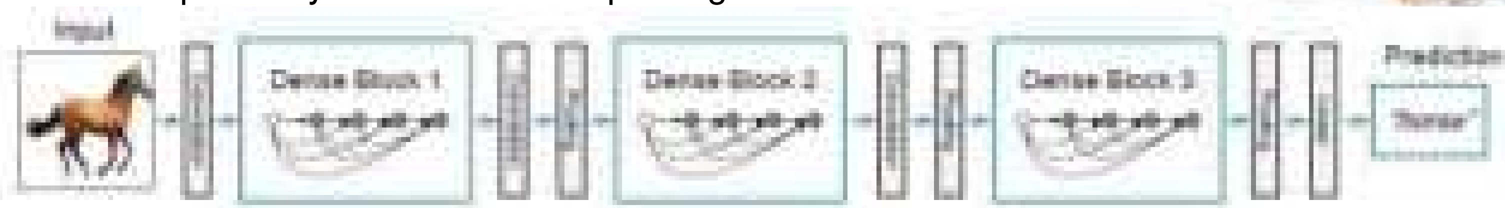
K. He, X. Zhang, S. Ren, and J. Sun, Deep Residual Learning for Image Recognition.

Densely Connected Networks

- DenseNet: create short paths from early layers to later layers (connect all layers with matching feature-map sizes directly with each other)
- Unlike short-cuts in ResNet, DenseNet combines features by concatenating them. Hence, the l'th layer has l inputs, consisting of feature-maps of all preceding convolutional blocks.

The feature-maps of l'th layer are passed on to all $L-l$ subsequent layers. This introduces $L(L+1)/2$ connections in an L-layer DenseNet, instead of just L connections, as in the traditional ConvNet architecture.

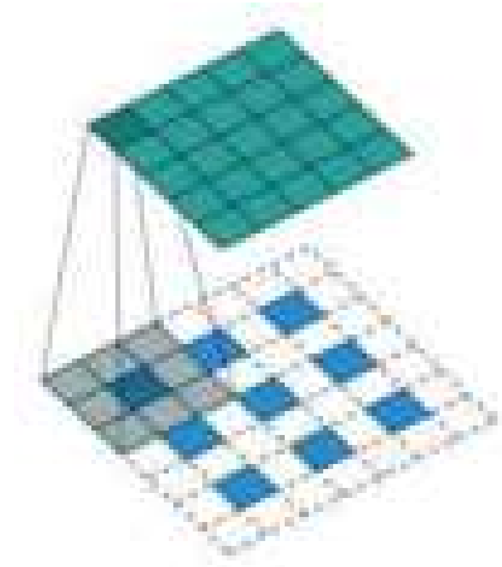
Transition layers between ‘dense blocks’ are used to change feature-map size by convolution and pooling.



G. Huang, Z. Liu, L. van der Maaten, and K Q. Weinberger, Densely connected convolutional networks, arXiv, Aug. 2017.

Upsampling: Transposed Convolution

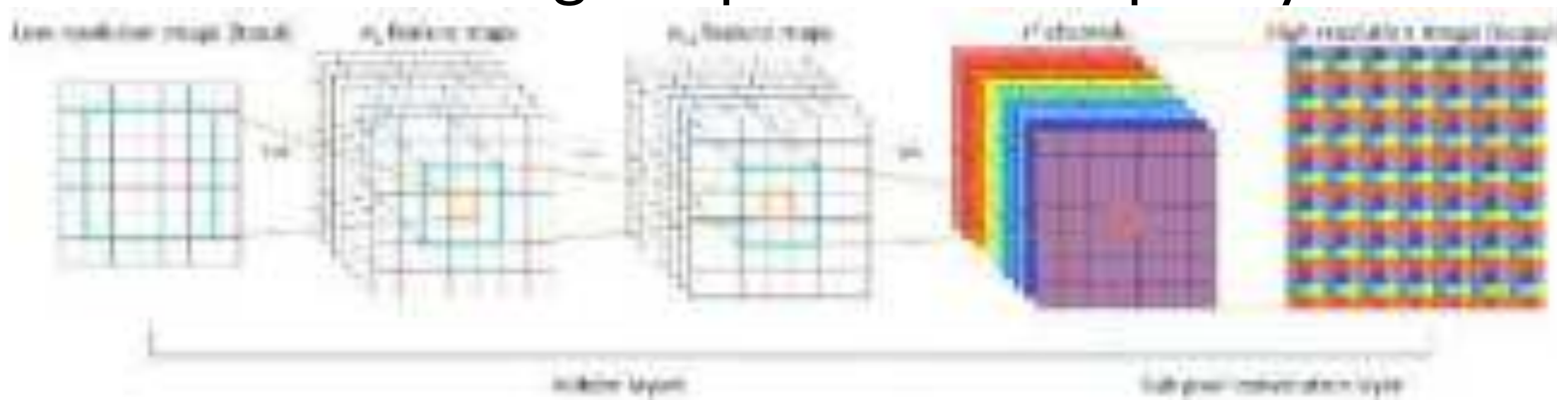
- This operator enlarges the input tensor in height and width dimensions.
- Also known as fractionally strided convolution or deconvolution (although has nothing to do with it)
- Useful for SR, Autoencoders, GANs etc.



V. Dumoulin and F. Visin, A guide to convolution arithmetic for deep learning, arXiv, 2016

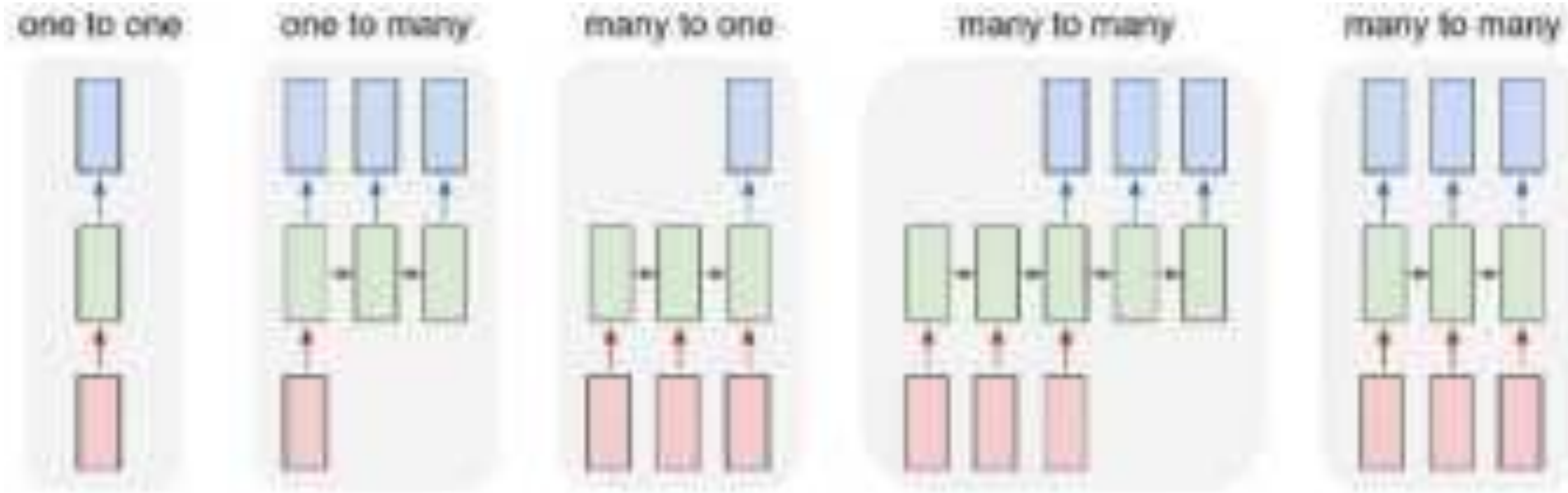
Upsampling: Pixelshuffler

- Transposed convolution has many redundant operations due to zeros. Pixelshuffler interpolates the tensor in the same scale by increasing number of features leading to ease of computation.
- There is no need to upsample images at the input or in the middle of the network. Instead we can do it at the end, decreasing computational complexity.



W. Shi et al., Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), June, 2016

Processing Tasks



(a) (b) (c) (d) (e)

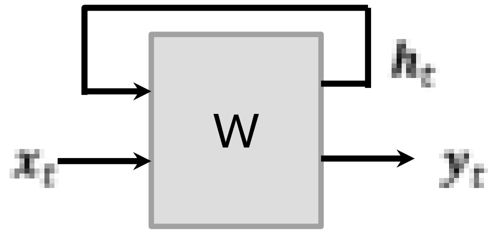
(a) fixed-sized input to fixed-sized output (e.g., ConvNet); (b) single input to sequence output (e.g. captioning an image with multiple words); (c) sequence input, single output (e.g. classify a sentence or video with a label) (d) sequence input, sequence output (e.g., machine translation) (e) synced sequence input and output (e.g., video processing)

- Sequential processing of sequential or time series data

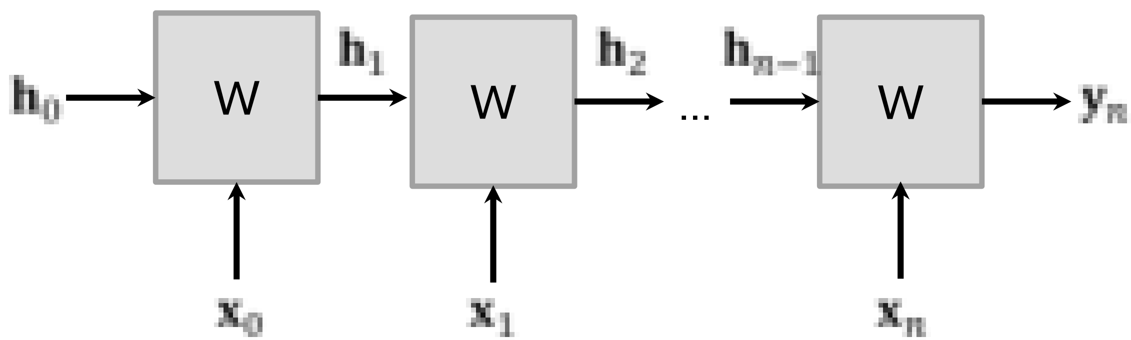
Andrej Karpathy, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Networks

- Dynamic (Temporal) Model



$$\mathbf{h}_t = \varphi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{xh}\mathbf{x}_t)$$
$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t$$



Unrolled recurrent neural network

- LSTM
- GRU

J. Chung, C. Gulcehre, K.H. Cho, Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” NIPS Workshop, 2014.

Training RNN

- Backpropagation-Through-Time
  - Present a sequence of timesteps of input and output pairs to the network.
  - Unroll the network then calculate and accumulate errors across each timestep.
  - Roll-up the network and update weights.
  - Repeat.

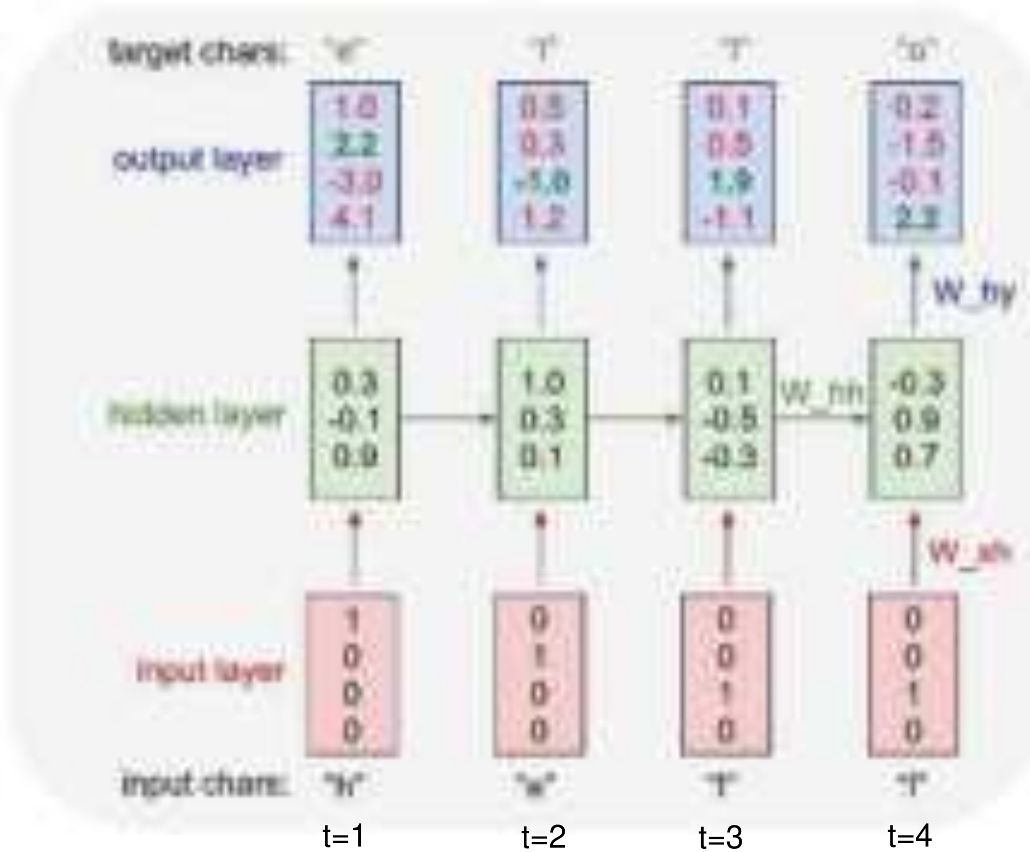
  BPTT is computationally expensive as the number of timesteps increases.
  If input sequences have thousands of timesteps, thousands of derivatives are required for a single weight update. This can cause weights to vanish or explode (go to zero or overflow) and make learning slow.

- Truncated Backpropagation-Through-Time
  - **TBPTT(n,n)**: Updates are performed at the end of each sequence across all timesteps (standard BPTT).
  - **TBPTT(1,n)**: update after each timestep based on all timesteps seen so far.
  - **TBPTT(k1,k2), where k1<k2<n**: Multiple updates are performed per sequence which can accelerate training.
  - **TBPTT(k1,k2), where k1=k2**: A common configuration where a fixed number of timesteps are used for both forward and backward-pass timesteps (e.g. 10s to 100s).

https://machinelearningmastery.com/gentle-introduction-backpropagation-time/

Example: Character-level language model

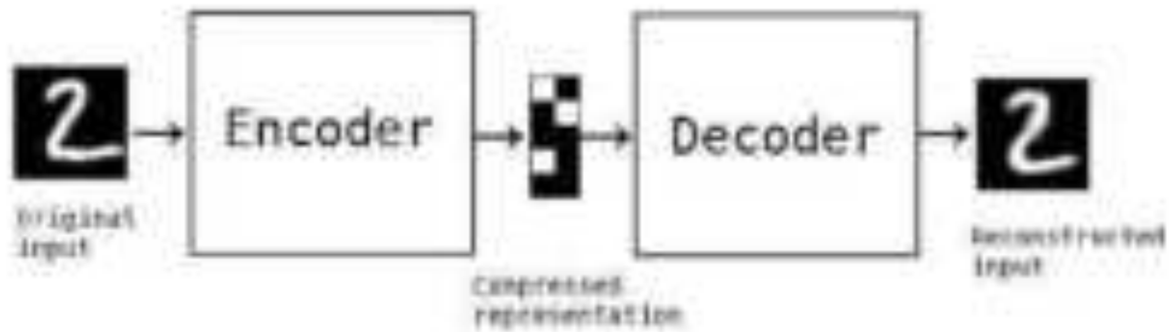
- Four-character dictionary {h,e,l,o}
- Single hidden layer with three nodes



Andrej Karpathy, http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Autoencoders

- Auto-encoders create a latent or compressed representation of raw input data. They achieve dimensionality reduction; i.e., the vector serving as a hidden (latent) representation compresses the input into a small no of salient dimensions.



- Auto-encoders can be paired with a decoder, which allows reconstruction of input data from its latent representation.
- Denoising auto-encoders (add noise to input)
- Sparse auto-encoders (sparse hidden representation(s))

Transfer Learning

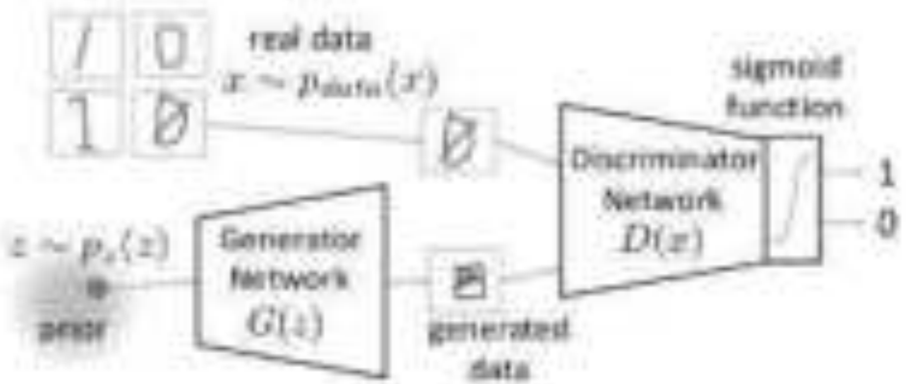
- TL refers to ability to generalize a pre-trained DNN to conditions that are different from those during training.
- ConvNet as fixed feature extractor: Take a ConvNet pretrained on ImageNet, remove the last fully-connected layers, then treat the remaining layers as a fixed deep feature extractor for the new dataset.
- Fine-tuning the ConvNet: Fine-tuning the weights of a pretrained network by continuing the backpropagation for a new task using a smaller number of training images is usually much faster and easier than training a network from scratch with randomly initialized weights. It is possible to fine-tune all layers or keep some of the earlier layers fixed and only fine-tune some higher-level portion of the network.
- Pre-trained Models: The Caffe library has a Model Zoo where people share their network weights

A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," arXiv, 12 May 2014.

KOÇ ÜNIVERSITESI

Generative Adversarial Networks

- GAN is an architecture that poses the training process as a game between two networks, a generator network and a discriminator net, against each other (thus "adversarial").



- The generator learns to generate realistic reconstructed solutions (samples) while the discriminator learns to determine if these samples are original data or reconstructed solutions.
- If we train both networks to equilibrium, then generated solution samples are indistinguishable from original data by a perfect discriminator.
- Adversarial learning enables learning entirely from data as opposed to relying on an engineered objective function to guide the optimization.

Ian Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks, arXiv.
https://www.analyticsvidhya.com/blog/2017/06/introductory-generative-adversarial-networks-gans/

GAN – Image generation

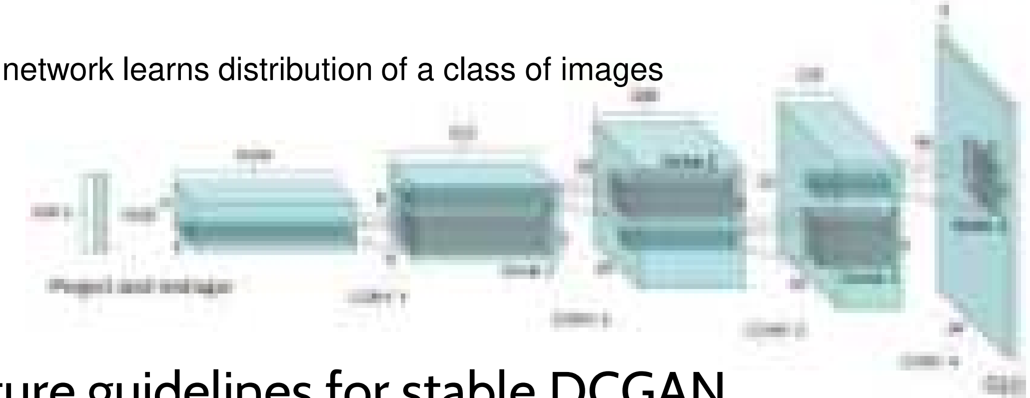
- Unsupervised: There are no ground truth labeled images, just sample images.
- Suppose we have a randomly-initialized image generator network that outputs 200 images, each from a different random code.
- We introduce a *discriminator* network (e.g., a standard CNN) to classify if an input image is real or generated. We feed 200 generated images and 200 real images into the discriminator and train it as a standard classifier to distinguish real and fake images.
- We backpropagate mismatch error through both discriminator and generator to find how we should change generator's parameters to make 200 generated samples slightly more confusing for the discriminator.
- Mathematically, we have a dataset of examples $x_1, \ldots, x_n$ as samples from a true data distribution p(x). Images generated by our network also have a distribution $\hat{p}_\theta\ (x)$ that is defined implicitly by taking points from a unit Gaussian distribution and mapping them through a deterministic neural network – our generative model that is a function of parameters θ. Tweaking these parameters will tweak the generated distribution of images. Our goal then is to find parameters θ that produce a distribution that closely matches the true data distribution (for example, by having a small KL divergence loss).
- GANs generate data in fine, granular detail; images generated by VAEs tend to be more blurred.

DCGAN

- Introduces convolutional networks into GAN architecture

network learns distribution of a class of images

Input: 100 random numbers drawn from uniform distribution



- Architecture guidelines for stable DCGAN
  - Remove fully connected hidden layers for deeper architectures.
  - Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
  - Apply batchnorm in both the generator and the discriminator. Applying batchnorm to all layers resulted in sample oscillation and model instability. This was avoided by not applying it to the generator output layer and the discriminator input layer.
  - Use ReLU activation in generator for all layers, except for the output layer, which uses Tanh.
  - Use LeakyReLU activation in the discriminator for all layers in contrast to the original GAN, which used maxout activation.

Ref.: A. Radford, L. Metz, S. Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, ICLR 2016.

KOÇ ÜNIVERSITESI

Tips for Training a GAN

- When training the discriminator, hold the generator constant; and when training the generator, hold the discriminator constant. Each should train against a static adversary.
- Pretraining the discriminator before you start training the generator will establish a clearer gradient.
- Each side can overpower the other.
  - If the discriminator is too good, it will return values so close to 0 or 1 that the generator will struggle to read the gradient.
  - If the generator is too good, it will exploit weaknesses in the discriminator that lead to false negatives. This may be mitigated by the nets' respective learning rates.
- Difficult to tune hyperparameters.
- GANs take a long time to train.

PART 2

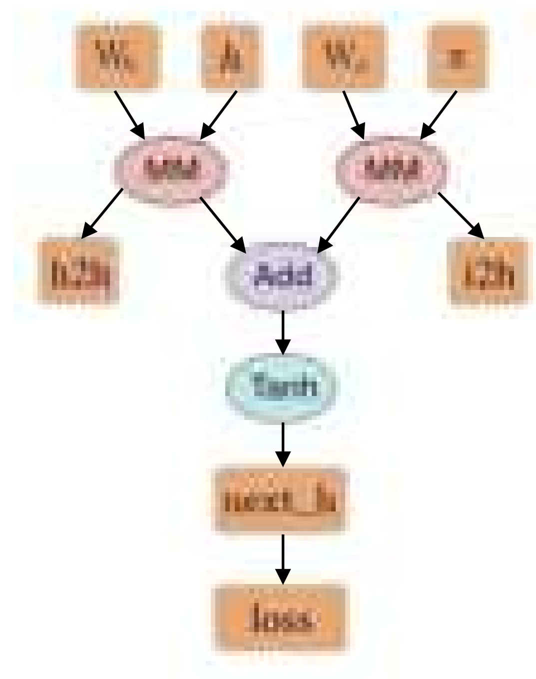
DEEP-LEARNED IMAGE and VIDEO PROCESSING

Frameworks

- Static Graph Methods:
  - **Tensorflow**
  - Theano
  - Mxnet
  - Caffe
- Dynamic Graph Methods:
  - **PyTorch**
  - Chainer
  - Tensorflow-Eager
  - DyNet

Static Graph

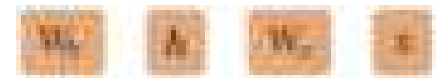
- A computational graph is a directed graph where the nodes correspond to operations or variables.
- Static graphs are defined first and then they are run. (Define-and-Run)



$$\mathbf{h}_t = \varphi(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

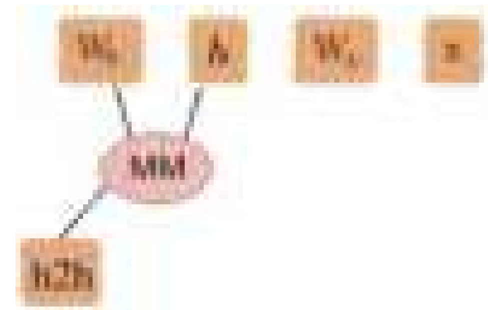
TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)
```



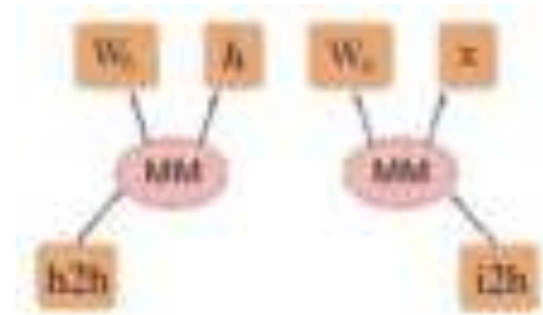
TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)

h2h = tf.matmul(W_h, tf.transpose(H))
```



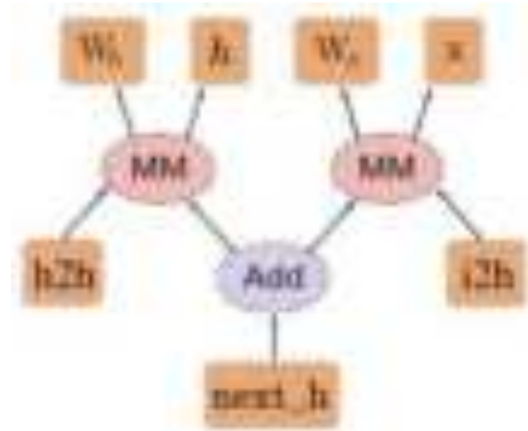
TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)

h2h = tf.matmul(W_h, tf.transpose(H))
i2h = tf.matmul(W_x, tf.transpose(X))
```



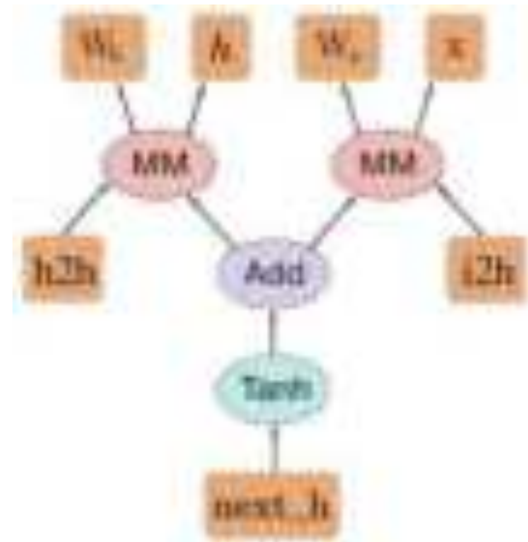
TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)

h2h = tf.matmul(W_h, tf.transpose(H))
i2h = tf.matmul(W_x, tf.transpose(X))
next_h = h2h + i2h
```



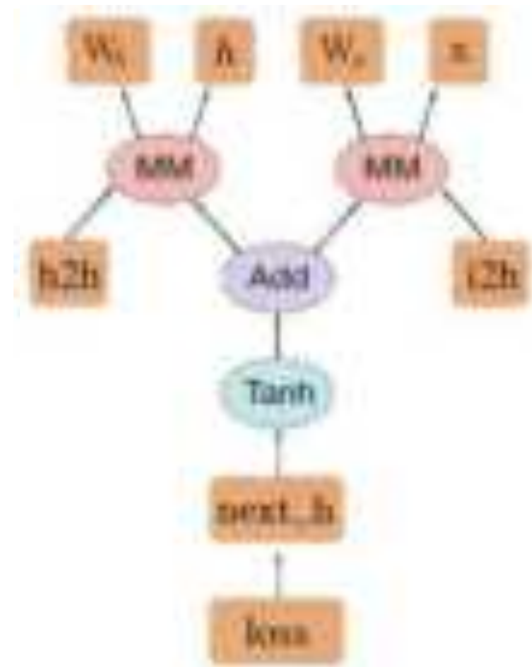
TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)

h2h = tf.matmul(W_h, tf.transpose(H))
i2h = tf.matmul(W_x, tf.transpose(X))
next_h = h2h + i2h
next_h = tf.tanh(next_h)
```



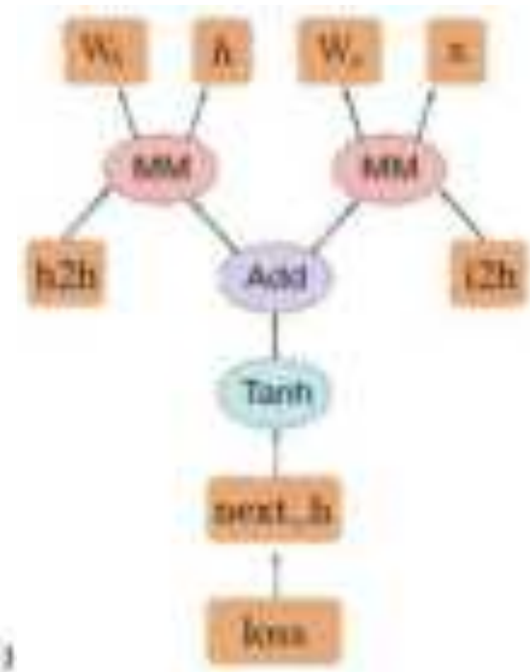
TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)

h2h = tf.matmul(W_h, tf.transpose(H))
i2h = tf.matmul(W_x, tf.transpose(X))
next_h = h2h + i2h
next_h = tf.tanh(next_h)

loss = tf.reduce_sum(next_h)
grad = tf.gradients(loss, [W_h, W_x])
```



TensorFlow implementation

```
W_h = tf.Variable(np.random.randn(20, 20), name="W_h")
W_x = tf.Variable(np.random.randn(20, 10), name="W_x")
X = tf.placeholder(tf.float32)
H = tf.placeholder(tf.float32)

h2h = tf.matmul(W_h, tf.transpose(H))
i2h = tf.matmul(W_x, tf.transpose(X))
next_h = h2h + i2h
next_h = tf.tanh(next_h)

loss = tf.reduce_sum(next_h)
grad = tf.gradients(loss, [W_h, W_x])

x = np.random.randn(1,10)
prev_h = np.random.randn(1,20)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    grads,h,l = sess.run([grad,next_h,loss], {X: x, H: prev_h})
```



Dynamic Graph

- In contrast, dynamic graph methods create the computational graph while running the code.
- Writing conditional statements and loops are natural.

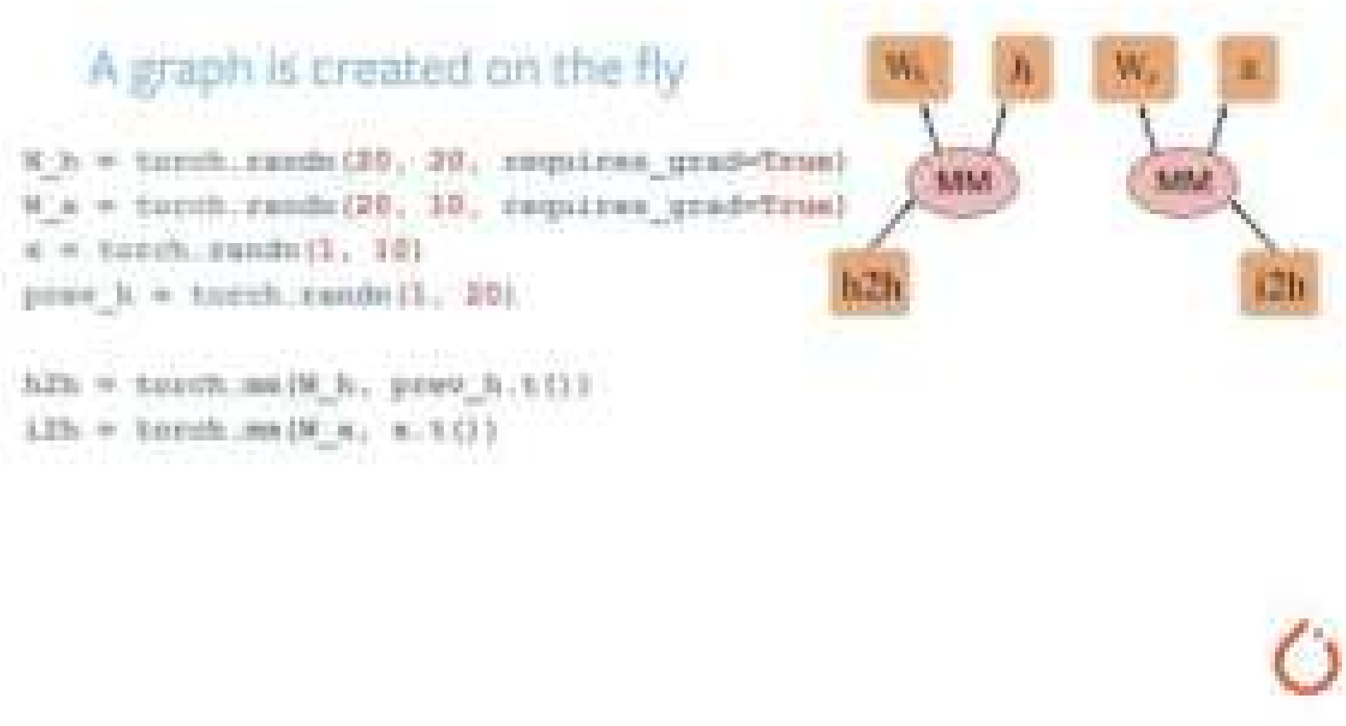
PyTorch Implementation

https://pytorch.org/about/

Dynamic Graph (PyTorch)

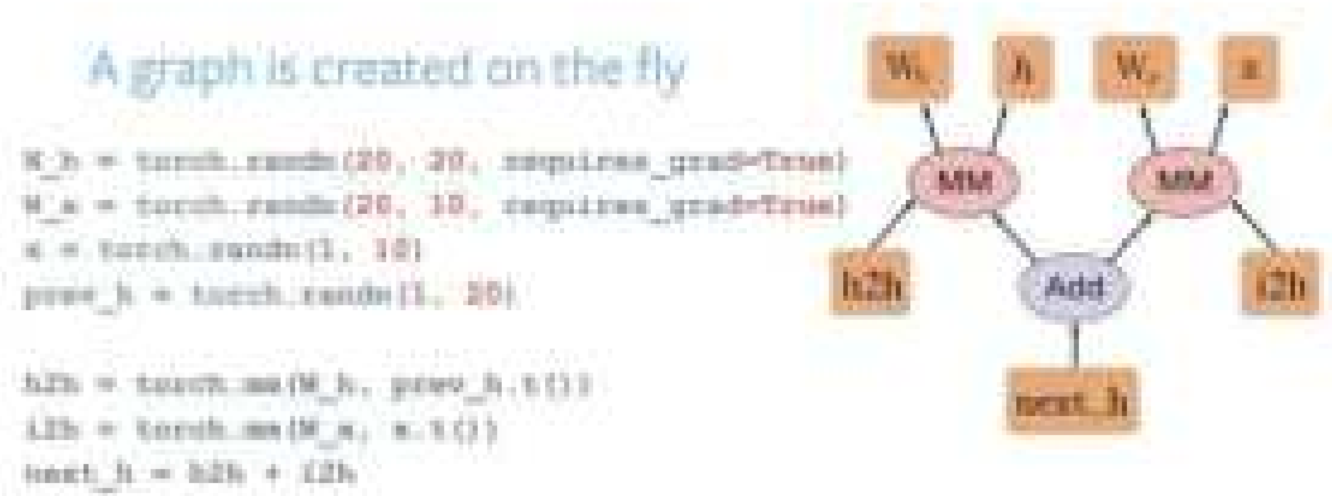
https://pytorch.org/about/

Dynamic Graph (PyTorch)



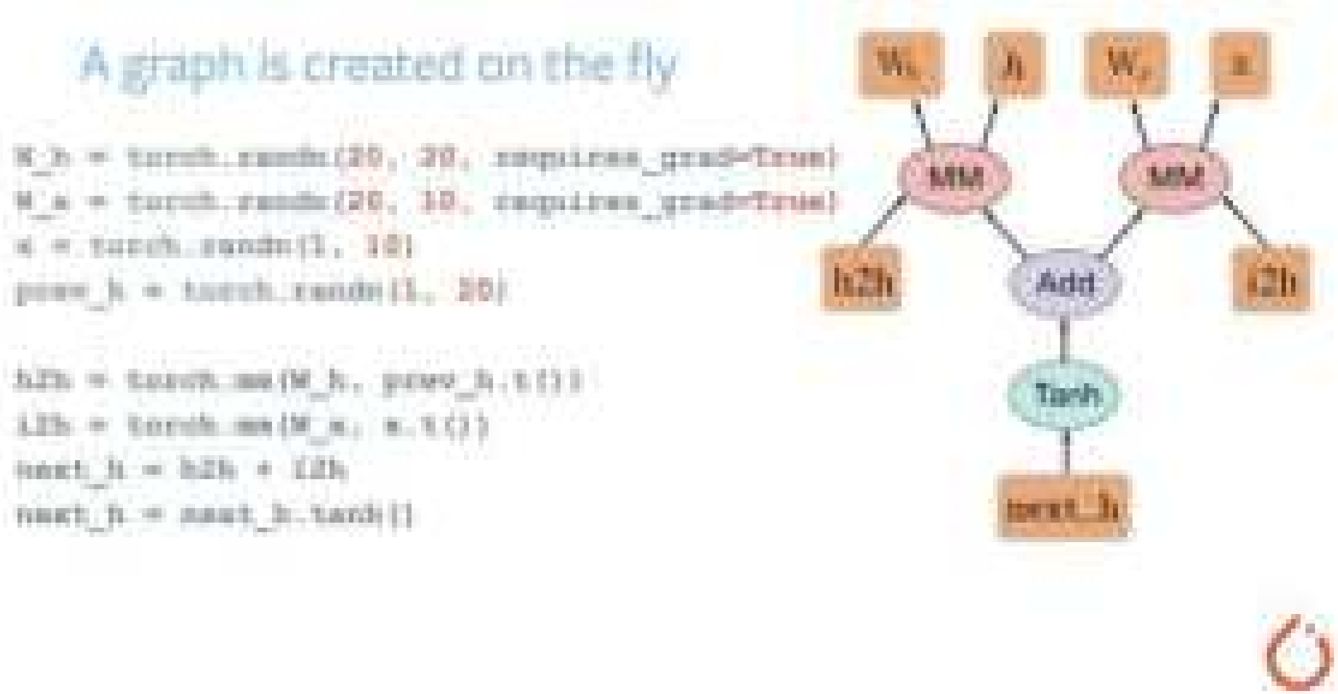
https://pytorch.org/about/

Dynamic Graph (PyTorch)



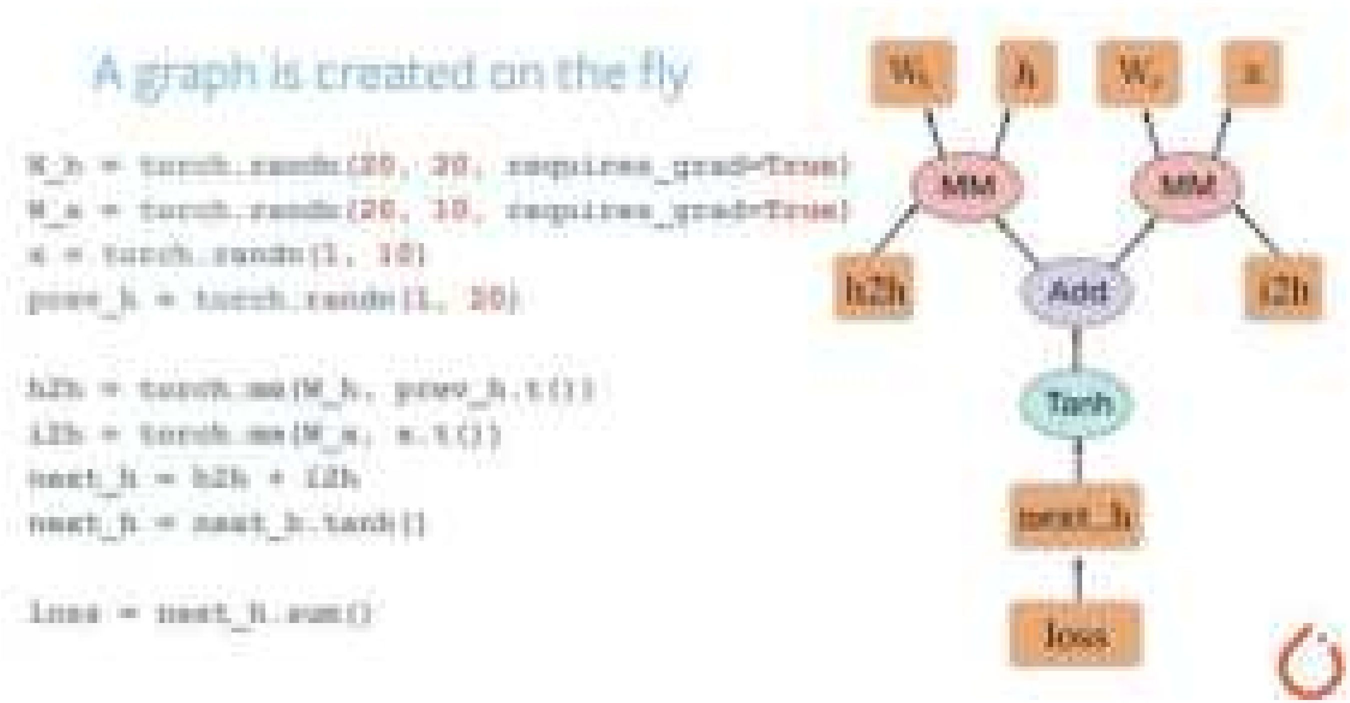
https://pytorch.org/about/

Dynamic Graph (PyTorch)



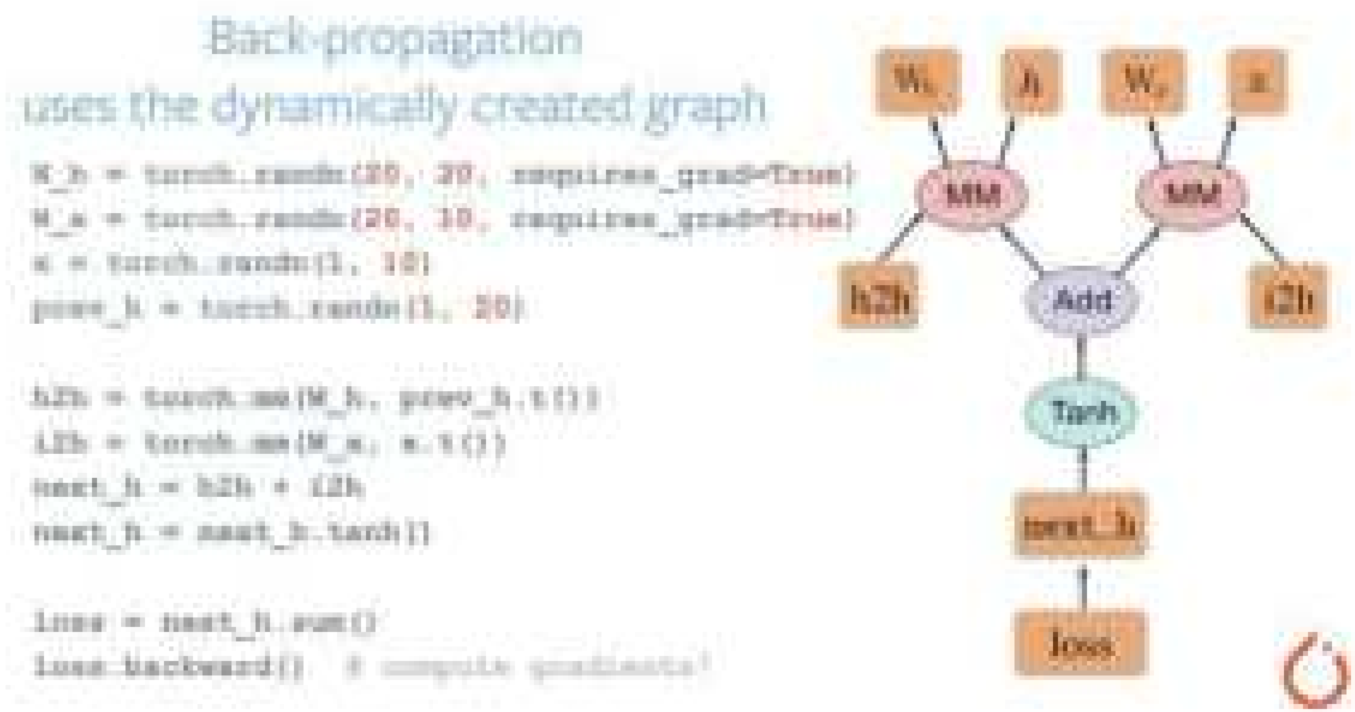
https://pytorch.org/about/

Dynamic Graph (PyTorch)



https://pytorch.org/about/

Dynamic Graph (PyTorch)



https://pytorch.org/about/

Comparison of Static vs. Dynamic

Static

- Define and run.
- Special control flow operations.
- Hard to debug. Use special tools.

Dynamic

- Define by run.
- Control flow is trivial.
- Easy to debug. Use standard debugging tools.

Example while loop

Tensorflow

```
cond = lambda t1, t2: tf.less(t1,t2)
body = lambda t1, t2: [tf.add(t1, 1), t2]
t1 = tf.constant(1)
t2 = tf.constant(5)

res = tf.while_loop(cond, body, [t1, t2])

with tf.Session() as sess:
    print(sess.run(res))
```

Pytorch

```
t1 = torch.tensor(1)
t2 = torch.tensor(5)

while t1 < t2:
    t1 = t1 + 1
```

Deep-Learned Solution of Inverse Problems

- Image Denoising
- Image Deblurring
- Single-image Super-resolution
- Image Inpainting

Inverse Problems

- Traditional Linear, Space-Invariant Image Degradation Model

$$\mathbf{y} = \mathbf{D}\mathbf{H}\mathbf{x} + \mathbf{v}$$

where $\mathbf{v}$ is noise, $\mathbf{H}$ is a convolution matrix with known blur kernel and $\mathbf{D}$ is an observation matrix.

  - *Image Denoising*
    - $\mathbf{D}$ and $\mathbf{H}$ are the identity matrix
  - *Image Deblurring*
    - $\mathbf{D}$ is the identity matrix, $\mathbf{H}$ is known
  - *Single Image Super-resolution* (SISR)
    - $\mathbf{D}$ is a down-sampling matrix , $\mathbf{H}$ is known
  - *Image Inpainting*
    - $\mathbf{H}$ is the identity matrix, $\mathbf{D}$ has some missing entries.

- New Trends in Image Restoration and Enhancement (NTIRE) Example-based single image super-resolution challenge

KOÇ ÜNİVERSİTESİ

Classic Image Deblurring

- Linear Space-Invariant (convolution)
  - Fourier Domain

$$H_W(e^{j\omega_1}, e^{j\omega_2}) = \frac{H^*(e^{j\omega_1}, e^{j\omega_2})}{|H(e^{j\omega_1}, e^{j\omega_2})|^2 + \sigma^2}$$

  - Space-Domain, Iterative
    - Landweber
    - POCS
- Linear Space-Varying (superposition)
  - Space-Domain Iterative, POCS
- Blind Image Restoration – limited to linear space-invariant model

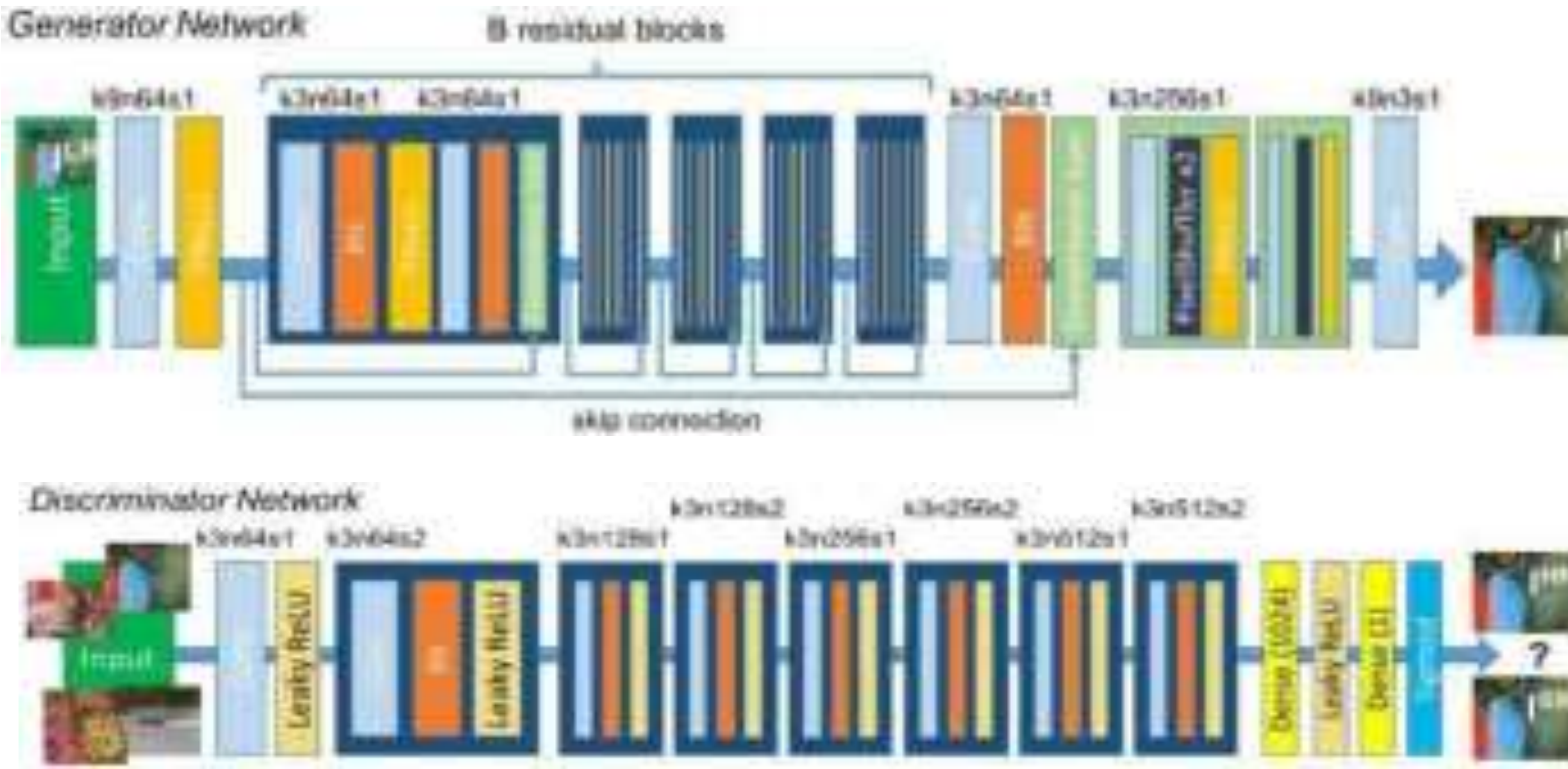
Deep Restoration/SISR Results

- DIV2K dataset (NTIRE 2017)

800 training,
100 test images
2K resolution

SRGAN

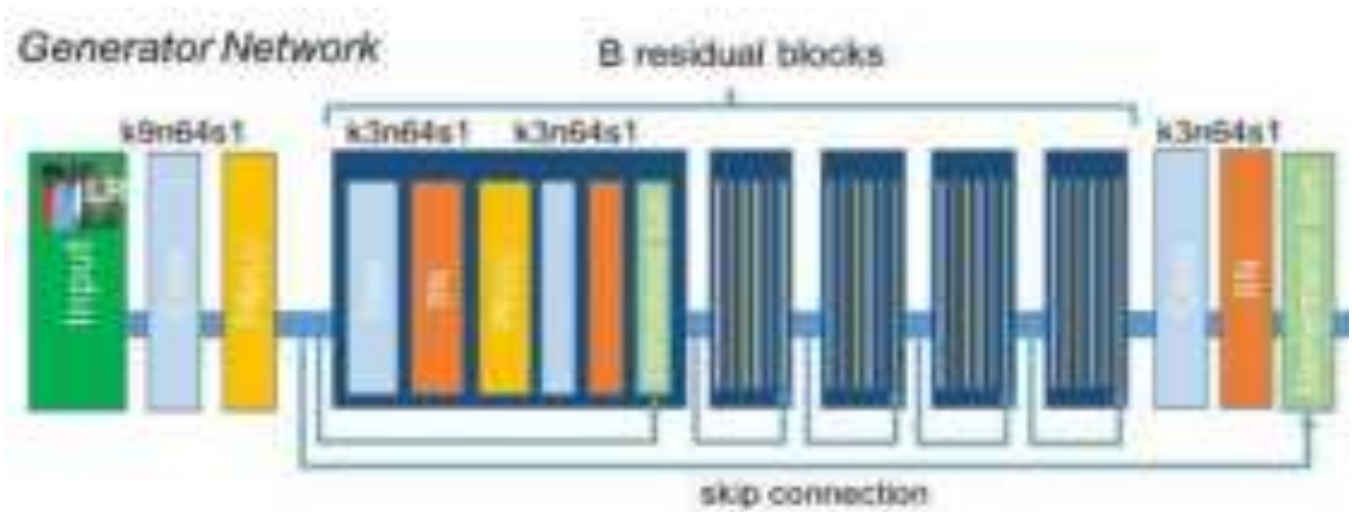
- Supervised training from noisy data
- SRResNET



C. Ledig, et al., “Photo-realistic single image super-resolution using a generative adversarial network,” arXiv, 13 April 2017

Image Deblurring using SRResNet without upscale layer

- Supervised training from blurred+noisy data at the input and the groundtruth image at the output



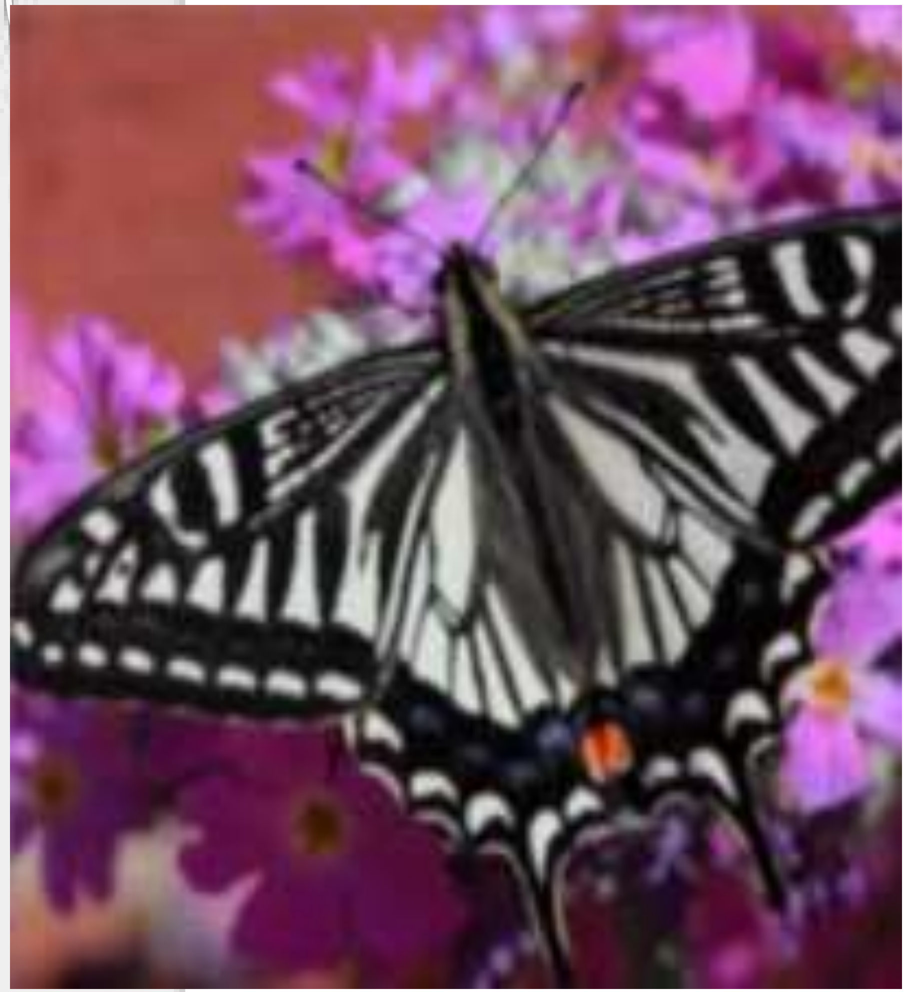
Modified from SRResNet

11x11 blurred + 40dB noise input

PSNR: 36.9666002257 dB, SSIM: 0.971302986145 – generator trained 96x96 patches, Minibatch size 16

KOÇ ÜNİVERSİTESİ

x4 SISR



Linear interpolated

Deep SR

x4 SISR

Linear interpolated

Deep SR

SRGAN x4 Results - Evaluation

- PSNR
- SSIM
- MOS (Perceptual quality)

Set5	nearest	bicubic	SRCNN	SelfExSR	DRCN	ESPCN	SRResNet	SRGAN	HR
PSNR	26.26	28.43	30.07	30.33	31.52	30.76	**32.05**	29.40	∞
SSIM	0.7552	0.8211	0.8627	0.872	0.8938	0.8784	**0.9019**	0.8472	1
MOS	1.28	1.97	2.57	2.65	3.26	2.89	3.37	**3.58**	4.32
Set14									
PSNR	24.64	25.99	27.18	27.45	28.02	27.66	**28.49**	26.02	∞
SSIM	0.7100	0.7486	0.7861	0.7972	0.8074	0.8004	**0.8184**	0.7397	1
MOS	1.20	1.80	2.26	2.34	2.84	2.52	2.98	**3.72**	4.32
BSD100									
PSNR	25.02	25.94	26.68	26.83	27.21	27.02	**27.58**	25.16	∞
SSIM	0.6606	0.6935	0.7291	0.7387	0.7493	0.7442	**0.7620**	0.6688	1
MOS	1.11	1.47	1.87	1.89	2.12	2.01	2.29	**3.56**	4.46

- How to stop training a GAN?

CVPR NTIRE 2017
(New Trends in Image Restoration and Enhancement)
Challenge on Single Image SR

- Two Tracks:
  - Track 1: Bicubic Downsampling
  - Track 2: Unknown Downsampling
- Three competitions
  - Upsample by x2, x3, x4
- New dataset: DIV2K (DIVerse 2k resolution images)
  - 800 training, 100 validation, 100 test images

R. Timofte, et al. NTIRE 2017 Challenge on Single Image Super-Resolution:Methods and Results, IEEE Conf. On Computer Vision and Pattern Recognition (CVPR) Workshops, July, 2017

NTIRE 2017 Results

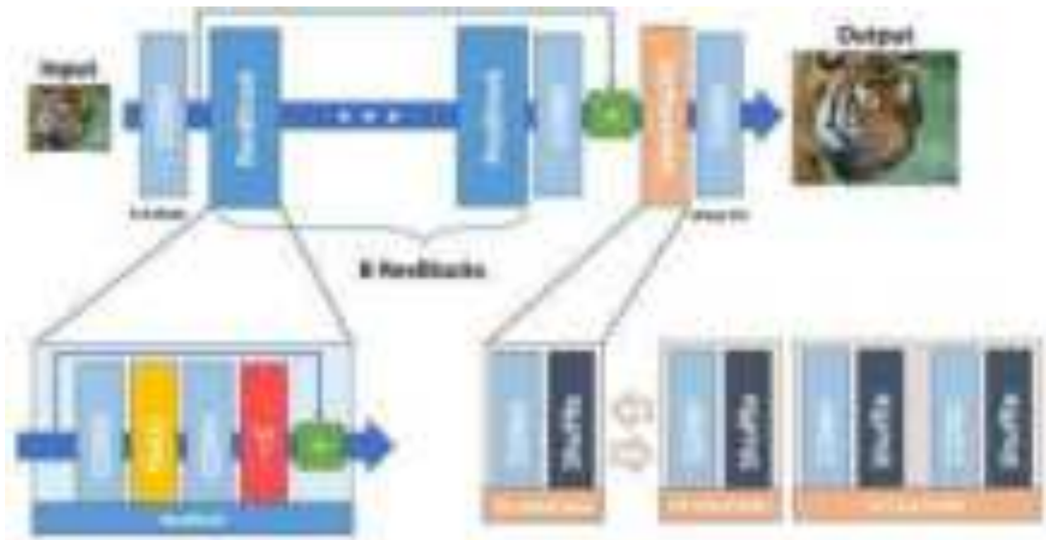
		Track 1: bicubic downscaling						Track 2: unknown downscaling					
		×2		×3		×4		×2		×3		×4	
Team	User	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
SNU_CVLab[1]	limbee	34.93	0.948	31.13	0.889	26.91*	0.752*	34.00	0.934	30.78	0.881	28.77	0.826
SNU_CVLab[2]	sanghyun	34.83	0.947	31.04	0.888	29.04	0.836	33.86	0.932	30.67	0.879	28.62	0.821
HelloSR	sparkfive	34.47	0.944	30.77	0.882	28.82	0.830	33.67	0.930	30.51	0.876	28.54	0.819
Lab402	iorism	34.66	0.946	30.83	0.884	28.83	0.830	32.92	0.921	30.31	0.871	28.14	0.807
VICLab	JSChoi	34.29	0.943	30.52	0.880	28.55	0.845						
UIUC-IFP	fyc0624	34.19	0.942	30.44	0.877	28.49	0.822	28.54	0.840	28.73	0.830	26.96	0.717
HIT-ULSee	chenxuanjia	34.07	0.941	30.21	0.871	28.49	0.822	33.40	0.927	30.21	0.871	28.30	0.812
I hate mosaic	[illegible]	34.05	0.940	30.47	0.878	28.59	0.824						
nicheng	nicheng									30.24	0.871	28.28	0.811
GTY	[illegible]	34.07	0.941	30.24	0.874	28.34	0.817	31.55	0.916	30.14	0.869	27.11	0.795
DL-61-86	[illegible]							31.33	0.922	30.09	0.867	28.07	0.805
faceall_Xlabs	xjc_faceall	33.73	0.937	30.07	0.869	27.39	0.805	24.08	0.707	29.87	0.862	26.84	0.762
SR2017	[illegible]	33.54	0.934	29.89	0.865	27.87	0.800	29.57	0.871	28.84	0.836	26.05	0.734
SDQ_SR	XibaSong	33.49	0.936					32.38	0.912				
HCILab	[illegible]	33.47	0.934	29.82	0.866	28.03	0.807	31.13	0.896	29.26	0.848	25.96	0.740
iPAL	[illegible]	33.42	0.932	29.89	0.865	27.99	0.806						
WSDSR	[illegible]	33.19	0.933	29.74	0.864	27.92	0.805						
Resonance	[illegible]							30.31	0.899	28.43	0.841	24.79	0.724
zrfanzy	zrfan	31.87	0.927	28.89	0.858	27.67	0.800	21.94	0.618	18.03	0.490	26.05	0.773
assafsho	assafsho	30.79	0.894	27.27	0.806	25.74	0.743						
UESTC-kb545	naiven											25.08	0.714
spectrum	spectrum							28.76	0.854				
Bicubic interp.	baseline	31.01	0.900	28.22	0.822	26.65	0.764	25.08	0.713	25.83	0.736	21.84	0.583

(*) the checked SNU_CVLab[1] model achieved 29.09dB PSNR and 0.837 SSIM.

Note: SNU_CVLab1 obtained 32.64 dB on set 5, 28.94 dB on set14 and 27.74 dB on BSD100 datasets compared to SRResNet which obtained 32.05 dB, 28.49 dB and 27.58 dB, respectively.

EDSR (SNU_CVLab[1])

- Winner of both tracks for all subsampling factors
- Modified from SRResNet with new building blocks:
  - Remove BN
  - Constant multiplication at the end (xC) (for better training)
- B=36 ResBlocks, 256 feature maps, C=0.1 (affects learning rate)

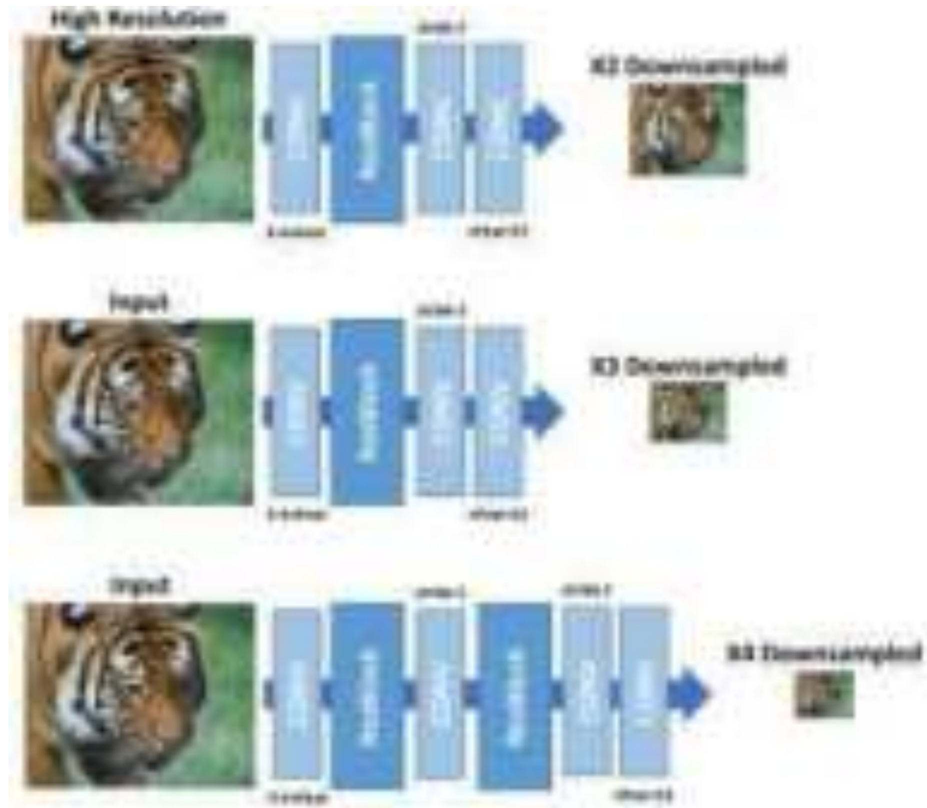


B. Lim, et al., Enhanced deep residual networks for single image super-resolution*, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshop, July, 2017

*Source code and model available on Github

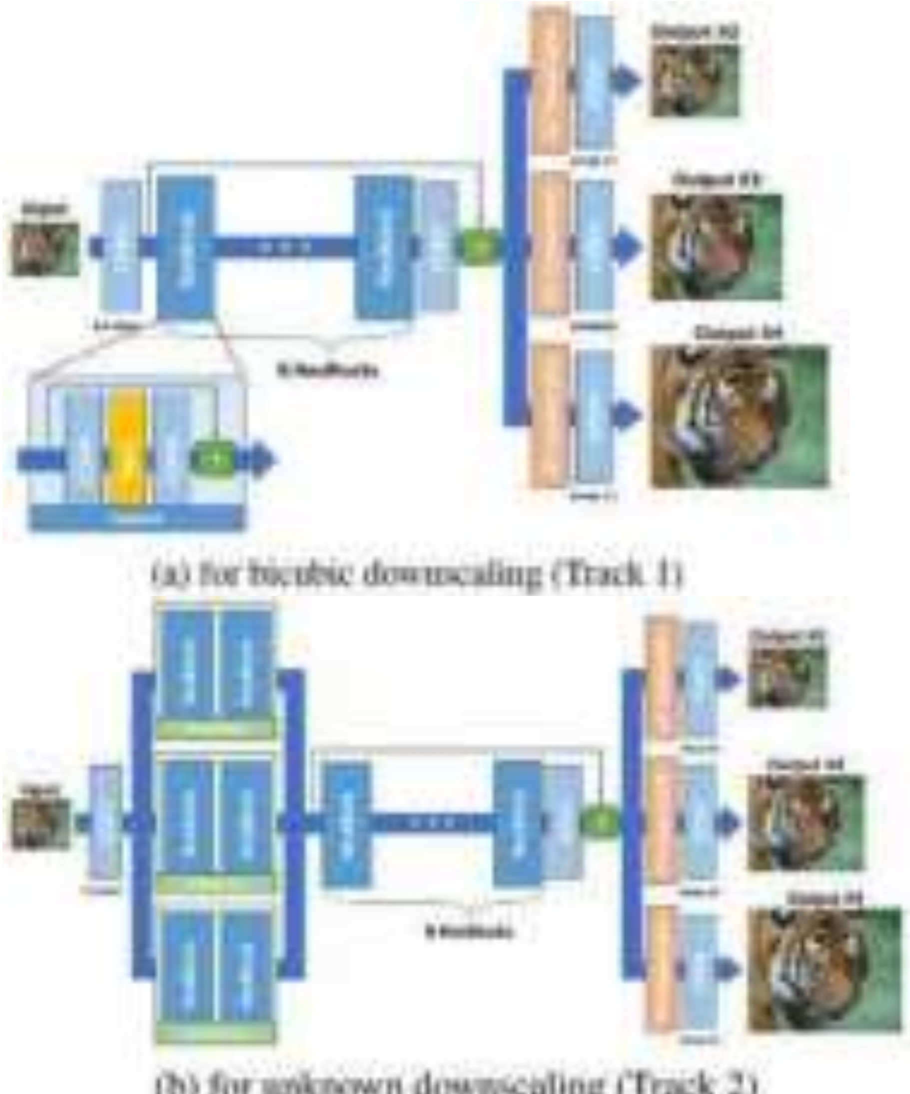
Unknown Degradation Model

- Learn degradation model using a NN from given (HR, LR) training data



- Data augmentation: Flip, rotate, etc. the original HR image, and generate synthetic LR images from them by applying the learned degradation model

MDSR (Multi-Scale EDSR) (SNU_CVLab[2])



(a) for bicubic downscaling (Track 1)

(b) for unknown downscaling (Track 2)

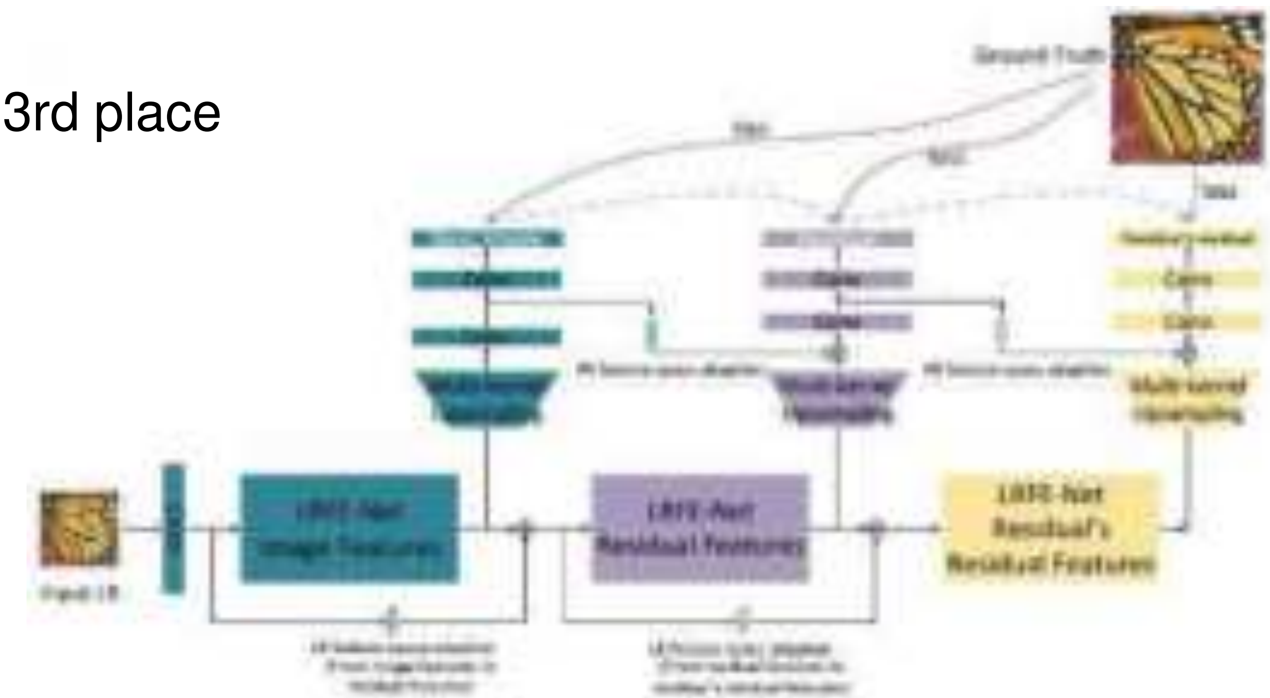
2nd place

- Single network for all three factors
- Deeper - 80 ResBlocks
- Narrower - 64 feature maps
- No constant scaling

*Source code and model available on Github

Stacked Residual-Refined Network (HelloSR)

3rd place



- Coarse-to-fine improvement
- Intermediate supervision
- LRFE-Net blocks consists of residual blocks

CVPR NTIRE 2018 Challenge on Single Image SR

- Four Tracks:
  - Classic bicubic downsampling (x8)
  - Realistic mild adverse conditions (x4)
  - Realistic difficult adverse conditions (x4)
  - Realistic wild conditions (x4)
- Realistic conditions emulate the image acquisition process from a digital camera.
- For Track 2 and 3 degradation model are the same within respective tasks.
- Track 4 has different degradation models from one image to another.

Challenge on SISR Results (Bicubic x8)

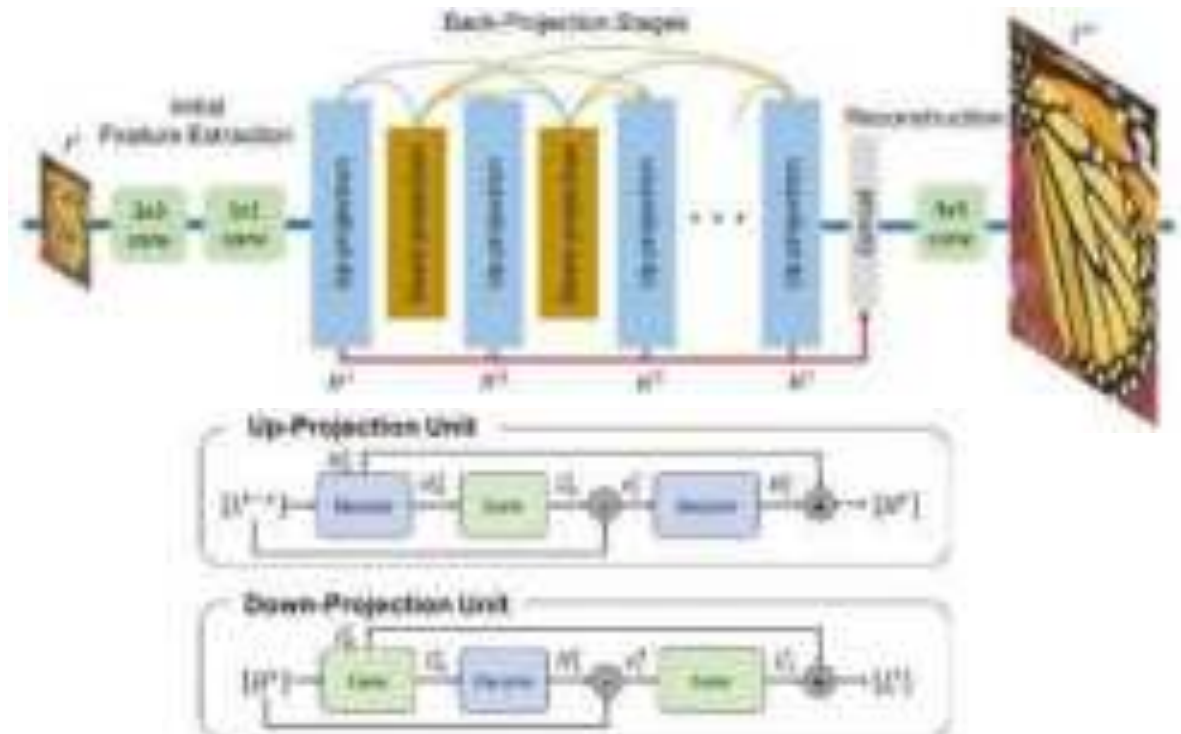
Team	Author	PSNR	SSIM
Toyota-TI	tim.lab	25.455	0.7088
Pixel_Overflow	McCourt_Hu	25.433	0.7067
rainbow	zheng222	25.428	0.7055
DRZ	yifita	25.415	0.7068
Faceall_Xlabs	xjc_faceall	25.360	0.7031
Duke Data Science	adrnian98	25.356	0.7037
UIUC-IFP	jhyume	25.347	0.7023
Haiyun_XMU	rr2018	25.338	0.7037
BMIPL_UNIST	BMIPL_UNIST	25.331	0.7026
Ajou-LAMDA-Lab	nmhkahn	25.318	0.7023
SIA	mikigom	25.290	0.7014
DeepSR	enoch	25.288	0.7015
	Mrobot0	25.175	0.6960
reveal.ai	muneebaadil	25.137	0.6942
HIT-VPC	cskzh	25.088	0.6943
MCML	ghgh3269	24.875	0.7025
BOE-SBG	boe_sbg	24.822	0.6817
SRFun	ccook	24.819	0.6829
KAIST-VICLAB	JSChoi	24.817	0.6810
	aiwcihe	24.773	0.6813
	jingliang	24.714	0.6913
CEERI	harshakoundinya	24.687	0.6719
APSARA	MingQiu	24.618	0.6817
UW18	zzsmg	24.192	0.6531
Baseline	Bicubic	23.703	0.6387

Challenge on SISR Results (Realistic x4)

		Track 2 Mild		Track 3 Difficult		Track 4 Wild	
Team	Author	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
UIUC-IFP	fbyune	$23.631_{(1)}$	0.6316	$22.329_{(1)}$	0.5721	$23.080_{(2)}$	0.6038
PDN	xixihaha					$23.374_{(1)}$	0.6122
BMIPL_UNIST	BMIPL_UNIST	$23.579_{(2)}$	0.6269	$22.074_{(2)}$	0.5590		
HIT-VPC*	lpj008			22.249	0.5637	22.879	0.5936
HIT-VPC	cskzh	$23.493_{(3)}$	0.6174	$21.450_{(9)}$	0.5339	$22.795_{(3)}$	0.5829
SIA	mikigom	$23.406_{(5)}$	0.6275	$21.899_{(3)}$	0.5623	$22.766_{(4)}$	0.6023
KAIST-VICLAB	jschoi	$23.455_{(4)}$	0.6175	$21.689_{(6)}$	0.5434	$22.732_{(6)}$	0.5844
DRZ	yifita	$23.397_{(6)}$	0.6160	$21.592_{(8)}$	0.5438	$22.745_{(5)}$	0.5881
srFans	yyuan13	$23.218_{(9)}$	0.6222	$21.825_{(4)}$	0.5573	$22.707_{(7)}$	0.5932
Duke Data Science	adamian98	$23.374_{(7)}$	0.6252	$21.658_{(7)}$	0.5400		
	bighead	$23.247_{(8)}$	0.6165				
ISP_Team	hot_milk	$23.098_{(11)}$	0.6167	$21.779_{(5)}$	0.5550	$22.496_{(8)}$	0.5867
BOE-SBG	boe_sbg	$23.123_{(10)}$	0.6008	$21.443_{(10)}$	0.5275	$22.352_{(10)}$	0.5612
MCML	ghgh3269	$22.951_{(12)}$	0.6115	$21.337_{(11)}$	0.5354	$22.472_{(9)}$	0.5842
DeepSR	enoch	$21.742_{(15)}$	0.5572	$20.674_{(16)}$	0.5168	$21.589_{(11)}$	0.5444
	jingliang	$21.710_{(16)}$	0.5384	$20.973_{(12)}$	0.5187	$20.956_{(14)}$	0.5214
Haiyun_XMU	cr2018	$21.519_{(17)}$	0.5313	$20.866_{(13)}$	0.5072	$21.367_{(13)}$	0.5321
Ajou-LAMDA-Lab	nmhkahn	$21.240_{(18)}$	0.5376				
Juanluisgonzales	juanluisgonzales	$22.625_{(13)}$	0.5868				
APSARA	mingqiu			$20.718_{(14)}$	0.4977		
NMH	nmh			$20.645_{(17)}$	0.4890		
	join16	$20.453_{(19)}$	0.4928				
Baseline	Bicubic	$22.391_{(14)}$	0.5336	$20.830_{(15)}$	0.4631	$21.761_{(12)}$	0.4989

Deep Back-Projection Network

- Winner of Track 1.
- Deep learning version of the well-known Iterative Back-Projection Method

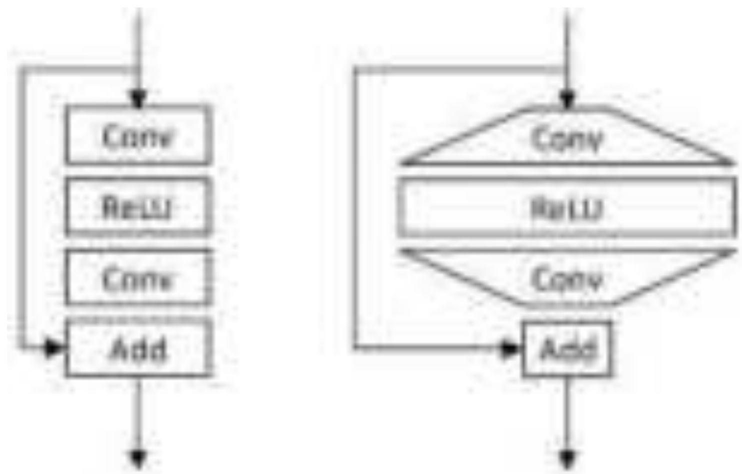


R. Timofte, *et al.*, NTIRE 2018 challenge on single image superresolution: Methods and results, CVPR 2018.

KOÇ ÜNİVERSİTESİ

Wide Activation and Weight Normalization for Accurate Image SR

- WDSR: Winner of Track 2-3. Second in Track 4.
- Modified EDSR.
- Weight normalization enables higher learning rates.

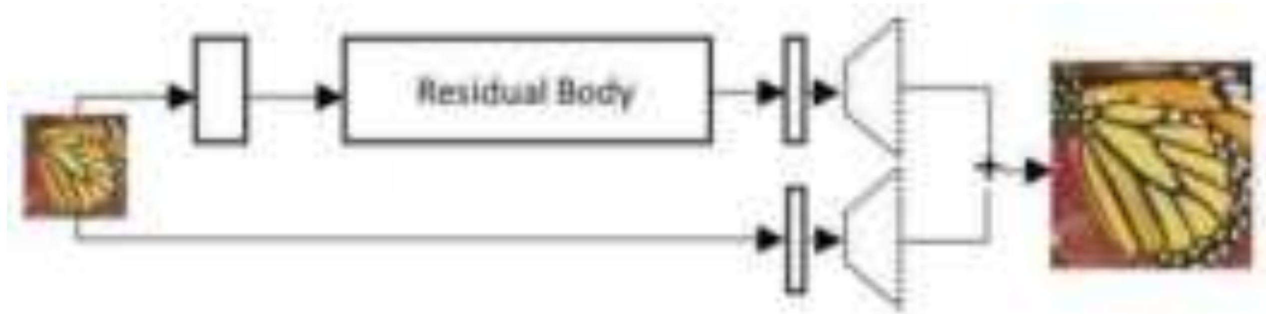


Comparison of Residual blocks in EDSR (left) and WDSR (right)

J. Yu, et al., Wide activation for efficient and accurate image superresolution, arXiv, 27 Aug. 2018.

WDSR (continued)

- Input image is upsampled with learned parameters.
- Convolutional layer before pixelshuffler is removed.



CVPR NTIRE 2018 Challenge on Image Dehazing

- First challenge on Image Dehazing
- 2 datasets for 2 tracks:
  - I-Haze: Indoor dehazing (35 scenes for training, 5 for validation)
  - O-Haze: Outdoor dehazing (45 scenes for training, 5 for validation)

Ground truth

Hazy

C. Ancuti, *et al.*, NTIRE 2018 challenge on image dehazing: Methods and results, CVPR 2018.

KOÇ ÜNIVERSITESI

Performance Evaluation

- Performance of restoration/SR methods vary over the test dataset
- Standard deviation of PSNR is in the same order as the PSNR
- PSNR depends on the frequency content of images in the dataset.

	Dataset	compared on RGB			compared on Y channel				
		MSE	PSNR	SSIM	MSE	Sal. MSE	PSNR	SSIM	Sal. SSIM
Training set	DIV2K Train	2.06e-3	25.70	0.8648	2.62e-3	2.62e-3	25.82	0.8688	0.0818
Test set	DIV2K Val	2.39e-3	25.86	0.8640	2.34e-3	2.34e-3	25.95	0.8679	0.0834
	GoPro	1.02e-3	29.91	0.9366	9.94e-4	7.52e-4	30.02	0.9380	0.0352
	Sun-Hays80	2.19e-3	26.60	0.8603	2.15e-3	1.86e-3	26.67	0.8633	0.0733
	BSD100	3.79e-3	24.21	0.7996	3.78e-3	2.92e-3	24.23	0.7988	0.1028
	Set5	1.72e-3	27.64	0.8993	1.58e-3	1.28e-3	28.01	0.9134	0.0227
	Set14	3.52e-3	24.54	0.8173	3.23e-3	2.22e-3	24.91	0.8305	0.1010
	Urban100	7.61e-3	21.19	0.7633	7.36e-3	6.21e-3	21.32	0.7678	0.0863
	Kodak	2.91e-3	25.37	0.8356	2.89e-3	2.27e-3	25.39	0.8381	0.0825

	Dataset	Resolution	Mean M_h	Var. M_h	Mean M_v	Var. M_v
Training set	DIV2K Train	Full HD	2251.5	5.4738e+06	3.8617e-3	2.975e-05
Test set	DIV2K Val	Full HD	2226.7	4.6755e+06	4.530e-3	5.4387e-05
	GoPro	HD	274.25	8456.5	1.262e-3	1.4337e-07
	Sun-Hays80	HD	520.86	1.3307e+05	2.991e-3	5.4475e-06
	BSD100	SD	226.8	23911	7.189e-3	3.1625e-05
	Set5	SD	86.842	4305.7	3.561e-3	2.1294e-06
	Set14	SD	291.91	53827	5.278e-3	1.3152e-05
	Urban100	SD	459.93	95847	1.010e-2	4.7518e-05
	Kodak	SD	392.86	82041	4.952e-3	1.2454e-05

O. Kırmemiş and A.M. Tekalp, Effect of training and test datasets on image restoration and super-resolution by deep learning, EUSIPCO 2018. (Tuesday 14:30)

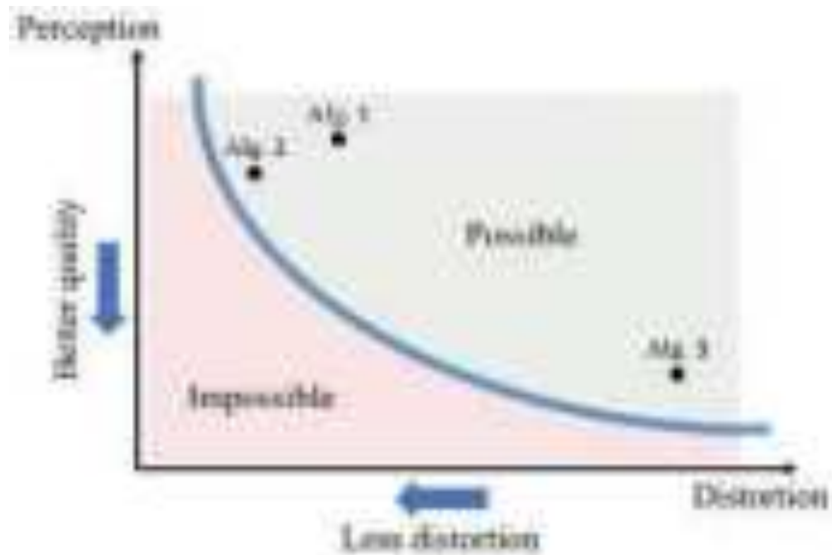
KOÇ ÜNIVERSITESI

ECCV 2018 Challenge on Perceptual Image Restoration and Manipulation

- Definition of Perceptual Quality
- Three topics:
  - Enhancement on Smartphnoes: Focuses on SR and image enhancement in mobile devices. Metric is accuracy per runtime. Also constraints on max. model size and max. RAM consumption
  - Super Resolution: Focuses on perceptual quality. Perceptual quality is compared within predefined regions according to thresholds on MSE.
  - Spectral Reconstruction

The Perception-Distortion Tradeoff

- There exists a region in P-D plane which is unattainable.
- If the performance of an algorithm is along the blue curve, it can be improved only in terms of distortion or in terms of its perceptual quality, but not in both.



Y. Blau and T. Michaeli. The Perception-Distortion Tradeoff. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2018.

Predicting Perceptual Quality

- Good PSNR does not guarantee better perceptual quality.

- A no-reference IQA metric that predicts MOS to evaluate the performance of SR algorithms

  C. Ma, et al. Learning a no-reference quality metric for single-image super-resolution. Comp. Vision and Image Understand (CVIU), 2017.

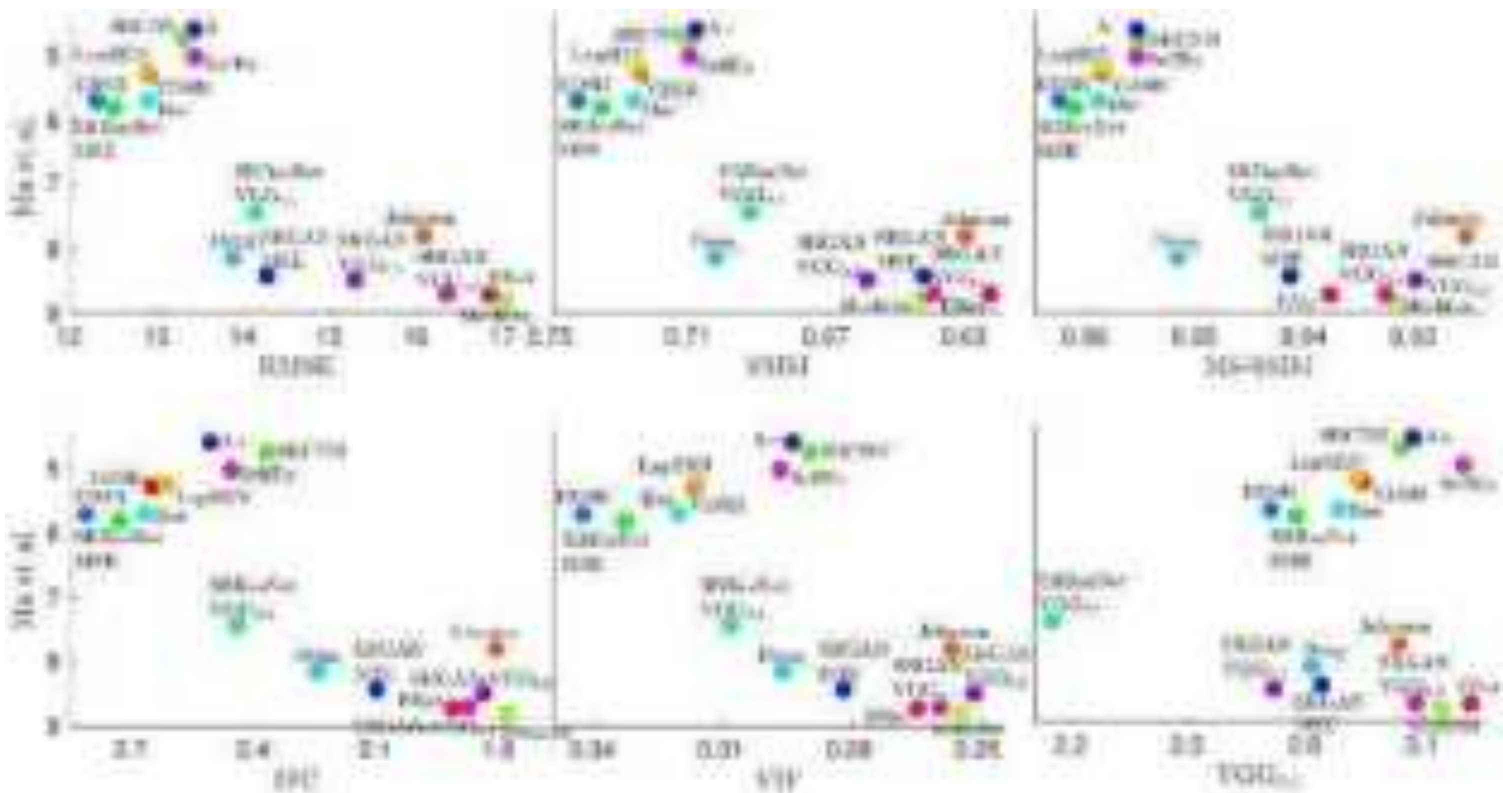
- Natural Image Quality Evaluator (NIQE)

  A. Mittal, et al. Making a completely blind image quality analyzer. IEEE Signal Processing Letters, vol. 20, no. 3, pp. 209-212, March 2013.

KOÇ ÜNIVERSITESI

Perception-Distorton Evaluation of SR algorithms

- The location of an algorithm on the P-D plane depends on the distortion metric.



Deep-Learned Image/Video Compression

- End-to-end Image Compression
  - Auto-encoder
  - Generative Compression
- Enhancing performance of standards-based encoders
  - HEIF (BPG) Encoder
  - HEVC Encoder

KOÇ ÜNİVERSİTESİ

End-to-end Image Compression

- Learned-transform
  - Auto-encoders learn latent-space representation of images
- Differentiable approximation to quantization
  - Soft quantization
- Generative Codecs
  - O. Rippel and L. Bourdev, “Real-time adaptive image compression,” ICML 2017, arXiv, 16 May 2017.
  - S. Santurkar, D. Budden, and N. Shavit, “Generative compression, ” arXiv, June 2017.

KOÇ ÜNİVERSİTESİ

Soft Quantization

- Hard quantization for d-bits:
  - $q = Q(z) = \lfloor z \times 2^d \rfloor$
- However this function yields zero gradients except at decision boundaries. Therefore soft quantization is employed in training phase.

- Soft quantization for d-bits:
  - $\tilde{z} = \sum_{i=0}^{2^d-1} \frac{\exp(-\|z \times 2^d - i\|)}{\sum_{j=0}^{2^d-1} \exp(-\|z \times 2^d - j\|)} \times i$

E. Agustsson, et al. Soft-to-hard vector quantization for end-to-end learning compressible representations. arXiv preprint arXiv:1704.00648, 2017

Enhancing Standard Codecs

- Deep networks learn free parameters of state of the art standards-based encoders
  - Block partitioning
  - Mode selection
  - Quantization parameter selection
  - In-loop filter
- Pre-processing and/or post-processing
  - Learned smoothing for pre-processing
  - Artifact removal

CVPR-CLIC 2018
Challenge on Learned Image Compression

- Rules
  - Compression rate of the whole test set must not exceed 0.15 bpp (average).
  - Participants are ranked according to
    - PSNR
    - Scores provided by human raters (MOS)
- Dataset
  - New: 1633 training, 102 validation, 286 test images.
    - DatasetP (professional)
    - DatasetM (mobile)

CLIC 2018 Winners

- Best MOS (also best MS-SSIM)
  - An Autoencoder-based Learned Image Compressor: Description of Challenge Proposal by NCTU
- Best PSNR
  - CNN-Optimized Image Compression with Uncertainty based Resource Allocation
- Fastest:
  - xvc codec

CLIC 2018 Results

- Only submissions which are evaluated for MOS scores are shown.

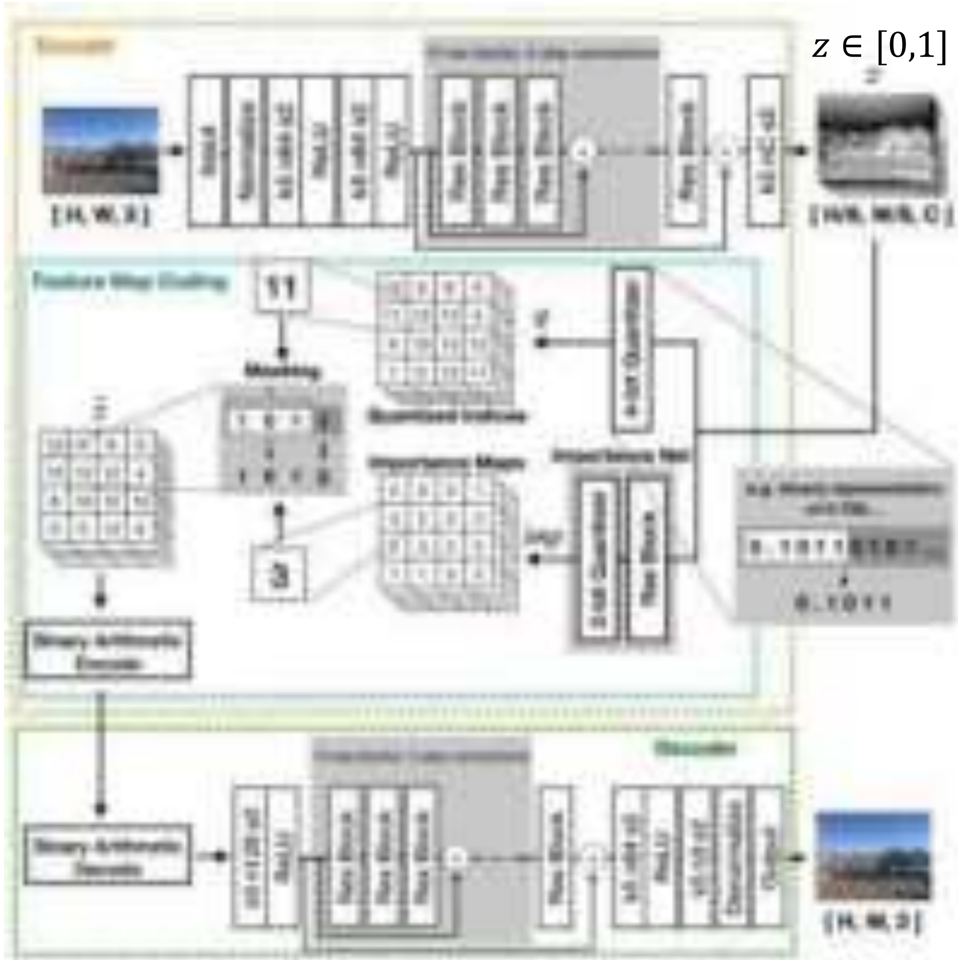
Entry	PSNR	MOS	MS-SSIM	Images Size (bytes)	Decoding Time (ms)	Decoder Size (bytes)
TucodecTNGcomp	[illegible]	[illegible]	0.964	[illegible]	[illegible]	[illegible]
iic	[illegible]	[illegible]	0.954	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
TucodecTNG	[illegible]	[illegible]	0.955	[illegible]	[illegible]	[illegible]
yfan	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Amnesiack	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
MVGL	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
BPG	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
JPEG	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Fastest

- xvc – A conventional codec
  - proprietary
  - Block based
  - Traditional approach for prediction, residual representation
- Originally developed for video compression.
- No machine learning is involved.

J.Samuelsson, P. Hermansson, Image compression with xvc, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June, 2018

Best MOS and MS-SSIM



- Based on autoencoder
- 4-bit quantization
- Soft quantization
- Importance Network

D. Alexandre, et al., An autoencoder-based learned image compressor: Description of challenge proposal by NCTU, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June, 2018

Importance Net

- Importance Net learns the important parts of the representation so that the system allocates more bits to complicated areas.
- It is made of residual blocks and another quantizer to select the number of bits

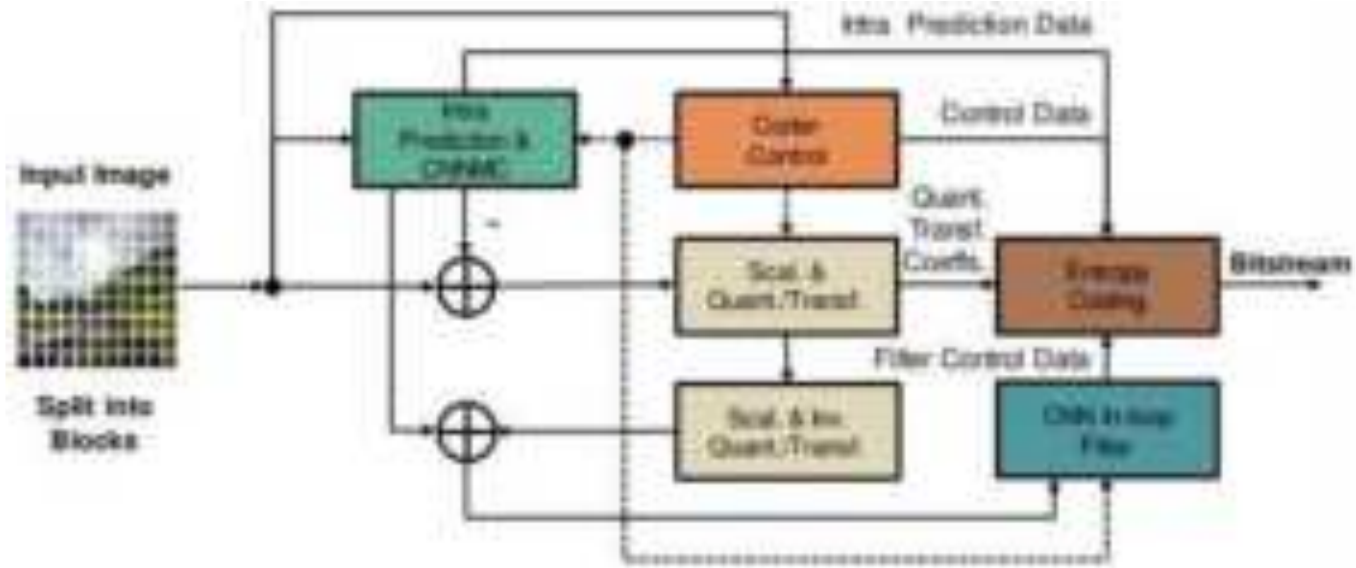
Optimization - Loss Function

- Loss function is a weighted sum of rate and distortion
  - $L = \lambda \times H(imp) + L_d$

  where

  - $L_d = \frac{MSE}{2\sigma_1^2} + \frac{MSSSIM}{2\sigma_2^2} + \log(\sigma_1^2) + \log(\sigma_2^2)$
- Rate loss H(imp) is estimated by summing up all values of the importance maps imp
- σ_1 and σ_2 are learnable parameters

Best PSNR

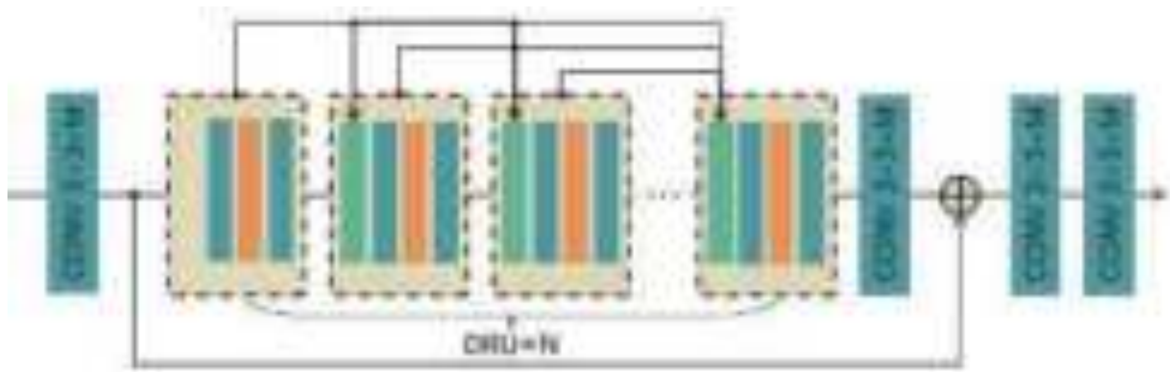


- Based on the JEM platform (the HEVC codec)
- Contributions:
  - CNN based in-loop filter (CNNIF) and
  - CNN based mode coding (CNNMC)

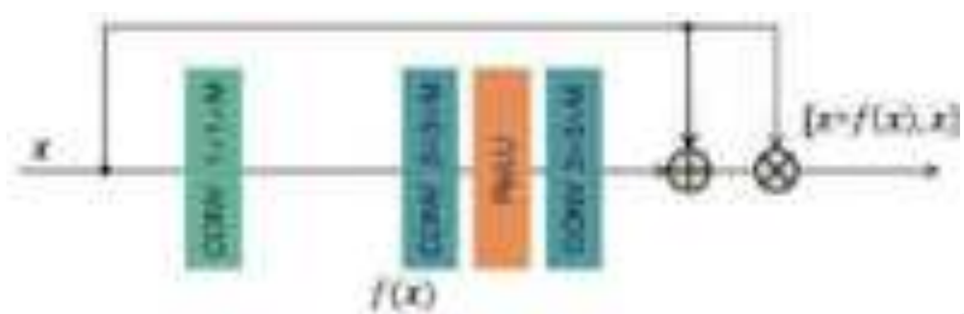
Z. Chen, et al., CNN-optimized image compression with uncertainty based resource allocation, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June, 2018

CNN based In-Loop Filter (CNNIF)

- In-loop filter consists of stacked Dense Residual Units (DRU)



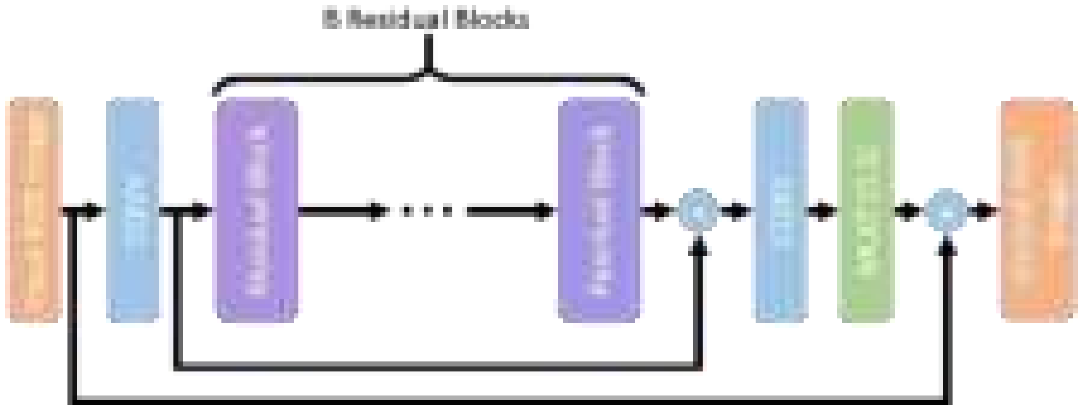
A Dense Residual Unit (DRU)



KOÇ ÜNİVERSİTESİ

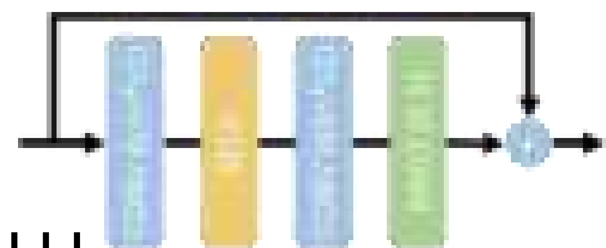
Learned Artifact Suppression

- Compression artifacts have structure that can be learned



EDSR Network

Each residual block



- We use SELU activation instead of ReLU
- It is trained to remove artifacts introduced by BPG codec.
- Although we trained our network with a single QP (40), it can improve images encoded by QP between 39 and 43.

O. Kirmemis, G. Bakar and A.M. Tekalp, Learned Compression artifact removal by deep residual networks, IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2018

Questions ?

Lunch Break

home.ku.edu.tr/~mtekalp