ResNet for Image Classification (MNIST)

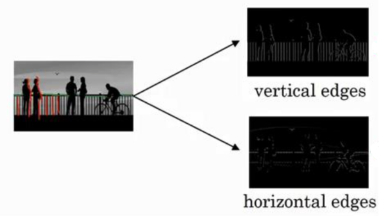
Elif Yılmaz

Boğaziçi University

27.06.2021

Convolutional Neural Networks

- CNNs generally uses in computer vision problems (image classification, object detection, neural style transfer, etc.)
- Edge detection



- Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

3 x 3

=

-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4 x 4

- Horizontal edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

$*$

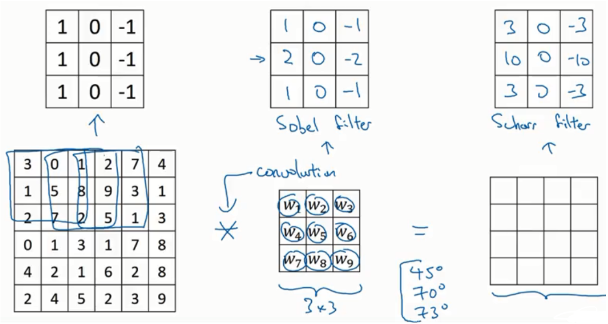
1	1	1
0	0	0
-1	-1	-1

$=$

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

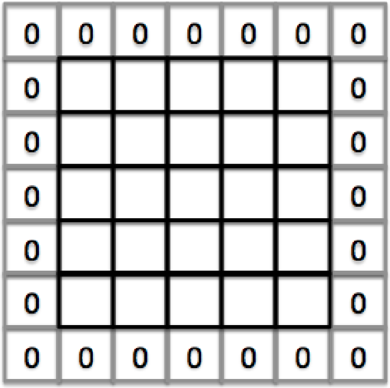
Edge Detection

- To detect edges, we need to filters.



Padding

- To prevent loosing information on corners of the image, we can use padding. In general, we use zero padding.



Valid and Same Convolution

- A **valid** convolution is a type of convolution operation that does not use any padding on the input. For an nxn input matrix and an fxf filter, a valid convolution will return an output matrix of dimensions:

$$\left\lfloor \frac{n-f}{s} + 1 \right\rfloor x \left\lfloor \frac{n-f}{s} + 1 \right\rfloor$$

  where s represents stride.

- A **same** convolution is a type of convolution where the output matrix is of the same dimension as the input matrix. For an nxn input matrix and an fxf filter, a valid convolution will return an output matrix of dimensions:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor x \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

  where s represents stride and p is padding.

Pooling

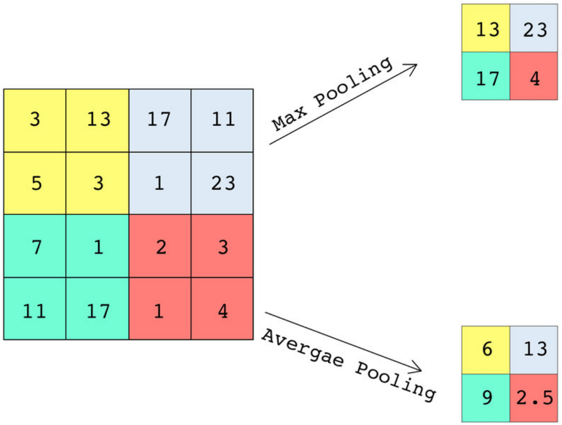
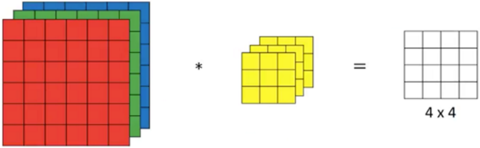


Figure: Pooling for 2x2 filter and 2 stride value

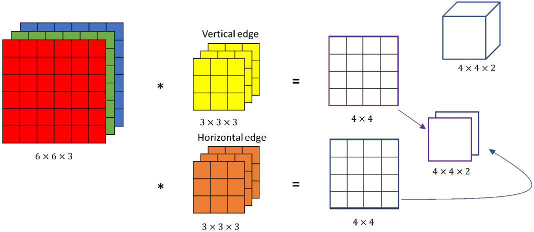
Note: No parameters to learn!

Convolutions over Volumes

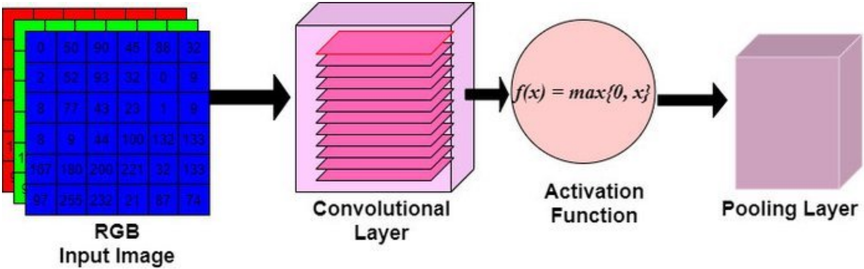
- Convolutions over RGB images



- Multiple filters



Example of a Layer in CNNs



Summary of Notation

If layer l is a convolution layer:

- $f^{[l]}$: filter size
- $p^{[l]}$: padding
- $s^{[l]}$: stride
- $n_c^{[l]}$: number of filters

$$\Downarrow$$

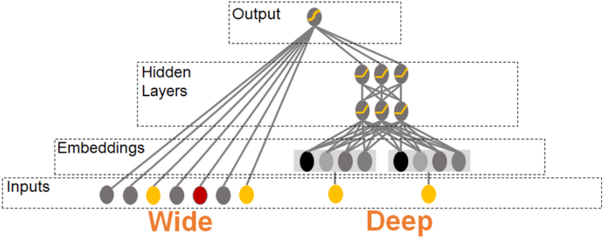
- Input: $n_H^{[l-1]}$ x $n_W^{[l-1]}$ x $n_C^{[l-1]}$
- Each filter is: $f^{[l]}$ x $f^{[l]}$ x $n_C^{[l-1]}$
- Activation: $a^{[l]}$: $n_H^{[l]}$ x $n_W^{[l]}$ x $n_C^{[l]}$
- Weights: $f^{[l]}$ x $f^{[l]}$ x $n_C^{[l-1]}$ x $n_C^{[l]}$
- Bias: 1 x 1 x 1 x $n_C^{[l]}$
- Output: $n_H^{[l]}$ x $n_W^{[l]}$ x $n_C^{[l]}$

Why convolutions?

- Parameter sharing: A feature detector that is useful in one part of the image is useful in another part of the image.
- Sparsity of connections: In each layer, each output value depends only on a small number of inputs.

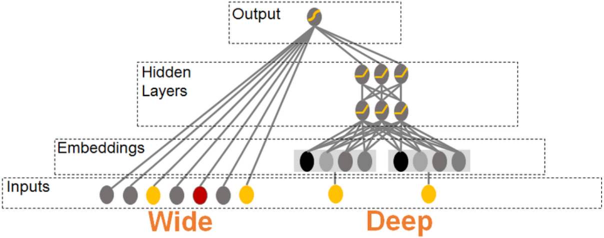
Why deeper networks work better?

- The "levels" of features can be increased by increasing depth.
- Less parameters to optimize than wider networks.
- Could be more generalizable than wider networks.



Why deeper networks work better?

- The "levels" of features can be increased by increasing depth.
- Less parameters to optimize than wider networks.
- Could be more generalizable than wider networks.



However, working with deeper neural networks are not that easy because:

- The vanishing/exploding gradients inhibits convergence.
- Degradation (of training accuracy) problem.

ResNet Block

- In a plain network, the output of l^{th} layer calculates as

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l-1]} \longrightarrow a^{[l]} = g(z^{[l]})$$

where $W^{[l]}$ is weight matrix in l^{th} layer, $b^{[l-1]}$ is bias parameter in $(l-1)^{th}$ layer, g is activation function and $a^{[l]}$ is output of l^{th} layer.

ResNet Block

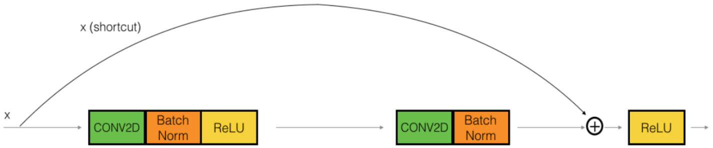
- In a plain network, the output of l^{th} layer calculates as

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l-1]} \longrightarrow a^{[l]} = g(z^{[l]})$$

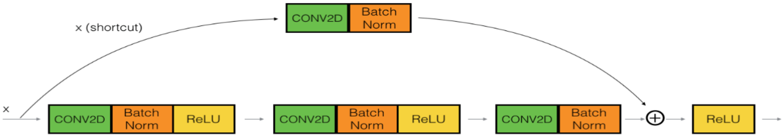
where $W^{[l]}$ is weight matrix in l^{th} layer, $b^{[l-1]}$ is bias parameter in $(l-1)^{th}$ layer, g is activation function and $a^{[l]}$ is output of l^{th} layer.

- In a residual network, we calculate the output of l^{th} layer by using

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l-1]} \longrightarrow a^{[l]} = g(z^{[l]} + a^{[l-2]})$$



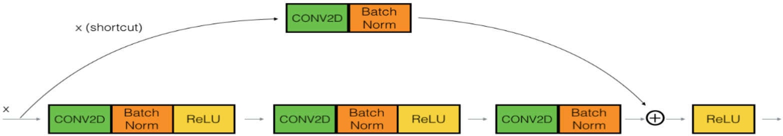
ResNet Block



- We can apply convolution or extra operations(e.g. batch normalization) to residual block.

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l-1]} \longrightarrow a^{[l]} = g(z^{[l]} + W_s a^{[l-2]})$$

ResNet Block



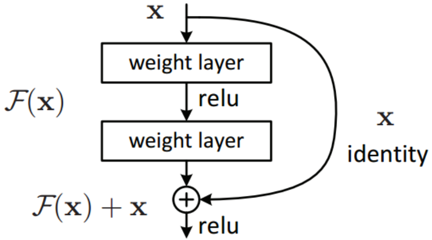
- We can apply convolution or extra operations(e.g. batch normalization) to residual block.

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l-1]} \longrightarrow a^{[l]} = g(z^{[l]} + W_s a^{[l-2]})$$

- Even if $z^{[l]} = 0$, we get a non-zero output. Therefore, we get rid of vanishing gradient problem.

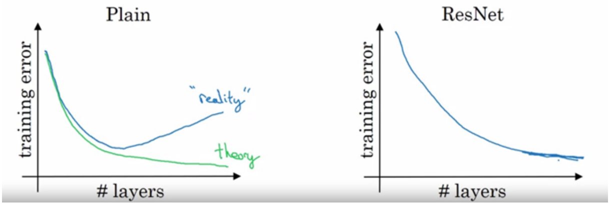
ResNet (2015)

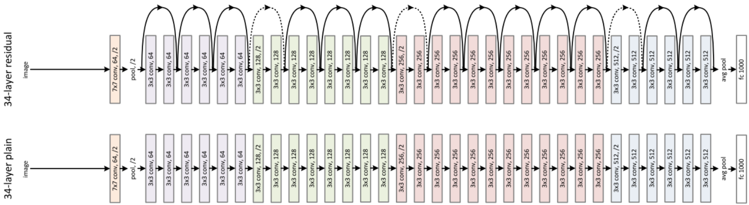
- Solves the degradation problem.
- Applicable to both vision and non-vision problems.
- Uses shortcut connections without adding extra parameter and complexity.



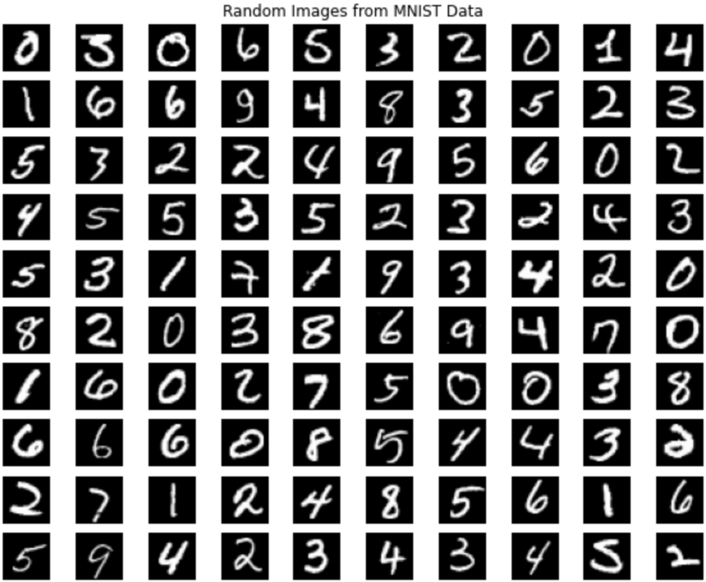
ResNet (2015)

- Prevents vanishing gradient problem.
- Having a shortcut connection to produce identity mapping, rather than waiting it to learn from scratch.
- Allows us to go deeper in the network.





MNIST dataset



Model for MNIST Classification

- The model composes of basically 6 layers.
- First layer has convolution operation, batch normalization, ReLU activation function and max pooling.
- Second layer has the same operations with first layer; however, here we add a residual block.
- In residual block, we perform the following operations, respectively: convolution operation, batch normalization, ReLU activation function, convolution operation and batch normalization.
- Other layers are similar to second layer.
- The last layer is fully connected layer which has 10 groups as outputs.
- We calculate the model error by using cross entropy loss.
- After 10 epochs, the accuracy is 0.99 and the loss 0.0001. Also, the loss decreases during epochs.

Variational Autoencoders

- VAE is a generative model. Generative models are used for data generation, data imputation, denoising, etc.
- VAE learns an encoding distribution during the training in order to generate new data.
- The term "variational" comes from variational inference. Variational inference is a method of approximating probability densities through optimization.

Variational Autoencoders

- VAE is a generative model. Generative models are used for data generation, data imputation, denoising, etc.
- VAE learns an encoding distribution during the training in order to generate new data.
- The term "variational" comes from variational inference. Variational inference is a method of approximating probability densities through optimization.
- VAE encodes a distribution over the latent space instead of encoding the input as a single point and the latent space allows us to generete new data.
- We can generate new data by decoding points that are randomly sampled from the latent space. The quality and relevance of generated data depend on the regularity of the latent space.
- VAE can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.

Variational Autoencoders Training Procedure

1. Input is encoded as a distribution over the latent space
2. A point is sampled from that distribution
3. The sampled point is decoded and the reconstruction error is computed.

VAE Architecture

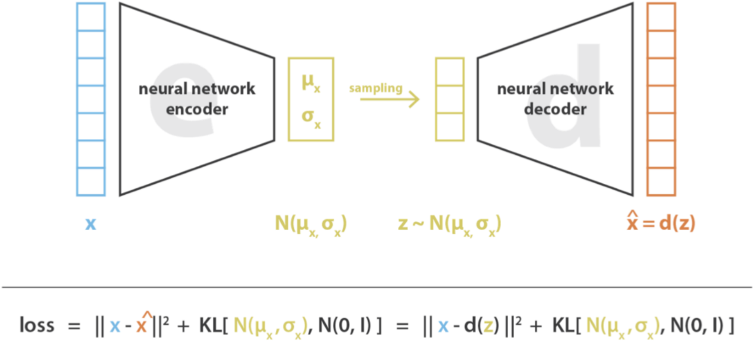


Figure: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE Model for MNIST

- In the encoding part, we use LSTM layer which as 28 input size and 28 sequence length.
- After the LSTM layer, we use ReLU activation function and fully connected layer with linear vectors. Then, we calculate mean and variance of the latent space and create a sample by using them.
- In decoding part, we use 3 layer CNN and we apply transpose convolutional layers. The transpose convolution is opposite to convolution operation.
- The transpose convolutional layers contain transpose convolution operation, ReLU activation function, batch normalization, pooling and dropout. At the end of the decoder part, we should 28x28 images.

VAE Model for MNIST

<table>
<tr><th colspan="4">Table2: Summary of Models and Their Comparison</th></tr>
<tr><td rowspan="3">sequence length = l = 28
input size = i = 28
num layers = n = 1
hidden size1 = h_1 = 32
hidden size2 = h_2 = 64
hidden size3 = h_3 = 128
hidden size4 = h_4 = 256
hidden size4 = h_5 = 16
strides = s = (3,1)
kernel size = k = 5x5
pooling kernel = p = (3,2)
dropout prob. = pr = 0.01

batch size = 100
number of epochs = 50
learning rate = 0.001</td><td></td><td>Model 1 (Our Best Model)</td><td>Model 2</td></tr>
<tr><td>Encoder Part</td><td>$a_1 = LSTM(l, i, h_1, n)$
$a_2 = ReLU(a_1)$
$a_3 = Fully\ Connected(a_2, h_4)$
$a_4 = Sampling(a_3)$

$a_1 \rightarrow 28x32$
$a_2 \rightarrow 28x32$
$a_3 \rightarrow 28x256$
$a_4 \rightarrow 256x28x1$</td><td>$a_1 = LSTM(l, i, h_1, n)$
$a_2 = ReLU(a_1)$
$a_3 = Fully\ Connected(a_2, h_5)$
$a_4 = Sampling(a_3)$

$a_1 \rightarrow 28x32$
$a_2 \rightarrow 28x32$
$a_3 \rightarrow 28x16$
$a_4 \rightarrow 16x28x1$</td></tr>
<tr><td>Decoder Part</td><td>Layer1
$a_5 = TranposeConv(a_4, h_4, h_3, k, s)$
$a_6 = ReLU(a_5)$
$a_7 = Batch\ Normalization(a_6)$
$a_8 = Pooling(a_7, p)$
$a_9 = Dropout(a_8, pr)$

Layer2
$a_{10} = TranposeConv(a_9, h_3, h_2, k, s)$
$a_{11} = ReLU(a_{10})$
$a_{12} = Batch\ Normalization(a_{11})$
$a_{13} = Pooling(a_{12}, p)$
$a_{14} = Dropout(a_{13}, pr)$

Layer3
$a_{15} = TranposeConv(a_{14}, h_3, 1, k, s)$
$a_{16} = ReLU(a_{15})$
$a_{17} = Batch\ Normalization(a_{16})$
$a_{18} = Pooling(a_{17}, p)$
$a_{19} = Dropout(a_{18}, pr)$

$a_{20} = Fully\ Connected(a_{19}, 28)$

$a_9 \rightarrow 128x28x2$
$a_{14} \rightarrow 64x28x3$
$a_{19} \rightarrow 1x28x3$
$a_{20} \rightarrow 1x28x28$</td><td>Layer1
$a_5 = TranposeConv(a_4, h_5, h_3, k, s)$
$a_6 = ReLU(a_5)$
$a_7 = Batch\ Normalization(a_6)$
$a_8 = Pooling(a_7, p)$

Layer2
$a_9 = TranposeConv(a_8, h_3, h_2, k, s)$
$a_{10} = ReLU(a_9)$
$a_{11} = Batch\ Normalization(a_{10})$
$a_{12} = Pooling(a_{11}, p)$

Layer3
$a_{13} = TranposeConv(a_{12}, h_3, 1, k, s)$
$a_{14} = ReLU(a_{13})$
$a_{15} = Batch\ Normalization(a_{14})$
$a_{16} = Pooling(a_{15}, p)$

$a_{17} = Fully\ Connected(a_{16}, 28)$

$a_8 \rightarrow 128x28x2$
$a_{12} \rightarrow 64x28x3$
$a_{16} \rightarrow 1x28x3$
$a_{17} \rightarrow 1x28x28$</td></tr>
</table>

Loss Change for VAE

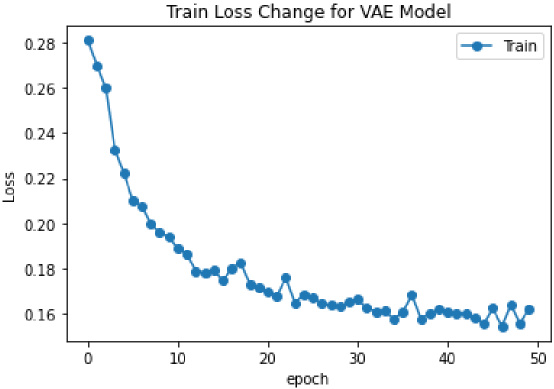
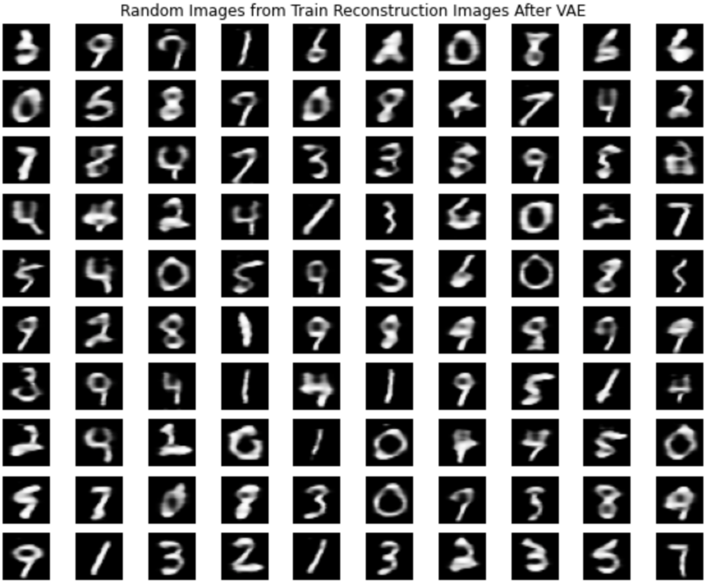


Figure: Loss is calculated bu using binary cross entropy and regularization term with KL divergence with 50 epochs and 100 batch size.

Generated Data with Model 1



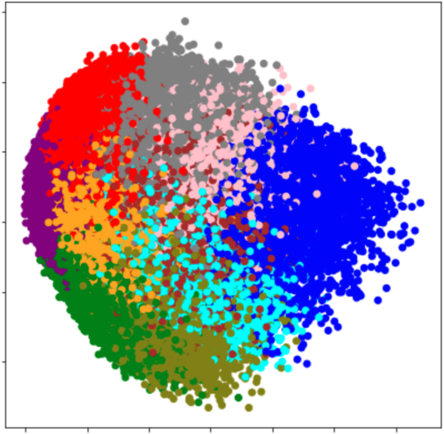
Generated Data with Model 2

k-means and PCA for Reconstruction Images after VAE

- After reconstruction images by using VAE, we use k-means model with 10 groups to get labels. Then, we also apply PCA to plot them and to control the groups for generated images.

k-means and PCA for Reconstruction Images after VAE

- After reconstruction images by using VAE, we use k-means model with 10 groups to get labels. Then, we also apply PCA to plot them and to control the groups for generated images.



References

- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 770-778.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. *In European conference on computer vision*, 630-645.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097-1105.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324.
- Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.
- Nutan (2021). PyTorch Recurrent Neural Networks With MNIST Dataset. Retrived from https://medium.com/@nutanbhogendrasharma/pytorch-recurrent-neural-networks-with-mnist-dataset-2195033b540f
- Ranjan, C. (2019). Step-by-step understanding LSTM Autoencoder layers. Retrived from https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352
- Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., & Carin, L. (2016). Variational autoencoder for deep learning of images, labels and captions. arXiv preprint arXiv:1609.08976.
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 1-9.
- Zhang, K., Sun, M., Han, T. X., Yuan, X., Guo, L., & Liu, T. (2017). Residual networks of residual networks: Multilevel residual networks.*IEEE Transactions on Circuits and Systems for Video Technology*, 28(6), 1303-1314.
- DeepLearning.AI - Convolutional Neural Networks on Coursera, https://www.coursera.org/learn/convolutional-neural-networks/lecture/HAhz9/resnets
- ResNet Model in Pytorch, https://github.com/pytorch/vision/blob /master/torchvision/models/resnet.py