Advanced Topic in Computer Vision

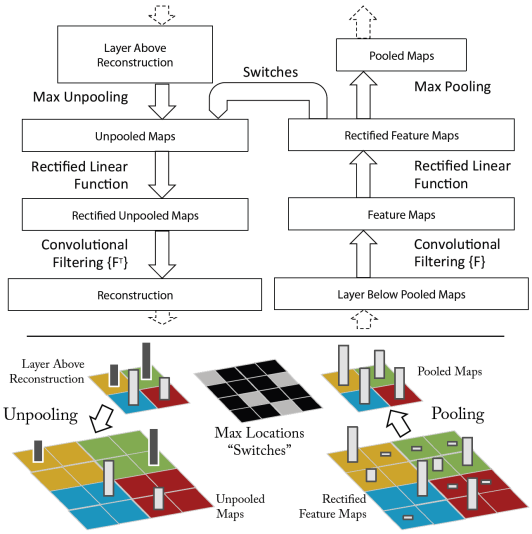
Himanshu Singh

Github

Vizualization

Visualizing and Understanding Convolutional Networks

Visualization with Decovnet

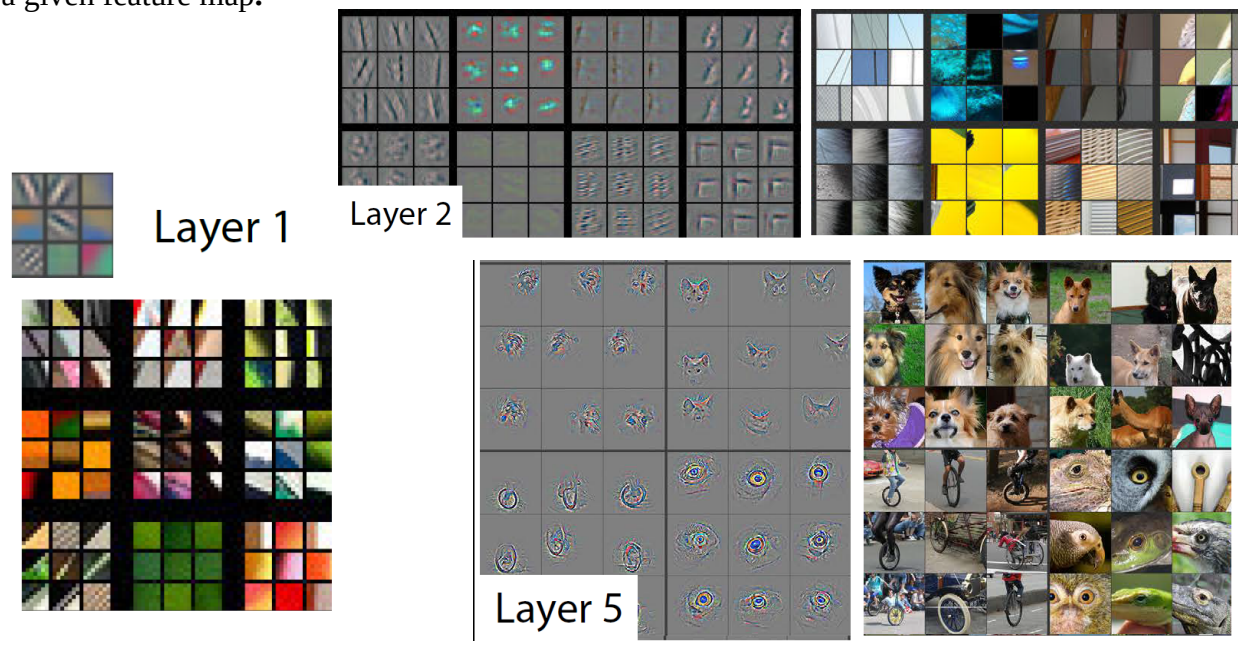


Unpooling : Sine, Pooling is non-invertible, we'll never be able to retrieve the same elements! However, we can obtain an approximate inverse by recording the locations of the maxima within each pooling region. The DeConvNet places the reconstructions from the layer above into appropriate locations based on the maxima found.

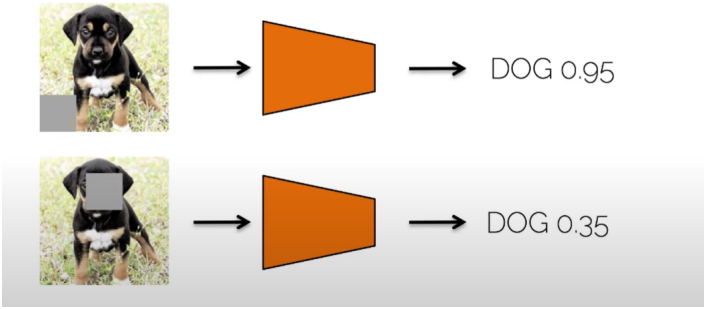
Rectification : ConvNet uses ReLU non-linearity to rectify the feature maps. To obtain valid feature reconstructions at each layer, the reconstructed signal is thus also passed through a ReLU non-linearity.

Filtering : The DeConvNet uses the transpose of this learned filter to the rectified representation from the step above to reconstruct the deconvolved layer output.

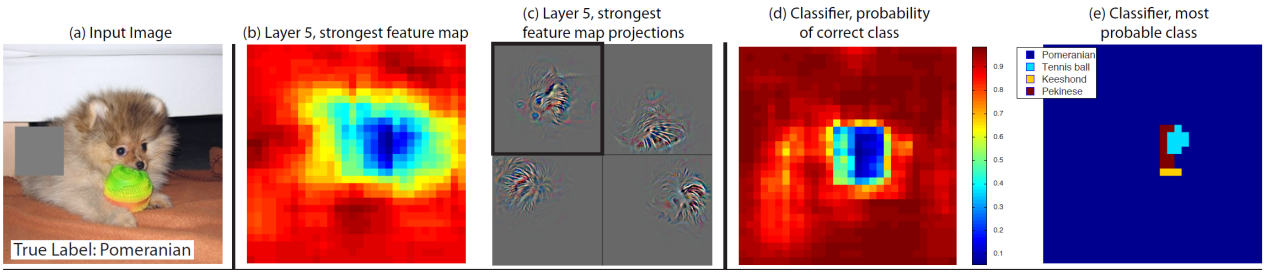
The top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using deconvolutional network approach. The reconstructions are reconstructed patterns from the validation set that cause high activations in a given feature map.



Occlusion Experiment



Create a map, where each pixel represents the classification probability if an occlusion square is placed in that region.



(b): record the total activation in one layer 5 feature map (the one with the strongest response in the unoccluded image).
(c): a visualization of this feature map projected down into the input image (black square), along with visualizations of this map from other images.
(d): a map of correct class probability, as a function of the position of the gray square.
(e): the most probable label as a function of occluder position.

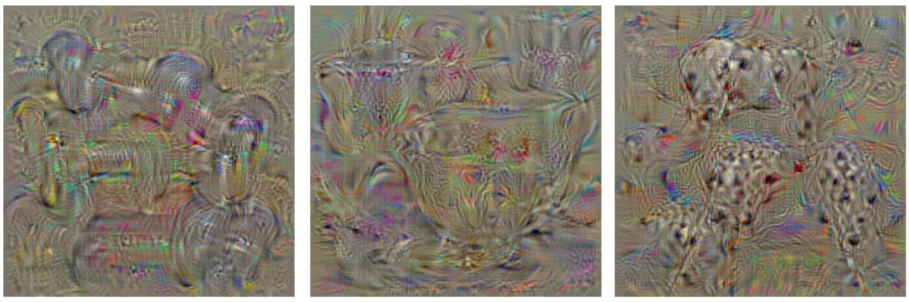
Visualising Image Classification Models and Saliency Maps

Gradient Ascent: Generate synthetic images that maximally activates the filter. We want to find an image that maximizes the score for particular class.

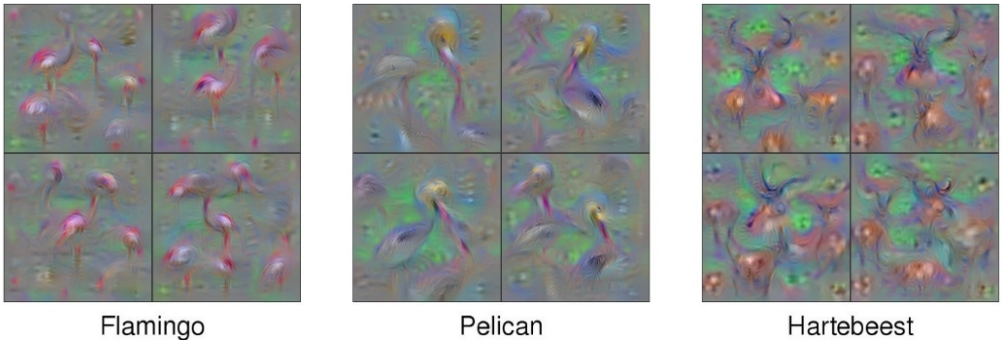
$$arg \max_I S_C(I) + \lambda \|I\|_2^2$$

The score is taken before the softmax layer. Direct output of fully connected layer.

1. Get Trained CNN
2. Pass a zero image through CNN
3. Maximize the score: Use Gradient ascent and backpropagation
4. Make a small update on the image
5. Repeat
6. Visualize by adding the training mean image.



We can also use different regularization : using Gaussian blur on image, clipping pixel with small value to 0, clipping gradients with small value to zero. [Ref]

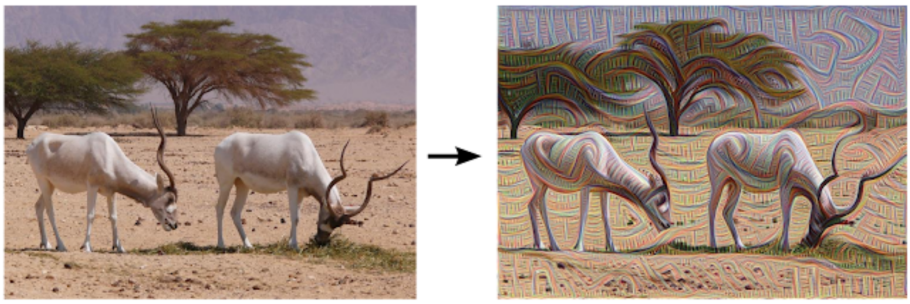


DeepDream

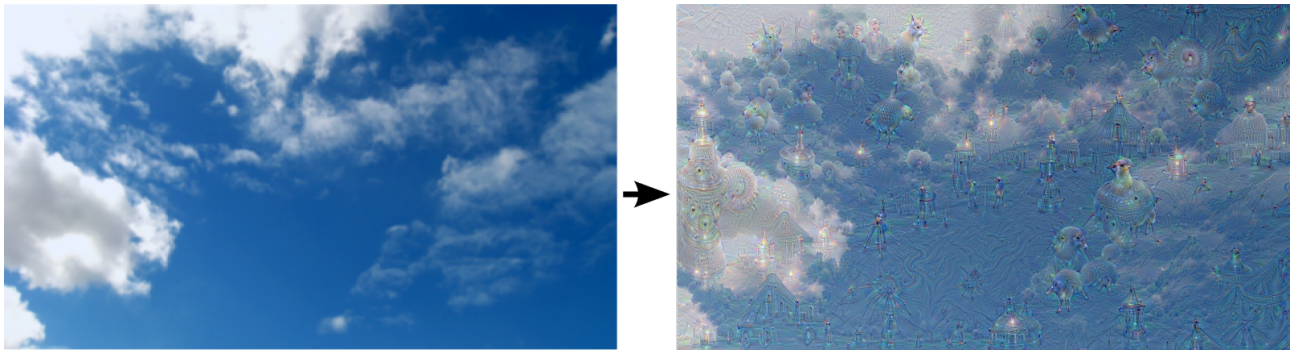
Feed an Image, Choose a layer and ask network to enhance whatever was detected : If you see dogs, show me more dogs!

1. Forward pass of the image up to layer L
2. Set Gradients of the layer = Activations
   Large Activations for dog filter will create large gradients
3. Backpropagate
4. Update the image

Low Layer : basic features



Deep Layers

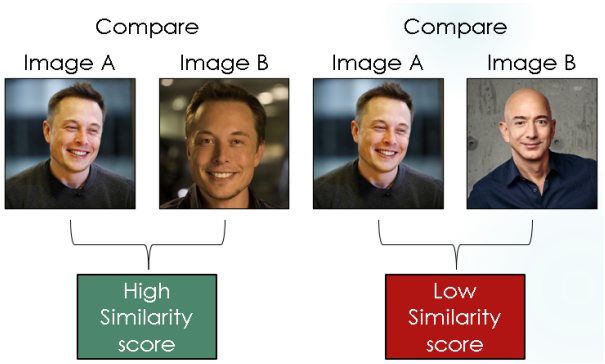


Similarity Learning

Application : Face Recognition

$d(A,B) > \tau$ *Different Person*

$d(A,B) < \tau$ *Same Person*



We process our face images into some embeddings f(A) and f(B).

Distance function $d(A,B) = \|f(A) - f(B)\|^2$

If A and B are same person : d(A,B) is small

$\mathscr{L}(A,B) = \|f(A) - f(B)\|^2$

else d(A,B) is large

$\mathscr{L}(A,B) = max(0, m^2 - \|f(A) - f(B)\|^2)$ Hinge Loss

Contrastive Loss :

$$\mathscr{L}(A,B) = \underbrace{y^* \|f(A) - f(B)\|^2}_{\text{Positive pair}} + \underbrace{(1 - y^*) max(0, m^2 - \|f(A) - f(B)\|^2)}_{\text{Negative Pair}}$$

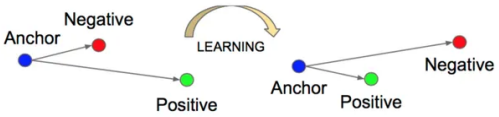
Triplet Loss allow us to learn a ranking.

We want,(A is Anchor Image, P is positive Image, N is Negative Image)

$$\|f(A) - f(P)\|^2 < \|f(A) - f(N)\|^2$$
$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 < 0$$
$$\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m < 0$$
$$\mathscr{L}(A,P,N) = max(0, \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + m)$$

Hard Negative Mining:

1. Train for a few epochs
2. Choose the hard cases where $d(A,P) \approx d(A,N)$
3. Train with those to refine the distance learned



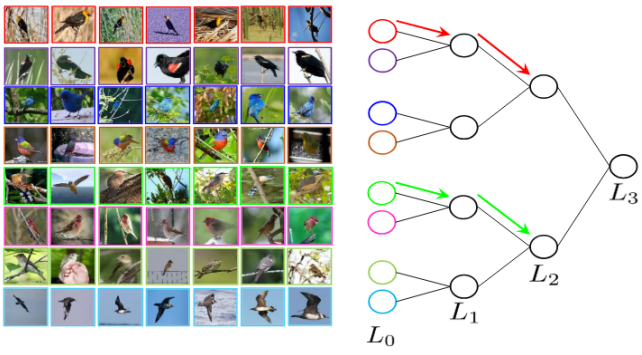
Challenges:

1. Random sampling does not work- the number of possible triplet is $O(n^3)$ so the network would take along time.
2. Even with Negative Mining, there is risk of being stuck in local minima.

Sampling: Hierarchical Triplet Loss

Build hierarchical tree where leaves of tree represent image class, recursively merge them until you reach the root node. In order to create the tree, we define distance between classes. If the distance is small, they will be merged in the next level of tree.

$$d(p,q) = \underbrace{\frac{1}{n_p n_q}}_{\text{Cardinality}} \sum_{i \in p, j \in q} \underbrace{\|r_i - r_j\|^2}_{\text{Deep Features}}$$



Finding Anchor:

1. Randomly select l' nodes at 0^{th} level: Done to preserve class diversity in the mini-batch.
2. m-1 nearest classes at 0^{th} level are selected for each epoch of l' nodes based on distance in feature space: want model to learn discriminative features from visual similar classes.
3. t images per class are selected randomly.

Mini batch: t*m*l’
Loss Formulation

$$\mathscr{L}_M = \frac{1}{2Z_M} \underbrace{\sum_{T^z \in T^M}}_{Triplets} [\|x_a^z - x_p^z\| - \|x_a^z - x_n^z\| + \alpha_z]$$

The margin actually depends on the distance computed on the hierarchical tree. The idea is to adapt to class distribution and difference of samples within the classes.
Ensembles: Divide space into K clusters, and have one learner per cluster.
1. Cluster the embedding space in K cluster using K-means
2. Build K independent learners (fully connected) on top of CNN, where each learner corresponds to one cluster.
3. Until convergence, sample each mini-batch from one random cluster, and update only its corresponding learner.
4. After the network has converged finetune using all learners at same time.
5. Go back to (1) and repeat several times.

Encoder-Deocder

SegNet

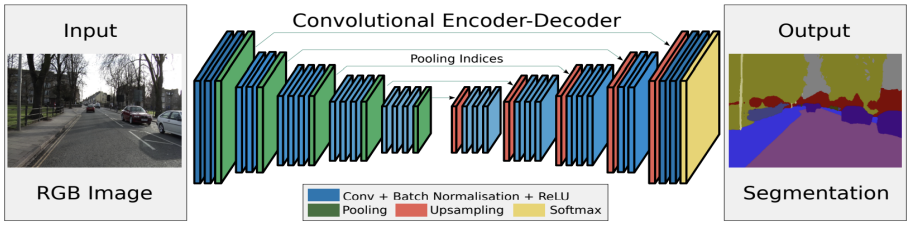
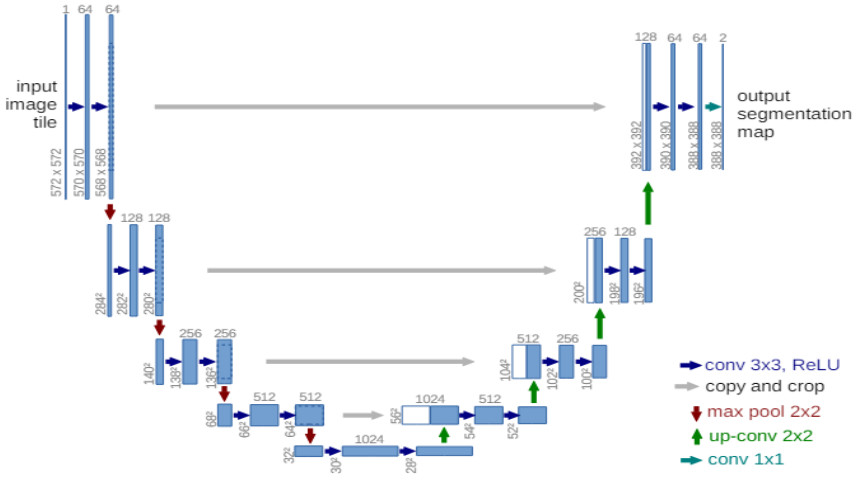


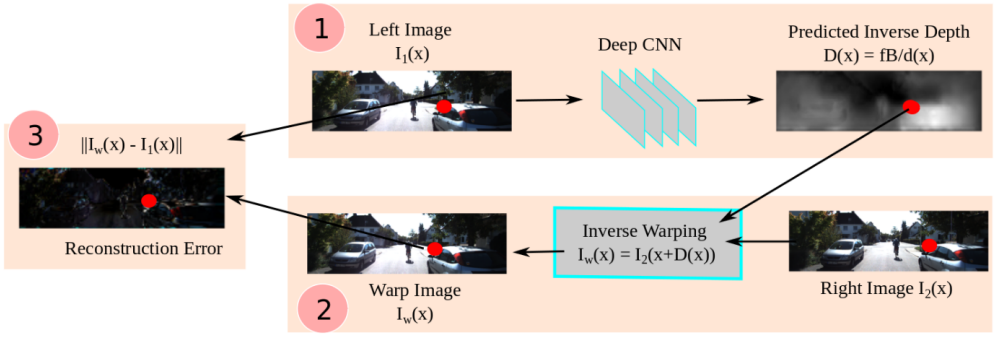
Fig. 2. An illustration of the SegNet architecture. There are no fully connected layers and hence it is only convolutional. A decoder upsamples its input using the transferred pool indices from its encoder to produce a sparse feature map(s). It then performs convolution with a trainable filter bank to densify the feature map. The final decoder output feature maps are fed to a soft-max classifier for pixel-wise classification.

UNet

Pasees the high level information into deep layers of encode through skip connection



Monocular Depth



$\{I_1, I_2\}$:

Rectified stereo pair f: focal length B: Camera Baseline distance

d(x): predicted depth for left Image

Motion of pixel along the scan-line is $D(x)=fB/d(x)$

Using right Image, a warp image can be synthesized $I_w=I_2(x+fB/d(x))$

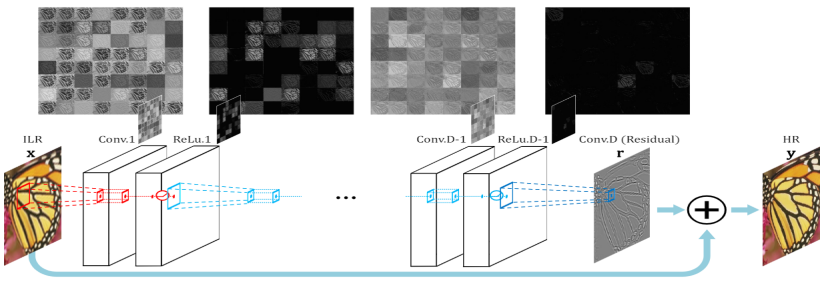
Since photometric loss function is non-informative in homogeneous regions of the scene. Thus multiple disparities can generate equally good warps I_w, use very simple L2 regularization on the disparity discontinuities as our prior to deal with the aperture problem:

$$E_{smooth}=\|\nabla D(x)\|^2$$

$$E=\sum_{i=1}^{N} E^i_{recon}+\gamma E^i_{smooth}$$

Image Super Resolution

Learn Residual through CNNs



Style Transfer

Perceptual loss

Perceptual Loss/ Content Loss/ Feature reconstruction Loss

$$l_{feat}^{\phi,j}(\hat{y},y)=\underbrace{\frac{1}{C_j H_j W_j}}_{\text{Feature Map Size}}\underbrace{\|\phi_j(\hat{y})-\phi_j(y)\|_2^2}_{\text{feature map differences}}$$

1. Take VGG network trained for Image classification
2. Pass the generated image and GT through the network
3. Compare the feature maps

Intuition: If there was an object in original image, we want to have “similar” feature triggered for the generated image.

Content Image *Style Image* *Neural Style Transfer*

Style loss

$$f_{style}^{\phi,j}(\hat{y},y)=\|G_j^{\phi}(\hat{y})-G_j^{\phi}(y)\|_F^2$$

Compare Gram Matrices

1. Take VGG network trained for Image classification
2. Pass the generated image and GT through the network
3. Compare the Gram matrices at certain layer.

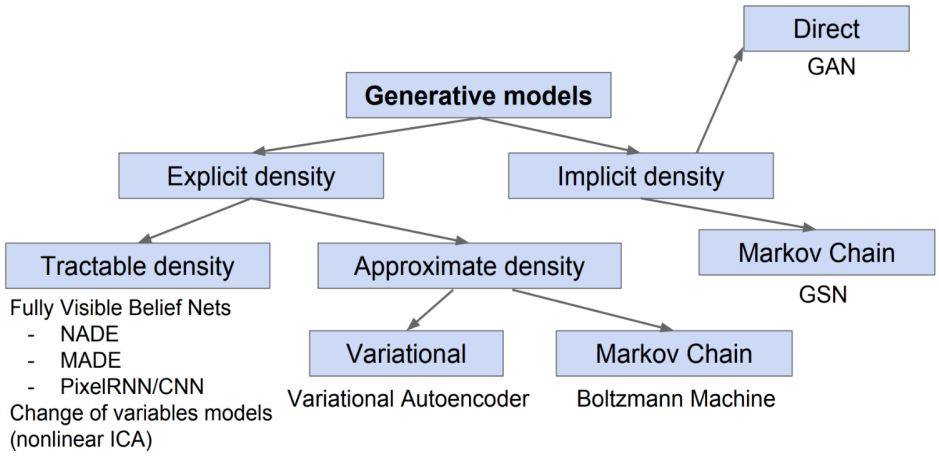
$$G_j^{\phi}(x)_{c,c'}=\frac{1}{C_j H_j W_j}\sum_{h=1}^{H_j}\sum_{w=1}^{W_j}\phi_j(x)_{h,w,c}\phi_j(x)_{h,w,c'}$$

Intuition: It captures information about which features tend to activate together. Correlation of activations across channels.

The method is slow, and require many forward/backward passes through VGG.

Fast Neural Style Transfer

Generative Neural Networks

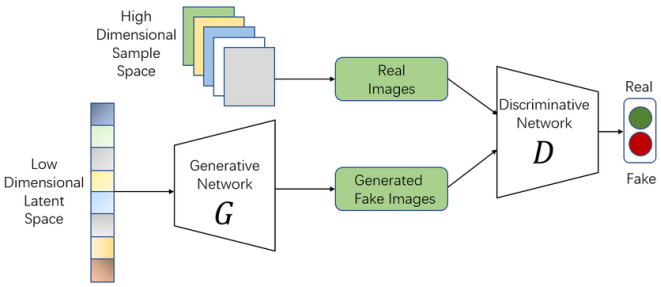


Generative Adversarial Networks (GANs)

Discriminator tries to make D(G(z)) near to 0, Generator tries to make D(G(z)) near to 1.

Minimax Game :

- G minimizes probability that D is correct
- Equilibrium is saddle point of Discriminator Loss



Generator Loss:

$J^{(G)} = -J^{(D)}$

Discriminator Loss:

$$J^{(D)} = -\frac{1}{2}\boldsymbol{E}_{x\sim p_{data}} \log D(x) - \frac{1}{2}\boldsymbol{E}_z \log(1 - D(G(z)))$$

Heuristic Method:

- G maximizes the log-probability of D being mistaken.
- G can still learn even when D rejects all generator samples

$$J^{(G)} = -\frac{1}{2}\boldsymbol{E}_z \log D(G(z))$$

---

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

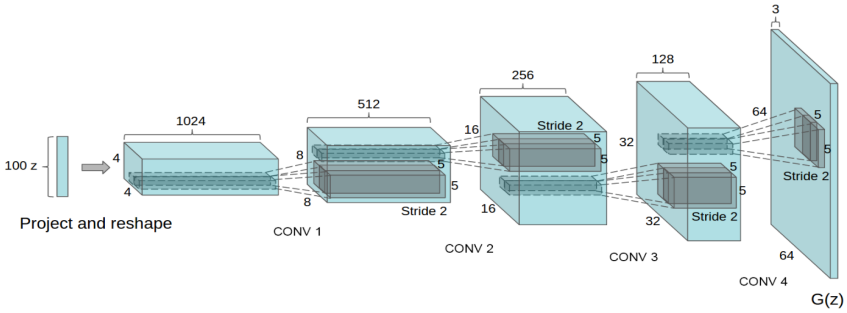
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

DCGANs



Mode Collapse in GANs

$$\min_G \max_D V(G,D) \neq \max_D \min_G V(G,D)$$

- D in inner loop: Convergence to correct distance
- G in inner loop: Easy to converge to one sample.

Mode collapse occurs when a GAN generates only a limited set of output examples instead of exploring the entire distribution of the training data. In other words, the generator of the GAN becomes stuck in a particular mode or pattern, failing to generate diverse outputs that cover the entire range of the data. This can result in the generated output appearing repetitive, lacking in variety and detail, and sometimes even being completely unrelated to the training data.

Two main reasons:

1. Catastrophic forgetting: It refers to the phenomenon in which a model trained on a specific task forgets the knowledge it has gained while learning a new task. During the t^{th} training step, the generator produces samples with distribution p_G that approximates the true data distribution p_R. At each training step, the discriminator is trained to differentiate between the real data p_R and the generated samples p_G. However, as p_G shifts with each step, the discriminator must adapt to new classification tasks.
2. Discriminative over-fitting

GAN Hacks

1. Normalize input ([0,1], [-1,1]), Use Sigmoid/tanh at output layer.
2. Sample from Spherical distribution (e.g. Gaussian). Helps in interpolation.
3. Use BatchNorm
4. Adam for Generator, SGD for discriminator.
5. Prevent discriminator from giving too large gradient signal to generator. This reduces confidence and encourages “extreme samples’

$$J(D) = -\frac{1}{2}\boldsymbol{E}_{x\sim p_{data}} \lambda \log D(x) - \frac{1}{2}\boldsymbol{E}_z \log(1 - D(G(z))), \lambda < 1; \sim 0.9$$

5. Update Discriminator using history of refined images.
6. Avoid Sparse Gradients: Leaky ReLU, AveragePool for downsampling, deconvolution of upsampling.

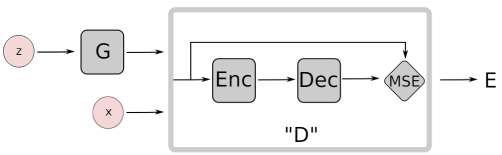
Energy Based GANs (EBGAN)

Loss Function $\mathscr{L}_D(x,z) = D(x) + [m - D(G(z))]^+$

$$\mathscr{L}_G = D(G(z))$$

$[a]^+ = max(0,a)$

AutoEncoder used in Discriminator



$$D(x)=\|Dec(Enc(x))-x\|$$

Wasserstein GAN (WGAN)

Earth Mover Distance/ Wasserstein distance: Minimum cost of transporting mass in converting data distribution q to data distribution p. Wasserstein distance for real data distribution P_r and generated data distribution P_g

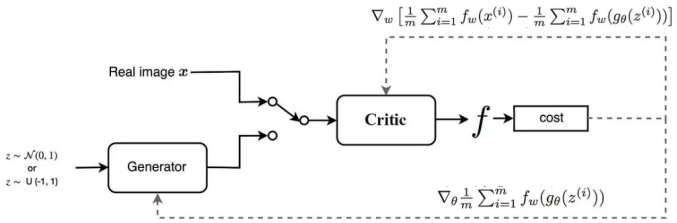
$$W(P_r,P_g)=\inf_{\gamma\in\prod(P_r,p_g)}\left(\underset{(x,y)\sim\gamma}{E}[\|x-y\|]\right)$$

$$\prod(P_r,p_g)\rightarrow\text{Set of al joint distribitions }\gamma(x,y)$$

The above equation is highly intractable. So we approximate it to:

$$W(P_r,P_g)=\sup_{\|f\|_L\le 1}\left(\underset{x\sim P_r}{E}[f(x)]-\underset{x\sim P_\theta}{E}[f(y)]\right)$$

$|f(x_1)-f(x_1)|\le|x_1-x_2|$ 1-Lipschitz function



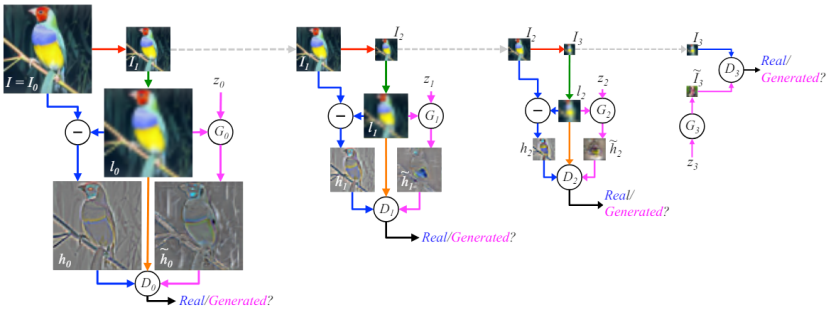
To learn, we build a deep network. This network is very similar to the discriminator D, without the sigmoid function and outputs a scalar score. This score can be interpreted as how real the input images are. To enforce f, Lipschitz constraint, WGAN applies clipping to discriminator weight, restricting hem in certain range.

$$w\leftarrow w+\alpha.\text{RMSprop}(w,g_w)$$

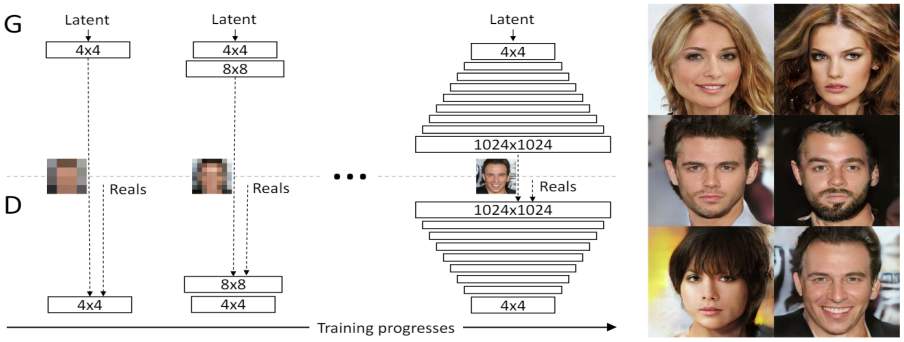
$$w\leftarrow clip(w,-c,c)$$

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d}\frac{1}{m}\sum_{i=1}^m[\log D(x^{(i)})+\log(1-D(G(z^{(i)})))]$	$\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^m[\log(D(G(z^{(i)})))]$
WGAN	$\nabla_w\frac{1}{m}\sum_{i=1}^m[f(x^{(i)})-f(G(z^{(i)}))]$	$\nabla_\theta\frac{1}{m}\sum_{i=1}^m f(G(z^{(i)}))$

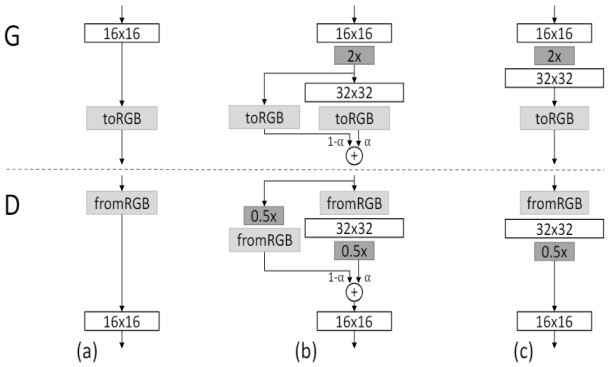
Multiscale GANs



Progressive Growing GANs

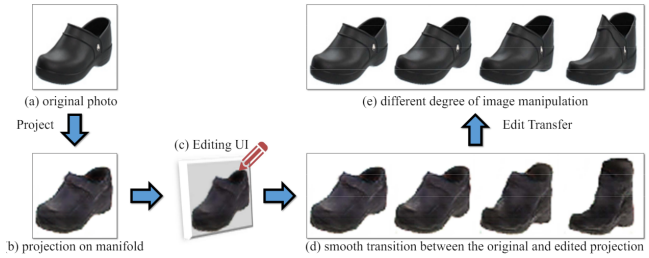


When doubling the resolution of the generator and discriminator, fade in the new layers smoothly. We treat the layers that operate on the higher resolution like a residual block, whose weight α increases linearly from 0 to 1. The toRGB represents a layer that projects feature vectors to RGB colours and fromRGB does the reverse; both use 1 × 1 convolutions. 2X: Nearest neighbourhood upscaling filtering and 0.5x: AveragePool downscaling



Interactive GANs (iGANs)

Take Original Image, project on latent dimension and regenerate it using GAN. Modify the colour and shape of Image, And apply those changes to original Image.



Projecting an Image onto Manifold

Input: Real Image x^R, Output: Latent vector z

Optimization: $z^* = arg\,min\,\mathscr{L}(G(z), x^R)$ Highly non linearly Problem, so we train a neural network to learn these mapping. z = P(x)

$$\theta_P^* = \underset{\theta_P}{arg\,min} \sum_{x_n^R} \mathscr{L}\big(\underbrace{G(P(x^R;\theta_P))}_{\text{Auto-Encoder, FIxed G}}, x^R\big)$$

Hybrid Method: Use the network as Initialization of the optimization problem.

$$z^* = \underset{z \in Z}{arg\,min}\left\{ \sum_g \big(\underbrace{f_g(G(z), v_g)}_{\text{Data term}} + \underbrace{\lambda_s \|z - z_0\|_2^2}_{\text{Manifold smoothness}} \big) \right\}$$

Edit Transfer: Motion(u, v) + Colour (A_{3x4})

$$\int\int \underbrace{\|I(x,y,t) - A.I(x+u, y+v, t+1)\|^2}_{\text{Data Term}} + \underbrace{\sigma_s(\|\nabla u\|^2 \|\nabla v\|^2)}_{\text{Spatial Reg}} + \underbrace{\sigma_c \|\nabla A\|^2}_{\text{Color Reg}} dx\,dy$$

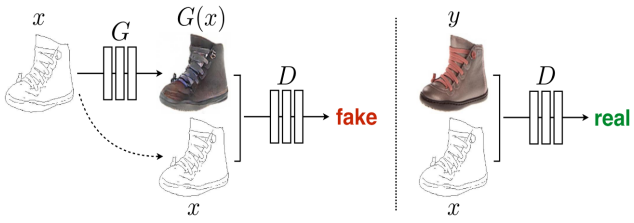
pix2pix

$$\mathscr{L}_{cGAN}(G,D) E_{x,y}[\log D(x,y)] + E_{x,z}[\log(1 - D(x, G(x,z)))]$$

x: Observed Image; z: Noise; y: Generator o/p

G tries to minimize the objective against adversarial D that is trying to maximize it.

$arg\,\underset{G}{min}\,\underset{D}{max}\,\mathscr{L}_{cGAN}(G,D)$



U-Net as Generator: skip connections for preserving structure.

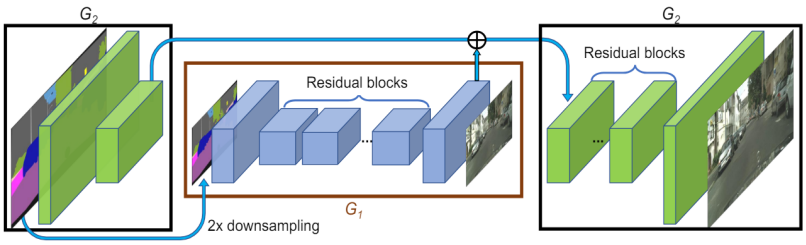
PatchGAN as discriminator : Instead of predicting whole image, model taken NxN patch imageand predict every pixel in patch. Encourages high frequency details.

Also has L1 loss constraint on Generator

$$G^* = \underset{G}{arg\,min}\,\underset{D}{max}\,\mathscr{L}_{cGAN}(G,D) + \lambda \mathscr{L}_{L1}(G)$$

pix2pixHD

Train Residual network G1 on lower resolution. Then another residual network G2 is appended to G1 and 2 networks are trained jointly on high resolution images.



Multi-scale Discriminator

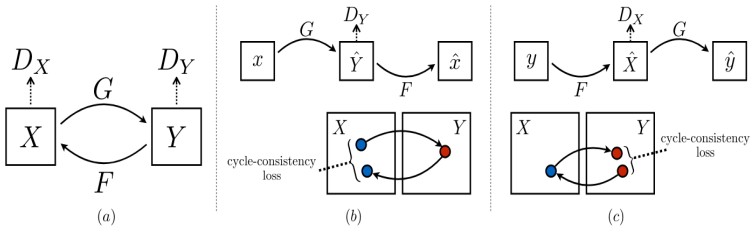
$$\min_{G} \max_{D_1,D_2,D_3} \sum_{k=1,2,3} L_{GAN}(G, D_k)$$

cycle GAN

To regularize mappings, they introduce cycle consistency loss

$\|F(G(x))-x\|_1$ and $\|G(F(y))-y\|_1$

Total loss function



$$\mathcal{L}(G,F,D_Y,D_Y) = \mathcal{L}_{GAN}(G,D_Y,X,Y) + \mathcal{L}_{GAN}(G,D_X,X,Y) + \lambda \mathcal{L}_{cycle}(G,F)$$

$$\mathcal{L}_{GAN}(G,D_Y,X,Y) = E_{y\sim p_{data(y)}}[\log D_Y(y)] + E_{x\sim p_{data(x)}}[\log(1-D_Y(G(x)))]$$

$$\mathcal{L}_{GAN}(G,D_X,X,Y) = E_{x\sim p_{data(x)}}[\log D_X(x)] + E_{x\sim p_{data(y)}}[\log(1-D_X(G(y)))]$$

$$\mathcal{L}_{cycle}(G,F) = E_{x\sim p_{data(x)}}[\|F(G(x))-x\|_1] + E_{y\sim p_{data(y)}}[\|F(G(y))-y\|_1]$$

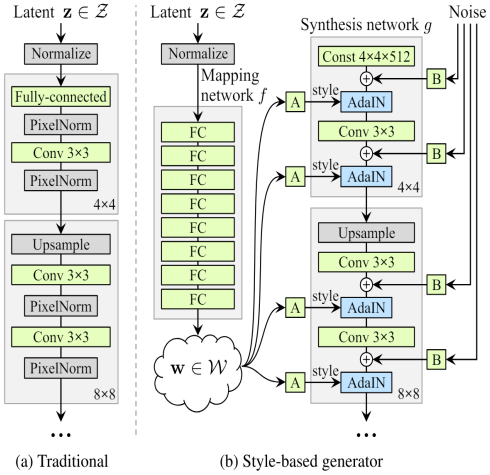
Style GAN

Given latent code z, a non linear mapping $f: Z \to W$ produce $w \in W$. Learned affine transformations specialize w to styles.

y = (y_s, y_b)

$$AdaIN(x,y) = y_{s,i}\frac{x_i-\mu(x_i)}{\sigma(x_i)} + y_{b,i}; x_i = \Re^{H\times W}$$

$$w \xrightarrow[\text{affine}]{A} (y_s, y_b) \in \Re^{2C}$$



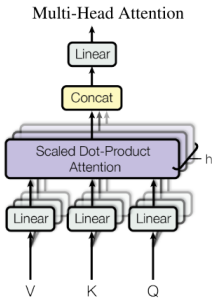
Transformers

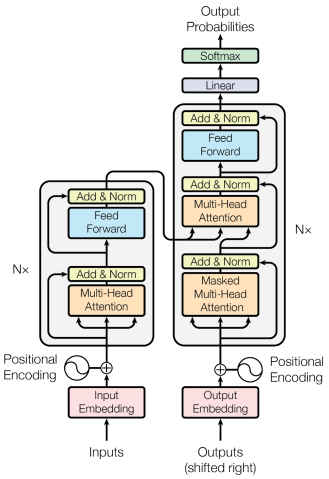
Problem with RNN:

- Each word is dependent on the words coming before it.
- Vanishing Gradient problem.
- Even LSTMs dependencies are not that long.

Attention

MultiHead Attention





Intuition Take Query Q, find most similar key K, and then find the value V that corresponds to the key. Learn V, K, Q where

V – Bunch of interesting things

K – how we can store some index

Q – Interesting things

$$\text{Attention}(Q,K,V)=\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Divide by $\sqrt{d_k}$, because for large values of key's dimension, the dot product grows large in magnitude, pushing softmax into regions where it has small gradients.

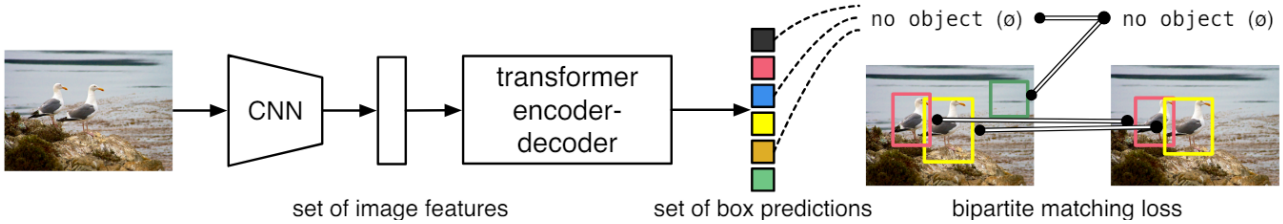
h parallel attention heads. After scaled dot product, concatenate h vector and pass through linear layer.

Masked Multihead Attention Same as multihead attention, but masked. Ensure that the predictions for position i can depend only on the known outputs at positions less than i.

Self Attention acts like weighted average but does not know from which position the information is coming. Hence uses fixed positional encoding bases on trigonometric series, to make use of order of sequence.

$$PE_{(pos,2i)}=\sin\left(pos/10000^{2i/d_{model}}\right)$$
$$PE_{(pos,2i+1)}=\cos\left(pos/10000^{2i/d_{model}}\right)$$

DEtection TRansformer



CNN is used for feature learning the Transformer is used to make predictions.

During training, bipartite matching uniquely assigns prediction with ground truth boxes.

$\hat{y}$ be the N predictions. Assuming N is larger than number of objects in the image, we consider y also as set of size N padded with ϕ. We need to search over permutation of N elements $\sigma \in \wp_N$

$\mathscr{L}_{match}(y_i,\hat{y}_i)$ is pairwise matching cost.

Loss function

$$\hat{\sigma}=\underset{\sigma\in\wp_N}{arg\,min}\sum_{i=1}^{N}\mathscr{L}_{match}(y_i,\hat{y}_{\sigma(i)})$$

$$\mathscr{L}_{match}(y_i,\hat{y}_{\sigma(i)})=\underbrace{-\mathbf{1}_{\{c\neq\phi\}}\hat{p}_{\sigma(i)}c(i)}_{\text{Classification loss}}+\underbrace{\mathbf{1}_{\{c\neq\phi\}}\mathscr{L}_{box}(b_i,\hat{b}_{\sigma(i)})}_{\text{Bounding Box loss}}$$

Optimal Assignment is computed by Hungarian Loss

$$\mathcal{L}_{Hungarian}(y,\hat{y}) = \sum_{i=1}^{N}\left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbf{1}_{c_i \neq \phi}\mathcal{L}_{box}(b_i, \hat{b}_{\hat{\sigma}}(i))\right]$$

Bounding box loss

$$\mathcal{L}_{box}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{iou}\mathcal{L}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{L1}\|b_i - \hat{b}_{\sigma(i)}\|_1$$