Computer Vision Papers-1 (2022)

Himanshu Singh

Github

AlexNet

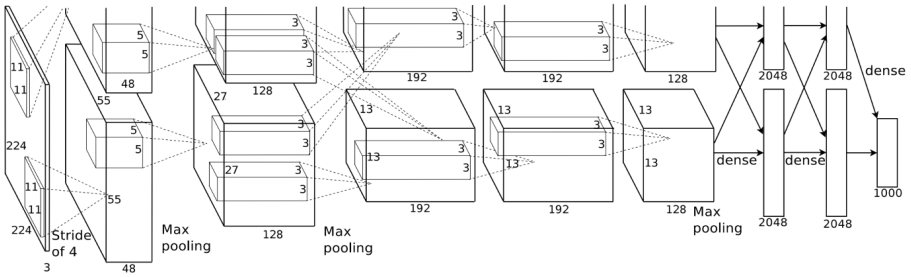
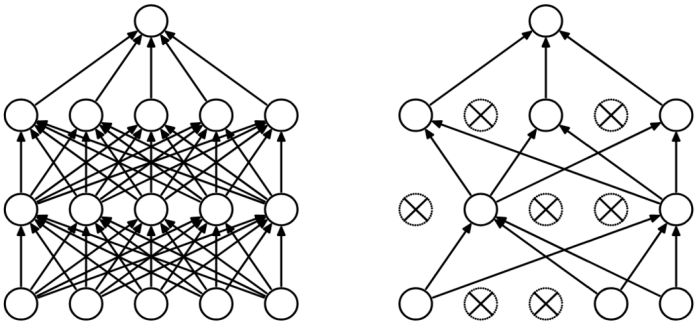


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The architecture consists of eight layers: five convolutional layers and three fully-connected layers. Due to GPU constraints, it was trained 2 parallel stream on 2 GPUs.

Dropout

The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. Each hidden unit with dropout must learn to work with randomly chosen sample of other unit.

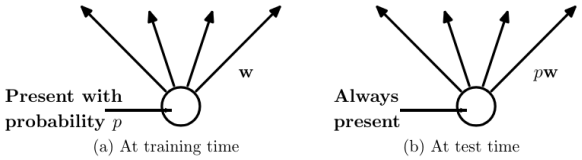


(a) Standard Neural Net (b) After applying dropout.

1. Prevent Over-fitting.

2. Approximately combine exponentially many different neural network efficiently.

$$r_j^{(l)} \sim Bernoulli(p)$$
$$y^{(l)} = \boldsymbol{r}^{(l)} * \boldsymbol{y}^{(l)}$$
$$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \boldsymbol{y}^{(l)} + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$



(a) At training time (b) At test time

VGG-16 & VGG-19

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

LRN: Local Response Normalization

Create competition for big activities amongst neuron outputs computed using different kernels.

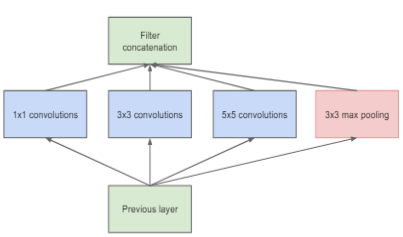
$$b_{x,y}^i = \frac{a_{x,y}^i}{\left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a_{x,y}^i\right)^2\right)^\beta}$$

$b_{x,y}^i$ response-normalized activity

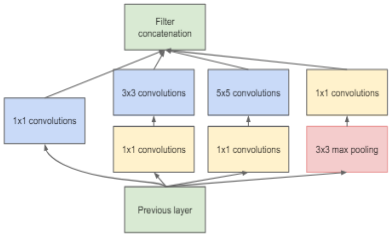
$a_{x,y}^i$ Activity of neuron computed by applying kernel i at position (x,y) and then applying the ReLU non-linearity

$k=2, n=2, \alpha=10^{-4}, \beta=0.75$

GoogLeNet



(a) Inception module, naïve version



(b) Inception module with dimension reductions

In Naive module, 5×5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters. The number of output filters equals to the number of filters in the previous stage. The merging of the output of the pooling layer with the outputs of convolutional layers would lead to an inevitable increase in the number of outputs from stage to stage. This leads to the second idea of the proposed architecture: 1×1 convolutions are used to compute reductions before the expensive 3×3 and 5×5 convolutions.

Batch Normalization

As the training of network continuously changes the parameter of the network, the distribution of activation changes, this is known as Internal Covariance shift.

Training

$$BN(x;\gamma,\beta,\underbrace{x_1,x_2,\ldots,x_m}_{\text{mini-batch}})=\gamma\frac{x-\mu(x_1,\ldots,x_m)}{\sqrt{\sigma^2(x_1,\ldots,x_m)+\epsilon}}+\beta$$

$$x\in\{x1,x_2,\ldots,x_m\} \qquad x_i,\gamma,\beta,\mu,\sigma\in\Re^d$$

Inference

$$BN(x;\gamma,\beta)=\gamma\frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}+\beta$$

μ and σ can be taken as running averages from training data.

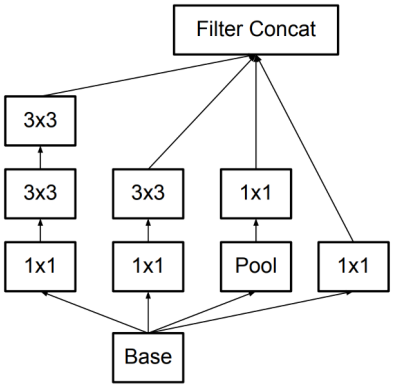
Benefits:

- Higher Learning Rate
- Less sensitive to initialization
- Less Sensitive to activation function
- Regularization Effect
- Preserve Gradients Magnitude (may be)

One more aspect can be look in this, which suggest the Batch Normalization helps in smoothing the Loss function landscape in the weight dimension.

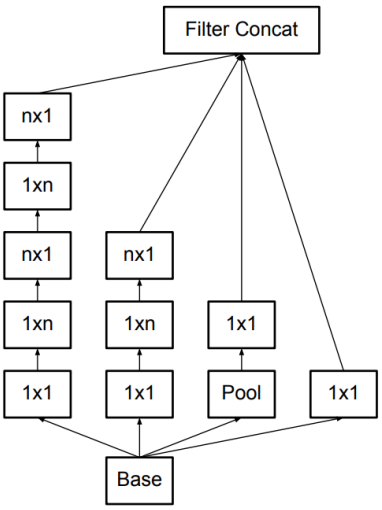
Inception-v3

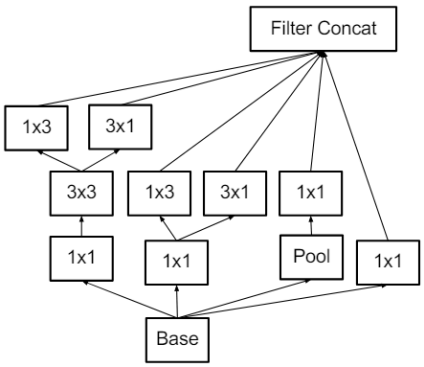
In first layer of Inception , 5x5 convolution can be replaced by two 3x3 convolution stacked together, in second layer 7x7 convolution can be divided into 1x7 and 7x1 convolution. In third inception layer, a mix of 1x3, 3x1, 3x3 and 1x1 are used.

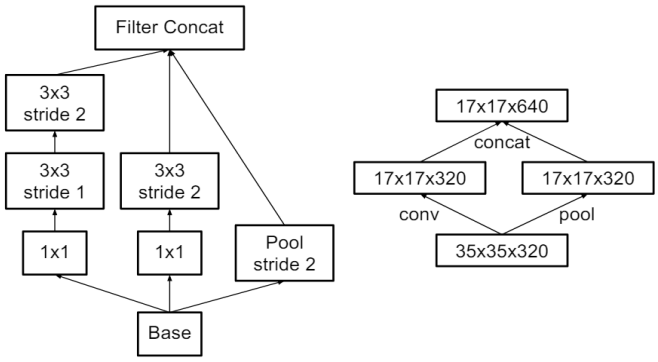


Module where 5x5 convolution is replaced by 2 3x3

Factorization of nxn conv module







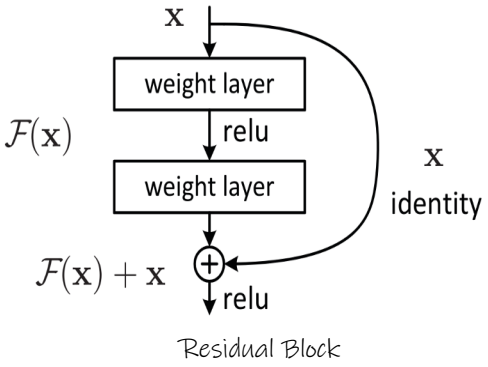
Module that reduces grid size while expanding filter banks

Label Smoothing Normalization

Replace $q(k|x)$ with $\widetilde{q}(k|x)=(1-\epsilon)q(k|x)+\epsilon u(k); k=\{1,2,\ldots,K\}$

ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,64\\3\times3,64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,64\\3\times3,64\\1\times1,256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,128\\3\times3,128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1,128\\3\times3,128\\1\times1,512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,256\\3\times3,256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1,256\\3\times3,256\\1\times1,1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3,512\\3\times3,512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1,512\\3\times3,512\\1\times1,2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



Residual Block

One way to look at skip connection is, we are giving model an ability to choose a path, suppose $F_i(.) = 0$, then input directly flow to next layer. These skip also helps in vanishing gradient problem.

Identity Mapping

What we have is

$$y_l = h(x_l) + F(x_l, W_l)$$
$$x_{l+1} = f(y_l)$$

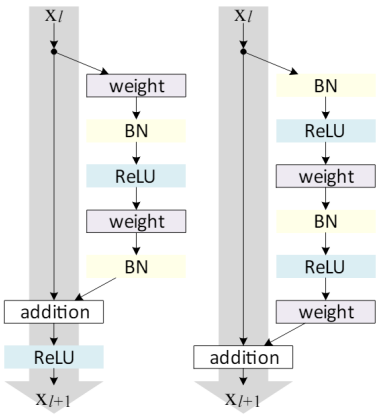
If $h(x_l) = x_l$ and f is ReLU, we have original ResNet framework.

If f and h are identity, then $x_{l+1} = x_l + F(x_l, W_l)$

$$x_L = x_l + \sum_{i=1}^{L-1} F(x_i, W_i)$$

One way to look at this is, information from very lower level is flowing to the final layer. (ignoring the effect of BN, MaxPool)

$$\frac{\partial \varepsilon}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i, W_i)\right)$$



(a) original (b) proposed

During backpropagation, error are propagating from deep layers to initial layers.

If $h(x_l) = \lambda_l x_l$, then

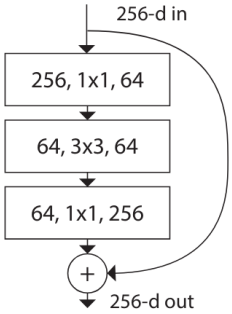
$$x_{l+1} = \lambda_l x_l + F(x_l, W_l)$$
$$x_L = \left(\prod_{i=l}^{L-1}\lambda_i\right)x_l + \sum_{i=l}^{L-1}\left(\prod_{j=i+1}^{L-1}\lambda_j\right)F(x_i, W_i)$$

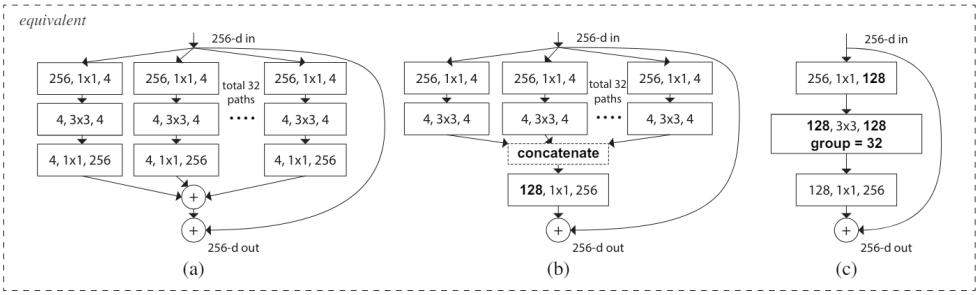
$$x_L = \left(\prod_{i=1}^{L-1} \lambda_i\right) x_l + \sum_{i=1}^{L-1} \hat{F}(x_i, W_i)$$

$$\frac{\partial \varepsilon}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\left(\left(\prod_{i=l}^{L-1} \lambda_i\right) + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} \hat{F}(x_i, W_i)\right)$$

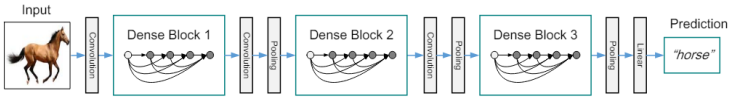
Now, the first term has lot of multiplication, so its either gonna explode or vanish. Same way if h is nor linear, its going to have product of h', which will produce same issue.

ResNeXt





DenseNet



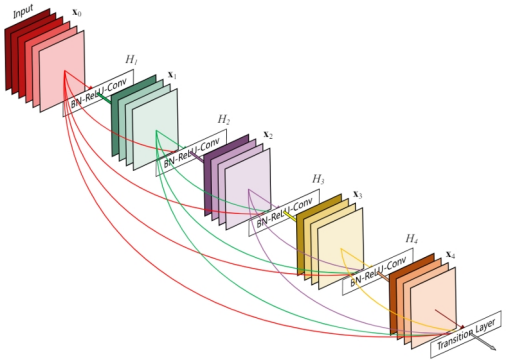
$x_l \rightarrow$ outpuut of l^{th} layer

$$x_l = H(\underbrace{[x_0, x_1, \ldots, x_{l-1}]}_{\text{concatenate}})$$

$H(.) \rightarrow$ NonLinear Transformation

$k \rightarrow$ Growth rate of network

if each layer H_l produces k feature maps then l^{th} layer has $k_0 + k(l-1)$ input feature maps



mixup

Empirical Risk Minimization (ERM)

$f \in F \rightarrow$ function

$X \rightarrow$ Random feature vector

$Y \rightarrow$ random target vector

$P(X, Y) \rightarrow$ Joint Distribution

$l \rightarrow$ loss function

$R(f) = \int l(f(x), y)\, dP(x, y) \rightarrow$ expcted risk

$D = \{(x_i, y_i)\}_{i=1}^{N} \rightarrow$ Training Data

$$R_\delta(f) = \int l(f(x), y)\, \delta P(x, y) = \frac{1}{n}\sum_{i=1}^{N} l(f(x_i), y_i) \rightarrow \text{Emperical Risk}$$

Vicinal Risk Minimization (VRN)

$$P_\nu(\tilde{x}, \tilde{y}) = \frac{1}{n}\sum_{i=1}^{N} \nu(\tilde{x}, \tilde{y} | x_i, y_i)$$

$\nu \rightarrow$ vicinity distribution (Measure the probability of finding virtaul feature target $(\tilde{x}, \tilde{y})$ in vicinity of training feature (x_i, y_i)

$$\nu(\tilde{x},\tilde{y}|x_i,y_i)=N(\tilde{x}-x_i,\sigma^2)\delta(\tilde{y}=y_i)\rightarrow \text{Gaussian vicinity}$$

$$R_\nu(f)=\frac{1}{m}\sum_{i=1}^{m} l(f(\tilde{x}_i,\tilde{y}_i))\rightarrow \text{Emperical Vicinal Risk}$$

mixup

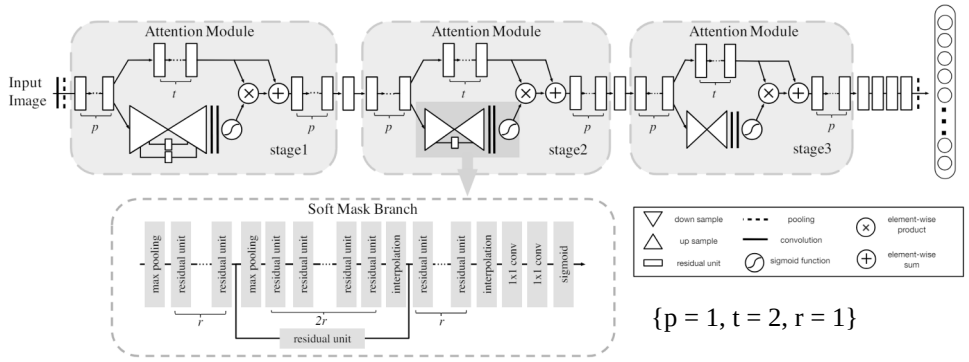
$$\mu(\tilde{x},\tilde{y}|x_i,y_i)=\frac{1}{n}\sum_{j=1}^{n} E[\delta(\tilde{x}=\lambda x_i+(1-\lambda)x_j,\tilde{y}=\lambda y_i+(1-\lambda)y_j)]$$

$$\lambda\sim Beta(\alpha,\alpha)\text{ for }\alpha\in(0,\infty)$$

mixup GAN

$$\max_g \min_d \underset{x,z,\lambda}{E}\, l(d(\lambda x+(1-\lambda)g(z)),\lambda)$$

Residual Attention Network



{p = 1, t = 2, r = 1}

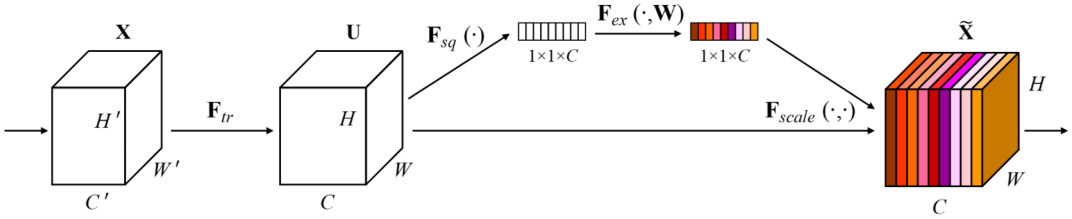
Attention Module

- Mask branch (bottom-up top-down structure)
- Trunk branch (perform feature processing)

$$H_{i,c}(x)=(1+\underbrace{M_{i,c}(x)}_{mask})*\underbrace{F_{i,c}(x)}_{trunk}$$

$i\rightarrow$ ranges over all spatial positions
$c\rightarrow$ index of the channel

Squeeze-and-Excitation Networks



$$X\in\Re^{H'\times W'\times C'} \quad U\in\Re^{H\times W\times C}$$

$$F_{tr}: X\rightarrow U$$

$$V=[v_1,v_2,\ldots,v_C] \text{ learned set of filter kernels}$$

$$v_c \text{ parameters of the } c^{th} \text{ filter}$$

$$U=[u_1,u_2,\ldots,u_C] \quad u_c\in\Re^{H\times W}$$

$$u_c=v_c*X=\sum_{s=1}^{C'} v_c^s*x^s$$

Squeeze

$$z \in \Re^C \qquad z_c = F_{sq}(u_c) = \frac{1}{HW}\sum_{i,j} u_c(i,j)$$

Excitation

$$s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2 \delta(W_1 z))$$

$$W_1 \in \Re^{\frac{C}{r} \times C} \qquad W_2 \in \Re^{C \times \frac{C}{r}}$$

Scale

$$\widetilde{x_c} = F_{scale}(u_c, s_c) = s_c u_c$$

$$\widetilde{X} = [\widetilde{x_1}, \widetilde{x_2}, \ldots, \widetilde{x_C}]$$

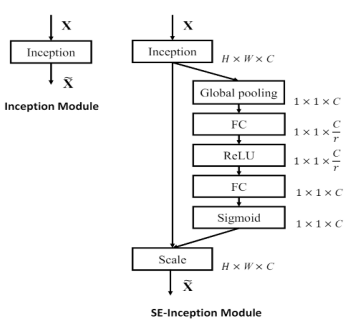


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

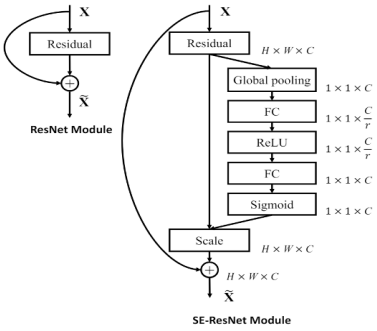


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

CBAM: Convolutional Block Attention Module

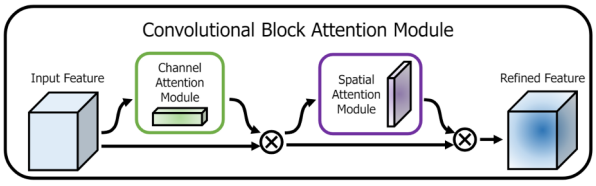
$F \in \Re^{C \times H \times W}$ *Intermediate feature map*

$M_c \in \Re^{C \times 1 \times 1}$ *1D channel attention output*

$M_s \in \Re^{1 \times H \times W}$ *2d spatial attention map*

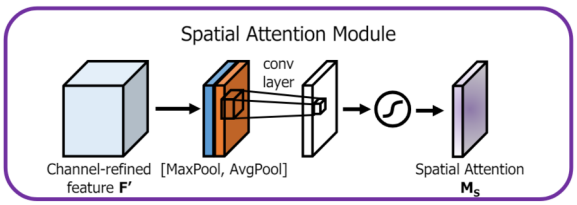
$$F' = M_c(F) \odot F$$

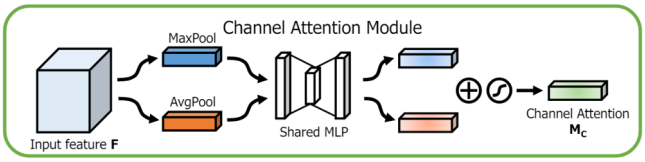
$$F'' = M_s(F') \odot F'$$

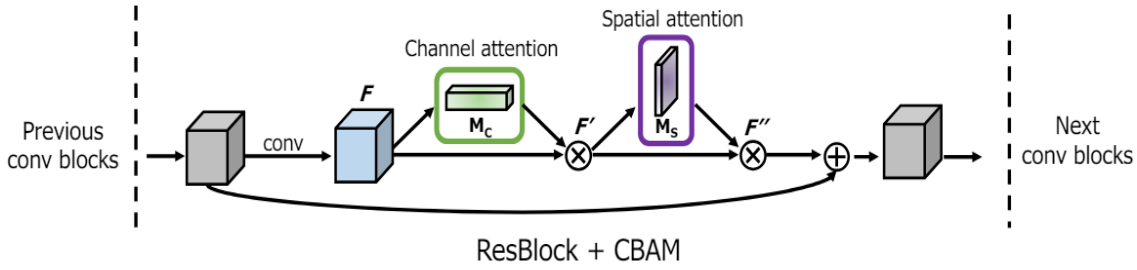


$$M_s(F) = \sigma(f^{7\times7}([AvgPool(F); MaxPool(F)]))$$

$$= \sigma(f^{7\times7}([F^s_{avg}; F^s_{max}]))$$







Random Erasing

Algorithm 1: Random Erasing Procedure

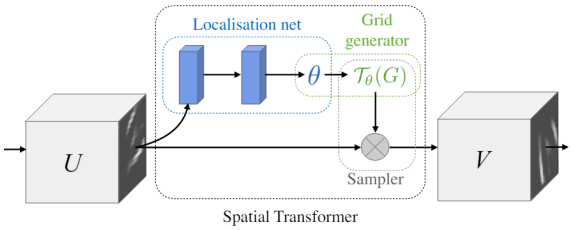
Input : Input image I;
Image size W and H;
Area of image S;
Erasing probability p;
Erasing area ratio range s_l and s_h;
Erasing aspect ratio range r_1 and r_2.
Output: Erased image I^*.
Initialization: $p_1 \leftarrow$ Rand (0, 1).

if $p_1 \geq p$ **then**
  $I^* \leftarrow I$;
  return I^*.
else
  while *True* **do**
    $S_e \leftarrow$ Rand $(s_l, s_h) \times S$;
    $r_e \leftarrow$ Rand (r_1, r_2);
    $H_e \leftarrow \sqrt{S_e \times r_e}$, $W_e \leftarrow \sqrt{\frac{S_e}{r_e}}$;
    $x_e \leftarrow$ Rand $(0, W)$, $y_e \leftarrow$ Rand $(0, H)$;
    if $x_e + W_e \leq W$ *and* $y_e + H_e \leq H$ **then**
      $I_e \leftarrow (x_e, y_e, x_e + W_e, y_e + H_e)$;
      $I(I_e) \leftarrow$ Rand (0, 255);
      $I^* \leftarrow I$;
      return I^*.
    end
  end
end

Spatial Transformers Network

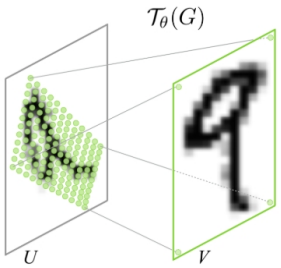
Three main differential blocks:

1. Localization Network
2. Grid Generator
3. Sampler



$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathrm{T}(G_i) = A_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

$$\underbrace{V_i^c}_{\text{Bilinear Sampling}} = \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$



Capsule Networks

Neuron → *capsules* → *layers*

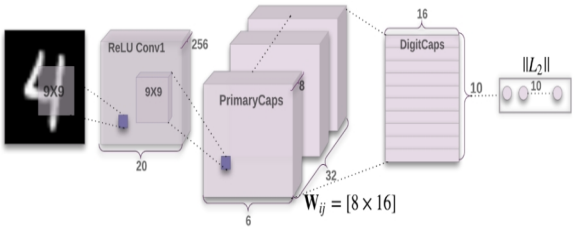
capsules = *group of neurons*

s_j → total input to capsule j

v_j → vector output of capsule j

$\|v_j\|$ → probability that entity represented by the capsule

$\frac{v_j}{\|v_j\|}$ → properties of entity



Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

$$v_j = \underbrace{\frac{\|s_j\|^2}{1+\|s_j\|^2}}_{\text{squashing function}} \frac{s_j}{\|s_j\|}$$

$$s_j = \sum_i c_{ij} u_{j|i}$$

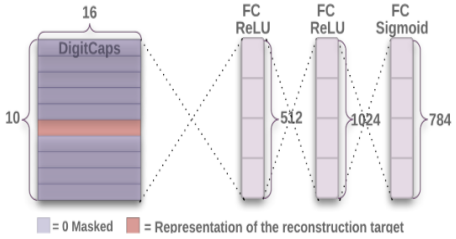
c_{ij} → Coupling coefficients

$$u_{j|i} = W_{ij} u_i$$

$u_{j|i}$ → prediction vector from capsules in layer below

u_i → output of capsule in layer below

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$



b_{ij} → Log prob. that capture i should be coupled with capsule j

Loss function

$$L_k = T_k \max(0, m^+ - \|v_k\|)^2 + \lambda(1-T_k)\max(0, \|v_k\| - m^-)^2$$

$T_k = 1$ iff a digit class of class k is present

$m^+ = 0.9$ and $m^- = 0.1$

$\lambda = 0.5$

Vision Transformers

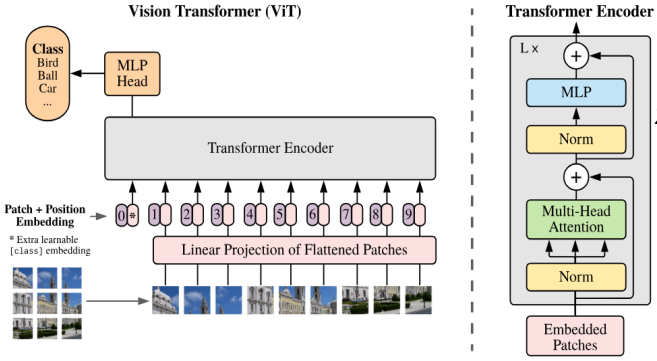


Image Patches ≡ Tokens in NLP

$x \in \Re^{H \times W \times C}$ → *image*

$x_p \in \Re^{N \times (P^2 C)}$ → sequence of flattened 2D patches

$P \times P$ → resoluiton of each image patch

$N = \frac{HW}{P^2}$ → resulting number of patches

D → latent vector size

$$z_0 = [x_{class}; x_p^1 E; x_p^2 E; \ldots; x_p^N E] + E_{pos}$$

$$E \in \Re^{(P^2 C) \times D}, E_{pos} \in \Re^{(N+1) \times D}$$

$$z_l' = MSA(\text{LN}(z_{l-1}) + z_{l-1}), l = 1, \ldots, L$$

$$z_l = MLP(\text{LN}(z_l')) + z_l', l = 1, \ldots, L$$

$$y = \text{LN}(z_L^0)$$

LN → Layer Normalization

MSA → Multihead self-attention

$$z \in \Re^{N \times D}$$
$$[q, k, v] = zU_{qkv}, U_{qkv} \in \Re^{D \times 3D_h}$$
$$A = softmax(qk^T / \sqrt{D_h}), A \in \Re^{N \times N}$$
$$SA(z) = Av$$
$$MSA(z) = [SA_1(z), SA_2(z), \ldots, SA_k(z)]U_{msa}, U_{msa} \in \Re^{kD_h \times D}$$
$$D_h = D/k$$

ML-Mixer

$X \in \Re^{S \times C}$ → real-time input table

S → No. of image patches

C → Hidden Dimension

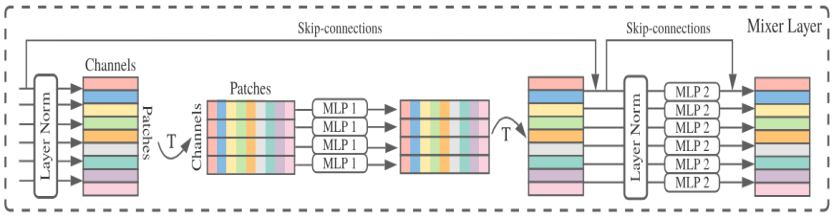
Token Mixing

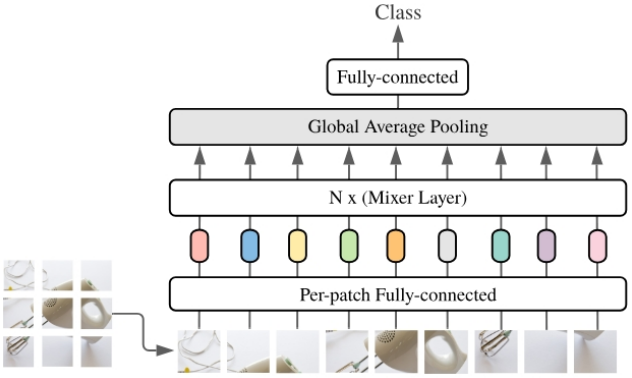
$$U_{*,i} = X_{*,i} + W_2 \sigma(W_1 \mathrm{LN}(X)_{*,i})$$
$$i = 1, \ldots, C$$

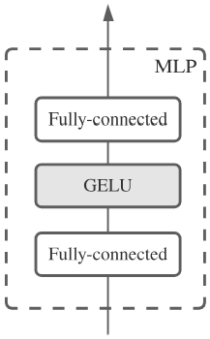
Channel Mixing

$$Y_{j,*} = U_{j,*} + W_4 \sigma(W_3 \mathrm{LN}(U)_{j,*})$$
$$j = 1, \ldots, S$$

σ → GELU(Gaussian Error Linear unit)







Normalizer-Free ResNets

Gradient Clipping

L → $Loss$

θ → model parameters

$G = \frac{\partial L}{\partial \theta}$ → Gradient vector

$$G \rightarrow \begin{cases} \lambda \frac{G}{\|G\|} & if \|G\| > \lambda \\ G & otherwise \end{cases}$$

λ → Clipping threshold

Adaptive Gradient Clipping: $\frac{\|W^l\|_F}{\|G^l\|_F}$ measure of gradient step change with respect to original weight.

$W^l \in \Re^{N \times M}$ → weight matrix of l^{th} layer

$G^l \in \Re^{N \times M}$ → Gradient wrt W^l

$\Delta W^l = -hG^l, h$ → learning rate

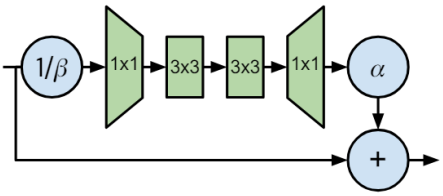
$$\frac{\|\Delta W^l\|_F}{\|W^l\|_F} = h\frac{\|G^l\|_F}{\|W^l\|_F}$$

If $\frac{\|\Delta W^l\|_F}{\|G^l\|_F}$ is large, we expect training to be unstable! Update i[th] row of Matrix G, G_i^l

$$G_i^l \rightarrow \begin{cases} \lambda \frac{\|W_i^l\|_F^*}{\|G_i^l\|_F^*} G_i^l & if \ \frac{\|G_i^l\|_F^*}{\|W_i^l\|_F^*} > \lambda \\ G_i^l & otherwise \end{cases}, \ \|W_i\|_F^* = max(\|W_i\|_F, \epsilon)$$

Normalizer Free ResNet

$h^{l+1} = h^l + \alpha f^l\left(\frac{h^l}{\beta^l}\right) \rightarrow$ Residual Block



We want $Var(f(z)) = Var(z)$

$\beta^l = \sqrt{(Var(h^l))} \quad ; \alpha = 0.2$

Scale Weight Standardization

$$\hat{W}_{ij} = \frac{W_{ij} - \mu_i}{\sqrt{N}\,\sigma_i}$$

$$\mu_i = \frac{1}{N}\sum_j W_{ij}$$

$$\sigma_i^2 = \frac{1}{N}\sum_j (W_{ij} - \mu_i)^2$$

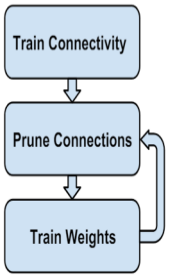
The activation functions are also scaled by non-linearity specific scalar gain γ, $\gamma = \sqrt{2/1 - 1/\pi}$ for ReLU. Combination of γ-scaled activation and scaled weight Standardization is variance preserving.

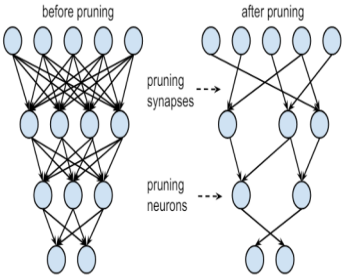
Knowledge Distillation

Knowledge is transferred to distilled model by training it on a transfer set and using soft target distribution for each case in the transfer set that is produced by using the cumbersome model with a high temperature in its softmax. The same high temperature is used when training distilled model, but after it has trained it uses a temperature of 1.

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}; T \rightarrow \text{Temperature}$$

Pruning





Dropout Ratio Adjustment

Do: Original Dropout rate

Dr: Dropout rate during training

C_{ir}: #connections in layer i after retraining.

C_{io}: #connections in layer i for original network.

$$D_r = D_o\sqrt{\frac{C_{ir}}{C_{io}}}$$

C_{io} and C_{ir} vary quadratically with no. Of neurons!

Deep Compression

Network Pruning

All connections with weights below a threshold are removed from the network.

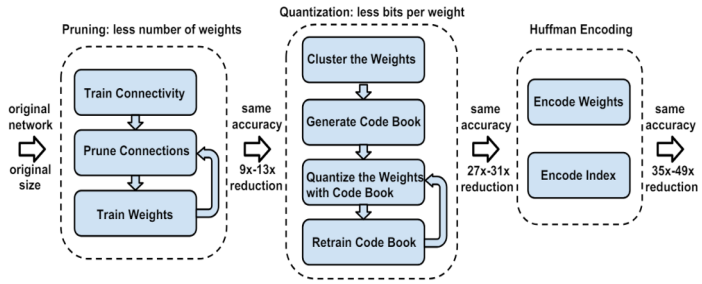
- Compresses Sparse Row
- Compressed Sparse Column

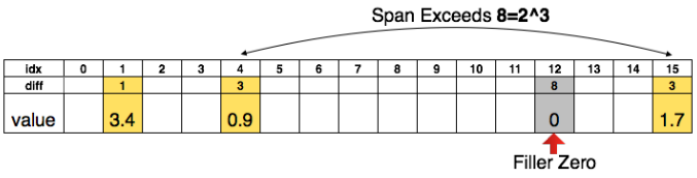
No. of Elements = 2a + n + 1

a: No. of Non-zero elements

n: No. Of Rows

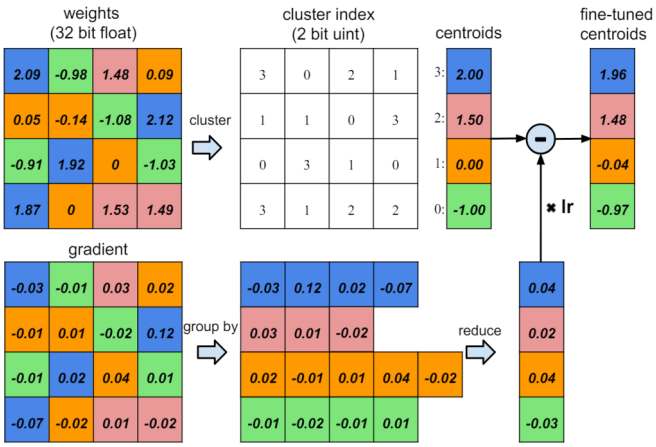
To compress further, store the index difference instead of the absolute position, and encode this difference in 8 bits for conv layer and 5 bits for fc layer.





Trained Quantization and weight sharing

Suppose 4x4 weight matrix. These weights are quantized into 4 bins (by colour). We store the index of weight into the weight matrix. During update, all gradients are grouped by colour, ,multiplied by learning rate and subtracted form centroid.



Compression rate

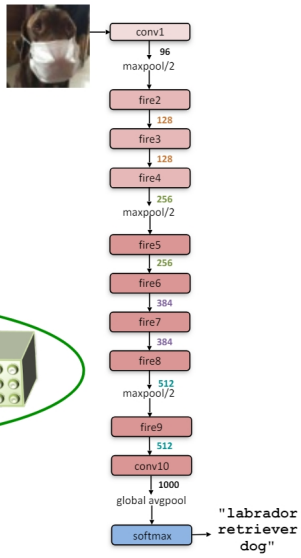
$$r = \frac{nb}{n\log_2(k) + kb}$$

k: #clusters; log2(k): bits required to encode index; n: #weights; b:bits representing each connection

Use k-means cluster to get he centroid. Original weights $W = \{w_1, \ldots, w_n\}$, Clusters $C = \{c_1, \ldots, c_k\}$

Minimize the within-cluster sum of squares (WCSS)

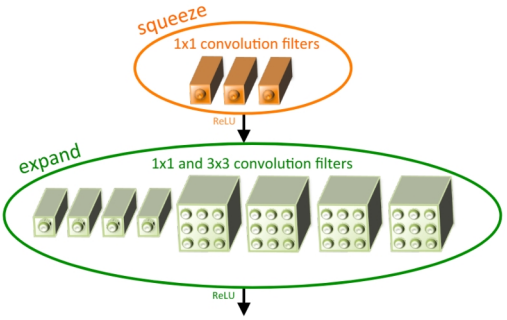
$$\arg\min_{C} \sum_{i=1}^{k} \sum_{w \in c_i} |w - c_i|$$



SqueezeNet

Strategies:

- Replace 3x3 filter with 1x1
- Decrease number of input channels to 3x3 filters
- Downsample late in network so that convolution have large activation map



Fire Module

$s_{1\times1}$ → Number of 1×1 filters in squeeze layer

$e_{1\times1}$ → Number of 1×1 filters in expand layer

$s_{3\times3}$ → Number of 3×3 filters in expand layer

Also used skip connections over module.

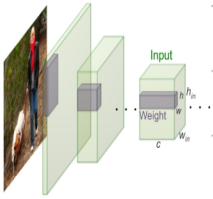
XNOR-Net

Two variations:

Binary weights here the weight matrix is binary tensor.

Estimate weight matrix by $W \approx \alpha B$, B is binary filter $B \in \{-1, +1\}^{c\times w\times h}$, $\alpha \in \Re^{+}$.

Convolution can be approximated as $I * W \approx (I \oplus B)\alpha$. $\oplus$ indicates convolution without multiplication.



Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	+, −, ×	1x	1x	%56.7
Binary Weight	+, −	~32x	~2x	%56.8
BinaryWeight Binary Input (XNOR-Net)	XNOR, bitcount	~32x	~58x	%44.2

For Estimation, assume W, B are vector in $\Re^n$, where $n = c\times w\times h$ and solve

$$J(B,\alpha) = \|W - \alpha B\|^2$$

$$\alpha^*, B^* = \arg\min_{\alpha, B} J(B,\alpha)$$

$J(B,\alpha) = \alpha^2 B^T B - 2\alpha W^T B + W^T W$, $B^TB = n$, W^TW is constant.

$$B^* = \arg\max_{B} W^T B \qquad s.t. B \in \{-1, +1\}^n$$

This optimization can be solved $B^* = \text{sign}(W)$ and $\alpha^* = \frac{W^T B^*}{n}$, replacing B* with sign(W), we get

$$\alpha^* = \frac{1}{n}\|W\|_1$$

Training

Given, W_t, calculate $\alpha_t = \frac{1}{n}\|W_t\|_1$ and $B_t = sign(W_t)$

$$\widetilde{W}_t = \alpha_t B_t$$

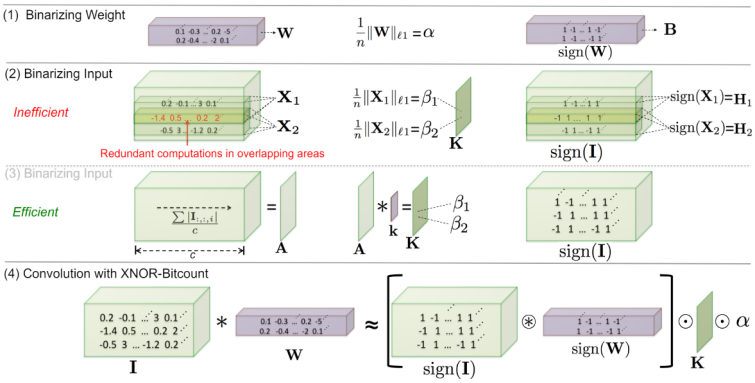
$$W_{t+1} = W_t - \eta \nabla_{\widetilde{W}_t} J$$

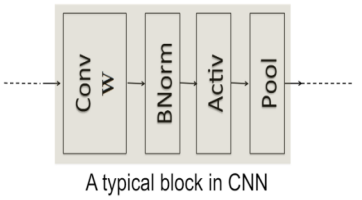
XNOR-Networks here elements of both input feature map and weight matrix are binary.

For efficiency you compute A, which is average of feature map across channels.

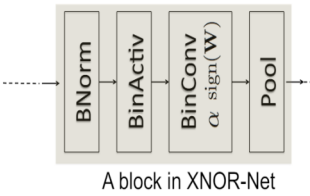
Convolve A with k to get K, where $k_{ij}=1/(w\times h)\,\forall\, i,j$. Hence we can approximate convolution of Image I with Weight W as

$I*W\approx(sign(I)*sign(W))\odot K\,\alpha$





A typical block in CNN



A block in XNOR-Net

BinActiv computes k and I, BinConv does the convolution shown in above figure

MobileNets

During Normal convolution, we have

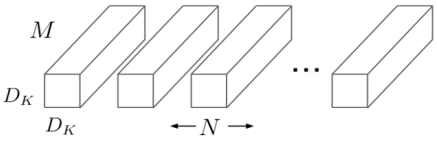
$F\in\Re^{D_F\times D_F\times M}$ → Input feature maps

$G\in\Re^{D_F\times D_F\times N}$ → Output feature maps

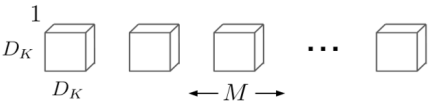
$K\in\Re^{D_K\times D_K\times M\times N}$

$G_{k,l,n}=\sum_{i,j,m} K_{i,j,m,n}F_{k+i-1,l_j-1,m}$

Computational cost: $D_K^2\,MN\,D_F^2$



(a) Standard Convolution Filters



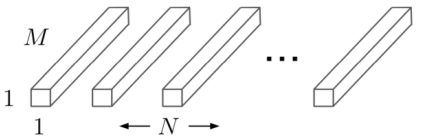
(b) Depthwise Convolutional Filters

Depthwise separable convolution: For every input channel, produces same number of output channels, by using 2D filter of size $D_K\times D_K$

$\hat{G}_{k,l,m}=\sum_{i,j} K_{i,j,m}F_{k+i-1,l_j-1,m}$

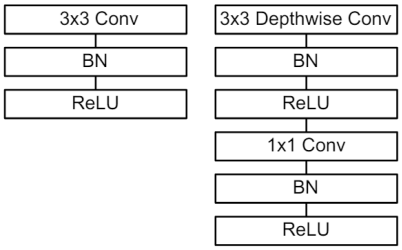
$\hat{K}\in\Re^{D_K\times D_K\times M}$

Computational cost: $D_K^2\,M\,D_F^2$



Pointwise convolution: Uses 1×1 convolution to produce linear combination of input feature maps

Total computation Cost $D_K^2\,M\,D_F^2+MND_F^2$



ShuffleNet

channel shuffle

g → Number of groups

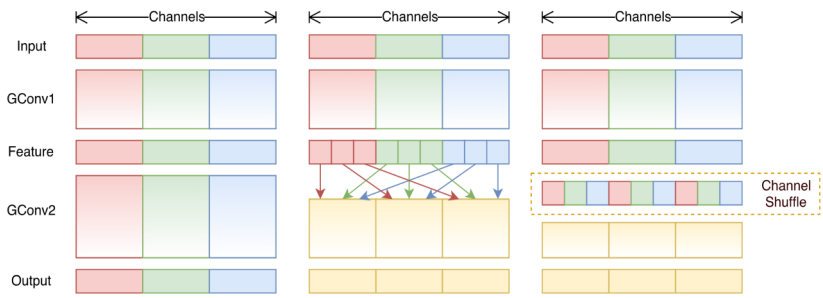
n → Number of channels per group

gn → *number of channels*

reshape to (g,n)

transpose

flatten



Adversarial Examples

$x \in \Re^m \rightarrow$ an input image

$f : \Re^m \rightarrow \{1,2,\ldots,k\}$

For a given $x \in \Re^m$, we try to solve

$$\text{minimize} \|r\|_2$$

$$\text{subject to} \quad \begin{cases} f(x+r) = l \\ x+r \in [0,1]^m \end{cases}$$

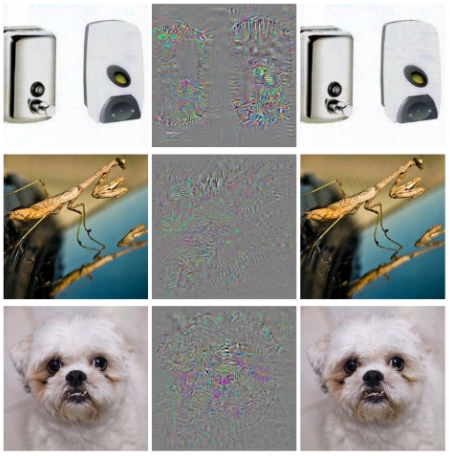
label $l \in \{1,\ldots,k\}$

Informally, $x+r$ is closest image to x classified to l by f. The minimizer is not unique and problem in non-trivial. So we try to solve below problem

$$\text{minimize } c|r| + \text{loss}_f(x+r, l)$$

$$\text{subject to } x+r \in [0,1]^m$$

Box constraint L-BFGS for optimization



Left: correctly predicted sample
Middle: Noise magnified to 10x
Right: Left Image + Noise, predicted as "ostrich, Struthio, camelus"

Fast Gradient Sign Method

$$\underbrace{\tilde{x}}_{\text{Adversarial Input}} = \underbrace{x}_{\text{Input}} + \eta$$

The classifier should assign same class to $\tilde{x}$ and x, as long as $\|\eta\|_\infty < \epsilon$.

$w^T \tilde{x} = w^T x + w^T \eta$ Activation grows by $w^T \eta$

$\eta = sign(w)\epsilon \Rightarrow \|\eta\|_\infty = \epsilon$

$w^T \eta = \sum_{i=1}^{n} w_i \eta_i = \sum_{i=1}^{n} |w_i| \epsilon = \epsilon n m$

'm' is average magnitude of element of weight vector

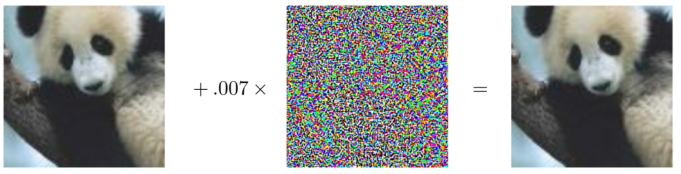
"Fast gradient sign" method of generating adversarial example

$\theta \rightarrow$ parameters of model

$x \rightarrow$ input to the model

$y \rightarrow$ target associated with x $J(\theta, x, y) \rightarrow$ cost

$\eta = sign(\nabla_x J(\theta, x, y))$



$\boldsymbol{x}$
"panda"
57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence

$\boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Adversarial Training

$$\tilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1-\alpha) J(\theta, x + sign(\nabla_x J(\theta, x, y))\epsilon, y)$$

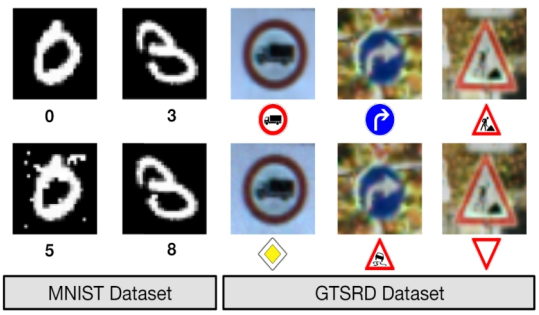
$$\begin{aligned} \tilde{J}(\theta, x, y) &\approx \alpha J(\theta, x, y) + (1-\alpha)\epsilon\, sign(\nabla_x J(\theta, x, y))^T \nabla_x J(\theta, x, y) \\ &= \alpha J(\theta, x, y) + (1-\alpha)\epsilon \|\nabla_x J(\theta, x, y)\|_1 \end{aligned}$$

Black-Box Attacks

The adversary has no information about the structure or parameters of DNN, and does not have access to any large training data. The adversary is only capable of assigning labels assigned by the DNN for chosen input.

Attack Strategy: Train a local substitute DNN with "synthetic" dataset

$O \rightarrow$ Oracle (Targeted DNN)



$x \rightarrow$ input

$\tilde{O}(x) \rightarrow$ label $= arg\,max_j O_j(x)$

$O(x) \rightarrow$ Probability vector

$x^* = x + \delta_x \rightarrow$ Adversarial Example

$\delta_x = arg\,min\{z : \tilde{O}(x_z) \neq \tilde{O}(x)\}$

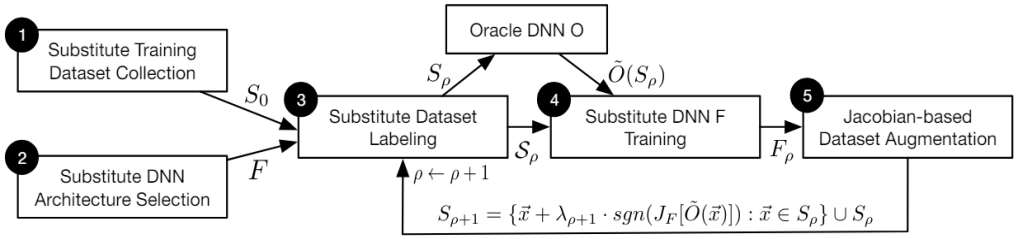
$F \rightarrow$ Subsitute Network (learn similar decision boundary to O)

$J_F(x) = \left[\frac{\partial F_j(x)}{\partial x_i}\right]_{i,j} \rightarrow$ model's Jacobian Matrix

$x + \lambda\, sign(J_F(x)[\tilde{O}(x)])$

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initial training set S_0.

Input: $\tilde{O}$, max_ρ, S_0, λ
1: Define architecture F
2: **for** $\rho \in 0$ .. $max_\rho - 1$ **do**
3: // *Label the substitute training set*
4: $D \leftarrow \left\{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\right\}$
5: // *Train F on D to evaluate parameters θ_F*
6: $\theta_F \leftarrow \text{train}(F, D)$
7: // *Perform Jacobian-based dataset augmentation*
8: $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
9: **end for**
10: **return** θ_F



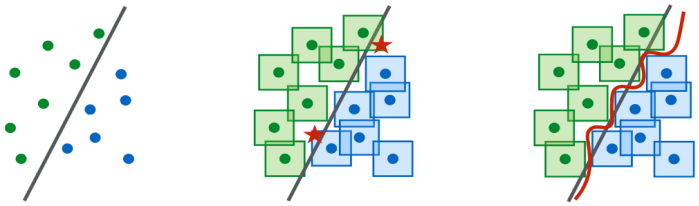
Robust Optimization

Empirical Risk $\min_\theta \rho(\theta) = E_{(x,y)\sim D}\left[\max_{\delta \in S} L(\theta, x+\delta, y)\right]$ $S \subset \Re^d$ Set of allowed perturbation

Fast Gradient sign method, L_∞ bounded adversary $x + \epsilon\, sign(\nabla_x L(\theta, x, y))$

Projected Gradient Ascent $x^{t+1} = \prod_{x+S}\left(x^t + \alpha\, sign(\nabla_x L(\theta, x, y))\right)$

A conceptual illustration of standard vs. adversarial decision boundaries. Left: A set of points that can be easily separated with a simple decision boundary. Middle: The simple decision boundary does not separate the l_∞-balls. Right: Separating the l_∞-balls requires a significantly more complicated decision boundary.



One pixel Attack

$f \rightarrow$ target Image classifier

$x = (x_1, .., x_n) \rightarrow$ original image correctly classified as class t

$f_t(x) \rightarrow$ probability of x belonging to class t

$e(x) = (e_1, \ldots, e_n) \rightarrow$ Additive adversarial perturbation

Black Box Attack

$$\max_{e^*(x)} \quad f_{adv}(x + e(x))$$
$$s.t. \|e(x)\|_0 \leq d, \text{here } d = 1$$

Candidate solution $\{\underbrace{(x_1, y_1, r_1, g_1, b_1)}_{\chi_1}, \ldots, (x_d, y_d, r_d, g_d, b_d)\}$

Initially, $x_i, y_i \sim \underbrace{U(1,32)}_{\text{CIFAR-10}}$ or $\underbrace{U(1,127)}_{\text{ImageNet}}$ and $r_i, g_i, b_i \sim \mathbb{N}(\mu = 128, \sigma = 127)$

$\chi_i(g+1) = \chi_j(g) + \lambda(\chi_k(g) - \chi_l(g))$ $j \neq k \neq l \rightarrow$ random number $g \rightarrow$ Current index of solution

$\lambda = 0.5 \rightarrow$ Scale Parameter

In CIFAR-10, paper did targeted attack while in ImageNet they did untargeted attack.

Fitness function = prob. Label of target class (CIFAR-10)
= prob. Label of the true class (ImageNet)
fitness function > 90% ⇒ stop (CIFAR-10 targeted attack)
fitness function < 5% ⇒ stop (ImageNet, non-targeted attack)
max ≠ iter = 100