Deep Learning for Computer Vision

Dr. Konda Reddy Mopuri
Mehta Family School of Data Science and Artificial Intelligence
IIT Guwahati
Aug-Dec 2022

So far in the class..

- Brief introduction to ML

So far in the class..

- Brief introduction to ML
- Artificial neuron models, Perceptron

So far in the class..

- Brief introduction to ML
- Artificial neuron models, Perceptron
- MLP, CNNs and different families of architecture

So far in the class..

- Brief introduction to ML
- Artificial neuron models, Perceptron
- MLP, CNNs and different families of architecture
- (today) Some of the important training aspects of CNNs

Data preprocessing for Computer vision

- Mean subtraction (e.g. AlexNet: $32 \times 32 \times 3$, VGG: $1 \times 1 \times 3$)

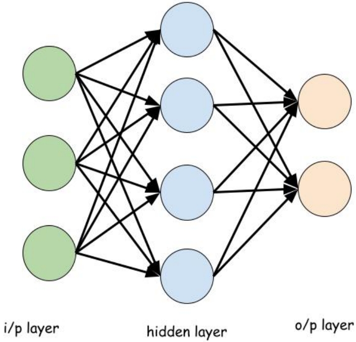
Data preprocessing for Computer vision

- Mean subtraction (e.g. AlexNet: $32 \times 32 \times 3$, VGG: $1 \times 1 \times 3$)
- Mean subtraction and division by standard deviation per channel (e.g. ResNet)

Data preprocessing for Computer vision

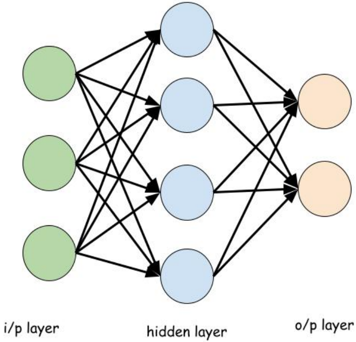
- Mean subtraction (e.g. AlexNet: $32 \times 32 \times 3$, VGG: $1 \times 1 \times 3$)
- Mean subtraction and division by standard deviation per channel (e.g. ResNet)
- PCA or whitening are not common

Weight Initialization



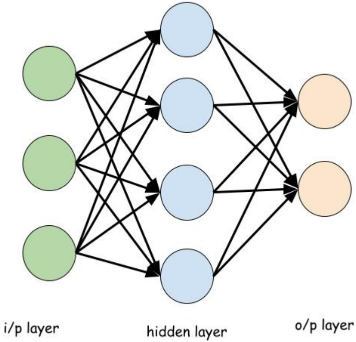
- What if all the parameters are initialized to zero?

Weight Initialization



- What if all the parameters are initialized to zero?
- Or, a different constant?

Weight Initialization



- What if all the parameters are initialized to zero?
- Or, a different constant?
- Leads to a failure mode (often known as the 'symmetry' problem)

Weight Initialization

- How about randomly initializing?
  `W = 0.001 * np.random.randn(`d_l, d_{l-1}`)`

Figure credits: Dr Justin Johnson, U Michigan

Weight Initialization

- How about randomly initializing?
  `W = 0.001 * np.random.randn(`d_l, d_{l-1}`)`
- Okay for the shallow nets

Figure credits: Dr Justin Johnson, U Michigan

Weight Initialization

- How about randomly initializing?
  `W = 0.001 * np.random.randn`(d_l, d_{l-1})
- Okay for the shallow nets
- However, the dynamic range of the activations at later layers goes on shrinking $\rightarrow$ activations tend to zero at deeper layers (e.g. 6 layer MLP with a tanh nonlinearity)

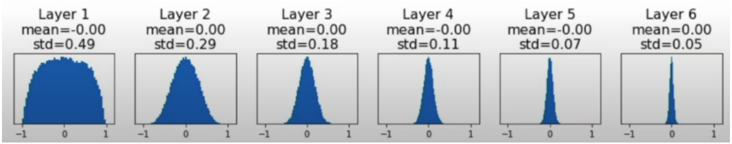
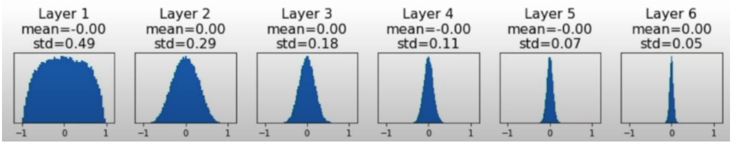


Figure credits: Dr Justin Johnson, U Michigan

Weight Initialization

- How about randomly initializing?
  `W = 0.001 * np.random.randn(`d_l, d_{l-1}`)`
- Okay for the shallow nets
- However, the dynamic range of the activations at later layers goes on shrinking $\rightarrow$ activations tend to zero at deeper layers (e.g. 6 layer MLP with a tanh nonlinearity)



- All zero gradients, no learning!

Figure credits: Dr Justin Johnson, U Michigan

Xavier Initialization

- `W = 0.001 * np.random.randn(`d_l, d_{l-1}`)/np.sqrt(`d_{l-1}`)`

Figure credits: Dr Justin Johnson, U Michigan

Xavier Initialization

- W = 0.001 * np.random.randn((d_l, d_{l-1}))/np.sqrt((d_{l-1}))

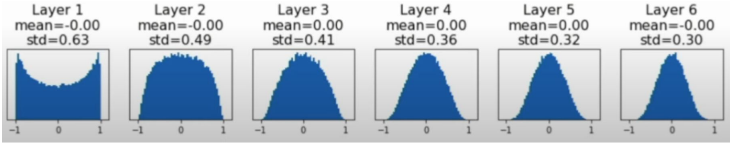
- 

Figure credits: Dr Justin Johnson, U Michigan

Xavier Initialization

- We prefer the o/p to have similar variance as the input

Xavier Initialization

- We prefer the o/p to have similar variance as the input
- Consider a single layer, $y = Wx$, i.e. $y_i = \sum_{j=1}^{d_{l-1}} x_j \cdot w_j$

Xavier Initialization

- We prefer the o/p to have similar variance as the input
- Consider a single layer, $y = Wx$, i.e. $y_i = \sum_{j=1}^{d_{l-1}} x_j \cdot w_j$
- var$(y_i) = d_{l-1} \cdot var(x_i \cdot w_i)$ (Assuming w_i and x_i are i.i.d)

Xavier Initialization

- We prefer the o/p to have similar variance as the input
- Consider a single layer, $y = Wx$, i.e. $y_i = \sum_{j=1}^{d_{l-1}} x_j \cdot w_j$
- var$(y_i) = d_{l-1} \cdot var(x_i \cdot w_i)$ (Assuming w_i and x_i are i.i.d)
- var$(y_i) = d_{l-1} \cdot \left(E({x_i}^2) \cdot E({w_i}^2) - E(x_i)^2 \cdot E(w_i)^2 \right)$ (Assuming x and w are independent)

Xavier Initialization

- We prefer the o/p to have similar variance as the input
- Consider a single layer, $y = Wx$, i.e. $y_i = \sum_{j=1}^{d_{l-1}} x_j \cdot w_j$
- var$(y_i) = d_{l-1} \cdot var(x_i \cdot w_i)$ (Assuming w_i and x_i are i.i.d)
- var$(y_i) = d_{l-1} \cdot \left(E({x_i}^2) \cdot E({w_i}^2) - E(x_i)^2 \cdot E(w_i)^2 \right)$ (Assuming x and w are independent)
- var$(y_i) = d_{l-1} \cdot$ var$(x_i) \cdot$ var(w_i) Assuming (x_i and w_i are zero-mean)

Xavier Initialization

- We prefer the o/p to have similar variance as the input
- Consider a single layer, $y = Wx$, i.e. $y_i = \sum_{j=1}^{d_{l-1}} x_j \cdot w_j$
- var$(y_i) = d_{l-1} \cdot var(x_i \cdot w_i)$ (Assuming w_i and x_i are i.i.d)
- var$(y_i) = d_{l-1} \cdot \left(E({x_i}^2) \cdot E({w_i}^2) - E(x_i)^2 \cdot E(w_i)^2 \right)$ (Assuming x and w are independent)
- var$(y_i) = d_{l-1} \cdot$ var$(x_i) \cdot$ var(w_i) Assuming (x_i and w_i are zero-mean)
- $\rightarrow$ var$(w_i) = \frac{1}{d_{l-1}}$

Weight Initialization with ReLU activations

- Kaiming He or MSRA initialization

Figure credits: Dr Justin Johnson

Weight Initialization with ReLU activations

- Kaiming He or MSRA initialization
- std=sqrt$(2/d_{l-1})$

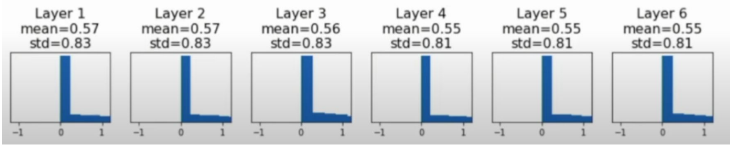
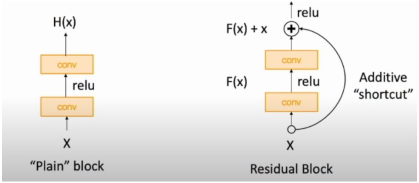


Figure credits: Dr Justin Johnson

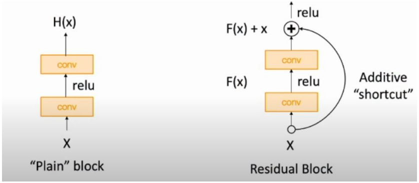
Weight Initialization: Residual Networks



- MSRA initialization: $Var(F(x)+x) > Var(x)$

Figure credits: Dr. Justin Johnson

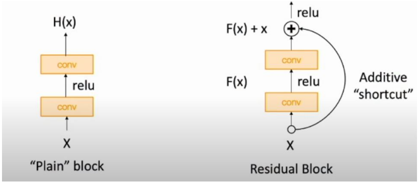
Weight Initialization: Residual Networks



- MSRA initialization: $\mathrm{Var}(F(x)+x) > \mathrm{Var}(x)$
- Variance grows!

Figure credits: Dr. Justin Johnson

Weight Initialization: Residual Networks



- MSRA initialization: $\text{Var}(\text{F}(\text{x})+\text{x}) > \text{Var}(\text{x})$
- Variance grows!
- Solution: Initialize the first Conv layer with MSRA, and the second one with zero $\rightarrow$ $\text{Var}(\text{x}+\text{F}(\text{x})) = \text{Var}(\text{x})$

Figure credits: Dr. Justin Johnson

Deep Regularization

1. Most of the regularization techniques for deep learning are based on regularizing estimators

Deep Regularization

1. Most of the regularization techniques for deep learning are based on regularizing estimators
2. Trade increased bias for decreased variance

Deep Regularization

1. An overly complex model family need not include the target function

Deep Regularization

1. An overly complex model family need not include the target function
2. In practice we almost never have access to the true data generating process, and which is almost certainly outside the model family

Deep Regularization

1. Most often the best-fitting model is a large model that has been appropriately regularized

Deep Regularization

- Parameter Norm penalties (l_2, l_1, etc.)
- Dataset Augmentation
- Noise Robustness
- Semi-Supervised Learning
- Multi-Task Learning (Parameter sharing)
- Sparse Representation
- Dropout
- etc.

Parameter Norm Penalties

1. For neural networks, typically only the weights of the affine transformations are regularized leaving the biases unregularized

Parameter Norm Penalties

1. For neural networks, typically only the weights of the affine transformations are regularized leaving the biases unregularized
2. Bias controls only a single variable as opposed to weight which connects two

Parameter Norm Penalties

1. For neural networks, typically only the weights of the affine transformations are regularized leaving the biases unregularized
2. Bias controls only a single variable as opposed to weight which connects two
3. Regularizing biases may induce underfitting

Parameter Norm Penalties

1. L_2 parameter regularization: $\tilde{\mathcal{J}} = \frac{\alpha}{2} w^T w + \mathcal{J}(w; X, y)$

Parameter Norm Penalties

1. L_2 parameter regularization: $\tilde{\mathcal{J}} = \frac{\alpha}{2} w^T w + \mathcal{J}(w; X, y)$
2. L_1 regularization: $\tilde{\mathcal{J}} = \alpha |w|_1 + \mathcal{J}(w; X, y)$

Parameter Norm Penalties

1. L_2 parameter regularization: $\tilde{\mathcal{J}} = \frac{\alpha}{2} w^T w + \mathcal{J}(w; X, y)$
2. L_1 regularization: $\tilde{\mathcal{J}} = \alpha |w|_1 + \mathcal{J}(w; X, y)$
3. Norm penalties induce different desired behaviors based on the exact penalty imposed

Dataset Augmentation

1. Bestway to make ML model generalize better is to train with more data

Dataset Augmentation

1. Bestway to make ML model generalize better is to train with more data
2. In practice training data is limited

Dataset Augmentation

1. Bestway to make ML model generalize better is to train with more data
2. In practice training data is limited
3. Create fake data and add it to the training data, called Dataset augmentation

Dataset Augmentation

1. Easier for classification

Dataset Augmentation

1. Easier for classification
2. Difficult for density estimation task (unless we have solved the estimation problem)

Dataset Augmentation

1. Has been particularly effective for specific classification problems such as object recognition

Dataset Augmentation

1. Has been particularly effective for specific classification problems such as object recognition
2. Operations such as translation by few pixels, rotating slightly, adding mild noise, etc. greatly improve generalization

Dataset Augmentation

1. Has been particularly effective for specific classification problems such as object recognition
2. Operations such as translation by few pixels, rotating slightly, adding mild noise, etc. greatly improve generalization
3. Hand-designed augmentations in some domains can result in dramatic improvements
4. Should restrict to label preserving transformations

Multi-Task Learning

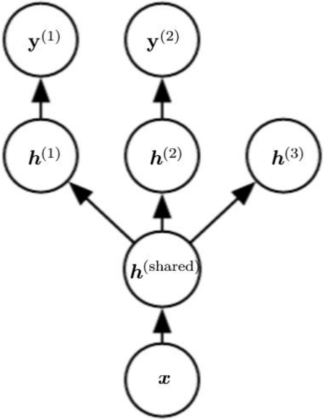
1. Improves generalization by collecting samples arising out of multiple taks

Multi-Task Learning

1. Improves generalization by collecting samples arising out of multiple taks
2. Similar to additional data samples, multi-task samples also put more pressure on the parameters of the shared layers to be better

Multi-Task Learning

1. Improves generalization by collecting samples arising out of multiple taks
2. Similar to additional data samples, multi-task samples also put more pressure on the parameters of the shared layers to be better



Dropout

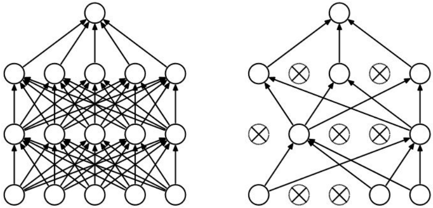
1. Key ideas and contributions in DL have been to engineer architectures for making them easier to train

Dropout

1. Key ideas and contributions in DL have been to engineer architectures for making them easier to train
2. Dropout is one such ('deep') regularization technique (Srivastava et al. 2014)

Dropout

1. During the forward pass, some of the units are randomly 'zeroed' out (neurons are removed)



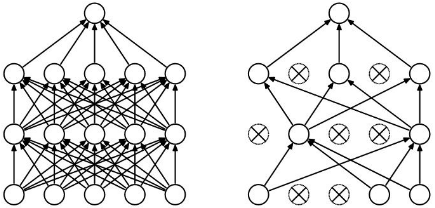
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure from Srivastava et al. 2014

Dropout

1. During the forward pass, some of the units are randomly 'zeroed' out (neurons are removed)
2. Dropped units are randomly selected in each layer independent of others



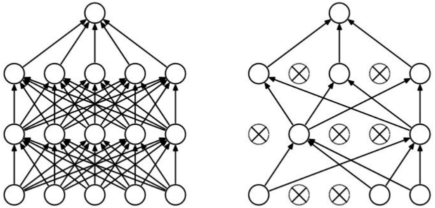
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure from Srivastava et al. 2014

Dropout

1. During the forward pass, some of the units are randomly 'zeroed' out (neurons are removed)
2. Dropped units are randomly selected in each layer independent of others
3. Resulting network has a different architecture



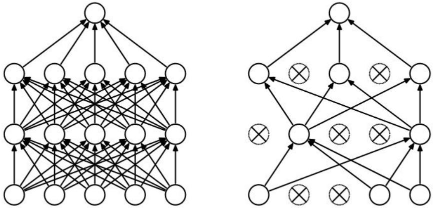
(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure from Srivastava et al. 2014

Dropout

1. During the forward pass, some of the units are randomly 'zeroed' out (neurons are removed)
2. Dropped units are randomly selected in each layer independent of others
3. Resulting network has a different architecture
4. Backpropagation happens through the remaining activations



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

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Figure from Srivastava et al. 2014

Dropout: Interpretation

1. Improves independence between the units (prevents co-adaptation of the units in the network)

Dropout: Interpretation

1. Improves independence between the units (prevents co-adaptation of the units in the network)
2. Distributes the representation among all the units (forces the network to learn redundancy)

Dropout

1. We will decide on which units/layers to use dropout, and with what probability p units are dropped.

Dropout

1. We will decide on which units/layers to use dropout, and with what probability p units are dropped.
2. For each sample, as many Bernoulli variables as units are sampled independently for dropping the units.

Dropout: Another Interpretation

1. Results in a large ensemble of networks (with shared parameters)

Dropout: Another Interpretation

1. Results in a large ensemble of networks (with shared parameters)
2. Every possible binary mask results in a member of the ensemble

Dropout: Another Interpretation

1. Results in a large ensemble of networks (with shared parameters)
2. Every possible binary mask results in a member of the ensemble
3. E.g. a dense layer with 10 units has 2^{10} masks!

Dropout: test time

1. Which model from the ensemble to use?
   $y = f(x, w, m)$ (m is the chosen binary mask)

Dropout: test time

1. Which model from the ensemble to use?
   $y = f(x, w, m)$ (m is the chosen binary mask)
2. How about taking the opinion of all the experts? $\rightarrow$ 'average out' and make the o/p deterministic

Dropout: test time

1. Which model from the ensemble to use?
   $y = f(x, w, m)$ (m is the chosen binary mask)
2. How about taking the opinion of all the experts? $\rightarrow$ 'average out' and make the o/p deterministic
3. $y = \mathbb{E}_m[f(x, w, m)] = \sum_m p(m) \cdot f(x, w, m)$

Dropout: test time

1. Which model from the ensemble to use?
   $y = f(x, w, m)$ (m is the chosen binary mask)
2. How about taking the opinion of all the experts? $\rightarrow$ 'average out' and make the o/p deterministic
3. $y = \mathbb{E}_m[f(x, w, m)] = \sum_m p(m) \cdot f(x, w, m)$
4. Leads to dropping no unit but multiply the activations with the probability of retaining

Dropout: test time

1. Which model from the ensemble to use?
   $y = f(x, w, m)$ (m is the chosen binary mask)
2. How about taking the opinion of all the experts? $\rightarrow$ 'average out' and make the o/p deterministic
3. $y = \mathbb{E}_m[f(x, w, m)] = \sum_m p(m) \cdot f(x, w, m)$
4. Leads to dropping no unit but multiply the activations with the probability of retaining
5. The standard variant uses the 'inverted dropout'. Multiplies activations by $\frac{1}{(1-p)}$ during train and keeps the network untouched during test.

Dropout

1. Which layers to regularize with the Dropout?

Dropout

1. Which layers to regularize with the Dropout?
2. More parameters are the dense layers $\rightarrow$ usually applied there

Dropout

1. Which layers to regularize with the Dropout?
2. More parameters are the dense layers $\rightarrow$ usually applied there
3. Not much used after ResNets!

Batch Normalization (BN)

1. Gradient Descent converges faster with feature scaling $\left(x \leftarrow \frac{x-\mu}{\sigma}\right)$

Batch Normalization (BN)

1. Gradient Descent converges faster with feature scaling $(x \leftarrow \frac{x-\mu}{\sigma})$
2. Batch Normalization (BN) is a normalization method for intermediate layers of NNs $\rightarrow$ performs whitening to the intermediate layer activations

Batch Normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

γ and β are learn-able parameters

Batch Normalization (BN)

1. Originally introduced to handle the internal covariate shift (ICS)

Batch Normalization (BN)

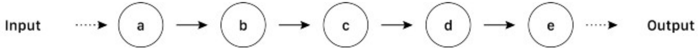
1. Originally introduced to handle the internal covariate shift (ICS)
2. BN makes the activation of each neuron to be Gaussian distributed

Batch Normalization (BN)

1. Originally introduced to handle the internal covariate shift (ICS)
2. BN makes the activation of each neuron to be Gaussian distributed
3. ICS is undesirable because the layers need to adapt to the new distribution of activations
4. With BN, it is reduced to new pair of parameters, but the distribution remains Gaussian

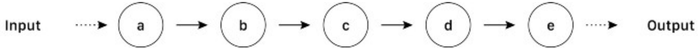
Batch Normalization (BN)

1. Mitigates interdependency between hidden layers during training



Batch Normalization (BN)

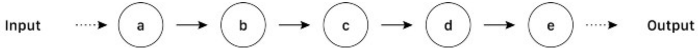
1. Mitigates interdependency between hidden layers during training



2. $\partial(a) = \partial(b) \cdot \partial(c) \cdot \partial(d) \cdot \partial(e)$

Batch Normalization (BN)

1. Mitigates interdependency between hidden layers during training



2. $\partial(a) = \partial(b) \cdot \partial(c) \cdot \partial(d) \cdot \partial(e)$
3. if we want to adjust the input distribution of a specific hidden unit, we need to consider the whole sequence of layers (w/o BN)

Batch Normalization (BN)

1. Mitigates interdependency between hidden layers during training

$$\text{Input} \dashrightarrow a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \dashrightarrow \text{Output}$$

2. $\partial(a) = \partial(b) \cdot \partial(c) \cdot \partial(d) \cdot \partial(e)$
3. if we want to adjust the input distribution of a specific hidden unit, we need to consider the whole sequence of layers (w/o BN)
4. BN acts like a valve which holds back the flow, and allows its regulation using β and γ

Batch Normalization (BN)

1. Reduces training time (less ICS)

Batch Normalization (BN)

1. Reduces training time (less ICS)
2. Reduces the demand for additional regularizers (Batch statistics)

Batch Normalization (BN)

1. Reduces training time (less ICS)
2. Reduces the demand for additional regularizers (Batch statistics)
3. Allows higher learning rates (less danger of vanishing/exploding gradients)

Regularization: General idea

1. Add some randomness during the training

Regularization: General idea

1. Add some randomness during the training
2. Have a mechanism for marginalizing while testing

Regularization: General idea

1. Add some randomness during the training
2. Have a mechanism for marginalizing while testing
3. Some of the instances
   Dropout
   Batch Normalization
   Data Augmentation
   Drop Connect (drop weights instead)
   Fractioinal MaxPooling
   Stochastic Depth
   Mixup
   Cutout
   CutMix, etc.