## Computer Vision

#### Naeemullah Khan

naeemullah.khan@kaust.edu.sa



جامعة الملك عبدالله للعلوم والتقنية King Abdullah University of Science and Technology

KAUST Academy King Abdullah University of Science and Technology

November 19, 2023

## Practical Deep Learning



- Practical Implementation of Deep Learning algorithms is just as much an art as it is a science.
- The main takeaways if not to start from scratch rather to build on top of the previous knowledge.
- ► Today, we will look at some important tools used in the practical implementation of Deep Learning algorithms.

### Outline



- Data Handling
- ► Data Augmentation
- ► Transfer Learning
- Ensembling
- Dropout
- ► Batch Normalization

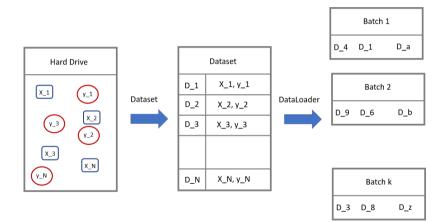
#### **DataLoaders**



- ► As we have previously established that Deep Learning has been made possible by large amount of data and computational resourse
- An important aspect to keep in mind is the data handling:
  - How do we handle large amounts of data?
  - How to we read different components of data (from possible different parts of our hard drive) and provide it to our training algorithms?
  - How do we feed this data to SGD algorithms in a streamlined manner?
- PyTorch provides Dataset and DataLoaders to handle data in an efficient manner.
- We will extend the Dataset and DataLoaders class to construct our own Dataloaders

## DataLoaders (cont.)





## **Data Augmentation**



- Data is the fundamental building block of any machine learning algorithm
- In several applications we don't have access to unlimited data
- So, we use Data Augmentation techniques to improve the performance of our models
- Note: It is better to spend time on data rather than fine-scale architecture search in deep learning

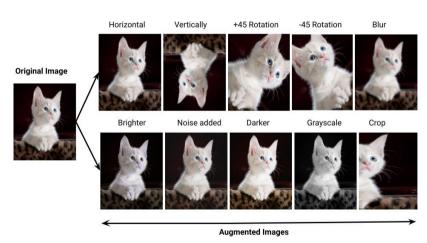
# Data Augmentation (cont.)



- Create virtual training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - · Geometric distortion
  - Translation
  - Rotation

## Data Augmentation (cont.)





<sup>&</sup>lt;sup>0</sup>https://pranjal-ostwal.medium.com/data-augmentation-for-computer-vision-b88b818b6010

## Ensembling



- ► Team-work is the best policy
- ► Multiple networks for the same task
- Max Voting for final classification

## Ensembling - A simple Analysis



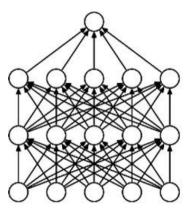
- ► Let's assume that we have a test dataset with *N* elements and an ensemble of *M* models.
- Also assume that the probability of error of the label for an image on a model in the ensemble is denoted by p(e) and is i.i.d
- For an example assume M = 3 and e = 0.1
- Then probability of error of label for the max voting ensemble will be

$$p(e) = e^3 + {3 \choose 2} e^2 (1 - e)$$

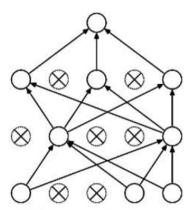
For the above example p(e) = 0.028, which is significantly lower than a single model

## Dropout





(a) Standard Neural Net

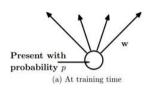


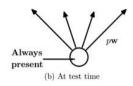
(b) After applying dropout.

<sup>&</sup>lt;sup>0</sup>Srivastava JMLR 2014

## Dropout







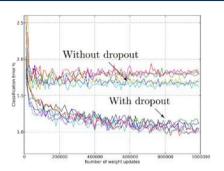
#### **Intuition:** successful conspiracies

- 50 people planning a conspiracy
- Strategy A: plan a big conspiracy involving 50 people
  - Likely to fail. 50 people need to play their parts correctly.
- Strategy B: plan 10 conspiracies each involving 5 people
  - Likely to succeed!

Main Idea: approximately combining exponentially many different neural network architectures efficiently

## Dropout





| Model  | Top-1<br>(val) | Top-5<br>(val) | Top-5<br>(test) |
|--|----------------|----------------|-----------------|
| SVM on Fisher Vectors of Dense SIFT and Color Statistics | - 2            |                | 27.3            |
| Avg of classifiers over FVs of SIFT, LBP, GIST and CSIFT | +              |                | 26.2            |
| Conv Net + dropout (Krizhevsky et al., 2012)             | 40.7           | 18.2           | -               |
| Avg of 5 Conv Nets + dropout (Krizhevsky et al., 2012)   | 38.1           | 16.4           | 16.4            |

Table 6: Results on the ILSVRC-2012 validation/test set.



14 / 31

- ightharpoonup Consider a single layer y = Wx
- ► The following could lead to tough optimazation
  - Inputs x are not centered around zero (need large bias)
  - Inputs x have different scaling per element (entries in W will need to vary a lot)



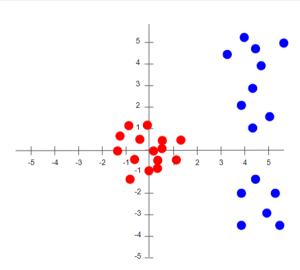
15 / 31

- ightharpoonup Consider a single layer y = Wx
- ► The following could lead to tough optimazation
  - Inputs x are not centered around zero (need large bias)
  - Inputs x have different scaling per element (entries in W will need to vary a lot)
- Idea: Force inputs to be "nicely scaled" at each layer!

## Normalization



16 / 31





 Consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$



18 / 31

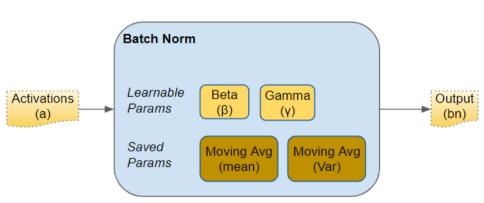
Consider a batch of activations at some layer. To make each dimension zero-mean unit-variance, apply:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}}$$

Problem: What if zero-mean, unit variance is too hard of a constraint?



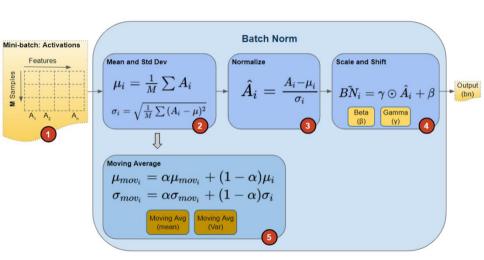
19 / 31



https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739 🔗 a, 🔈



20 / 31



https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739 🔊 🧠 🕒



Input: 
$$x: N \times D$$

# Learnable scale and shift parameters:

$$\gamma, \beta: D$$

Learning  $\gamma$  =  $\sigma$  ,  $\beta$  =  $\mu$  will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j} \quad \begin{array}{ll} \text{Per-channel mean,} \\ \\ \sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2 \quad \begin{array}{ll} \text{Per-channel var,} \\ \\ \text{shape is D} \end{array}$$
 
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \quad \begin{array}{ll} \text{Normalized x,} \\ \\ \text{Shape is N x D} \end{array}$$
 
$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j \quad \begin{array}{ll} \text{Output,} \\ \\ \text{Shape is N x D} \end{array}$$

KAUST Academy Computer Vision November 19, 2023 21 / 31



Estimates depend on minibatch; can't do this at test-time!

Input: 
$$x: N \times D$$

# Learnable scale and shift parameters:

$$\gamma, \beta: D$$

Learning  $\gamma = \sigma$ ,  $\beta = \mu$  will recover the identity function!

$$\mu_j = \frac{1}{N} \sum_{i=1}^N x_{i,j} \quad \text{Per-channel mean,} \\ \sigma_j^2 = \frac{1}{N} \sum_{i=1}^N (x_{i,j} - \mu_j)^2 \quad \text{Per-channel var,} \\ \text{shape is D}$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}} \quad \begin{array}{l} \text{Normalized x,} \\ \text{Shape is N x D} \end{array}$$
 
$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j \quad \begin{array}{l} \text{Output,} \\ \text{Shape is N x D} \end{array}$$



23 / 31

Input:  $x: N \times D$ 

# Learnable scale and shift parameters:

$$\gamma, \beta: D$$

During testing batchnorm becomes a linear operator! Can be fused with the previous fully-connected or conv layer

$$\mu_j=rac{ ext{(Running) average of}}{ ext{values seen during training}}$$

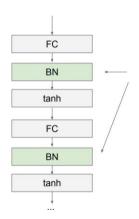
$$\sigma_j^2 = \frac{\text{(Running) average of }}{\text{values seen during training}}$$

Per-channel mean, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$





Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathbf{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$



Batch Normalization for fully-connected networks

$$x: N \times D$$

Normalize

$$\mu,\sigma$$
: 1 × D

$$y = \gamma(x-\mu)/\sigma+\beta$$

**Batch Normalization for** convolutional networks (Spatial Batchnorm, BatchNorm2D)

Normalize

$$\mu, \sigma: 1 \times C \times 1 \times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$



26 / 31

#### Advantages:

- Makes deep networks much easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!



27 / 31

#### Advantages:

- Makes deep networks much easier to train!
- · Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!

#### Disadvantages:

 Behaves differently during training and testing: this is a very common source of bugs!



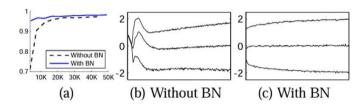


Figure 1: (a) The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy. (b, c) The evolution of input distributions to a typical sigmoid, over the course of training, shown as {15, 50, 85} th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.

## Things to Remember



- ► Training Deep Networks
  - Dropout
  - Data augmentation
  - Activation
  - · Batch normalization
- ► Transfer learning
  - Use Fine-tuning when possible

## Full Deep Learning Pipeline



- Data Pre-processing
- Architecture
- ► Loss
- Optimizer
- DataLoaders
- Data Augmentation
- ► Fine-Tuning
- Ensembling