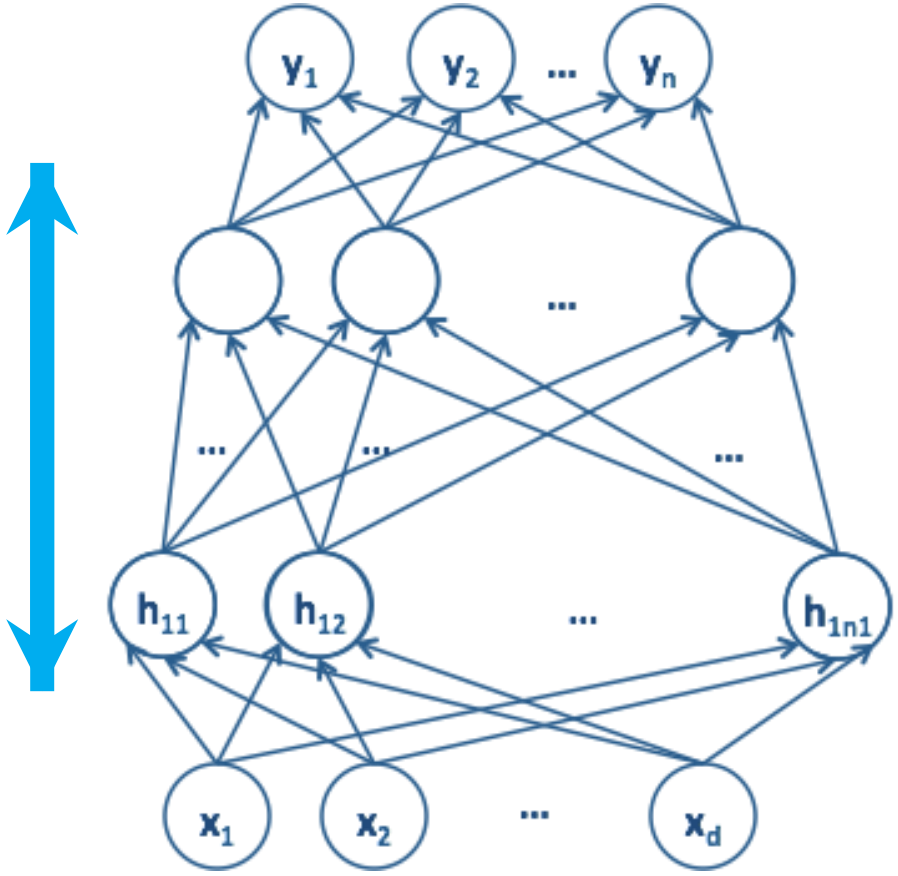
greatlearning

Deep Learning (for Computer Vision)

Arjun Jain

Previously: Building Blocks - Deep Neural Networks

greatlearning



- Feed forward
- Back propagation
- Fully connected layer
- Activation functions
- Softmax function
- Cross-entropy loss

Neural Network Constructed

greatlearning

Now that we have seen all the building blocks that define a neural networks, we need to understand:

- Data preprocessing
- Data augmentation
- Weight initialization
- Regularization

… and finally, to build a well-functioning network

- How to baby-sit the learning process

Neural Network Constructed

greatlearning

Now that we have seen all the building blocks that define a neural networks, we need to understand:

- Data preprocessing
- Data augmentation
- Weight initialization
- Regularization

… and finally, to build a well-functioning network

- How to baby-sit the learning process

Data Preprocessing

Data Preprocessing

- Assume **X** [NxD] is data matrix, each example in a row
- So, in our case we have 10 examples, each 4D

```
X = np.random.rand(10,4)
X
```

```
array([[0.45950097, 0.92404763, 0.22137901, 0.13240392],
       [0.90266789, 0.29224984, 0.60349916, 0.46773406],
       [0.59123575, 0.77915078, 0.07000597, 0.70054545],
       [0.12959827, 0.35025829, 0.90224388, 0.39091937],
       [0.35720569, 0.79027443, 0.51664116, 0.37450996],
       [0.4644068 , 0.25874088, 0.86884088, 0.12713947],
       [0.27220189, 0.07395416, 0.62251387, 0.97576962],
       [0.41323253, 0.01654946, 0.54861887, 0.05727298],
       [0.83307517, 0.10374767, 0.34526809, 0.73658896],
       [0.68461745, 0.33583938, 0.50562829, 0.81160747]])
```

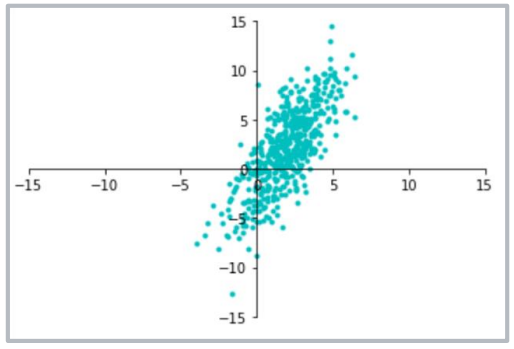
```
mean = np.array([np.mean(X, axis=0)])
mean
```

```
array([[0.51077424, 0.39248125, 0.52046392, 0.47744913]])
```

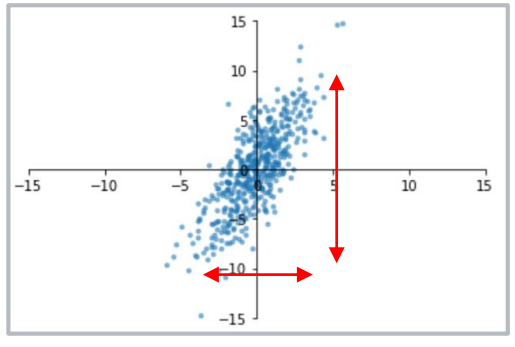
```
std = np.array([np.std(X, axis=0)])
std
```

```
array([[0.23113686, 0.30812185, 0.2466813 , 0.30222671]])
```

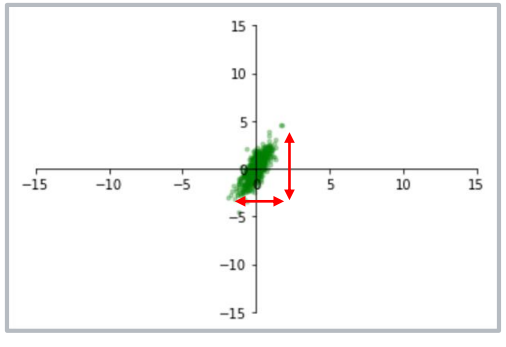
Original Data



Zero-centered Data



Normalized Data



greatlearning

Data Preprocessing

- Assume **X** [NxD] is data matrix, each example in a row
- So, in our case we have 10 examples, each 4D

```
X = np.random.rand(10,4)
mean = np.array([np.mean(X, axis=0)])
std = np.array([np.std(X, axis=0)])

mean_repeated = np.repeat(mean,[10], axis=0)
std_repeated = np.repeat(std, [10], axis=0)
```

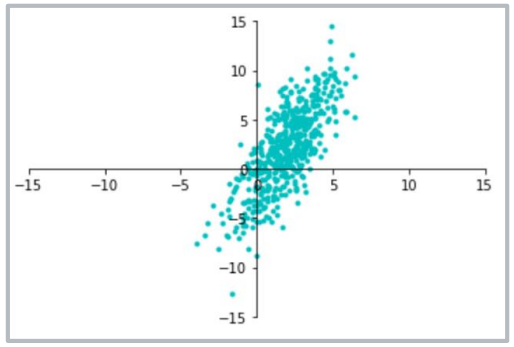
```
mean_repeated
```

```
array([[0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913],
       [0.51077424, 0.39248125, 0.52046392, 0.47744913]])
```

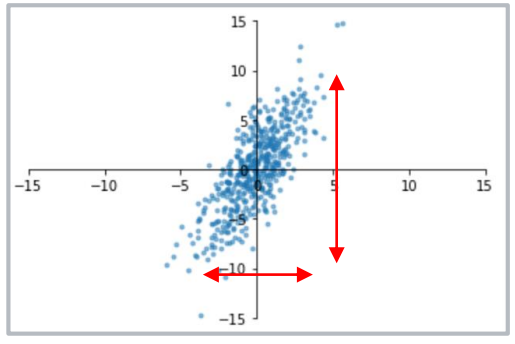
```
std_repeated
```

```
array([[0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671],
       [0.23113686, 0.30812185, 0.2466813 , 0.30222671]])
```

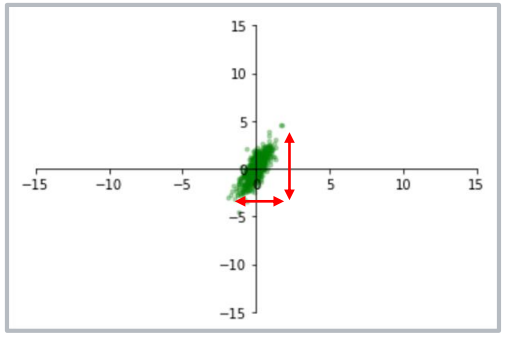
Original Data



Zero-centered Data



Normalized Data



Data Preprocessing

- Assume **X** [NxD] is data matrix, each example in a row
- So, in our case we have 10 examples, each 4D

```
X = np.random.rand(10,4)
mean = np.array([np.mean(X, axis=0)])
std = np.array([np.std(X, axis=0)])

mean_repeated = np.repeat(mean,[10], axis=0)
std_repeated = np.repeat(std, [10], axis=0)
```

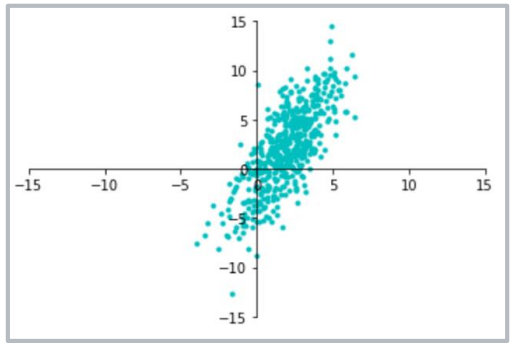
```
X = X - mean_repeated
```

```
X = X / std_repeated
```

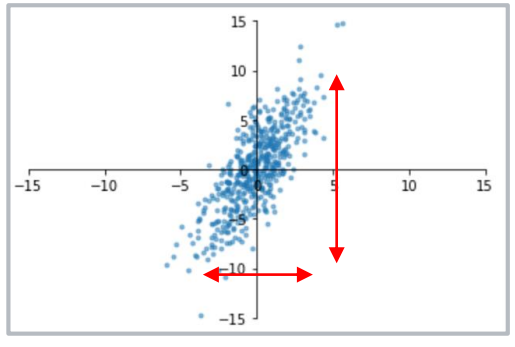
```
X
```

```
array([[-0.22183079,  1.72518233, -1.21243448, -1.14167674],
       [ 1.69550476, -0.32529797,  0.33660938, -0.03214495],
       [ 0.34811198,  1.25492408, -1.82607254,  0.73817542],
       [-1.64913533, -0.13703332,  1.54766481, -0.28630746],
       [-0.66440528,  1.29102554, -0.01549674, -0.34060249],
       [-0.20060599, -0.43405027,  1.41225527, -1.15909562],
       [-1.0321692 , -1.0337699 ,  0.4136915 ,  1.6488301 ],
       [-0.42200846, -1.22007506,  0.11413492, -1.39026808],
       [ 1.39441594, -0.93707597, -0.71021125,  0.85743525],
       [ 0.75212238, -0.18382945, -0.06014086,  1.10565458]])
```

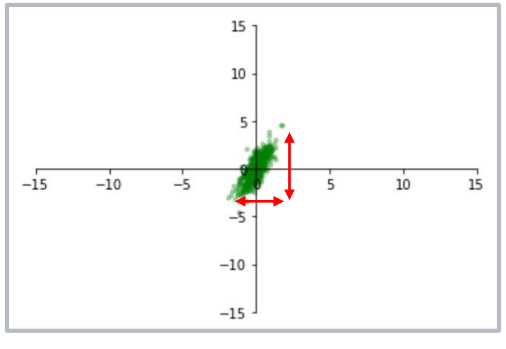
Original Data



Zero-centered Data



Normalized Data

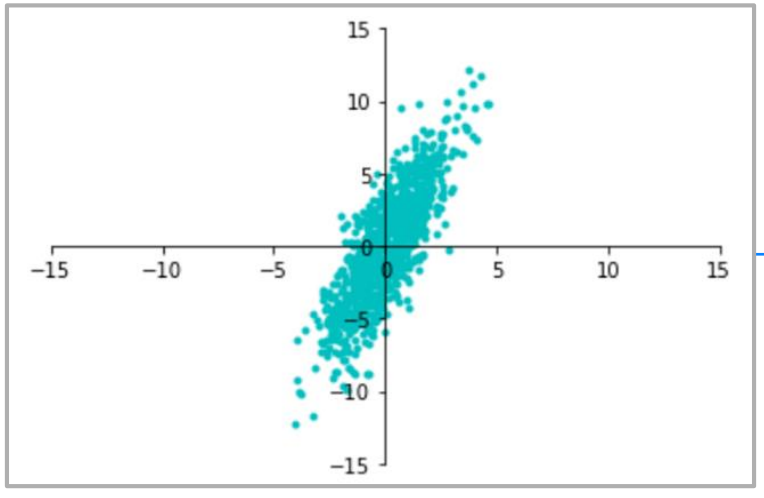


greatlearning

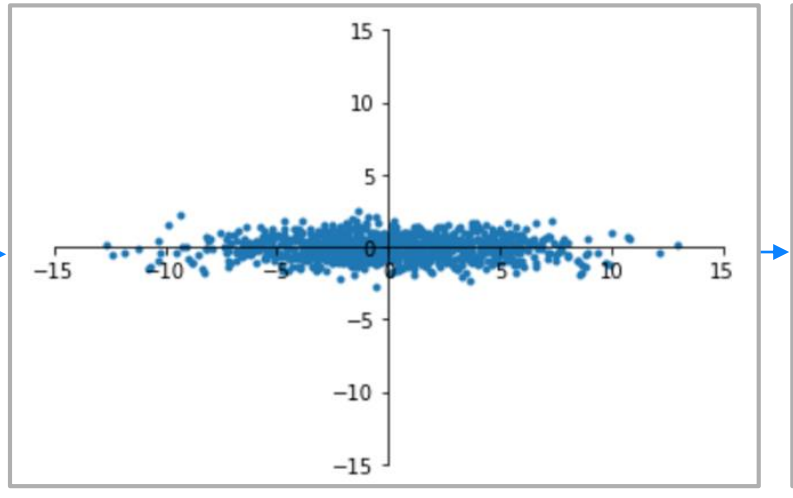
Data Preprocessing

greatlearning

Original Data

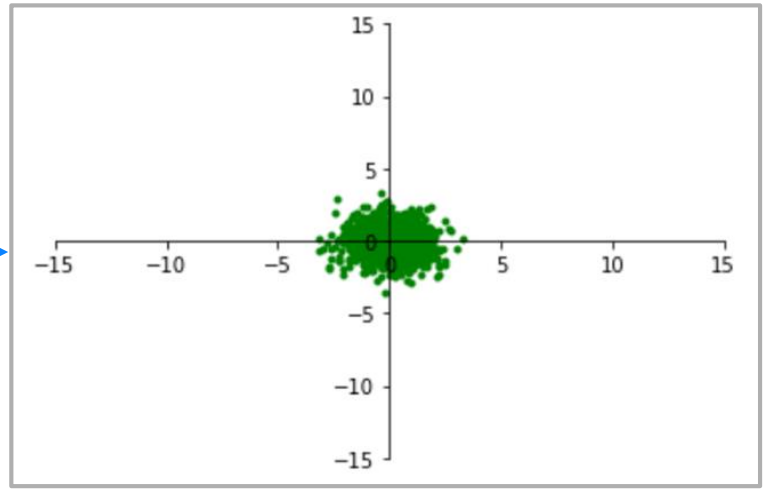


Decorrelated Data



data has diagonal covariance matrix

Whitened Data



covariance matrix is the identity matrix

In practice, you may also see **PCA** and **Whitening** of the data

Data Preprocessing

greatlearning

TL;DR: In practice for Images: center only

e.g. consider MNIST example with [28,28] images

- Subtract the mean image (e.g. AlexNet)

  (mean image = [28,28] array)
- Subtract per-channel mean (e.g. VGGNet)

  (mean along each channel = 1 numbers)

In practice, whitening, PCA and normalizing variance are not commonly used

Neural Network Constructed

greatlearning

Now that we have seen all the building blocks that define a neural networks, we need to understand:

- Data preprocessing
- Data augmentation
- Weight initialization
- Regularization

… and finally, to build a well-functioning network

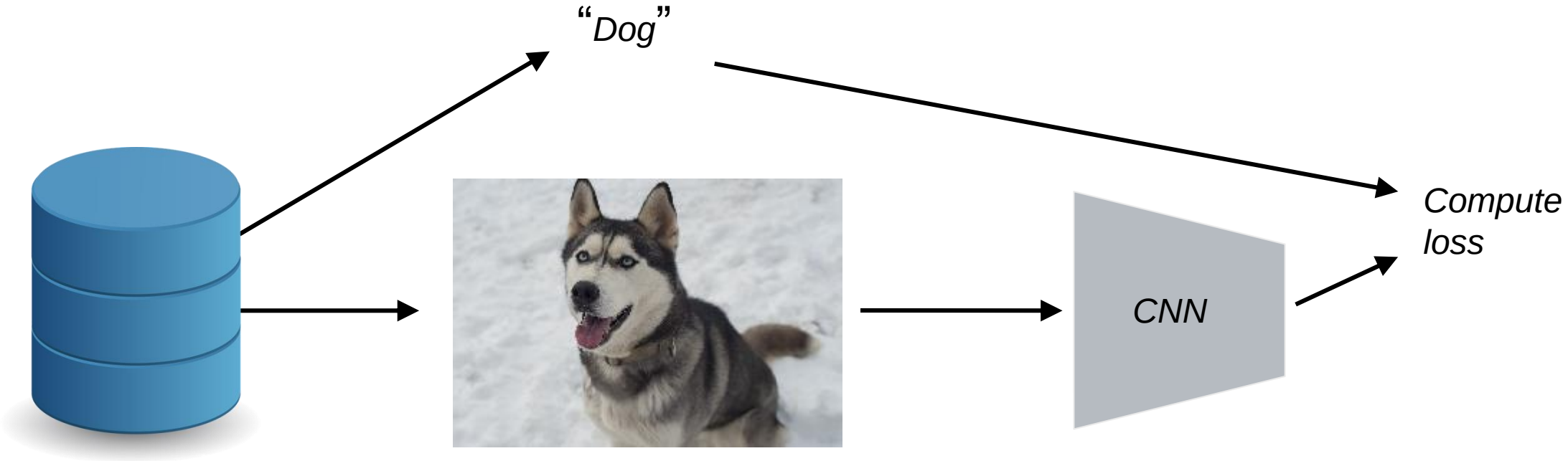
- How to baby-sit the learning process

Data Augmentation

Data Augmentation

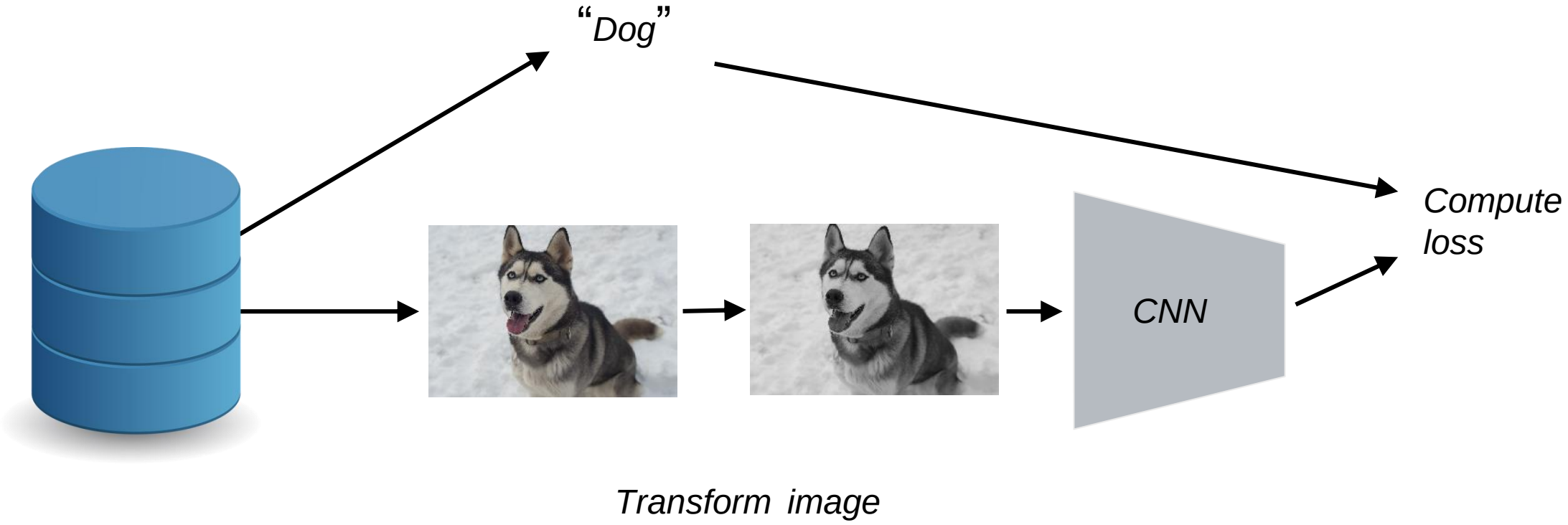
greatlearning

Load image and label



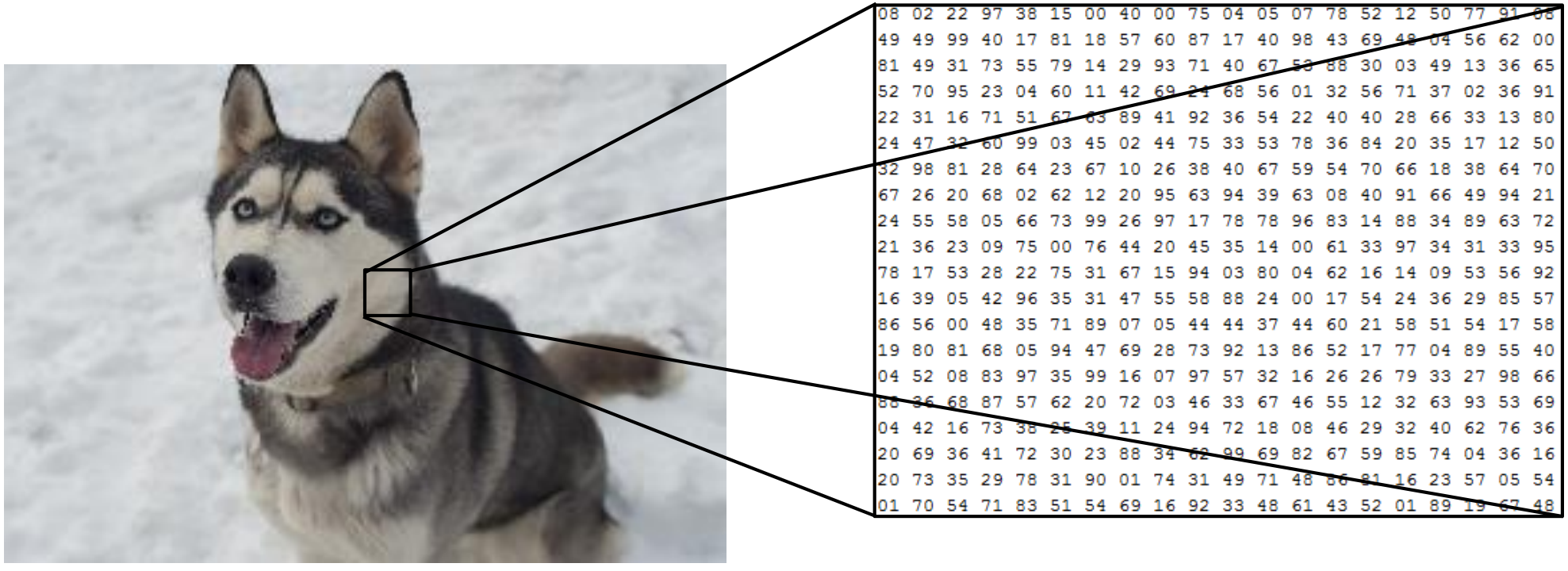
Data Augmentation

greatlearning



Data Augmentation

greatlearning



What the computer sees

Data Augmentation

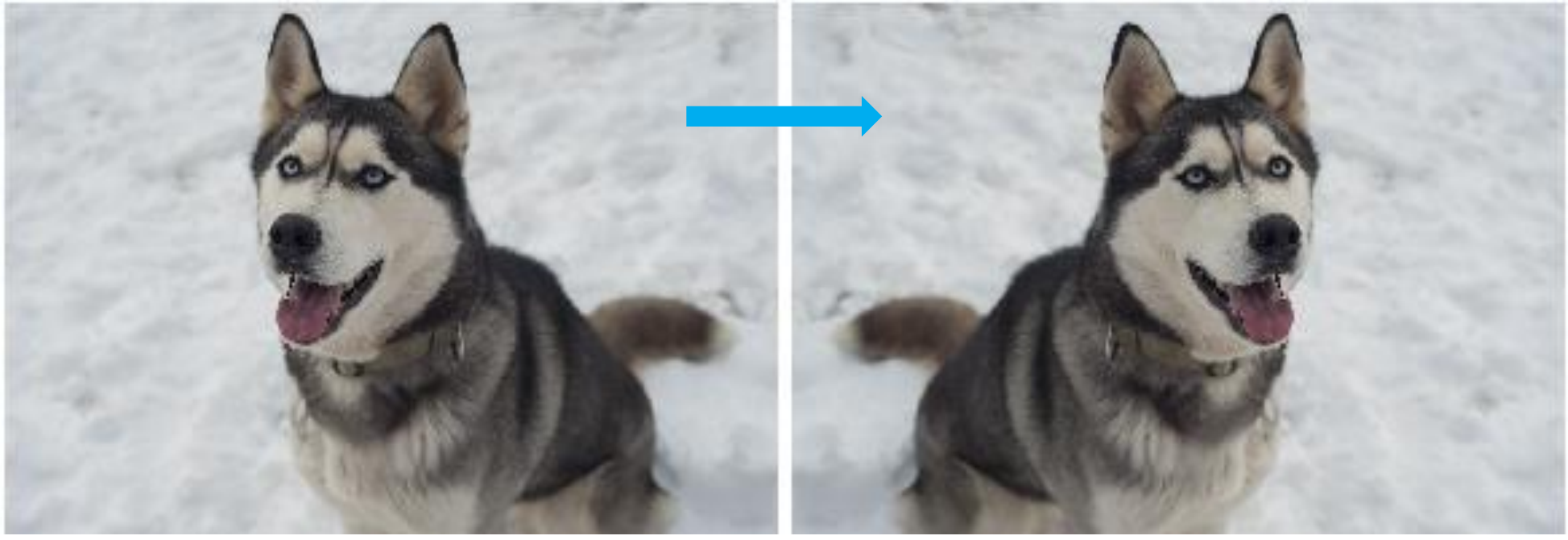
greatlearning

- Horizontal flips
- Rotation
- Crop/Scale
- Color Jitter
- Other Creative Techniques

Data Augmentation

greatlearning

1. **Horizontal flips**
   This scenario is more important for network to remove biasness of assuming certain features of the object is available in only a particular side

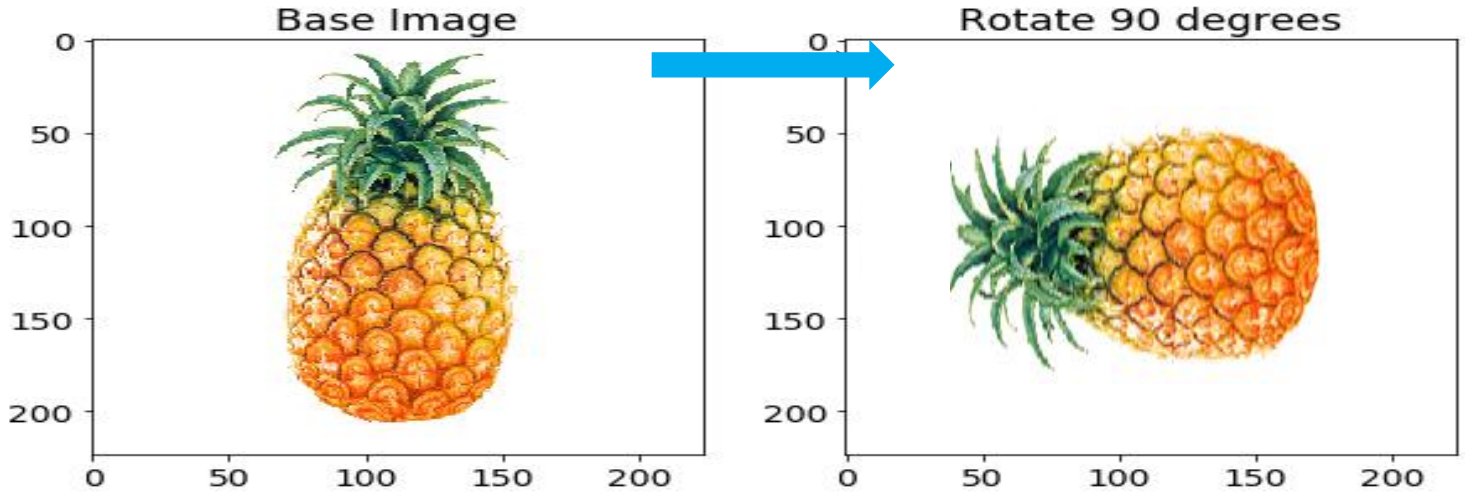


Data Augmentation

greatlearning

2. Rotations

- The network has to recognize the object present in any orientation. Assuming the image is square, rotating the image at 90 degrees will not add any background noise in the image.
- Depending upon the requirement, there maybe a necessity to orient the object at minute angles. However problem with this approach is, it will add background noise.



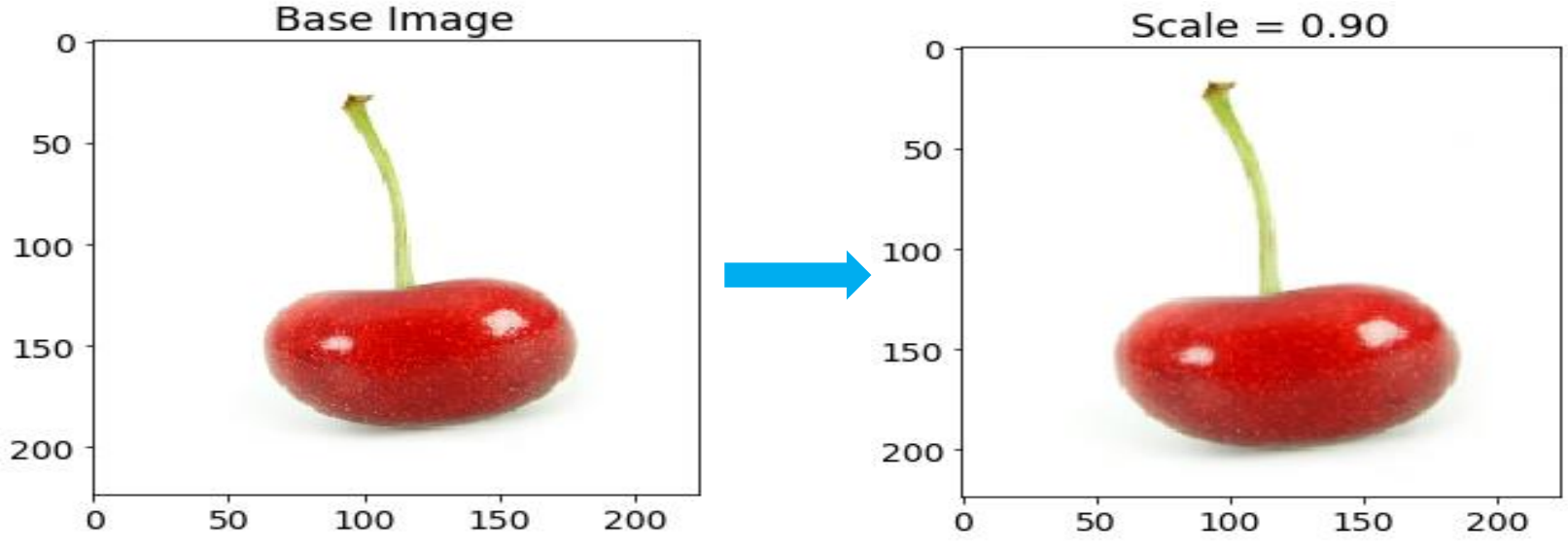
Data Augmentation

greatlearning

3. Random crops/scales

Scaling : The image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size

Image Example



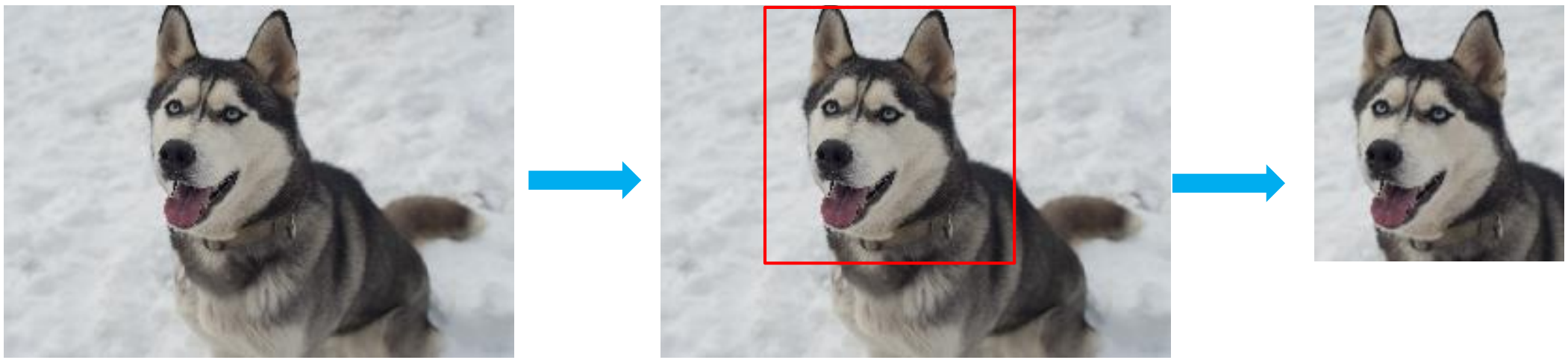
Data Augmentation

greatlearning

4. Random crops/scales

Cropping :Unlike scaling, we just randomly sample a section from the original image. We then resize this section to the original image size. This method is popularly known as random cropping.

Image Example



Data Augmentation

greatlearning

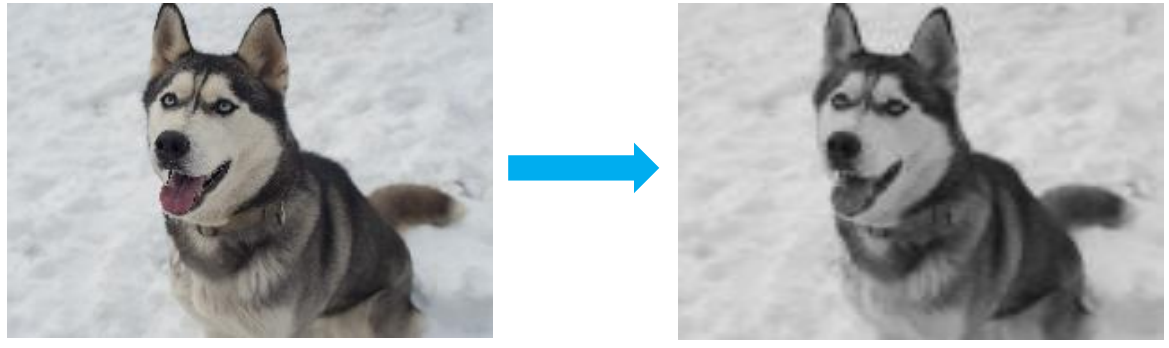
Simple:
Randomly jitter contrast

Complex:

1. Apply PCA to all [R, G, B] pixels in training set

2. Sample a “color offset” along principal component directions

1. Add offset to all pixels of a training image



Data Augmentation

greatlearning

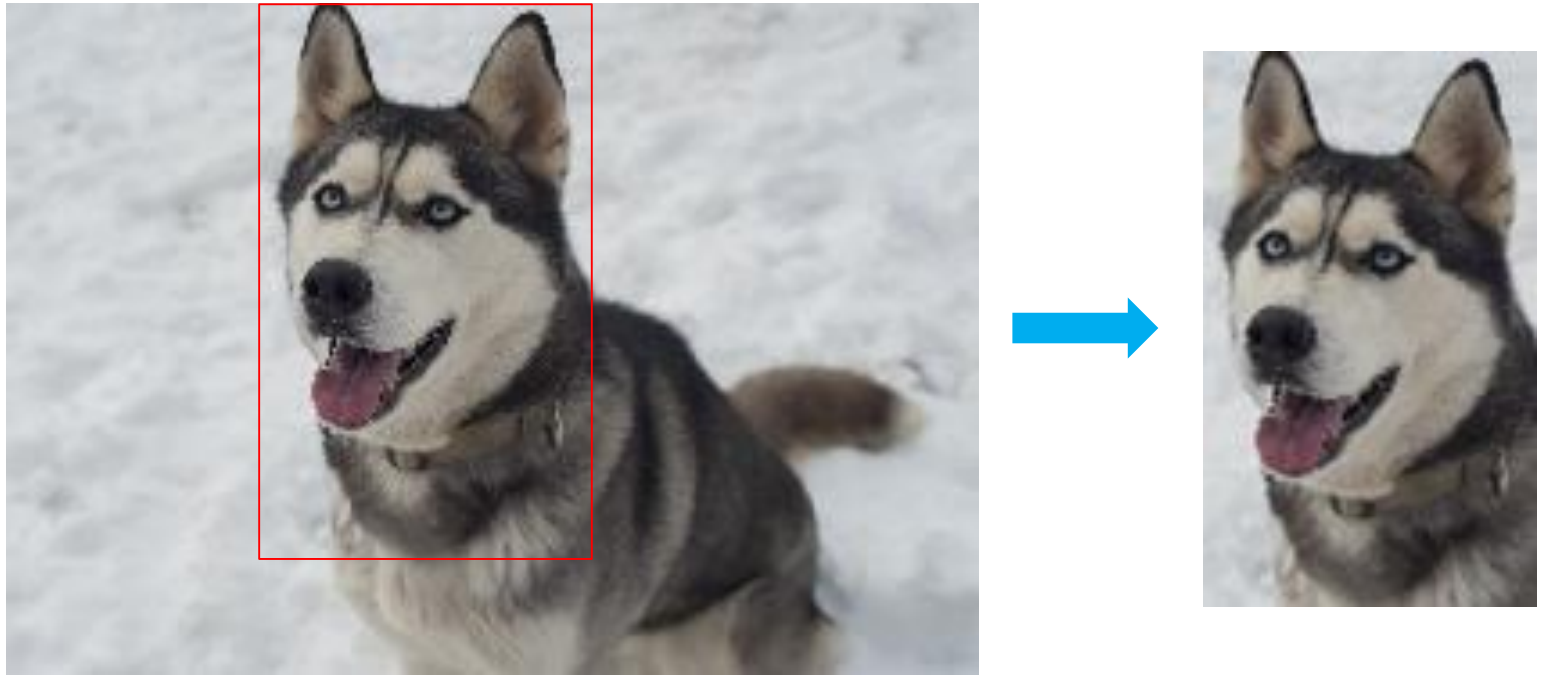
5. Other creative techniques

Random mix/combinations of :

- translation (what about a pure ConvNet?)
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Data Augmentation

- **Training**: Add random noise
- **Testing**: Marginalize over the noise



Data Augmentation

Batch normalization, Model ensembles

Data Augmentation

greatlearning

- **Image augmentation** artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips
- Very easy to implement, give good results especially on small datasets
- Easily fits into framework of noise / marginalization