Introduction to Deep Learning (I2DL)

Tutorial 8: Augmentation, Regularization & Autoencoders

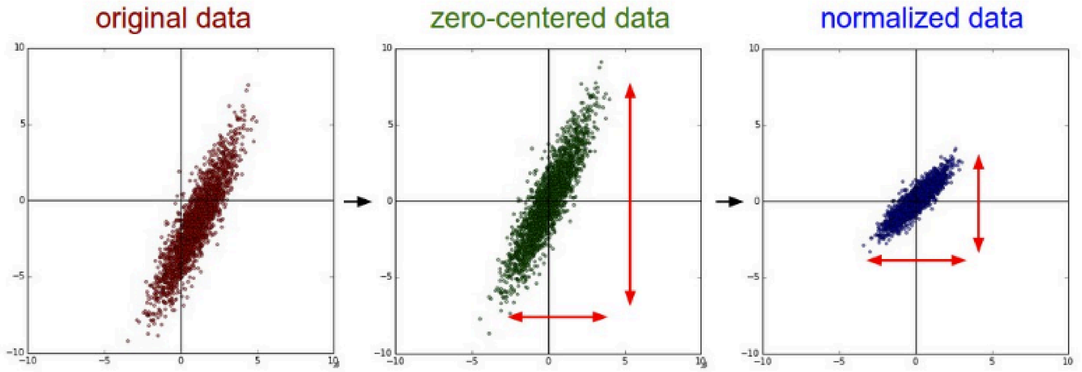
Today's Outline

- **Part 1: Data Augmentation & Regularization**
  - augmentation, weight regularization, dropout, batch normalization
- **Part 2: Exercise 8**
  - transfer learning and autoencoders

Part 1: Data Augmentation & Regularization

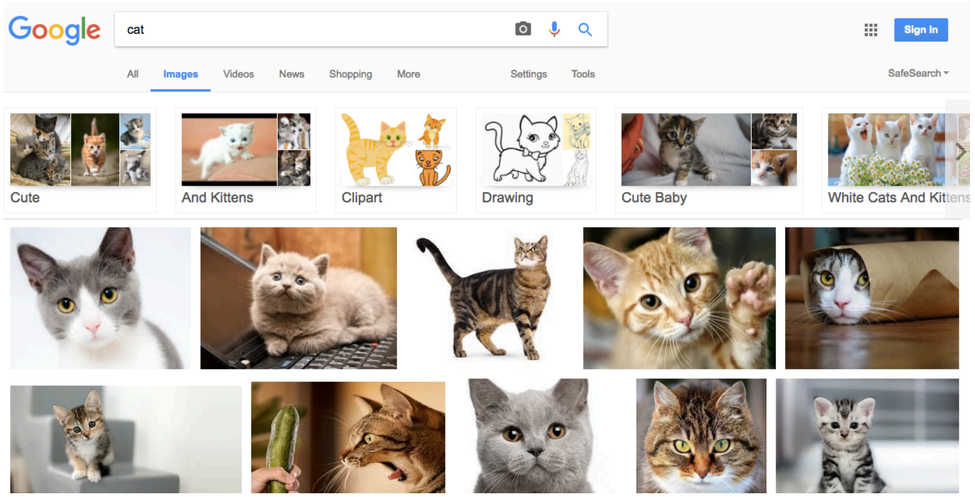
Data Pre-Processing

- Data augmentation is one part of **data pre-processing**.
- **A typical pre-processing pipeline has two parts:**
  - **Essential** — transform the raw inputs into tensors fed to the network
  - **Optional augmentation** — expands the variety of the training data
- Common techniques include **zero-centering** and **normalization**, shown below.



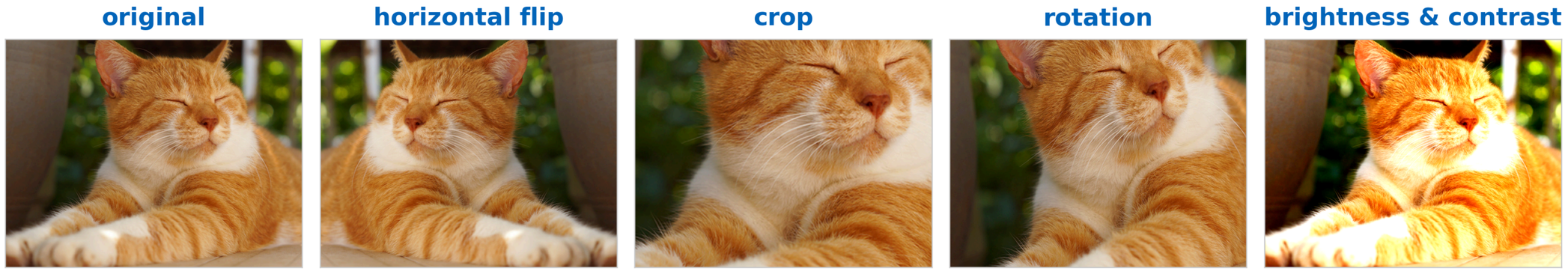
Data Augmentation: Learning Invariances

- Take an **image classification** network as an example.
- We want it to recognize a cat under any **pose, appearance, and illumination**.
- **This needs training data with large variety:**
  - collect more images, or
  - **data augmentation** — synthesize new training images for more variety



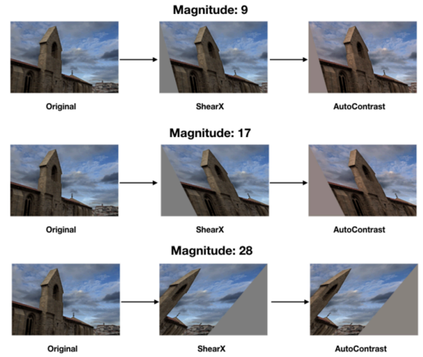
Common Augmentations

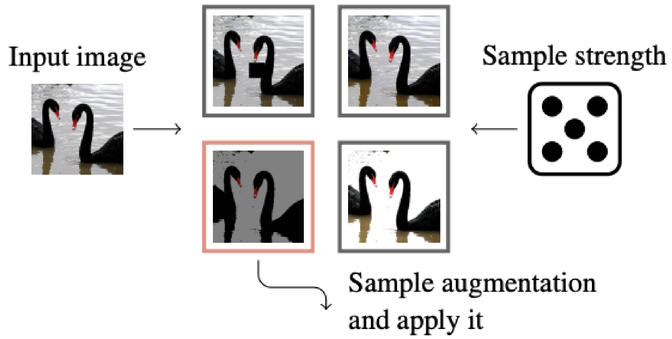
- **Horizontal flip** — a mirrored cat is still a cat.
- Or **crop**, **rotate** slightly, change **brightness and contrast**.
- Each variant adds **variety** beyond the original data.



Auto Augmentation

- **Random combination** of augmentations and random parameters
- Apply **at runtime** for more flexibility, vs **offline** for dataloader efficiency (trade time with space)





Augmentation in Practice

- **Augmentation should not "pollute" the training data:**
  - ✗ make the image unrecognizable
  - ✗ make it fall out of the test-time distribution
  - ✗ make it mismatch its label
- Augmentation **makes training harder**, so the model generalizes better.
- When comparing two networks, use the **same** augmentation.
- Treat it as a **hyperparameter to tune**.

Regularization: The Idea

- Regularization **makes training harder on purpose**, so the model does not overfit and generalizes better.
- Expected outcome: **validation error decreases**, while the **training loss may rise**.
- **Data augmentation** is one such technique, more follow in the next slides.

L2 Regularization

$$L = L_{\text{data}} + \frac{\lambda}{2} \|W\|_2^2$$

- **Penalizes large weights** — a single huge weight gives a high loss.
- Encourages spreading the signal **across many neurons** instead of relying on one.

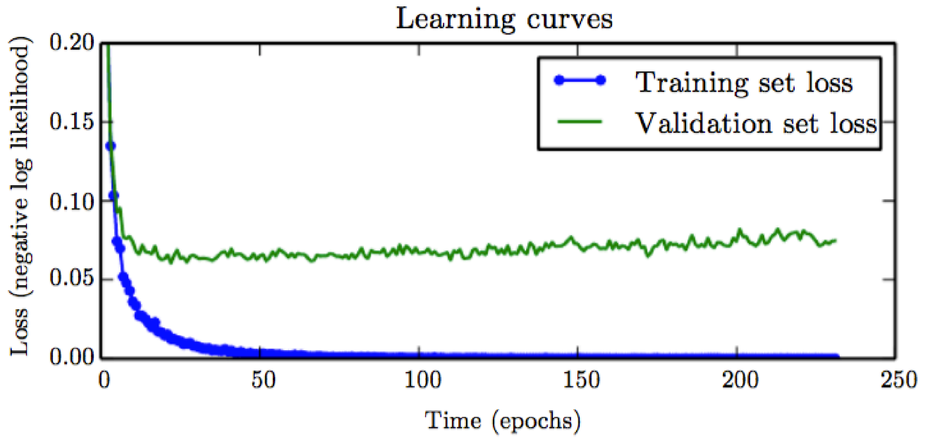
Weight Decay

$$\theta \leftarrow (1 - \eta\lambda)\,\theta - \eta\,\nabla L(\theta)$$

- **Weight decay** shrinks the weights a little on every update, the $(1 - \eta\lambda)$ factor.
- It equals L2 regularization for plain gradient descent, **but not for Adam**.
- **Reference:** a video explaining weight decay & regularization mathematically

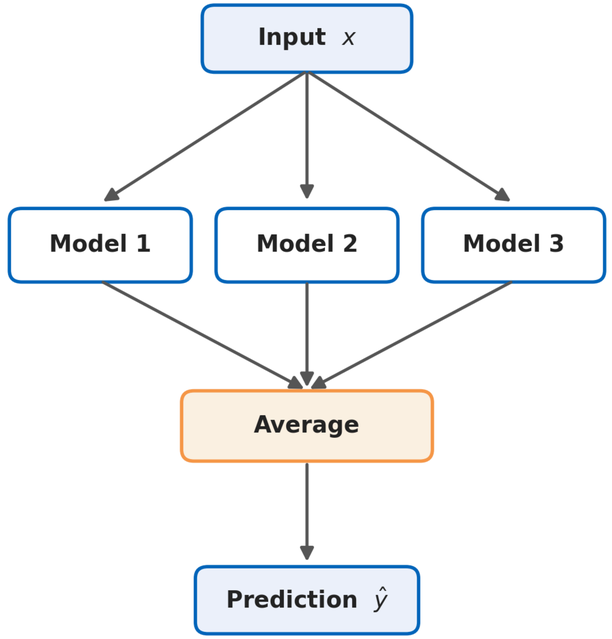
Early Stopping

- Watch the **validation curve** while the training loss keeps falling.
- **Stop** when validation error starts to rise, the onset of overfitting.
- The simplest form of regularization, and almost free.



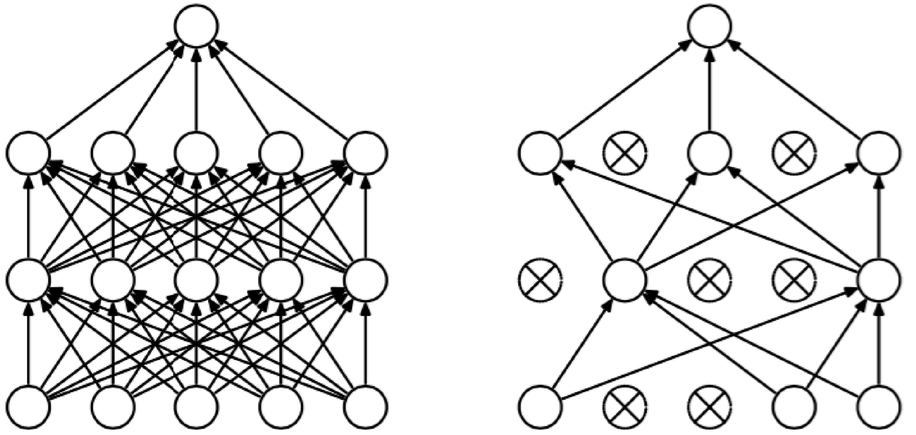
Bagging & Ensemble Methods

- **Train several models and average their predictions.**
  - different data subsets, different initializations, or different objectives
- If their errors are **uncorrelated**, the combined error drops with the ensemble size.
- Reliable, but it **multiplies the compute**.



Dropout

- On each forward pass, **randomly disable a fraction of the neurons** (the **drop probability** p, default a half).
- Different neurons drop each step, so over training **every neuron still learns**.
- Acts like an **ensemble of many tiny networks**, without training them separately.
- **Intuition:** lower capacity makes training harder, forcing **redundant representations**.



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

Dropout: Training vs. Testing

Scale at Test Time

- **Train:** drop with probability p, keep the rest as-is.
- **Test:** scale the layer output by $1 - p$:

$$a_{\text{test}} = (1 - p)\, a$$

- *original dropout*

Scale at Training Time

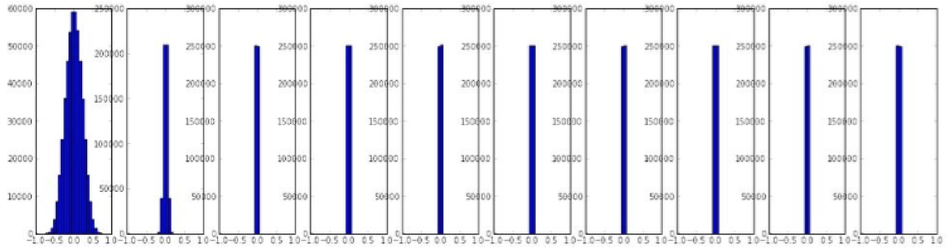
- **Train:** drop with probability p, scale the kept outputs up:

$$a_{\text{train}} = \frac{a}{1 - p}$$

- **Test:** use the layer output as-is, no scaling.
- **inverted dropout — what PyTorch does**

Batch Normalization: Internal Covariate Shift

- As parameters change, the **distribution of each layer's activations drifts**, the internal covariate shift.
- We want activations to stay in a good range, **not saturate and not die out**.
- The **wish**: let the network **actively shift the distribution** of each layer's activations.



Batch Normalization: The Idea

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

- (k) is the **feature dimension**; the mean and variance are taken **over the mini-batch**.

$$y^{(k)} = \gamma^{(k)}\, \hat{x}^{(k)} + \beta^{(k)}$$

- γ, β are **learnable** — they rescale and shift, restoring the network's representation power.

The Batch Norm Layer

- Insert the batch-norm layer **after a fully-connected or convolutional layer, before the non-linearity**.

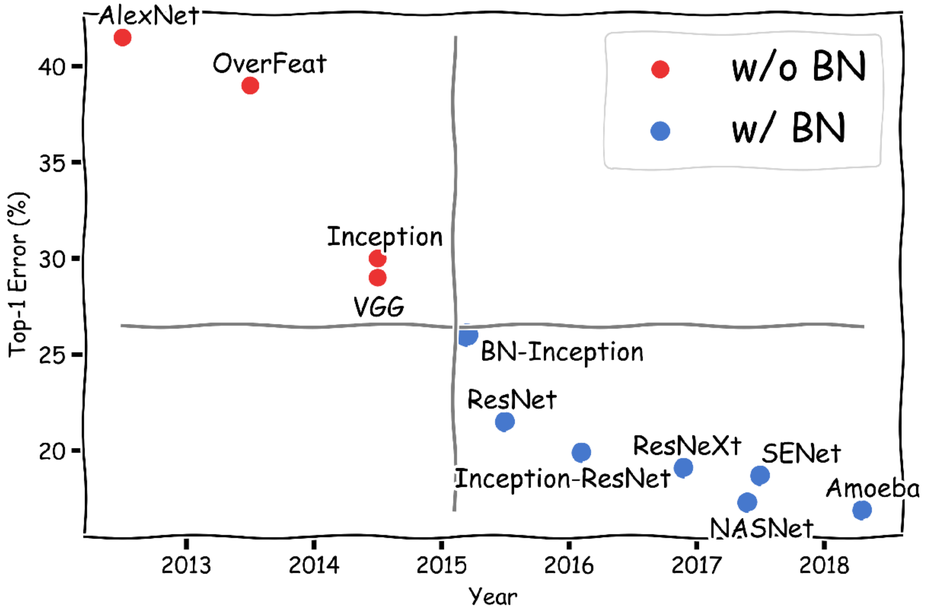
- **Training:** mean and variance from the current mini-batch, plus a **running average**:

$$\hat{\mu} \leftarrow (1-\rho)\,\hat{\mu} + \rho\,\mu_B \qquad \hat{\sigma}^2 \leftarrow (1-\rho)\,\hat{\sigma}^2 + \rho\,\sigma_B^2$$

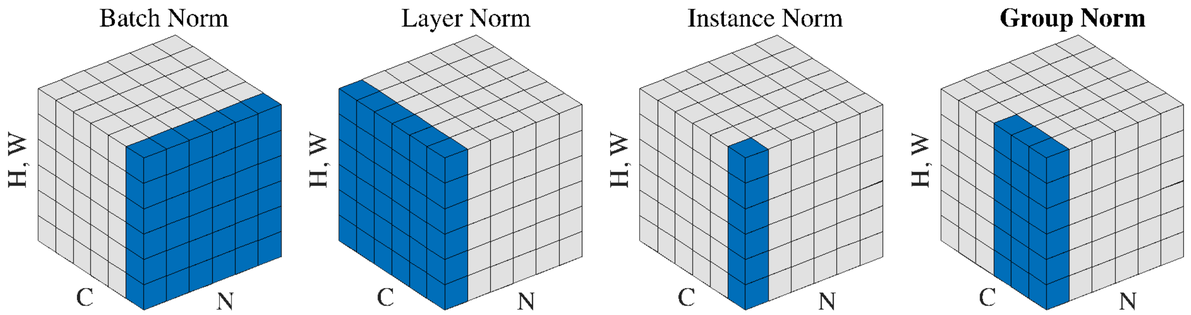
- **Testing:** use the running average, so single-sample predictions are **stable and deterministic**, independent of the other samples in the batch.

Batch Normalization: Why It Helps

- **Very deep networks train more easily**, with more stable gradients.
- A **much wider range of hyperparameters and learning rates** now works.
- A genuine **milestone** that made training deep nets routine.
- **Yet** batch norm **depends on the batch**, so it wants large mini-batches.



Normalizations on Images



- **Batch Norm:** each channel shares a distribution across samples and space, normalize across the **mini-batch and width, height**.
- **Layer Norm:** normalize across **all channels and width, height**.
- **Instance Norm:** normalize across **width, height only**.
- **Group Norm:** normalize across a **group of channels and width, height**.

Part 2: Exercise 8 — Transfer Learning & Autoencoders

Transfer Learning: Motivation

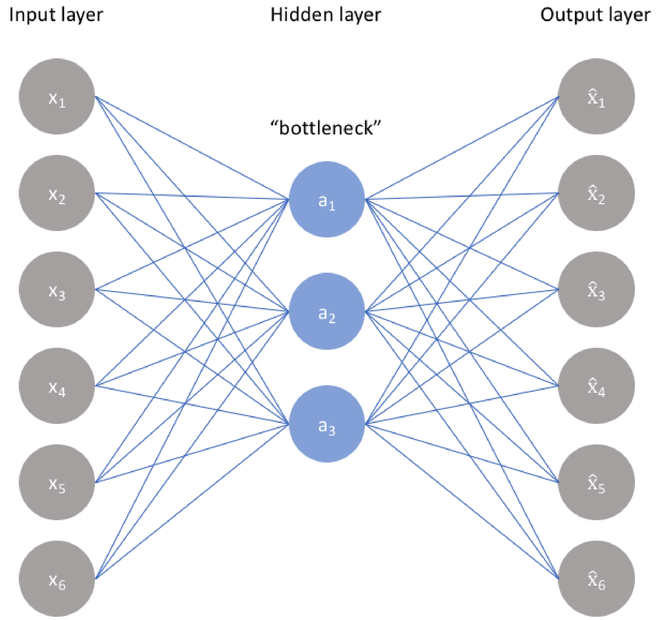
- Training a deep network needs **a lot of data**, and collecting it is expensive or impossible.
- **Idea:** many tasks are related, so **reuse a network pre-trained on one task** for another.

Transfer Learning: How It Works

- Keep the pre-trained **feature extractor**, replace the **classifier head** for your task.
- Train the new head, and optionally **freeze** the early layers or use a **lower learning rate**.
- A small target dataset can borrow the knowledge from a large one.

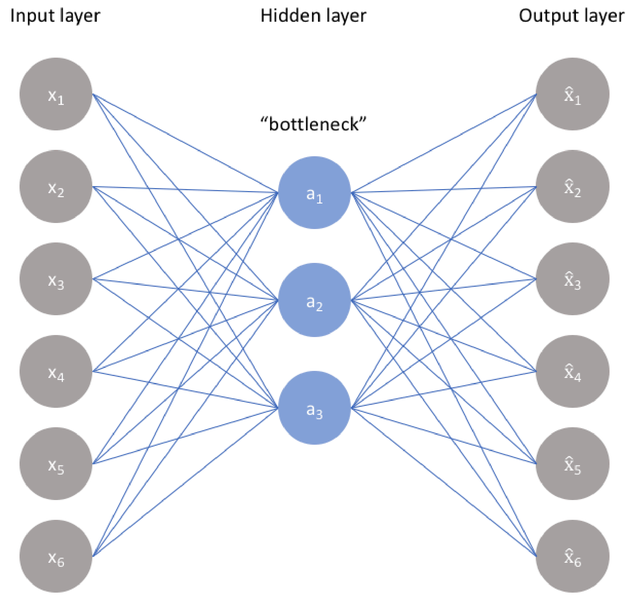
The Autoencoder

- **Reconstruct the input** through a lower-dimensional bottleneck, an encoder then a decoder.
- Loss is **L1 or L2 per pixel**, and crucially it needs **no labels**.
- So a **large amount of training data** is easy to gather, no labels needed.



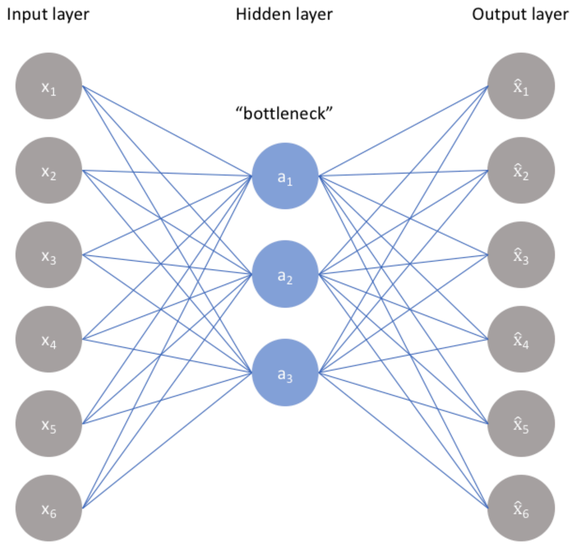
Transfer Using an Autoencoder

- **Step 1:** train an autoencoder on a large, possibly **unlabeled** dataset.
- **Step 2:** take the pre-trained **encoder** as the front of a classifier for your target task.



Exercise 8: Your Task

- **mnist:** 60k images, but only **300 are labeled**.
- Pretrain an autoencoder, transfer the encoder, and beat the baseline classifier, target **55% accuracy**.
- An **optional notebook** lets you implement the dropout and batch normalization from Part 1.
- Still **linear layers** only, no convolutions yet, that is next week.



Good Luck & See You Next Time!