Ask questions at menti.com - Code: 2515 1387

Data Augmentation

Lecture #8 - IN3310/4310



Marius Aasan, Martine Tan - 16.03.2023

Ask questions at menti.com - Code: 2515 1387

Introduction

Who are these guys anyway?

- **Marius Aasan**
  - PhD Image Processing - DSB@IFI + SFI Visual Intelligence
  - Fields: Self Supervised Learning, Vision Transformers and Hierarchical GNNs
- **Martine Tan**
  - PhD Image Processing - DSB@IFI
  - Fields: Self Supervised Learning, Probabilistic Learning, Hierarchical Modelling
- Lecturing two units in IN3310/4310.







Introduction

After this lecture you will be able to:

1. Describe distributional shifts, out-of-distribution samples, and outliers.
2. Apply methods for improving generalisation and domain adaptation in computer vision models.
3. Describe and compare image augmentations and their parametrisation.
4. Apply, combine, and implement new image augmentations using PyTorch and Pillow.
5. Analyse and compare data augmentations in other domains, such as natural language.
6. Formulate methods for applying fixed neural networks on varying input sizes.
7. Describe standard weight initialisation and standardisation strategies.
8. Compare teacher-student learning to standard neural network optimisation.
9. Justify contrastive learning approaches and analyse contrastive loss functions.

The Cat Lady's Request

Motivational Example

- A cat lover tasks you with training a network for classification, and provides you with a labeled dataset of images of (their) cats.



Image credit: Arielle Lewis Studios

The Cat Lady's Request

Motivational Example

- A cat lover tasks you with training a network for classification, and provides you with a labeled dataset of images of (their) cats.
- You notice that all the images are closeup indoor images of the (reluctant) faces of cats taken by the owner using an SLR.



The Cat Lady's Request

Motivational Example

- A cat lover tasks you with training a network for classification, and provides you with a labeled dataset of images of (their) cats.
- You notice that all the images are closeup indoor images of the (reluctant) faces of cats taken by the owner using an SLR.
- They want to use the classifier to detect cats in their backyard with a low resolution infrared surveillance camera.



The Cat Lady's Request

Motivational Example

- A cat lover tasks you with training a network for classification, and provides you with a labeled dataset of images of (their) cats.
- You notice that all the images are closeup indoor images of the (reluctant) faces of cats taken by the owner using an SLR.
- They want to use the classifier to detect cats in their backyard with a low resolution infrared surveillance camera.
- **Do you see any problems with this?**

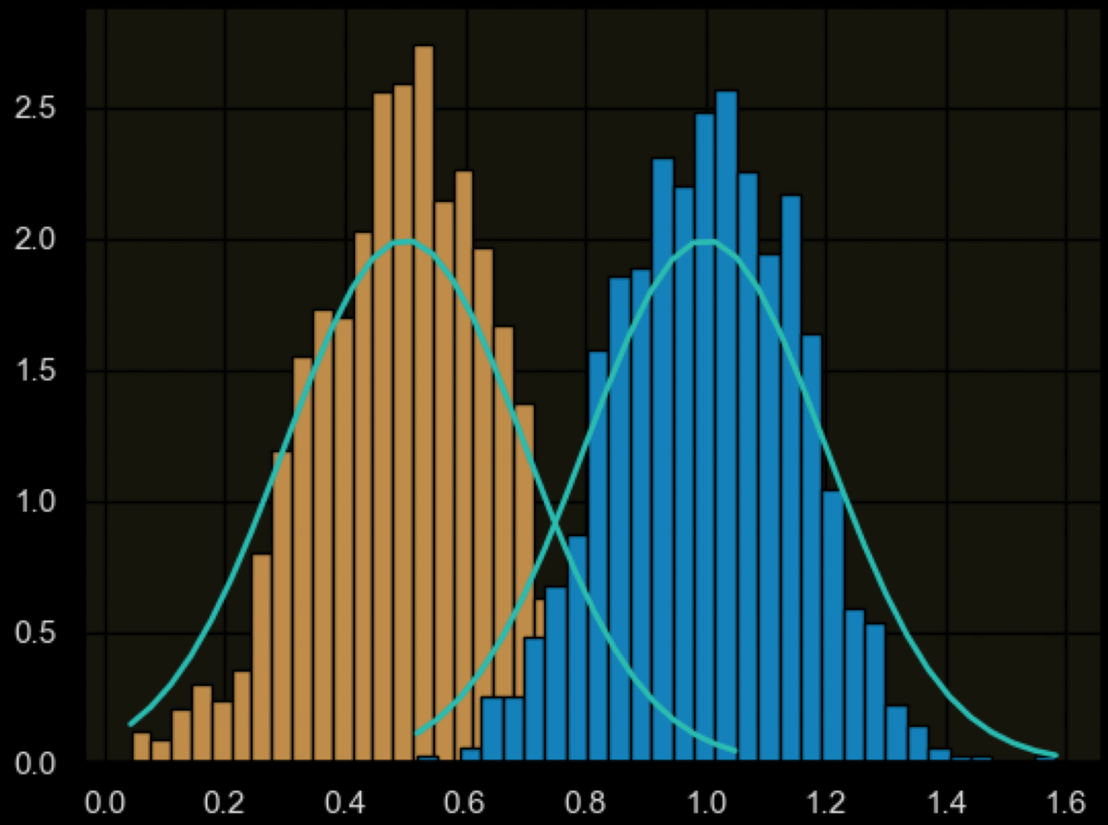


Problems with Example

Differences in distributions

The training images and backyard images are from ***different distributions.***

- Different ***resolution*** (high res. SLR vs. low res. surveillance)
- Different ***scale, perspective*** and ***pose*** (closeup of face vs. full body)
- Different ***lighting*** and ***colours*** (indoors colour vs. outdoor IR/BW)
- Different ***label distribution*** (other cats in image, multiple cats in image)



Terminology

Generalisation, Domain Adaptation, and Distribution

- ***Generalisation (model property):***
  $f(\mathscr{D}_{\text{train}}) \approx f(\mathscr{D}_{\text{target}})$ for $\mathscr{D}_{\text{train}} \sim \mathscr{D}_{\text{target}}$*
- ***Domain adaptation (model property):***
  $f(\mathscr{D}_{\text{train}}) \approx f(\mathscr{D}_{\text{target}})$ for $\mathscr{D}_{\text{train}} \nsim \mathscr{D}_{\text{target}}$**
- ***Out-of-distribution (data property):***
  Samples significantly different from the data our model was trained on.
- ***Outliers (data property)*:** Samples from the tails of training or target distributions.
- ***Distribution shift (data property):*** Samples with different statistical properties of the data our model was trained on.

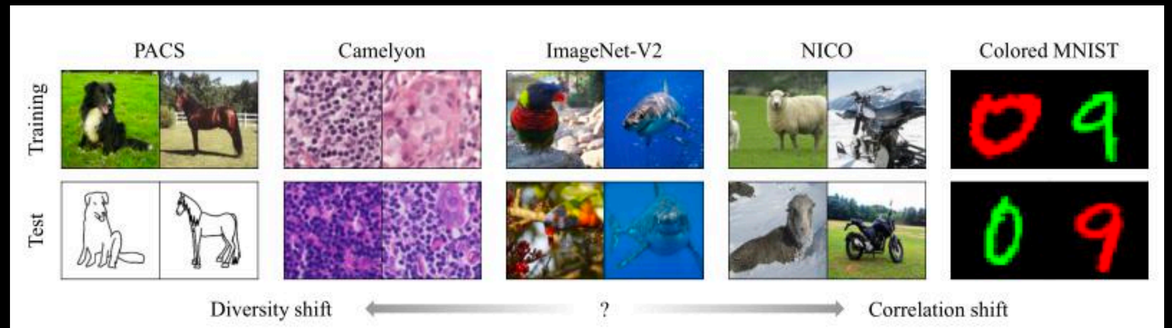


Figure 1: Examples of image classification datasets demonstrating different kinds of distribution shifts. While it is clear that the datasets at both ends exhibit apparent distribution shifts, in the middle, it is hard to distinguish the differences in distribution between the training dataset (ImageNet [21]) and the test dataset (ImageNet-V2 [58]), which represent a large body of realistic OoD datasets.

* $\mathscr{D}_{\text{train}} \sim \mathscr{D}_{\text{target}}$:= training data has similar distribution to target / test / inference data.

** $\mathscr{D}_{\text{train}} \nsim \mathscr{D}_{\text{target}}$:= training data has different distribution to target / test / inference data.

Rule #1:

Preprocess your input data!

- MinMax:

$$z = \frac{x - \min(x)}{\max(x) - \min(x)} \rightarrow z_i \in [0,1]$$

- MaxAbs:

$$z = \frac{x}{\max(|x|)} \rightarrow z_i \in [-1,1]$$

- Standardisation:

$$\frac{x - \mu_x}{\sigma_x} \rightarrow \mathbb{E}[z] = 0, \ \ \mathrm{Var}[z] = 1.$$

Why?

- Control **variance.**
- Improve **convergence**.
- Reduce numerical **instability**.
- Reduce sensitivity to **outliers.**

Issues with Generalisation

Isn’t inference data always different from training data?

- We cannot expect training data to match data at inference…
- …and we cannot expect our models to match training error in deployment!

Recall:

- training / validation / test
- bias-variance tradeoff

The Deep Learning Solution™:

- *more training and data*, let $|\mathscr{D}| \to \infty$!

Train longer, generalize better: closing the generalization gap in large batch training of neural networks

Elad Hoffer,* **Itay Hubara**,* **Daniel Soudry**
Technion - Israel Institute of Technology, Haifa, Israel
{elad.hoffer, itayhubara, daniel.soudry}@gmail.com

Show of hands

Which of these are problematic for our approach?

A. Takes somewhat longer ($t \rightarrow \infty$) to train our models.

B. Risk of creating a black hole as a result of infinite informational capacity.

C. Gathering infinite samples is time consuming, and we are lazy.



Show of hands

Which of these are problematic for our approach?

A. *Takes somewhat longer ($t \to \infty$) to train our models.*

Environmentally unfriendly.

B. *Risk of creating a black hole as a result of infinite informational capacity.*

C. *Gathering infinite samples is time consuming, and we are lazy.*

Show of hands

Which of these are problematic for our approach?

A. *Takes somewhat longer ($t \to \infty$) to train our models.*

Environmentally unfriendly.

B. *Risk of creating a black hole as a result of infinite informational capacity.*

Only a problem at $t = \infty$.

C. *Gathering infinite samples is time consuming, and we are lazy.*

Show of hands

Which of these are problematic for our approach?

A. *Takes somewhat longer ($t \to \infty$) to train our models.*

Environmentally unfriendly.

B. *Risk of creating a black hole as a result of infinite informational capacity.*

Only a problem at $t = \infty$.

C. *Gathering infinite samples is time consuming, and we are lazy.*

No problem, lazy = smart.

Data Augmentation

Artificially Extending our Dataset

How to get our hands on more data?

- We want to generate samples similar to our existing data.
- We want the variation to be indicative of natural variations of our data.

We use image transformations to *augment* our dataset.

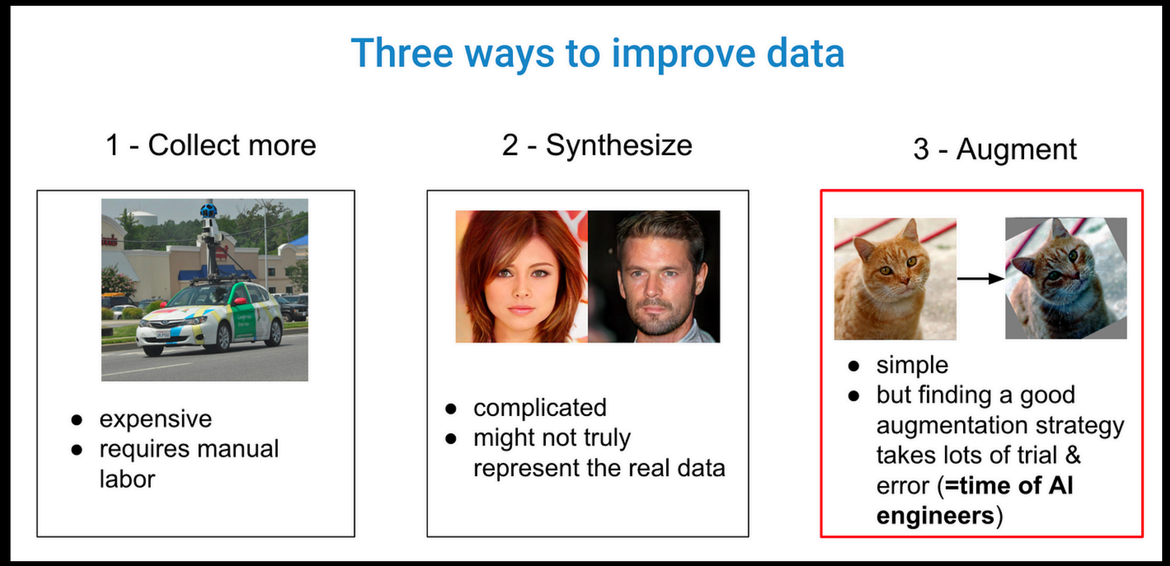


Image credit: Data Talks - Youtube

Geometric of Photometric?

Two main categories

Geometric image transformations:

- Changes to geometric properties.
- Size, shape, position, orientation, etc.

Photometric transformations:

- Changes photoreceptive properties.
- Brightness, contrast, colour, noise, etc.

Other filters:

- Can be interpreted as either, or both.
- Blur, edge filters (Sobel, Canny), etc.



Image credit: Rob Zwiercan



Geometric Transforms

Vertical / horizontal flip

Several common transformations are *geometric* in nature.

Simple example: flipping images in `torch.tensor` format.

```
def vflip(img):
  # Input dim: B,C,H,W
  # Dimension -2 := H
  return img.flip(dim=-2)

def hflip(img):
  # Input dim: B,C,H,W
  # Dimension -1 := W
  return img.flip(dim=-1)
```

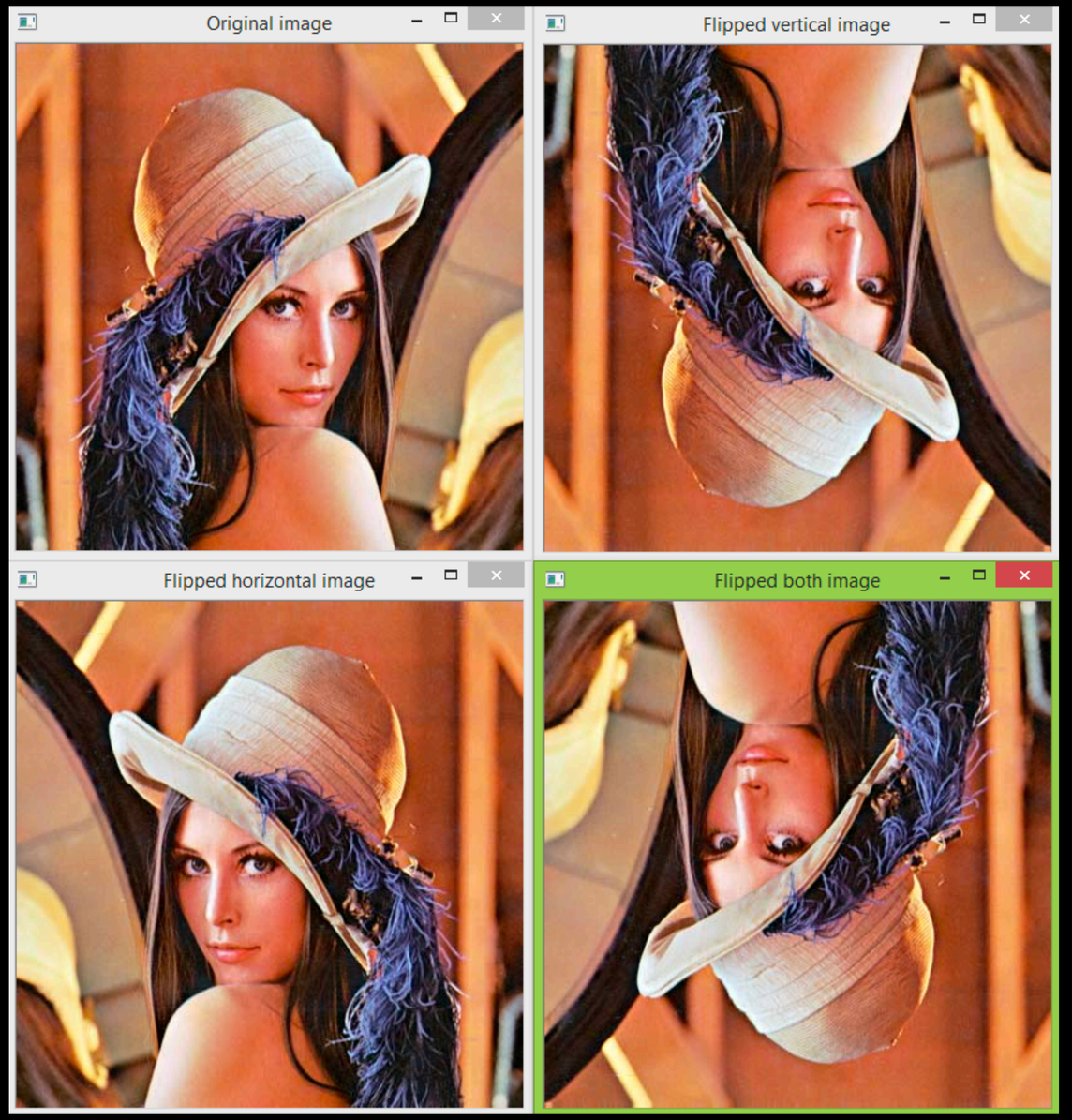


Image credit: OpenCV

Geometric Transforms

Image rotation

Rotation is another commonly applied geometric transform. In matrix form, coordinates are transformed by.

$$\begin{bmatrix} x_{\text{out}} \\ y_{\text{out}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_{\text{in}} \\ y_{\text{in}} \end{bmatrix}$$

PyTorch implementation:

```
import torchvision.transforms as T

def rotation(img, angle):
  # Input dim: B,C,H,W
  return T.functional.rotate(img, angle)
```



Image credit: PyTorch - torchvision documentation

Geometric Transforms

Affine transformations

More power, but more computationally expensive!

A generalisation of rotation and flips, which includes shifting, scaling, and shearing.

$$\begin{bmatrix} x_{\text{out}} \\ y_{\text{out}} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_{\text{in}} \\ y_{\text{in}} \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

PyTorch implementation:

```python
import torchvision.transforms as T

def affine(img, angle, shift, scale, shear):
  # Input dim: B,C,H,W
  return T.functional.affine(
    img, angle, shift, scale, shear
  )
```

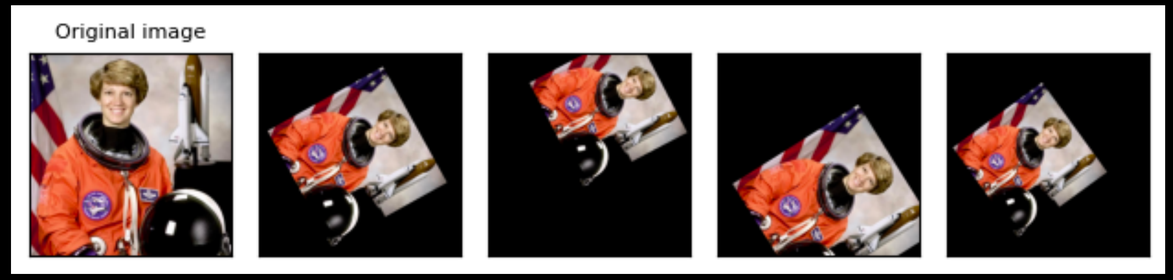


Image credit: PyTorch - torchvision documentation

Photometric Transforms

Color Jitter

Standard method:

- Random perturbation of either channel in HSV/HSL space.
- Note: Hue channels are angular!

```
def colour_jitter(img, intensity):
  hsvimg = rgb2hsv(img)
  eps = torch.randn(3,1,1)
  hsvimg = hsvimg + eps * intensity
  hsvimg[:,0].remainder_(1)
  hsvimg.clip_(0, 1)
  return hsv2rgb(hsvimg)
```



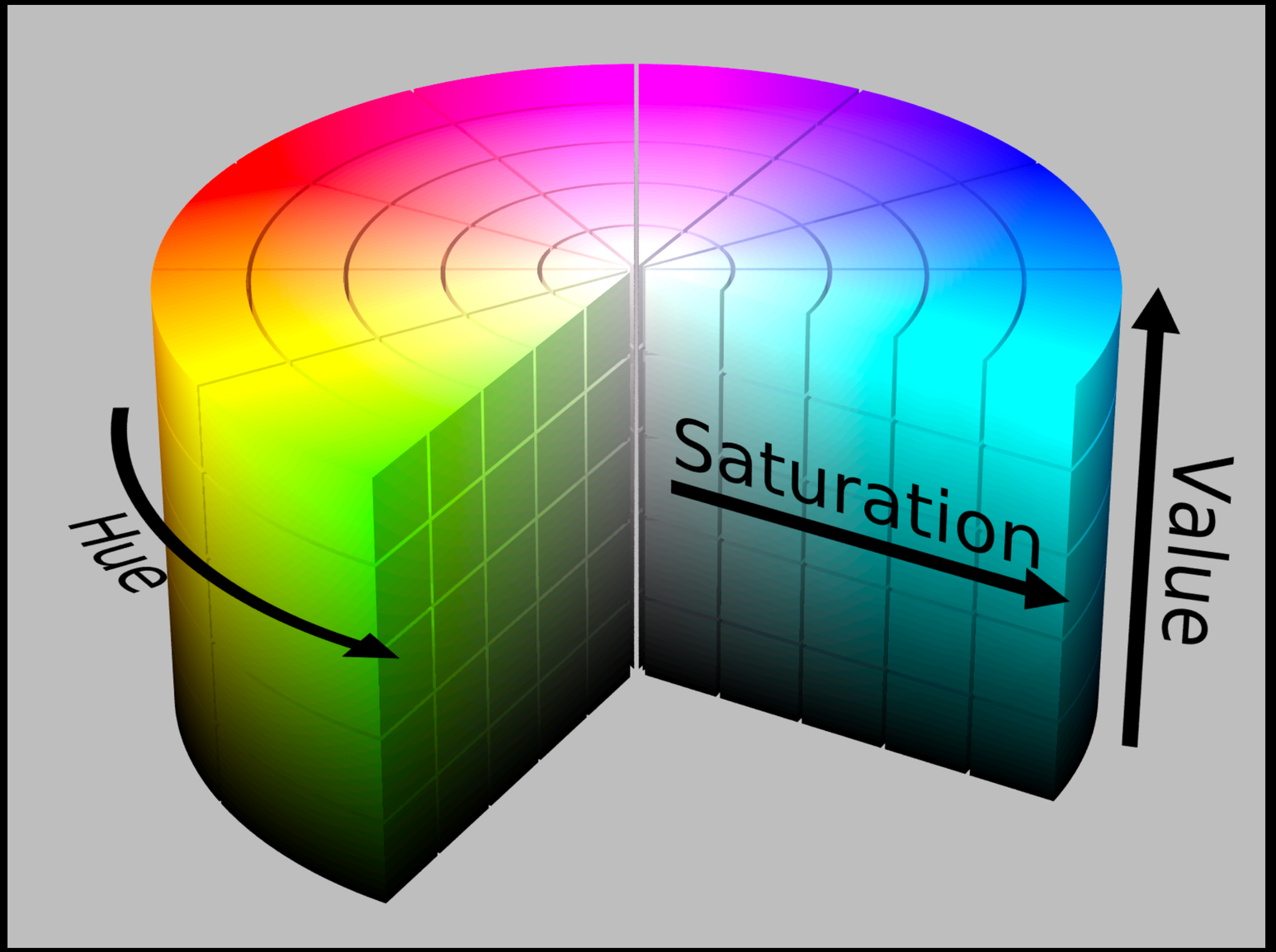
Image credit: Apache MXNet

Notes:
- The implementation is applied to a tensor of shape: `B,C,H,W`
- In-place version of functions in PyTorch use convention `x.foo_(arg)`
- We first calculate the remainder / modulo 1 of the hue channel.
- We then clip the image tensor to be in [0, 1] before returning.

In practice, we use `torchvision.transforms.ColorJitter`, which works on both `PIL` images and `torch.tensors`

HSV

Angular Color



Photometric Transforms

Solarization

PROTIP: Add stochasticity with probability parameter!

Solarized image: An image where colours above a certain threshold are inverted.

Implementation using PIL library:

```
def solarize(img, prob):
  if random.random() < prob:
    img = PIL.ImageOps.solarize(img)
  return img
```



Image credit: Mohammed Sameeruddin

Notes:
- The Pillow (PIL) library includes several image transforms.
- PIL uses CPU, and PyTorch data loaders can execute in parallel.

In practice, we can use `torchvision.transforms.RandSolarize`

Blur Transformation

Blurring the lines

A blurred image is a result of applying a ***low-pass-filter,*** removing high frequency components out from the image.

Very common: Using a Gaussian kernel

Implementation using PIL library:

```
def gaussblur(img, prob, r_min, r_max):
  if random.random() < prob:
    radius = random.uniform(
      r_min, r_max # Controls intensity
    )
    filter = PIL.ImageFilter.GaussianBlur(
      radius = radius
    )
    img = img.filter(filter)
  return img
```

Image credit: OpenCV





Notes:

- Again we use Pillow (PIL).
- For tensors, we can use a Gaussian convolution kernel:

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

Other Augmentations

The world is your oyster!

- *Geometric*: perspective transforms.
- *Photometric*: contrast, brightness, equalisation.
- *Other*: Additive noise, Sobel/Canny filters.

Popular choices:

- ***RandAugment*:** Random combination of 14 different photometric and geometric transforms.
- ***Mixup*:** Pairs all images in batch, blends them with a random convex combination.
- ***Cutmix***: Pairs all images in a batch, picks a random bounding box in the image, and copies slice from one image to another.

Mixup and cutmix also ***blends labels of the images!***

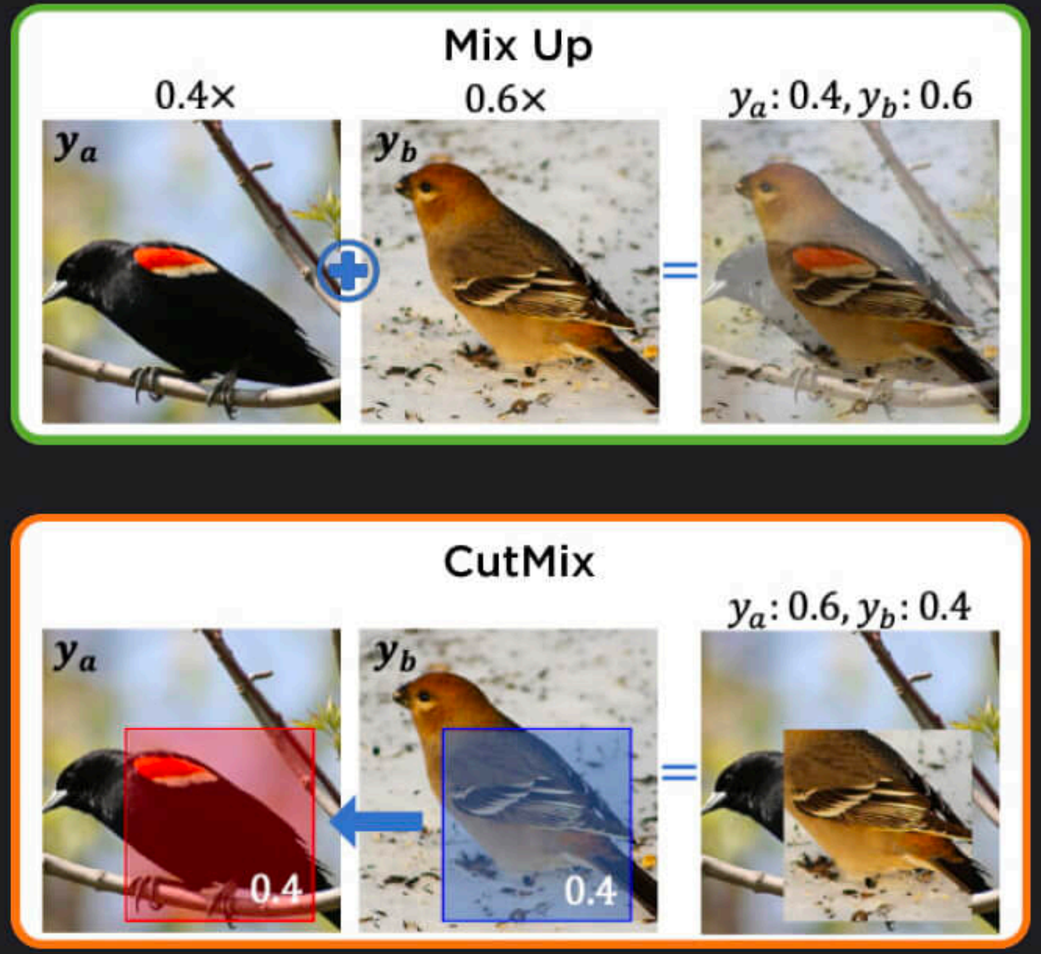
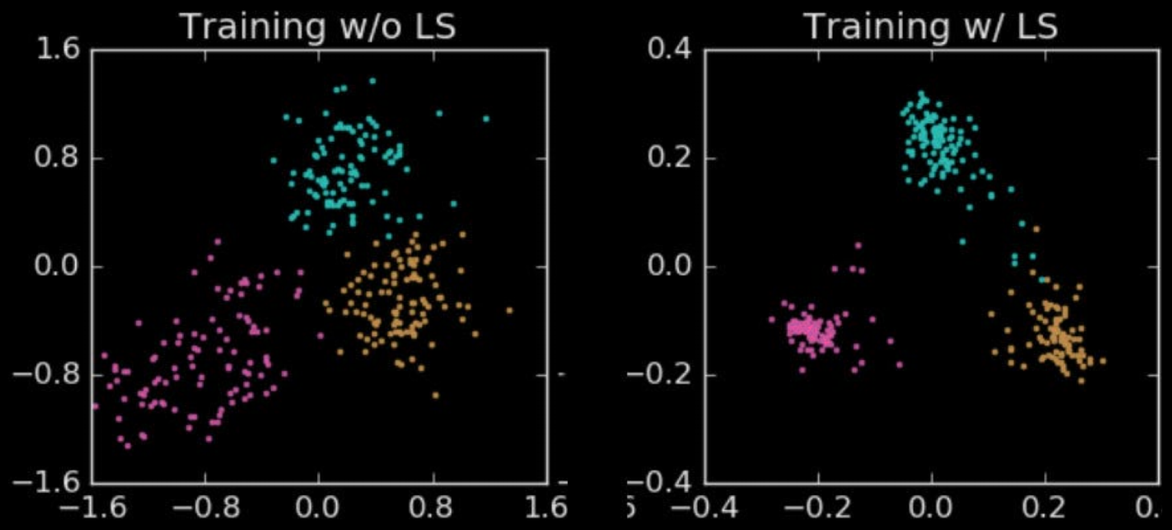


Image credit: Haung et al. 2020, https://arxiv.org/abs/2012.04846

Label Smoothing

Augmenting the labels

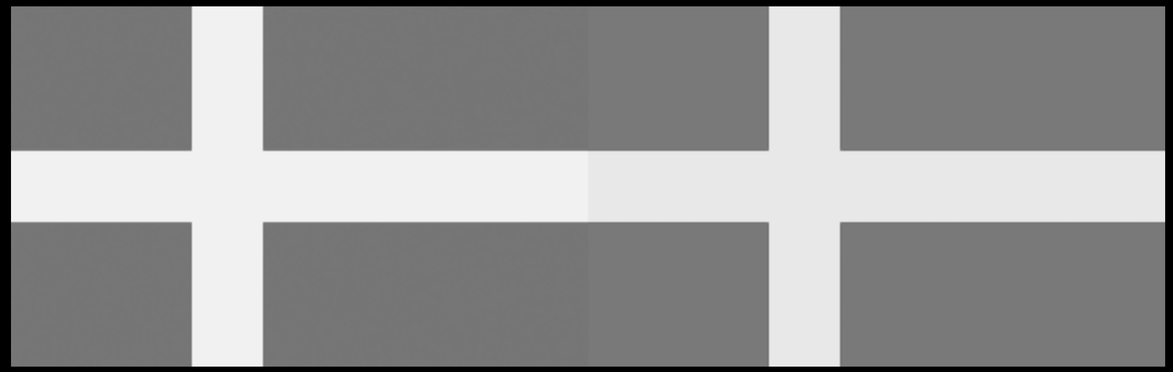
- Introduced by Szegedy et al. back in ancient times (2015).
- **Thought**: What happens if an image is mislabeled, or the label is ambiguous?
- Instead of optimising $p(y \mid x)$ directly, assume the label is correct with probability $1 - \epsilon$, and distribute the remaining probability mass on non-appearing labels.
- Effectively acts as a ***regularizer***.



Choices of Augmentation

Which augmentations should we use?

- Flags: Which flag is which?
  - Grayscale



Choices of Augmentation

Which augmentations should we use?

- Flags: Which flag is which?
  - Grayscale



Choices of Augmentation

Which augmentations should we use?

- Flags: Which flag is which?
  - Grayscale
  - Flip / rotation



Choices of Augmentation

Which augmentations should we use?

- Flags: Which flag is which?
  - Grayscale
  - Flip / rotation
- Pathology: different labs use different colouring of tissue.

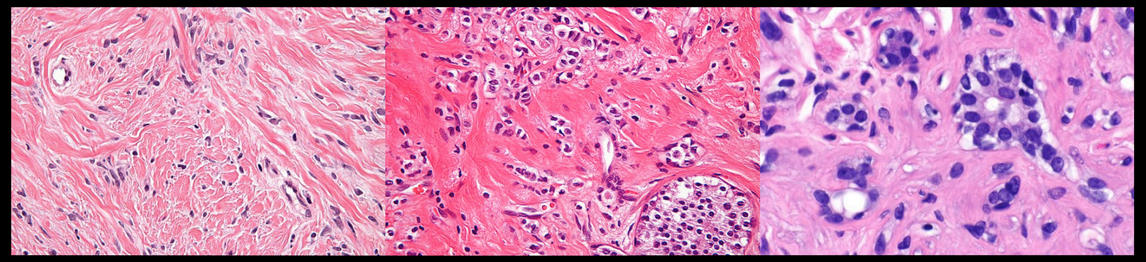


Image credit: Digital Pathology Association

Choices of Augmentation

Which augmentations should we use?

- Flags: Which flag is which?
  - Grayscale
  - Flip / rotation
- Pathology: different labs use different colouring of tissue.
- Augmenting validation set?

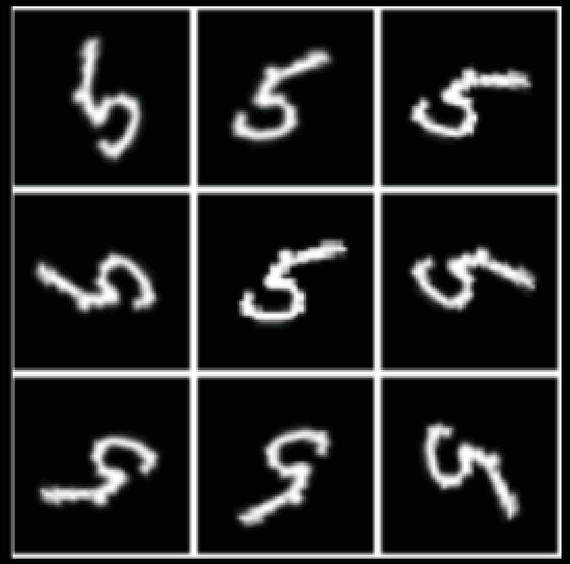


Image credit: MNIST - National Institute of Standards and Technology

Choices of Augmentation

Which augmentations should we use?

- Flags: Which flag is which?
  - Grayscale
  - Flip / rotation
- Pathology: different labs use different colouring of tissue.
- Augmenting validation set?

Conclusion: “No free lunch!”



Image credit: Harley Schwadron

Beyond Images

Augmentations in other domains

- **Images and audio**: Lots of existing work on filters and transformations.
  - *Examples from audio*: Pitch shift, resampling, modulation, equalisation…
- Can often be applied "*on the fly*", computationally effective.
- *Not trivial* to augment data in other domains!
- **Two general purpose solutions**:
  - Additive noise.
  - Sampling from generative models.

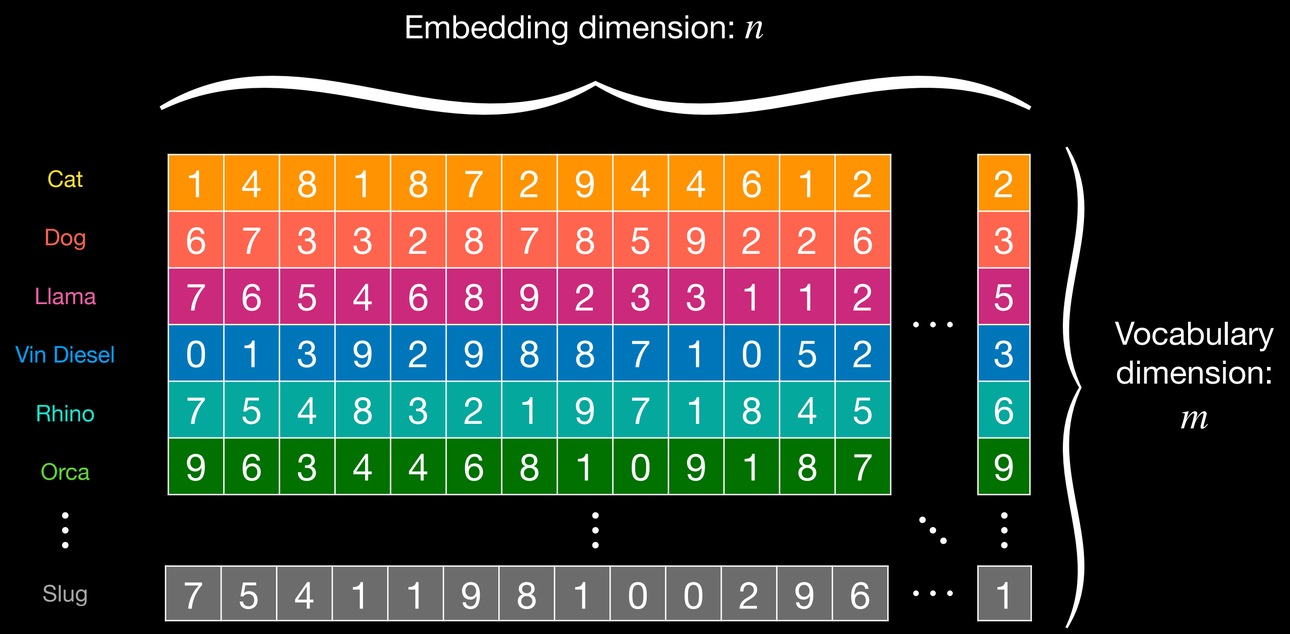
Problems with these approaches?

Case Study: NLP

Natural Language Processing

- Language is categorical, images are continuous.
- Ordering in language is grammar dependent, less clear cut order!
- Each word as one-hot encoded vectors:
  - Human vocabulary: ~30,000 words.
  - Then one hot encoded vectors consist of 29,999 zeros (very sparse, memory hungry)!
- To solve this, we learn an **embedding**, a continuous vector for each word.
- Makes parameter space **larger** (GPT-3: 175 billion)

Want: $n \ll m$.
GPT-3: $n = 384,\ m = 1.4 \times 10^7$



State of the Art (SOTA):

Previous: Recurrent Networks w. LSTM, GRU.

Current: Transformers with multihead self attention.

NLP Augmentations

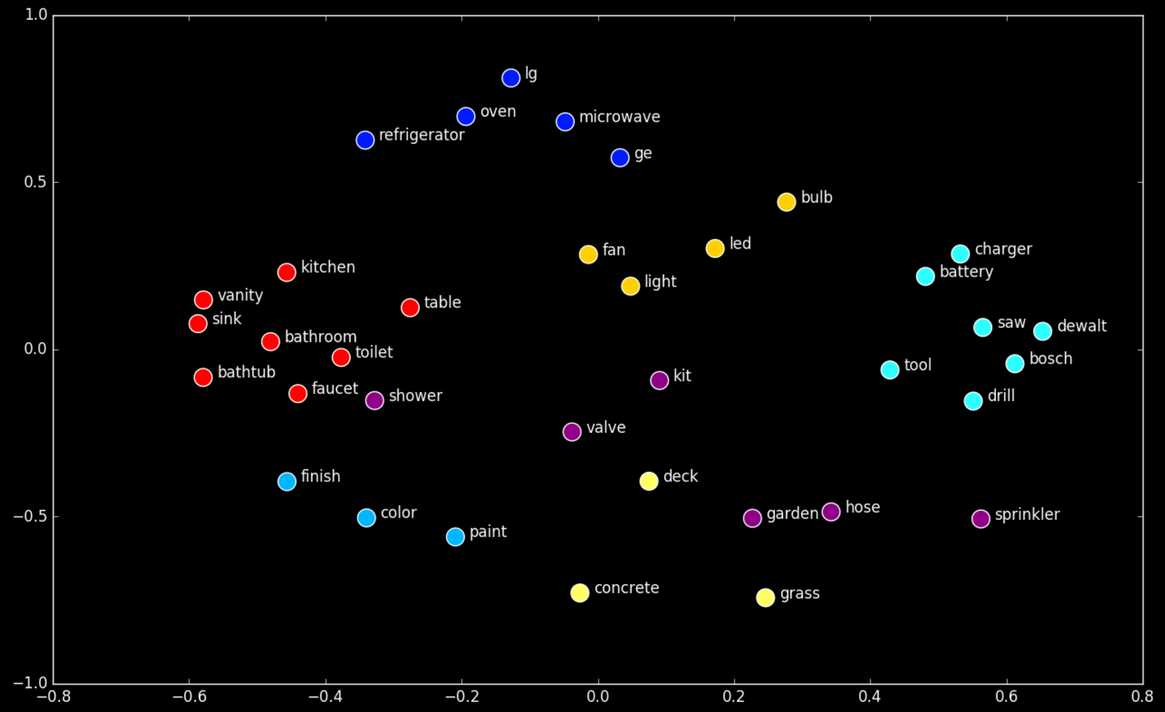
Careful Design to Preserve Grammar

***Common approach, masking*:**

- Replace random words with blank token.
- Encode masked sentence, use decoder to reconstruct full sentence.

***More advanced approaches*:**

- *Different levels*: Character, word, sentence.
- *$\mathcal{N}$-preserving*: Random insert/swap/delete/shuffle.
- *Synonyms*: Replace by KNN on embeddings.
- *Back translation*: Translate back and forth to another language.



Does Size Matter?

Well, sometimes…

How can we ensure our network is able to process different sized images?

- Simple approach: rescale images to match network input size.
- More advanced: convolution works on different image sizes.

$$d' = \frac{d - k - 2p}{s} + 1$$

d': output dim, d: input dim, k: kernel size, p: padding, s: stride.

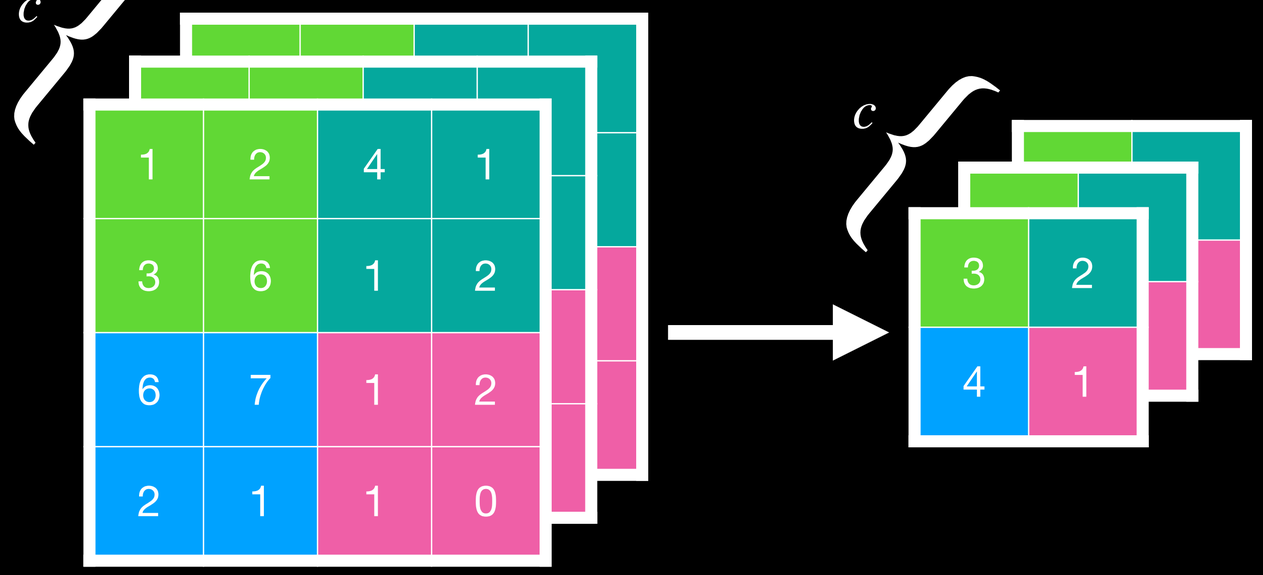
By ***downsampling*** features using ***global pooling layers***, we can ensure that our network can handle images of all sizes (above some minimum).

Pooling Operators in CNNs

Pooling with Kernel vs. Global Pooling

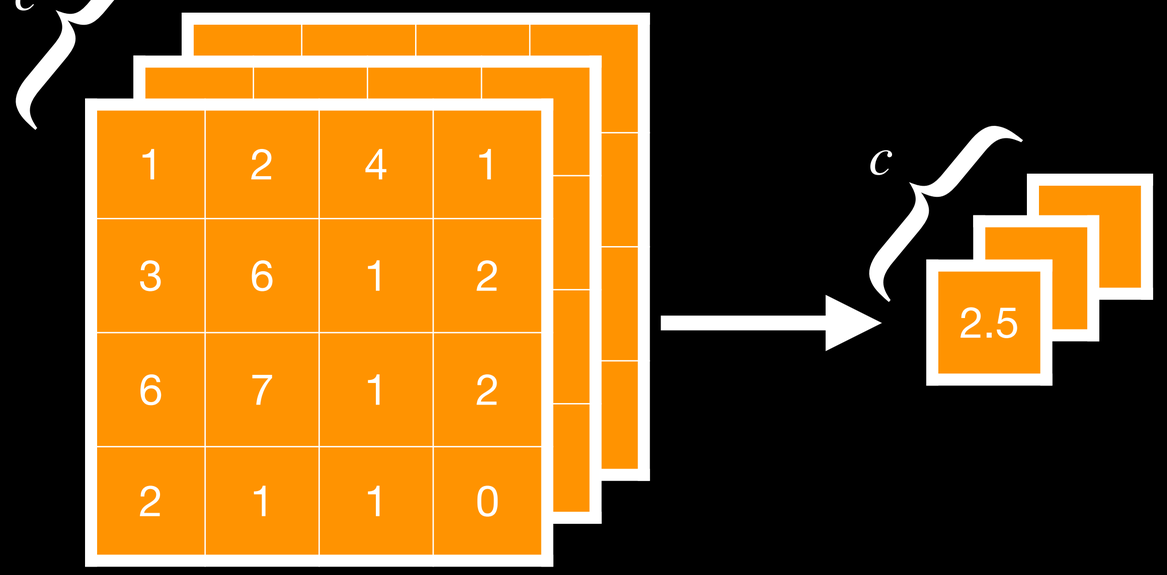
Pooling with $k = s = 2$

`nn.AvgPool2d(2,2)`



Global pooling

`nn.AdaptiveAvgPool2d((1,))`



Global pooling operators are generally applied before the last ***fully connected layers***.

Size Invariance

What about Vision Transformers?

Some notes on size invariance in ViTs:

- ViTs divide image into patches.
- Self-attention operators are invariant to number of patches / permutation.
- Permutation invariance implies no ordering, solved by positional embeddings.
- If the applied positional embedding can handle different image sizes, ***the transformer will handle different image sizes as well.***

Details on ViT architectures in a later lecture.

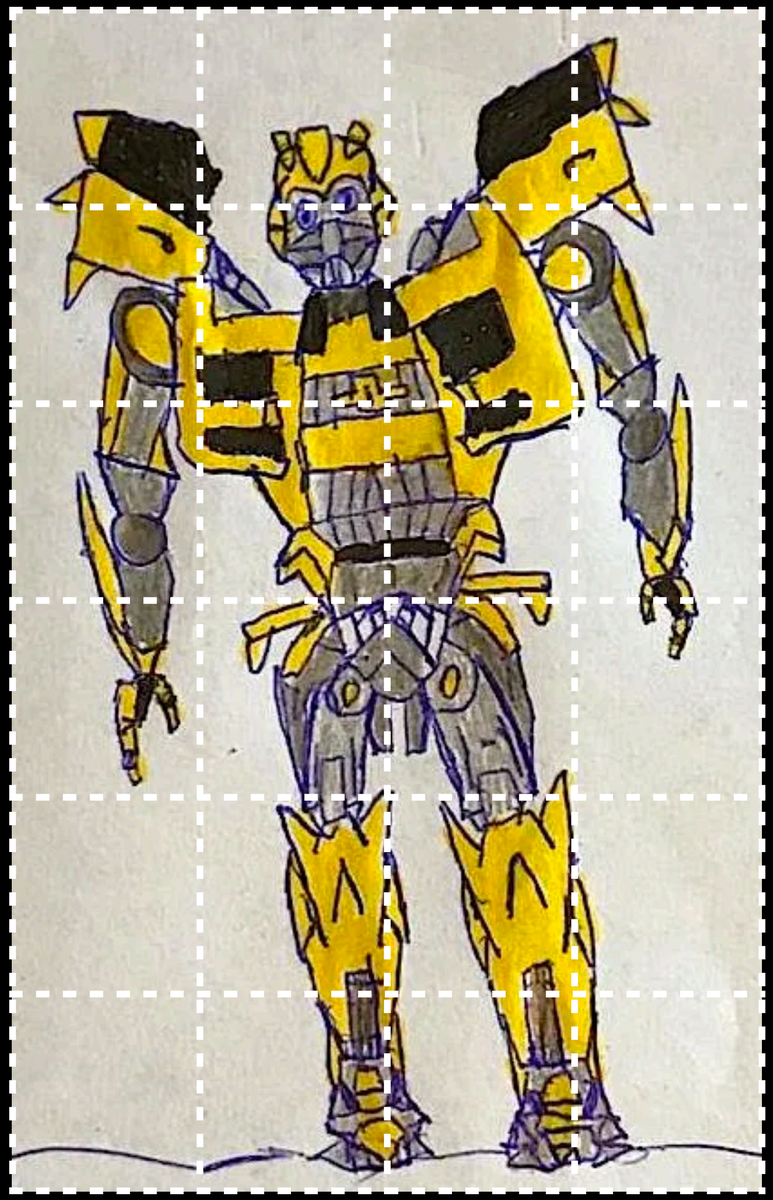
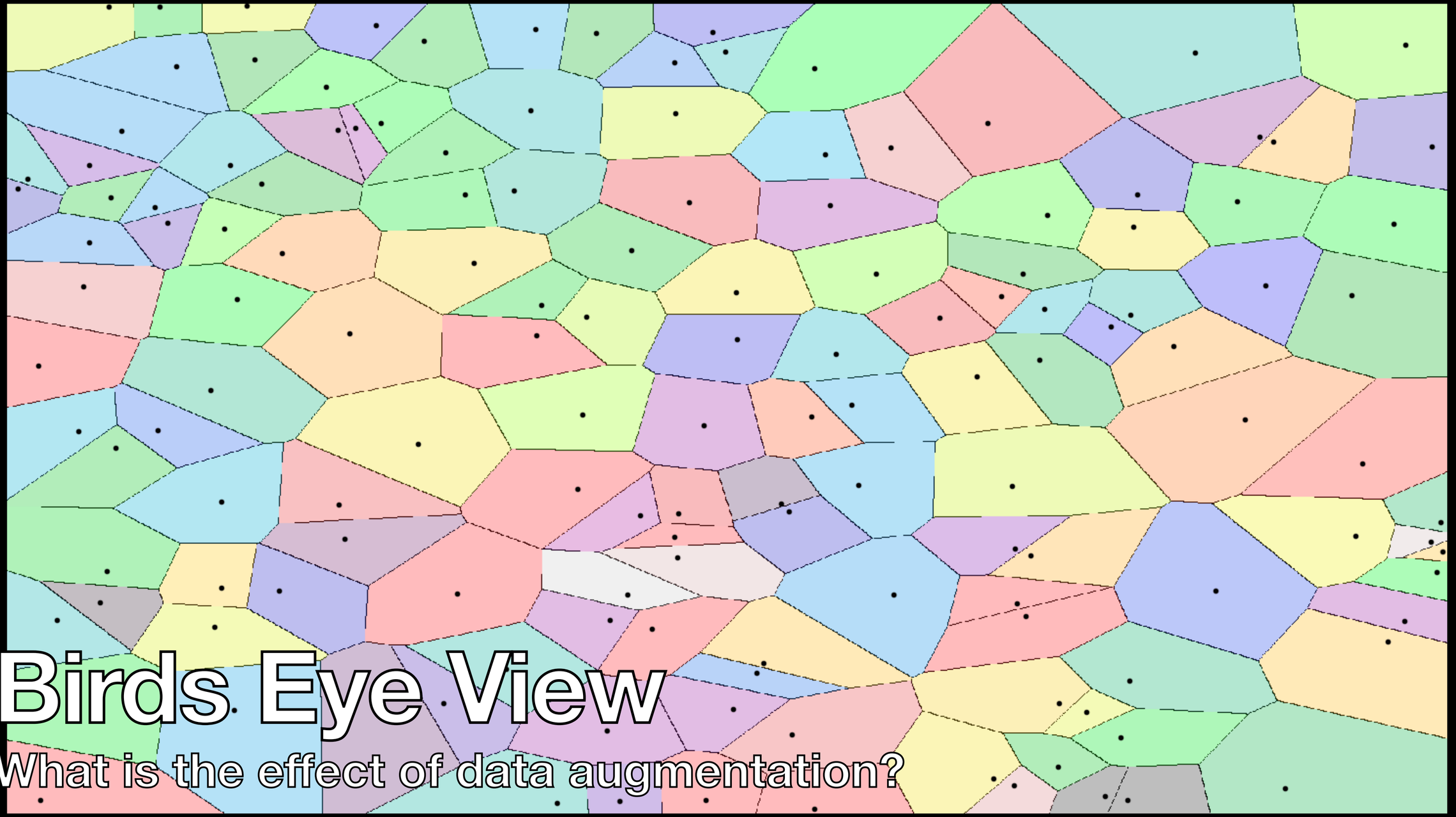


Image credit: V.B.B.B.Drawings - opensea.io

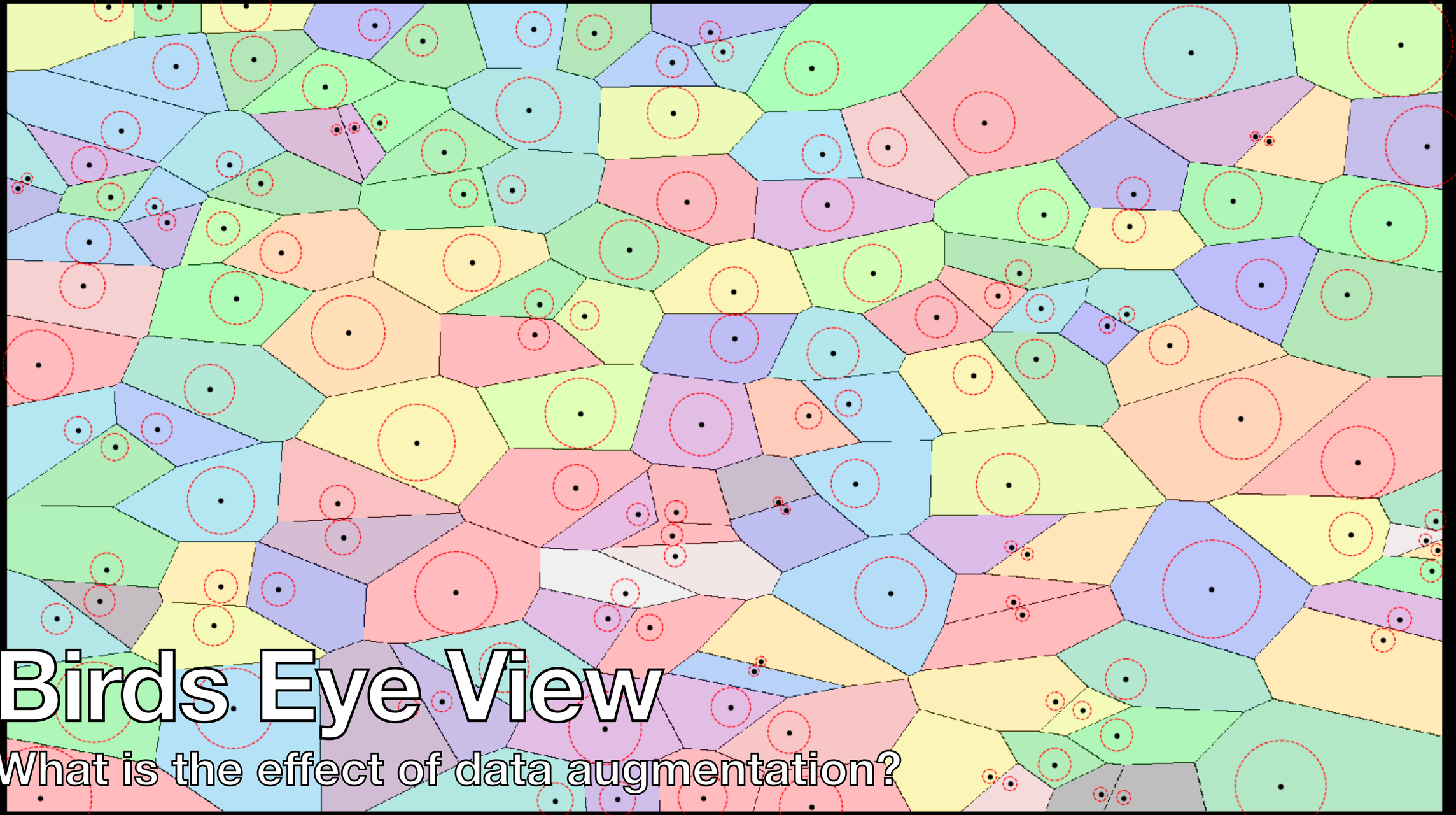
Ask questions at menti.com - Code: 2515 1387



Birds Eye View

What is the effect of data augmentation?

Ask questions at menti.com - Code: 2515 1387



Birds Eye View

What is the effect of data augmentation?

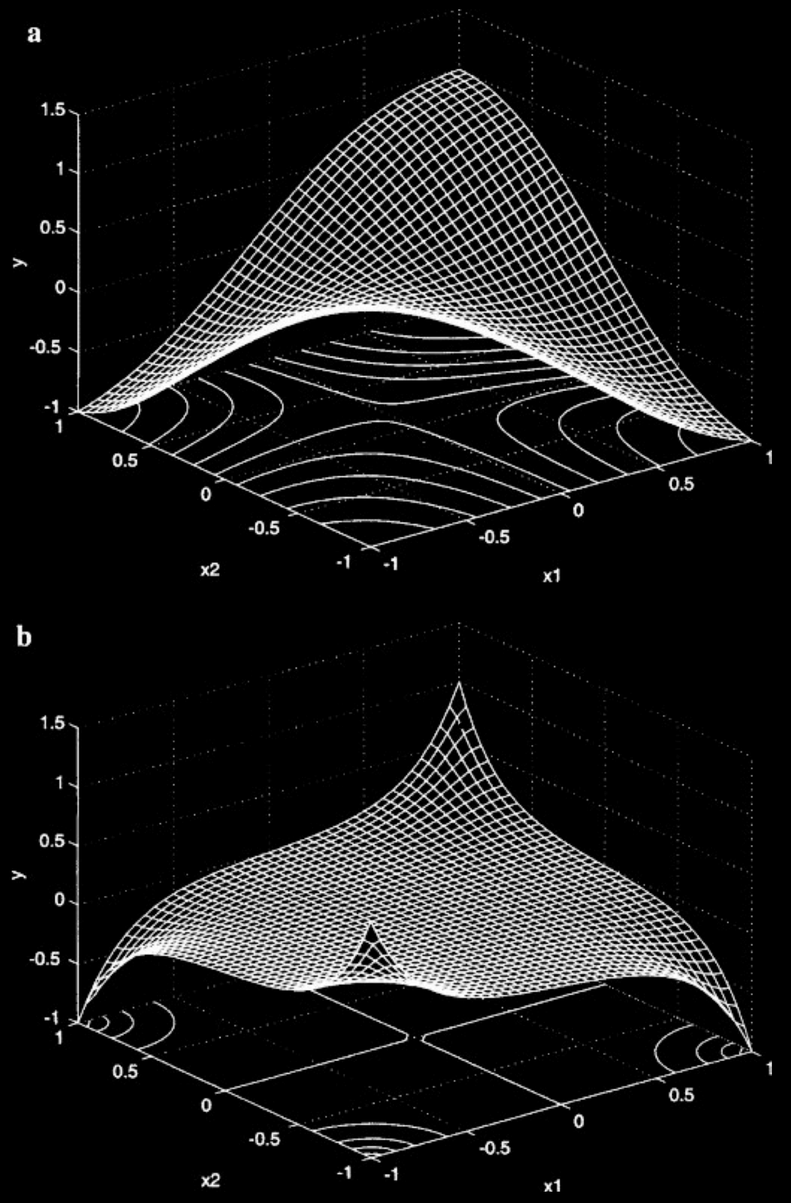
Ask questions at menti.com - Code: 2515 1387



Second Half

Plan for the next 45 minutes

- Initialisation and standardisation.
- Pre-trained weights and transfer learning.
- Noisy student.
- Contrastive learning.
- BYOL / SimSiam.



Weight Initialisation

Getting a head start

Good weight initialisation accelerates convergence and learning.

- **Identity**: Initialise the weights such that $W_\theta x = Ix = x$.
- **Orthogonal**: Initialise the weights with a random orthogonal matrix.
- **Random**: Sample the parameters randomly from some distribution.
- **Pre-trained**: Use pre-trained weights from another neural network.
- **Unsupervised pre-training**: Initialise weights using supervised pre-training.

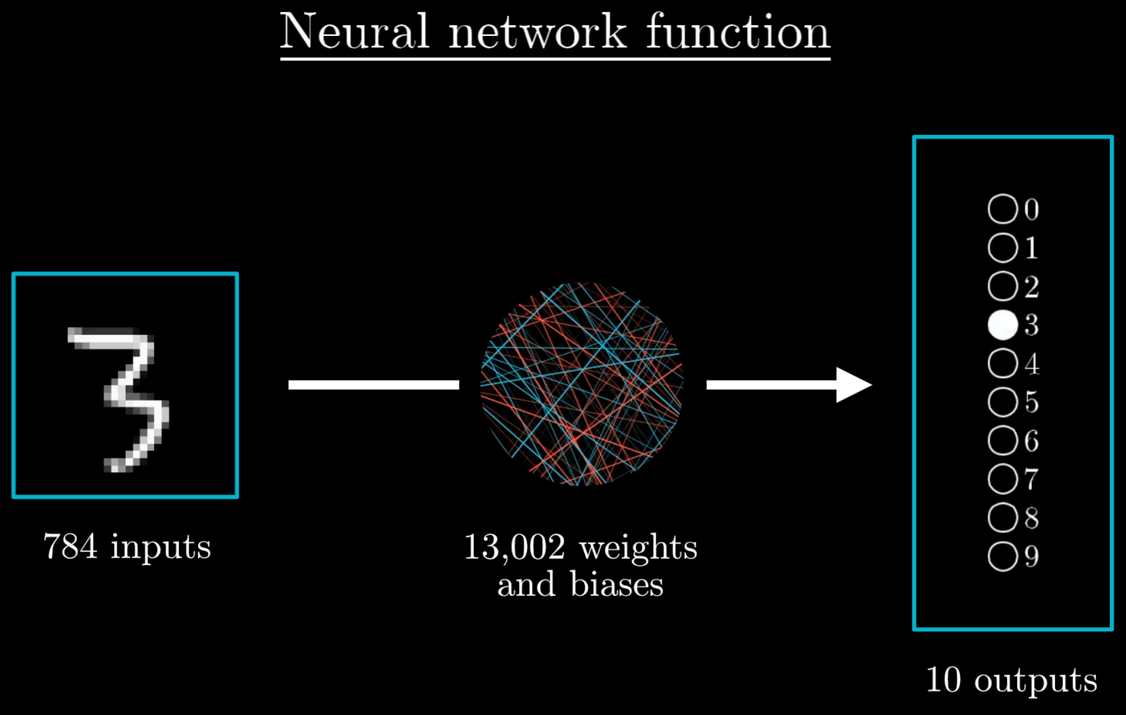


Image credit: 3Blue1Brown - Youtube

Identity Initialisation

The benefits of doing nothing

$$W_\theta x + b_\theta = Ix + 0 = x$$

Pros:

- Simple (Occam's Razor).
- No initial bias → better generalisation.
- No initial correlation → forces network to learn correlations between features.
- Unit gradients → optimal gradient propagation.
  - Dynamical Isometry property: $\|J_{W_\theta}\|_2 = 1$
  - *In other words*, the Jacobian of the weights has **unit norm.**

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix}$$

Identity Initialisation

The downside of doing nothing

$$W_\theta x + b_\theta = Ix + 0 = x$$

Cons:

- Multiplicative zeros will have no gradient signal.
  - *Dirty fix*: replace with $0 < \varepsilon \ll 1$ to learn correlations. Not optimal!
- Mostly applicable to feed-forward networks. Limited use in convolution*.
- Not optimal in all layers, mostly used in deeper / later layers.
- Drops input dimensions for matrices with fewer output dimensions.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{bmatrix}$$

* Identity convolution kernel := the Dirac kernel.

Orthogonal Initialisation

No correlation intended…

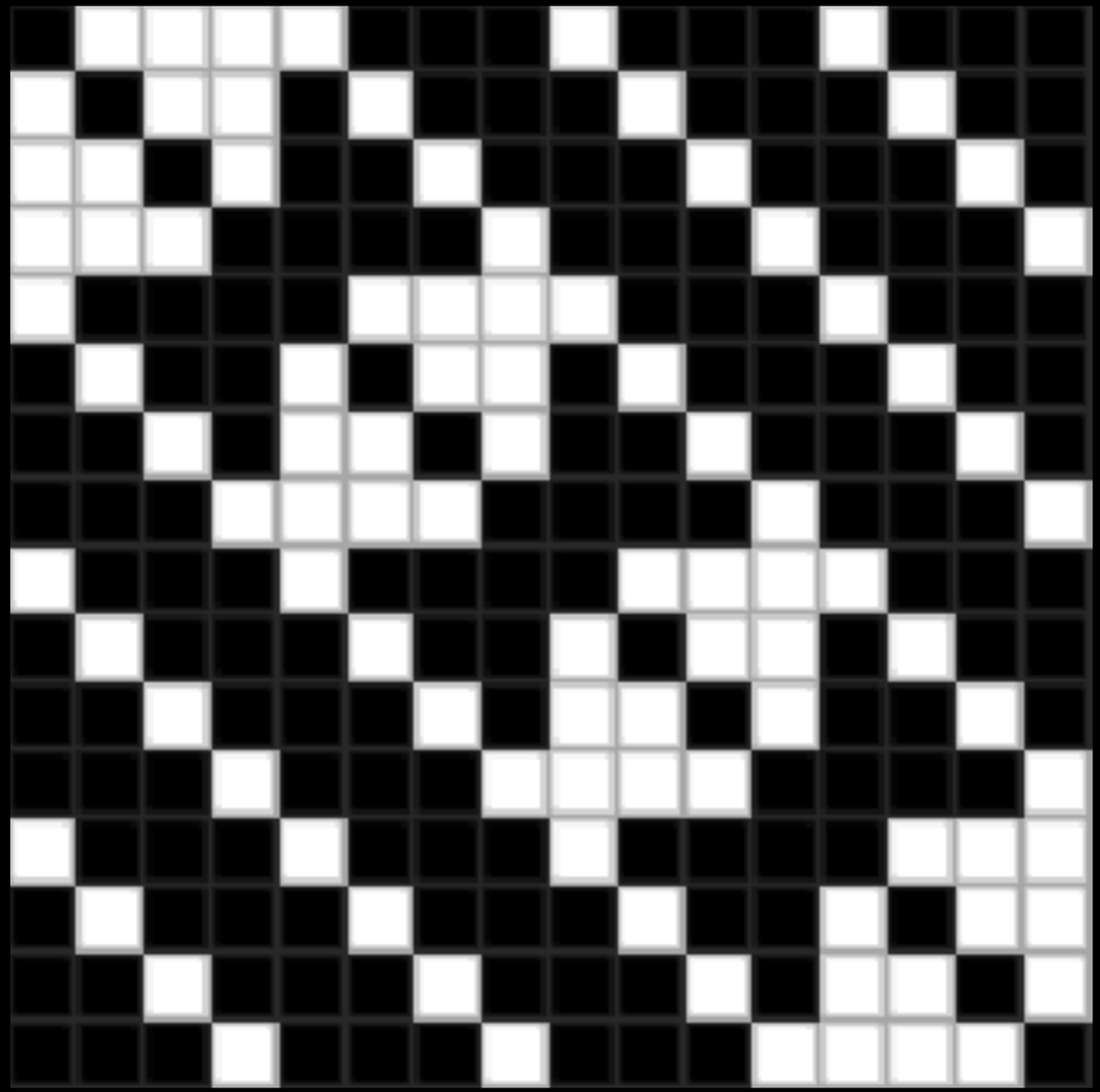
Orthogonal matrix definition: $A^T A = A A^T = I$

Pros:

- Generalisation of identity:
  - Optimal gradient prop., no correlation or bias.
- Initially invariant to rotation and scaling, can lead to better generalisation.

Cons:

- Limited applicability to smaller networks, can be numerically unstable.
- Only square matrices. Semi-orthogonal weights can be expensive to initialise for large weights.
- Tricky to initialise for convolutions.



Random Initialisation

The Vanilla Alternative

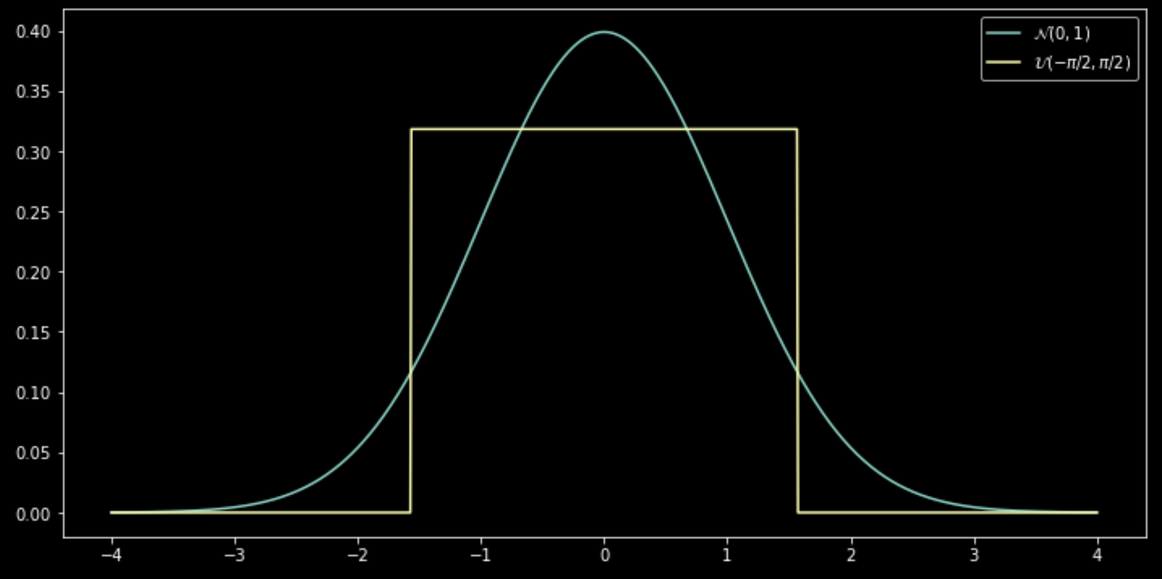
Most common method in practical use.

We commonly use either a Normal / Gaussian distribution or a Uniform distribution with zero mean for weight initialisation.

Why these distributions?

- Computationally efficient.
- Empirically sound, tried and tested.
- Uniform and normal distributions are considered **non-informative** distributions.

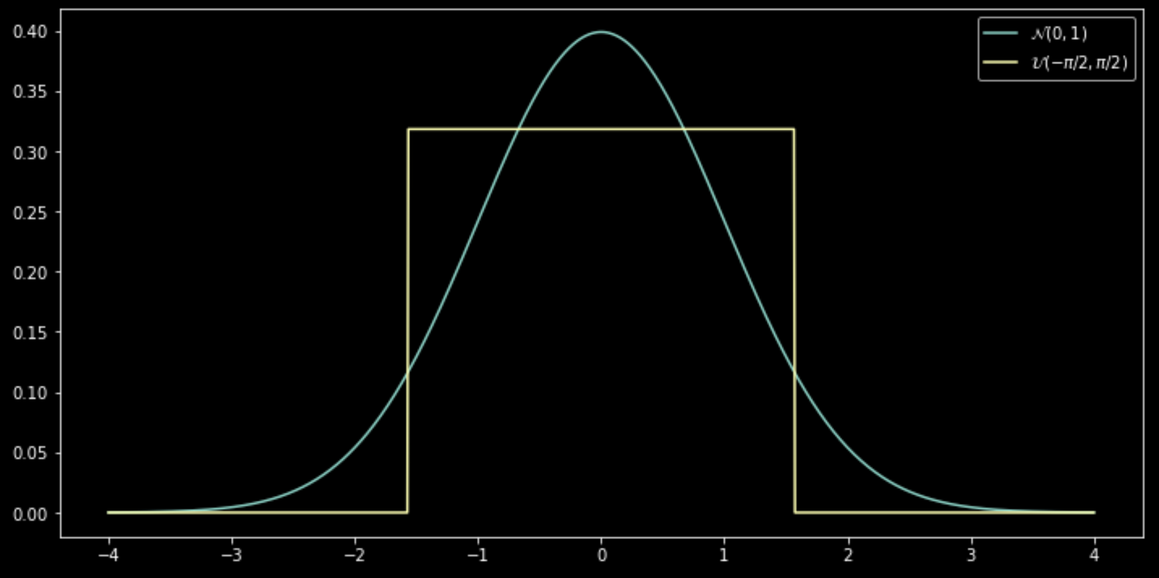
What does non-informative imply?



Random Initialisation

A non-informative choice

- An indirect reason for choosing Normal and Uniform distributions are that they are often used as ***prior distributions*** in Bayesian statistics.
- These distributions are ***non-informative,*** meaning they do not encode any specific assumptions or knowledge about the parameters being estimated.
- Non-informative → less bias.
- **NOTE**: Be wary of referring to the initialisation distributions as **priors**. We do not explicitly constrain the distributions over the weights with SGD!



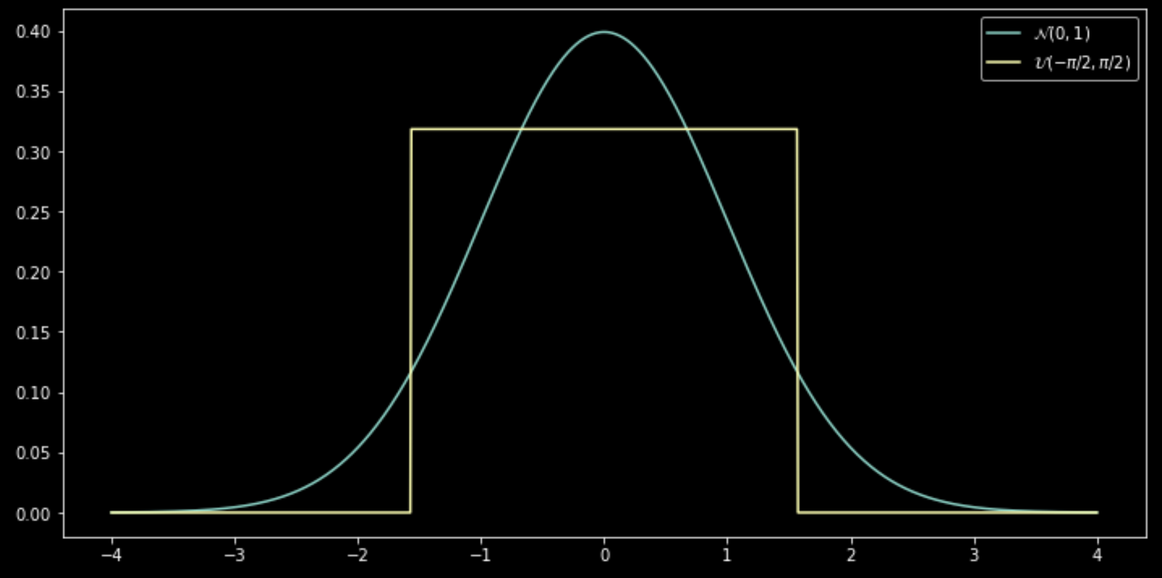
Random Initialisation

More good reasons to be random…

Asymptotic Orthogonality

- By LLN* the expected inner product of two zero-mean vectors with iid.** entries is asymptotically zero.
- This means large randomly initialised zero-mean matrices are approximately row-orthogonal, $A^T A \approx \mathrm{diag}(a)$.***
- Thus, *many of the nice properties from previous slides follow!*

Which properties do not follow?



* LLN := The Law of Large Numbers states that $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} \mathrm{X} = \mathbb{E}[\mathrm{X}]$

** iid. := Independent and Identically Distributed

*** Note that the inner products with a zero-mean random vector and itself is NOT asymptotically zero, since $\sum_{i=1}^{\infty} a_i^2 = 0$ iff. $(a_i = 0) \forall (i \in \mathbb{N}_+)$

Weight Normalisation

Achieving full unity

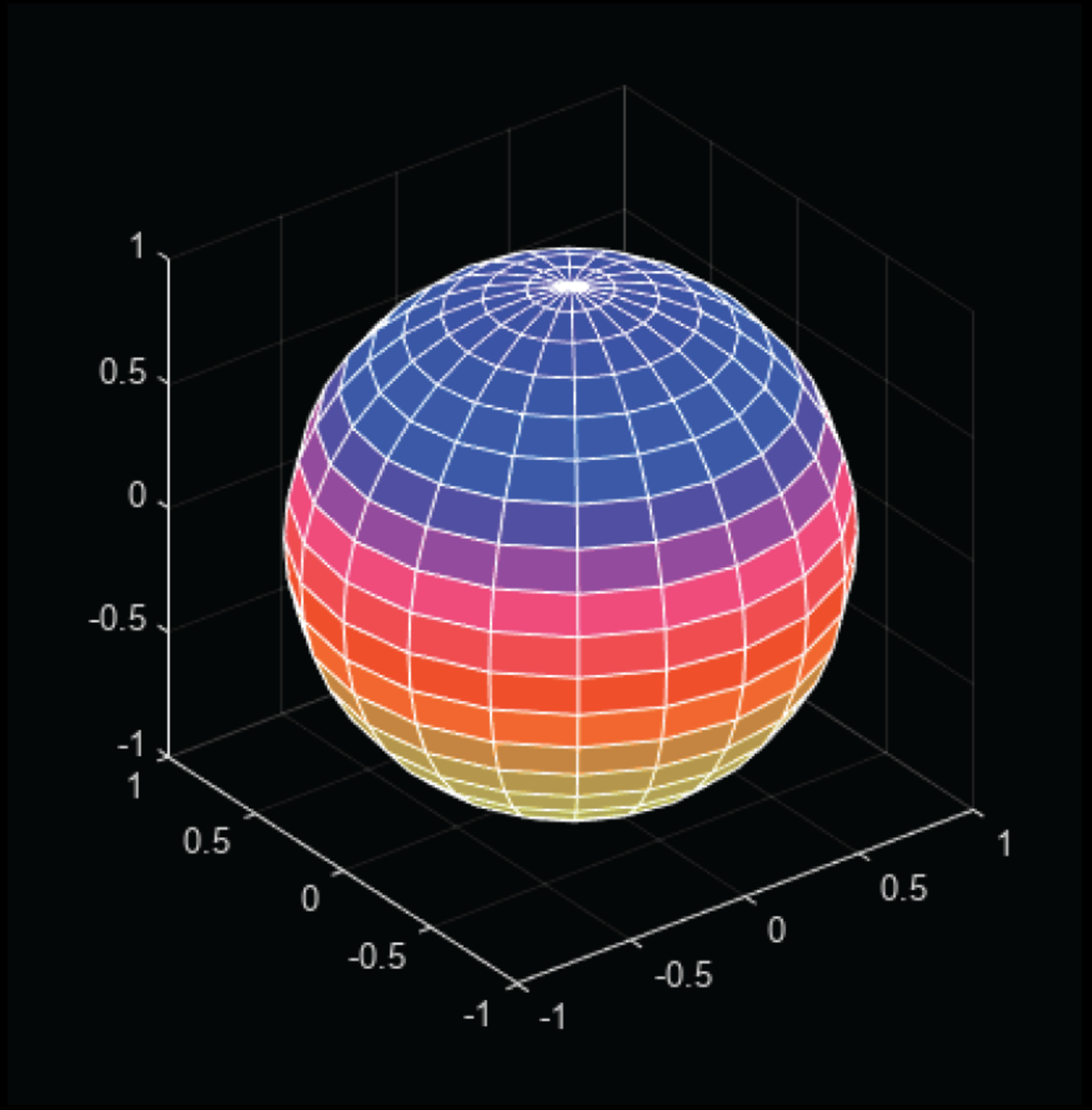
A quick method for getting unit norm weights

- Sample $\hat{W}_\theta \sim \mathbb{P}_{\text{init}}$
- Set $W_\theta = \frac{\hat{W}_\theta}{\|\hat{W}_\theta\|_2}$, where $\|\hat{W}_\theta\|_2 = \varsigma_{\max}(\hat{W}_\theta)$ *

Code example:

```
def init_unit_weights_(W, initfunc_, **params):
   # Remember, underscore functions are in-place
   initfunc_(W, **params)
   W.div_(W.svd().S[0])

# Example of use
initfunc_ = torch.nn.init.kaiming_uniform_
linear = nn.Linear(256, 128)
init_unit_weights_(linear.weight.data, initfunc_)
```



* $\varsigma_{\max}(\hat{W}_\theta)$:= Largest singular value of $\hat{W}_\theta$.

Standardisation

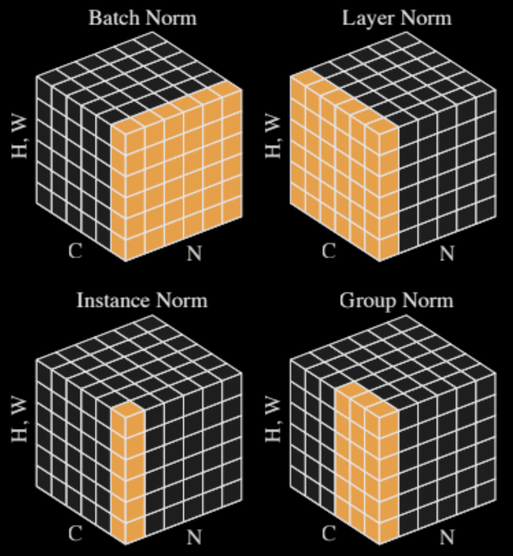
A bog standard trick…

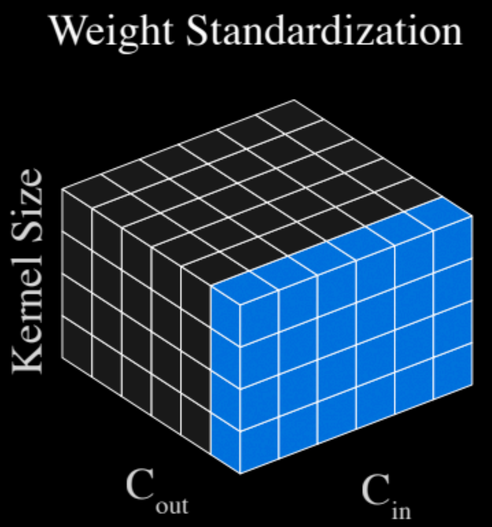
Instead of normalising to unit norm, we standardise to zero-mean unit variance:

- Sample $\hat{W}_\theta \sim \mathbb{P}_{\text{init}}$
- Let $\hat{\mu} = n^{-1} \sum_{j=1}^{n} \hat{W}_\theta$.
- Let $\hat{\sigma} = \sqrt{n^{-1} \sum_{j=1}^{n} (\hat{W}_\theta - \hat{\mu})^2}$.
- Set $W_\theta = \dfrac{\hat{W}_\theta - \hat{\mu}}{\hat{\sigma}}$.

Designed to work with Group/Instance/Layer normalisation instead of Batch normalisation.

Particularly useful for Convolution, where unit normalisation is more of a hassle.





Initialisation Schemes

Two Commonly Used Initialisations

Want: $\mathrm{Var}\big(\sigma(W_\theta x + b_\theta)\big) \approx \mathrm{Var}(x)$

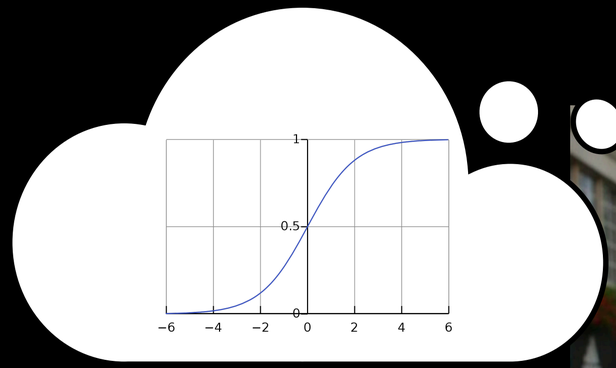
Xavier Glorot (Uniform or Normal)

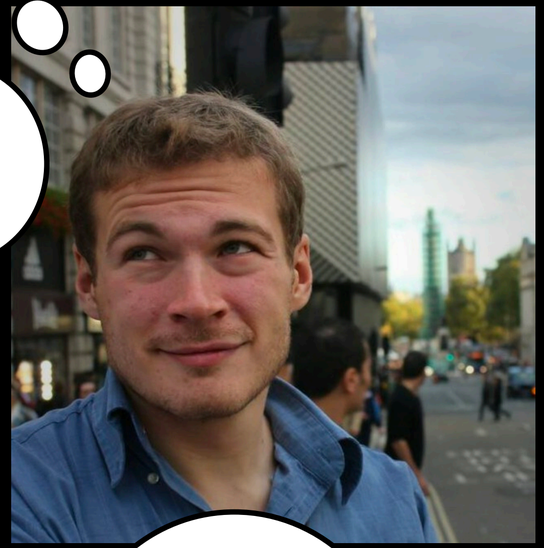
- Assume $x \perp\!\!\!\perp W_\theta$ *, and $\sigma =$ Sigmoidal.

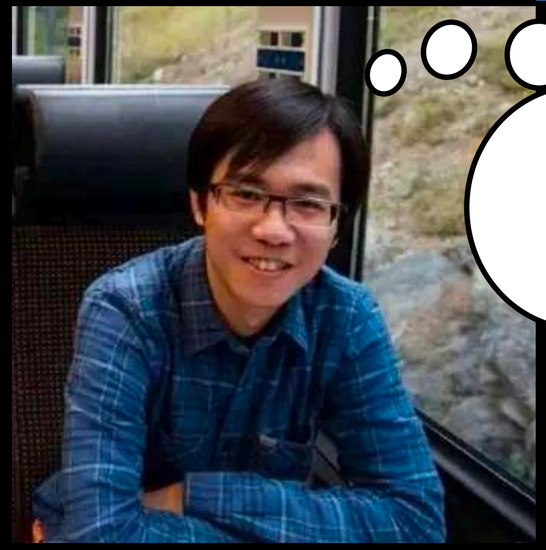
Kaiming He (Uniform or Normal):

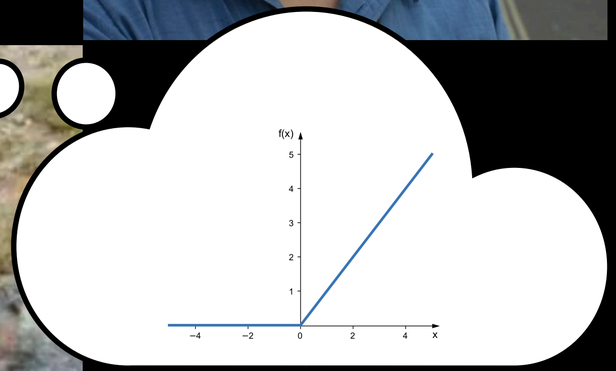
- Assumes $x \perp\!\!\!\perp W_\theta$, and $\sigma =$ Rectified.

For a full derivation of these results, check out last years notes by Alexander Binder.







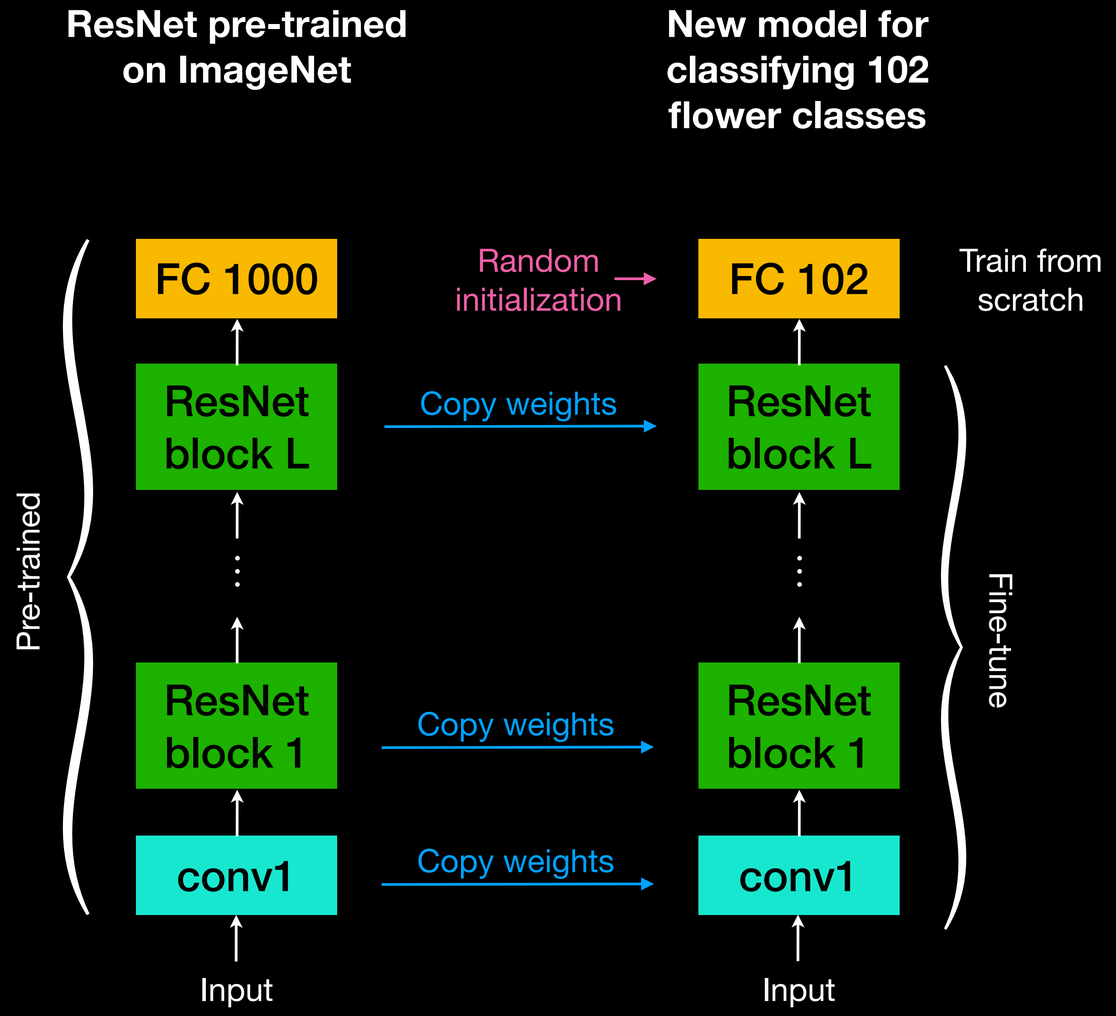


* $x \perp\!\!\!\perp W_\theta$:= x is independent of W_θ.

Pre-Trained Weights

Transfer Learning

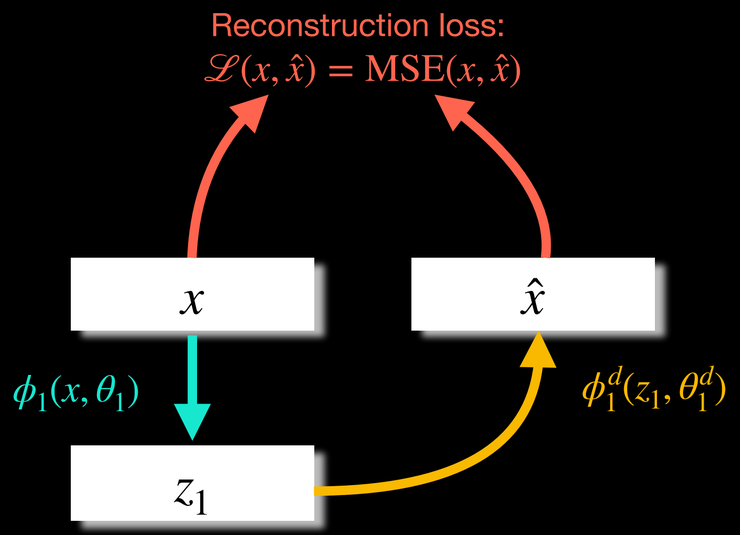
- Take weights from network that was trained to solve a similar task
  - Leverage what has been learned before
  - Avoid starting from scratch
  - Many pre-trained weights for various network architectures available with PyTorch!
- Example: ResNet trained for classification on ImageNet
- Then fine-tune network/weights on your data
- See previous lectures for more info on fine-tuning



Unsupervised Pre-Training

Pre-train your own weights!

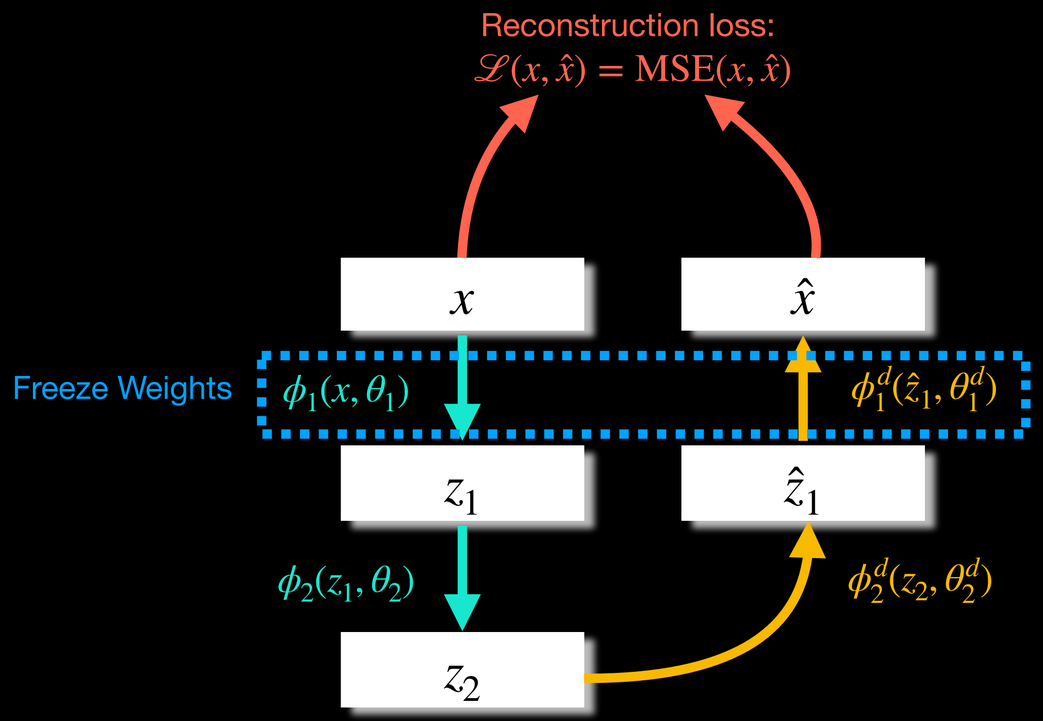
- In ancient times, Bengio (2007) thought of a clever way of initialising good weights.
- Train each layer $\phi_1, \ldots, \phi_\ell$ with an additional *decoder layer* $\phi_1^d, \ldots, \phi_\ell^d$.
- The *decoders* are trained to reconstruct the input of the layer.



Unsupervised Pre-Training

Pre-train your own weights!

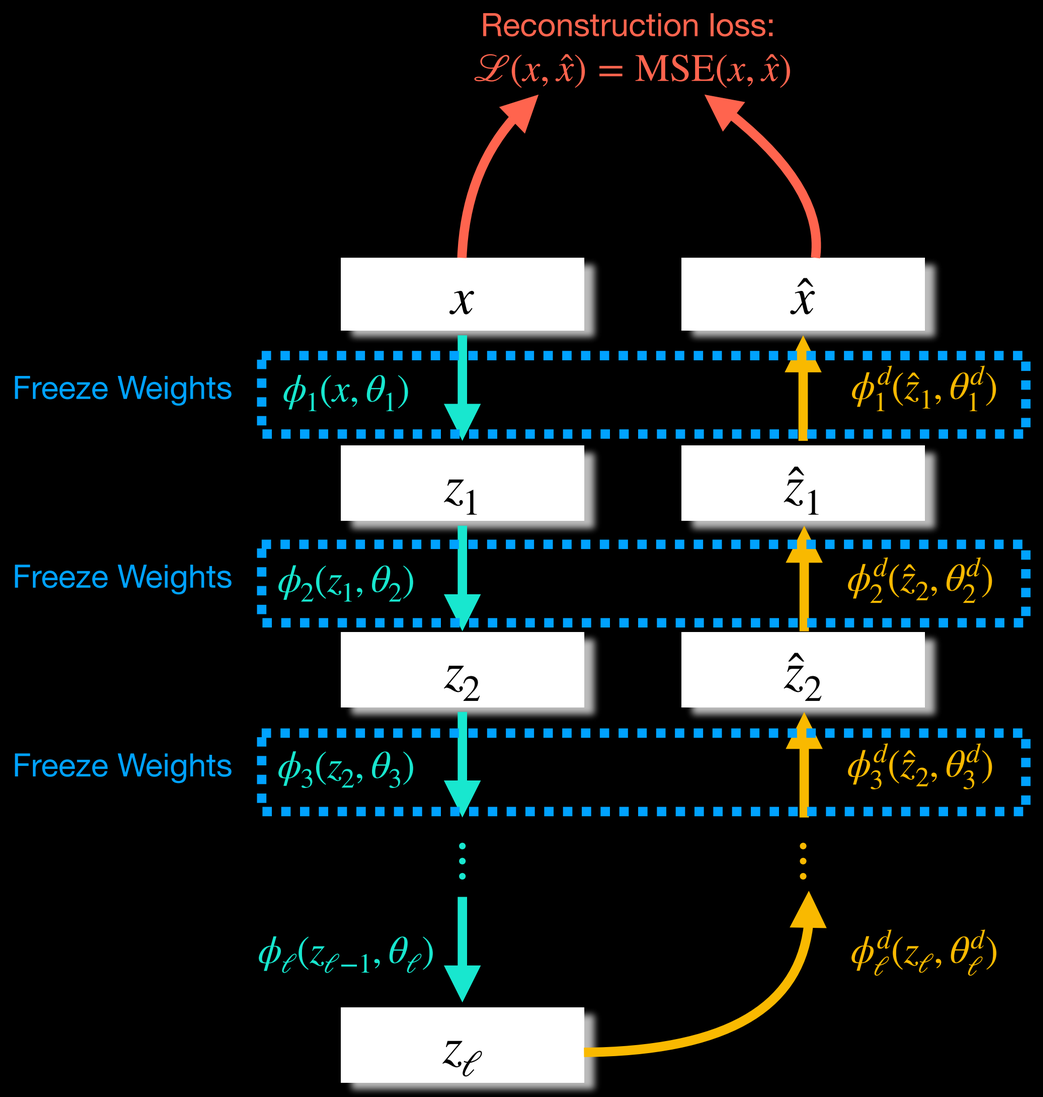
- In ancient times, Bengio (2007) thought of a clever way of initialising good weights.
- Train each layer $\phi_1, \ldots, \phi_\ell$ with an additional *decoder layer* $\phi_1^d, \ldots, \phi_\ell^d$.
- The *decoders* are trained to reconstruct the input of the layer.
- Original paper trains layers consecutively, and freezes the weights.



Unsupervised Pre-Training

Pre-train your own weights!

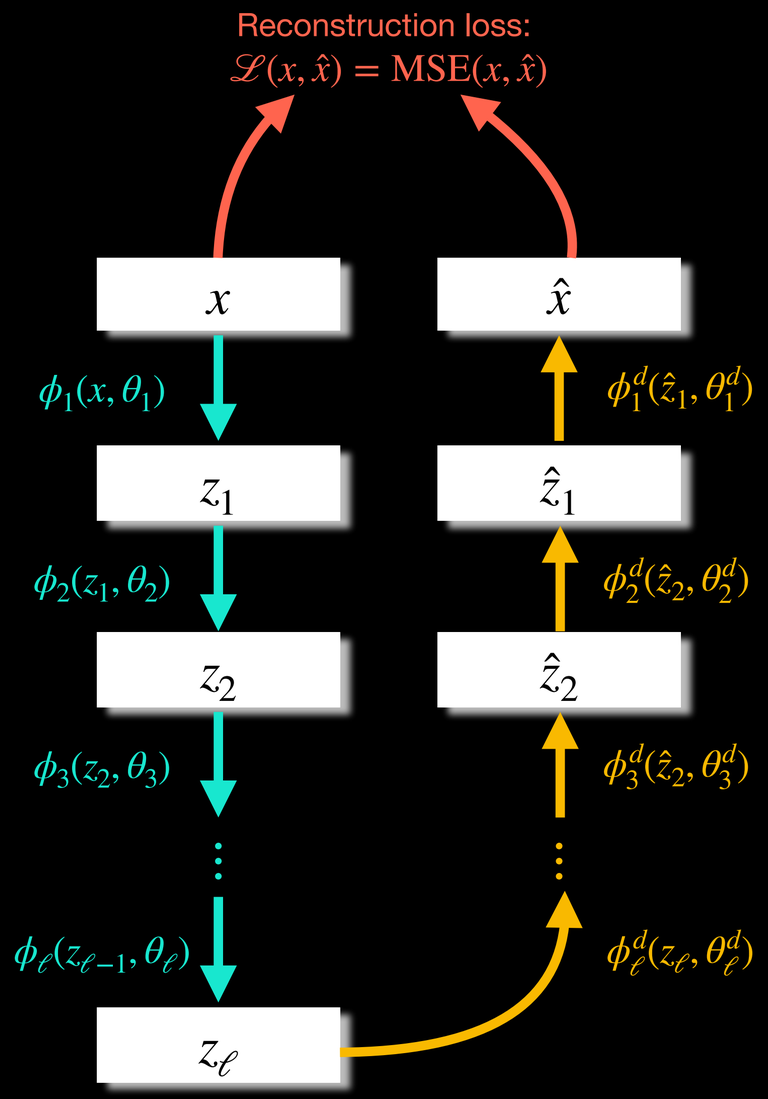
- In ancient times, Bengio (2007) thought of a clever way of initialising good weights.
- Train each layer $\phi_1, \ldots, \phi_\ell$ with an additional *decoder layer* $\phi_1^d, \ldots, \phi_\ell^d$.
- The *decoders* are trained to reconstruct the input of the layer.
- Original paper trains layers consecutively, and freezes the weights.



Unsupervised Pre-Training

Pre-train your own weights!

- In ancient times, Bengio (2007) thought of a clever way of initialising good weights.
- Train each layer $\phi_1, \ldots, \phi_\ell$ with an additional *decoder layer* $\phi_1^d, \ldots, \phi_\ell^d$.
- The *decoders* are trained to reconstruct the input of the layer.
- Original paper trains layers consecutively, and freezes the weights.
- Alternatively, layers can be trained all at once as an **autoencoder**.



Contrastive Learning

No labels, no problem

- Pre-train on large dataset without labels
- Use augmentations to create two different "views" of the same image
- Train a network to recognise augmented instances of the same image
- Similarity score: Measure of how similar two embeddings are
  - Example: Cosine similarity (see next slide)
- Augmentations of the same image should have high similarity scores
- Augmentations of different images should have low similarity scores
- Heavily relies on the chosen augmentation techniques!

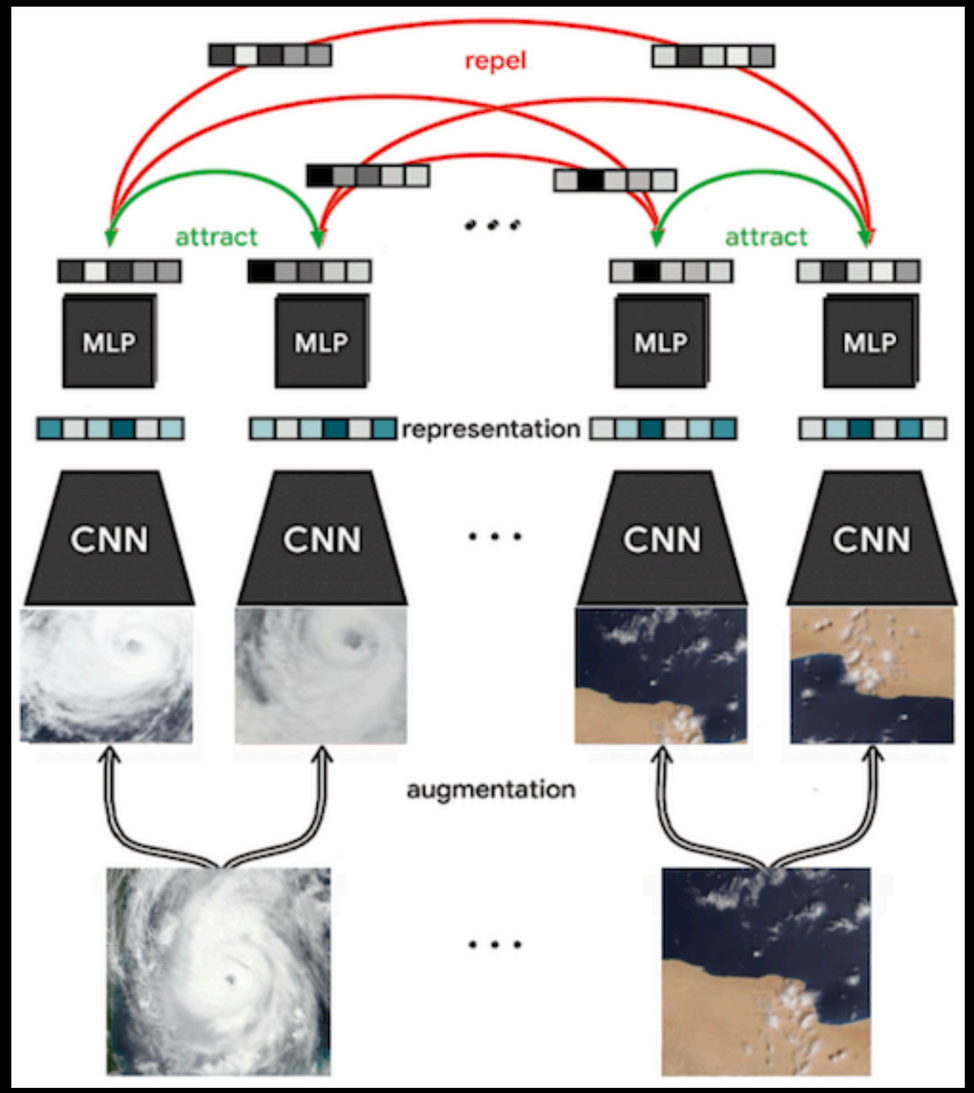


Illustration of SimCLR framework by Tom Small from the official SimCLR GitHub (https://github.com/google-research/simclr)

Contrastive Loss

SimCLR version

SimCLR: "Simple framework for Contrastive Learning of visual Representations"

- Sample a mini-batch of N images
- Create two views of each image by applying a random selection of simple augmentations
- Positive pairs = views of the same image
  Negative pairs = views of different images
- Cosine similarity: $\mathrm{sim}(u, v) = \frac{u^T v}{\|u\| \|v\|}$
  is equivalent to the cosine of the angle between two vectors
- Contrastive loss for positive pair $\tilde{x}_i, \tilde{x}_j$:

$$\ell_{i,j} = -\log \frac{\exp\left(\mathrm{sim}(z_i, z_j)/\tau\right)}{\sum_{k \neq i} \exp\left(\mathrm{sim}(z_i, z_k)/\tau\right)}$$

  where τ is a temperature parameter
- Downside: Requires vary large batch sizes!

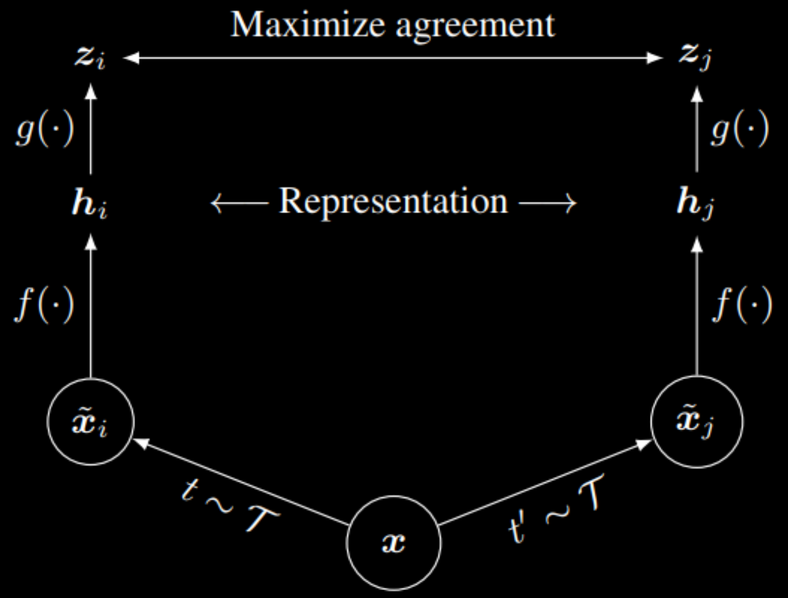


Figure 2. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.



Contrastive Learning

Think-pair-share

Question:

What happens if the negative samples in a batch are very similar to the positive samples?

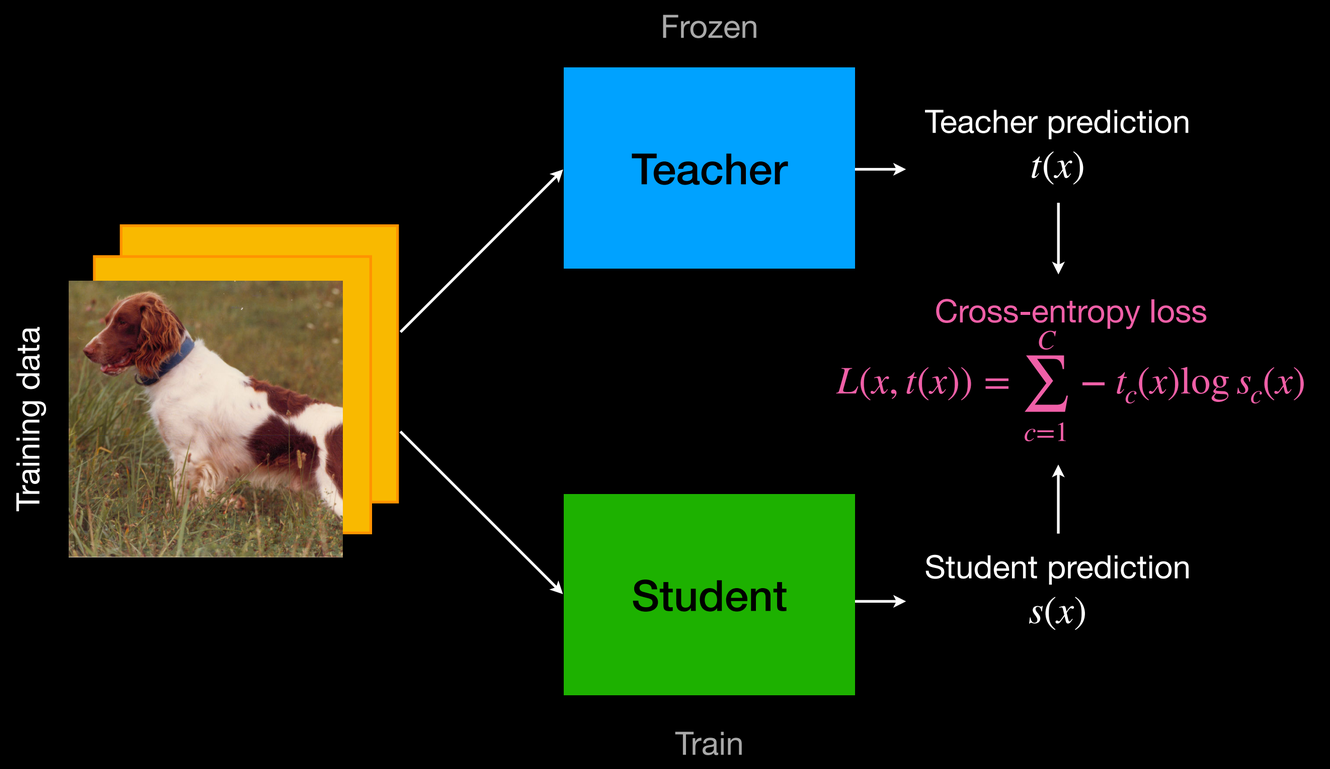
Instructions:

1. Think individually: 2 min
2. Discuss in pairs: 3 min
3. Share in menti

Student-Teacher Learning

Knowledge distillation/model compression

- Train a smaller "student" model from a large/complex "teacher" model
- Why? For situations where a large model is too expensive/slow to run e.g. on smaller devices
- Idea: train the student model $s(\,.\,)$ by using the softmax output $t(x)$ of the teacher model as soft labels
- Cross-entropy loss using teacher output $t(x)$ as ground truth labels
- This prioritises learning of easier samples!
  - Difficult samples: Class probabilities in $t(x)$ are all close to $1/C$. Then the cross-entropy loss is lower.
  - Easy samples: The teacher is very certain, i.e. $t(x)$ is close to a one-hot label. Then the cross entropy loss is higher.



Noisy Student

A semi-supervised teacher-student approach

- Semi-supervised: Some data with labels, some without
- Labeled images $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, and unlabeled images $\{\tilde{x}_1, \ldots, \tilde{x}_m\}$
- Student model is always equal-or-larger than teacher
- Cross-entropy loss:

$$\mathscr{L}(y, \hat{y}) = \sum_c -y_c \log \hat{y}_c$$

- Noising the models: RandAugment, dropout, and stochastic depth

Algorithm: *Noisy Student Training*

1. Train the teacher θ_*^t on the labeled images $\{(x_i, y_i)\}_{i=1}^n$ with cross entropy loss

$$\frac{1}{n} \sum L(y_i, f^{noised}(x_i, \theta^t))$$

2. Use the teacher to generate pseudo labels $\tilde{y}_i = f(\tilde{x}_i, \theta_*^t)$ on the unlabeled images $\{\tilde{x}_i\}_{i=1}^m$

3. Train a student model on all the data with ***noise*** added to the student model

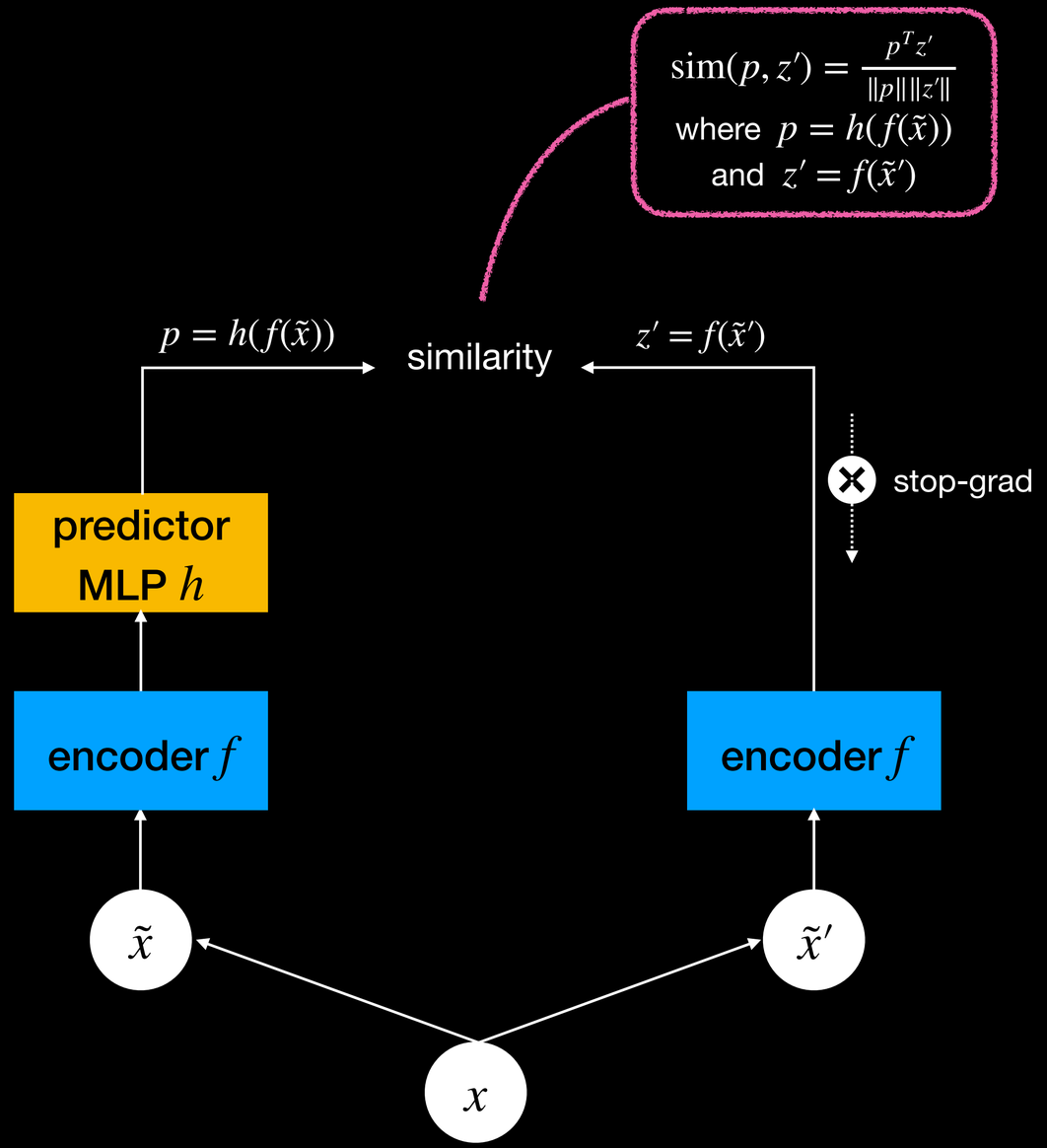
$$\frac{1}{n} \sum_{i=1}^{n} L(y_i, f^{noised}(x_i, \theta^s)) + \frac{1}{m} \sum_{i=1}^{m} L(\tilde{y}_i, f^{noised}(\tilde{x}_i, \theta^s))$$

4. Iterative training: Use the student as a teacher, and repeat step 2 and 3 to train a *new* student model

SimSiam

Simple siamese representation learning

- Trained by maximising the similarity of positive pairs i.e. minimising the negative cosine similarity of the outputs
  $d(p, z') = -\operatorname{sim}(p, z') = -\frac{p^T z'}{\|p\| \|z'\|}$
  where $p = h(f(\tilde{x}))$ and $z' = f(\tilde{x}')$
- No negative pairs, doesn't need large batch size like SimCLR
- Loss per image:
  $L = \frac{1}{2} d(p, \text{stopgrad}(z')) + \frac{1}{2} d(p', \text{stopgrad}(z))$
- Stop gradient operation: Treat z as a constant. Encoder receives no gradient from z.
- Q: What happens if we skip the stop-grad?
  A: Output collapses to a constant



BYOL

Bootstrap your own latent

- Online network tries to predict output of target network
- Parameters ξ of target network are an exponential moving average (EMA) of the parameters θ of the online network

$$\xi \leftarrow \tau\xi + (1-\tau)\theta$$

where $\tau \in [0,1]$ is the decay rate

- Minimise:

$$\mathscr{L}(p_\theta, z'_\xi) = \|\overline{p_\theta} - \overline{z_\xi}'\|_2^2 = 2 - 2 \cdot \frac{\langle p_\theta, z'_\xi \rangle}{\|p_\theta\|_2 \cdot \|z'_\xi\|_2}$$

- Loss: $L_{\theta,\xi} = \mathscr{L}(p'_\theta, z_\xi) + \mathscr{L}(p_\theta, z'_\xi)$
- At each training step: optimize loss wrt. θ only, then update ξ with the EMA step
- Downside: Two networks = double the parameters

