UNIVERSITY
OF OSLO

IN4310 Data Augmentation

Marius Aasan and Martine Hjelkrem Tan

mariuaas@uio.no matan@uio.no



Key Learning Goals

Distributional shifts, out-of-distribution samples, and outliers.

Methods for improving generalization and domain adaptation.

Image augmentation methods and their implementations.

Methods for applying fixed NNs on varying input sizes.

Weight initialization and standardization strategies

Introduction to contrastive learning.

Motivational Example: The Cat Lady's Request

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



Motivational Example: The Cat Lady's Request

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.



Motivational Example: The Cat Lady's Request

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.



Motivational Example: The Cat Lady's Request

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.

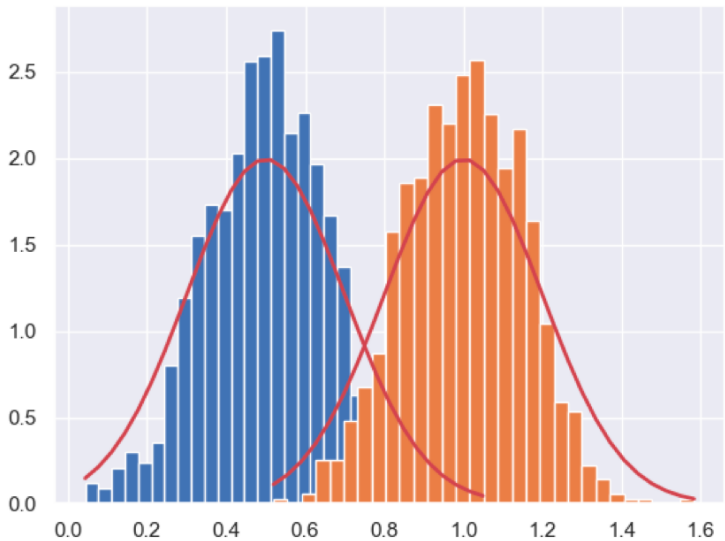


Do you see any problems with this?

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

Generalization, Domain Adaptation, and Distribution

Generalization (model property):

$f(\mathcal{D}_{\text{train}}) \approx f(\mathcal{D}_{\text{target}})$ for $\mathcal{D}_{\text{train}} \sim \mathcal{D}_{\text{target}}$

Domain adaptation (model property):

$f(\mathcal{D}_{\text{train}}) \approx f(\mathcal{D}_{\text{target}})$ for $\mathcal{D}_{\text{train}} \nsim \mathcal{D}_{\text{target}}$

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

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 from the data our model was trained on.

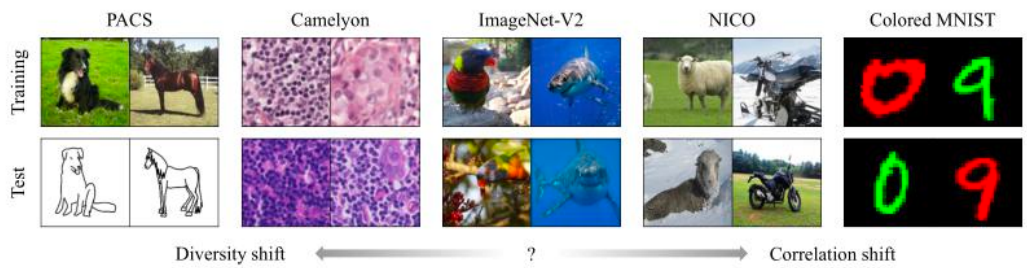


Figure 1: Examples of image classification datasets demonstrating different kinds of distribution shifts. While it is clear that the datasets at both ends exhibit apparent distribution shifts, in the middle, it is hard to distinguish the differences in distribution between the training dataset (ImageNet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

Rule #1: Preprocess your input data!

MinMax:

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

MaxAbs:

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

Standardization:

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

Why?

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

Issues with Generalization

Isn't inference data always different from training data?

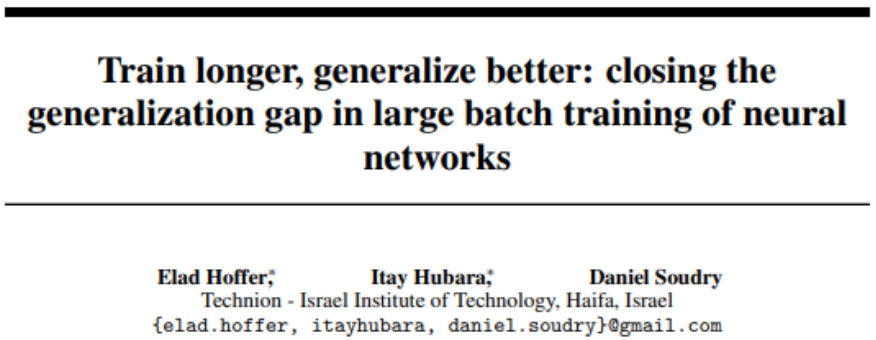
We cannot expect training data to match data at inference...

...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 $|\mathcal{D}| \to \infty$



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. It takes longer (time $\to \infty$) to train our models.

B. Infinite informational capacity risks creating a black hole.

C. We are too lazy to gather infinite samples.

Show of hands

Which of these are problematic for our approach?

A. It takes longer (time $\to \infty$) to train our models.

   Environmentally unfriendly

B. Infinite informational capacity risks creating a black hole.

C. We are too lazy to gather infinite samples.

Which of these are problematic for our approach?

A. It takes longer (time $\to \infty$) to train our models.

   Environmentally unfriendly

B. Infinite informational capacity risks creating a black hole.

   Only a problem at $t = \infty$

C. We are too lazy to gather infinite samples.

Which of these are problematic for our approach?

A. It takes longer (time $\to \infty$) to train our models.

   Environmentally unfriendly

B. Infinite informational capacity risks creating a black hole.

   Only a problem at $t = \infty$

C. We are too lazy to gather infinite samples.

   No problem, lazy = smart

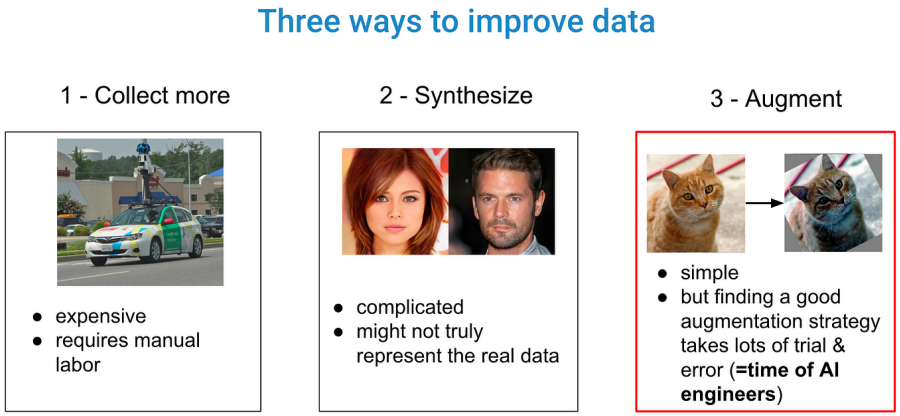
Data Augmentation

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.



Geometric of Photometric?

Two main categories of image transformations

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.



Geometric transformations

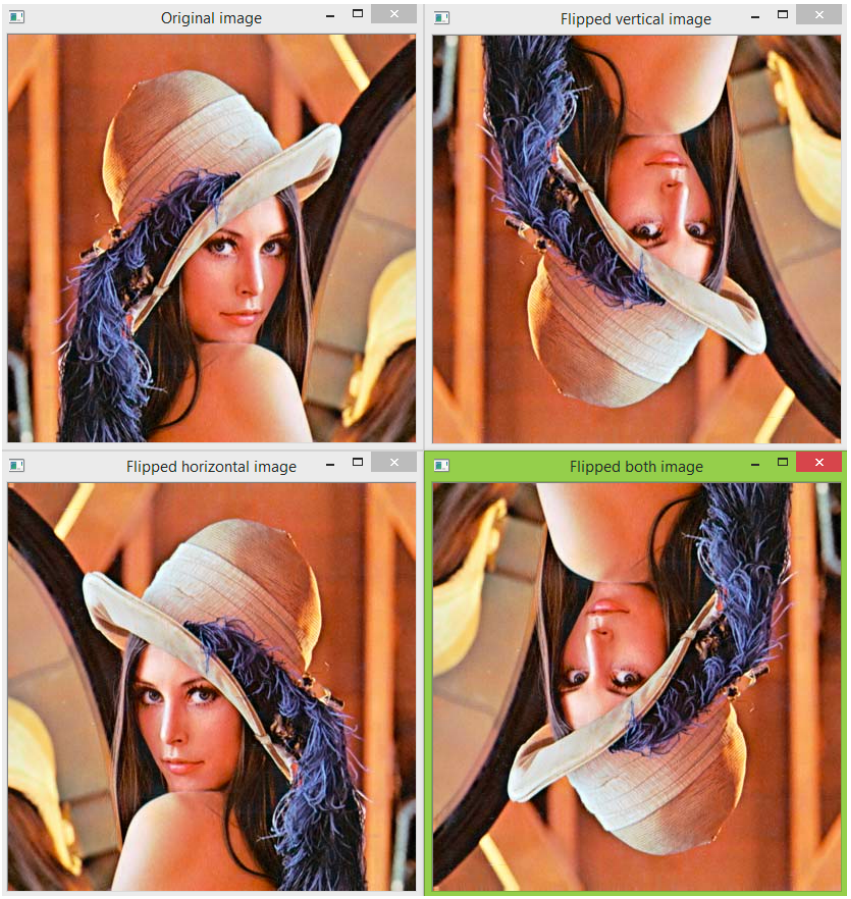
Vertical / Horizontal flip

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)

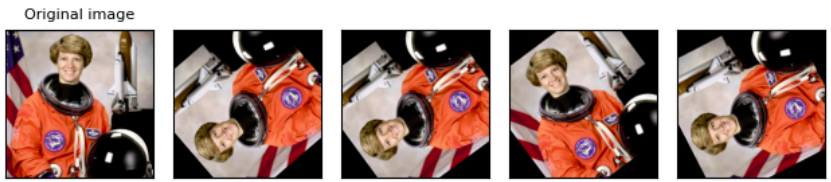
def hflip(img):
    # Input dim: B,C,H,W
    # Dimension -1 := W
    return img.flip(dim=-1)
```



Geometric transformations

Image Rotation

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



PyTorch implementation:

```
import torchvision.transforms as T

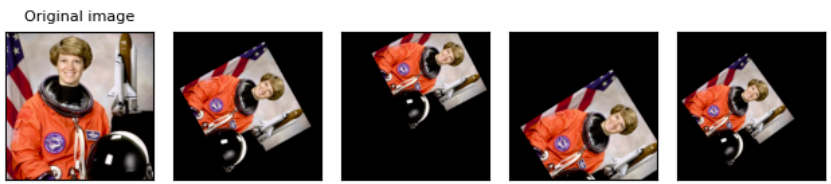
def rotation(img, angle):
    # Input dim: B,C,H,W
    return T.functional.rotate(img, angle)
```

Geometric transformations

Affine Transformations

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

More power, but more computationally expensive!



PyTorch implementation:

```
import torchvision.transforms as T

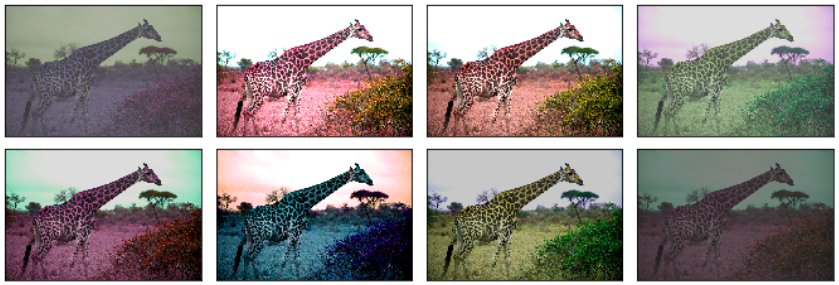
def affine(img, angle, shift, scale, shear):
    # Input dim: B,C,H,W
    return T.functional.affine(
        img, angle, shift, scale, shear
    )
```

Photometric Transforms

Color Jitter

Standard method:

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



PyTorch implementation:

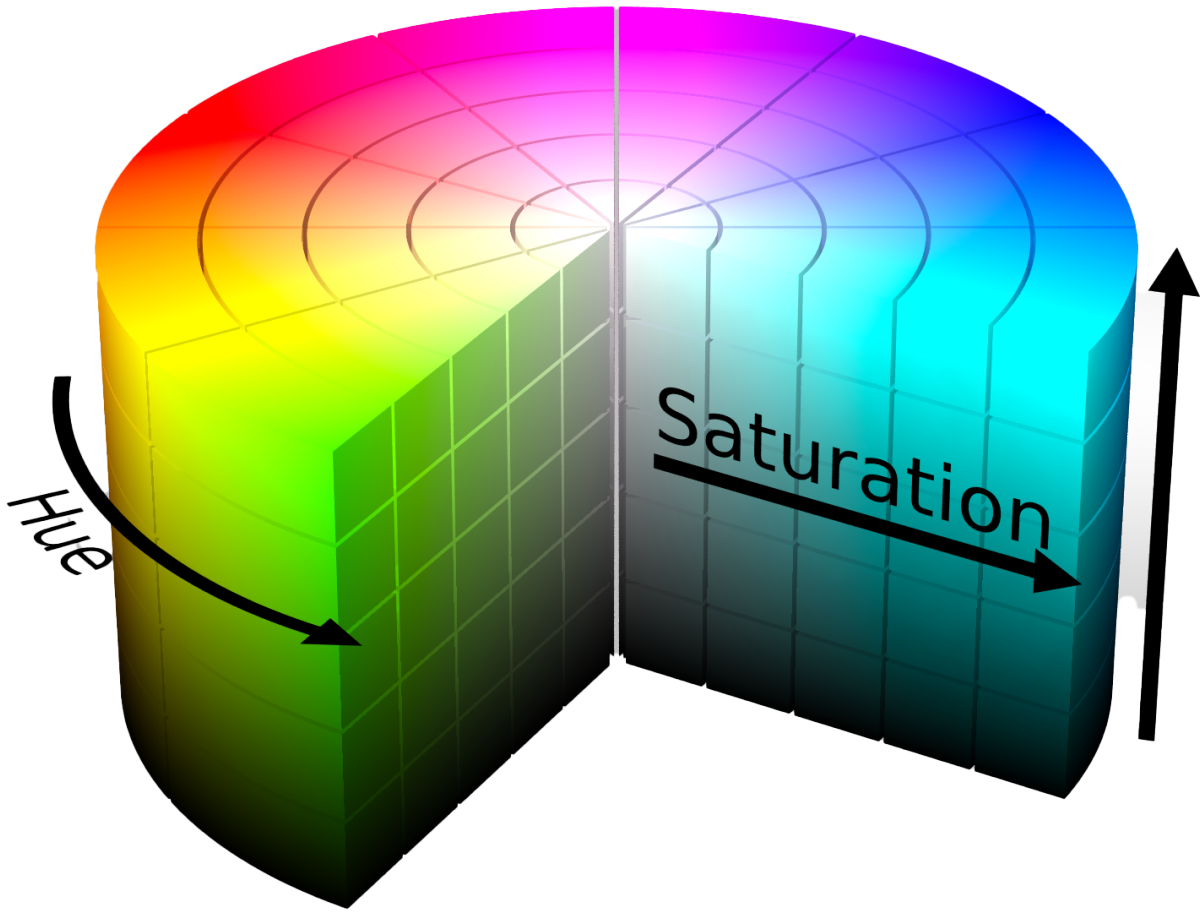
```
def color_jitter(img, intensity):
    hsvimg = rgb2hsv(img)
    eps = torch.randn(3,1,1)
    hsvimg = hsvimg + eps * intensity
    hsvimg[:,0].remainder_(1)
    hsvimg.clip_(0, 1)
    return hsv2rgb(hsvimg)
```

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 (recap)



Angular color

Photometric Transforms

Solarization

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

Implementation using PIL library:

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

PROTIP: Add stochasticity with a probability parameter!



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`

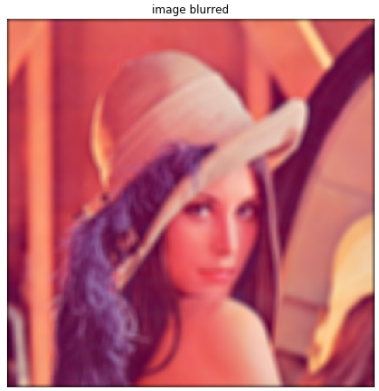
Blur Transformation

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
```

Notes:

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

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

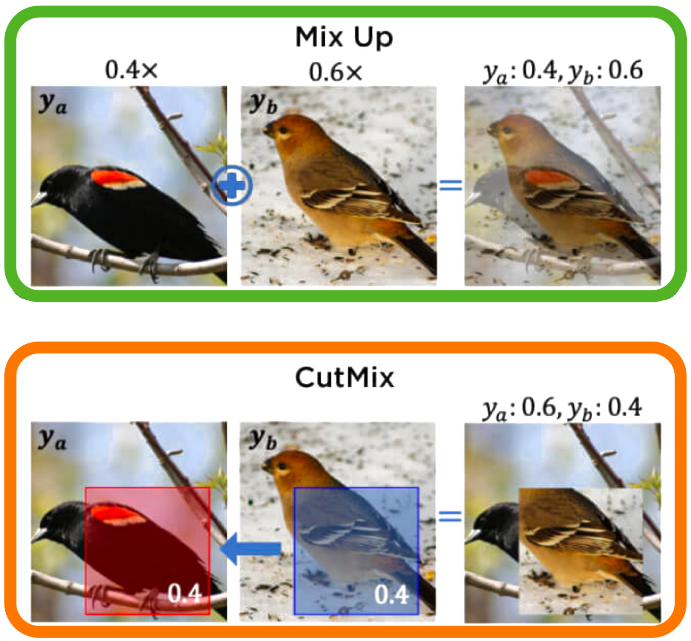
Other Augmentations

The world is your oyster!

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!



Label Smoothing

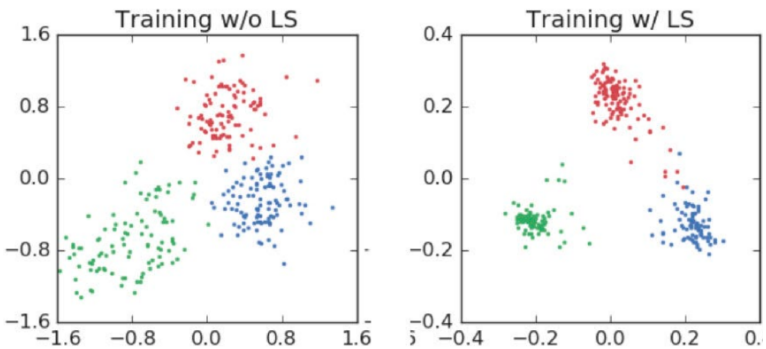
Augmenting the Labels

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 $p(y|x)$ 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



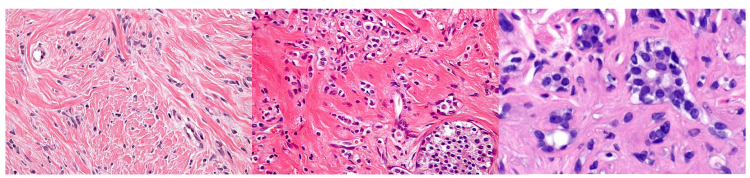
Choices of Augmentation

Which augmentations should we use?

Flags: Which flag is which?

- Grayscale
- Flip / Rotation

Pathology: Different labs use different colouring of tissue.



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?

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!”

Beyond Images

Augmentations in other domains

Images and audio: Lots of existing work on filters and transformations.

- Audio: Pitch shift, resampling, modulation, equalisation...

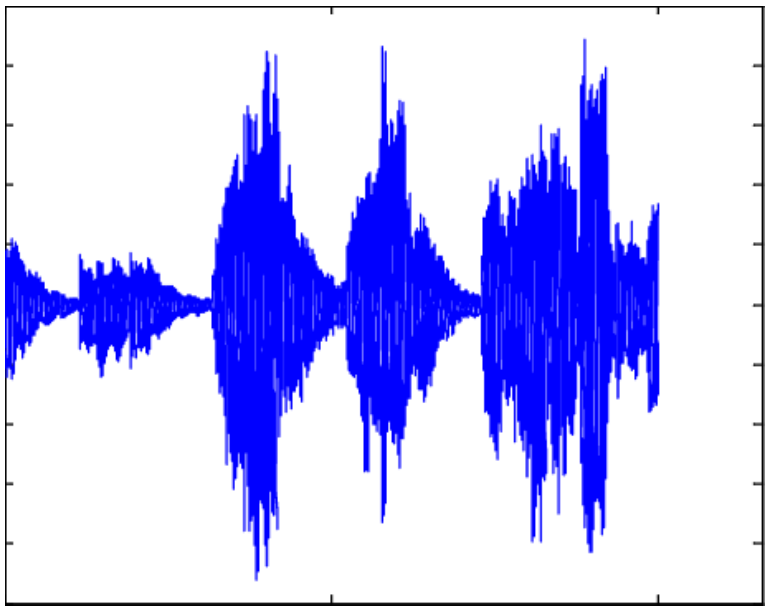
Can often be applied *"on the fly"*, computationally effective.

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.

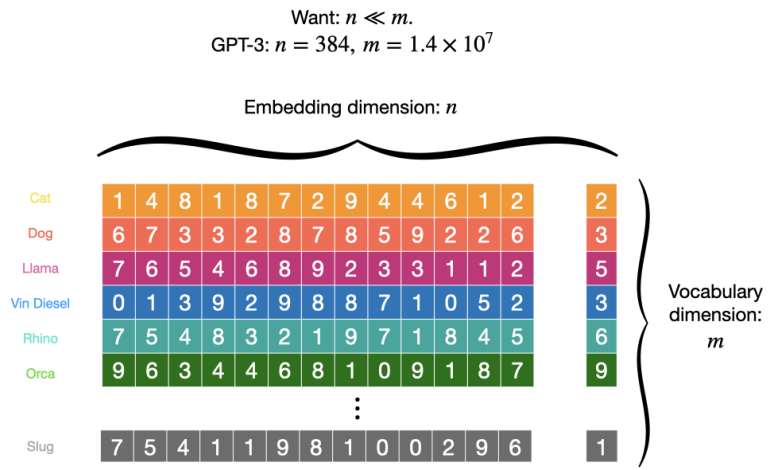
Ordering in language is grammar dependent, less clear cut order!

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 $\approx$ 175 billion)



Case Study: NLP Augmentations

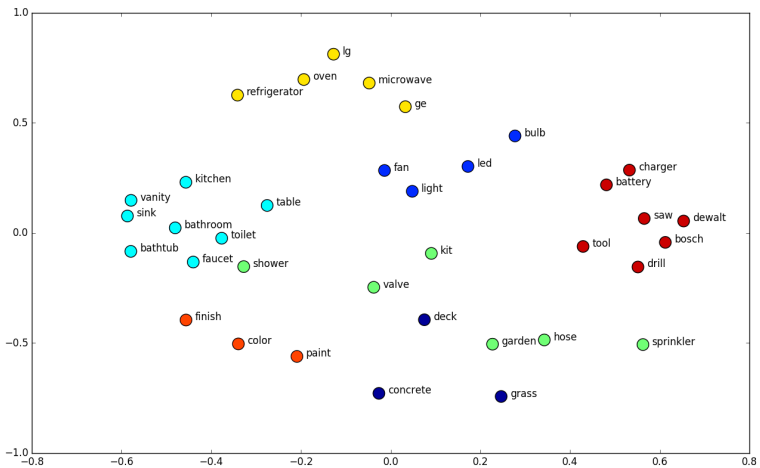
Careful Design to Preserve Grammar

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:* Re-scale images to match network input size
- *More advanced:* Convolution works on different image sizes.

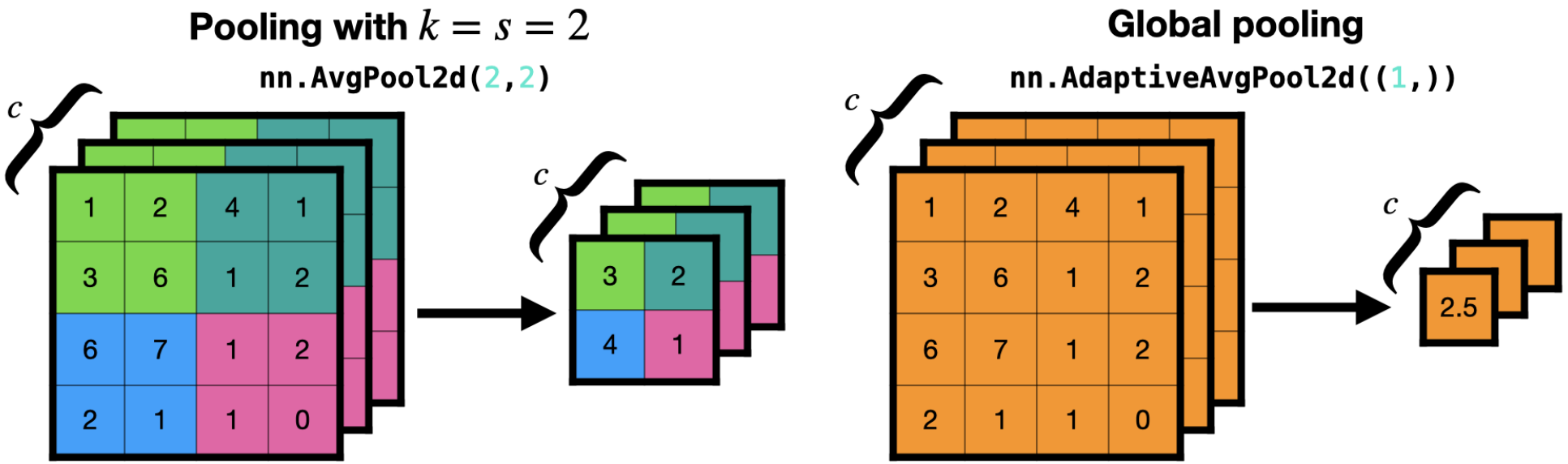
$$d' = \frac{d - k - 2p}{s} + 1$$

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

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

Pooling Operations in CNNs

Pooling with Kernel vs. Global Pooling



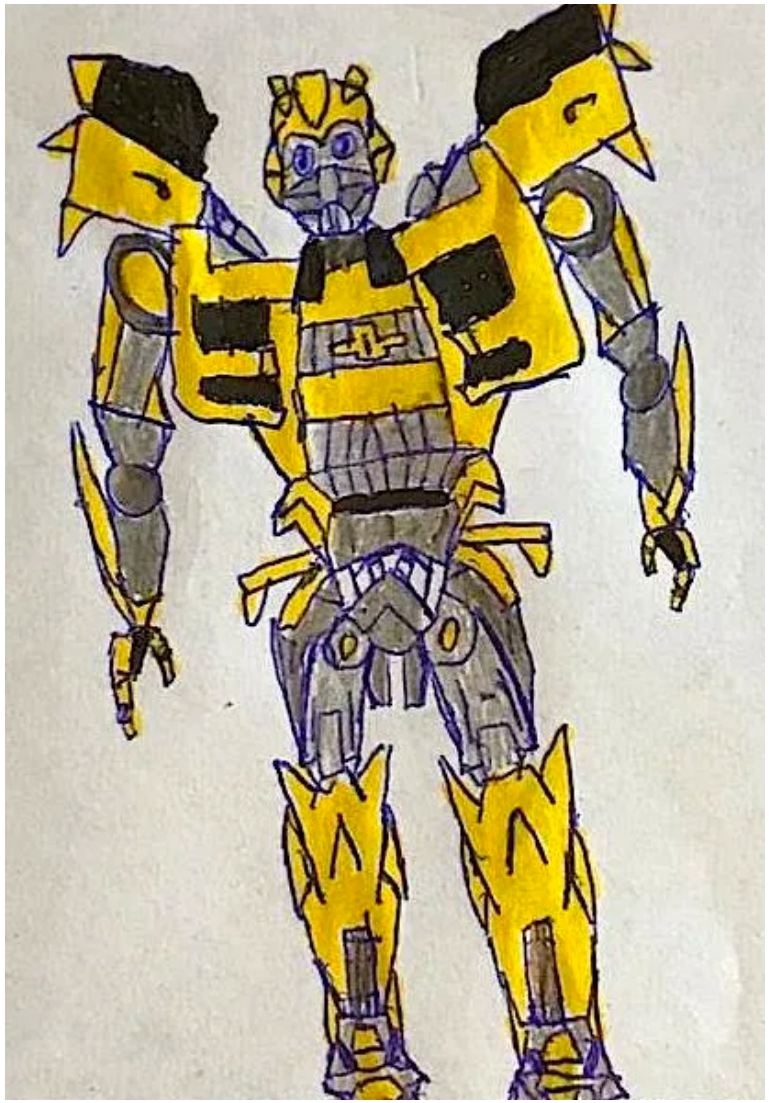
Global pooling: generally applied before the last **fully connected layers**.

Size Invariance

What about vision transformers (ViTs)?

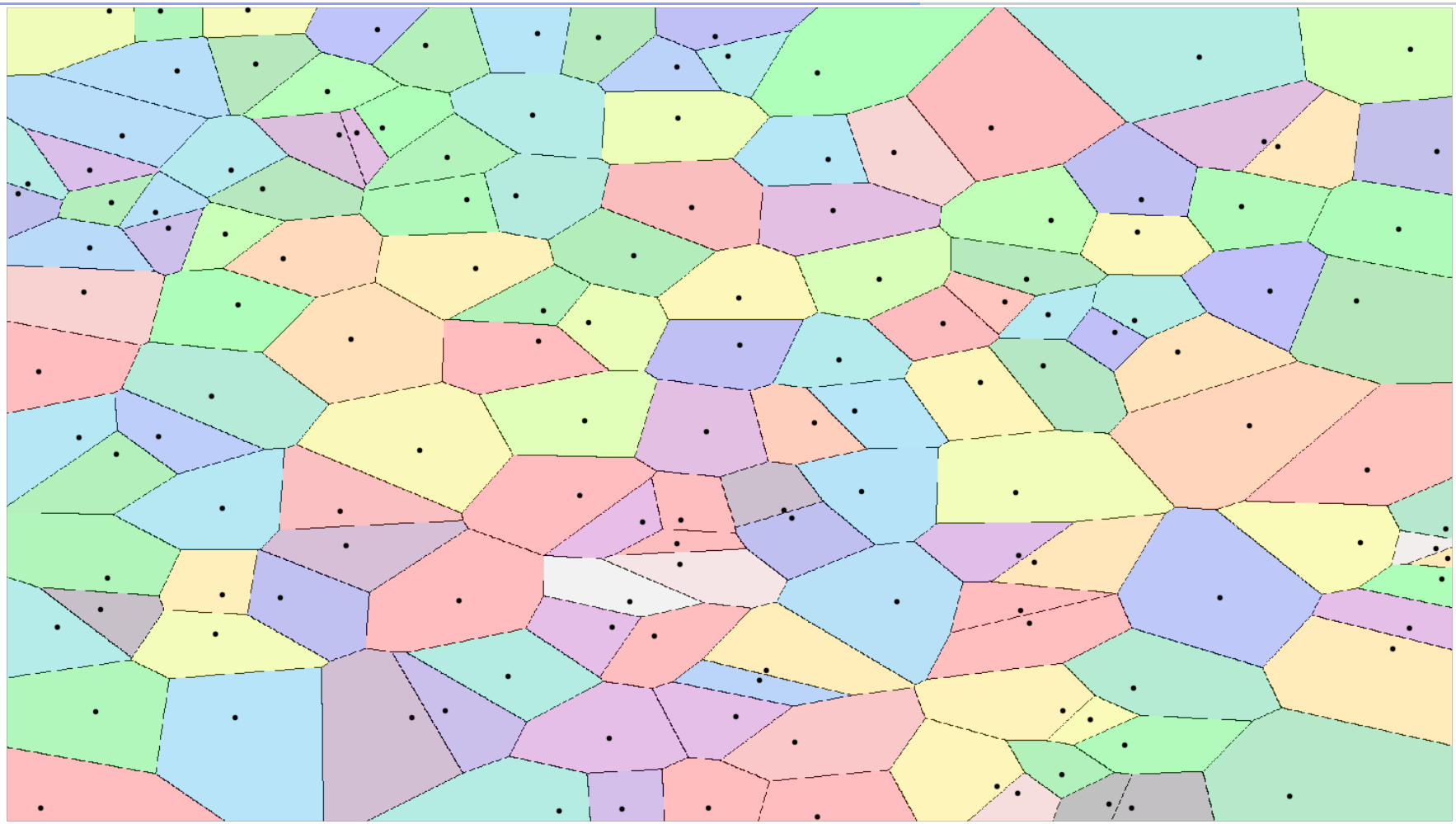
Notes on size invariance in ViTs:

- ViTs divide images 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**.



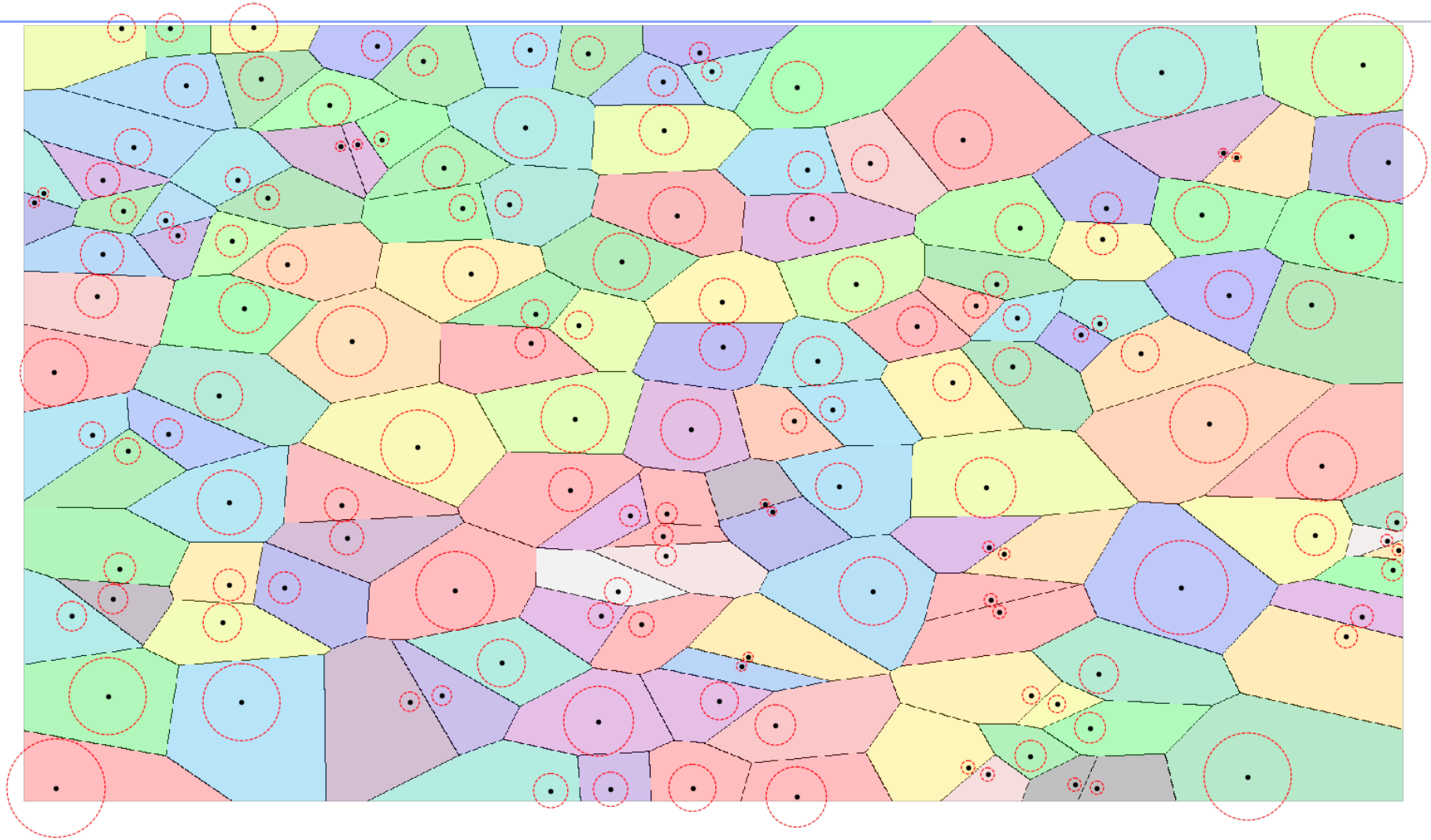
Birds Eye View

What is the effect of data augmentation?



Birds Eye View

What is the effect of data augmentation?



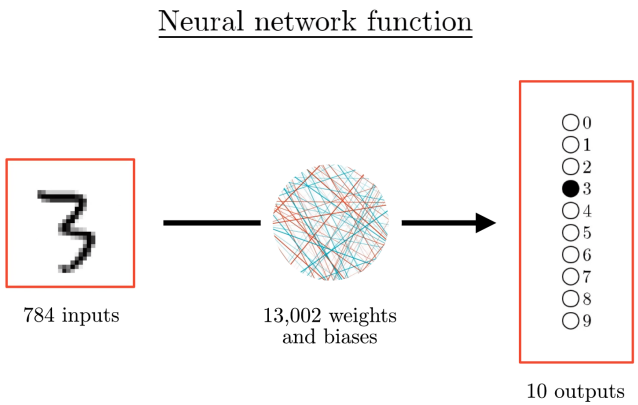

Intermission
You look like you could use a snack!
Mr. Pops
We'll meet you in the lobby.

Weight Initialization

Getting a head start

Good weight initialisation accelerates convergence and learning.

- **Identity**: Initialise weights such that $W_\theta x = Ix = x$.
- **Orthogonal**: Sample orthogonal matrix.
- **Random**: Sample the parameters.
- **Pre-trained**: Use weights from another neural network.



Identity Initialization
The benefits of doing nothing

Pros:

- Simple (Occams Razor).
- No bias → better generalization.
- No initial correlation → forces network to learn correlations between features.
- Unit gradients → optimal gradient propagation.
  - Dynamical Isometry: $\|\mathcal{J}_{W_\theta}\|_2 = 1$.
  - Jacobian of the weights has unit norm.

Cons:

- Off-diagonal zeros have no gradient signal.
  - Fix: add tiny values $\epsilon \ll 1$.
- Mostly useful in deeper layers.
- Ignores inputs for layers with fewer output dimensions.
- Can take longer to converge.

Orthogonal Initialization

No correlation intended...

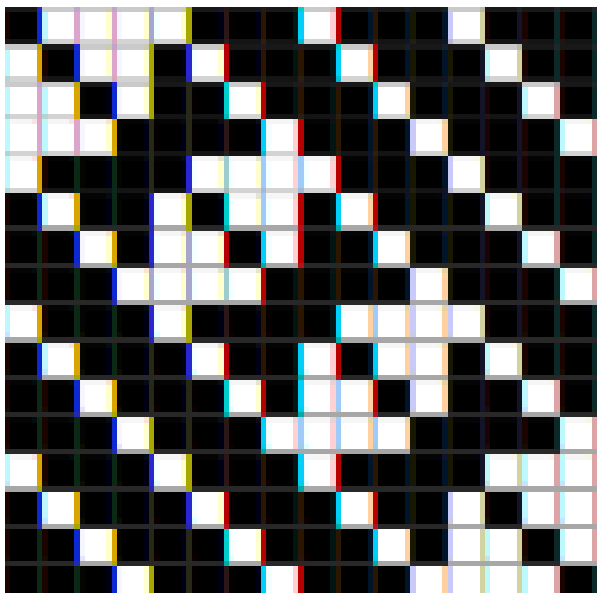
Orthogonal matrix definition: $A^T A = A A^T = I$

Pros:

- Similar properties to 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 Initialization

The Vanilla Alternative

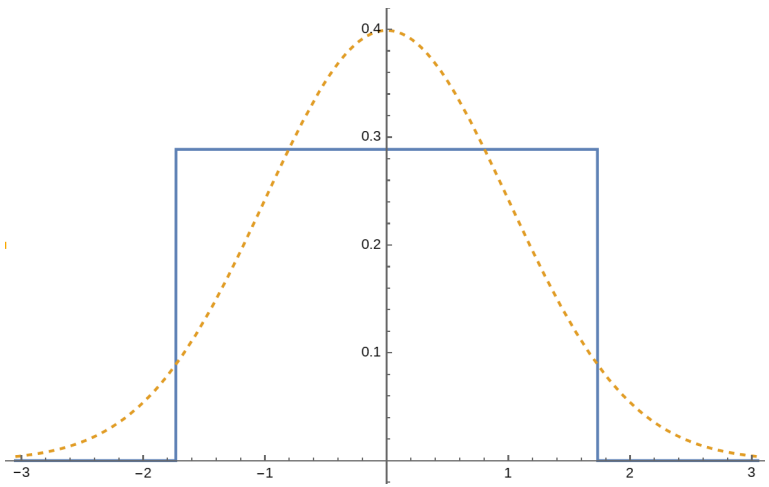
Most common method in practical use.

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.

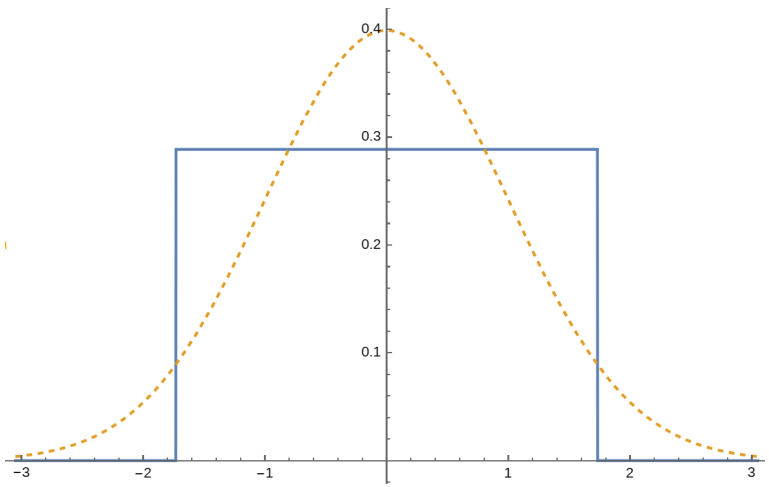
What does non-informative imply?



Random Initialization

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 weight distributions with SGD!



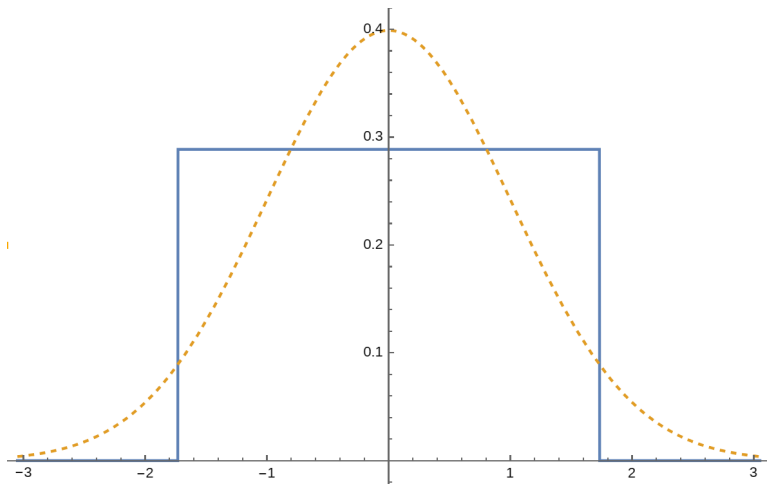
Random Initialization

More good reasons to be random...

Asymptotic Orthogonality

- By LLN* the expected inner product of two zero-mean vectors with sym. iid.** entries is asymptotically zero.
- This means large randomly initialised zero-mean matrices are approximately row-orthogonal, $A^T A \approx \text{diag}(a)$.***.

What does this tell us?



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

** 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 $(a_i = 0) \forall (i \in \mathbb{N}_+)$

Random Initialization

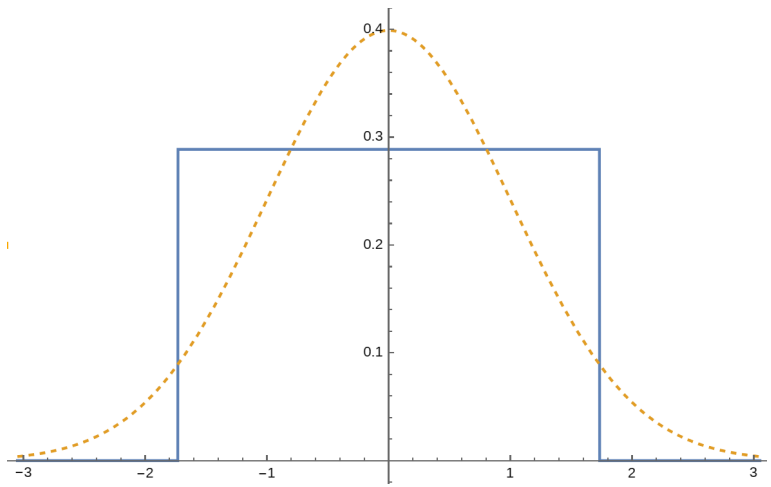
More good reasons to be random...

Asymptotic Orthogonality

- By LLN* the expected inner product of two zero-mean vectors with sym. iid.** entries is asymptotically zero.
- This means large randomly initialised zero-mean matrices are approximately row-orthogonal, $A^T A \approx \text{diag}(a)$.***.

What does this tell us?

Many of the nice properties from previous slides follow from this!



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

** 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 Normalization

Achieving full unity

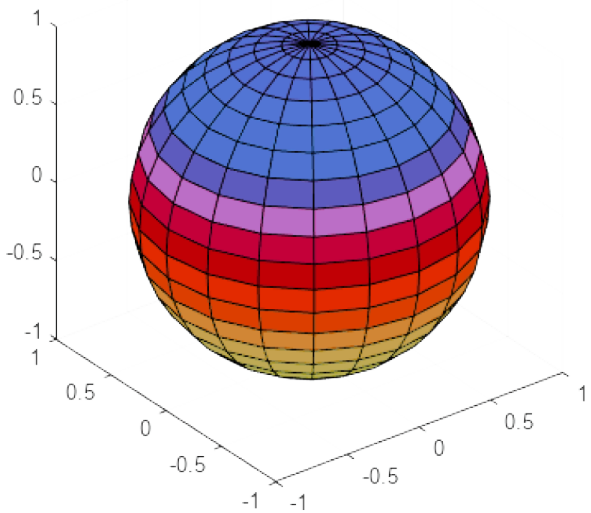
Quick method for unit norm weights:

- 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):
    # Note: 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_)
```



Standardization

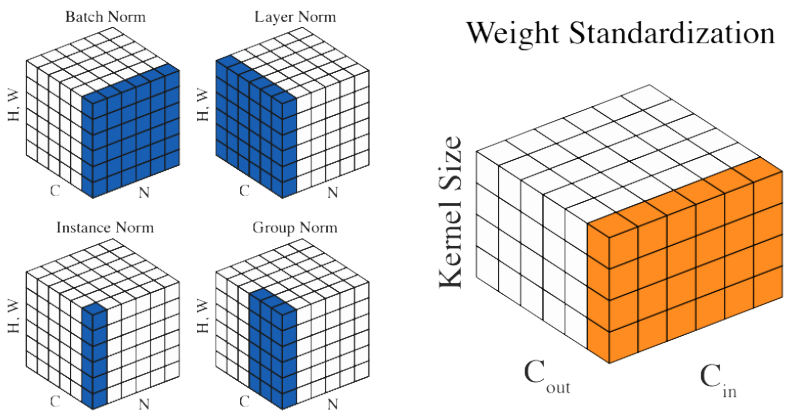
A bog standard trick...

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 $\hat{\sigma} = \sqrt{n^{-1} \sum_{j=1}^{n} (\hat{W}_\theta - \hat{\mu})}$

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

Why is it a poor choice for batch norm?



Initialization Schemes

Two commonly used initializations

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

Xavier Glorot (Uniform or Normal)

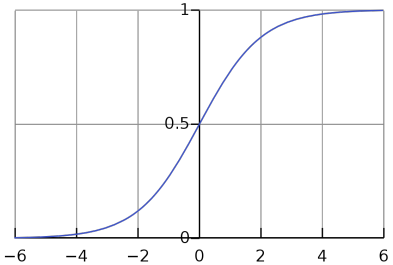
- Assume $x \perp\!\!\!\perp W_\theta$ and $\sigma = \text{Sigmoidal}$

Kaiming He (Uniform or Normal)

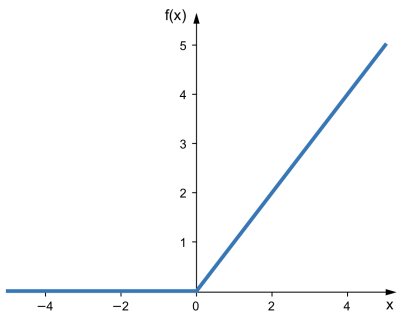
- Assume $x \perp\!\!\!\perp W_\theta$ and $\sigma = \text{Rectified}$

For a full derivation of these results, check out the notes by Alex Binder.







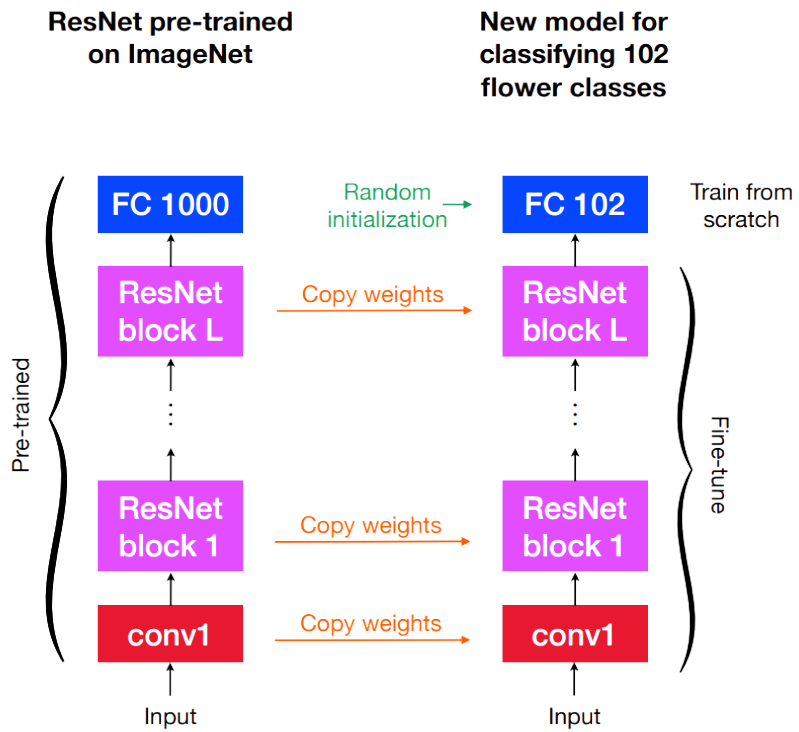


$x \perp\!\!\!\perp W_\theta :=$ x is independent of W_θ.

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.



Alternative training methods

Think-Pair-Share

Questions:

- Can we use pre-trained weights to train a different architecture (different weights and layer structure)?
- Do we always need to use annotated labels to train a network?

Instructions:

- Think individually for 2 min
- Discuss in pairs for 3 min
- Share in plenum!

Student-Teacher Learning

Knowledge distillation/model compression

Train a smaller "student" model from a large/complex "teacher" model

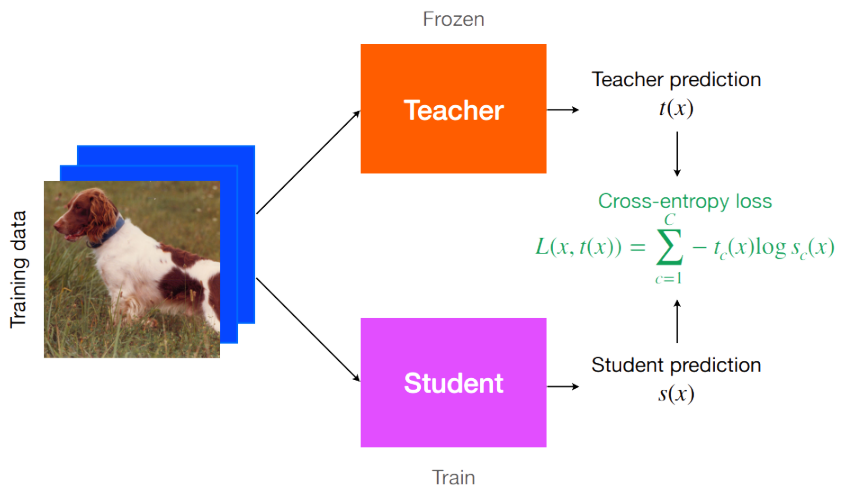
Idea: Train the student model by using the softmax output of the teacher model as soft labels

Cross-entropy loss using teacher output $t(x)$ as ground truth labels

Prioritizes 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 sampler: The teacher is very certain, i.e. $t(x)$ is close to a one-hot label. Then the cross entropy loss is higher.

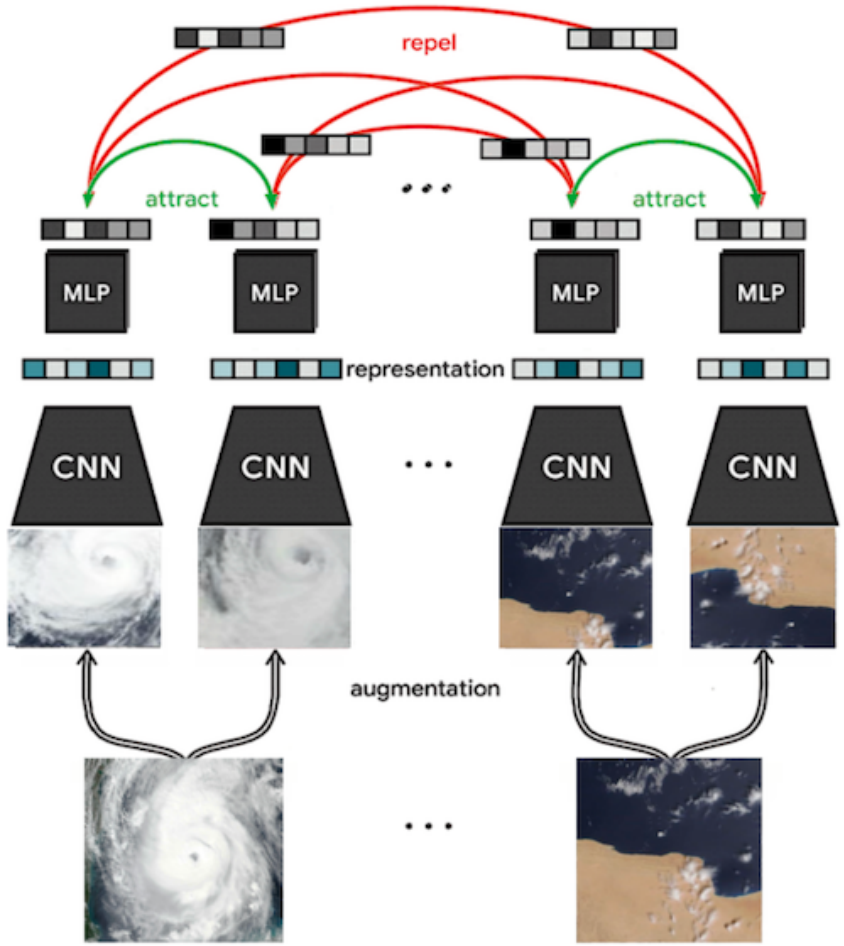
Why? For situations where a large model is too expensive/slow to run e.g. on smaller devices.



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 network to recognize augmented instances of the same image.
- **Positive samples:** Augmentations of the same image should have high similarity scores
- **Negative samples:** Augmentations of different images should have low similarity scores



Contrastive Learning

SimCLR version

- 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: $\text{sim}(u, v) = \frac{u^T v}{||u||||v||}$ equivalent to the cosine of the angle between two vectors
- **Contrastive loss** for positive pair $\tilde{x}_i, \tilde{x}_j$:

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

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

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

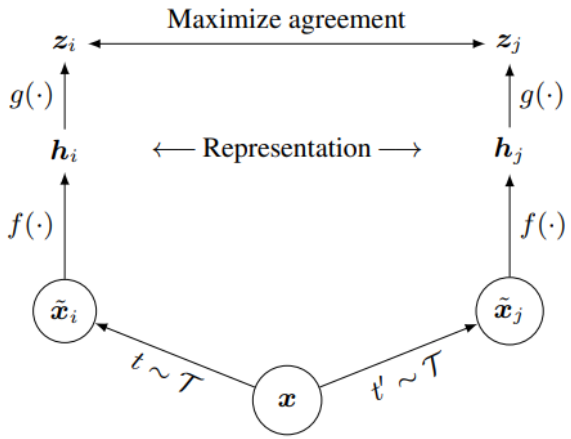


Figure 2. A simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder 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.

Question:

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

Instructions:

- Think individually for 2 min
- Discuss in pairs for 3 min
- Share in plenum!