Training [Deep] Neural Networks

Machine Learning

(Largely based on slides from Fei-Fei Li & Justin Johnson & Serena Yeung)

Prof. Sandra Avila

Institute of Computing (IC/Unicamp)

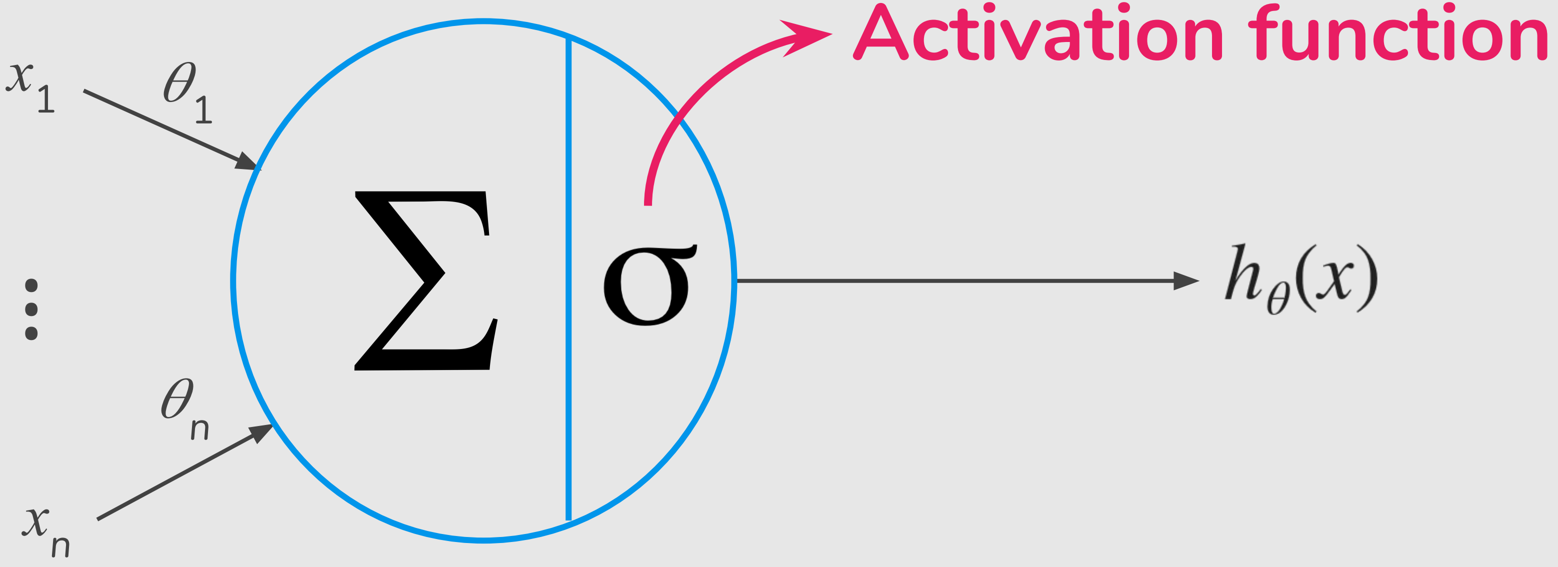
MC886, October 23, 2019

Today's Agenda

- Activation Functions
- Data Preprocessing
- Weight Initialization
- Batch Normalization
- Optimizers
- Regularization
- Transfer learning / fine-tuning

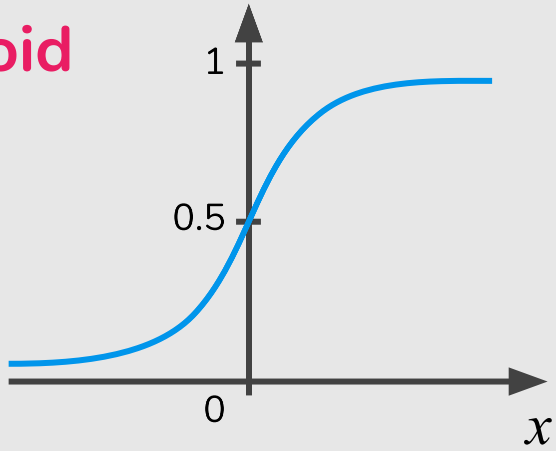
Activation Functions

Recall from last time ...



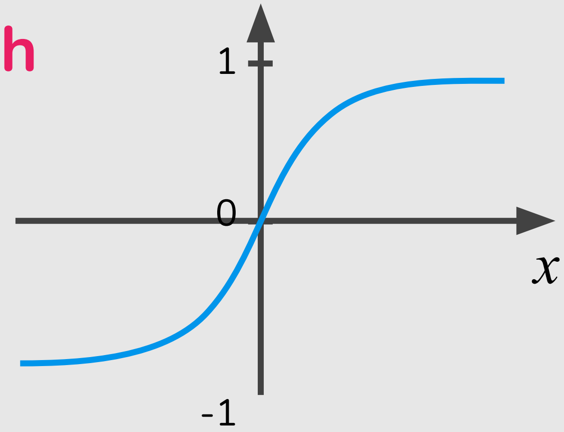
Activation Functions

Sigmoid



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Tanh

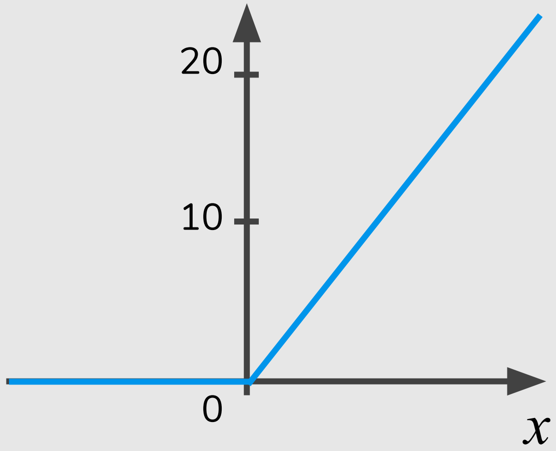


$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Hyperbolic Tangent

Activation Functions

ReLU



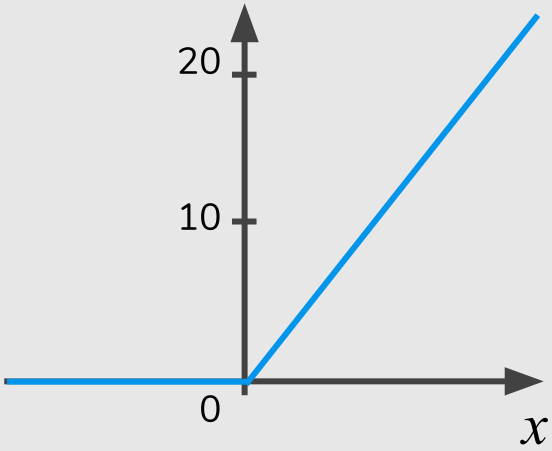
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

$$\mathrm{ReLU}(x) = max(0,x)$$

Rectified Linear Unit (ReLU)

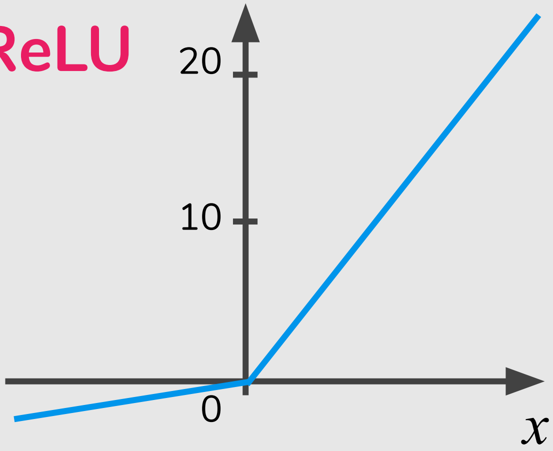
Activation Functions

ReLU



$$\text{ReLU}(x) = max(0,x)$$

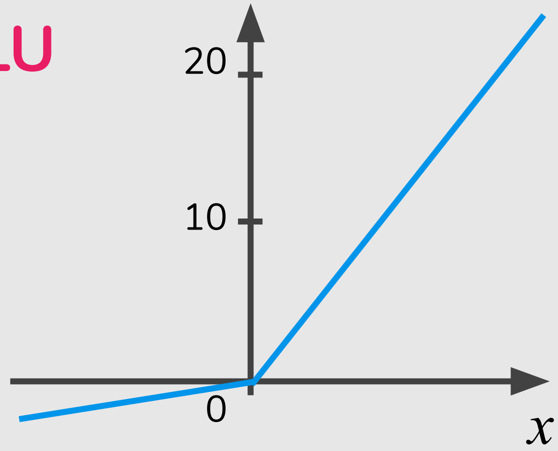
Leaky ReLU



$$\text{Leaky ReLU}(x) = \begin{cases} x \text{ if } x > 0 \\ 0.01x \end{cases}$$

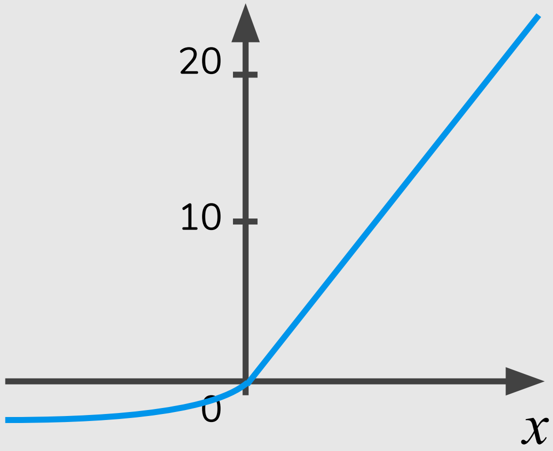
Activation Functions

PReLU



$$\mathrm{PReLU}(x) = \begin{cases} x \text{ if } x > 0 \\ \cancel{0.01x} \quad \alpha\, x \end{cases}$$

ELU



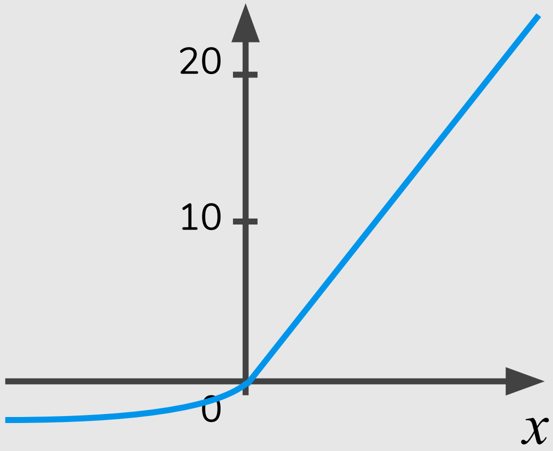
$$\mathrm{ELU}(x) = \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha\left(e^{-x} - 1\right) \end{cases}$$

Exponential LU

Activation Functions

- Combine the good parts of ReLU and leaky ReLU
- It doesn't have the dying ReLU problem
- It saturates for large negative values, allowing them to be essentially inactive

ELU



$$\mathrm{ELU}(x) = \begin{cases} x \text{ if } x \geqslant 0 \\ \alpha(e^{-x} - 1) \end{cases}$$

Exponential LU

In Practice: Activation Functions

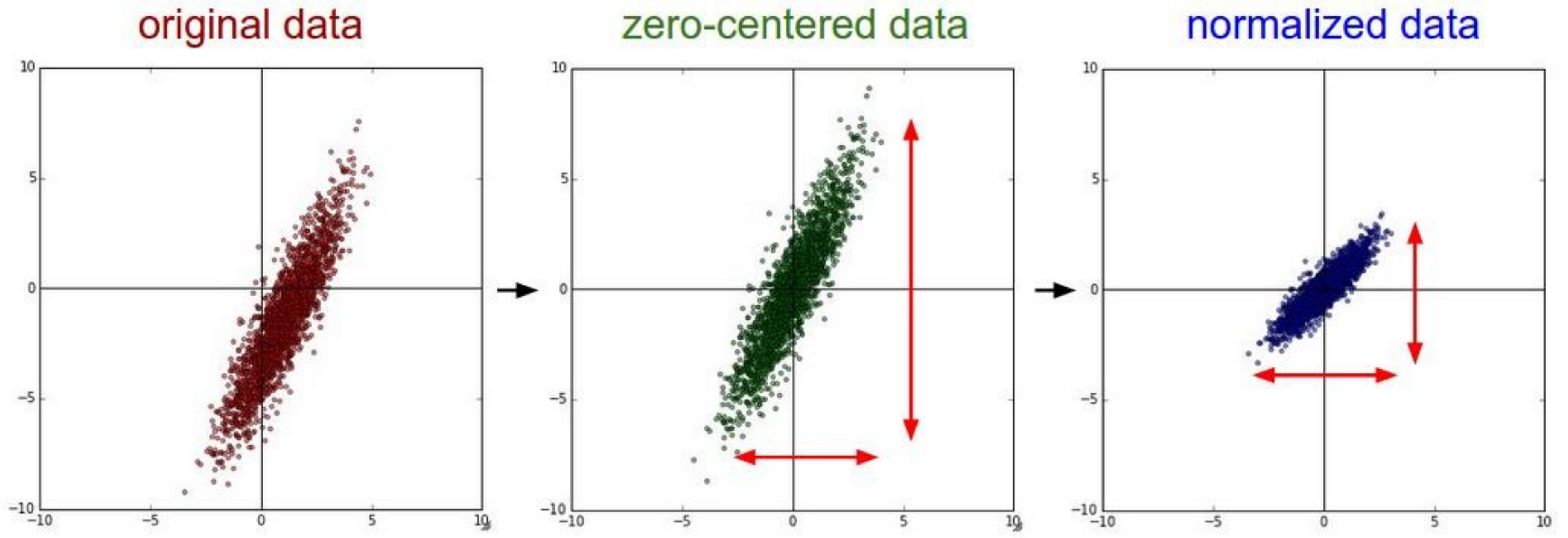
- Use **ReLU**
- Try out Leaky ReLU / ELU
- Try out tanh but don’t expect much
- **Don’t use sigmoid**

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Data Preprocessing

Data Preprocessing



Credit: http://cs231n.github.io/neural-networks-2/

Data Preprocessing

consider CIFAR-10 example with [32,32,3] images

- **Subtract the mean image** (e.g., AlexNet)
  (mean image = [32,32,3] array)
- **Subtract per-channel mean** (e.g., VGG)
  (mean along each channel = 3 numbers)
- **Subtract per-channel mean and Divide by per-channel std** (e.g., ResNet)
  (mean along each channel = 3 numbers)

Weight Initialization

Weight Initialization

- Xavier initialization [Glorot & Bengio, 2010]: “Understanding the difficulty of training deep feedforward neural networks”, http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

```
w = np.random.randn(n)*sqrt(2.0/n)
```

Weight Initialization

- Xavier initialization [Glorot & Bengio, 2010]: "Understanding the difficulty of training deep feedforward neural networks", http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

```
w = np.random.randn(n)*sqrt(2.0/n)
```

- He initialization [He et al., 2015]: "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification" https://arxiv.org/pdf/1502.01852

Weight Initialization

- Xavier initialization [Glorot & Bengio, 2010]:

  `n` = input + output

- He initialization [He et al., 2015]:

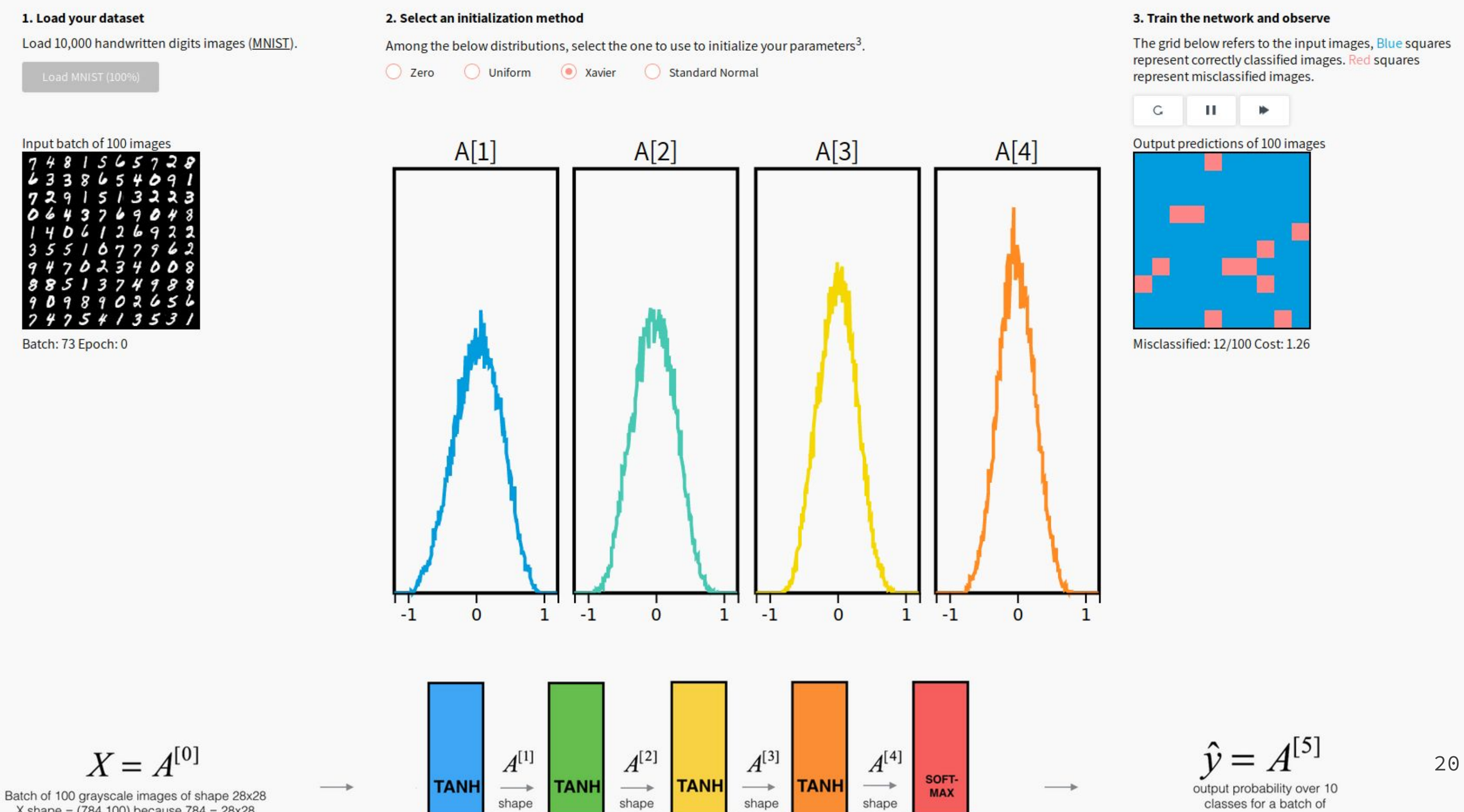
  `n` = input

```
w = np.random.randn(n)*sqrt(2.0/n)
```

Proper initialization is an active area of research…

- “Understanding the difficulty of training deep feedforward neural networks”, Glorot and Bengio, 2010
- “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks”, Saxe et al, 2013
- “Random walk initialization for training very deep feedforward networks”, Sussillo and Abbott, 2014
- “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification”, He et al., 2015
- “Data-dependent initializations of convolutional neural networks”, Krähenbühl et al., 2015
- “All you need is a good init”, Mishkin and Matas, 2015
- “Fixup initialization: Residual learning without normalization”, Zhang et al., 2019
- “The lottery ticket hypothesis: Finding sparse, trainable neural networks”, Frankle and Carbin, 2019

http://www.deeplearning.ai/ai-notes/initialization



Batch Normalization

Batch Normalization

To increase the stability of a neural network, batch normalization **normalizes the output** of a previous activation layer **by subtracting the batch mean** and **dividing by the batch standard deviation**.

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015, https://arxiv.org/pdf/1502.03167

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

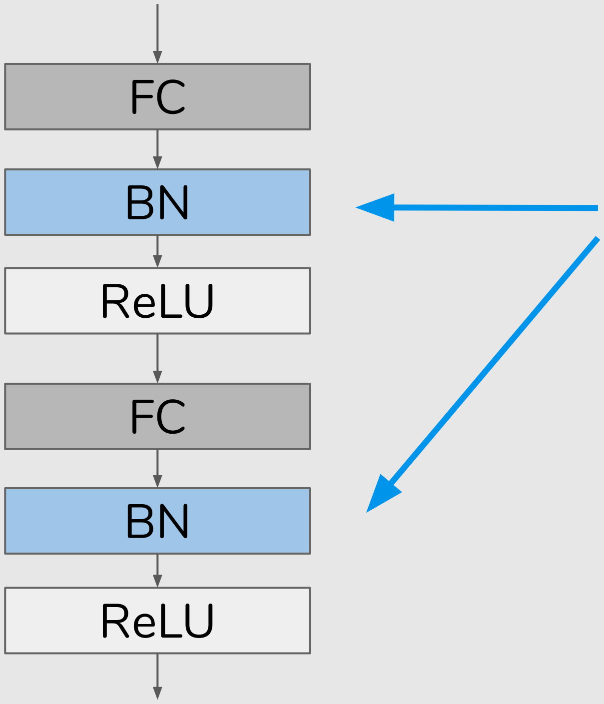
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

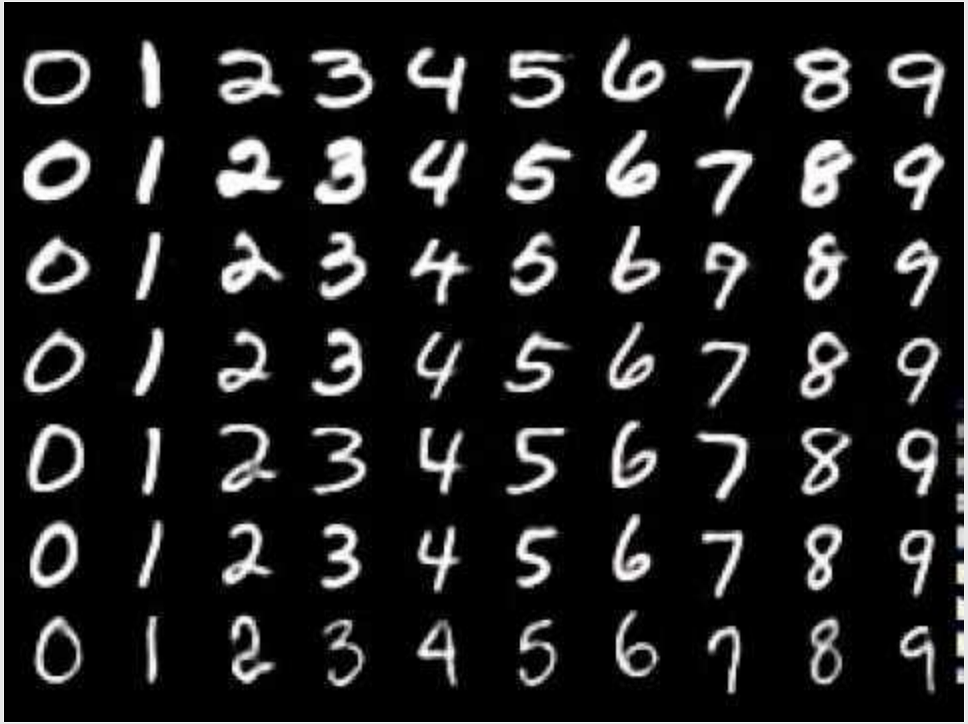
“Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, https://arxiv.org/pdf/1502.03167

Batch Normalization



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

Batch Normalization: An Example (MNIST)



http://yann.lecun.com/exdb/mnist/

Batch Normalization: An Example (MNIST)

Model **without** batch normalization:

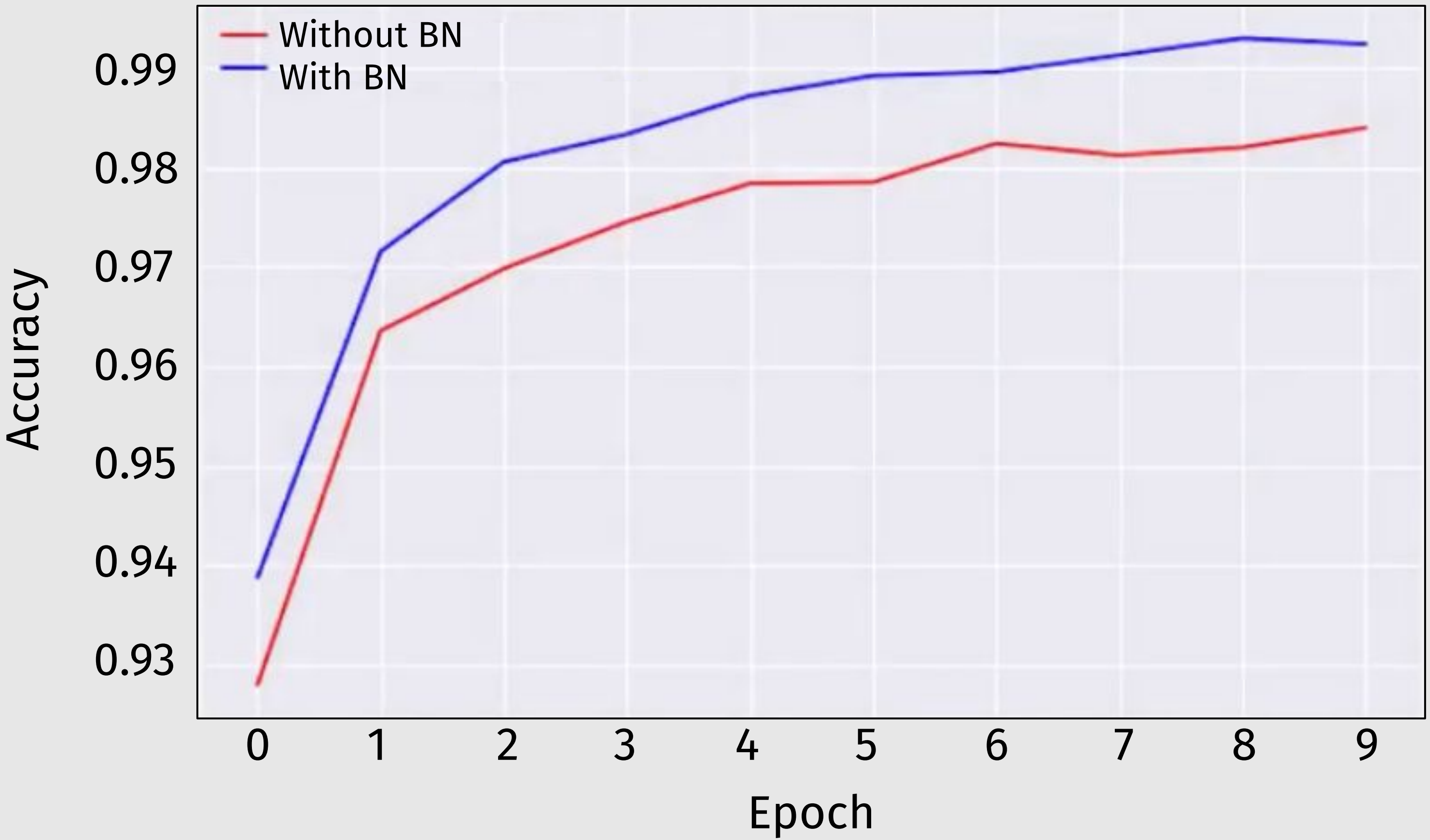
```
# Creating the model
model_without_bn = Sequential()

# Architecture
model_without_bn.add(Dense(256, activation='relu', input_shape=(784,)))
model_without_bn.add(Dense(128, activation='relu'))
model_without_bn.add(Dense(64, activation='relu'))
model_without_bn.add(Dense(10, activation='softmax'))
```

Model **with** batch normalization:

```
# Creating the model
model_without_bn = Sequential()

# Architecture
model_with_bn.add(Dense(256, use_bias=False, input_shape=(784,)))
model_with_bn.add(BatchNormalization())
model_with_bn.add(Activation('relu'))

model_with_bn.add(Dense(128, use_bias=False))
model_with_bn.add(BatchNormalization())
model_with_bn.add(Activation('relu'))

model_with_bn.add(Dense(64, use_bias=False))
model_with_bn.add(BatchNormalization())
model_with_bn.add(Activation('relu'))

model_with_bn.add(Dense(10, activation='softmax'))
```

Batch Normalization (1): An Example (MNIST)

- epochs = 10
- batch size = 128
- learning rate = 0.01
- data normalization: X/255
- weight init: glorot_uniform

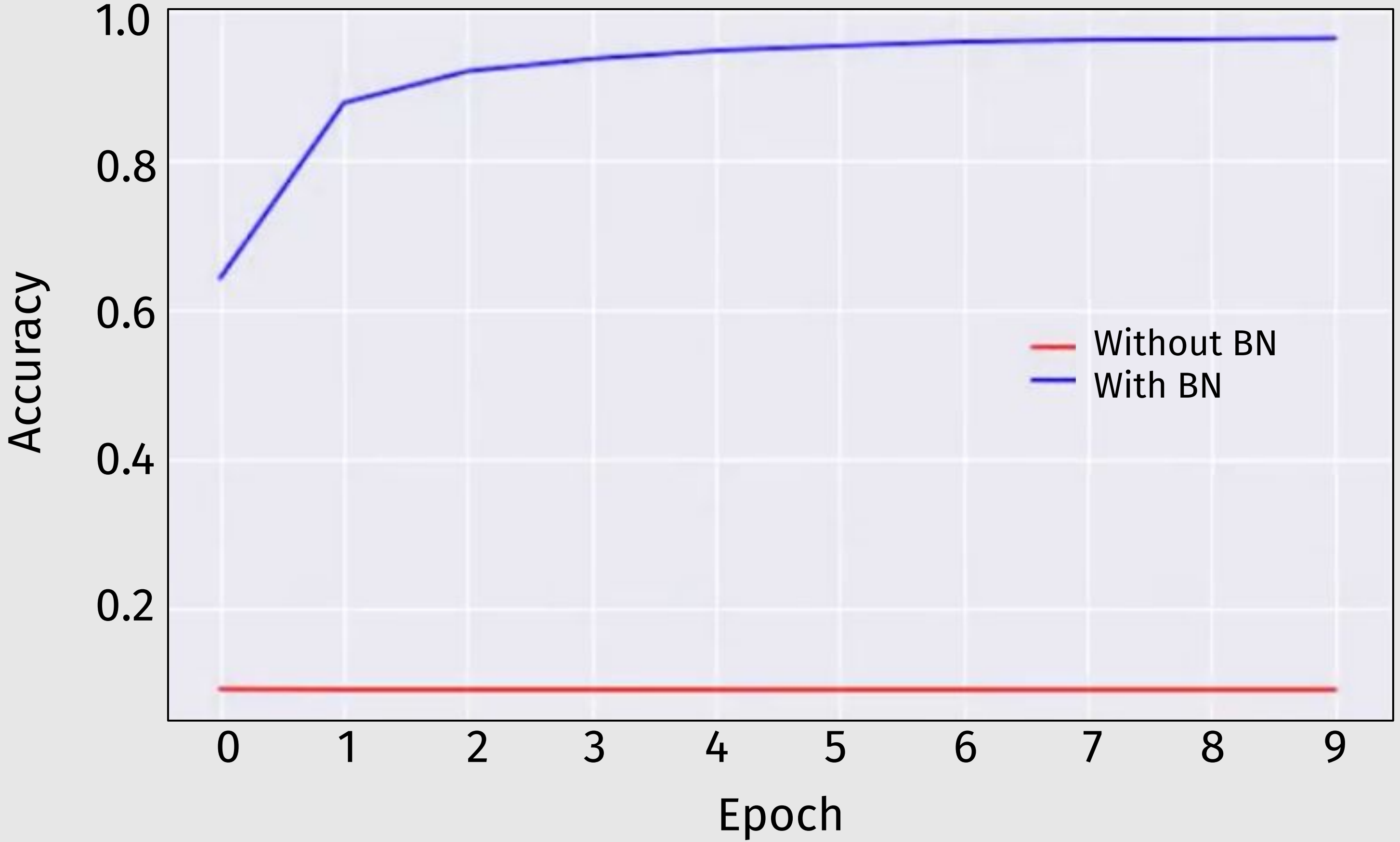


Without Batch Normalization test-acc: 0.966

With Batch Normalization test-acc: 0.974

Batch Normalization (2): An Example (MNIST)

- epochs = 10
- batch size = 128 $\rightarrow$ 1024
- learning rate = 0.01 $\rightarrow$ 1
- data normalization: X/255
- weight init: glorot_uniform

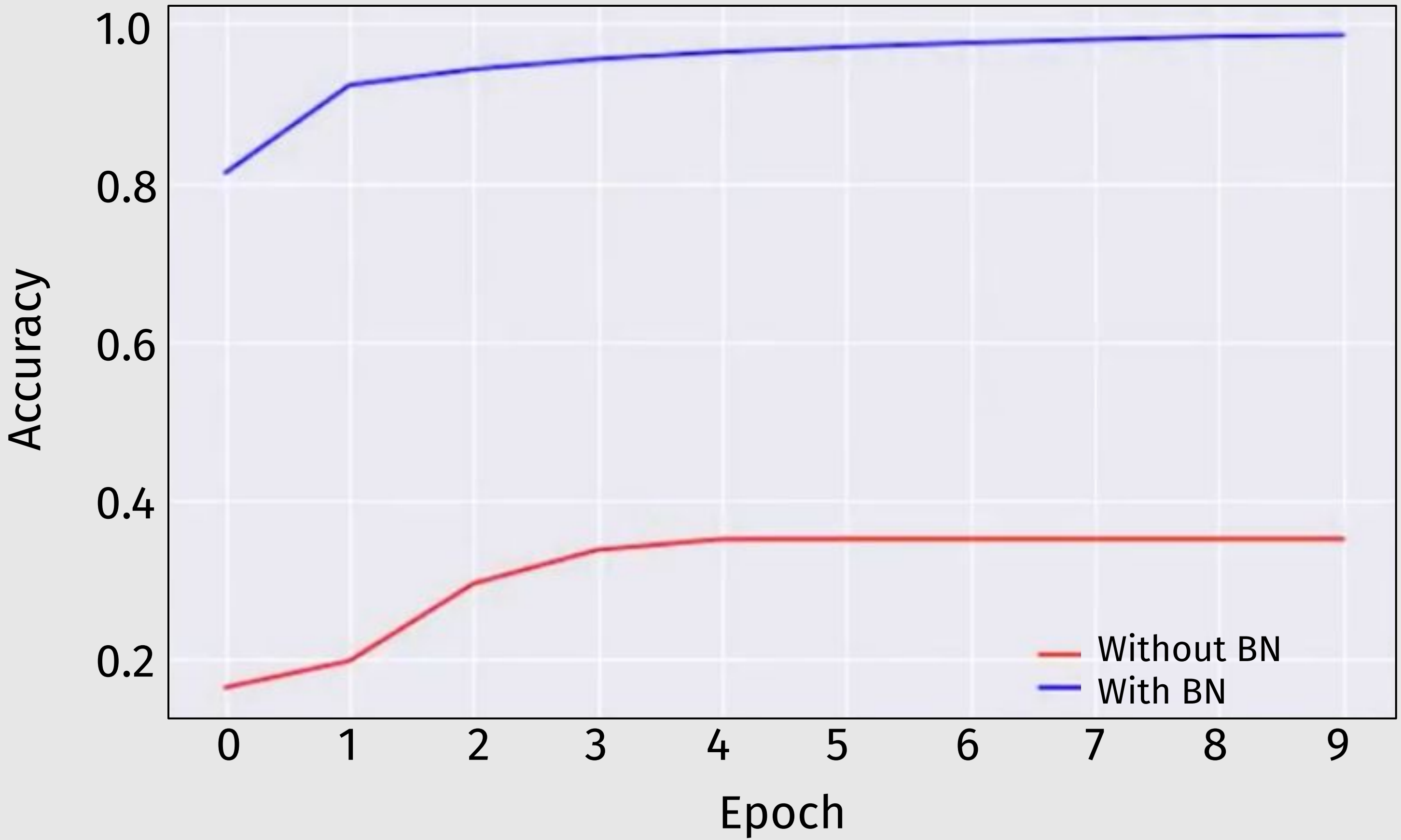


Without Batch Normalization test-acc: 0.089

With Batch Normalization test-acc: 0.950

Batch Normalization (3): An Example (MNIST)

- epochs = 10
- batch size = 128
- learning rate = 0.01
- data normalization: X/255
- weight init: glorot_uniform $\longrightarrow$ RandomUniform(minval=-5, maximal=5)

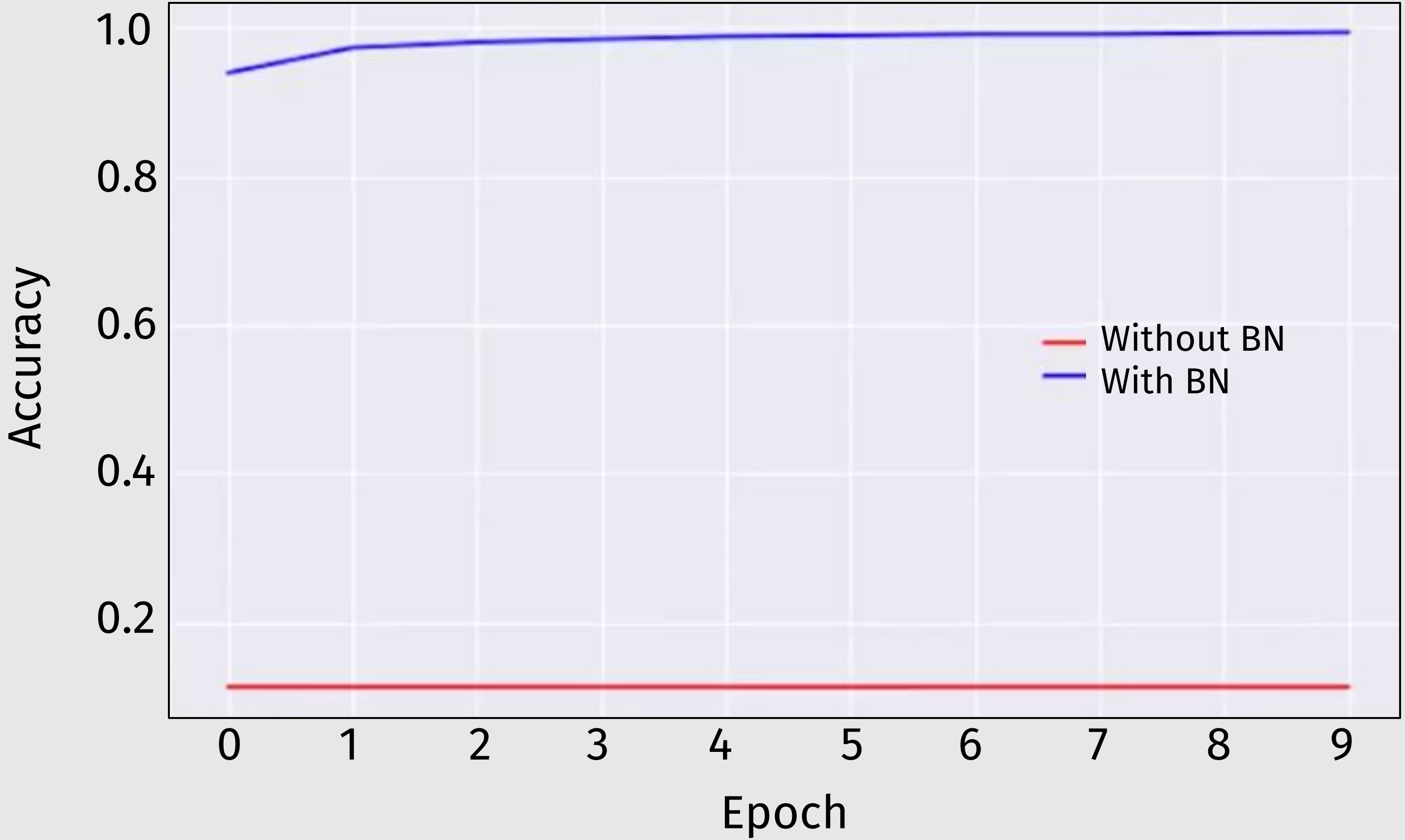


Without Batch Normalization test-acc: 0.352

With Batch Normalization test-acc: 0.966

Batch Normalization (4): An Example (MNIST)

- epochs = 10
- batch size = 128
- learning rate = 0.01
- data normalization: X/255 → no norm
- weight init: glorot_uniform



Without Batch Normalization test-acc: 0.113

With Batch Normalization test-acc: 0.977

Batch Normalization

- Makes deep networks much easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training

Batch Normalization

“Fitting Batch Norm Into Neural Networks”, deeplearning.ai
https://youtu.be/em6dfRxYkYU

“How does Batch Normalization Help Optimization?”, Ilyas et al., NeurIPS 2018, http://gradientscience.org/batchnorm/

“The Batch Normalization layer of Keras is broken”, http://blog.datumbox.com/the-batch-normalization-layer-of-keras-is-broken/

Normalization

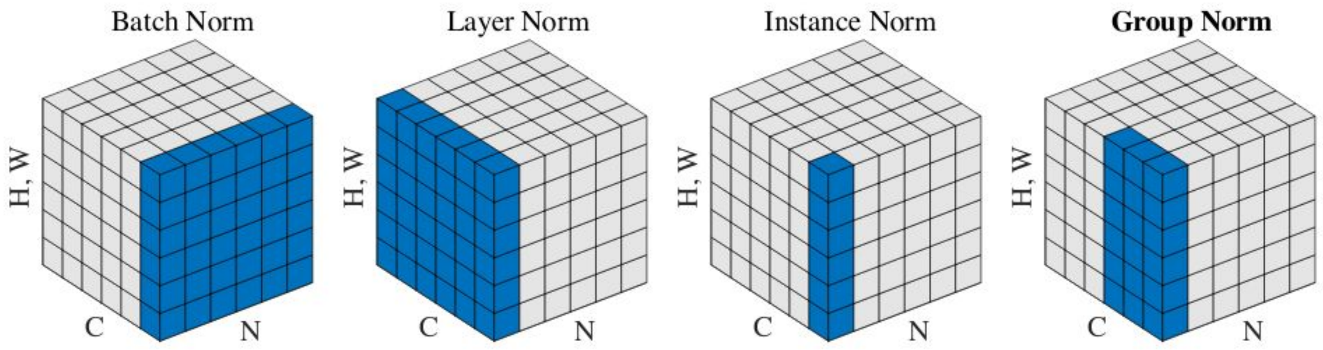
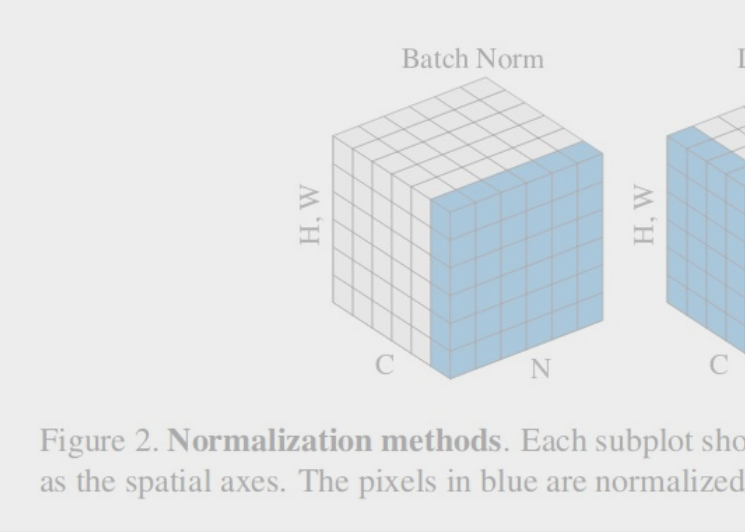
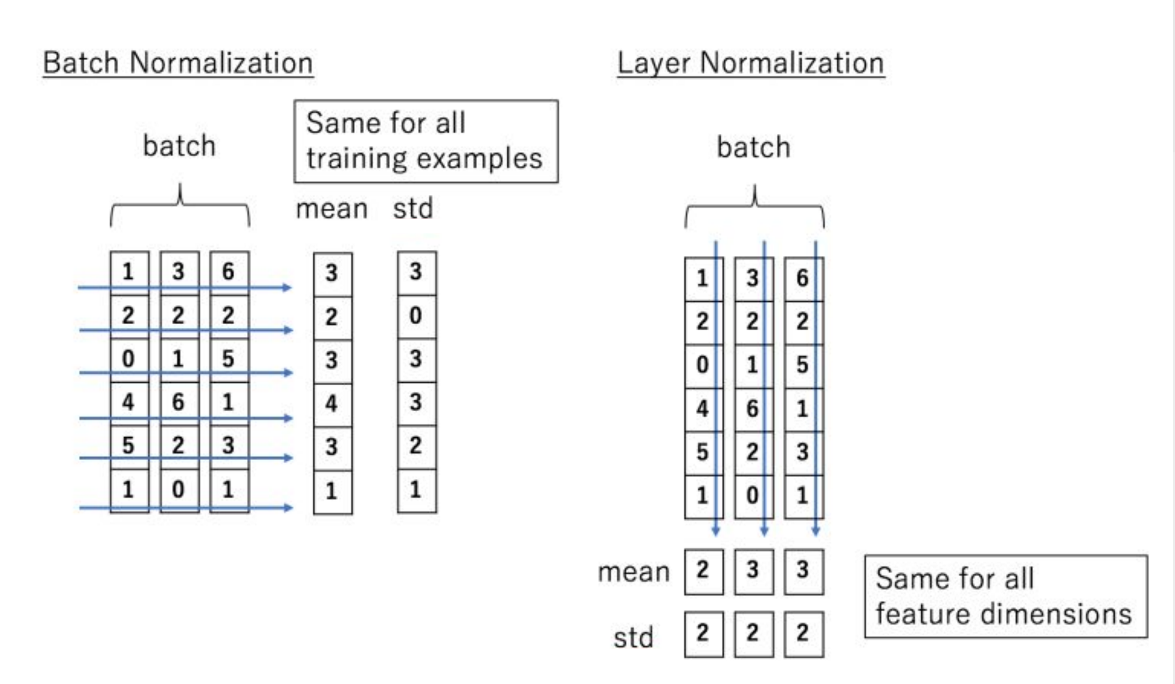


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

"Layer normalization", arXiv 2016, https://arxiv.org/pdf/1607.06450.pdf
"Improved texture networks: ...", CVPR 2017, https://arxiv.org/pdf/1701.02096.pdf
"Group normalization", ECCV 2018, https://arxiv.org/pdf/1803.08494.pdf

Normalization





“Layer normalization”, arXiv 2016, https://arxiv.org/pdf/1607.06450.pdf

“Improved texture networks: …”, CVPR 2017, https://arxiv.org/pdf/1701.02096.pdf

“Group normalization”, ECCV 2018, https://arxiv.org/pdf/1803.08494.pdf

Proper normalization is an active area of research...

- “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, Ioffe and Szegedy, 2015
- “Layer normalization”, Ba, Kiros, Hinton, 2016
- “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks”, Salimans and Kingma, 2016
- “Improved Texture Networks: Maximizing Quality and Diversity inFeed-forward Stylization and Texture Synthesis”, Ulyanov, Vedaldi and Vedaldi, 2017
- “Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models”, Ioffe, 2017
- “Group normalization”, Wu and He, 2018
- “Do Normalization Layers in a Deep ConvNet Really Need to Be Distinct?”, Luo, Peng, Ren, and Zhang, 2018
- “Batch-Instance Normalization for Adaptively Style-Invariant Neural Networks”, Nam and Kim, 2019

Normalization

An Overview of Normalization Methods in Deep Learning

https://mlexplained.com/2018/11/30/an-overview-of-normalization-methods-in-deep-learning/ Nov. 2018

Today's Agenda

- Activation Functions (use **ReLU**)
- Data Preprocessing (images: **subtract mean**)
- Weight Initialization (use **Xavier/He init**)
- ~~Batch~~ Normalization (use)
- Optimizers
- Regularization
- Transfer learning / fine-tuning

Optimizers

Optimizers

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

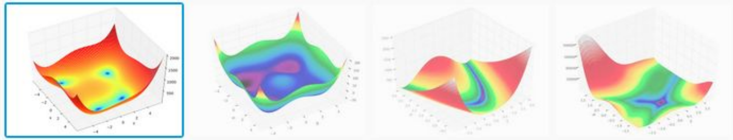
- Momentum
- Nesterov
- Adagrad
- Adadelta
- RMSprop

- Adam
- AdaMax
- Nadam
- AMSGrad
- RAdam

In this visualization, you can compare optimizers applied to different cost functions and initialization. For a given cost landscape (**1**) and initialization (**2**), you can choose optimizers, their learning rate and decay (**3**). Then, press the play button to see the optimization process (**4**). There's no explicit model, but you can assume that finding the cost function's minimum is equivalent to finding the best model for your task.

1. Choose a cost landscape

Select an artificial landscape $\mathcal{J}(w_1, w_2)$.



2. Choose initial parameters

On the cost landscape graph, drag the red dot to choose initial parameter values and thus the initial value of the cost.

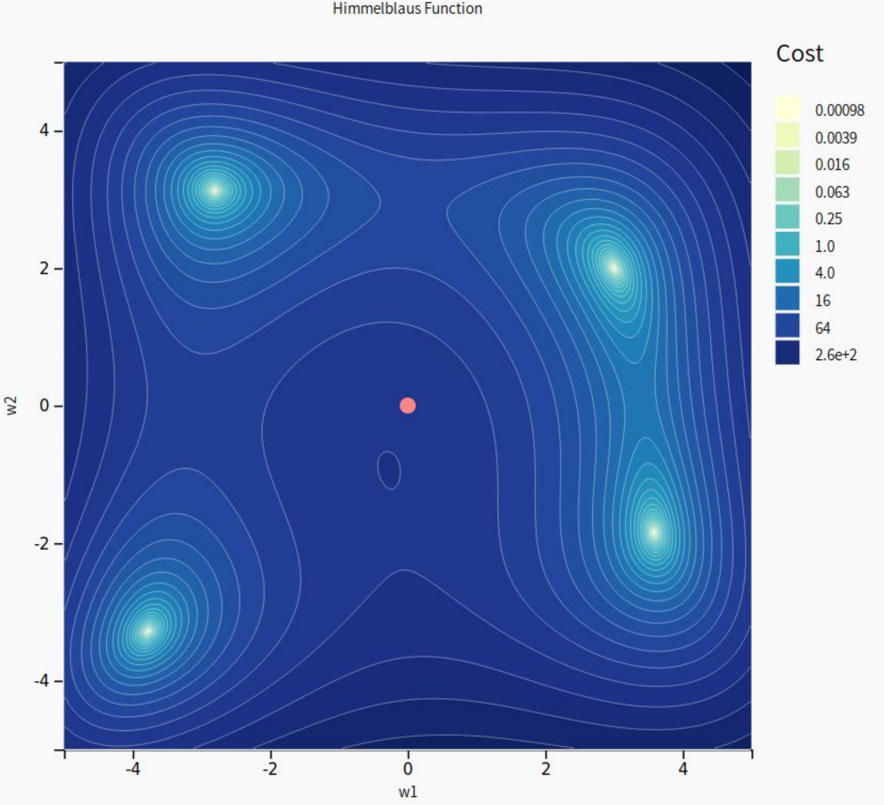
3. Choose an optimizer

Select the optimizer(s) and hyperparameters.

Optimizer	Learning Rate	Learning Rate Decay
☑ Gradient Descent	0.001	0
☑ Momentum	0.001	0
☑ RMSprop	0.001	0
☑ Adam	0.001	0

4. Optimize the cost function

This 2D plot describes the cost function's value for different values of the two parameters (w_1, w_2). The lighter the color, the smaller the cost value.



The graph below shows how the value of the cost changes through successive epochs for each optimizer.

Optimizer

In Practice: Optimizers

- **Adam** is a good default choice in many cases; it often works ok even with constant learning rate
- **SGD+Momentum** can outperform Adam but may require more tuning of LR and schedule

Regularization

Regularization

- Dropout
- Batch Normalization
- Data Augmentation
- DropConnect
- Fractional Max Pooling
- Stochastic Depth
- Cutout
- Mixup

In Practice: Regularization

- Consider **dropout for large fully-connected layers**
- **~~Batch~~ normalization and data augmentation** almost always a good idea
- Try **cutout and mixup** especially for small classification datasets

Transfer Learning

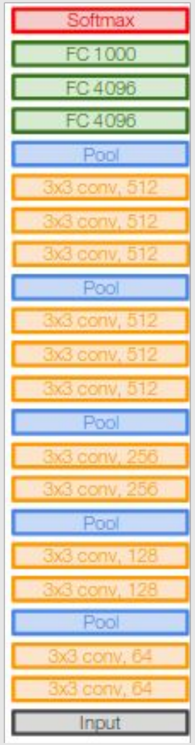
Transfer Learning

"You need a lot of a data if you want to train/use CNNs"

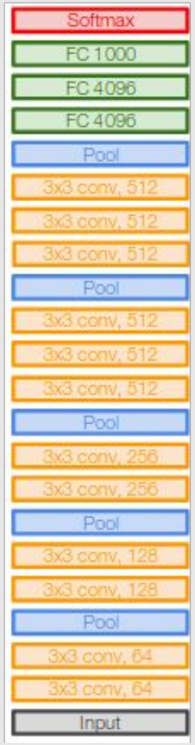
Transfer Learning

“You need a ... data if you want to t... e CNNs”

Transfer Learning with CNNs

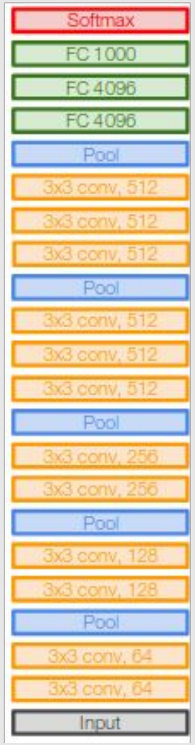


Transfer Learning with CNNs

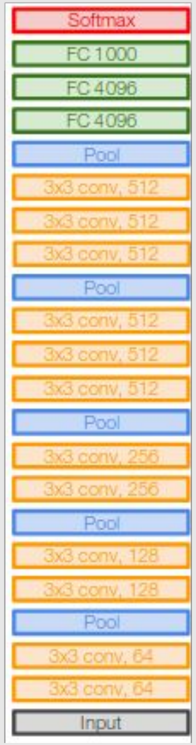


Train on ImageNet
(or large dataset)

Transfer Learning with CNNs

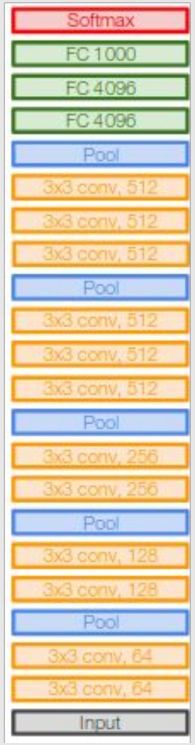


Train on ImageNet

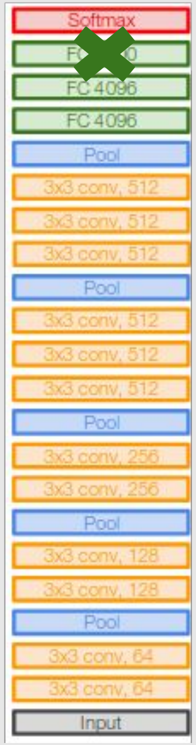


Small dataset (C classes):
Transfer learning with fine-tuning

Transfer Learning with CNNs

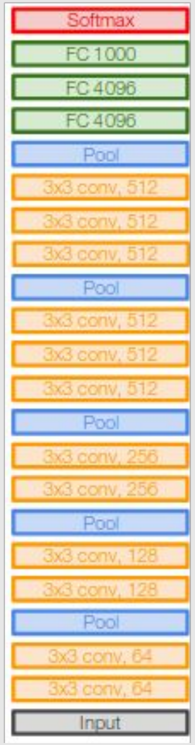


Train on ImageNet

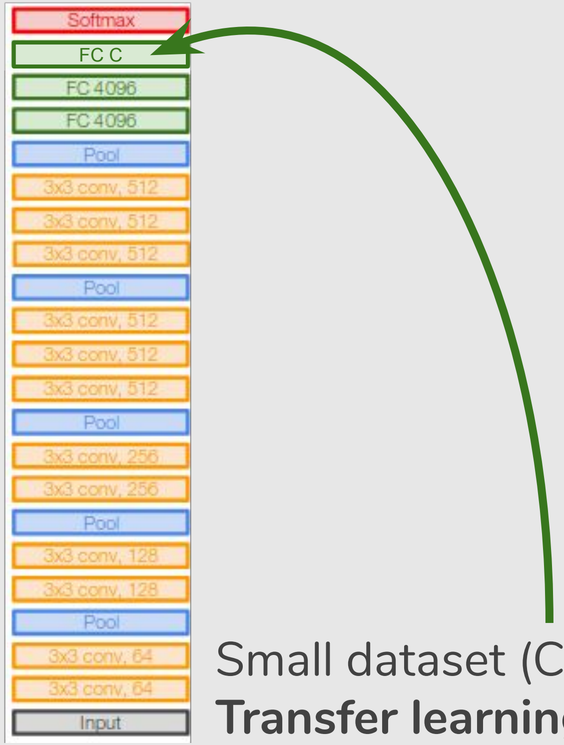


Small dataset (C classes):
Transfer learning with fine-tuning

Transfer Learning with CNNs

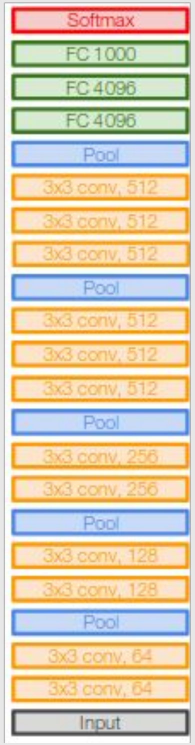


Train on ImageNet

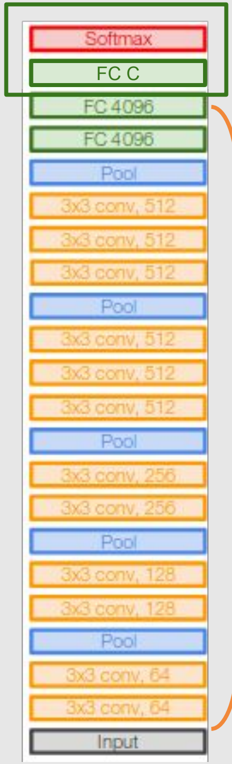


Small dataset (C classes):
Transfer learning with fine-tuning

Transfer Learning with CNNs



Train on ImageNet



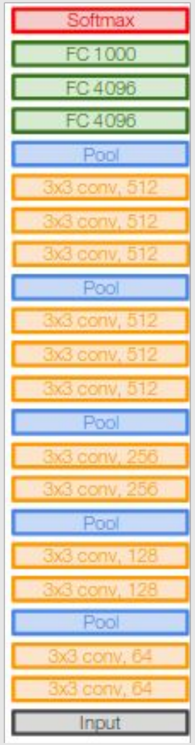
Reinitialize this and train

Freeze these

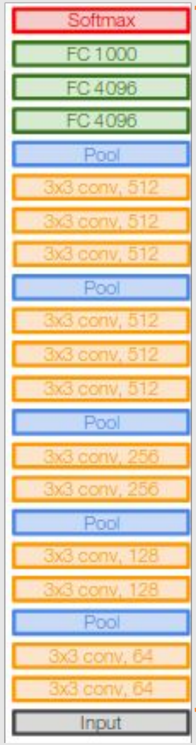
Small dataset (C classes):
Transfer learning with fine-tuning

```python
# Cria o modelo pré-treinado
# include_top: incluir ou não a camada totalmente conectada
# na parte superior da rede
base_model = VGG16(weights='imagenet', include_top=False)

# Adiciona nova camada com 10 classes
...
predictions = Dense(10, activation='softmax')(x)

# Modelo que será treinado
model = Model(inputs=base_model.input, outputs=predictions)

# Congela todas as camadas
for layer in base_model.layers:
    layer.trainable = False
```

Transfer Learning with CNNs



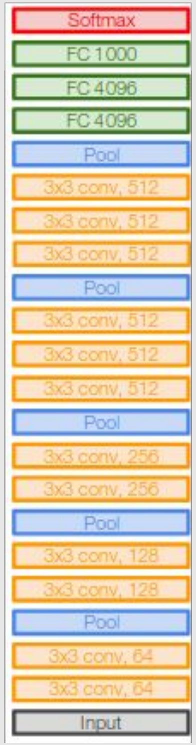
Train on ImageNet



Freeze these

Small dataset (C classes):
Transfer learning without fine-tuning

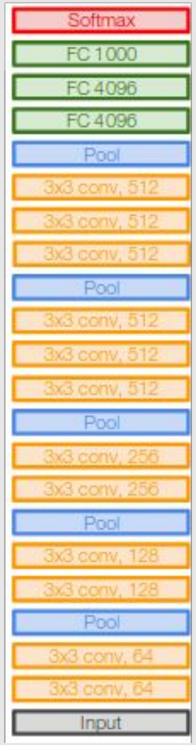
Transfer Learning with CNNs



[0.01 0.8 1 0.5 ... 0.3 0.07 0 0.4 0.6 0 0]
4096-d

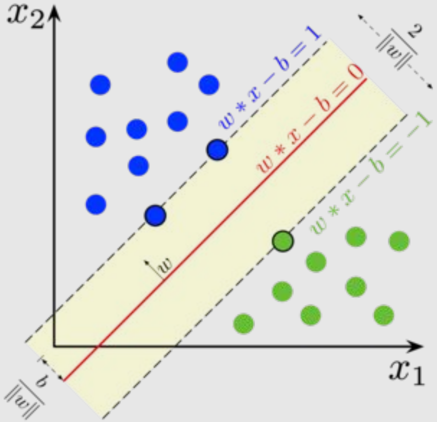
VGG as **Feature Extractor**

Transfer Learning with CNNs



[0.01 0.8 1 0.5 ... 0.3 0.07 0 0.4 0.6 0 0]
4096-d

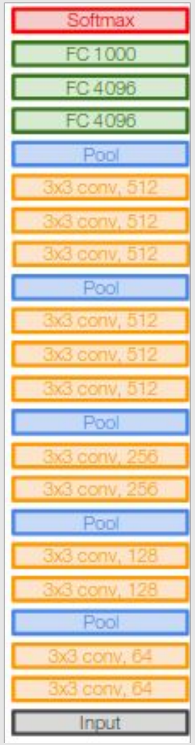
Train a classifier (e.g., SVM)



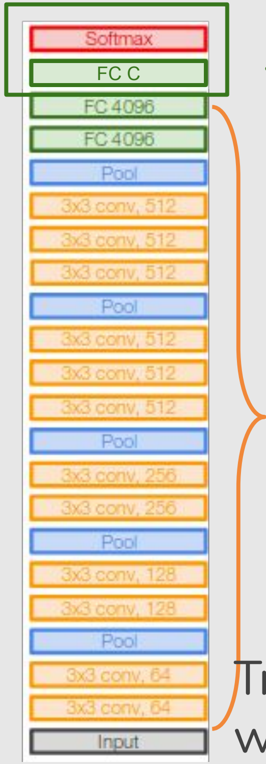
VGG as **Feature Extractor**

```python
# Cria o modelo pré-treinado
base_model = VGG16(weights='imagenet')

# Modelo que será treinado
model = Model(inputs=base_model.input,
              outputs=base_model.get_layer('fc7').output)

# Extração de features
img = ...
features = model.predict(img)
```

Transfer Learning with CNNs



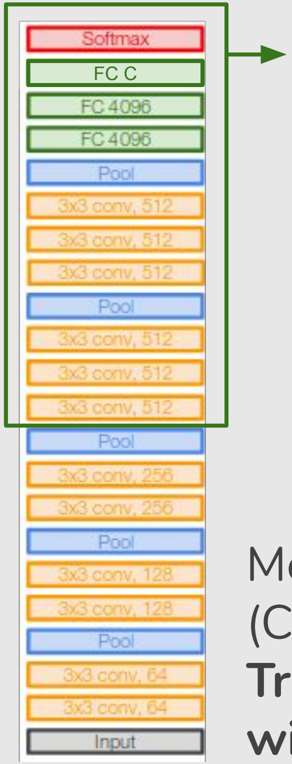
Train on ImageNet



Reinitialize this and train

Freeze these

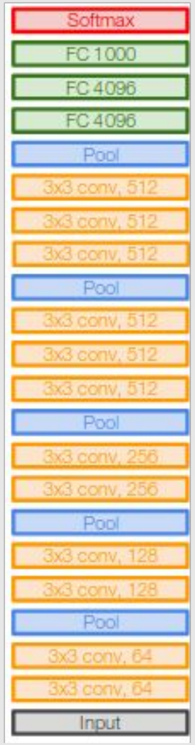
Transfer learning with fine-tuning



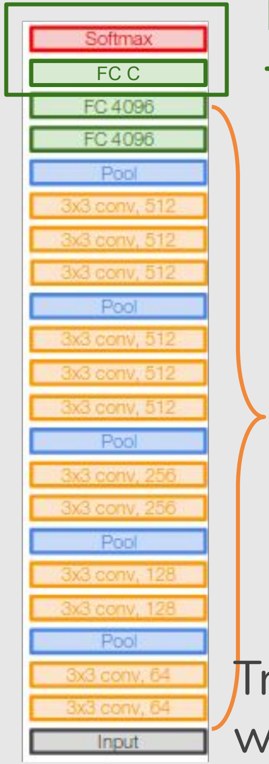
Train this

Medium dataset (C classes):
Transfer learning with fine-tuning

Transfer Learning with CNNs



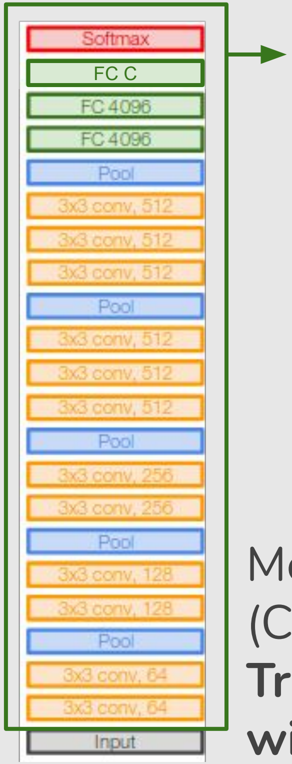
Train on ImageNet



Reinitialize this and train

Freeze these

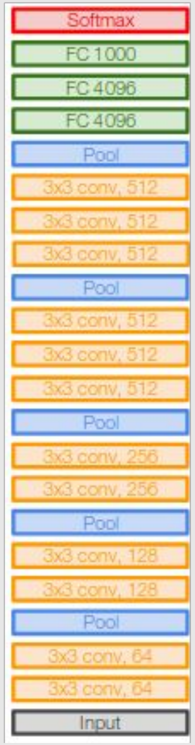
Transfer learning with fine-tuning



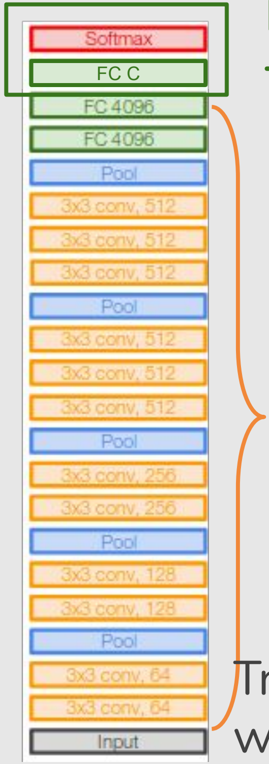
Train this

Medium dataset (C classes):
Transfer learning with fine-tuning

Transfer Learning with CNNs



Train on ImageNet



Reinitialize this and train

Freeze these

Transfer learning with fine-tuning

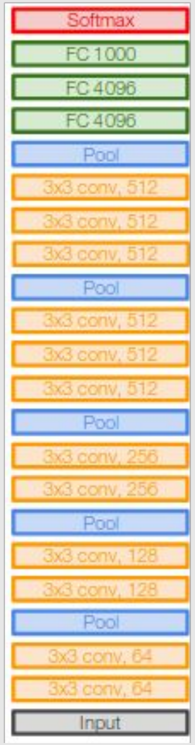
Train this

Lower learning rate when fine-tuning; 1/10 of original LR is good starting point

Transfer learning with fine-tuning

```python
# Cria o modelo pré-treinado
# include_top: incluir ou não a camada totalmente conectada
# na parte superior da rede
base_model = VGG16(weights='imagenet', include_top=False)

# Adiciona nova camada com 10 classes
...
predictions = Dense(10, activation='softmax')(x)

# Modelo que será treinado
model = Model(inputs=base_model.input, outputs=predictions)

# Congela algumas camadas
for layer in base_model.layers[:8]:
    layer.trainable = False
for layer in base_model.layers[8:]:
    layer.trainable = True
```

```python
# Cria o modelo pré-treinado
# include_top: incluir ou não a camada totalmente conectada
# na parte superior da rede
base_model = VGG16(weights='imagenet', include_top=False)

# Adiciona nova camada com 10 classes
...
predictions = Dense(10, activation='softmax')(x)

# Modelo que será treinado
model = Model(inputs=base_model.input, outputs=predictions)

# Congela algumas camadas
for layer in base_model.layers[:8]:
    layer.trainable = False
for layer in base_model.layers[8:]:
    layer.trainable = True
```



More specific

More generic

	Very similar dataset	Very different dataset
Very little data	?	?
Quite a lot of data	?	?

https://distill.pub/2017/feature-visualization

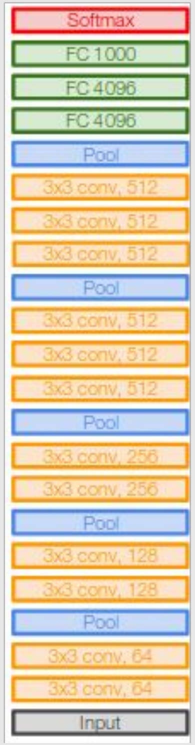
Edges Textures Patterns Parts Objects

More generic **More specific**

https://distill.pub/2017/feature-visualization

	Very similar dataset	Very different dataset
Very little data	?	?
Quite a lot of data	?	?

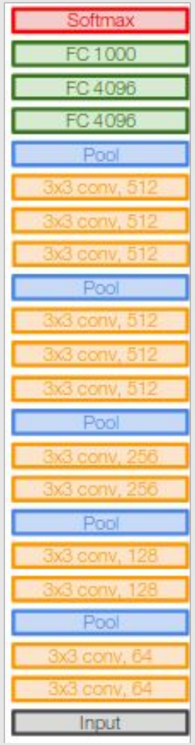
	Very similar dataset	Very different dataset
Very little data	Use Linear Classifier on top layer	?
Quite a lot of data	?	?



More specific

More generic

	Very similar dataset	Very different dataset
Very little data	Use Linear Classifier on top layer	?
Quite a lot of data	Finetune a few layers	?



More specific

More generic

	Very similar dataset	Very different dataset
Very little data	Use Linear Classifier on top layer	?
Quite a lot of data	Finetune a few layers	Finetune a larger number of layers

	Very similar dataset	Very different dataset
Very little data	Use Linear Classifier on top layer	You’re in trouble… Try linear classifier from different stages
Quite a lot of data	Finetune a few layers	Finetune a larger number of layers

http://www.ic.unicamp.br/~sandra/pdf/papers/menegola_ISBI17.pdf

Knowledge Transfer for Melanoma Screening with Deep Learning

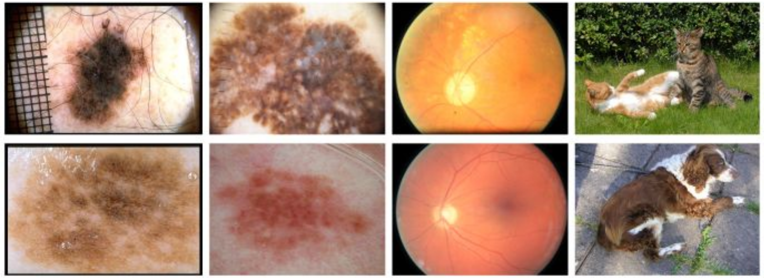
Afonso Menegola[†‡], Michel Fornaciali[†‡], Ramon Pires[°],
Flávia Vasques Bittencourt[•], Sandra Avila[†], Eduardo Valle[†*]

[†]RECOD Lab, DCA, FEEC, University of Campinas (Unicamp), Brazil
[°]RECOD Lab, IC, University of Campinas (Unicamp), Brazil
[•]School of Medicine, Federal University of Minas Gerais (UFMG), Brazil

ABSTRACT

Knowledge transfer impacts the performance of deep learning — the state of the art for image classification tasks, including automated melanoma screening. Deep learning's greed for large amounts of training data poses a challenge for medical tasks, which we can alleviate by recycling knowledge from models trained on different tasks, in a scheme called *transfer learning*. Although much of the best art on automated melanoma screening employs some form of transfer learning, a systematic evaluation was missing. Here we investigate the presence of transfer, from which task the transfer is sourced, and the application of fine tuning (i.e., retraining of the deep learning model after transfer). We also test the impact of picking deeper (and more expensive) models. Our results favor deeper models, pre-trained over ImageNet, with fine-tuning, reaching an AUC of 80.7% and 84.5% for the two skin-lesion datasets evaluated.

Index Terms— Melanoma screening, dermoscopy, deep learn-



(a) (b) (c) (d)

Fig. 1. Samples from datasets used here: (a) Atlas; (b) ISIC; (c) Retinopathy; (d) ImageNet. Each row shows a sample from a different class in the dataset. In this paper, datasets *c* and *d* are source datasets used for transferring knowledge to target models trained in the target task of melanoma screening, trained and evaluated in datasets *a* and *b*.

VGG-16

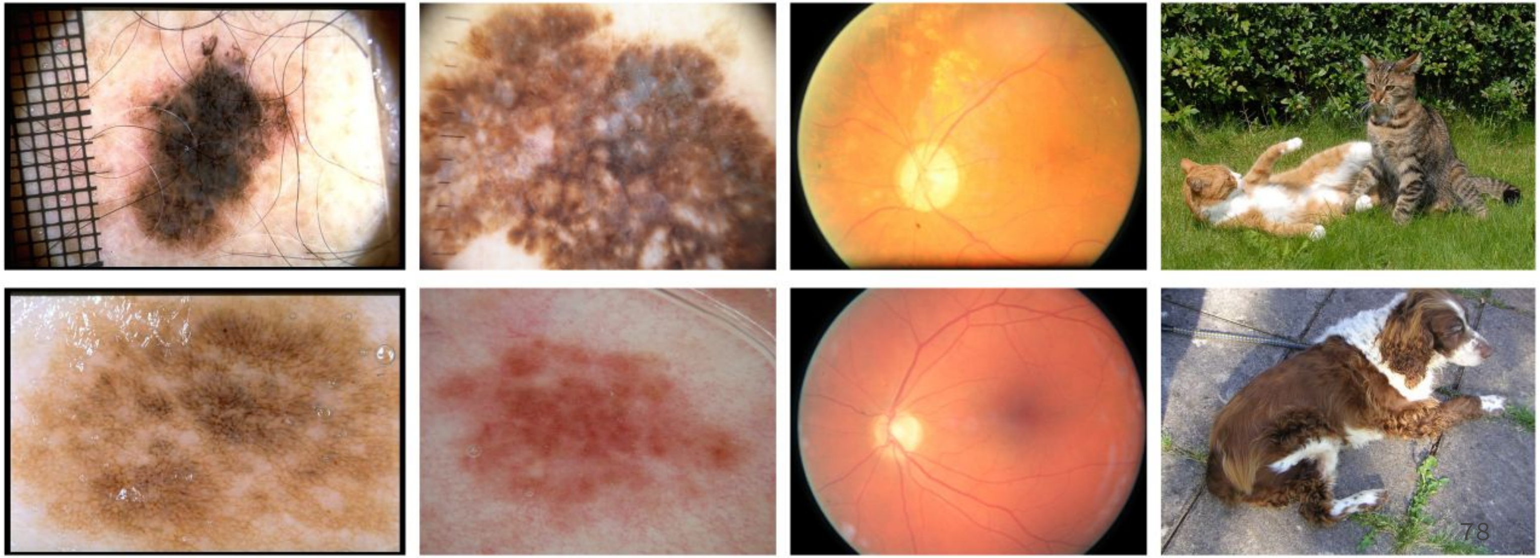
ImageNet -> Melanoma

Double Transfer:

ImageNet -> Retina &

Retina -> Melanoma

2.000 +35.000 +1.200.000



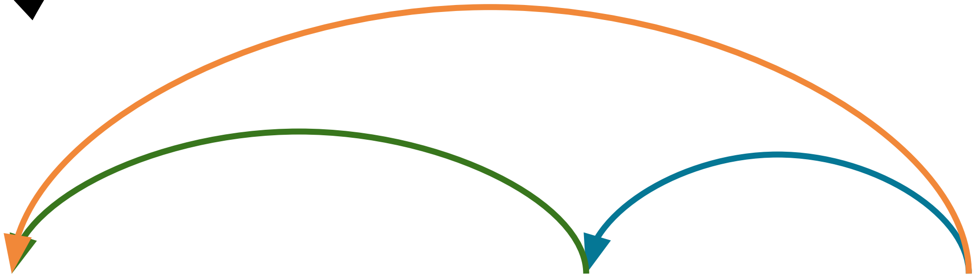
ImageNet -> Melanoma ✓

VGG-16

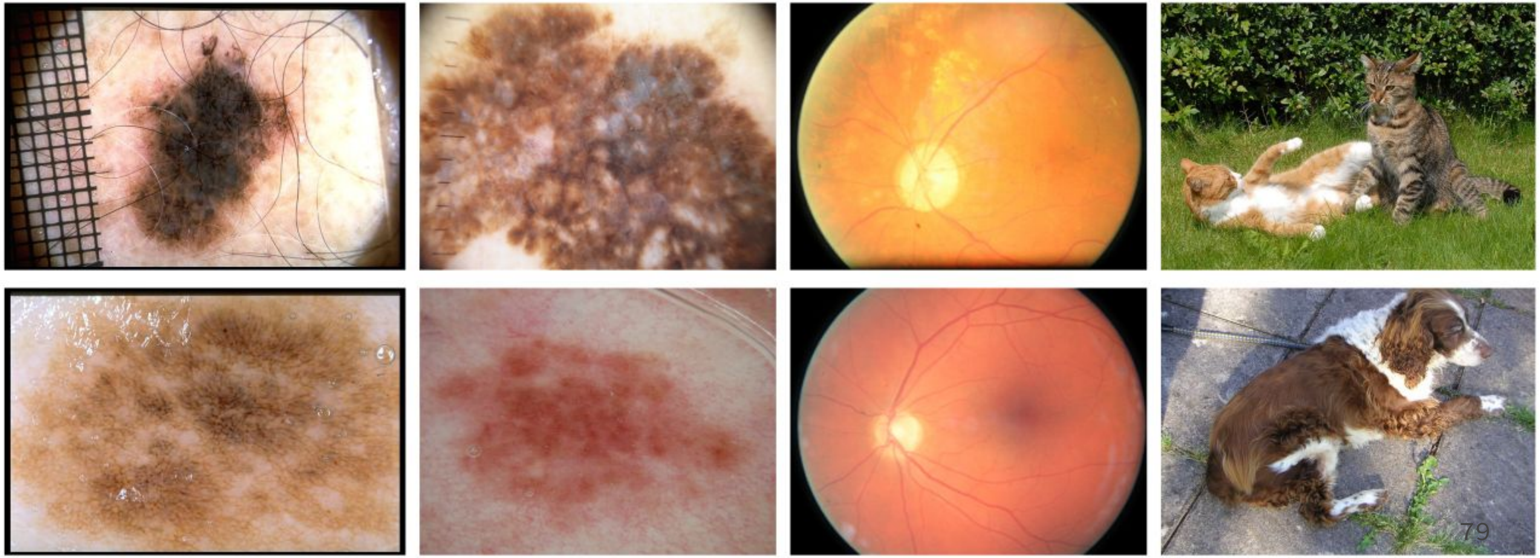
Double Transfer:

ImageNet -> Retina &

Retina -> Melanoma



2.000 +35.000 +1.200.000



http://www.sciencedirect.com/science/article/pii/S1047320317302377

Journal of Visual Communication and Image Representation 50 (2018) 303–313

Contents lists available at ScienceDirect

Journal of Visual Communication and Image Representation

journal homepage: www.elsevier.com/locate/jvci

Leveraging deep neural networks to fight child pornography in the age of social media☆

Paulo Vitorino[a,b], Sandra Avila[c,*], Mauricio Perez[d], Anderson Rocha[c,*]

[a] Department of Electrical Engineering, University of Brasilia, Brazil
[b] Brazilian Federal Police, Brazil
[c] Institute of Computing, University of Campinas, Brazil
[d] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

ARTICLE INFO

Keywords:
Child pornography
SEIC content
Deep learning
Transfer learning
Fine tuning

ABSTRACT

Over the past two decades, the nature of child pornography in terms of generation, distribution and possession of images drastically changed, evolving from basically covert and offline exchanges of content to a massive network of contacts and data sharing. Nowadays, the internet has become not only a transmission channel but, probably, a child pornography enabling factor by itself. As a consequence, most countries worldwide consider a crime to take, or permit to be taken, to store or to distribute images or videos depicting any child pornography grammar. But before action can even be taken, we must detect the very existence or presence of sexually exploitative imagery of children when gleaning over vast troves of data. With this backdrop, veering away from virtually all off-the-shelf solutions and existing methods in the literature, in this work, we leverage cutting-edge data-driven

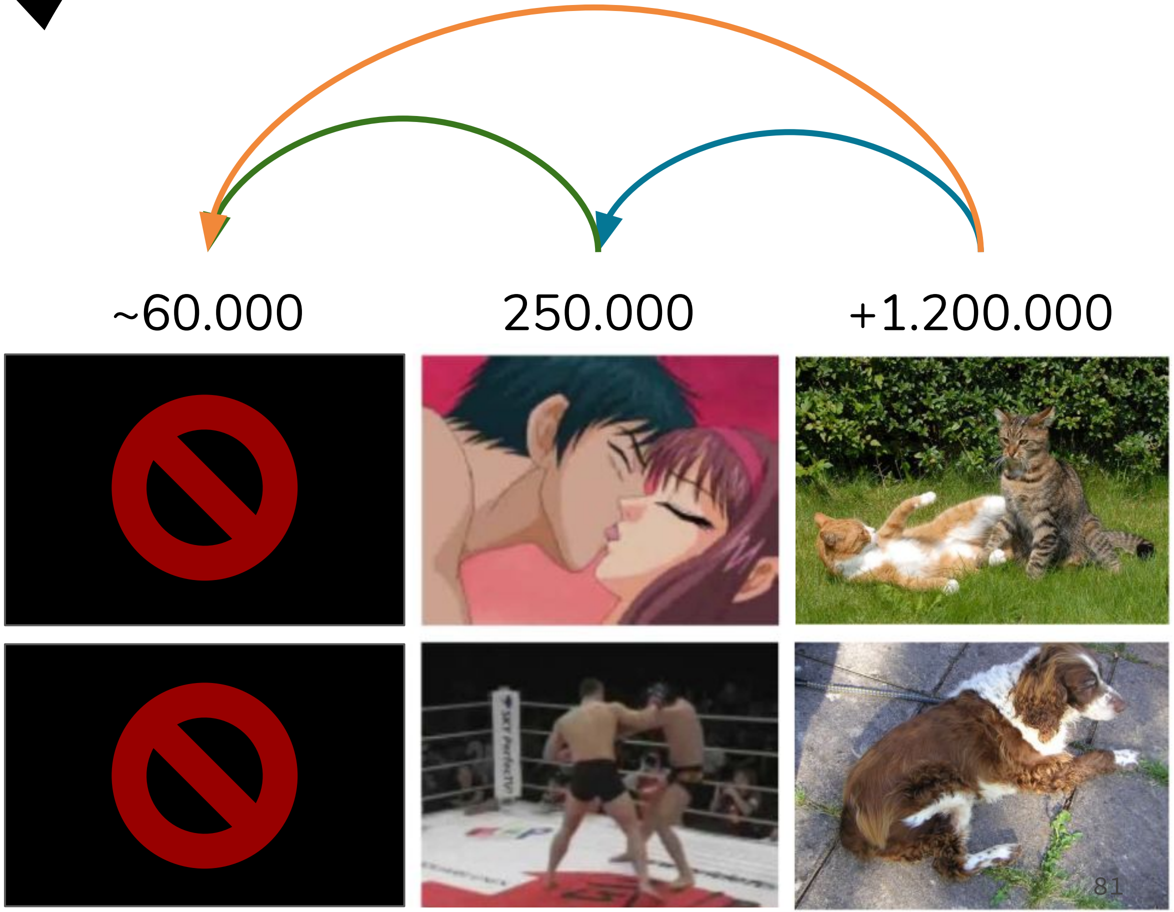
GoogLeNet

ImageNet -> Child Porn ✓

Double Transfer:

ImageNet -> Porn &

Porn -> Child Porn



Takeaway for your projects and beyond ...

- Have some dataset of interest but it has < ~1M images?
  - Find a very large dataset that has similar data, train a big CNN there
  - Transfer learn to your dataset

- You don't need to train your own:
  - TensorFlow: https://github.com/tensorflow/models
  - PyTorch: https://github.com/pytorch/vision

Today's Agenda

- Activation Functions (use **ReLU**)
- Data Preprocessing (images: **subtract mean**)
- Weight Initialization (use **Xavier/He init**)
- ~~Batch~~ Normalization (**use**)
- Optimizers (use **Adam**)
- Regularization (**use**)
- Transfer learning / fine-tuning (**use**)