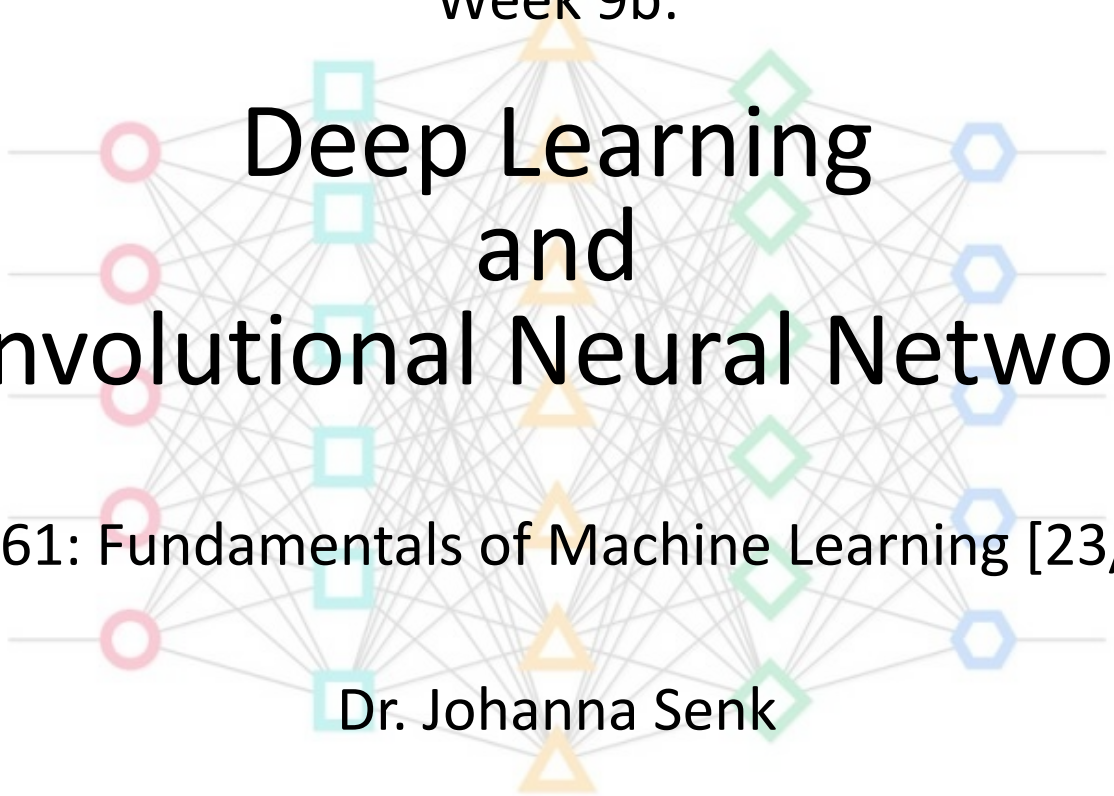
Week 9b:

Deep Learning and Convolutional Neural Networks

G6061: Fundamentals of Machine Learning [23/24]

Dr. Johanna Senk



UNIVERSITY OF SUSSEX

Learning Objectives

- Be able to select appropriate losses to use neural networks to solve particular tasks
- Know some approaches to train effective deep neural networks

Content

- Providing supervision
  - Mean squared error
  - Binary Cross-entropy
- Training Deep Convolutional Neural Networks
  - Stochastic gradient descent
  - Backpropagation
  - Training
- Preventing overfitting
  - Dropout and other regularizations
  - Data augmentation
- Feature extraction and Fine-tuning
- Visualization of CNNs
  - Neural Style Transfer

Loss Functions

- Mean Squared error
- Binary Cross-entropy

Loss Functions

- An important thing to remember about Neural networks is that the architecture (set of layers) only defines the flow of information through the system.
- In order to learn any useful values for the parameters of these layers, we need to ask it to achieve a task.
- We specify the task by providing a **loss function** on our model output.
  - In supervised learning scenarios, this loss function measures the difference between the model's predictions given the input and the labelled data.

Continuous Predictions

- What kind of task involves predicting continuous values?
- What loss function can we use for this?

Mean Squared Error

- As we've seen for regression type tasks, the squared residual error: $(\hat{f}(x) - f(x))^2$ is a useful function to minimize.
- This loss is often used in deep learning for predicting continuous values.
  - It's sometimes referred to as the MSE (mean squared error) or L2

```
from tensorflow import keras

model = keras.Model(…)
Model.compile(…,
loss=(keras.losses.MeanSquaredError))
```

Discrete Predictions

- What kind of task involves predicting discrete values?
- What loss function can we use for this?

Binary Cross-Entropy

- In classification tasks, we need to use an alternative loss function to account for the discrete nature of the data.
- Binary classification labels follow a Bernoulli distribution
- $\hat{f}(x)^{f(x)}(1-\hat{f}(x))^{(1-f(x))}$
- In binary classification, we try to maximise the log probability for predicting the correct class.
- This loss function is called the Binary Cross Entropy.
- $f(x)\log\left(\hat{f}(x)\right)+(1-f(x))\log\left(1-\hat{f}(x)\right)$

Binary Cross-Entropy

- This loss function can be calculated either before/after sigmoid, and it's worth checking what the framework you're using expects.

```
from tensorflow import keras
# We need a model that only contains a single
value as the output
model = keras.Model(…)
Model.compile(…,
loss=(keras.losses.BinaryCrossEntropy))
```

Categorical Cross-Entropy

- When you're predicting between more than 2 classes, you need to use the Categorical Cross Entropy.
  - Labels follow a *categorical distribution*
- This assumes your model predicts a probability for each of the classes
  - to ensure the probabilities sum to 1 we use a *softmax* activation function after the final layer.
- Note that the training data often needs to be converted to a 1-hot encoding. E.g. instead of saying this image is class 2, we have a binary vector[0, 0, 1]

```
from tensorflow import keras
# We need a model that outputs a vector of class probabilities
model = keras.Model(…)
Model.compile(…,
loss=(keras.losses.CategoricalCrossEntropy))
```

Multiple Losses

- It's quite common for your model to want to make multiple predictions of different types of data.
- This is referred to as *multi-task* learning
- You need to assign a weight (importance) to the model solving each of these tasks

```
from tensorflow import keras
# We need a model that outputs a vector of class probabilities
model = keras.Model(…)
Model.compile(…,
loss=(keras.losses.CategoricalCrossEntropy,
keras.losses.MeanSquaredError), loss_weights=(0.5, 10.0))
```

Losses

- The idea of losses is fundamental to neural network models.
- To learn a model for a different type of task, we need to choose the right loss.
- You can use losses to make your model self-, semi-, or weakly supervised.
- As well as losses, you can also print ***metrics***, these are useful measures of performance that you perhaps cannot directly optimize for

```
model.compile(…,
metrics=["sparse_categorical_accuracy"])
```

Training CNNs

- Stochastic gradient descent
- Backpropagation
- Training

Stochastic gradient descent (SGD)

(Mini-batch) SGD

Initialize the parameters

Loop over the whole training data (multiple times):

- **Sample** a datapoint (a batch of data)
- **Forward** propagate the data through the network, compute the classification loss.
- **Backpropagate** the gradient of the loss w.r.t. parameters through the network
- **Update** the parameters using the gradient

Stochastic gradient descent (SGD)

(Mini-batch) SGD

Initialize the parameters **randomly but smartly**

Loop over the whole training data (multiple times):

- **Sample** a datapoint (a batch of data)
- **Forward** propagate the data through the network, compute the classification loss. **For example:** $E = \frac{1}{2}(y_{predicted} - y_{true})^2$
- **Backpropagate** the gradient of the loss w.r.t. parameters through the network
- **Update** the parameters using the gradient

SGD: $w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$

SGD Code

- You need to specify which SGD type optimizer to use
- Very importantly, you also need to choose the learning rate!
  - This can make a huge difference to whether your model learns anything useful or not!

```
from tensorflow import keras
model = keras.Model(…)
Model.compile(…, loss=(…)
optimizer=keras.optimizers.Adam(learning_rate=0.001
)
```

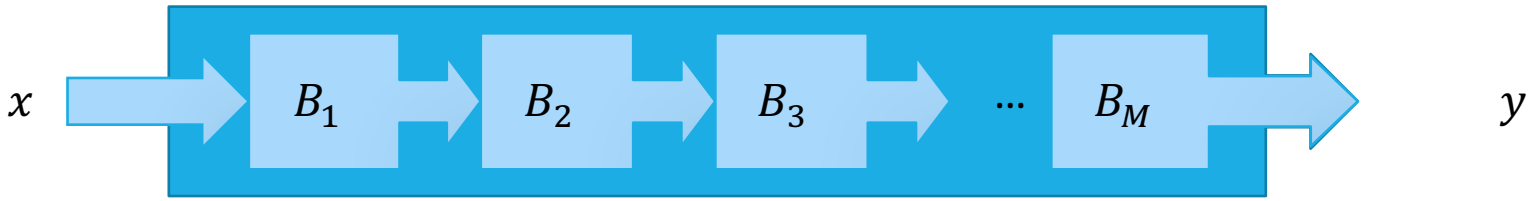
Backpropagation

- Backpropagation is recursive application of the chain rule along a computational flow of the network to compute gradients of the loss function w.r.t. all parameters/intermediate variables/inputs in the network

Backpropagation

- Implementations typically maintain a modular structure, where the nodes/bricks implement the forward and backward procedures

Sequential brick



Propagation

- Apply propagation rule to $B_1, B_2, B_3, \dots, B_M$.

Back-propagation

- Apply back-propagation rule to $B_M, \dots, B_3, B_2, B_1$.

Backpropagation

- Last layer used for classification

Square loss brick

Propagation

$$E = y = \frac{1}{2}(x - d)^2$$

Back-propagation

$$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y} = (x - d)^T$$

Backpropagation

- Typical choices

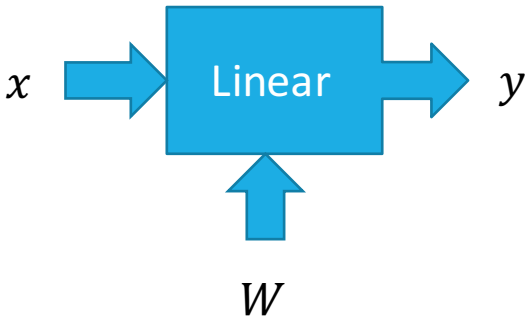
Loss bricks

		Propagation	Back-propagation
Square		$y = \frac{1}{2}(x-d)^2$	$\frac{\partial E}{\partial x} = (x-d)^T \frac{\partial E}{\partial y}$
Log	$c = \pm 1$	$y = \log(1+e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1+e^{cx}} \frac{\partial E}{\partial y}$
Hinge	$c = \pm 1$	$y = \max(0, m-cx)$	$\frac{\partial E}{\partial x} = -c \ \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax	$c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s}/\sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin	$c = 1 \dots k$	$y = \left[\max_{k \neq c}\{x_k + m\} - x_c\right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \ \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

Backpropagation

- Fully connected layers, convolutional layers (dot product)

Linear brick



Propagation

$$y = Wx$$

Back-propagation

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y} W$$

$$\frac{\partial E}{\partial W} = x \frac{\partial E}{\partial y}$$

Backpropagation

- Non-linear activations

Activation function brick

Propagation

$$y_s = f(x_s)$$

Back-propagation

$$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s f'(x_s)$$

Backpropagation

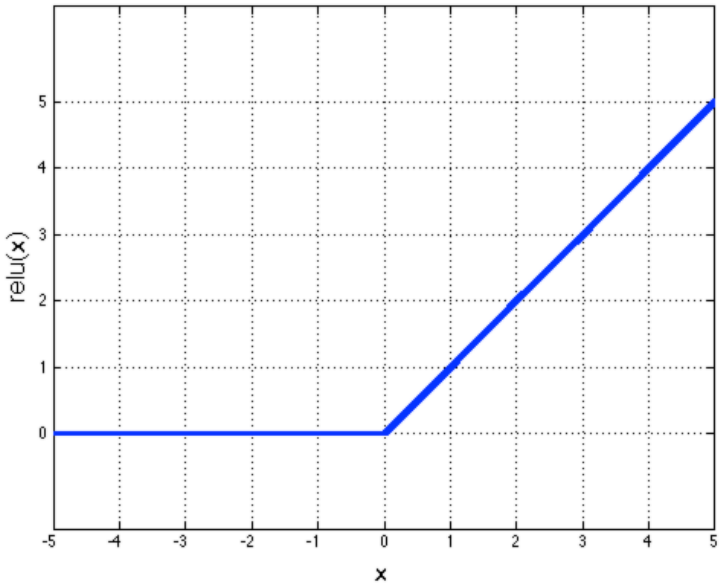
- Typical non-linear activations

Activation functions

	Propagation	Back-propagation
Sigmoid	$y_s = \frac{1}{1+e^{-x_s}}$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{(1+e^{x_s})(1+e^{-x_s})}$
Tanh	$y_s = \tanh(x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \frac{1}{\cosh^2 x_s}$
ReLu	$y_s = \max(0, x_s)$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{x_s > 0\}$
Ramp	$y_s = \min(-1, \max(1, x_s))$	$\left[\frac{\partial E}{\partial x}\right]_s = \left[\frac{\partial E}{\partial y}\right]_s \mathbb{I}\{-1 < x_s < 1\}$

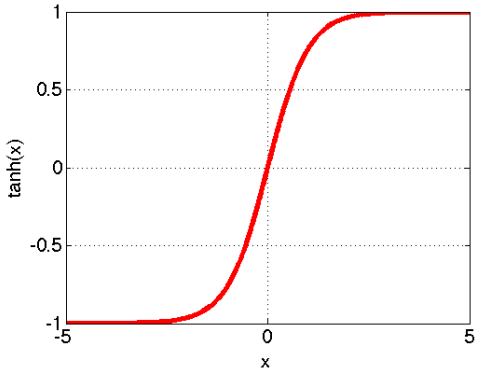
Recap: ReLU

- Non-linear activation function are applied per-element
- Rectified linear unit (ReLU):
  - max(0,x)
  - makes learning faster (in practice x6)
  - avoids saturation issues (unlike sigmoid, tanh)
  - simplifies training with backpropagation
  - preferred option (works well)

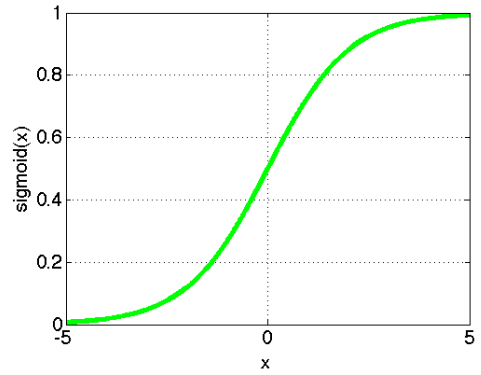


Other examples:

tanh(x)

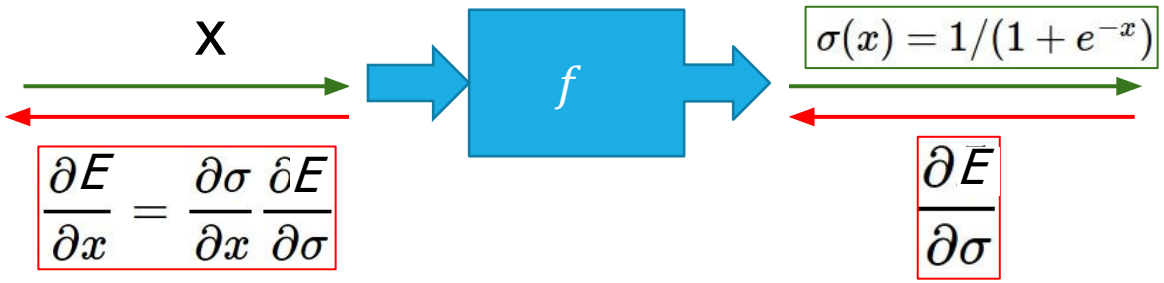


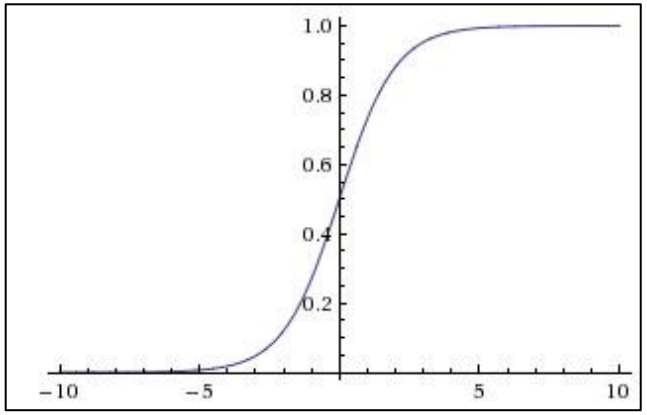
sigmoid(x)=$(1+e^{-x})^{-1}$



Quiz

- Saturation of the gradient of logistic sigmoid ?



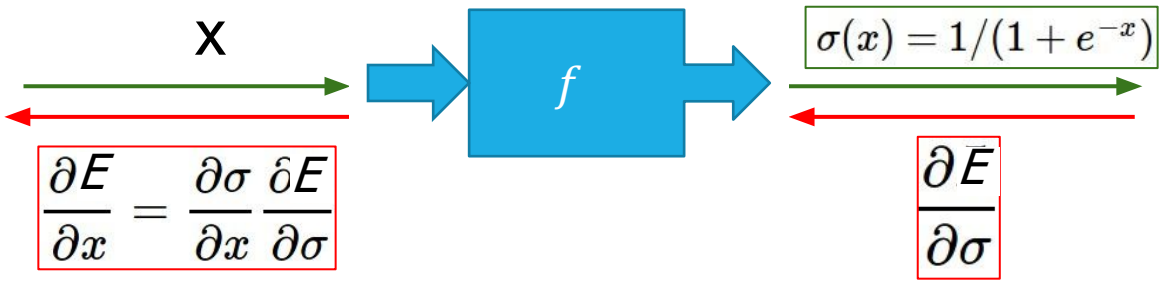


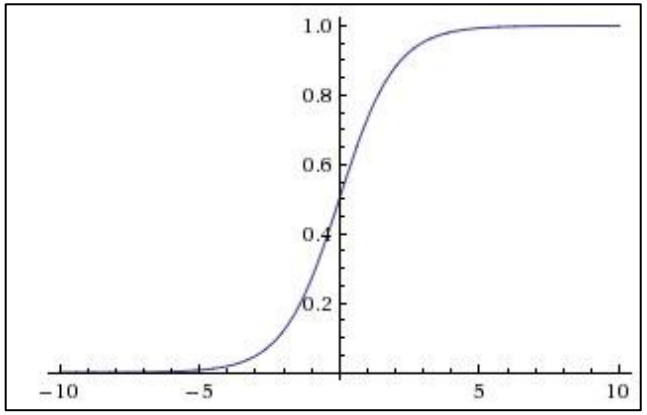
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Hint 1: Think about the gradient $\frac{\partial \sigma}{\partial x}$

Quiz

- Saturation of the gradient of logistic sigmoid ?



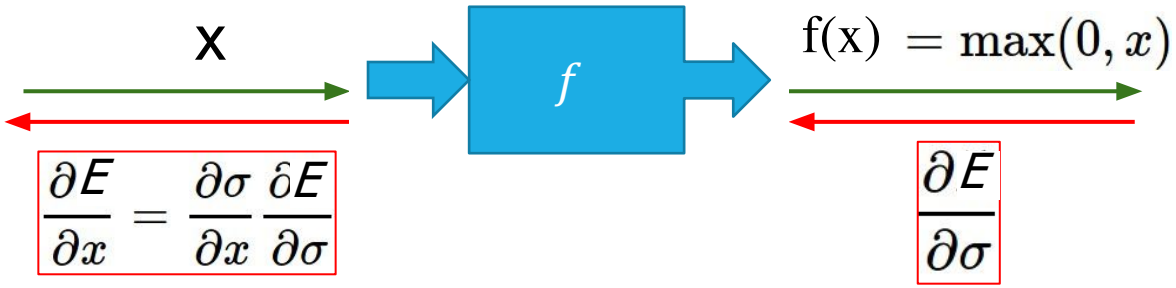


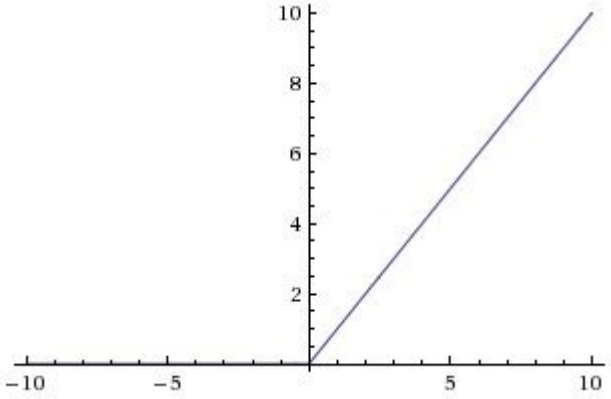
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Hint 2: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1- \sigma(x))$

Quiz

- Saturation of the gradient of ReLU max(0,x) ?

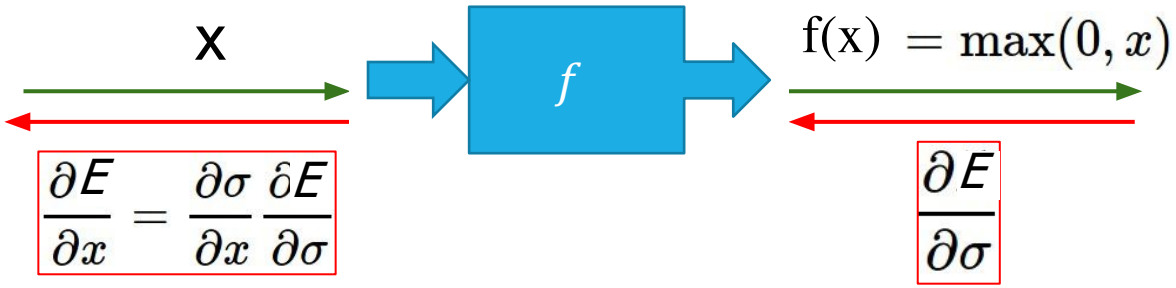


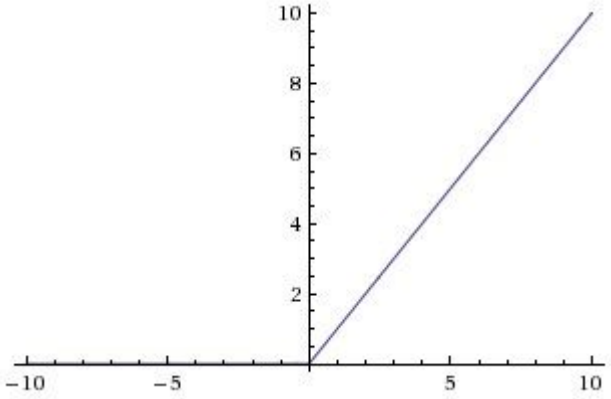


What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

Quiz

- Saturation of the gradient of ReLU max(0,x) ?



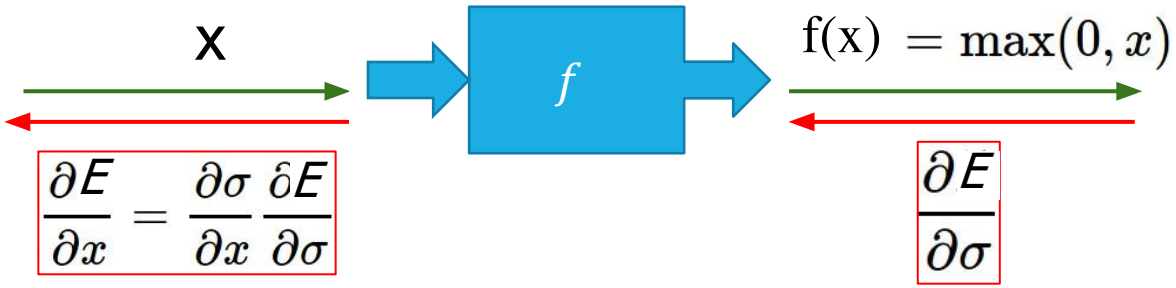


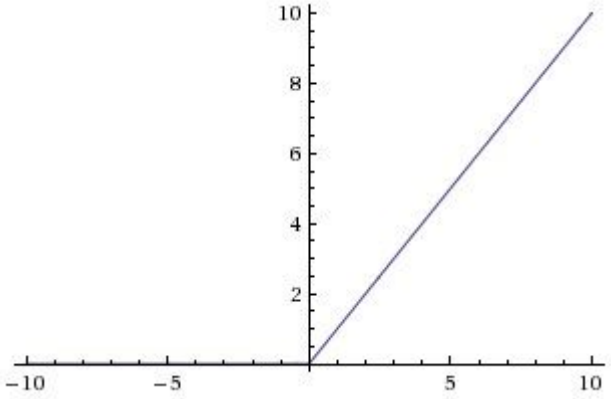
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

- gradient does not saturate in positive region (x>0)
- what happens when x<=0 ?

Quiz

- Saturation of the gradient of ReLU max(0,x) ?



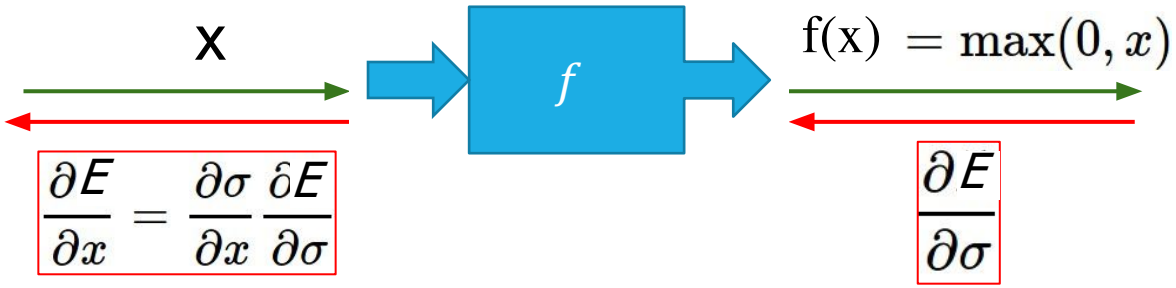


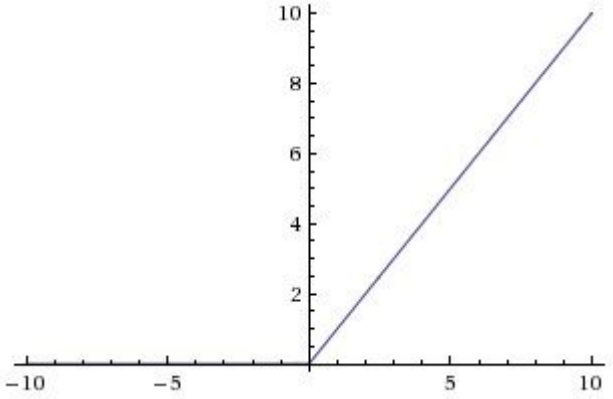
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

- gradient does not saturate in positive region (x>0)
- gradient is 0 when x<0, so ReLU "dies" ?

Quiz

❑ Saturation of the gradient of ReLU max(0,x) ?





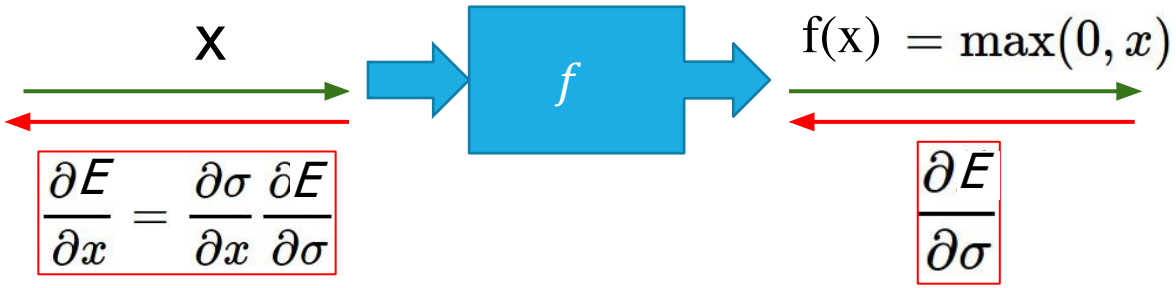
What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

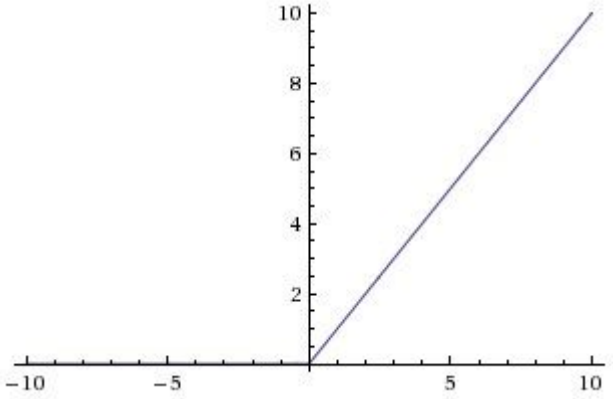
- gradient does not saturate in positive region (x>0)
- gradient is 0 when x<0, so ReLU "dies"

Good that we have many data points, so it can come back "alive"

Quiz

- Saturation of the gradient of ReLU max(0,x) ?



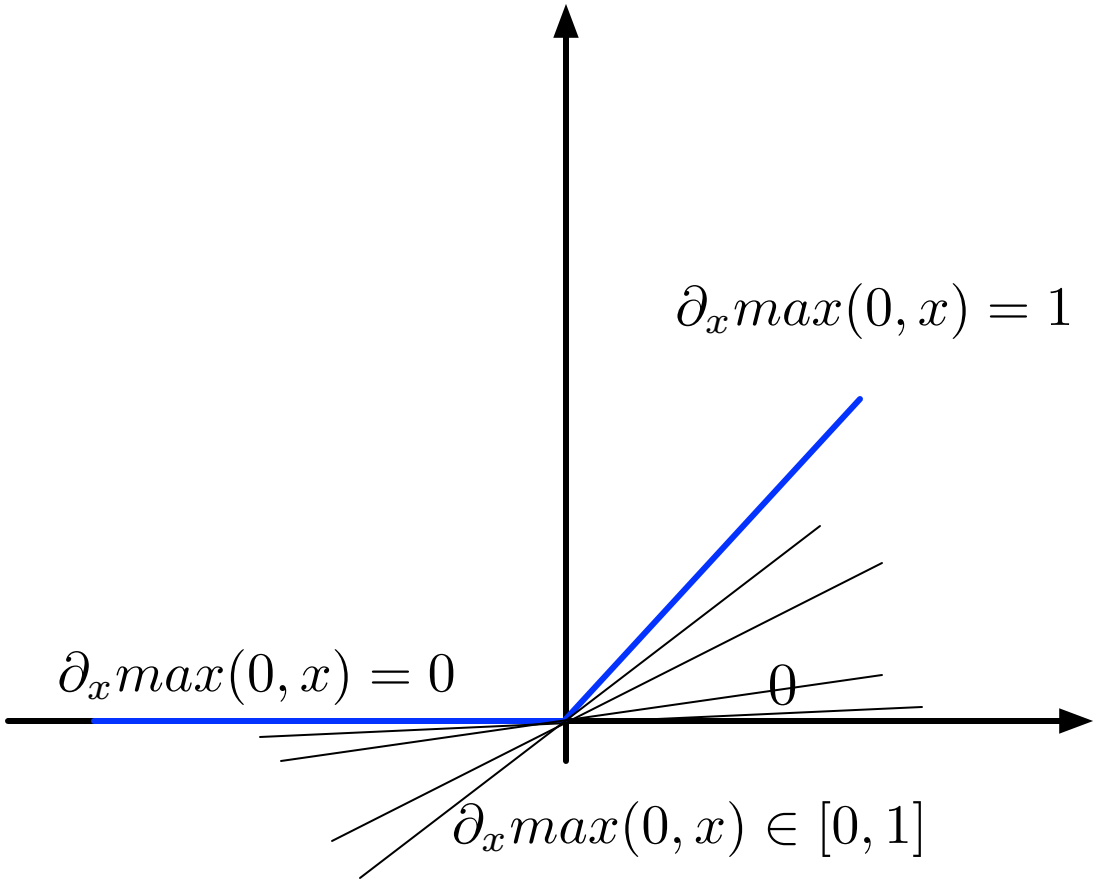


What happens when x = -10?
What happens when x = 0?
What happens when x = 10?

- gradient does not saturate in positive region (x>0)
- gradient is 0 when x<0, so ReLU "dies"
- what happens to gradient when x=0?

Subgradient

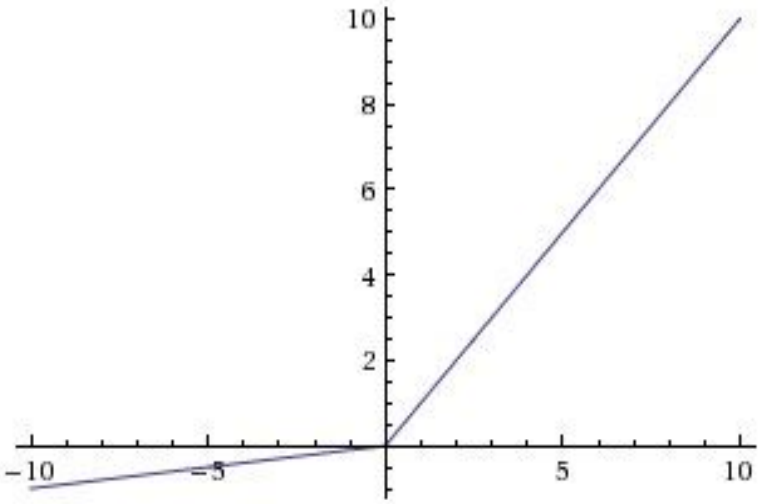
- ReLU gradient is not defined at x=0, use a subgradient instead



- Practice note: during training, when a 'kink' point was crossed, the numerical gradient will not be exact.

[Leaky ReLU: extra]

- In practice, people like to use *Leaky ReLU*, f(x) = max(0.01x, x) to avoid saturation of the gradient and this ReLU will not "die"



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Training CNNs

- ☑ Stochastic gradient descent
- ☑ Backpropagation
- ❑ Training

Training

- Initialization of the (filter) weights

  - don't initialize with zero
  - don't initialize with the same value
  - sample from uniform distribution U[-b,b] around zero or from Normal distribution

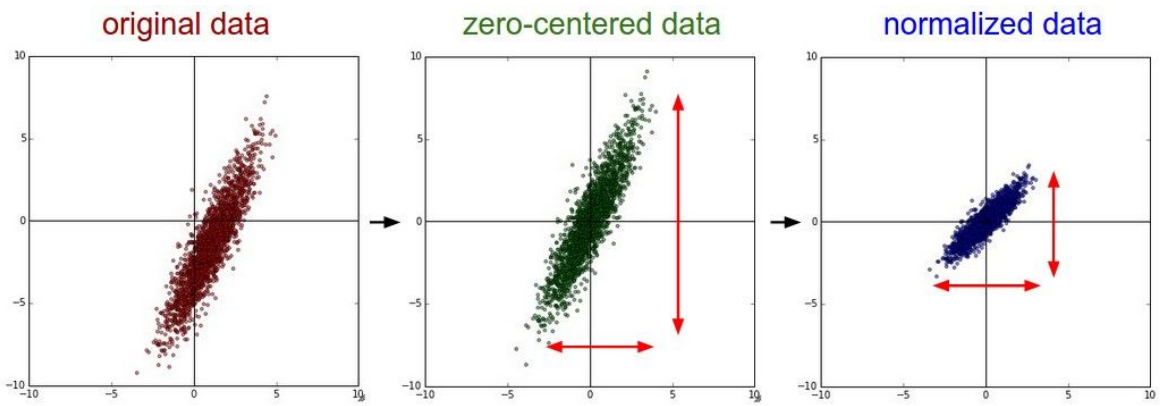
- Decay of the learning rate α

  - as we get closer to the optimum, take smaller update steps
  - start with large learning rate (e.g. 0.1)
  - maintain until validation error stops improving
  - divide learning rate by 2 and go back to previous step

$$w^{t+1} = w^t - \alpha \cdot \frac{dE}{dw}(w^t)$$

Training

- Data preprocessing: normalization/standardization



- In images you might:
  - Subtract the mean of RGB intensities of the whole dataset from each pixel
  - Scale between -1 and 1.
  - Crop/resize the images.
- It's also important to shuffle your training data.

Fitting Code

- You need to have your data in the right format (check documentation)

```
from tensorflow import keras
model = keras.Model(…)
model.compile(…)
model.fit(input_data, label_data, epochs=1)
```

Preventing overfitting

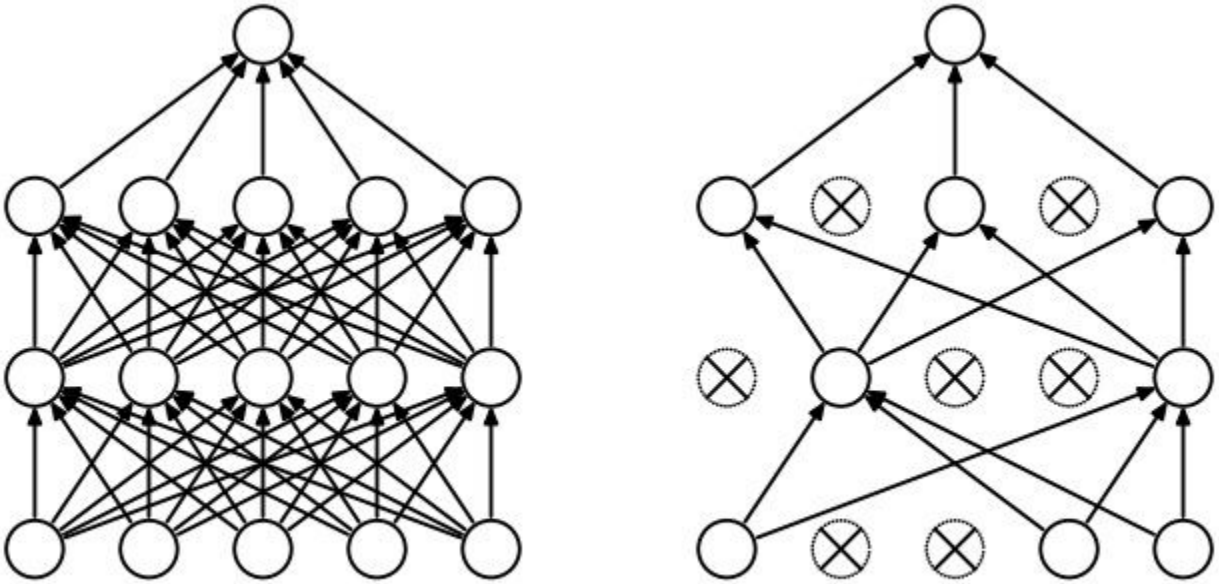
- Dropout regularization
- Data augmentation

Regularization

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"

(with probability 0.5)



(a) Standard Neural Net

(b) After applying dropout.

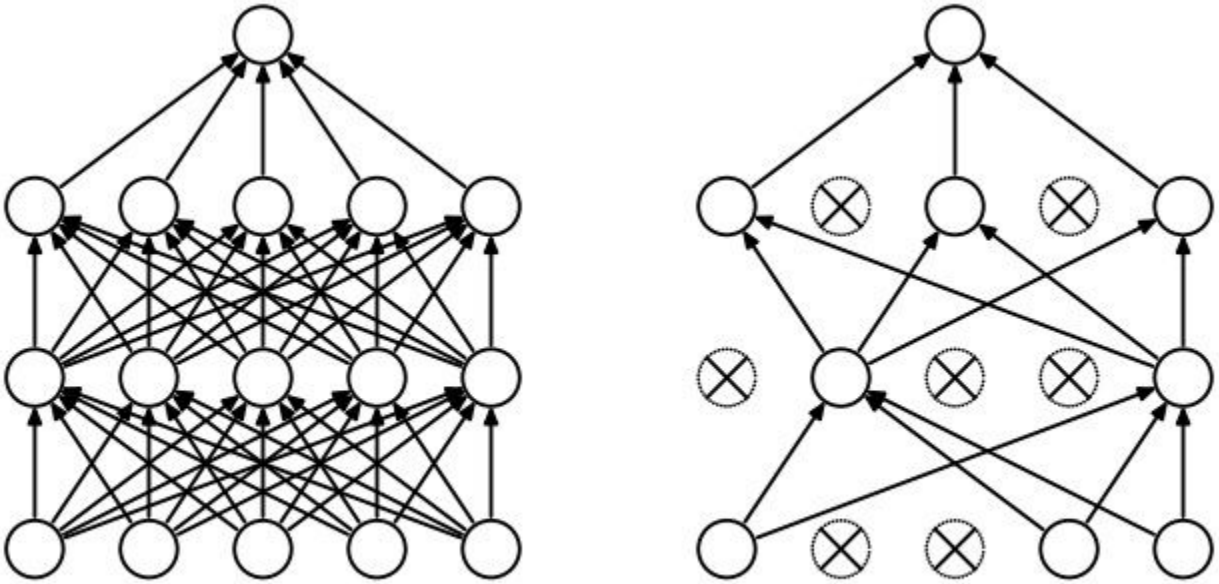
[Srivastava et al., 2014]

Regularization

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"

(with probability 0.5)



(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

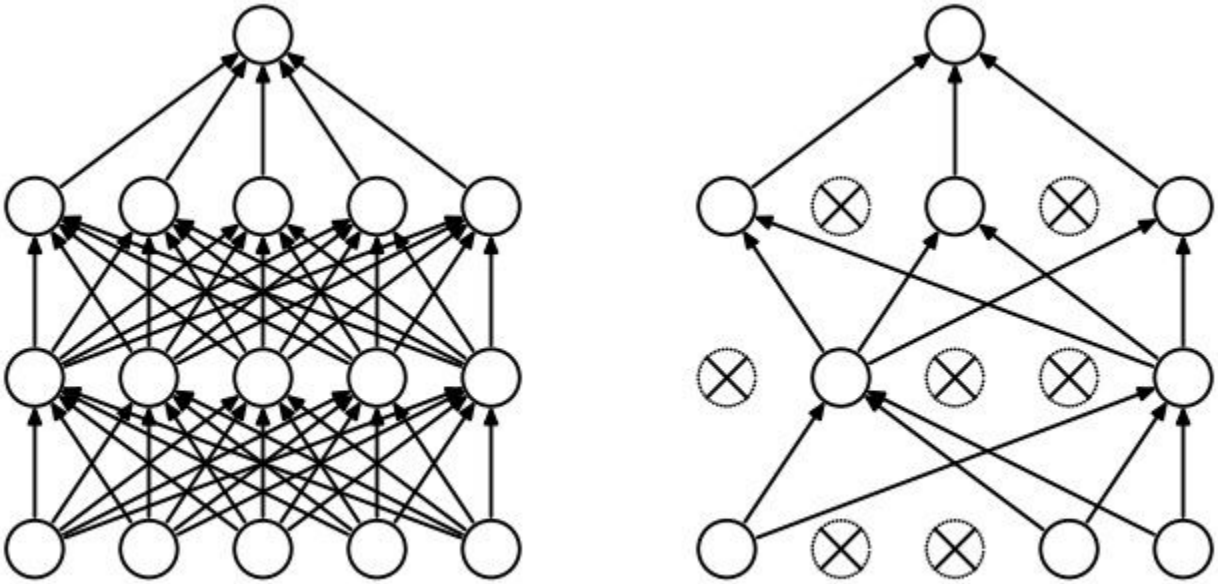
- The neurons which are "dropped out" do not contribute to the forward pass and do not participate in backpropagation.
- So every time an input is presented, the neural network samples different architecture, but all these architectures share weights.

Regularization

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"

(with probability 0.5)



(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

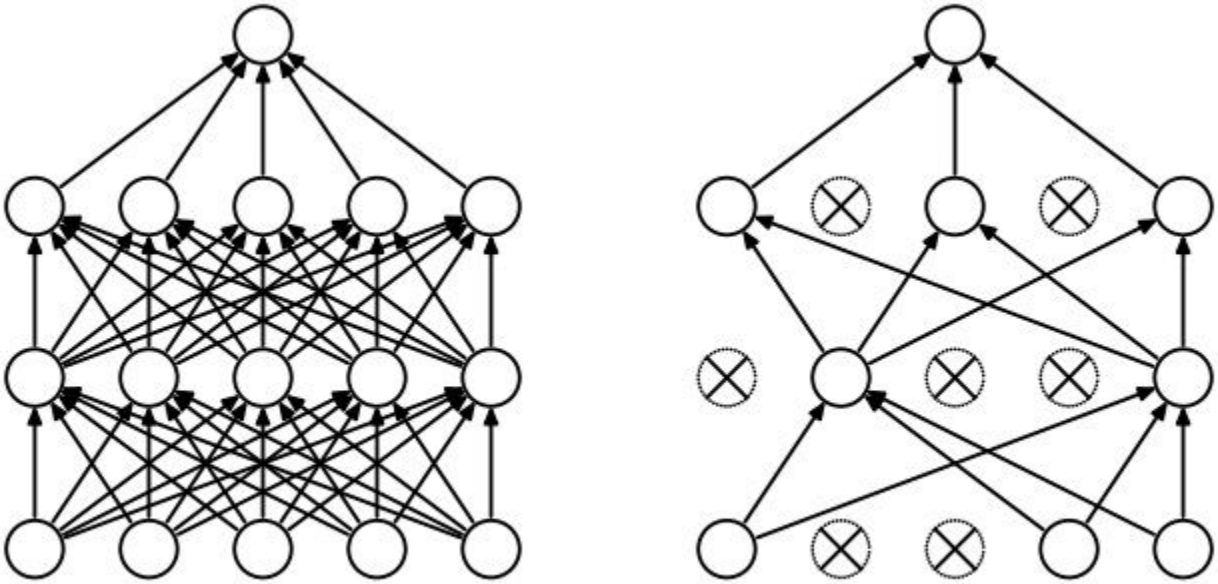
- Dropout could be seen as training a large ensemble of models (each model gets trained on one datapoint or on a batch of data)

Regularization

Regularization: **Dropout**

"randomly set some neurons to zero in the forward pass"

(with probability 0.5)



(a) Standard Neural Net

(b) After applying dropout.

[Srivastava et al., 2014]

- Dropout could be seen as training a large ensemble of models (each model gets trained on one datapoint or on a batch of data)
- At test time, use average predictions over all models (weighted with 0.5)

Dropout

Dropout: set the output of each hidden neuron to zero w.p. 0.5.

- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.
- It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
- Without dropout, CNNs exhibits substantial overfitting.
- Dropout roughly doubles the number of iterations required to converge.

Alternatives:

standard L_2 regularization of weights

BatchNormalizaton

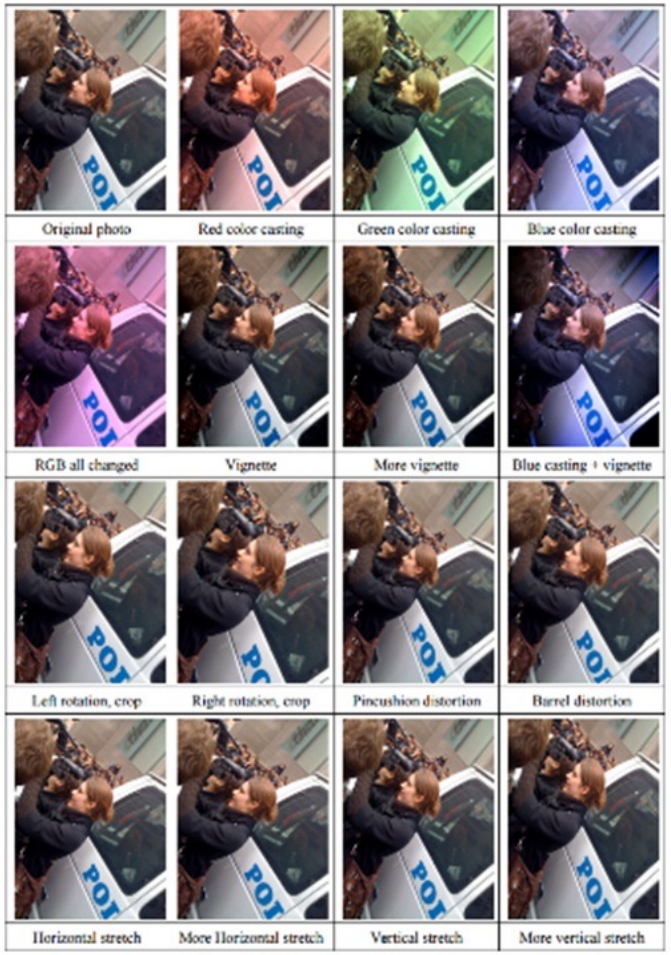
- **Batch Normalization** is a fairly recent (2015) approach to improve the training speed and performance of convolutional neural networks.
- The basic idea is to calculate the mean and variance of each neural activation map (channel) in a mini-batch.
- We then subtract the batch-wise mean and divide by the batch-wise standard deviation.
- We can optionally then apply a new translation and scaling to the data, which is learned.
- BatchNorm layers are typically applied after every set of convolutions.
- BatchNorm (amongst other recent deep learning approaches) has been referred to as alchemy! It's taken several years to try and understand why it works, although it does seem to be quite effective!

Data Augmentation

The easiest and most common method to **reduce overfitting** on image data is to artificially **enlarge the dataset** using supervision-preserving transformations.

Forms of data augmentation:

- horizontal reflections
- random crop
- changing RGB intensities
- image translation

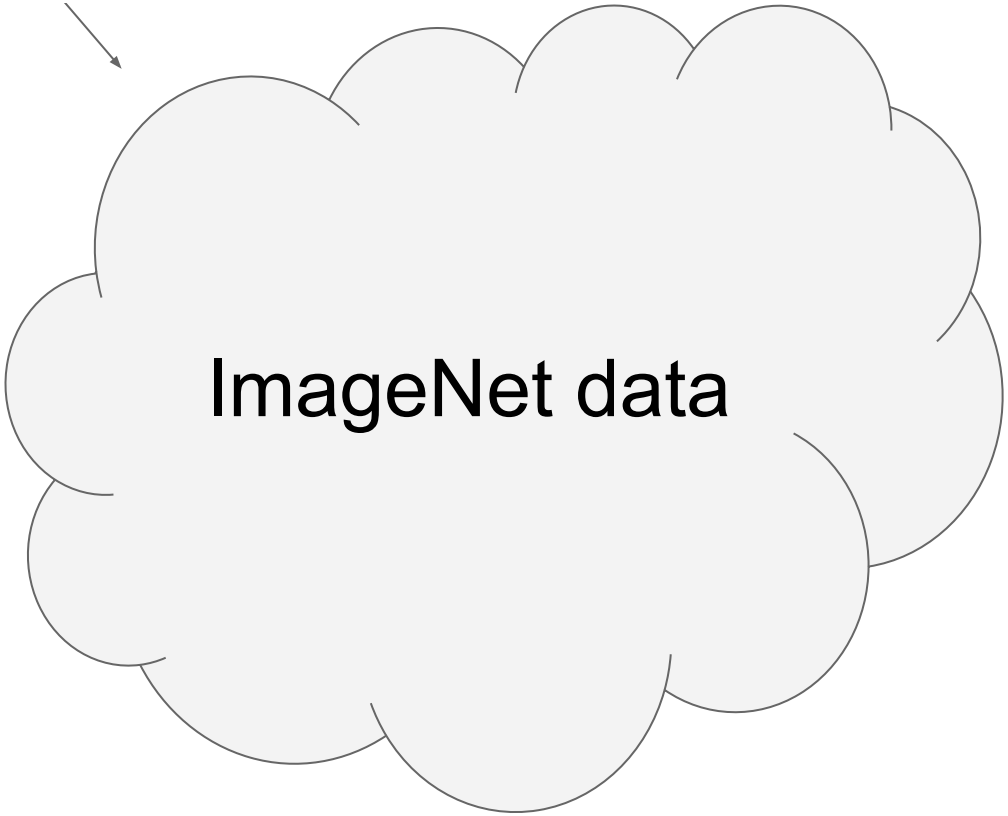


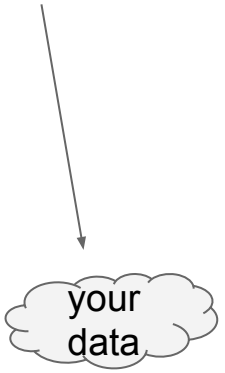
Feature extraction & Fine-tuning

Feature extraction

1. Use AlexNet architecture pre-trained on ImageNet

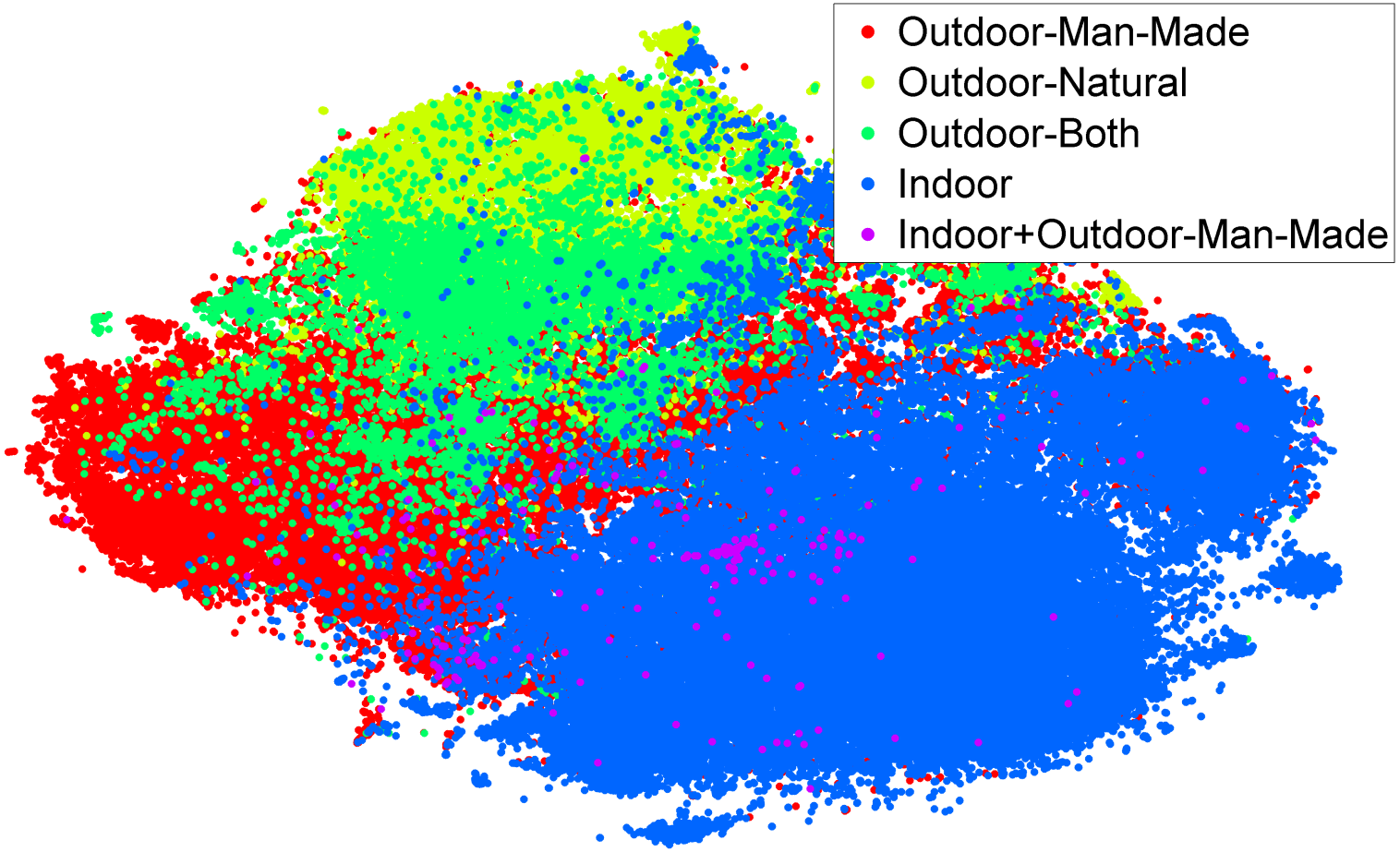
2. Extract features (from the fc6 or fc7 layer) using your own data and train a classifier





J Donahue et al, DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition, ICML 2014

Feature extraction

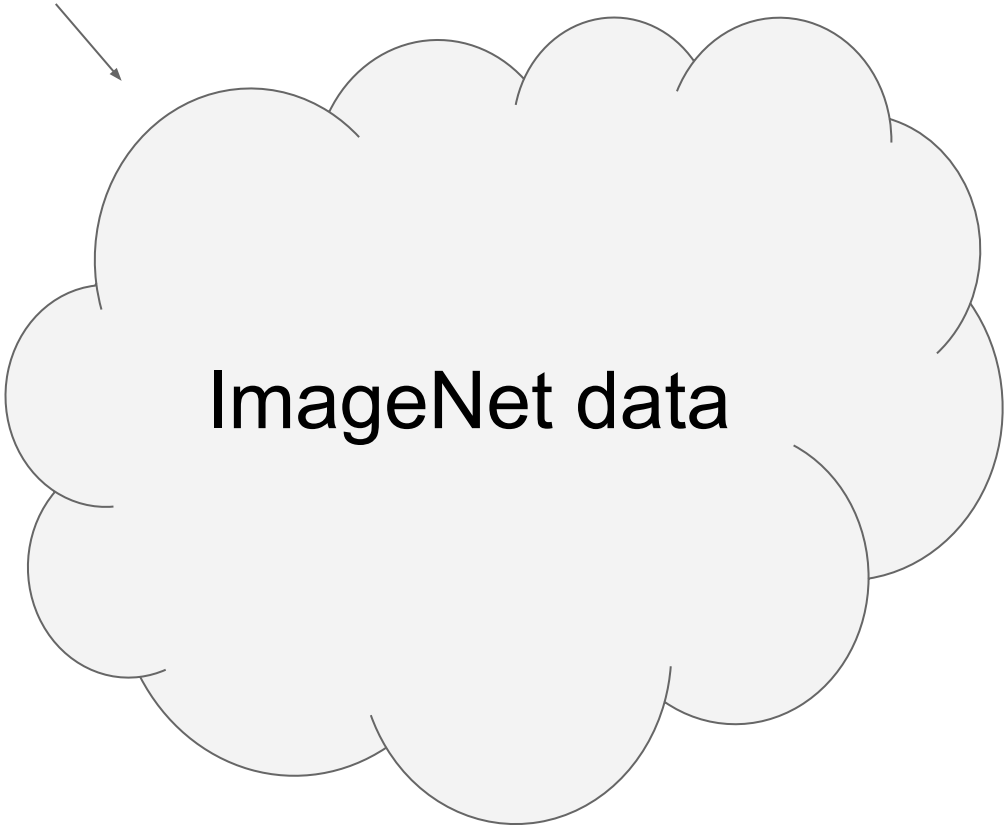


AlexNet fc6 features (trained on ImageNet) generalize to SUN-397 scene dataset when considering semantic groupings of labels

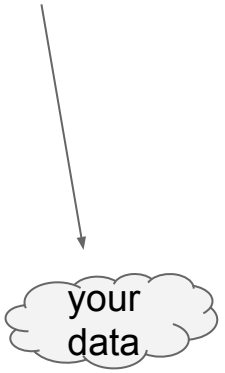
J Donahue et al, DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition, ICML 2014

Fine-tuning

1. Train on ImageNet



2. Finetune network on your own data



Take Home Messages

- Understanding training and testing of convolutional neural networks
- How to prevent overfitting in CNNs
- Visualizing CNNs

Credits

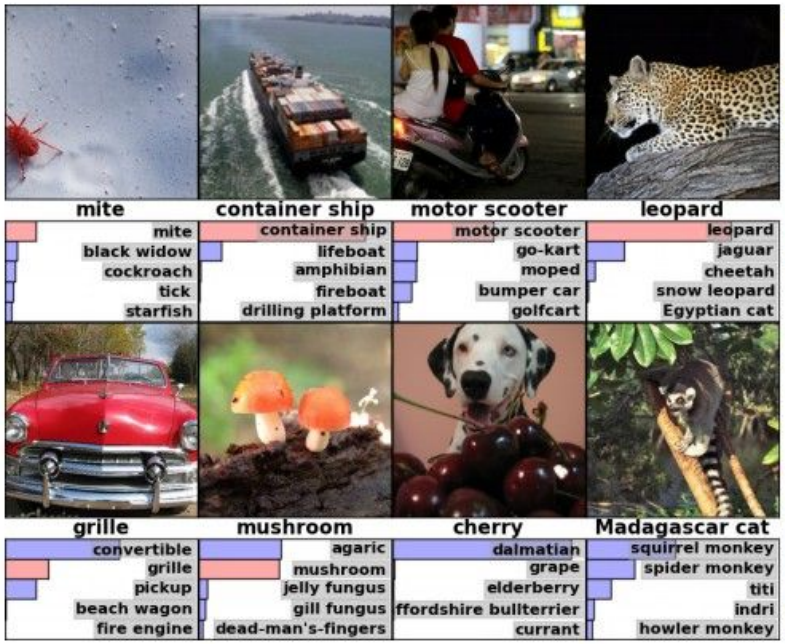
Many of the pictures, results, and other materials are taken from:

Ruslan Salakhutdinov
Joshua Bengio
Geoffrey Hinton
Yann LeCun
Barnabás Póczos
Aarti Singh
Fei-Fei Li
Andrej Karpathy
Justin Johnson
Rob Fergus
Adriana Kovashka
Leon Bottou

Appendix

Fast-forward to today: ConvNets are everywhere

Classification



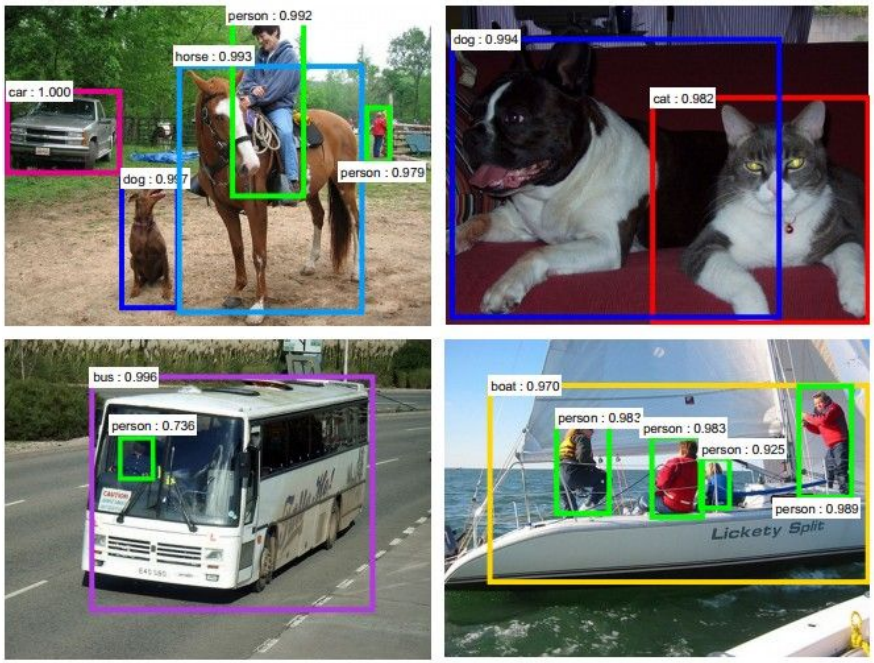
Retrieval

[Krizhevsky 2012]

Appendix

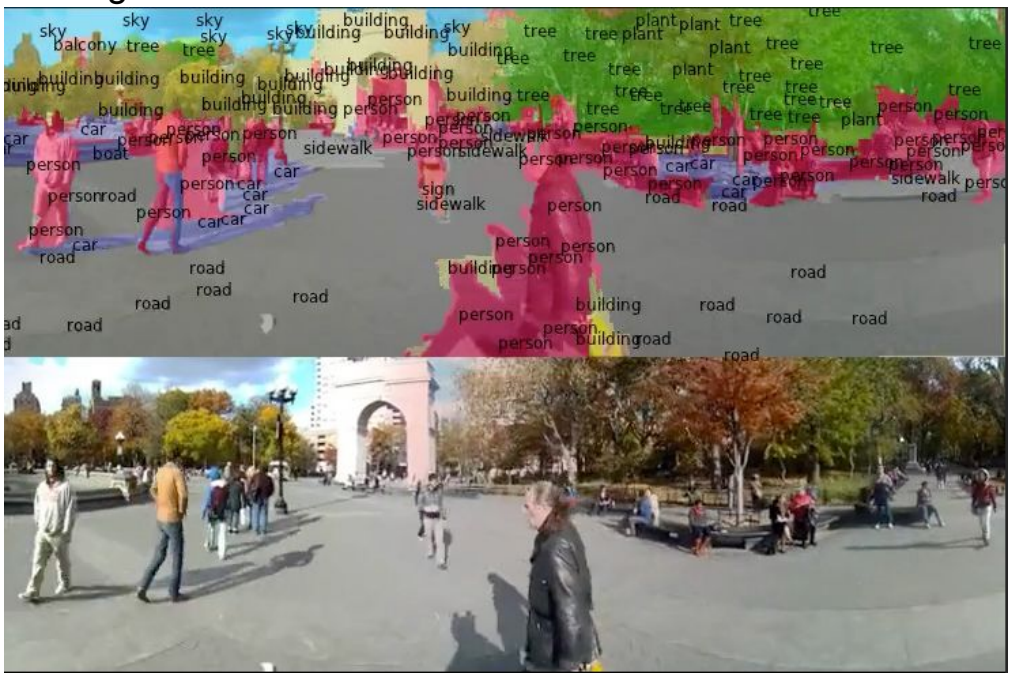
Fast-forward to today: ConvNets are everywhere

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

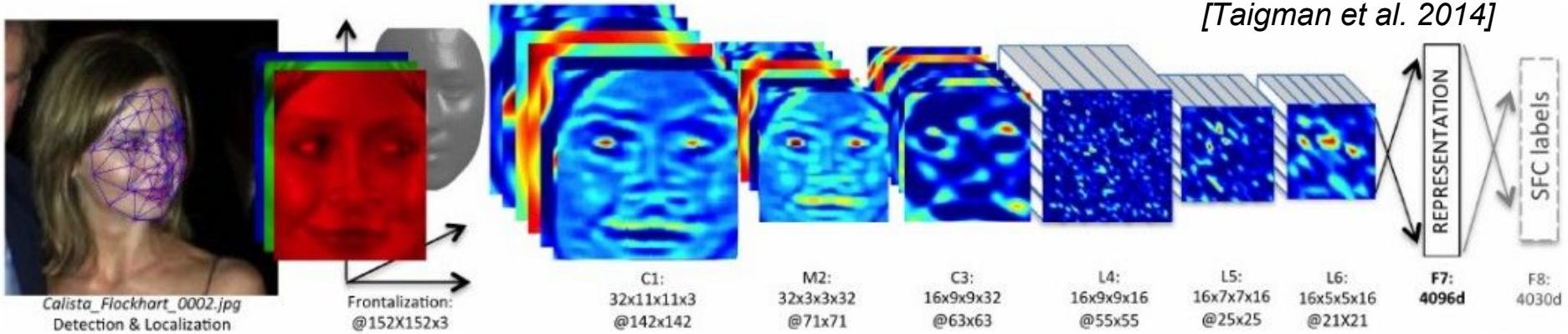


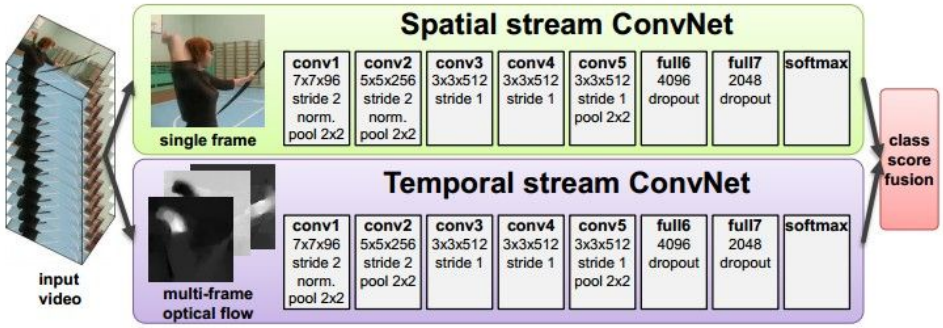
[Farabet et al., 2012]

Appendix

Fast-forward to today: ConvNets are everywhere

[Taigman et al. 2014]





[Simonyan et al. 2014]

[Goodfellow 2014]

Appendix

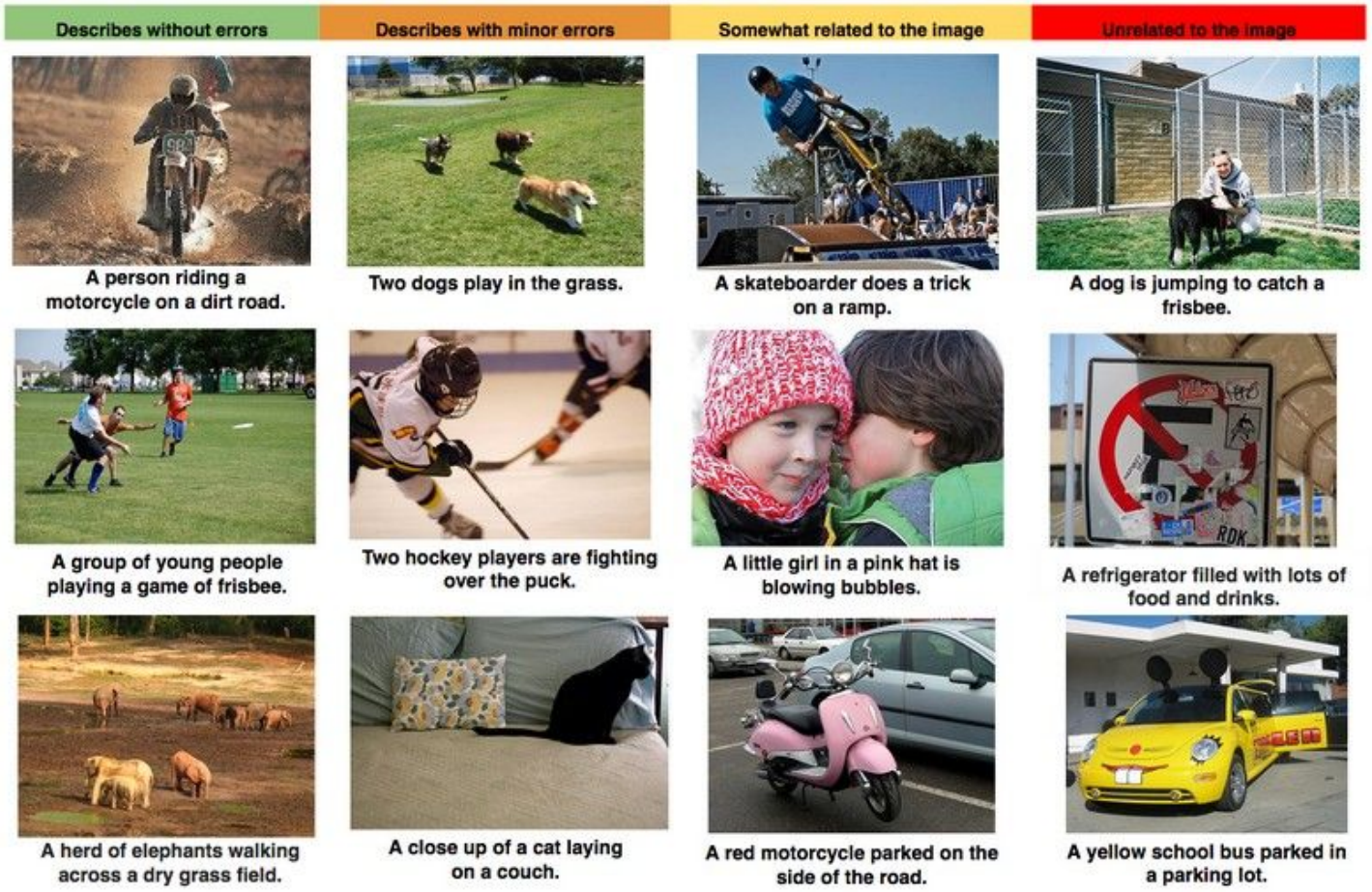


Image Captioning

[Vinyals et al., 2015]

Appendix

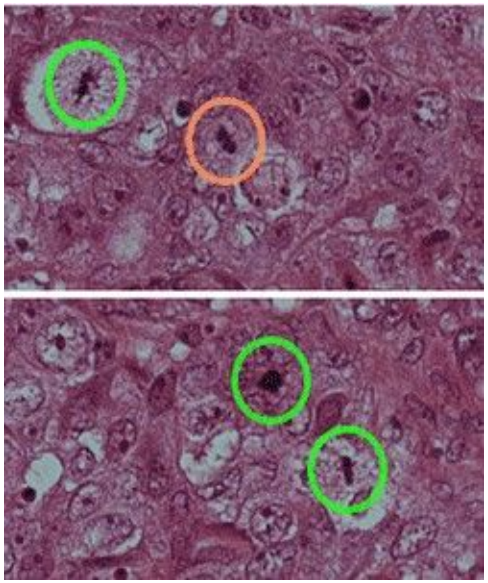
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]

Appendix

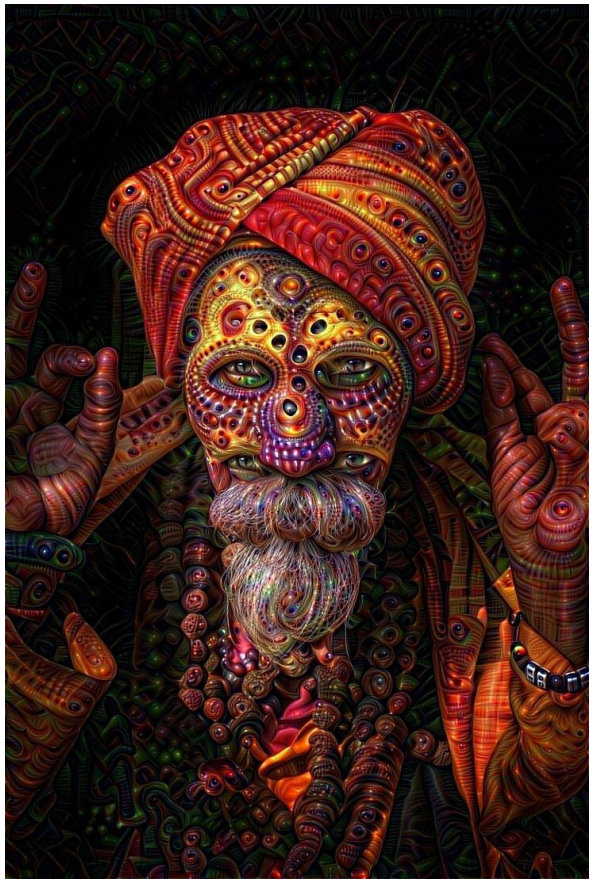
Fast-forward to today: ConvNets are everywhere

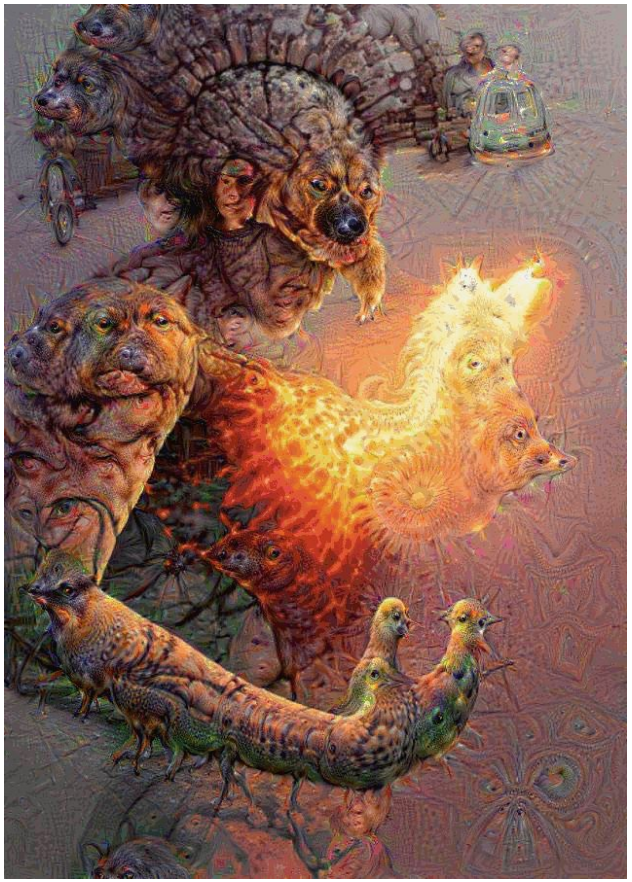


[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]

Appendix





reddit.com/r/deepdream

Appendix

- Resources

- Deep Learning course at Stanford:
  http://cs231n.stanford.edu/syllabus.html

- Course at Universite de Sherbrooke:
  http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html

- Deep Learning summer school 2015:
  http://videolectures.net/deeplearning2015_montreal/

- Deep learning resources:
  http://deeplearning.net/