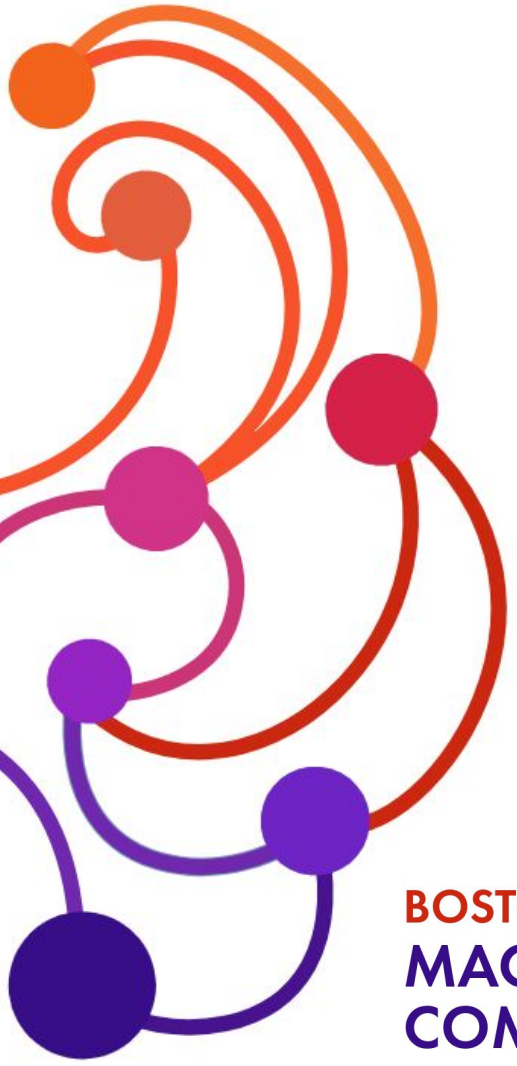


Style Transfer with TensorFlow™

BOSTON UNIVERSITY
MACHINE INTELLIGENCE COMMUNITY

Rachel Manzelli
3/28/18

https://goo.gl/bGiujV

Last Time

- ➔ Learned the concepts of TensorFlow 1.2
- ➔ Built and evaluated simple graphs
- ➔ “Trained” a linear model

https://github.com/bumic/TF-Workshops/tree/master/Workshop%201

Today's Plan

- ➔ *Begin project:* implementation of neural style transfer in TensorFlow (will last 3 workshops)
- ➔ *Concepts:* convolutional neural networks, VGG-19 architecture, neural style transfer
- ➔ *Implementations:* structure of a neural network implementation, skeleton code for NST

MIC

What You Need For Today

- A computer & text editor
- Installations of Python 2 or 3, TensorFlow, numpy, scipy, and Pillow
- These can all be installed via `pip install`

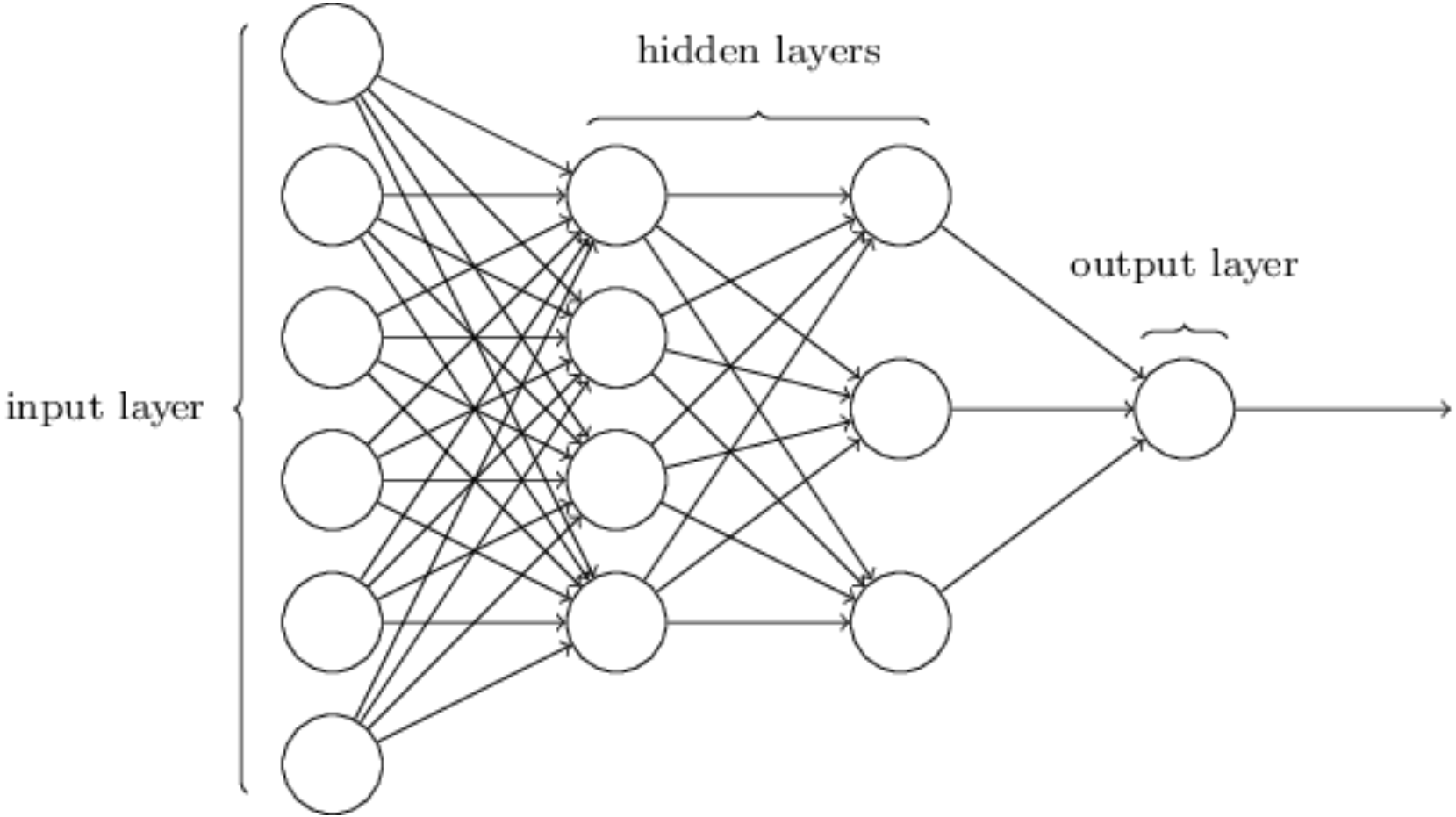
```
>> pip install numpy

>> pip install scipy

>> pip install Pillow
```

MIC

A Brief Overview of Neural Networks

- **Multilayer perceptrons:** the "plain vanilla" of machine learning
- In a fully connected network, each **neuron** (a tensor in TensorFlow) in the next layer is connected to all neurons in the previous layer, and **feeds forward** information to "activate" the next neuron
- What is this information?

MIC


hidden layers
output layer
input layer

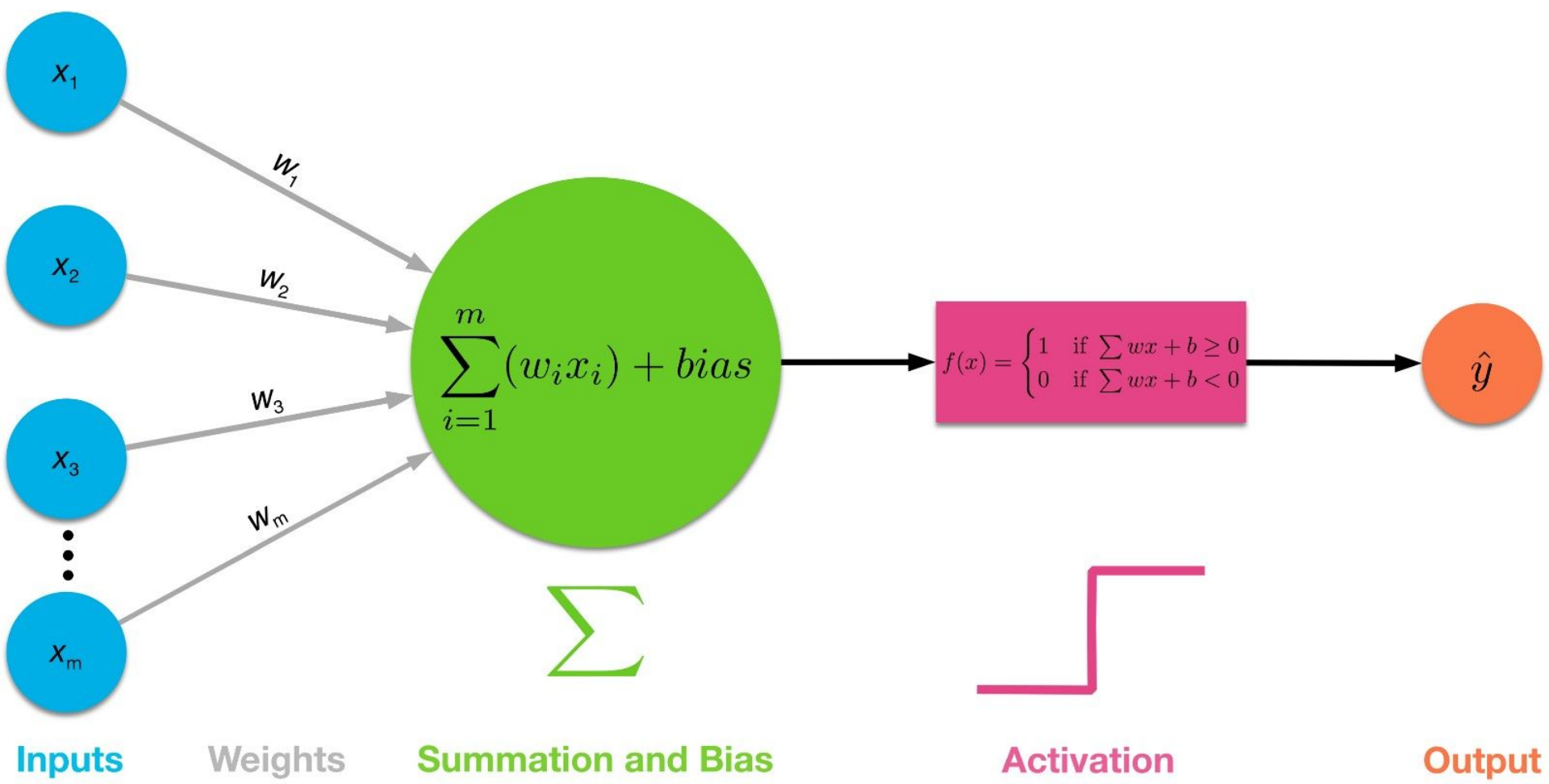
Neural Networks: Connections

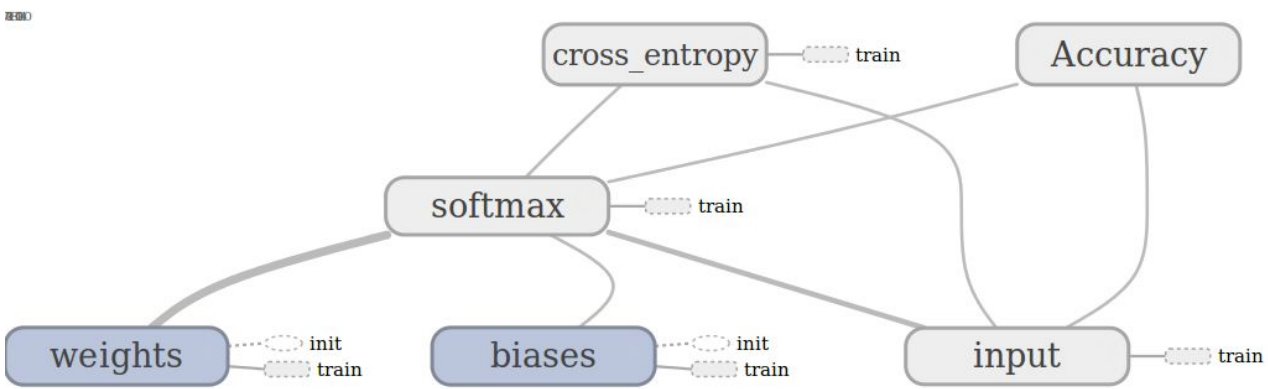
- **Weights**: how "important" that neuron is
- **Bias**: how high the weighted sum needs to be for neuron to be meaningfully active
- Initialized randomly, then through training they are adjusted toward the correct value

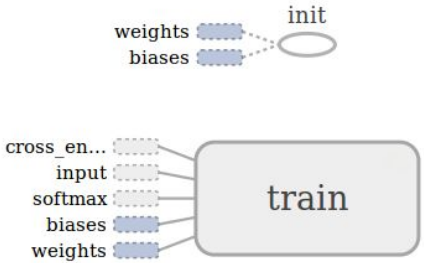
$$Y = \sum (weight * input) + bias$$

```
linear_model = W * x + b
```

MIC


x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output


cross_entropy
train
Accuracy
softmax
train
weights
init
train
biases
init
train
input
train


init
weights
biases
cross_en...
input
softmax
biases
weights
train

MIC

Neural Networks: Loss functions

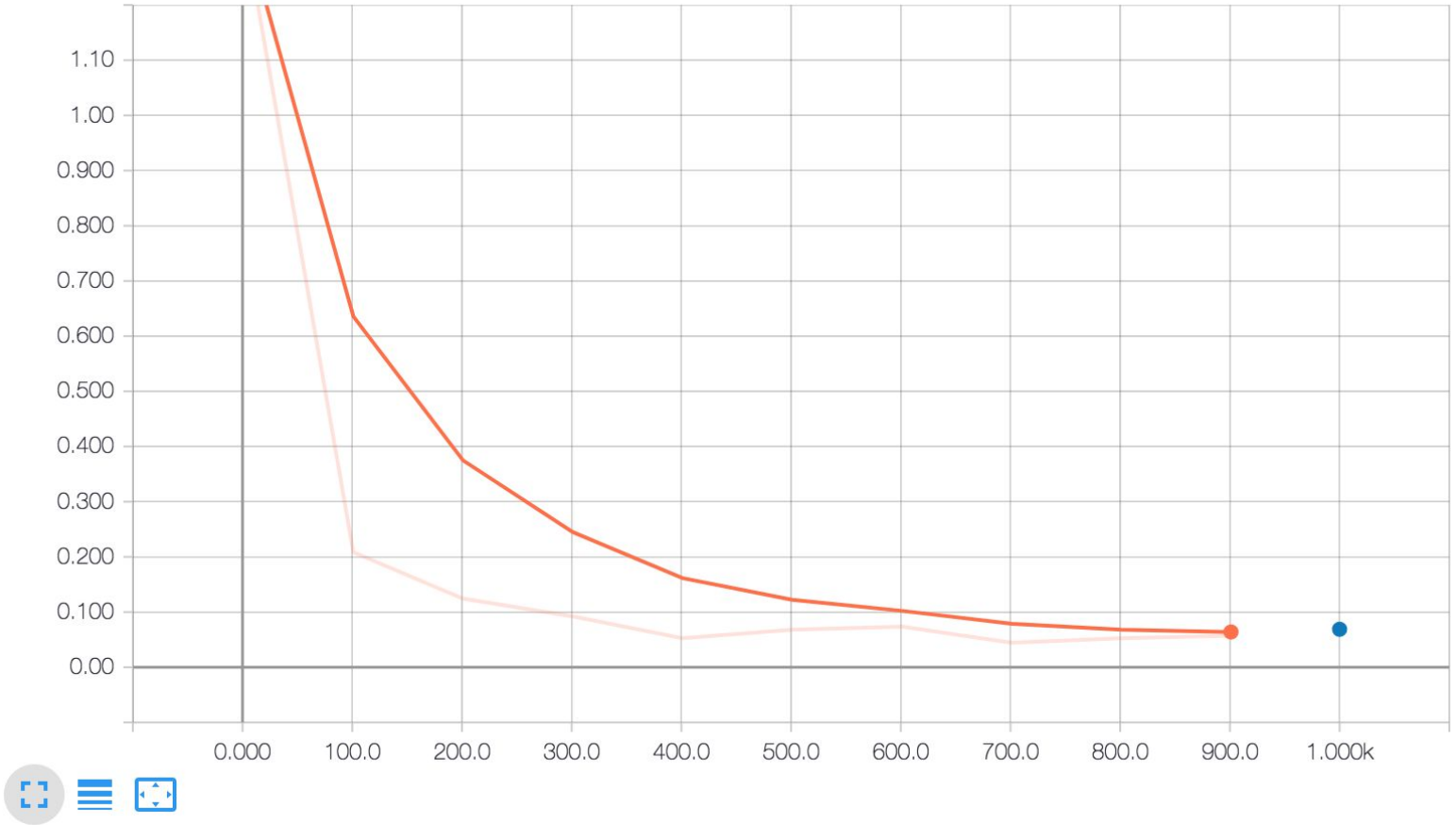
- **Loss functions** add a cost to the network's output, i.e. penalizing the network for performing poorly
  - Difference between output and desired output
  - We want to **minimize** this. There are many loss functions...

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

$$\mathcal{L}(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

MIC

loss


1.10
1.00
0.900
0.800
0.700
0.600
0.500
0.400
0.300
0.200
0.100
0.00
0.000
100.0
200.0
300.0
400.0
500.0
600.0
700.0
800.0
900.0
1.000k


M I C

Neural Networks: Training

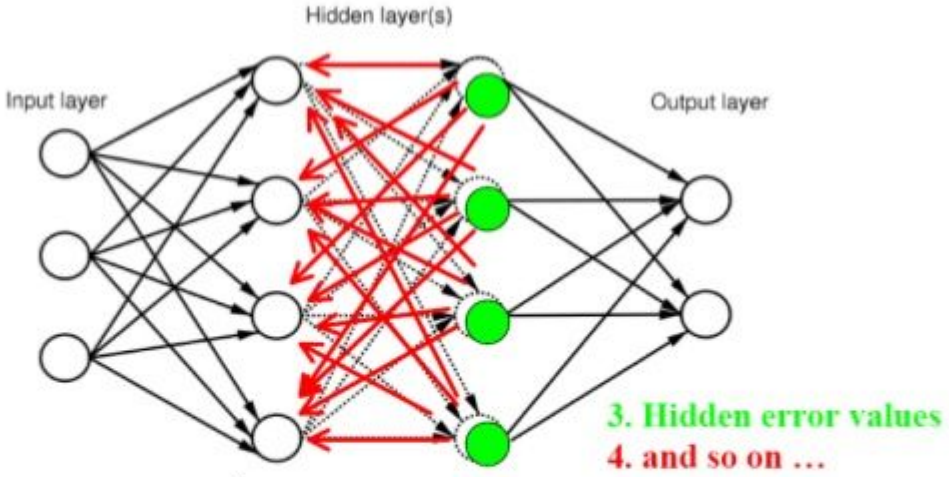
- ➔ **Training** a network means minimizing this cost function more and more with each step.
- ➔ **Gradient descent** is called an optimizer: it allows us to **adjust weights and biases** by telling them to increase or decrease to minimize the cost function
- ➔ There are many other optimizers...

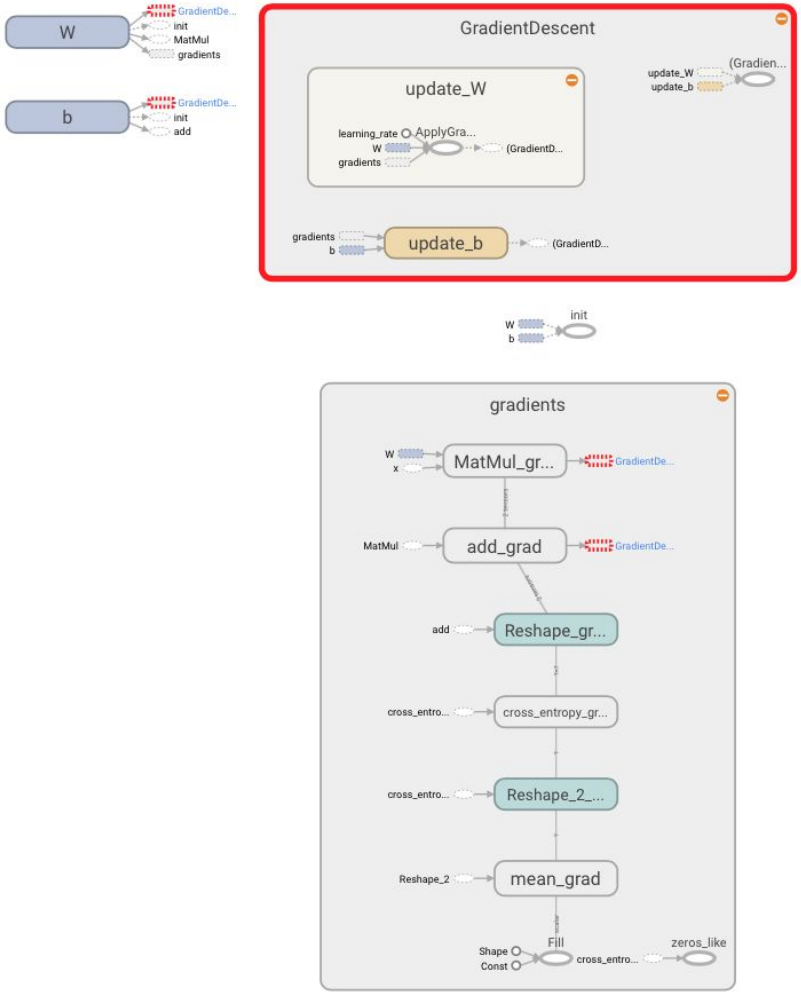
$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

MIC

Visual: Backpropagation




W
GradientDe...
init
MatMul
gradients
b
GradientDe...
init
add
GradientDescent
update_W
learning_rate
W
gradients
ApplyGra...
(GradientD...
update_W
update_b
(Gradien...
gradients
b
update_b
(GradientD...
W
b
init
gradients
W
x
MatMul_gr...
GradientDe...
MatMul
add_grad
GradientDe...
add
Reshape_gr...
cross_entro...
cross_entropy_gr...
cross_entro...
Reshape_2_...
Reshape_2
mean_grad
Shape
Const
Fill
cross_entro...
zeros_like

François Chollet 

@fchollet

Follow

"Neural networks" are a sad misnomer. They're neither neural nor even networks. They're chains of differentiable, parameterized geometric functions, trained with gradient descent (with gradients obtained via the chain rule). A small set of highschool-level ideas put together

11:58 AM - 12 Jan 2018

1,308 Retweets **3,413** Likes

116 1.3K 3.4K

François Chollet
@fchollet | 81,895 followers

Activity | Map | Analyse | Visualise | Alert

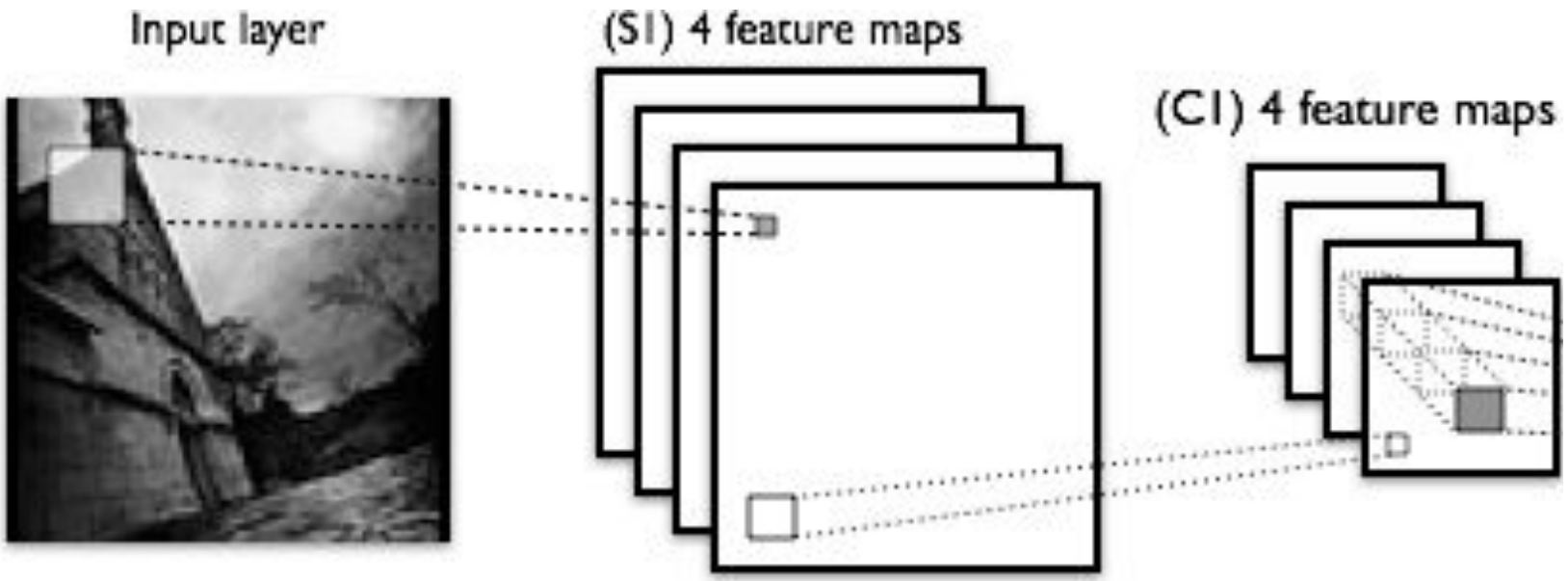
There's a point to be made that a neural network for food classification that can classify Tide pods as "not food" has already reached superhuman capabilities

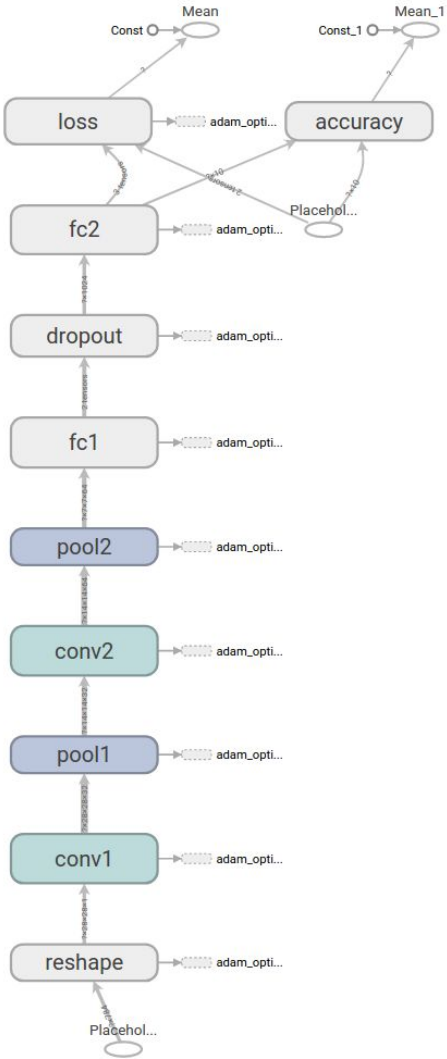
2 months ago · *Twitter Web Client* · *en*

MIC

Convolutional Neural Networks

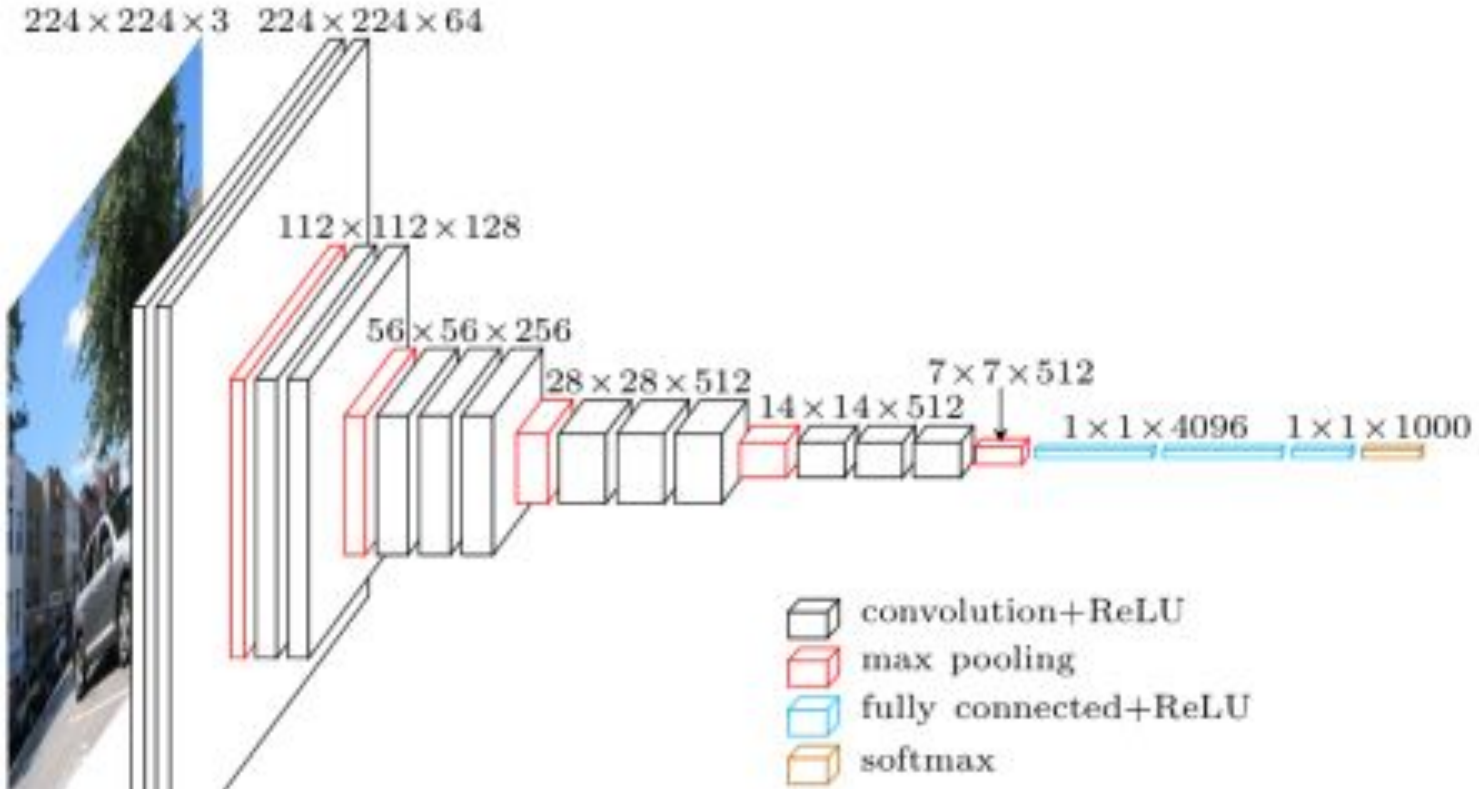
- Neural networks with **convolutional layers**: use convolutions instead of matrix multiplications to produce output in hidden layers
- Each convolution layer takes an input image and produces a set of **feature maps** by **filtering**
  - Each filter is “convolved” across the width and height of the input volume, producing a 2-dimensional activation map of that filter.

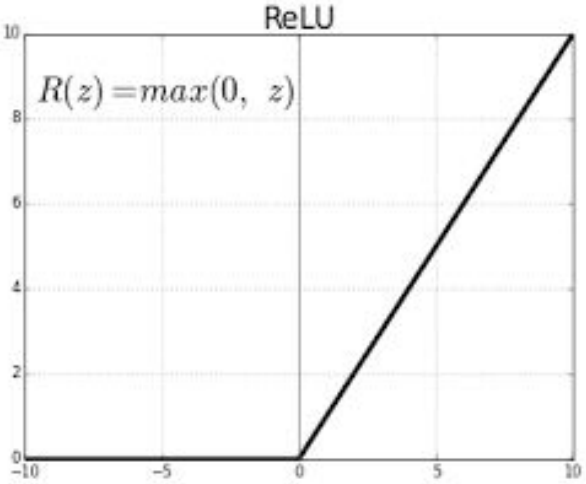

Input layer
(S1) 4 feature maps
(C1) 4 feature maps


Mean
Const
Mean_1
Const_1
loss
adam_opti...
accuracy
Placehol...
fc2
adam_opti...
dropout
adam_opti...
fc1
adam_opti...
pool2
adam_opti...
conv2
adam_opti...
pool1
adam_opti...
conv1
adam_opti...
reshape
adam_opti...
Placehol...

VGG-19

- A specific version of VGGNet with **19 weight layers**
- VGGNet is a convolutional neural network originally used for image classification
  - uses **3×3 convolutional layers** stacked on top of each other
  - reducing volume size is handled by **max pooling**
  - Two **fully-connected** layers, each with 4,096 nodes are followed by a **softmax classifier**.
- Original VGGNet paper: https://arxiv.org/abs/1409.1556

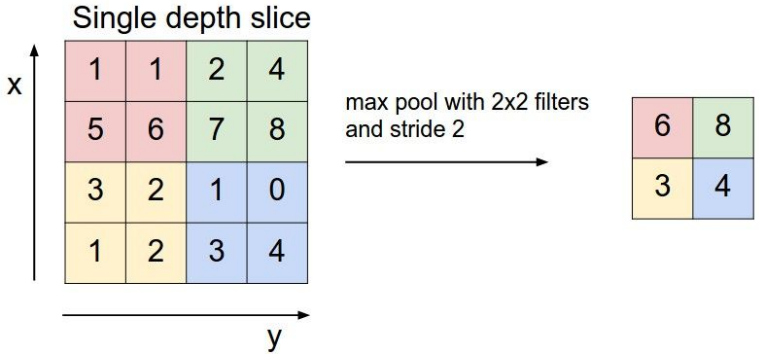

224×224×3
224×224×64
112×112×128
56×56×256
28×28×512
14×14×512
7×7×512
1×1×4096
1×1×1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax



Softmax

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

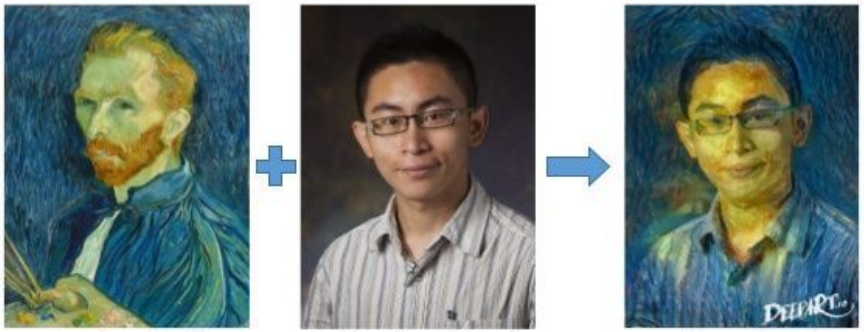
Max pooling



MIC

What is Neural Style Transfer?

- ➔ Representations of style and content can be isolated and manipulated separately from a convolutional NN!
- ➔ Contains three main components:
  - ◆ **Content image, style image, pastiche**



Source image (**Style**) Target image (**Content**) Output (deepart)

A Neural Algorithm of Artistic Style [Gatys et al. 2015]

MIC

Content image:
Picture we want to transfer style onto

Style image:
Artwork whose style we want to transfer

Pastiche:
Final stylized image (Initialized to random/white noise)

MIC

Content image: loss and reconstruction

- **Content Loss** - Mean Squared Error between feature maps for a given convolutional layer l
  - P_{ij}^{l}: Feature map of **original image**
  - F_{ij}^{l} : Feature map of **image content to be generated**

- Gradient descent is used to minimize the content loss

$$L_{content}(\overrightarrow{p}, \overrightarrow{x}, l) = \frac{1}{2} \sum_{ij} (F_{ij}^{l} - P_{ij}^{l})^2$$

Original image

Generated image

Style image: loss function

- Style Loss
  - **Gram Matrix**: inner product between two vectorised features maps *i* and *j*
  - Computes correlations between different features

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

  - Minimize mean-squared distance between entries of Gram matrix of style image and generated image

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^{l} - A_{ij}^{l})^2$$

Style image: reconstruction

- **Total Style Loss** across multiple layers
  - Weighted sum of styles losses across all layers of the convolutional network

$$L_{style}(\overrightarrow{a}, \overrightarrow{x}) = \sum_{l=0}^{L} w_l E_l$$

Original image

Generated image

Layer contribution weight

Total Loss Function for NST

- **Total Loss** - Jointly minimize error of a white noise image from
  - Content representation of **content image** $\vec{p}$
  - Style representation of **style image** $\vec{a}$

$$L_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha L_{content}(\vec{p}, \vec{x}) + \beta L_{style}(\vec{a}, \vec{x})$$

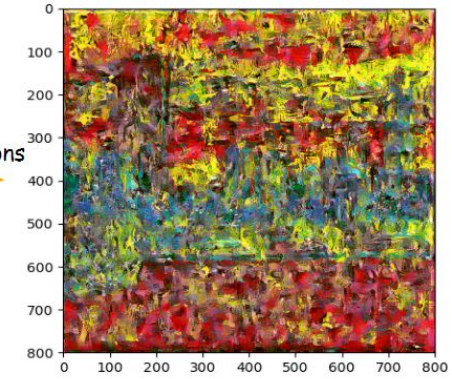
- α, β : weighting factors for content and style reconstruction respectively

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \left(\alpha \mathcal{L}_{\text{content}}(\mathbf{c}, \mathbf{x}) + \beta \mathcal{L}_{\text{style}}(\mathbf{s}, \mathbf{x}) \right)$$

MIC

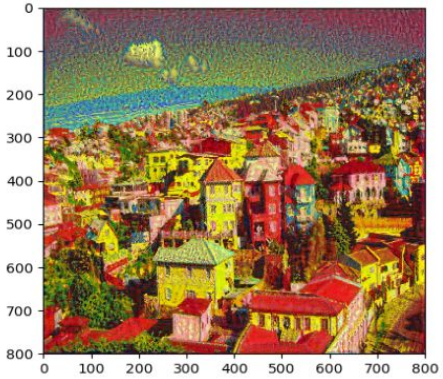
+

1000 iterations →



+

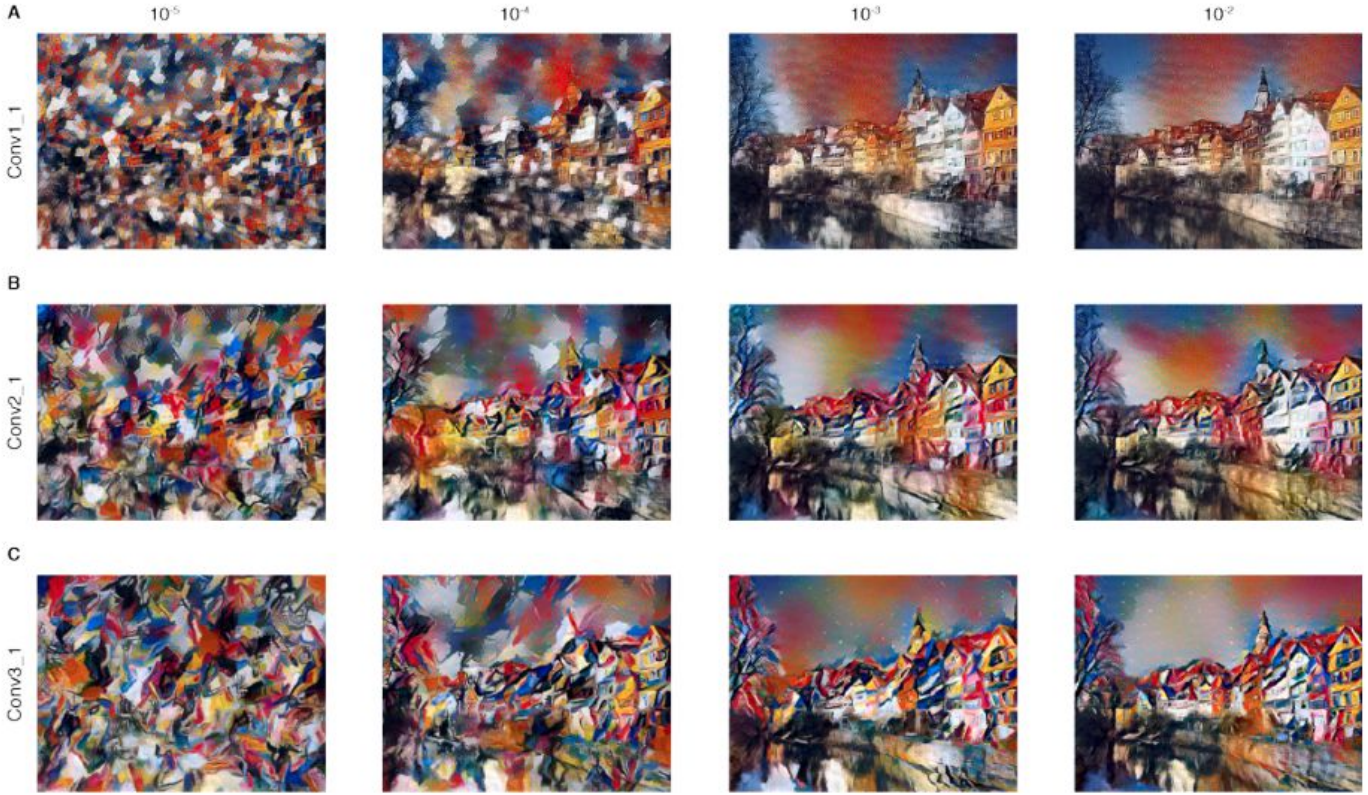
1000 iterations →



MIC

Style emphasis ← α/β ratio → Content emphasis

VGG content layer



Implementation of Neural Style Transfer

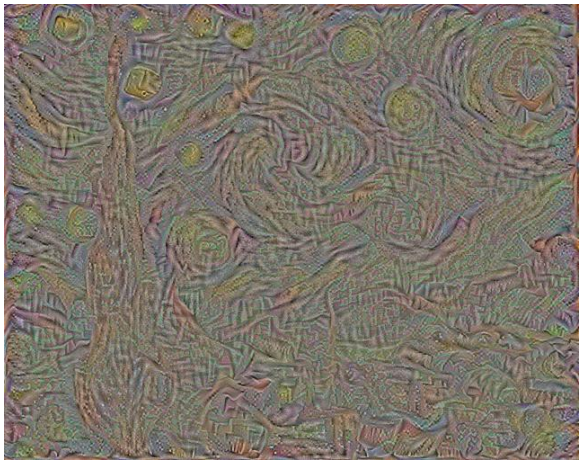
- The implementation of neural style we will be using is located here:

https://goo.gl/M1pBRx

- Let's go to it and play around!

MIC

Implementation of Neural Style Transfer



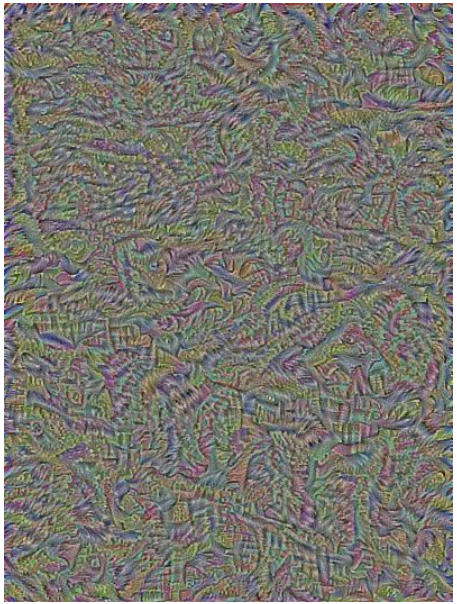
Iteration 10

Iteration 300

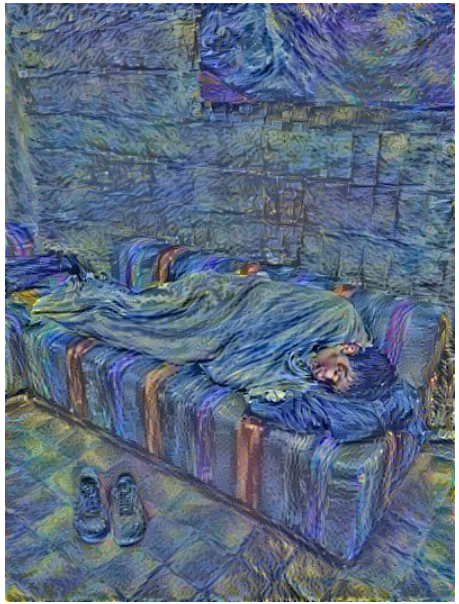
Iteration 990

MIC

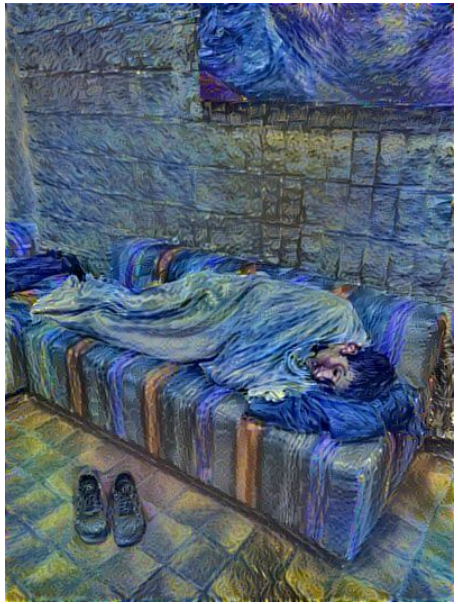
Implementation of Neural Style Transfer



Iteration 10



Iteration 300



Iteration 990

MIC

Let's (finally) start building it!

- ➔ Go to https://github.com/anishathalye/neural-style
- ➔ This is the NST implementation we will be rebuilding.
- ➔ Download the file at the bottom:

```
imagenet-vgg-verydeep-19.mat
```

- ➔ This is our pretrained VGGNet that we will be using in our implementation. (pre-training a convnet is time-consuming!)

MIC

The Pre-trained Network

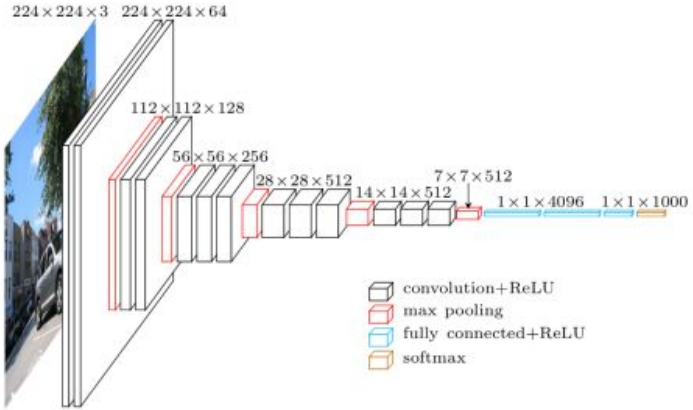
- ➔ This network was trained on ImageNet (it can presumably classify images)
- ➔ Open the file in MATLAB
- ➔ If you don't have MATLAB, that's ok - Python to come!

Implementing a Neural Network

- ➔ *Architecting*: constructing the architecture of the neural network
- ➔ *Preprocessing*: preprocessing the data so that it is in a form we can train on
- ➔ *Training*: getting those perfect weights
- ➔ *Testing/generating*: using the trained network to generate a result

Implementation: Architecture

- ➔ Open a new file. This will be our initialization of the architecture of the network.
- ➔ First, we will initialize a dictionary of keys to refer back to the weight layers of our network, based on the .mat file.



MIC

Upcoming Events with BUMIC

- ➔ Impact of AI discussion group: 4/4/18
- ➔ Next TensorFlow session: 4/11/18 @ 7pm
  - ◆ Continuing NST: Finishing architecture and training the network

MIC

Thank you to our sponsors!