CREATIVE PROGRAMMING AND COMPUTING

Lab 6:

Deep Learning

Luca Comanducci

luca.comanducci@polimi.it

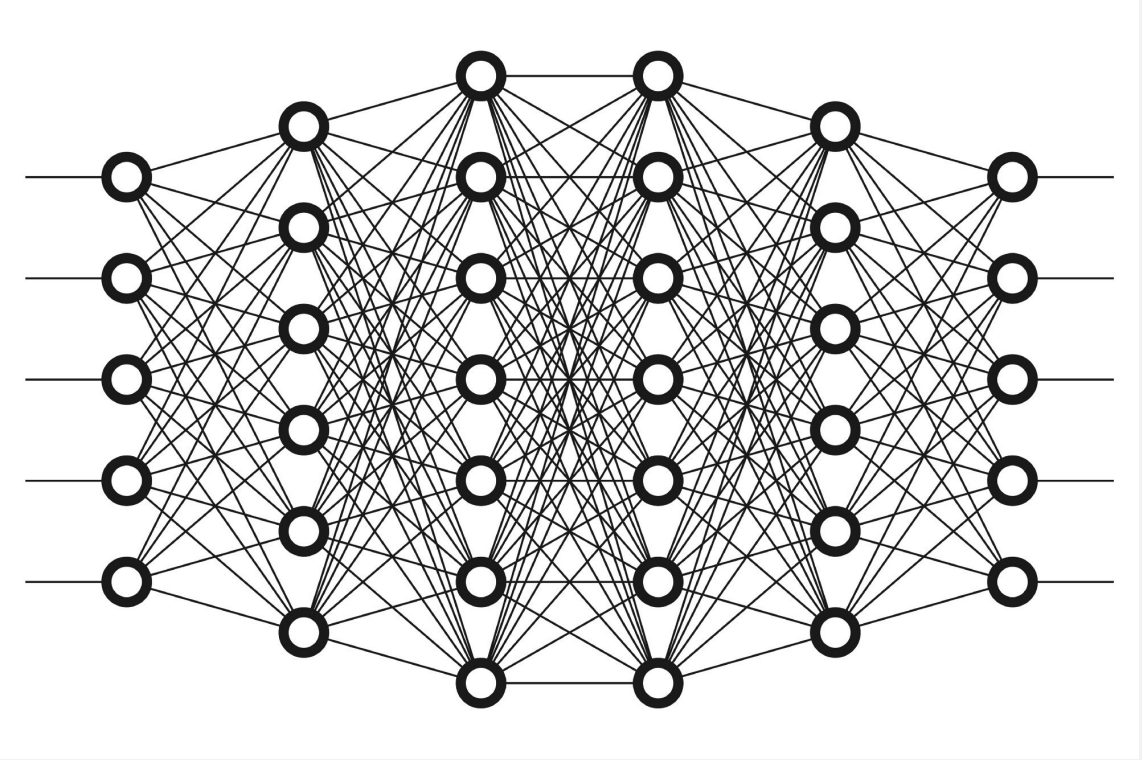
CONTENTS

DEEP LEARNING

- **Deep Learning is representation learning**
- It is based on the small computation blocks called neural networks

- Neural Networks are organized into layers

- Deep learning methods automatically learn the representations (features) needed to fullfill the task considered (e.g. classification/ regression)



DEEP LEARNING FRAMEWORKS

- ***Tensorflow***

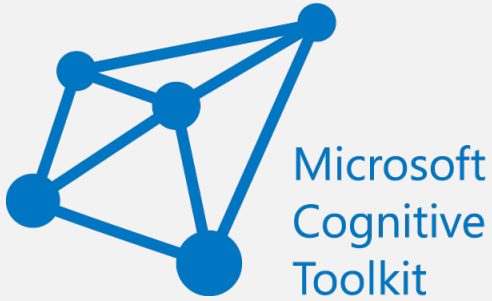
- ***Keras***

- ***PyTorch***

- **JAX**

- ***Caffe***

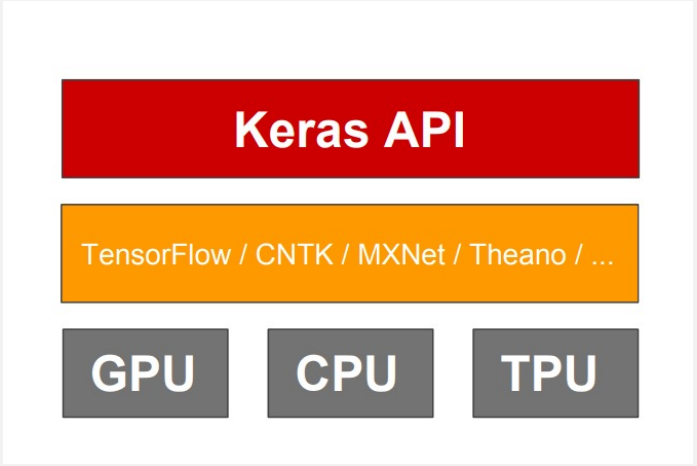
- ***Microsoft Cognitive Toolkit***



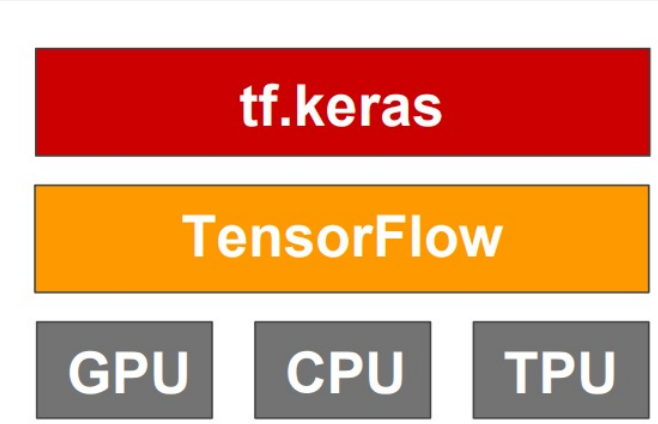
INTRODUCTION TO KERAS AND TENSORFLOW

INTRODUCTION TO KERAS AND TENSORFLOW

- **High-level neural networks API**
  - Written in Python
  - Runs on top of Tensorflow, Theano and CNTK
  - User – friendly
  - Runs both on CPU, GPU or TPU



- **Official high-level API of TensorFlow 2.0**
  - tensorflow.keras module
  - Better optimized for use with tensorflow
  - Better integration with tensorflow specific features
    - tf.estimator
    - eager execution



INTRODUCTION TO KERAS AND TENSORFLOW

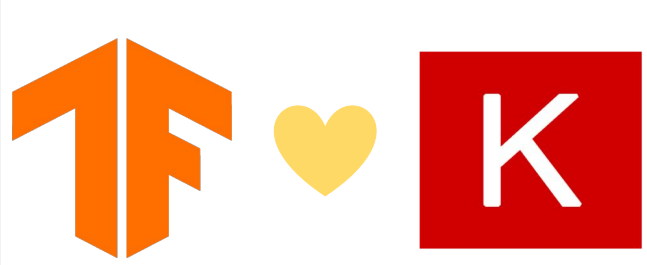
- Keras is built-in in tensorflow 2
- The right way to import is is from tensorflow.keras

```
# !pip install tensorflow==2.0.0-beta1, then

>>> from tensorflow.keras import layers # Right

>>> from keras import layers # Oops

Using TensorFlow backend. # You shouldn't see this
```



INTRODUCTION TO KERAS AND TENSORFLOW

- We will use several types of layers to build models, the most importants are
  - Fully connected layers `tf.keras.layers.Dense(units, activation = None, use_bias = True, kernel_initializer = 'glorot_uniform',….)`
  - Convolutional layers `tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,……)`
  - Flatten layers `tf.keras.layers.Flatten()`
    - Activation functions can be e.g. "relu", "sigmoid", "tanh"

- Three model building styles
  - Sequential
  - Functional
  - Subclassing

INTRODUCTION TO KERAS AND TENSORFLOW

- **Sequential-Model**
  - Extremely simple
  - Good only for one-input one-output stacks

```
model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(M,N),name='flatten'),
        tf.keras.layers.Dense(512, activation='relu', name='fc1'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax',name='output')
])
```

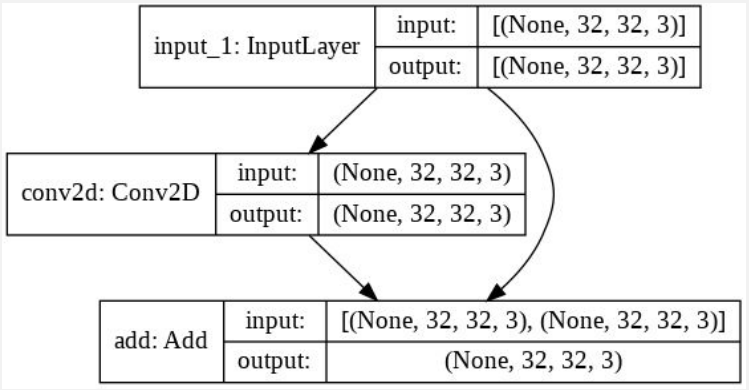
INTRODUCTION TO KERAS AND TENSORFLOW

- **Functional models**
  - Multi-input multi-output arbitrary topologies

```
inputs = keras.Input(shape=(32, 32, 3))

y = layers.Conv2D(3, (3, 3),activation='relu',padding='same')(inputs)
outputs = layers.add([inputs, y])

model = keras.Model(inputs, outputs)
```



INTRODUCTION TO KERAS AND TENSORFLOW

- **Subclassed Models**
  - Multi-input multi-output arbitrary topologies

```
class MyModel(tf.keras.Model):
        def __init__(self, num_classes=10):
                super(MyModel, self).__init__(name='my_model')
                self.dense_1 = layers.Dense(32, activation='relu')
                self.dense_2 =layers.Dense(num_classes,activation='sigmoid')

        def call(self, inputs):
                # Define your forward pass here
                x = self.dense_1(inputs)
                return self.dense_2(x)
```

INTRODUCTION TO KERAS AND TENSORFLOW

- **Compile**
  - After a model has been defined, it must be compiled, specifying the optimizer, loss, metrics and eventual callbacks

```
model.compile(
        optimizer='adam’,
        loss='sparse_categorical_crossentropy’,
        metrics=['accuracy’]
)
```

INTRODUCTION TO KERAS AND TENSORFLOW

- **Training**
  - Once a model is written and compiled it can be trained in two main different ways
    - **Built-in training loop**

```
model.fit(x_train, y_train, epochs=5)
```

    - **Custom-loop**

```
model = MyModel()
with tf.GradientTape() as tape:
        logits = model(images)
        loss_value = loss(logits, labels)

grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

INTRODUCTION TO KERAS AND TENSORFLOW

- Example of full model:

```
model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

HOW TO RUN THESE EXERCISES?

- You will find some jupyter notebooks on the github of the course
- Jupyter is a web-based interactive development environment for Jupyter notebooks

- You can use Google Colaboratory to run them

- Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud and allows you to access powerful computing resources.

INTRODUCTION TO KERAS

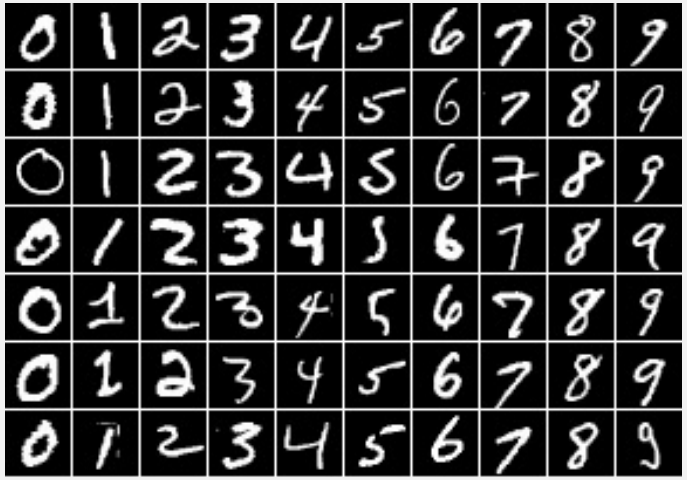
- Basic Classification of clothing images

- You will find the code in the file:

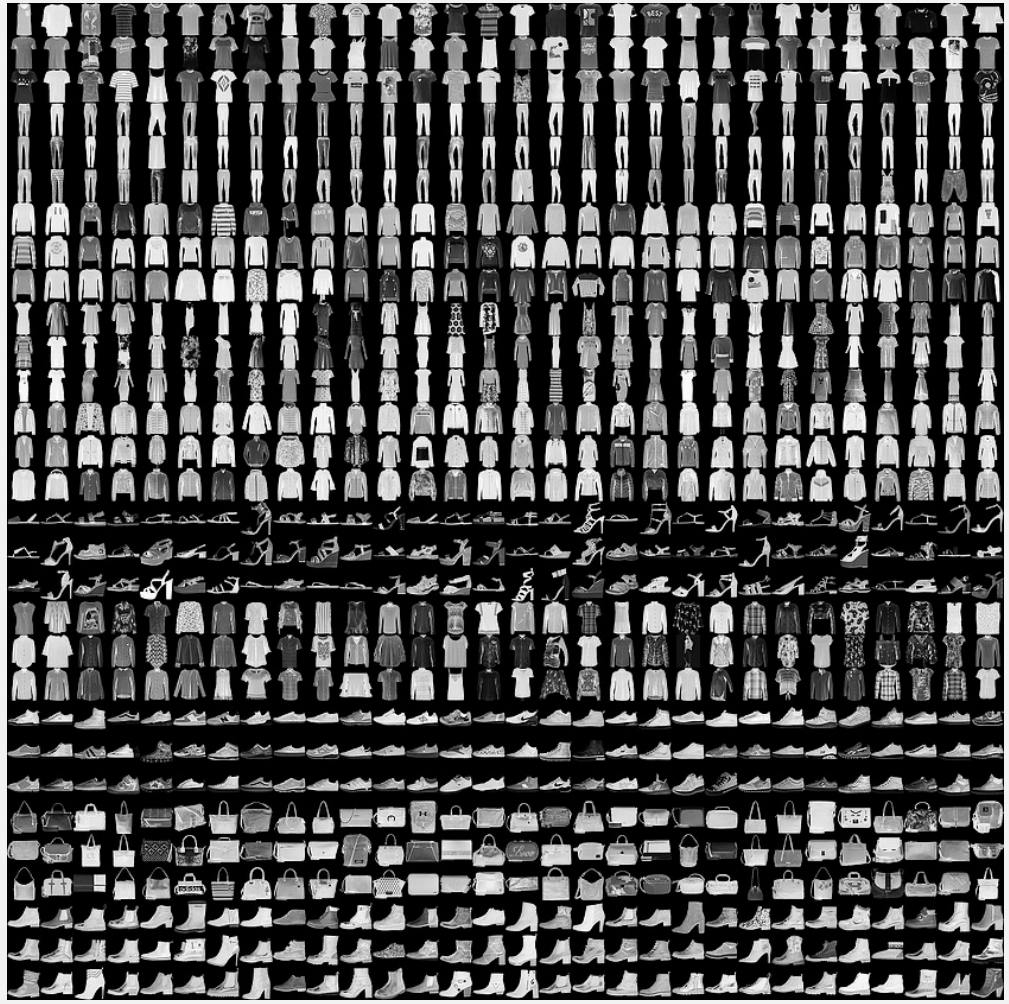
"*Ex.1 - Introduction to Keras - Classification of images of clothing.ipynb*"

CLASSIFICATION OF CLOTHING IMAGES

- We will consider the MNIST fashion dataset
  - 70,000 grayscale images in 10 categories
  - Replacement to MNIST dataset (benchmark for models)
  - MNIST dataset contains images of handwritten digits (0, 1, 2, etc.) in a format identical to that of the articles of clothing



  - You will learn how to train a network based only on feedforward NN, in order to classify the images



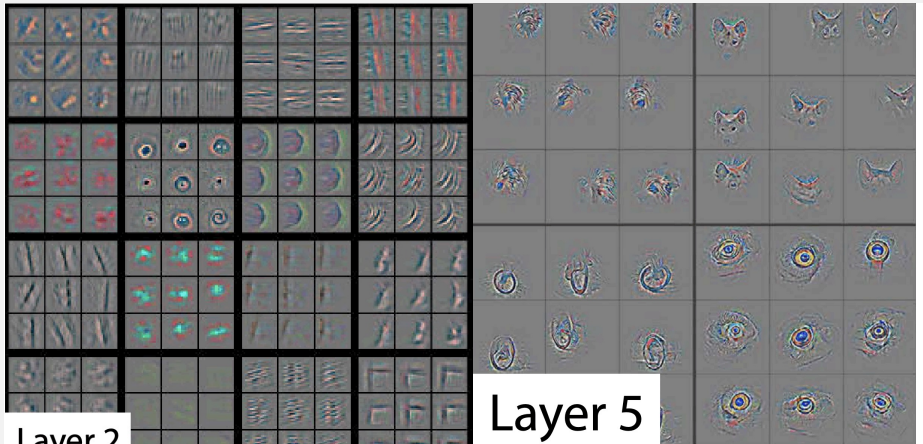
NEURAL STYLE TRANSFER

NEURAL STYLE TRANSFER

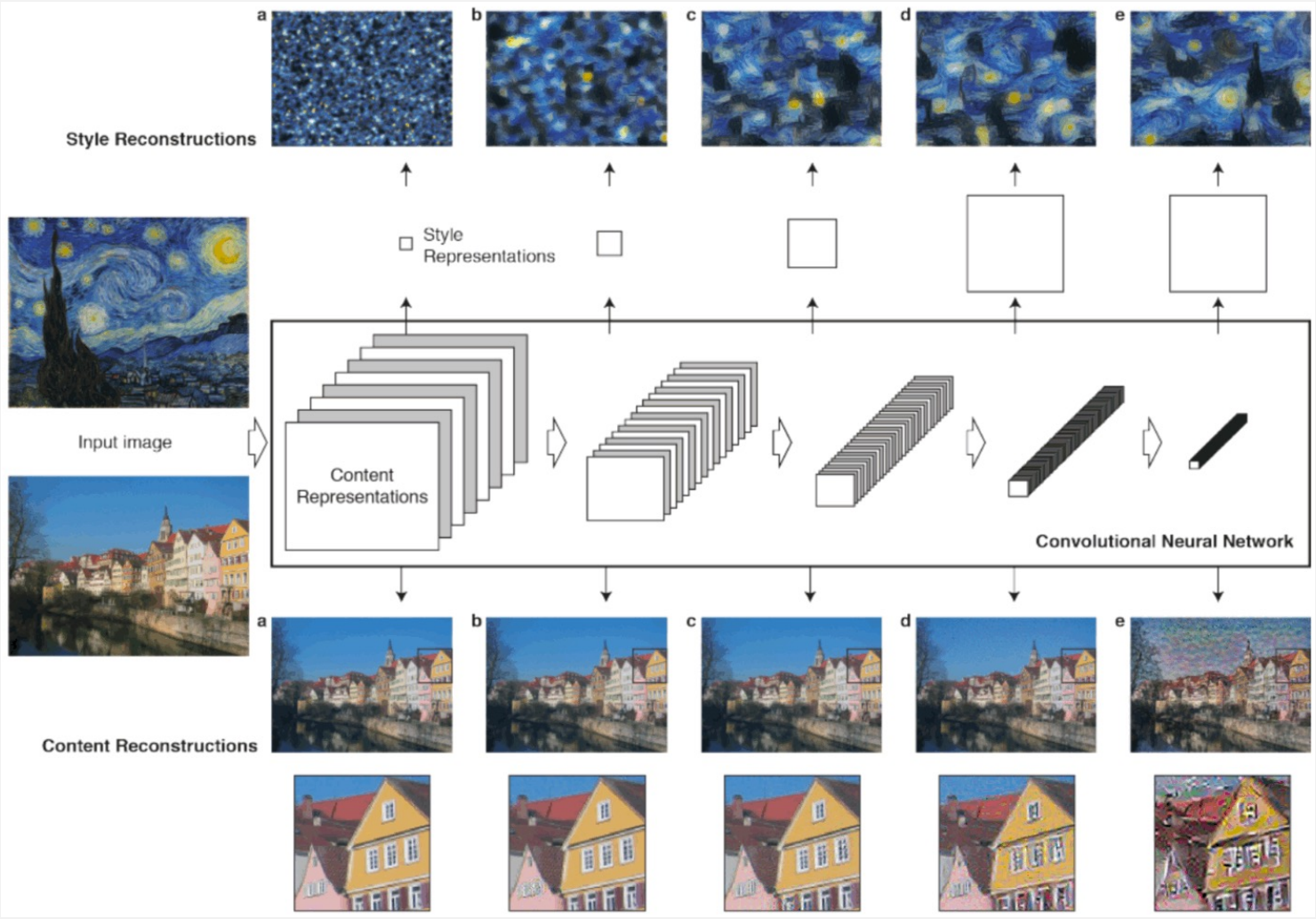
- Deep Neural Network that creates artistic images of high perceptual quality
- Separates and recombines the content and style of an image
- representations of content and style in the Convolutional Neural Networks (CNNs) are separable.

NEURAL STYLE TRANSFER

- CNNs consist of layers of small computational units that process information hierarchically in a feed-forward manner
- Each layer extract a certain feature-maps from the image
- CNNs trained for object recognition
  - Lower layers: learn low-level spatial feature
  - Higher layers: capture high-level content in terms of objects and their arrangement (***Content***)
  - Extracting the correlations between the different filter responses over the spatial extent of the feature maps we obtain a stationary, multi-scale representation of the input image, which captures its texture information but not the global arrangement (***Style***)



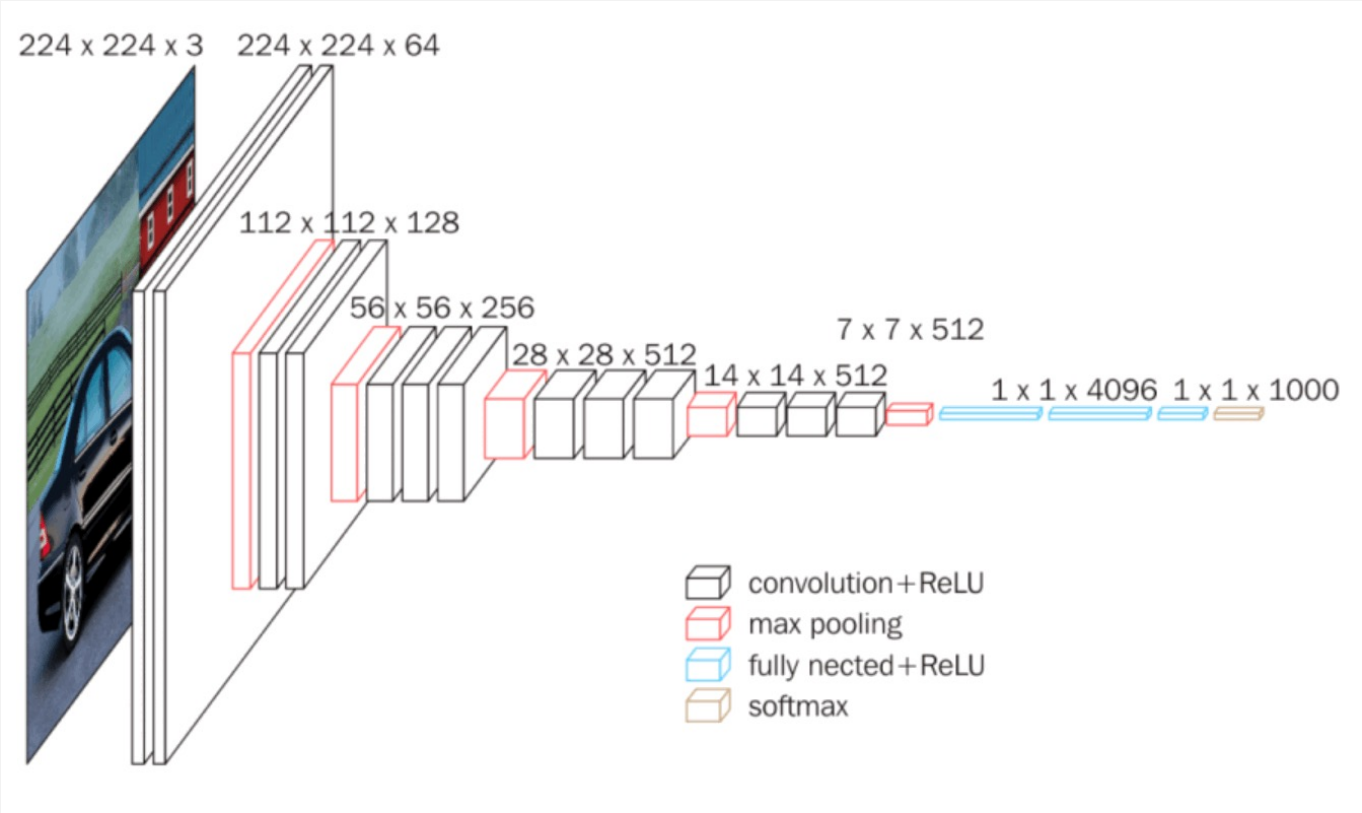
NEURAL STYLE TRANSFER



NEURAL STYLE TRANSFER

- images are synthesised by finding an image that simultaneously matches the content representation of the photograph and the style representation of the respective piece of art

- Features are extracted using the pre-trained classification model **VGG**
- Achieved 92.7 % accuracy in 2014 on the ImageNet dataset (1000 classes, 14 million images)



NEURAL STYLE TRANSFER

- each layer in the network defines a non-linear filter bank whose complexity increases with the position of the layer in the network
- A layer with N_l filters has N_l feature maps, each of size M_l = height*width
- Response in layer l can be stored in a matrix

where F_{ij}^l contains the activation of the i-th filter at position j in layer l

$$F^l \in \mathcal{R}^{N_l \times M_l}$$

NEURAL STYLE TRANSFER

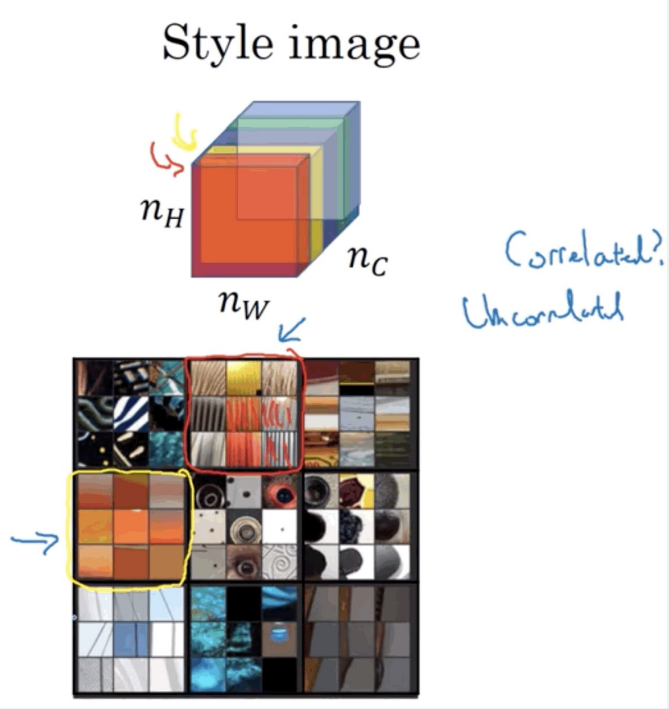
- ***Content loss***:
  - We use gradient descent to recover the content of the image
    - P → original image P → feature-maps
    - x → generated image X → feature-maps
  - We define the loss as the squared error between the two representations

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^{l} - P_{ij}^{l})$$

NEURAL STYLE TRANSFER

- ***Style loss***:
  - Compute the correlation between the different filter responses, where the expectation is taken over the spatial extent of the input image
  - We use the Gram matrix $$G^l = \sum_k F^l_{ik} F^l_{jk}$$
  - Through gradient descent we minimize the mean-squared distances between the Gram matrix of the original image A and of the generated one B

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \frac{1}{4N_l^2 M_l^2} \sum_{l=0}^{L} \sum_{i,j} (G^l_{ij} - A^l_{ij})$$

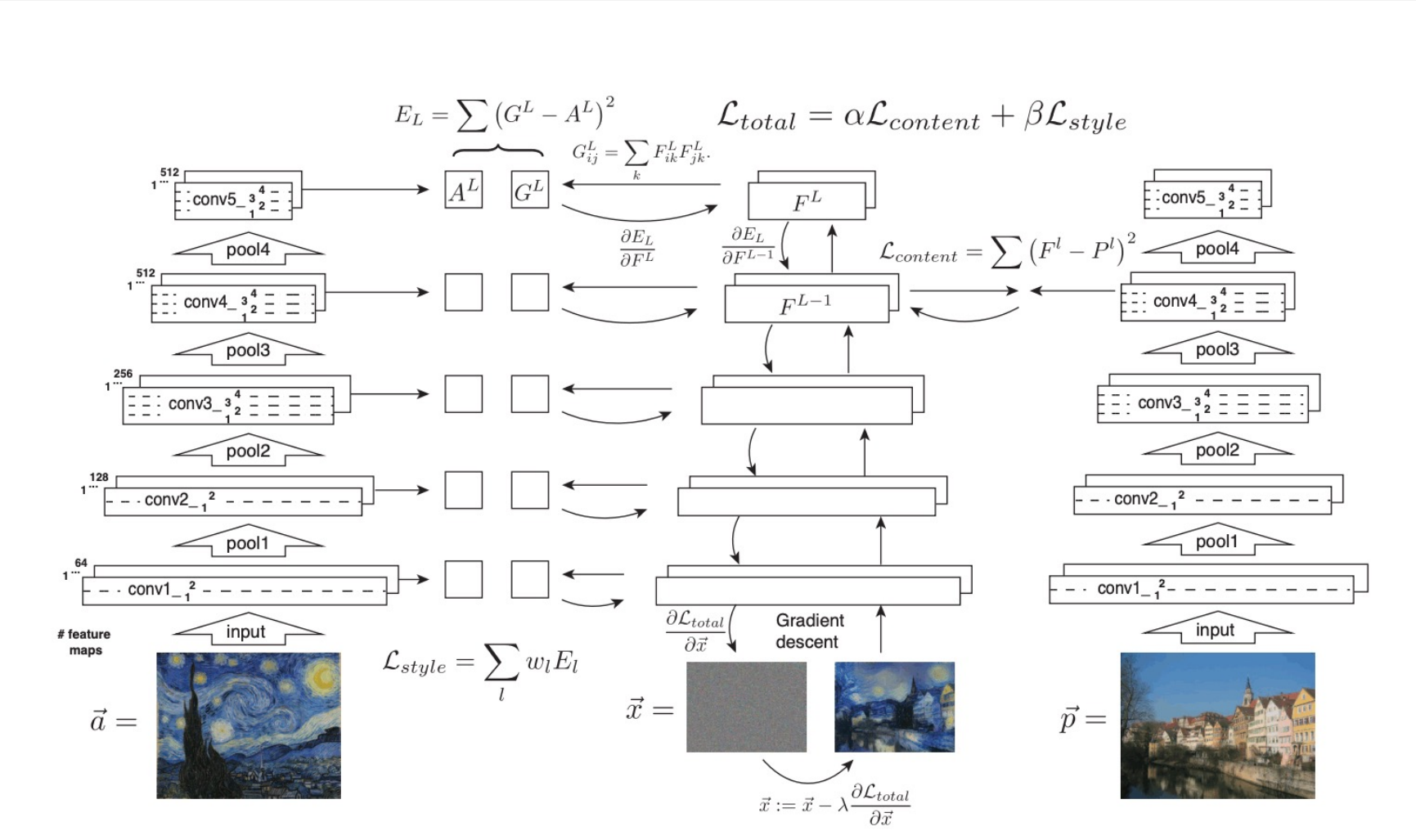


NEURAL STYLE TRANSFER

- ***Full loss***:
  - Compute the correlation between the different filter responses, where the expectation is taken over the spatial extent of the input image

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha\mathcal{L}(\vec{p}, \vec{x}) + \beta\mathcal{L}_{style}(\vec{a}, \vec{x})$$

NEURAL STYLE TRANSFER



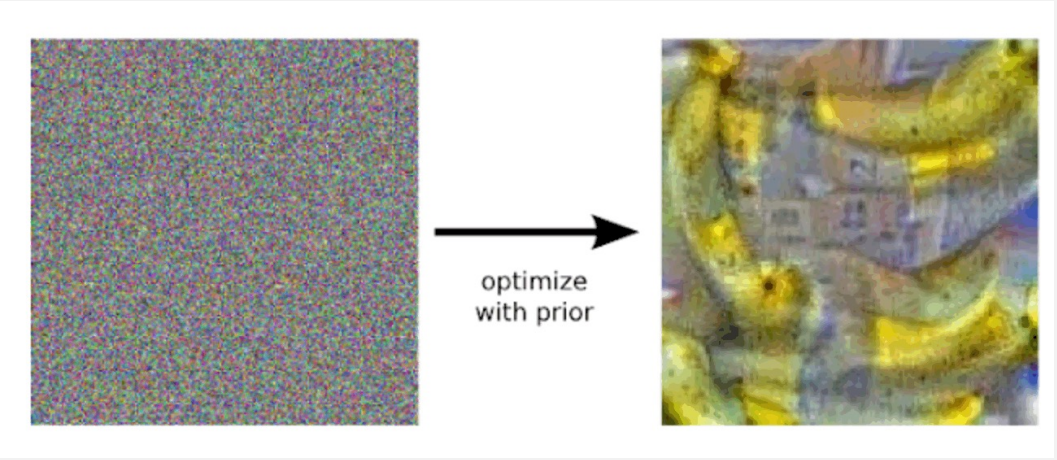
DEEPDREAM

- You will find the code in the file:

  "*DeepDream.ipynb*"

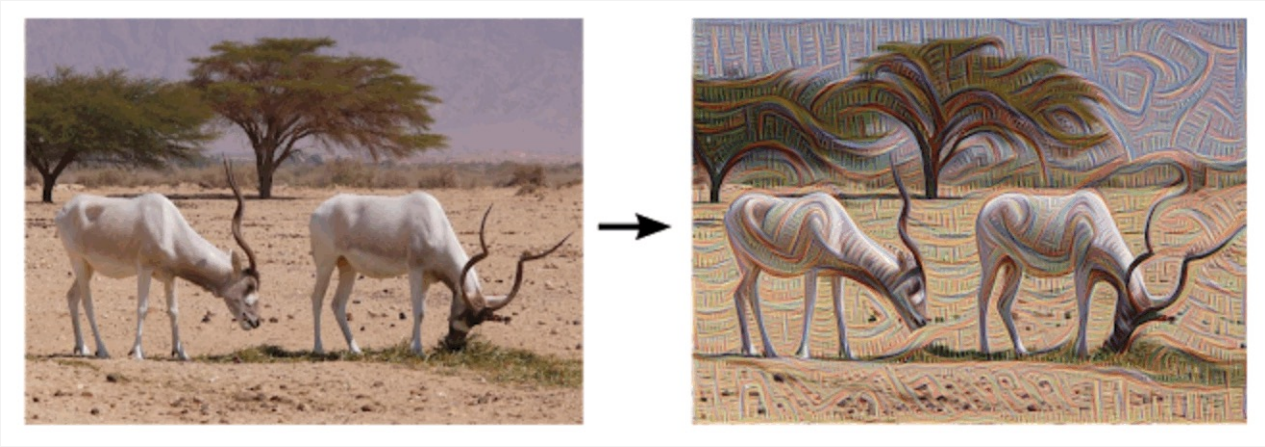
DEEPDREAM

- One of the challenges of neural networks is understanding what exactly goes on at each layer.
- In classification we know that after training, each layer progressively extracts higher and higher-level features of the image, until the final layer essentially makes a decision on what the image shows
- **Idea**:
  - turn the network upside down and ask it to enhance an input image in such a way as to elicit a particular interpretation
  - One way to visualize what goes on is to turn the network upside down and ask it to enhance an input image in such a way as to elicit a particular interpretation



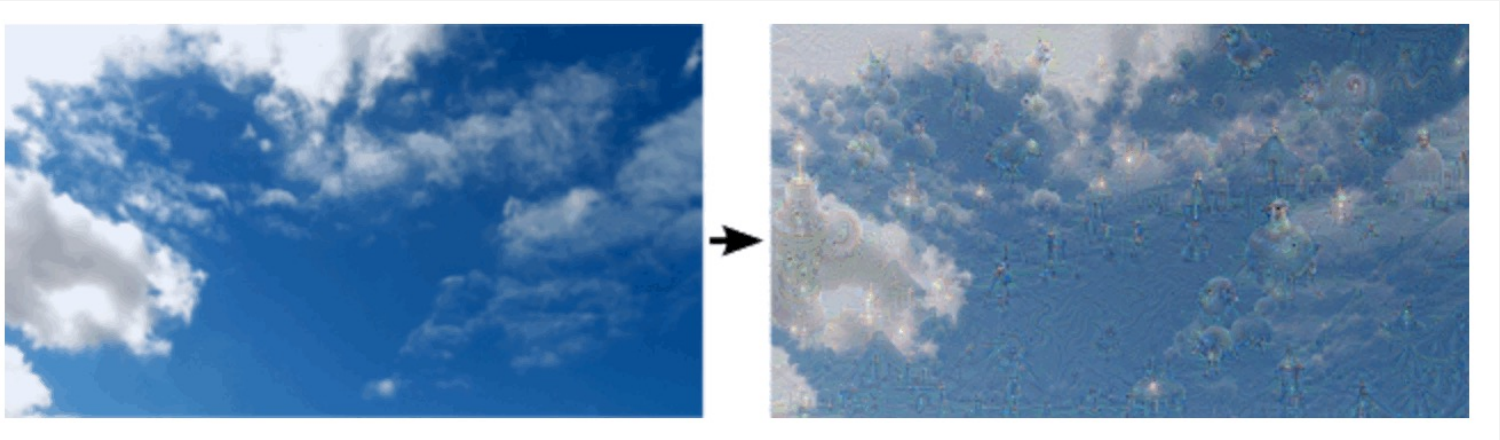
DEEPDREAM

- Instead of exactly prescribing which feature we want the network to amplify, we can also let the network make that decision.
- We feed the network an arbitrary image or photo and let the network analyze the picture. We then **pick a layer and ask the network to enhance whatever it detected**.
- Each layer of the network deals with features at a different level of abstraction
  - **Lower layers** -> strokes or simple ornament-like patterns



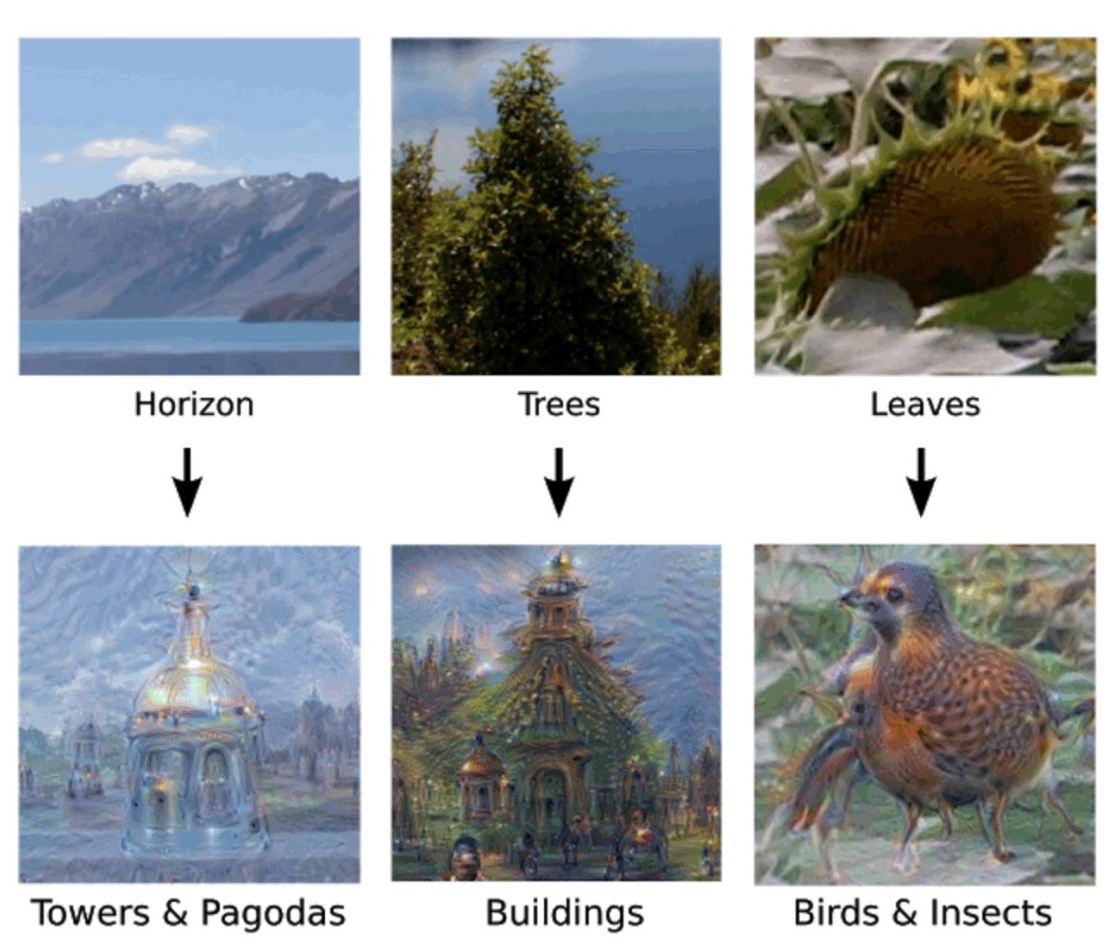
DEEPDREAM

- **Higher layers** -> identify more sophisticated features in images
  - complex features or even whole objects tend to emerge



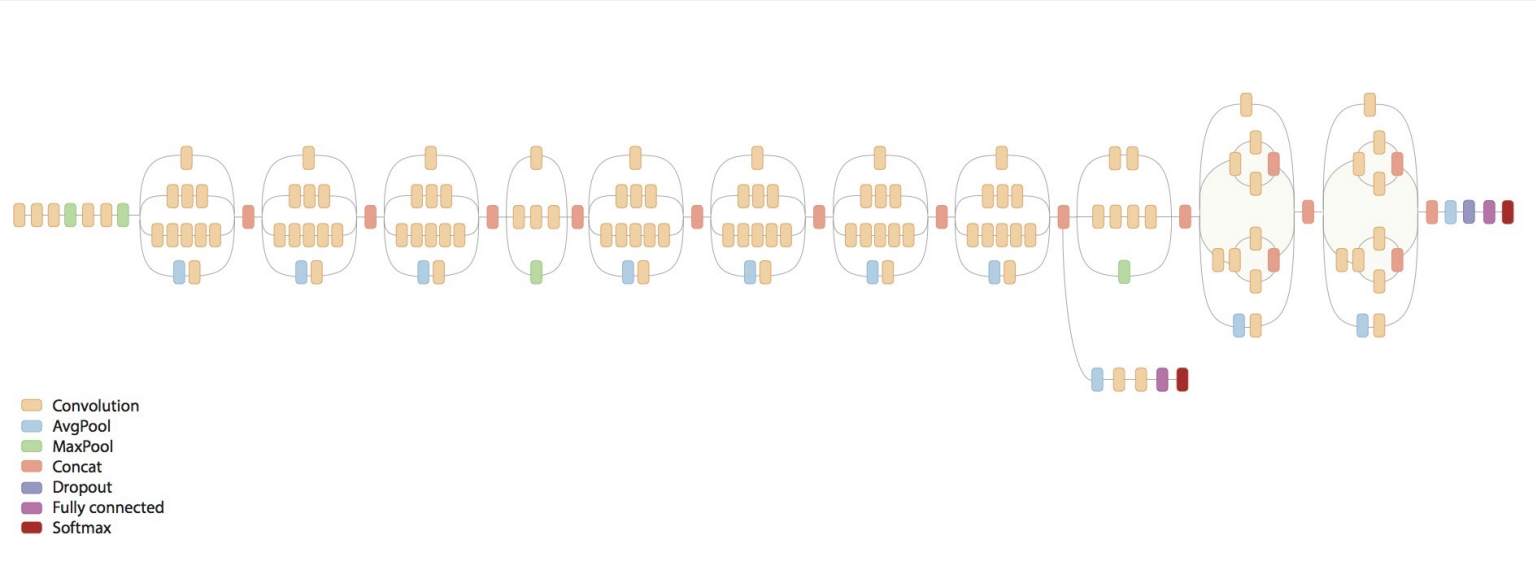
DEEPDREAM

- We can apply this technique to any kind of image. Results vary depending on the kind of image considered



DEEPDREAM

- To wrap up,
  - DeepDream Idea:
    - Feed a pre-trained classification network with images
    - Maximize the activations of chosen layers using **gradient ascent**
- the technique is also called **Inceptionism** referring to the pre-trained architecture used
  - Quite large architecture



DEEPDREAM

- If we repeat the algorithm endlessly on its own outputs and apply zooming, we get an endless stream of new impressions.

REFERENCES

- **Deep Learning**
  - http://www.deeplearningbook.org/
- **Style Transfer**
  - Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge, A neural algorithm of artistic style
  - Simonyan, Karen, and Andrew Zisserman, Very deep convolutional networks for large-scale image recognition.
  - Tensorflow tutorial
  - Fast style transfer on webcam, Gene Kogan
- **DeepDream**
  - Inceptionism: Going Deeper into Neural Networks
  - Szegedy, Christian, et al. Going deeper with convolutions
  - Tensorflow tutorial