Convolutional Neural Networks and Image Processing

Jiří Materna

Machine Learning College

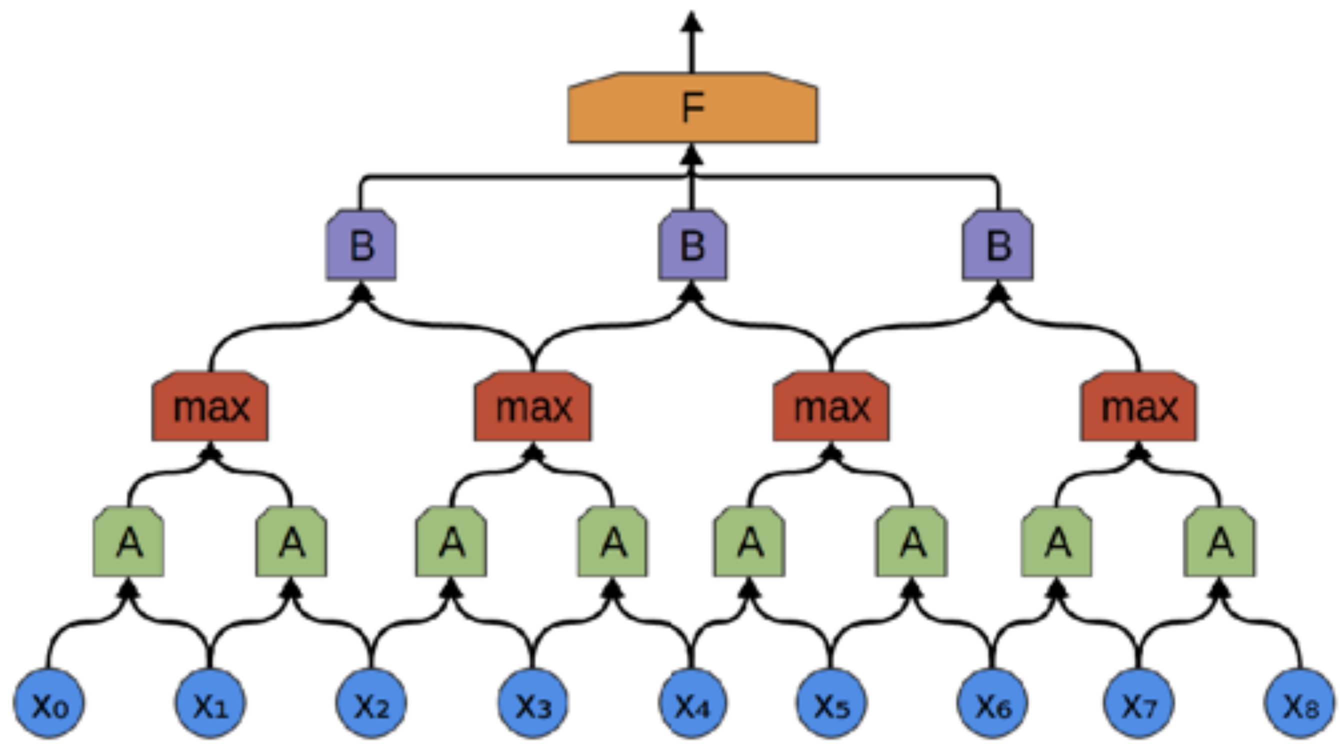
About me

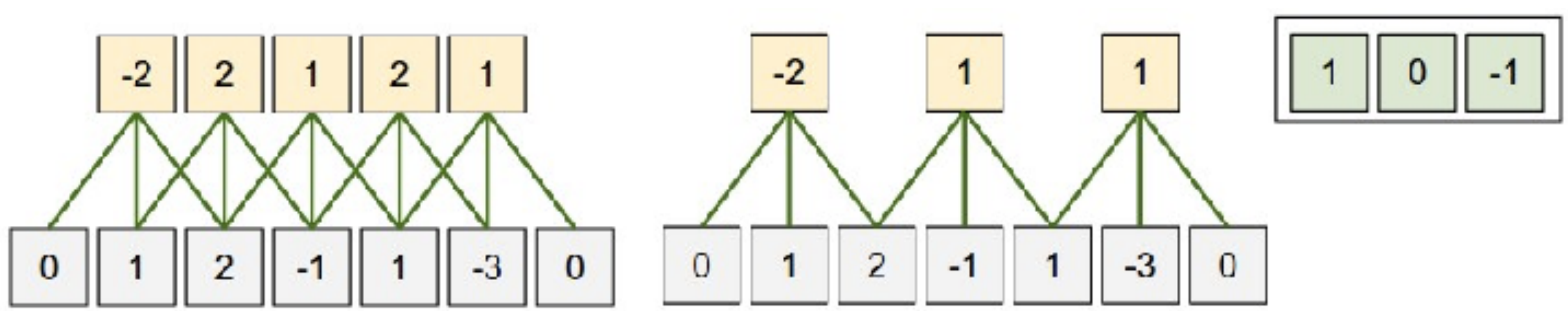
- Ph.D. in Natural Language Processing and Artificial Intelligence at Masaryk University
- 10 years at Seznam.cz (last 8 years as Head Of Research)
- Founder and lecturer at ML College
- Founder and co-organiser of ML Prague
- ML Freelance and consultant

Image processing

- Convolution
- VGG 16 and ResNet
- Transfer learning and fine-tuning
- Image classification
- Batch normalization and data augmentation
- U-net and Image segmentation
- GANs and superresolution
- Neural network explainability
- Adversarial patch

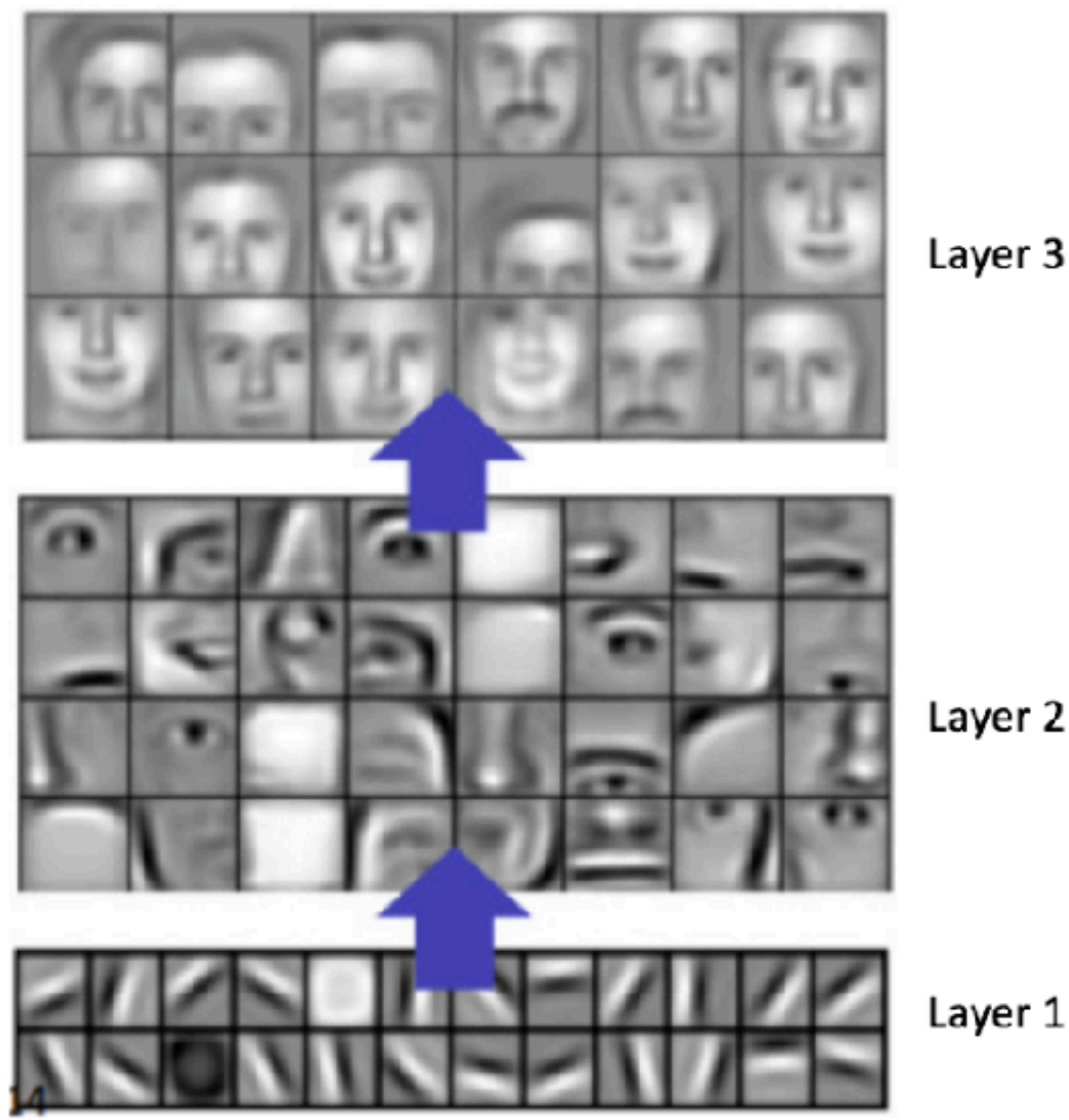
Convolution





Source: https://www.tensorflow.org

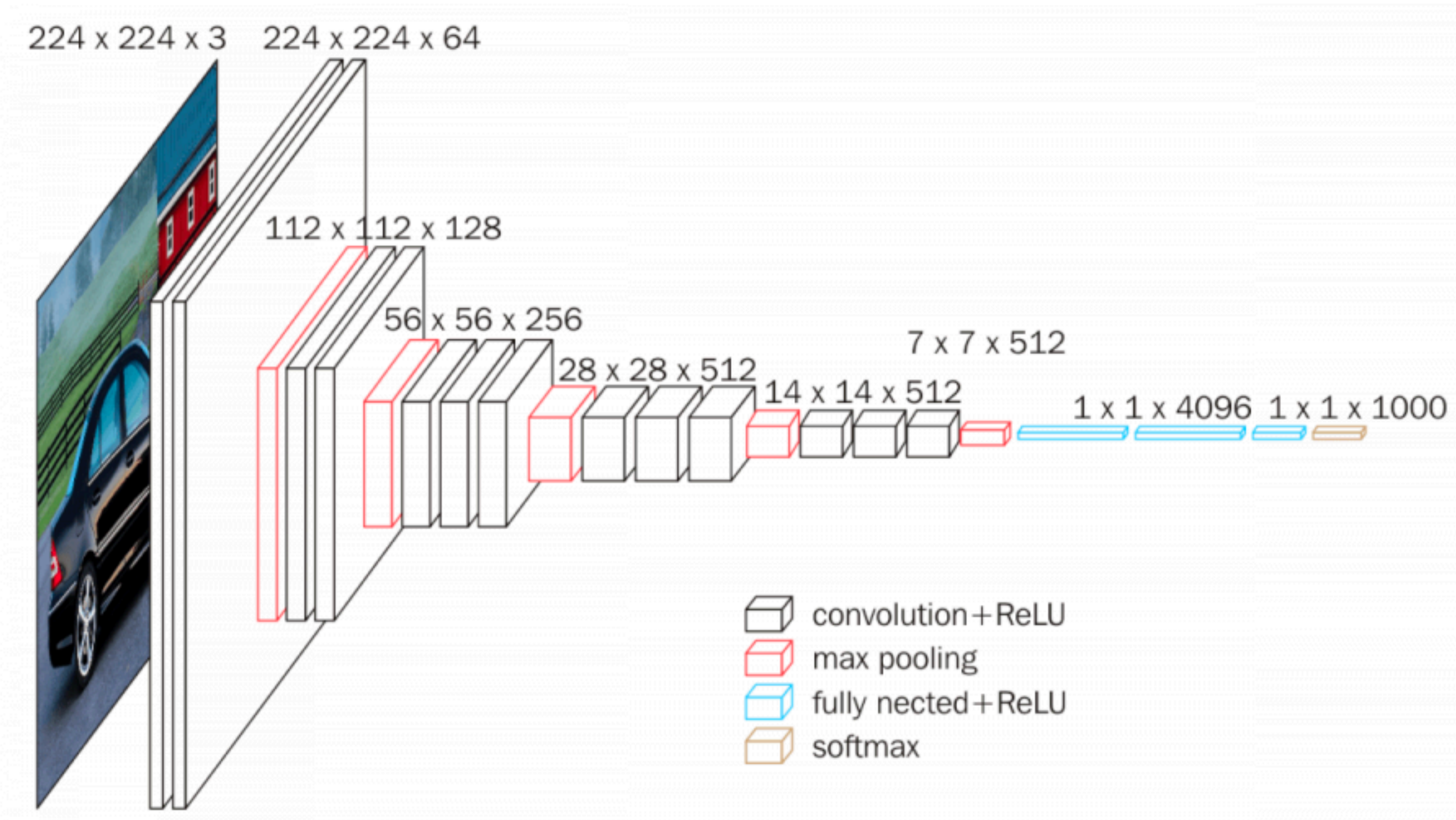
Weights visualization



MNIST Classification

<u>**01-MNIST-classification.ipynb**</u>

VGG 16

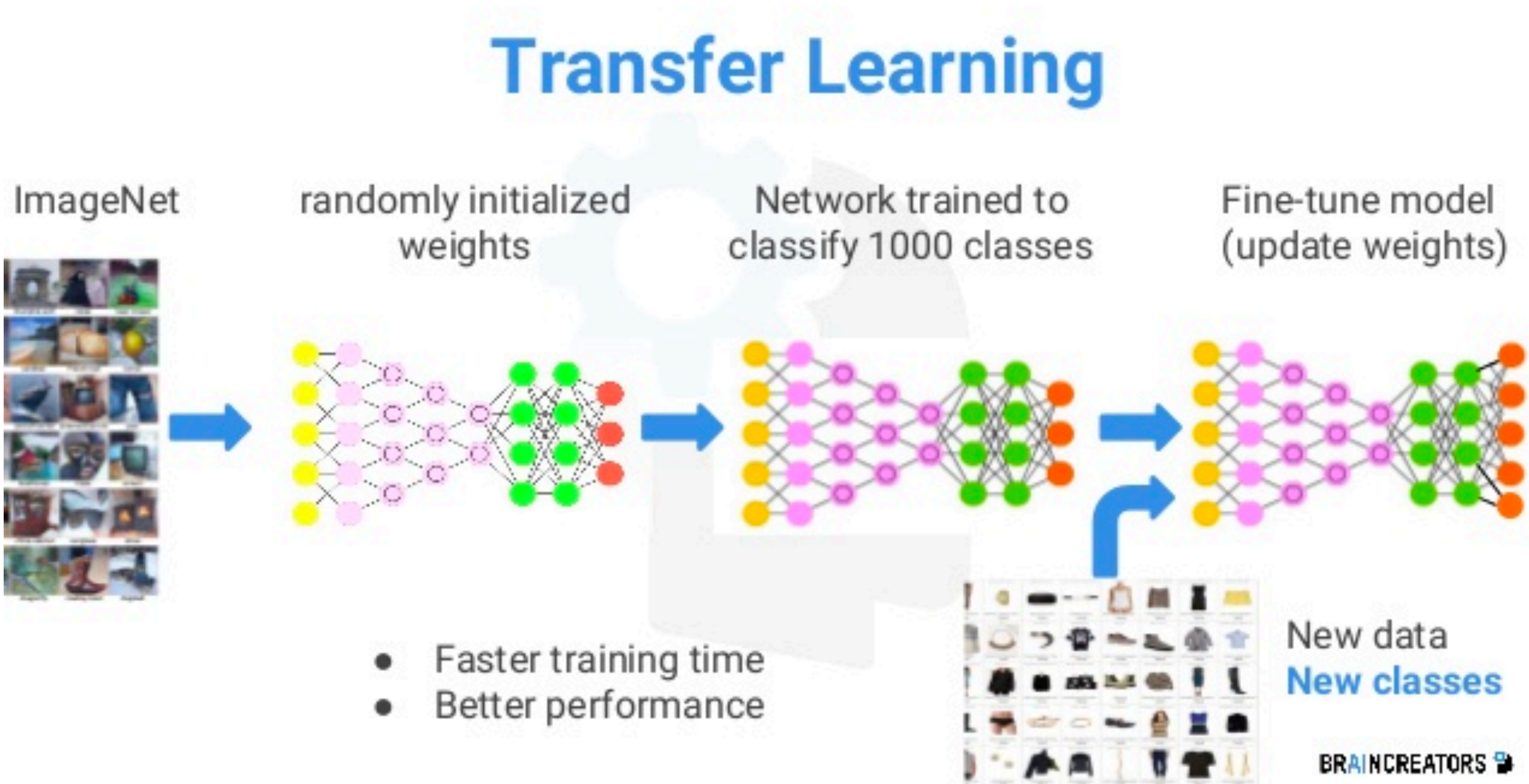


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution+ReLU
max pooling
fully nected+ReLU
softmax

ResNet

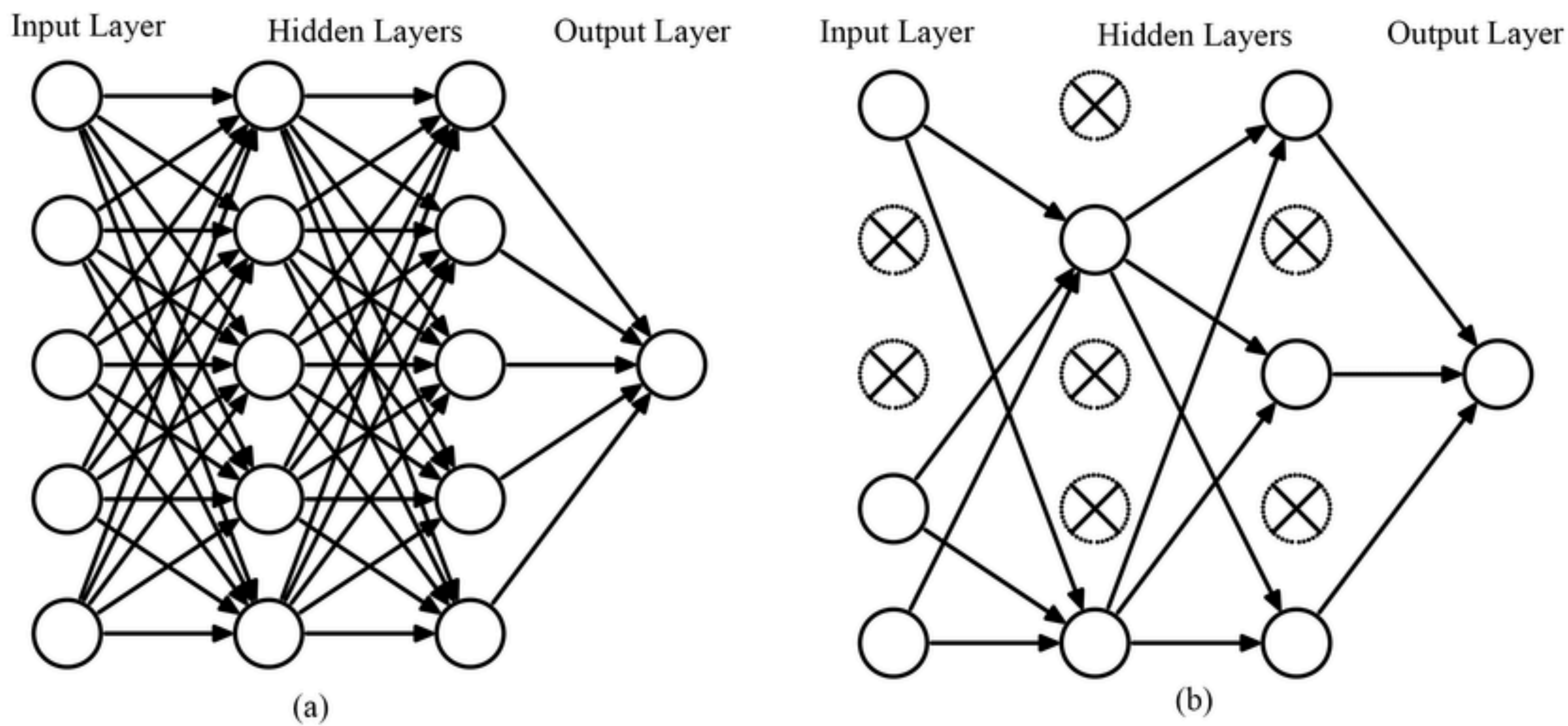

residual connection
x
weight layer
relu
$\mathcal{F}(\mathbf{x})$
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
Residual Network
Plain Network
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

Finetuning



Transfer Learning
ImageNet
randomly initialized weights
Network trained to classify 1000 classes
Fine-tune model (update weights)
Faster training time
Better performance
New data
New classes
BRAINCREATORS

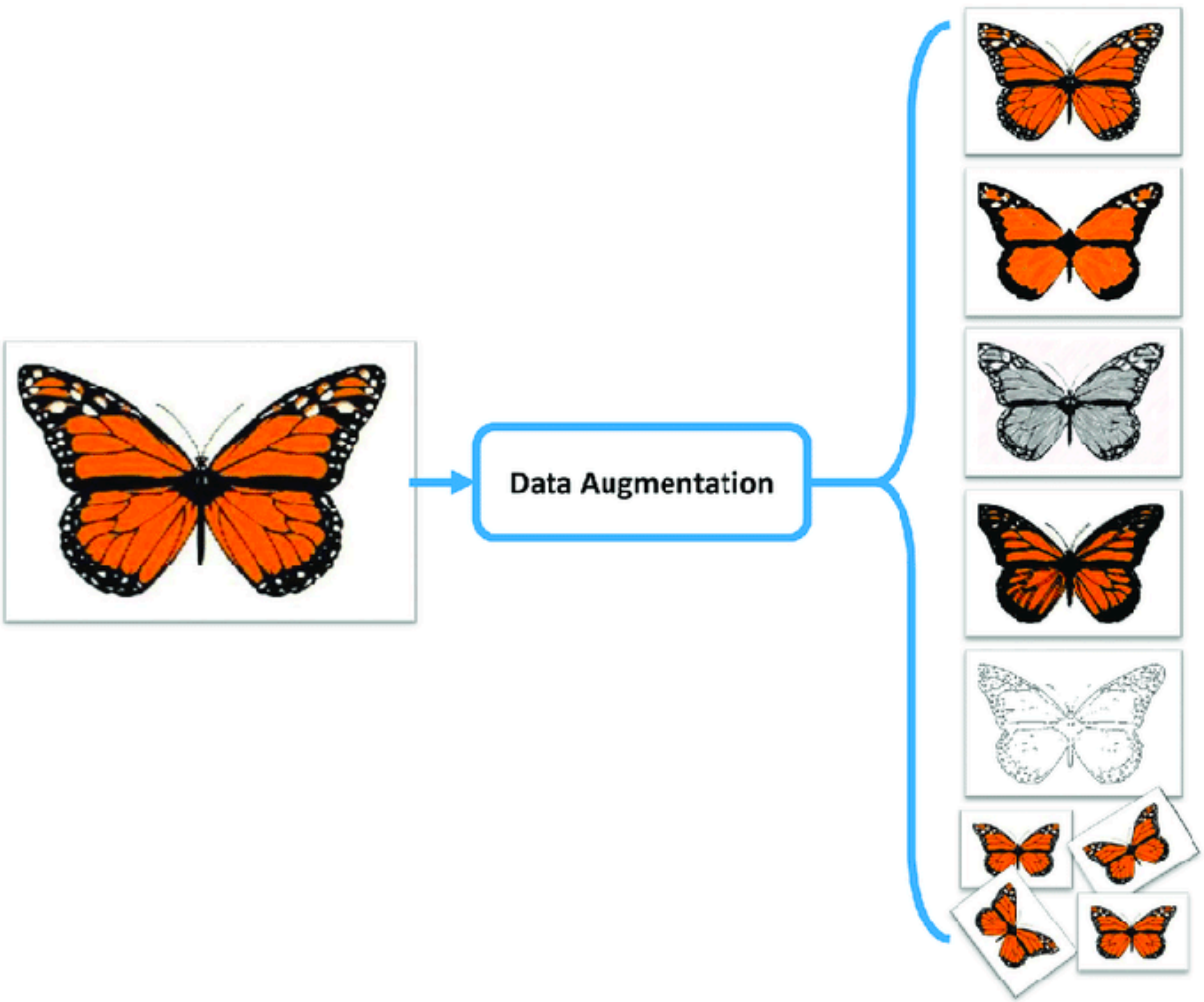
Dropout


Input Layer
Hidden Layers
Output Layer
(a)
Input Layer
Hidden Layers
Output Layer
(b)

Transfer learning example

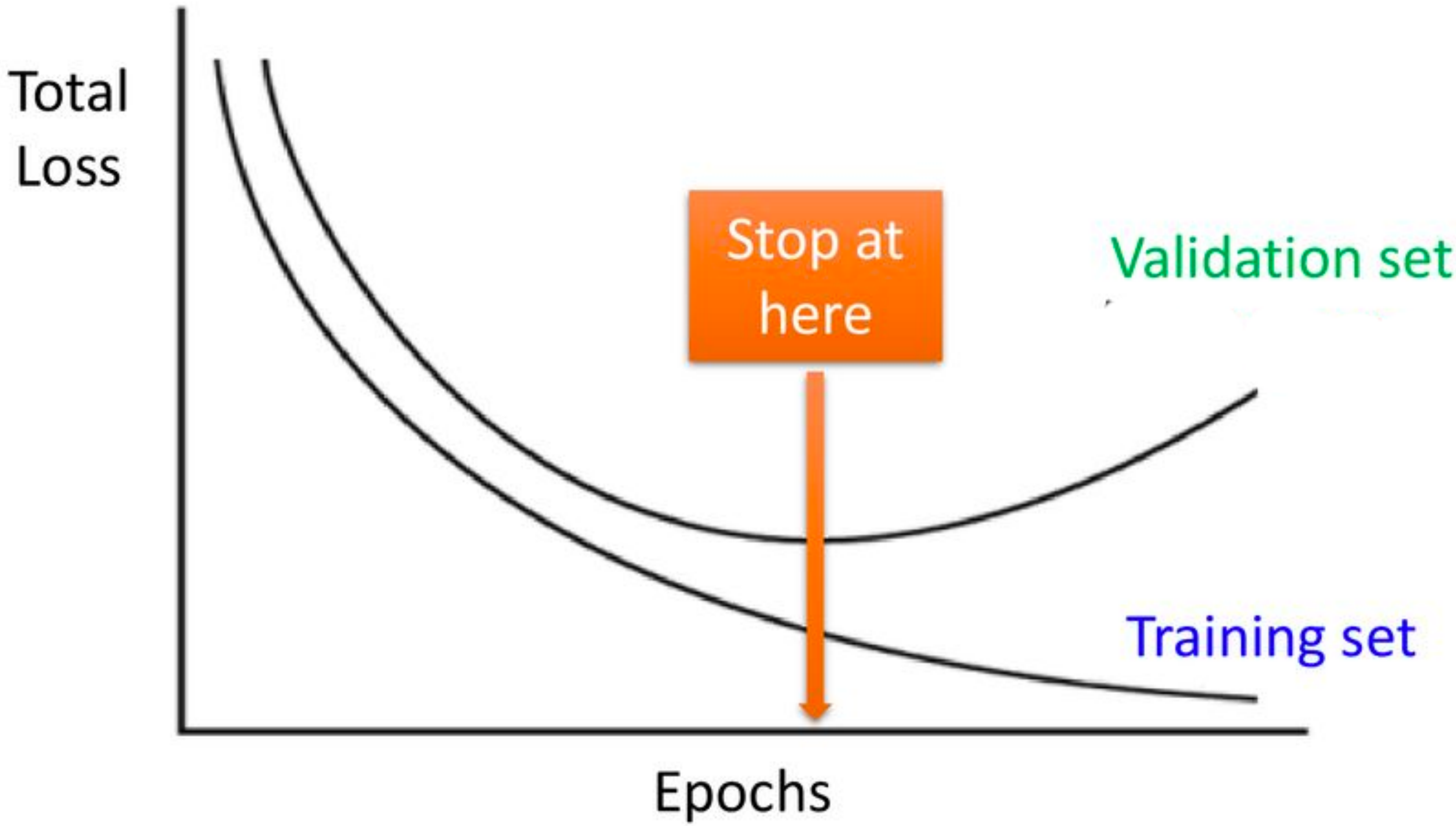
02-Transfer_learning.ipynb

Data augmentation



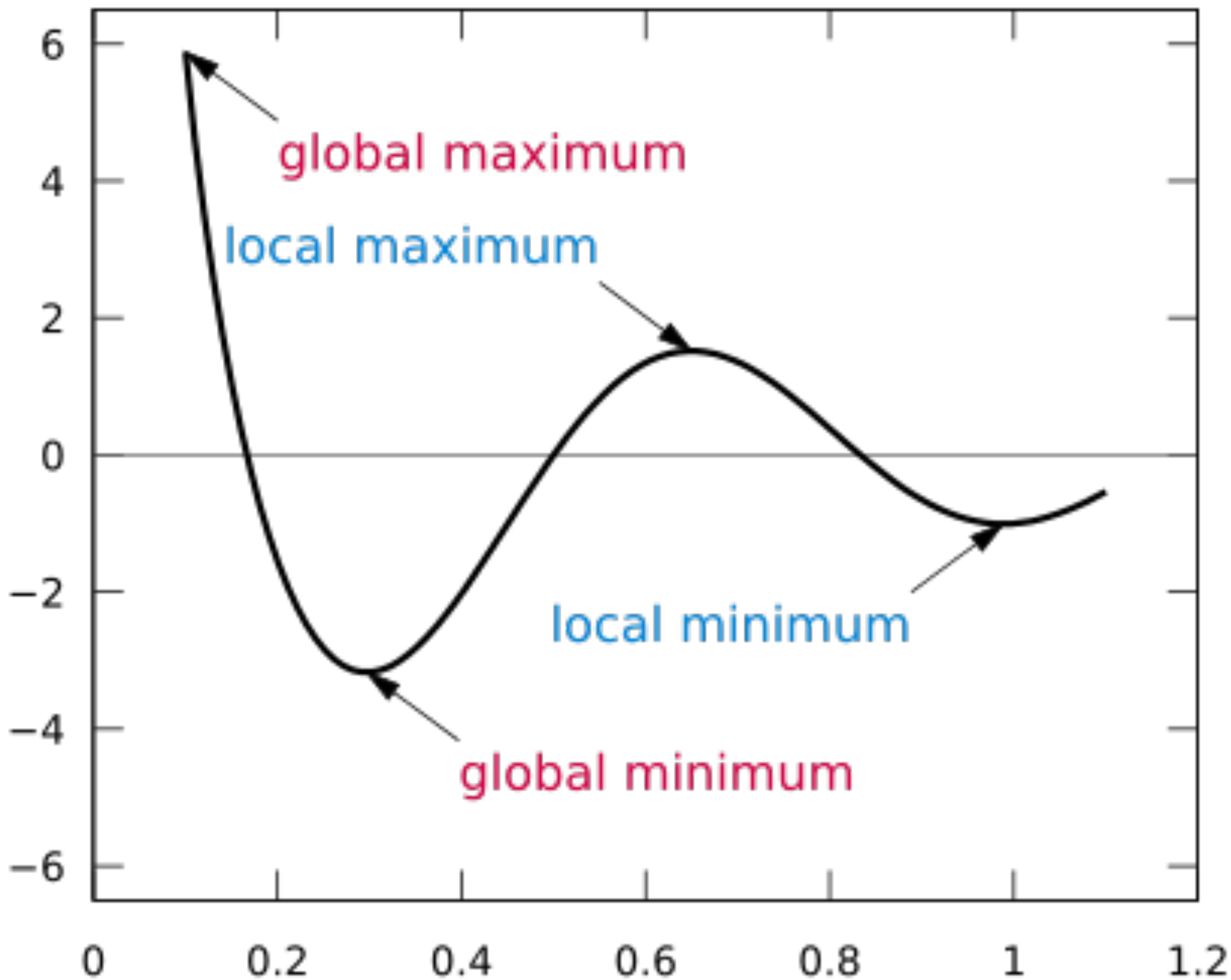
https://medium.com/secure-and-private-ai-writing-challenge/data-augmentation-increases-accuracy-of-your-model-but-how-aa1913468722

Early stopping



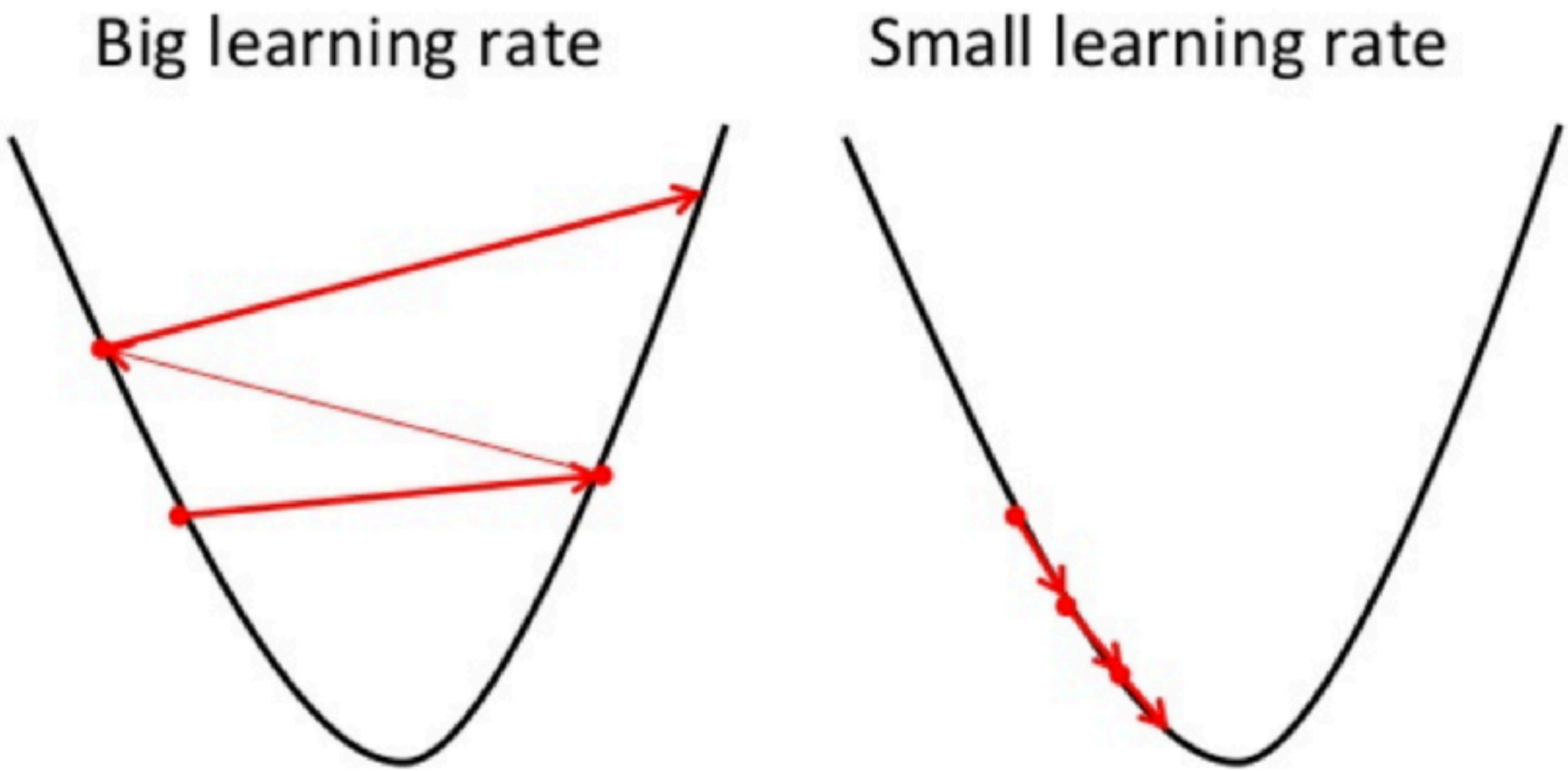
https://keras.io/getting_started/intro_to_keras_for_engineers/

Parameter optimization strategies

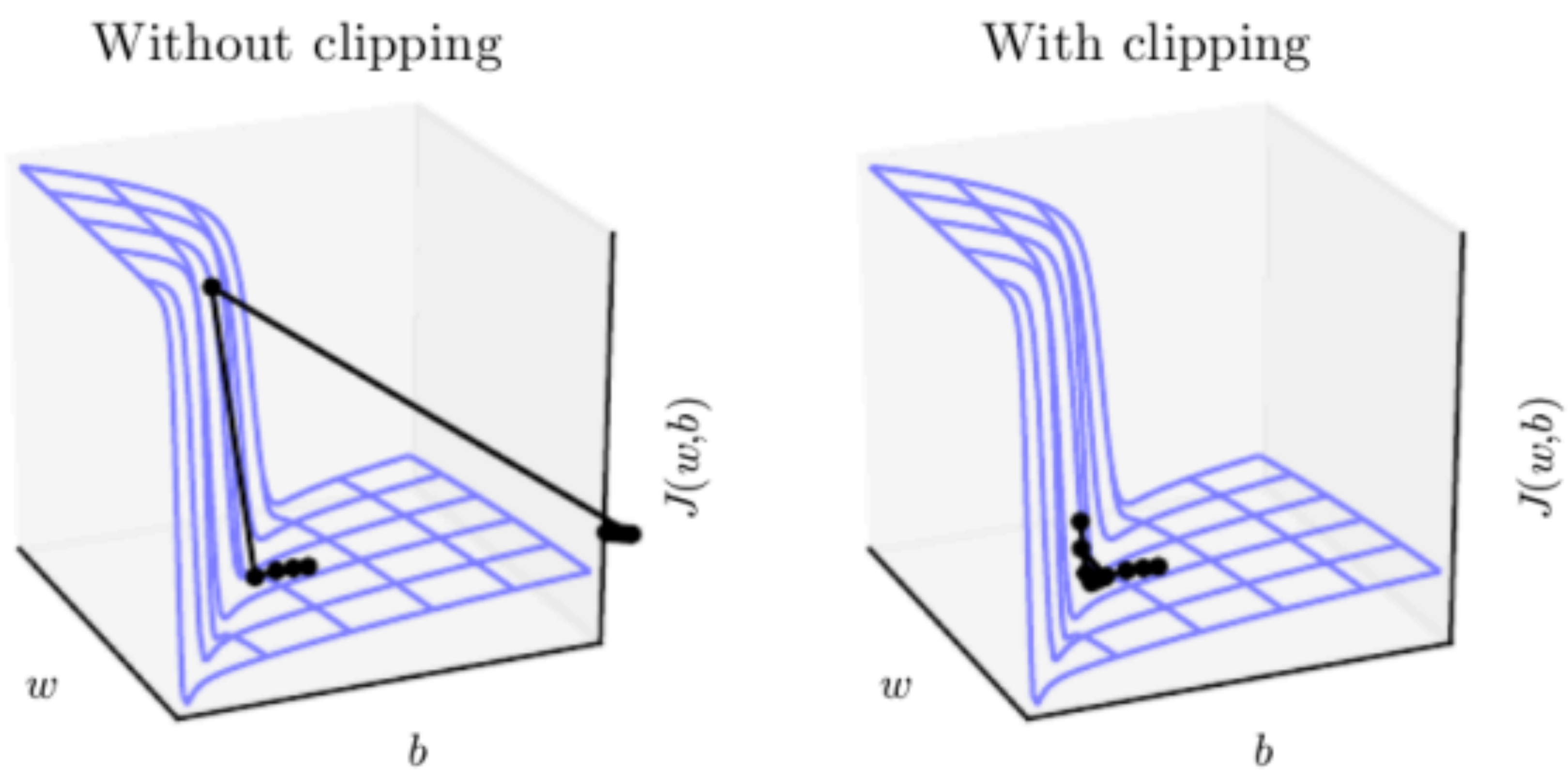


https://heartbeat.fritz.ai/an-empirical-comparison-of-optimizers-for-machine-learning-models-b86f29957050

Learning rate tuning



Gradient clipping



https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradients-problem

Batch normalization

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

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

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

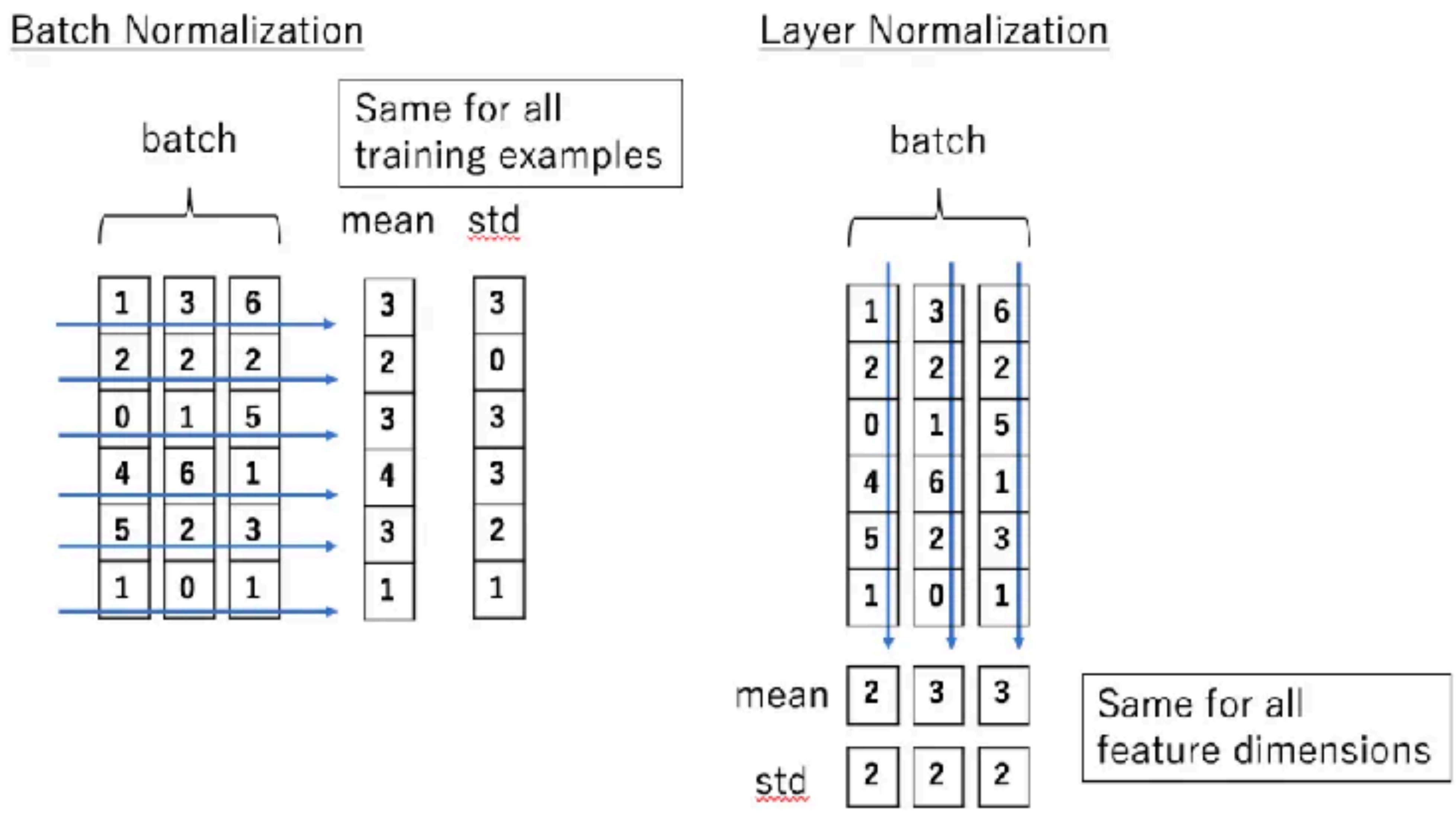
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

https://arxiv.org/abs/1502.03167

Layer normalization



https://mlexplained.com/2018/01/13/weight-normalization-and-layer-normalization-explained-normalization-in-deep-learning-part-2/

Functional API in Keras

```
# Sequential model
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(10, input_shape=(10,),activation='relu')
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu')
model.add(Dense(1, activation='sigmoid')
```

```
# Functional model
from keras.models import Model
from keras.layers import Input, Dense

visible = Input(shape=(10,))
hidden1 = Dense(10, activation='relu')(visible)
hidden2 = Dense(20, activation='relu')(hidden1)
hidden3 = Dense(10, activation='relu')(hidden2)
output = Dense(1, activation='sigmoid')(hidden3)
model = Model(inputs=visible, outputs=output)
```

Shared Input

```
# Shared Input Layer
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input, Dense, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
# input layer
visible = Input(shape=(64,64,1))
# first feature extractor
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
flat1 = Flatten()(pool1)
# second feature extractor
conv2 = Conv2D(16, kernel_size=8, activation='relu')(visible)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat2 = Flatten()(pool2)
# merge feature extractors
merge = concatenate([flat1, flat2])
# interpretation layer
hidden1 = Dense(10, activation='relu')(merge)
# prediction output
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```

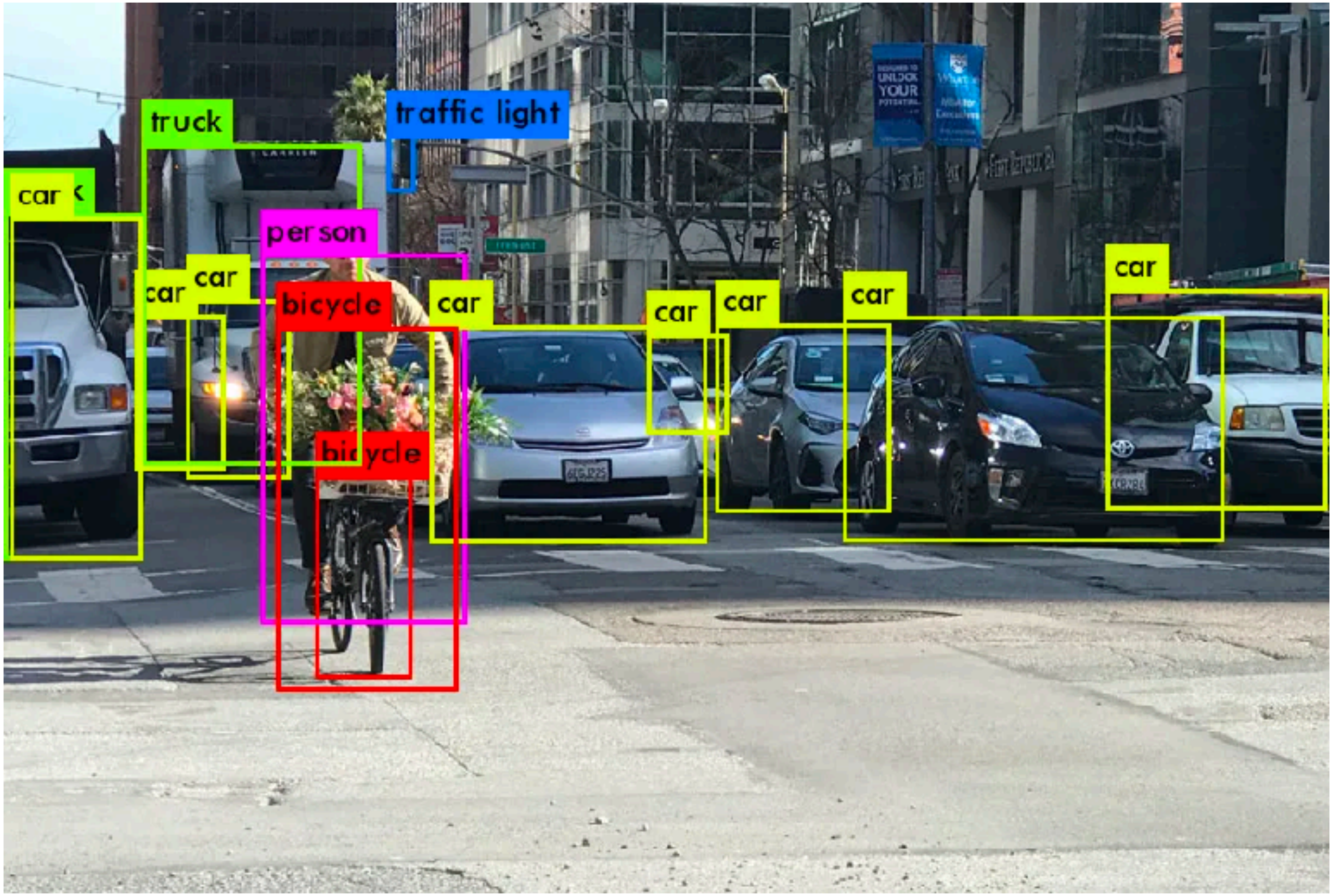
Multiple inputs (outputs)

```
# Multiple Inputs
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
# first input model
visible1 = Input(shape=(64,64,1))
conv11 = Conv2D(32, kernel_size=4, activation='relu')(visible1)
pool11 = MaxPooling2D(pool_size=(2, 2))(conv11)
conv12 = Conv2D(16, kernel_size=4, activation='relu')(pool11)
pool12 = MaxPooling2D(pool_size=(2, 2))(conv12)
flat1 = Flatten()(pool12)
# second input model
visible2 = Input(shape=(32,32,3))
conv21 = Conv2D(32, kernel_size=4, activation='relu')(visible2)
pool21 = MaxPooling2D(pool_size=(2, 2))(conv21)
conv22 = Conv2D(16, kernel_size=4, activation='relu')(pool21)
pool22 = MaxPooling2D(pool_size=(2, 2))(conv22)
flat2 = Flatten()(pool22)
# merge input models
merge = concatenate([flat1, flat2])
# interpretation model
hidden1 = Dense(10, activation='relu')(merge)
hidden2 = Dense(10, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
model = Model(inputs=[visible1, visible2], outputs=output)
```

Practical example on regularization and normalization

<u>03-Normalization-and-regularization-assignment.ipynb</u>

Object detection



YOLO v1 (You Only Look Once)

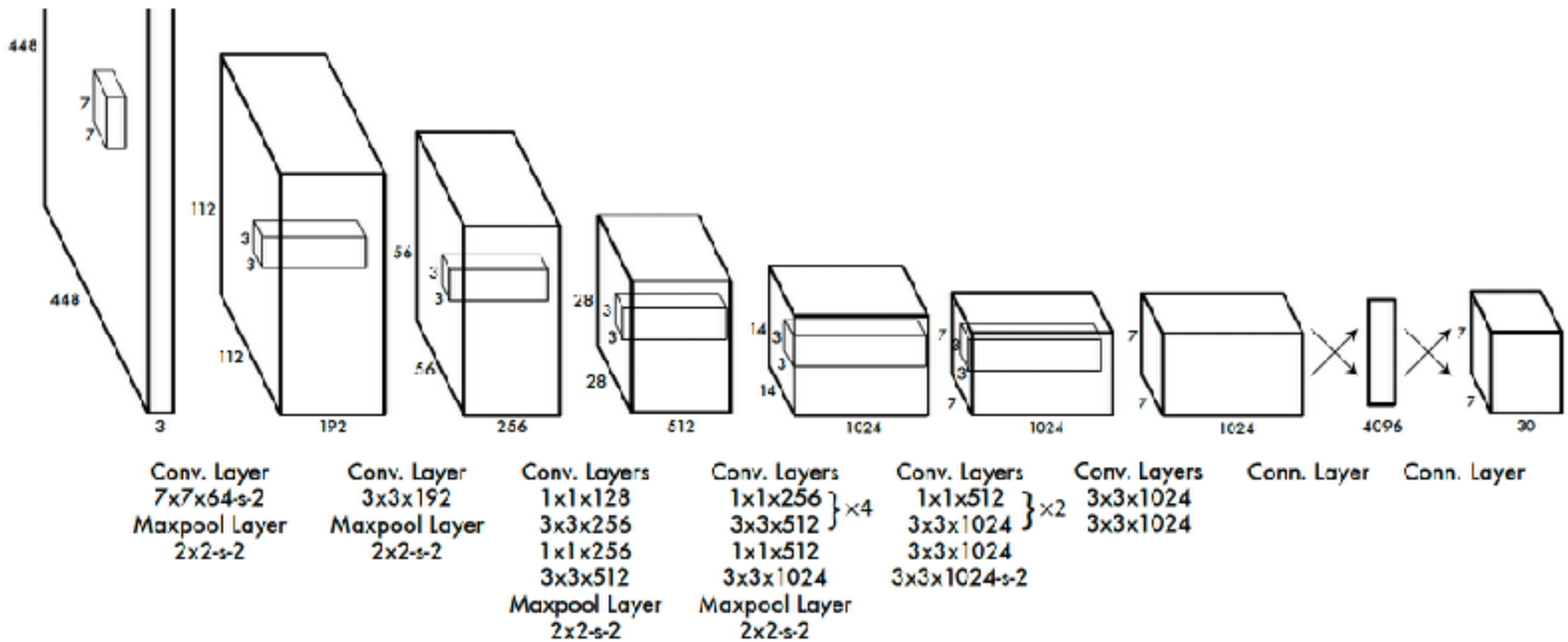


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Source: https://arxiv.org/abs/1506.02640

YOLO (You Only Look Once)

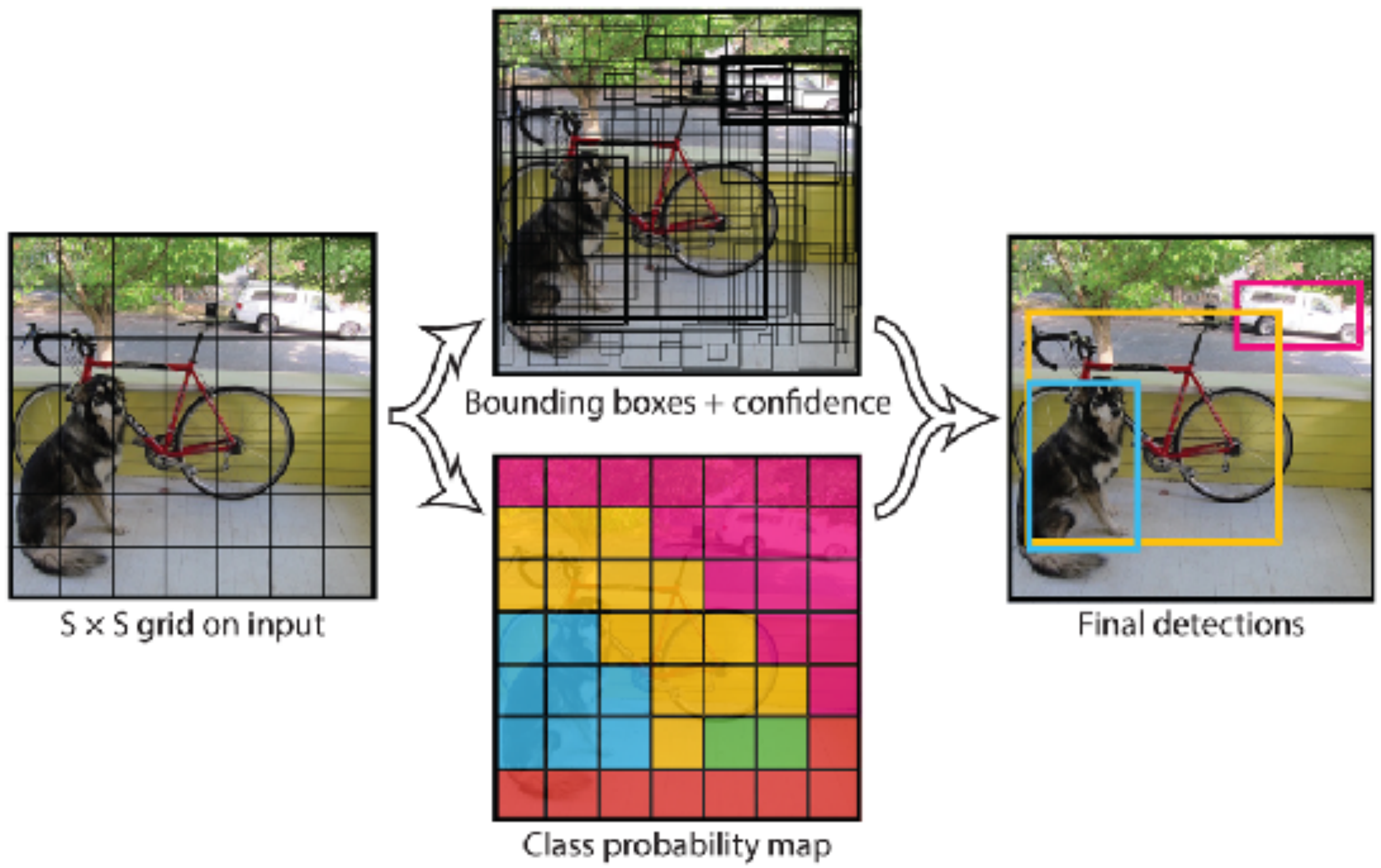


Figure 2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

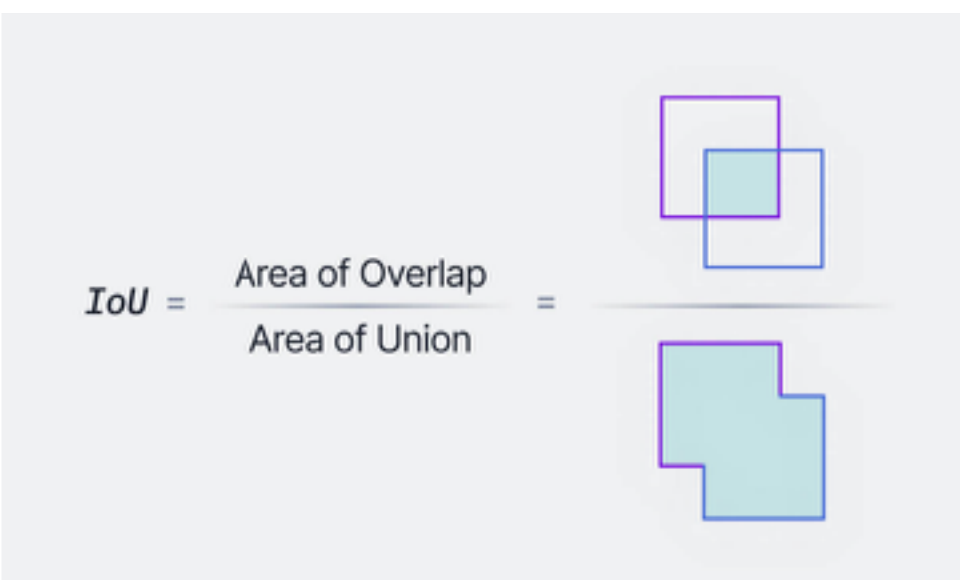
Source: https://arxiv.org/abs/1506.02640

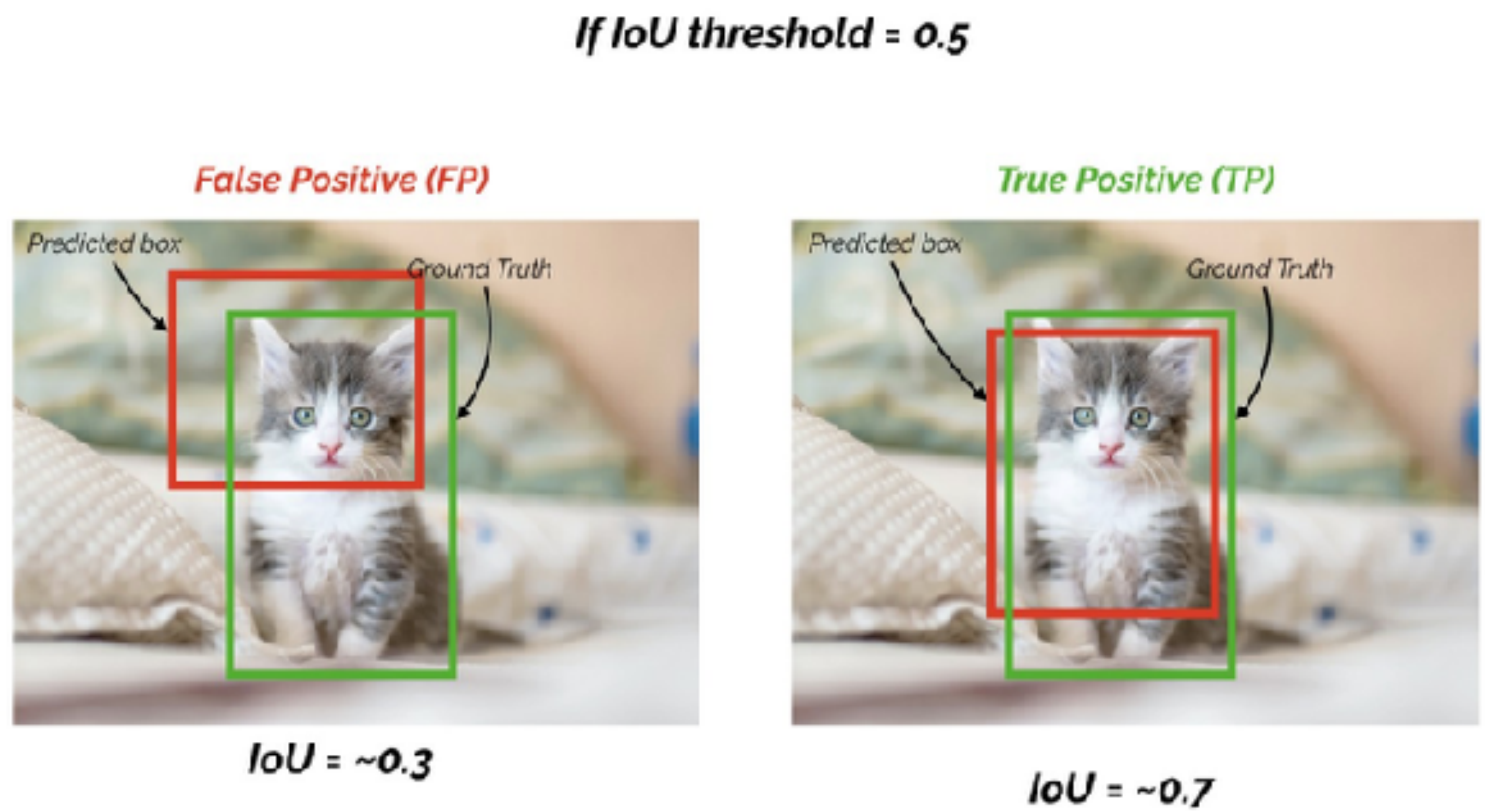
YOLO v8 usage

```python
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml")  # build a new model from scratch
model = YOLO("yolov8n.pt")  # load a pretrained model (recommended for training)

# Use the model
model.train(data="coco128.yaml", epochs=3)  # train the model
metrics = model.val()  # evaluate model performance on the validation set
results = model("https://ultralytics.com/images/bus.jpg")  # predict on an image
path = model.export(format="onnx")  # export the model to ONNX format
```

Source: https://github.com/ultralytics/ultralytics

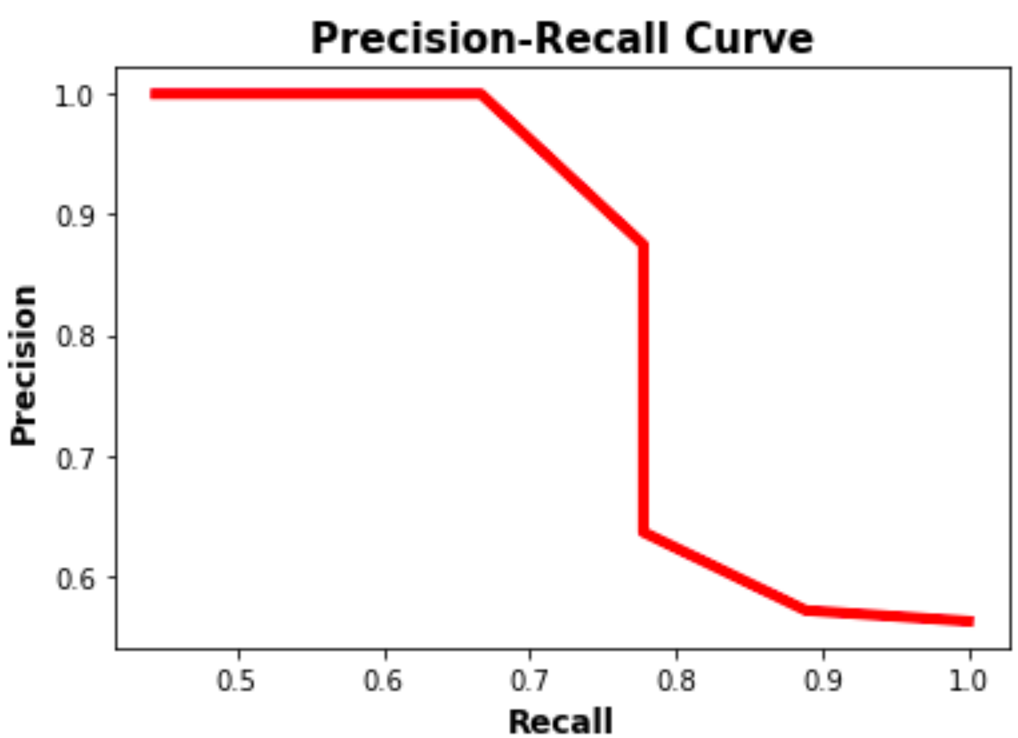
Object detection metrics
IoU





Source: https://www.v7labs.com/blog/yolo-object-detection

Object detection metrics
mAP



$$AP = \sum_{k=0}^{k=n-1} [Recalls(k) - Recalls(k+1)] * Precisions(k)$$

$$Recalls(n) = 0, Precisions(n) = 1$$

$$n = Number\ of\ thresholds.$$

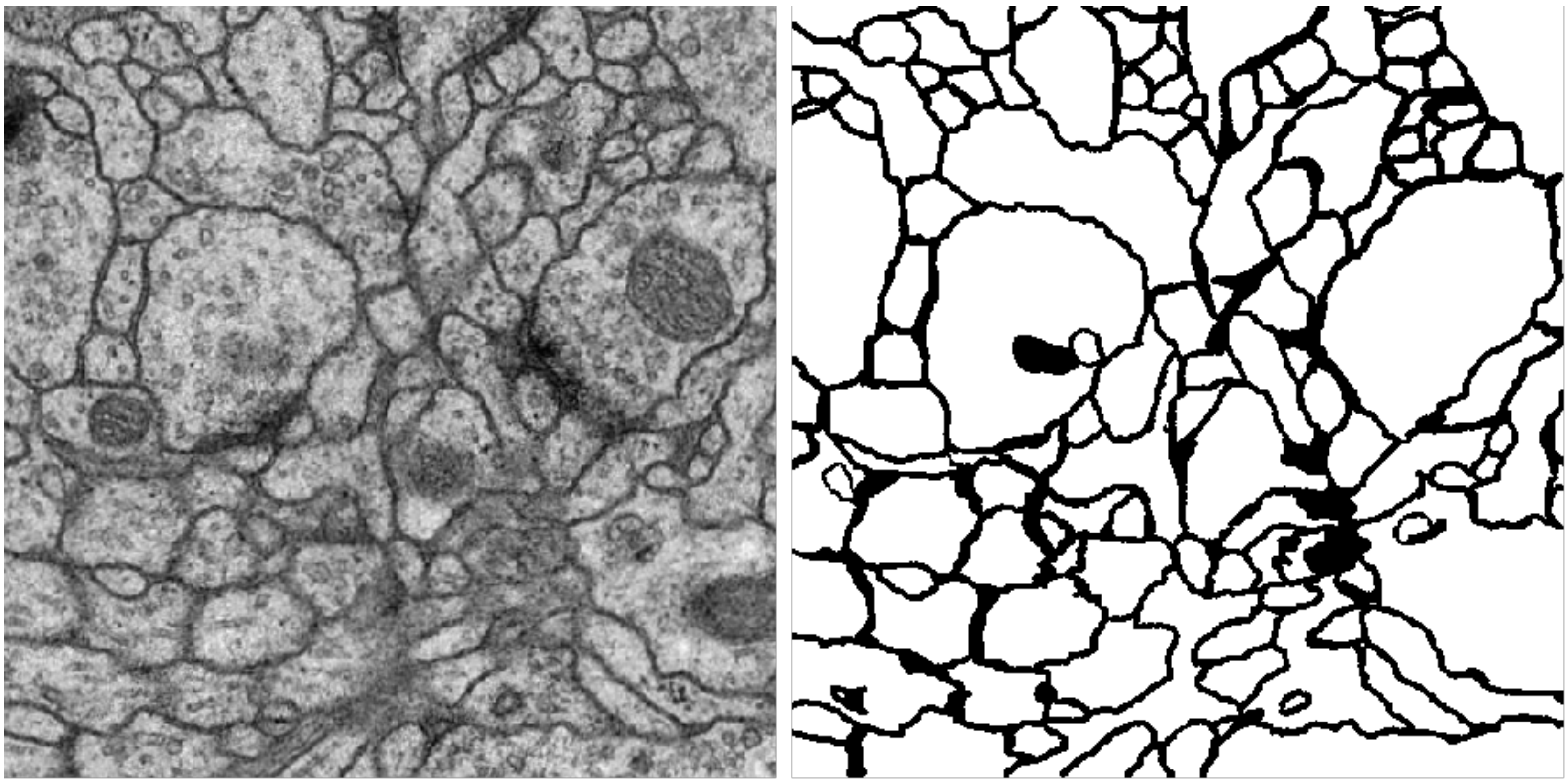
$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

$$AP_k = \text{ the AP of class } k$$

$$n = \text{ the number of classes}$$

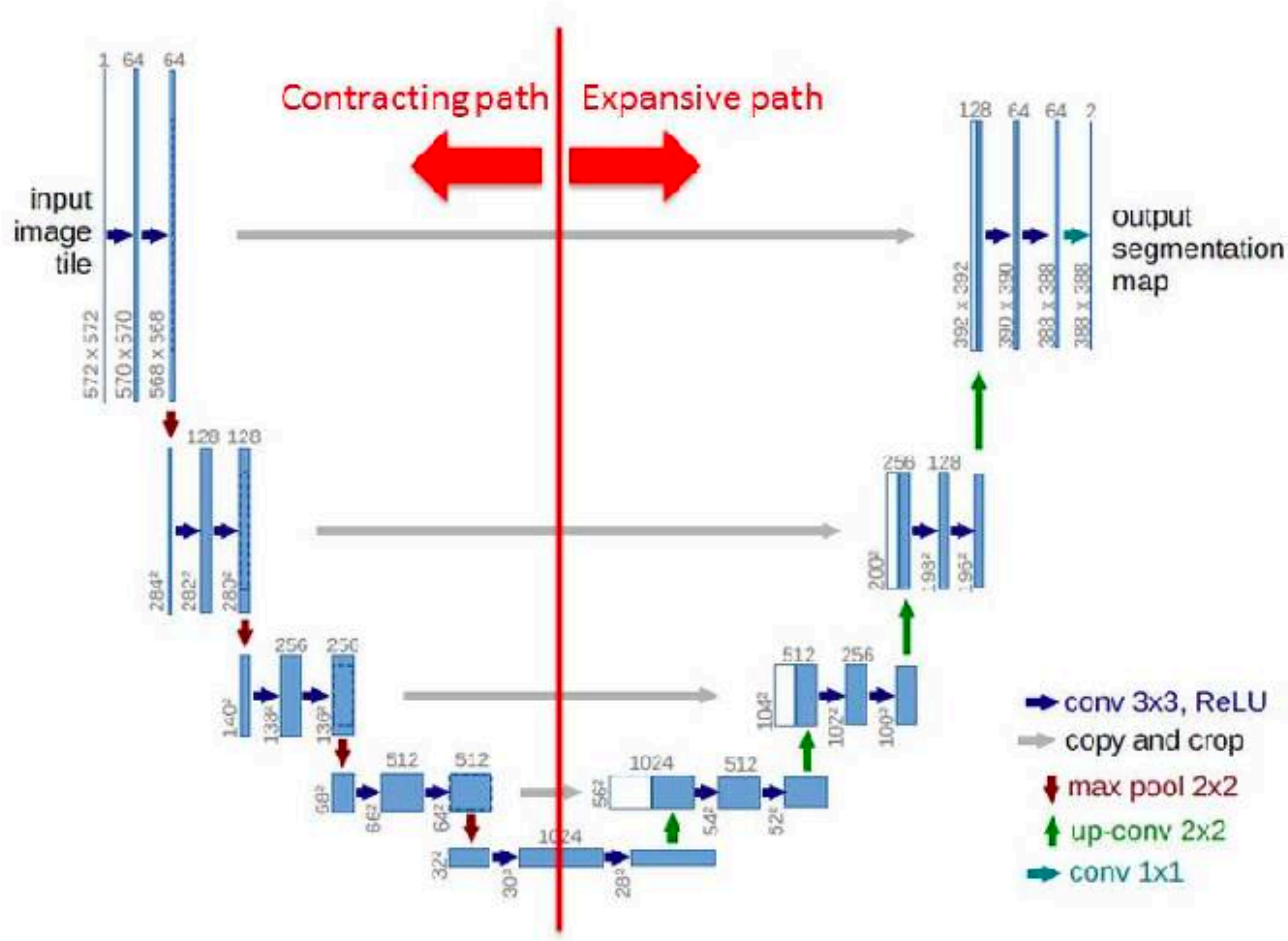
Source: https://blog.paperspace.com/mean-average-precision/amp/

Image segmentation



U-Net

Network Architecture

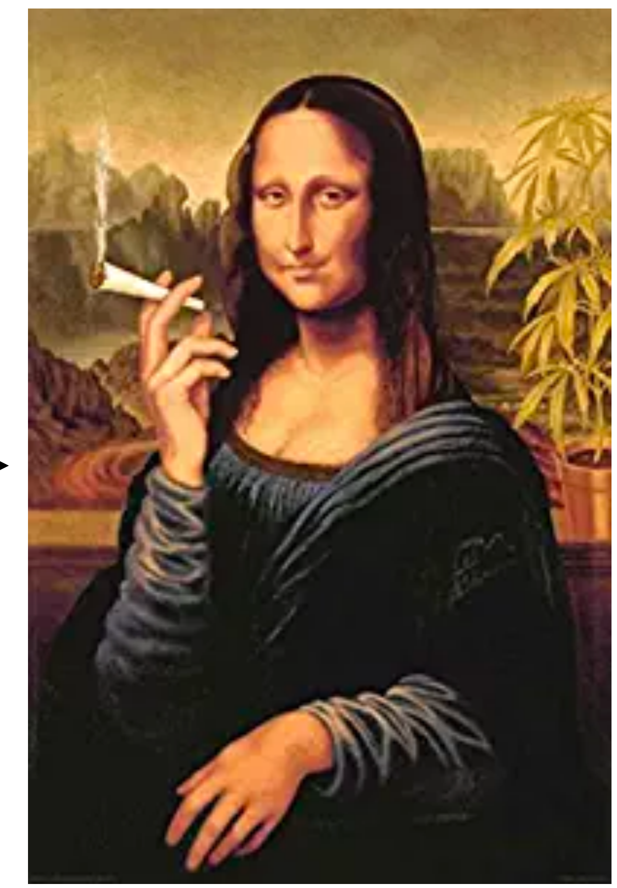


U-Net segmentation example

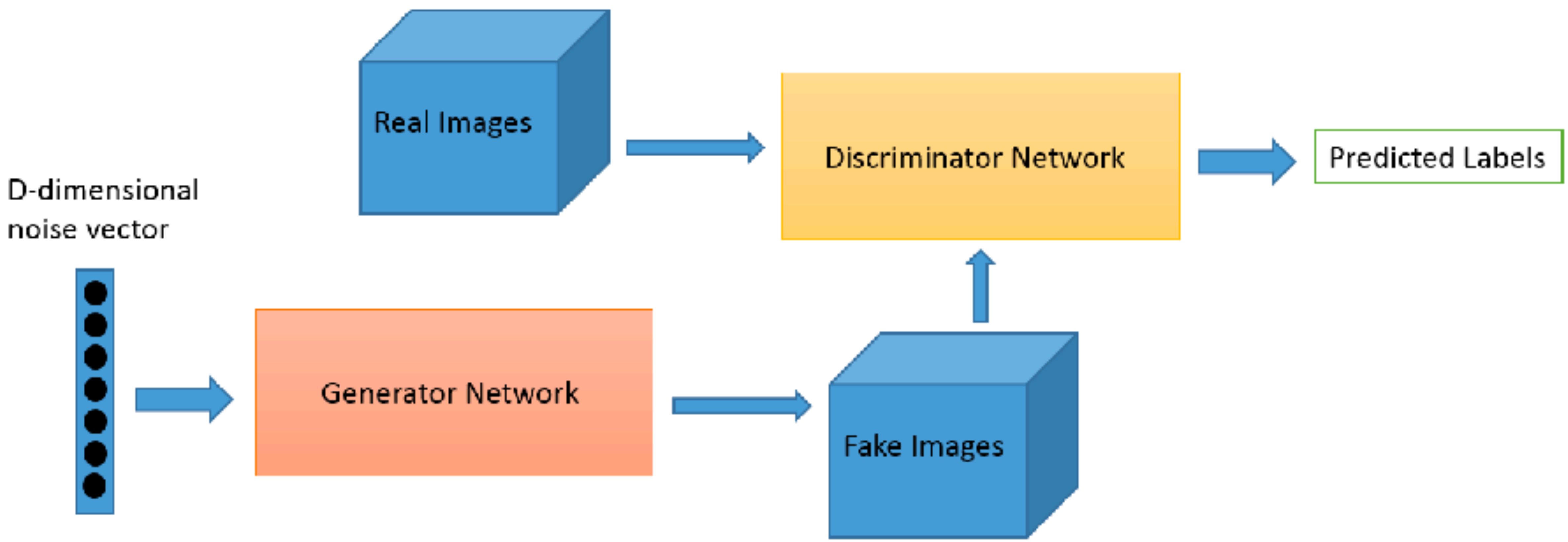
04-Segmentation.ipynb

Generative models with neural networks

Random noice → **Artificial Neural network** →



Generative Adversarial Networks



Superresolution

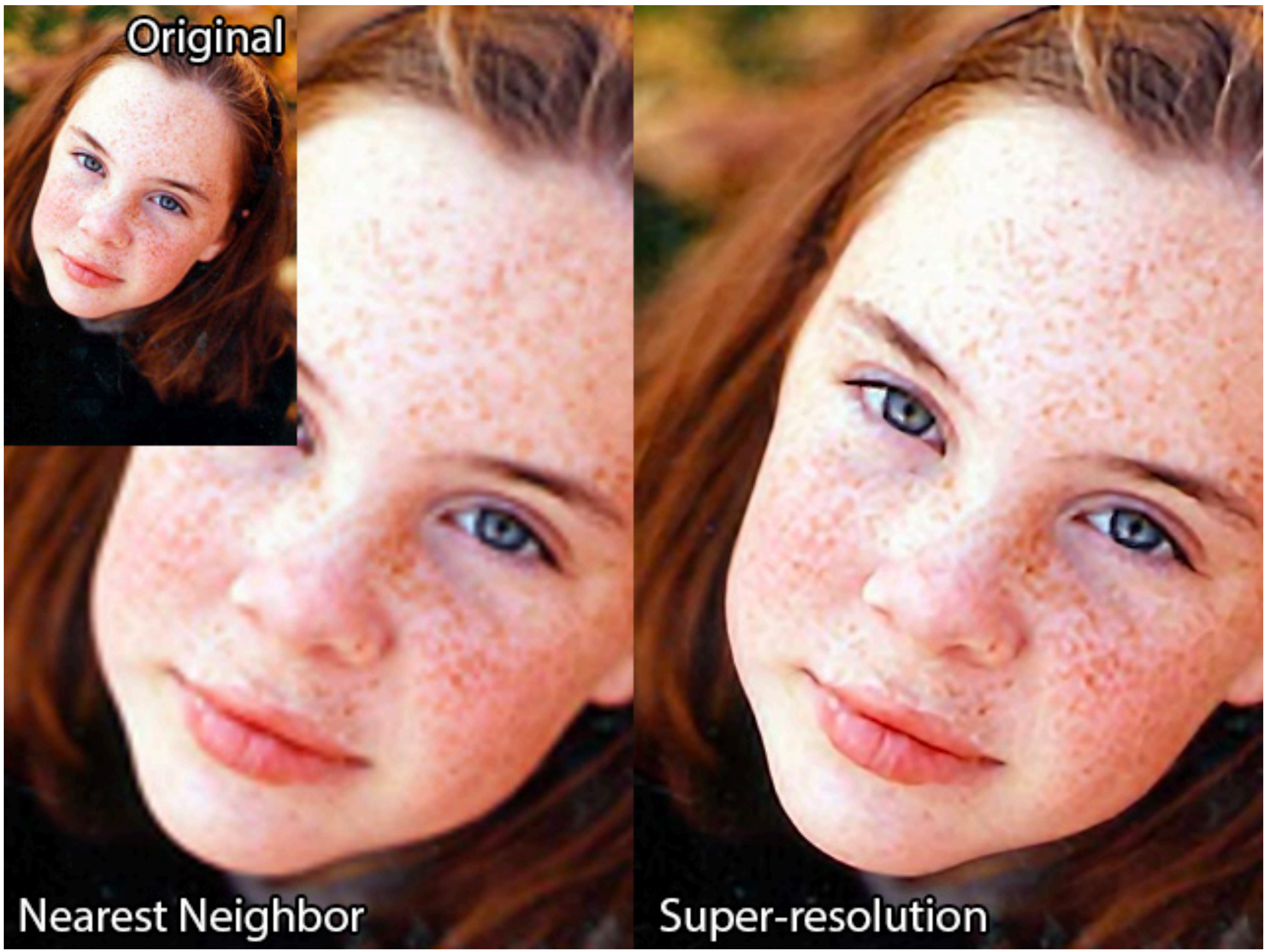
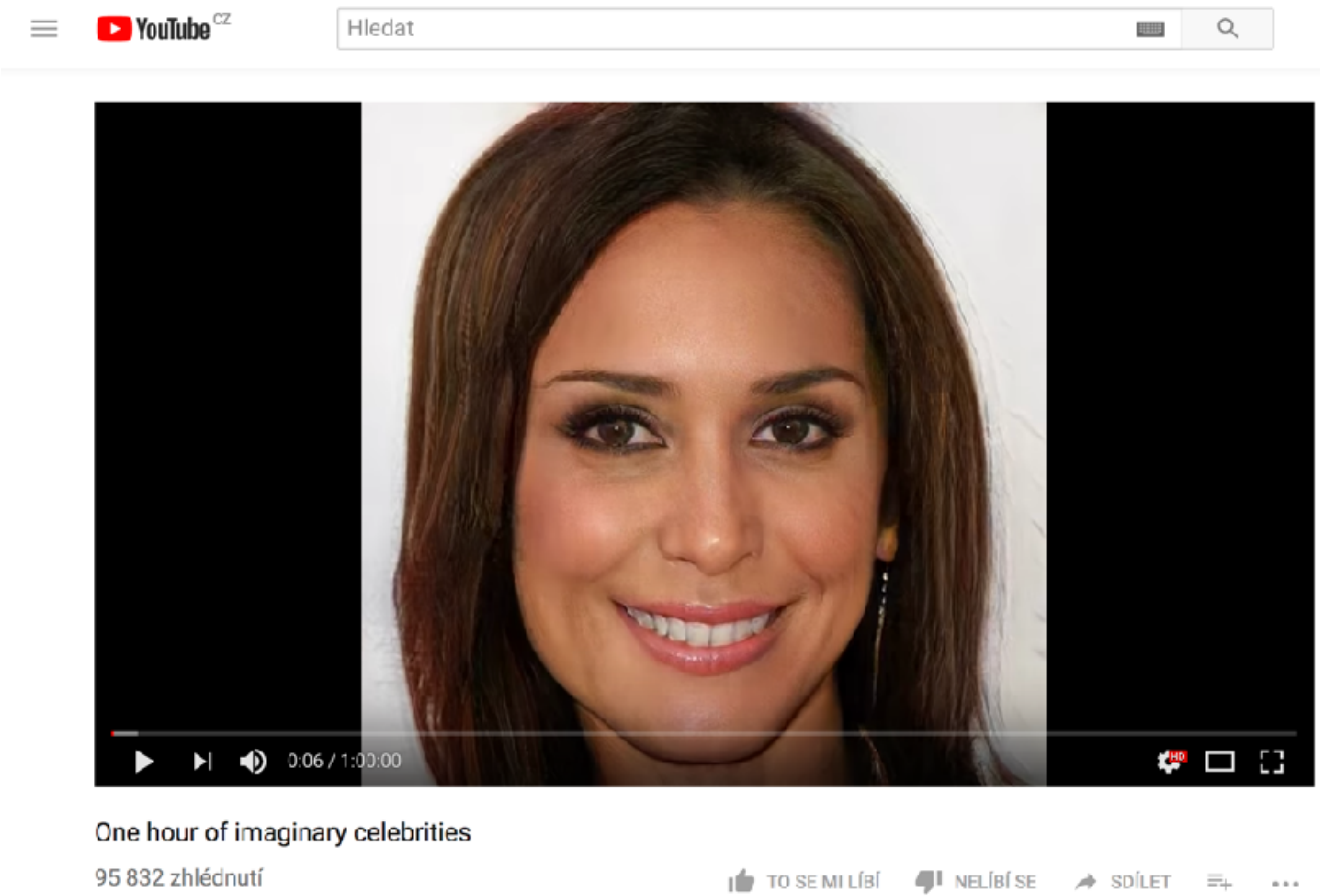
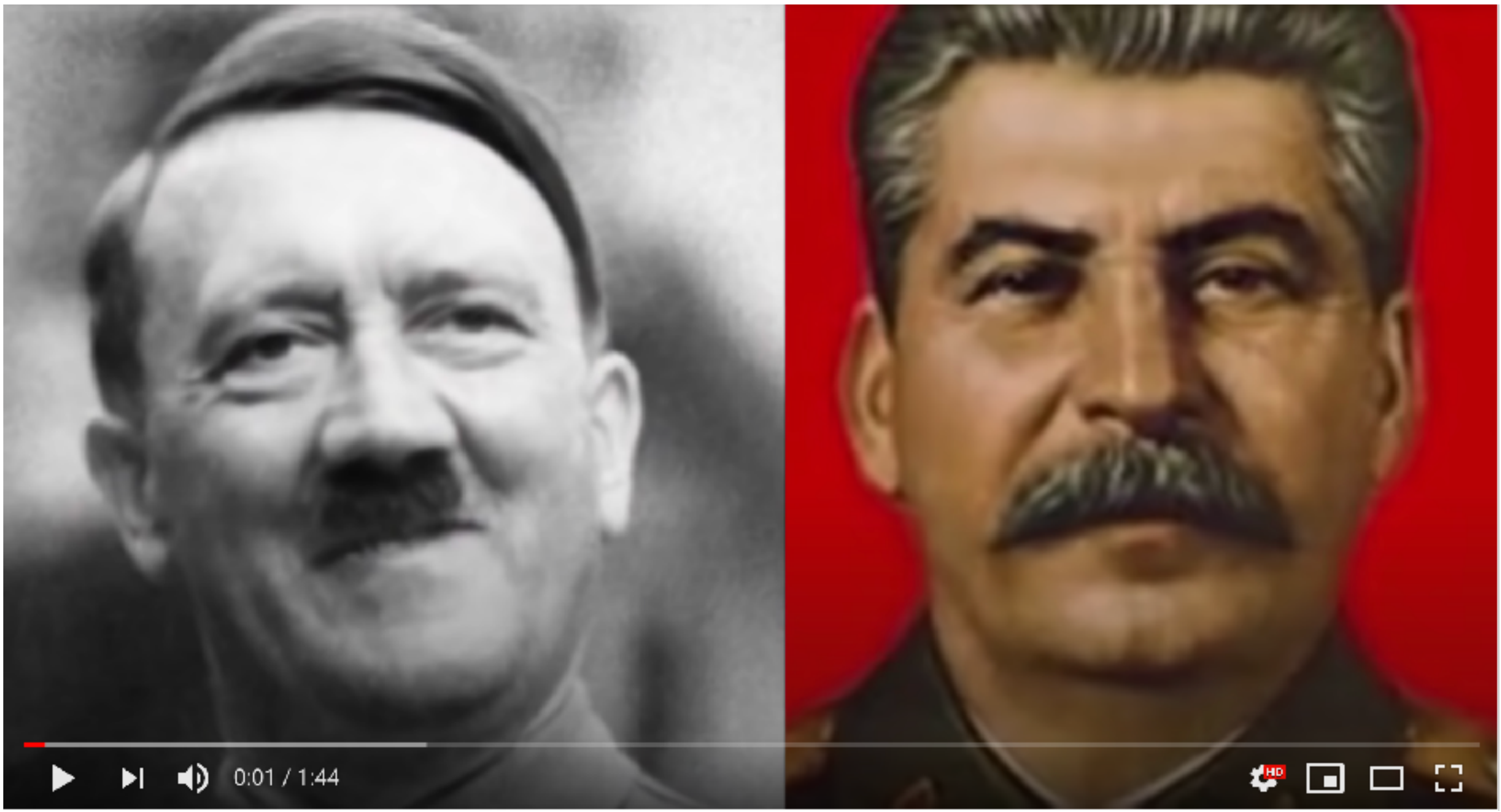


Image synthesis



Which one is fake?

Image manipulation



Diffusion models

vibrant portrait painting of Salvador Dalí with a robotic half face

a shiba inu wearing a beret and black turtleneck

a close up of a handpalm with leaves growing from it

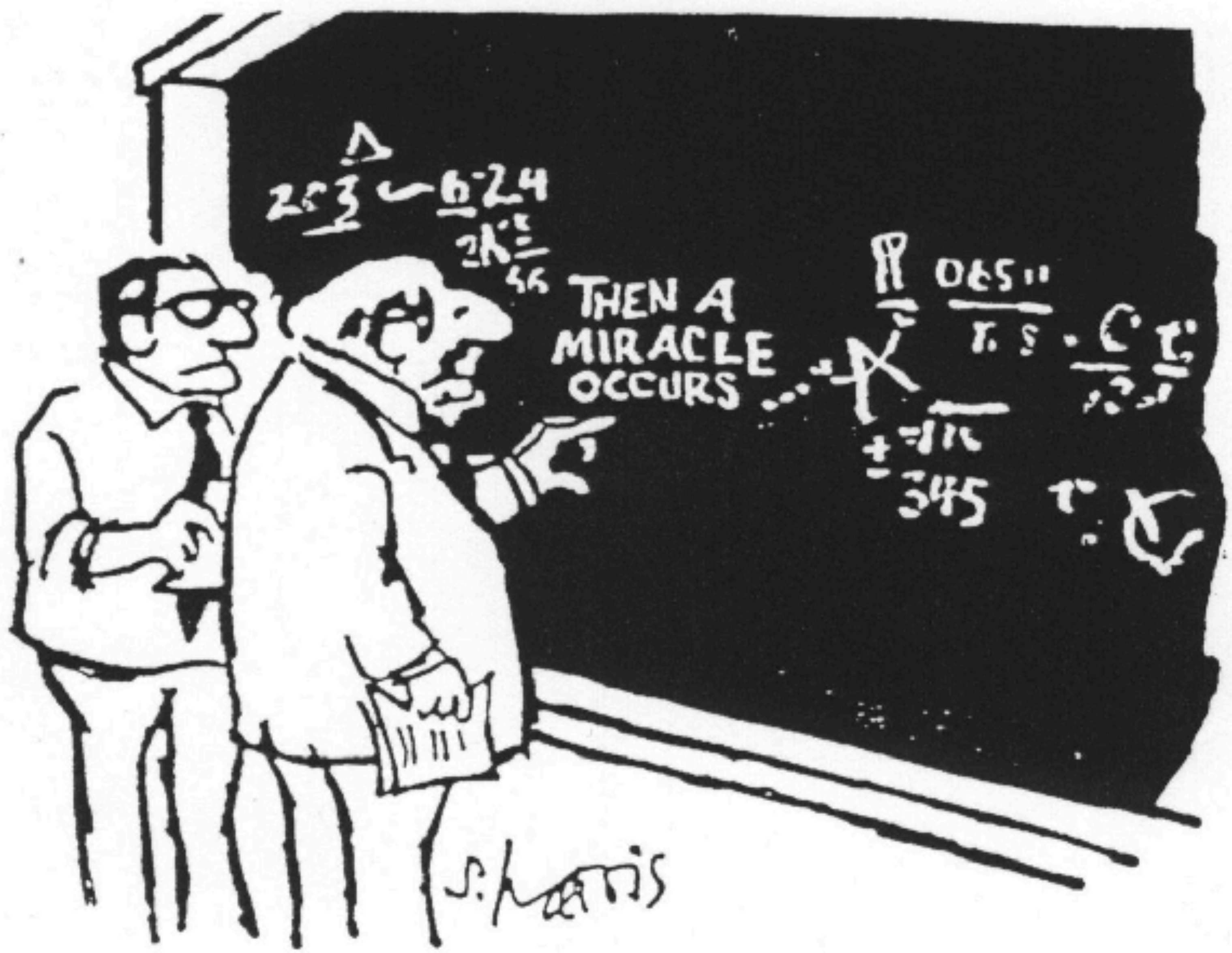
an espresso machine that makes coffee from human souls, artstation

panda mad scientist mixing sparkling chemicals, artstation

a corgi's head depicted as an explosion of a nebula

Stable Diffusion: https://huggingface.co/spaces/stabilityai/stable-diffusion

Neural network explainability



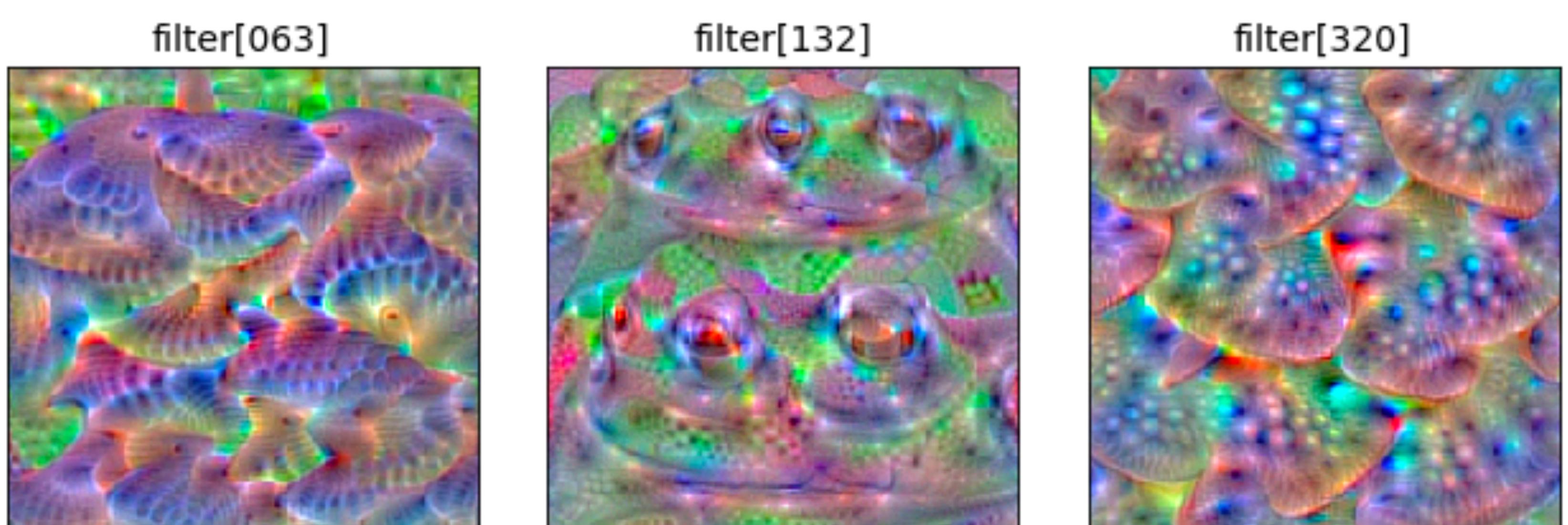
I think you should be a little more specific, here in Step 2

https://analyticsindiamag.com/deep-learning-is-not-so-much-a-black-box-anymore-and-thats-a-great-development/

Activation Maximization

Visualized output classification Layer

Visualized hidden convolutional layers

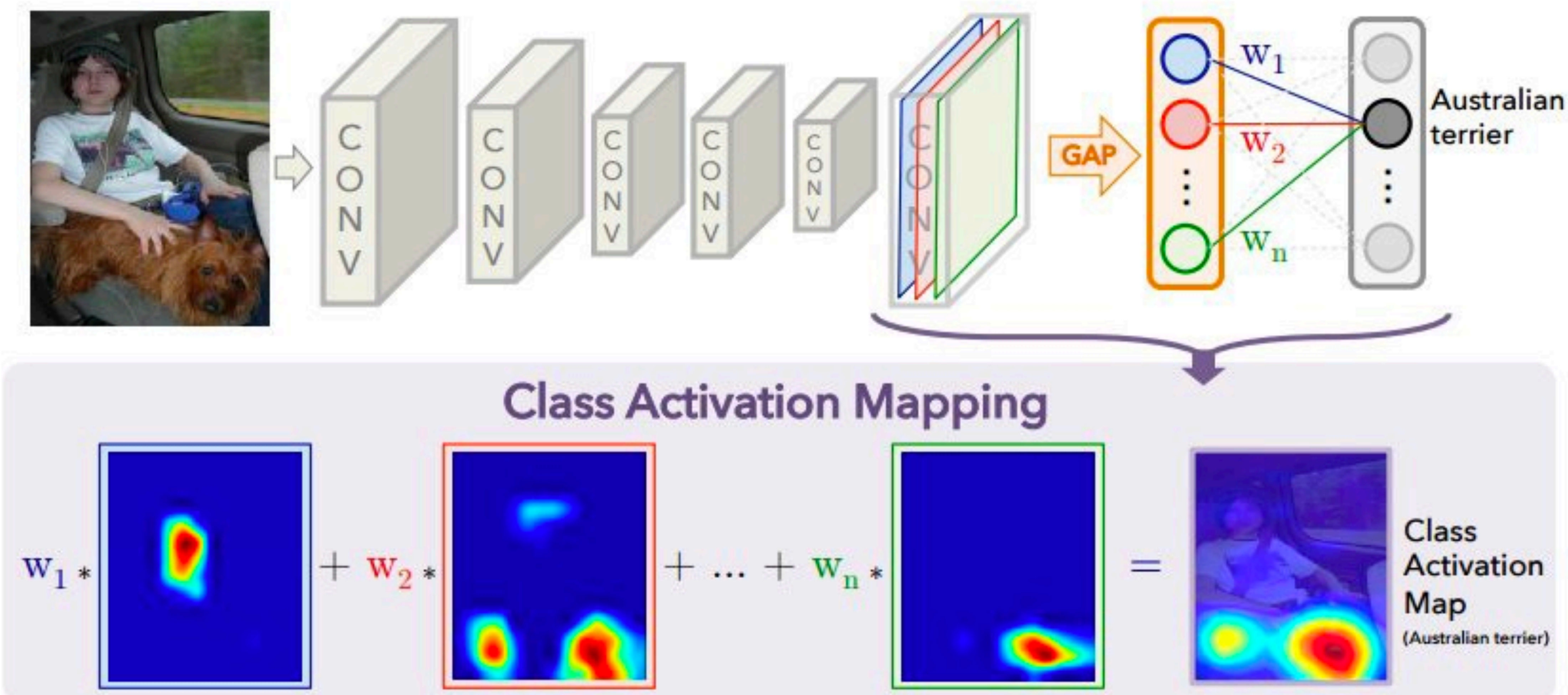


https://pypi.org/project/tf-keras-vis/

Grad-CAM heat maps

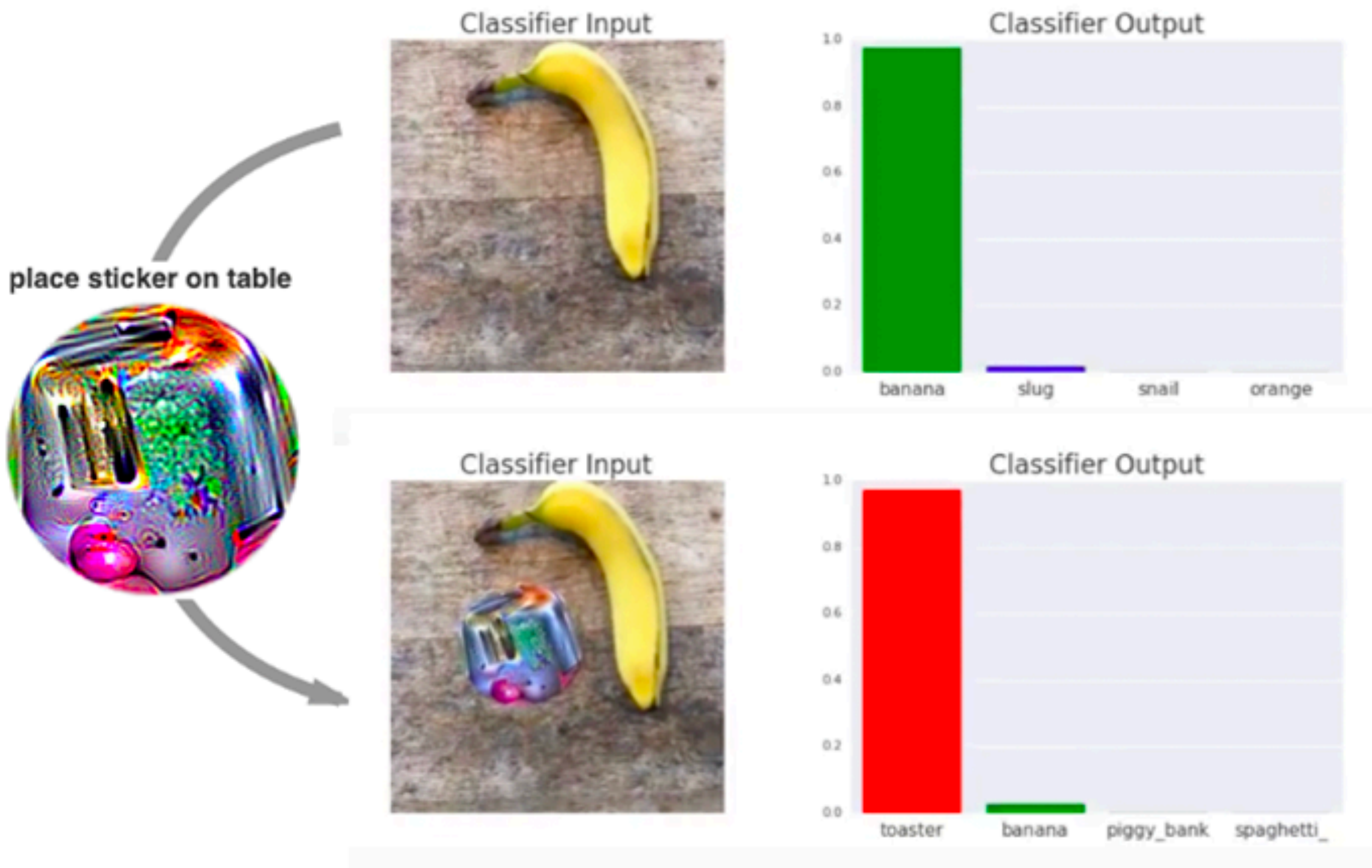
https://keras.io/examples/vision/grad_cam/

CAM heat maps



https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-class-activation-maps-fe94eda4cef1

Adversarial Patch



What next?

https://www.mlcollege.com/en/#courses

Machine Learning Prague

ML MU MACHINE LEARNING meetups

Thank you for your attention

e-mail: jiri@mlcollege.com

Web: www.mlcollege.com

Twitter: @JiriMaterna

Facebook: https://www.facebook.com/maternajiri

LinkedIn: https://www.linkedin.com/in/jirimaterna/