Deep Computer Vision

ICS/DATA 435 and ICS 635

Spring 2023

Tried and True Classical ML Methods

- Linear and Logistic Regression
- Decision Trees
- Random Forests
- Support Vector Machines
- Naïve Bayes

When to use:

- Small dataset
- Simple features (e.g., 100 columns in an Excel table rather than high resolution images or video)

Tried and True DL Methods

To Be Determined!

Better methods are coming out every year (and more frequently than that!)

Therefore, for the remainder of this class:

- Focus on big ideas rather than the details of any specific method
- See HW5 Question 1 for examples of the level of detail to take away (for the exam and in general)
- Therefore, we will be switching to slides rather than iPad writing (unless there is high student demand to switch back)

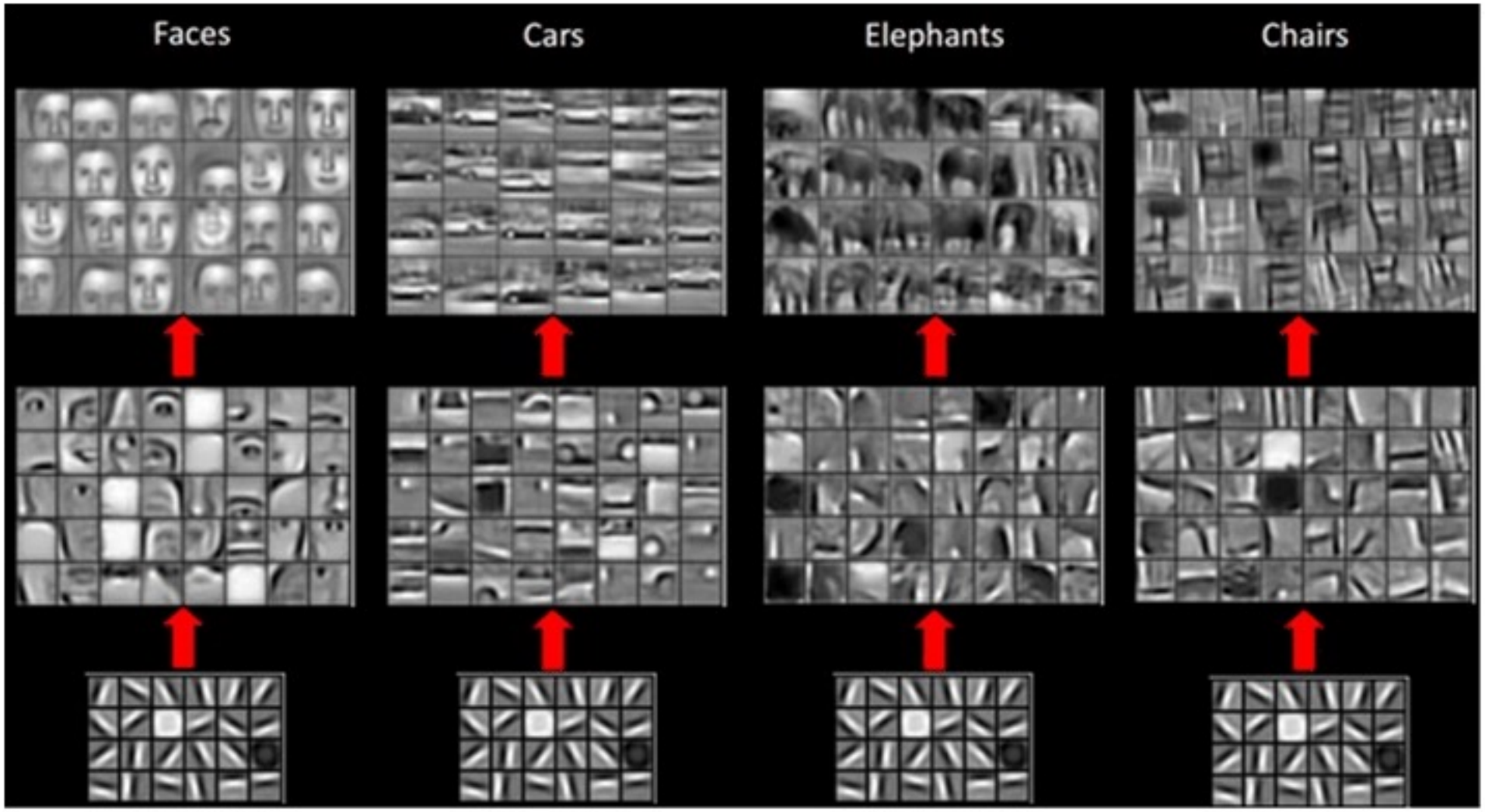
A lot of these methods are developed by creative graduate students

Take an existing method and tweak it slightly based on intuition (mathematical, creative, or otherwise)

What's to say that your ideas for alternative approaches wouldn't work or be better as well?

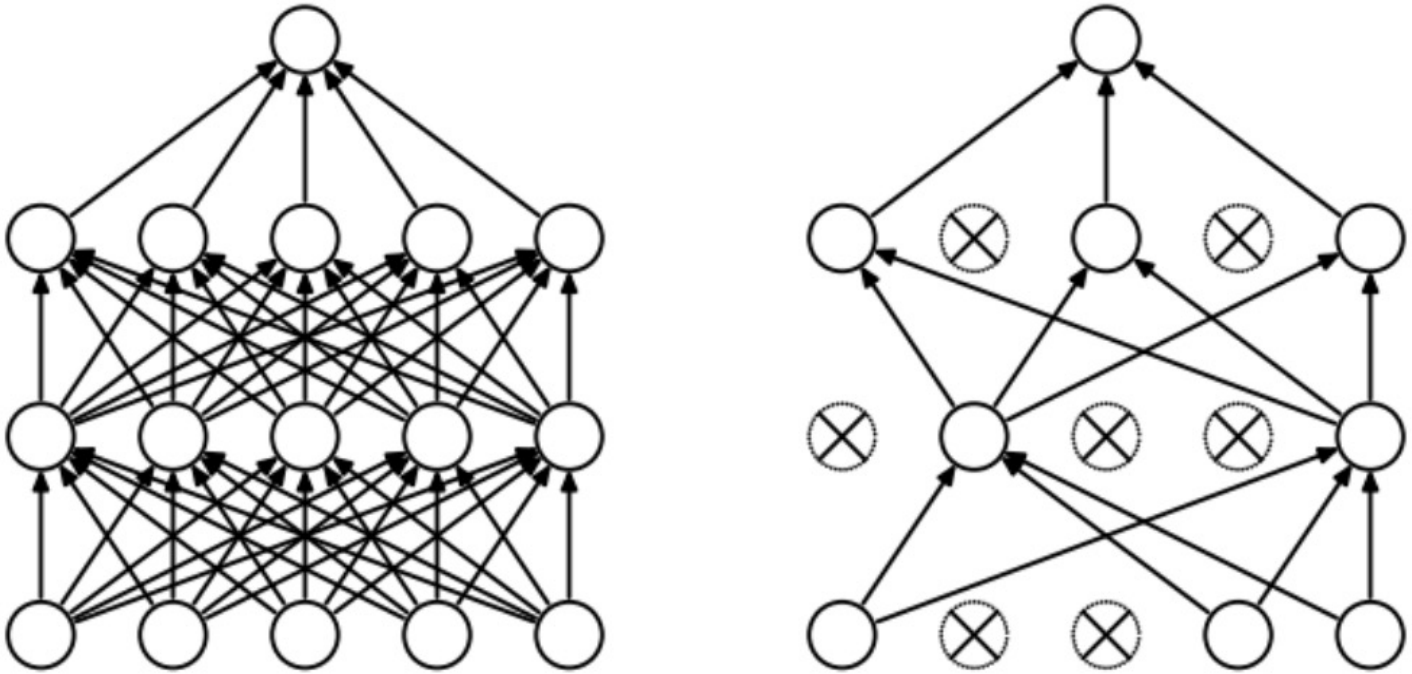
Review of Concepts to Know from Coding Lectures

Visualizing Feature Maps



Regularization for Neural Nets: Dropout

Dropout nodes with probability p (hyperparameter) during each step of training. Helps prevent overfitting.



(a) Standard Neural Net

(b) After applying dropout.

Saliency Maps

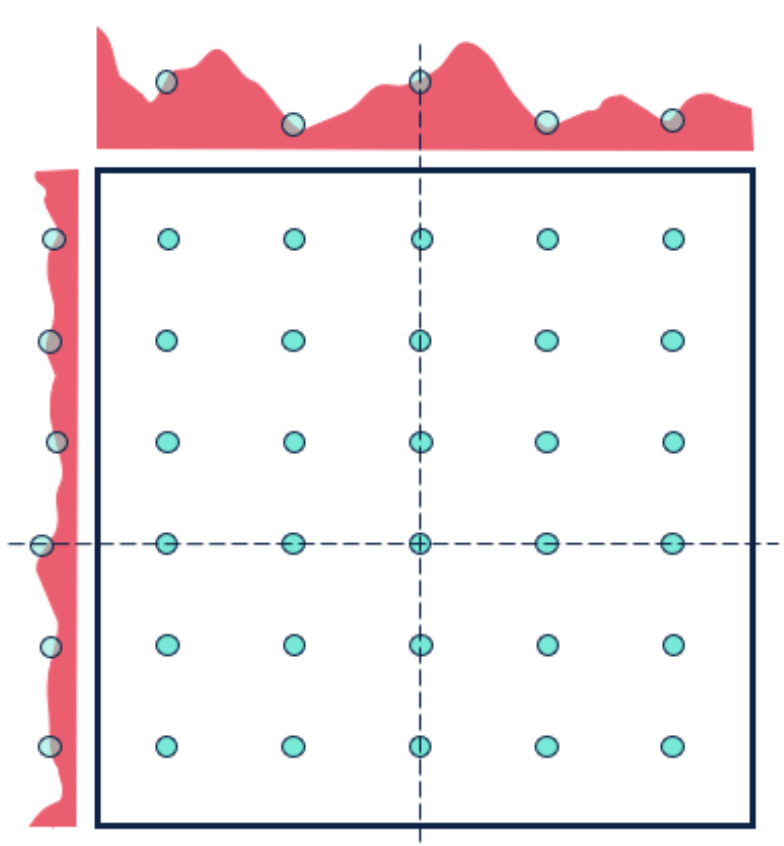
(a) Husky classified as wolf

(b) Explanation

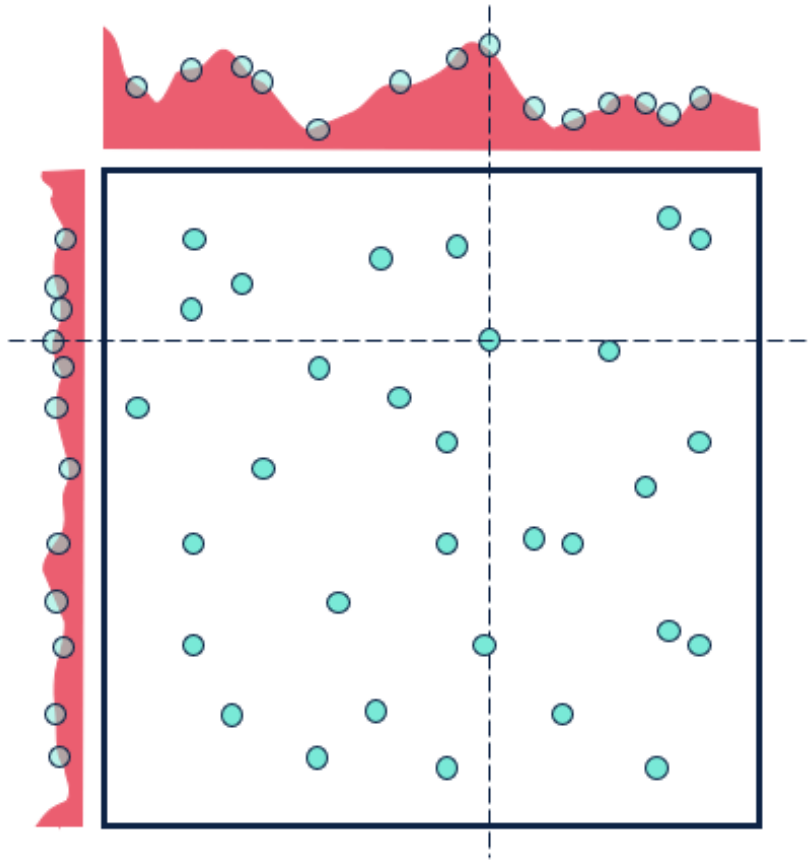
$$D = |\frac{\Delta probability}{\Delta pixel}|$$

- Consider each pixel value in turn: R, G, B, then the next pixel.
- Make a copy of the image array before you change anything!
- Make the pixel value larger or smaller by various amounts. Each time, find the CNN's prediction with the changed value, and calculate the value of D.
- Repeat the previous step a few times, and calculate the pixel's saliency: the average value of D.
- Store the saliency of each pixel in a list, so that we can visualize it later.

Hyperparameter Optimization



Grid Search



Random Search

How to implement…

- A specified neural network architecture
- Training and evaluating a neural network
- Transfer learning

Important NN Decisions (besides the architecture)

(e.g., for Homework 5)

```
def train(model, x, y, x_val, y_val, lr, epochs):
    log_dir = "logs/" + model.name + "/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
    print ("MODEL NAME:", model.name)
    optimizer = tf.optimizers.Adam(learning_rate=lr)
    model.compile(
        optimizer=optimizer,
        loss=tf.keras.losses.BinaryCrossentropy(),
        metrics=["accuracy"],
    )
    return model.fit(
        x,
        y,
        epochs=epochs,
        validation_data=(x_val, y_val),
        callbacks=[tensorboard_callback],
        verbose=1,
    )
```

(1) Learning rate
(2) Optimizer (which fancy variation of Gradient Descent to use)
(3) Loss function

(4) How many epochs to train for

(5) When to stop training (can be specified in a callback function)

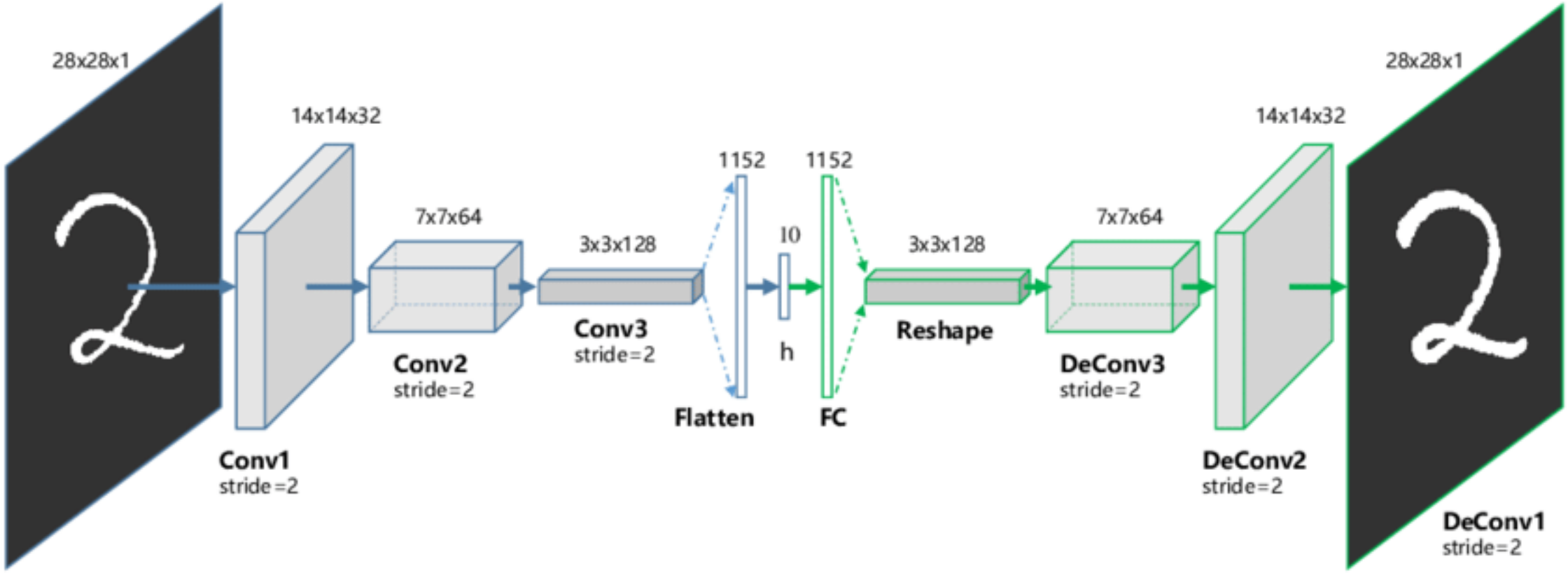
Semantic Segmentation

Goal of Semantic Segmentation

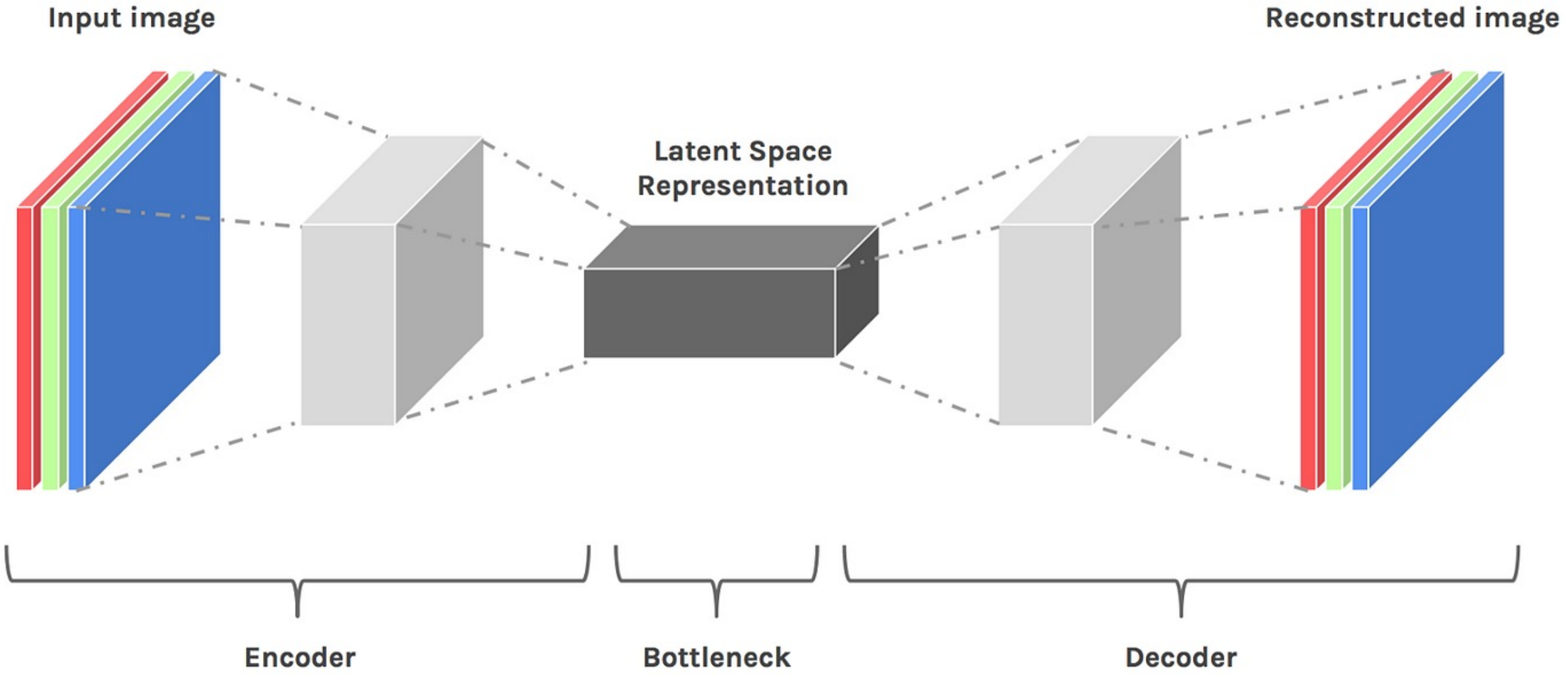
predict

Person
Bicycle
Background

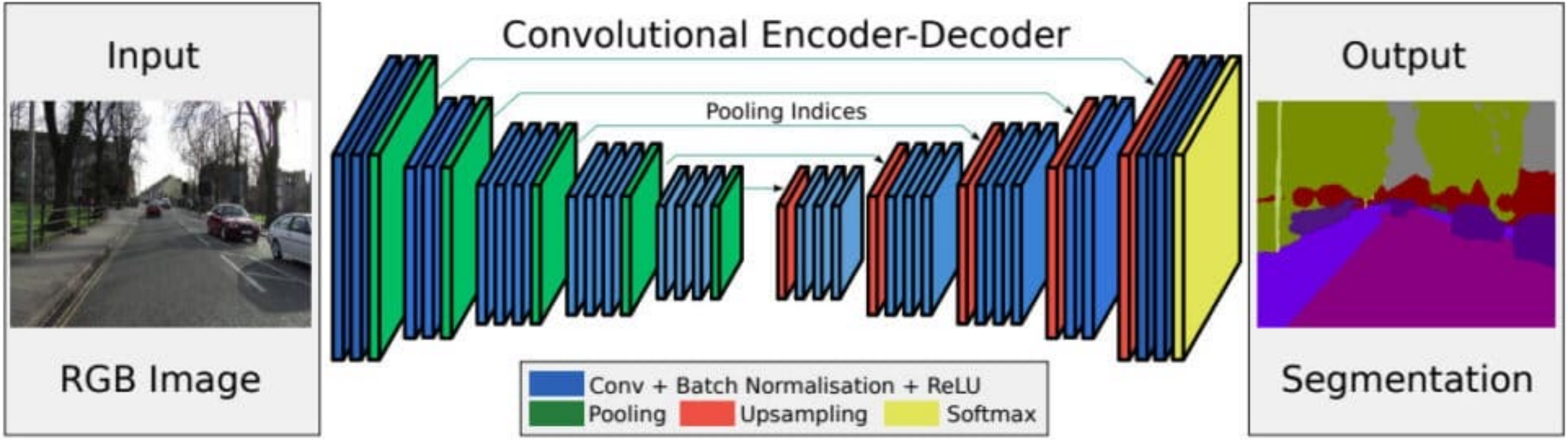
Recall: Convolutional Autoencoders



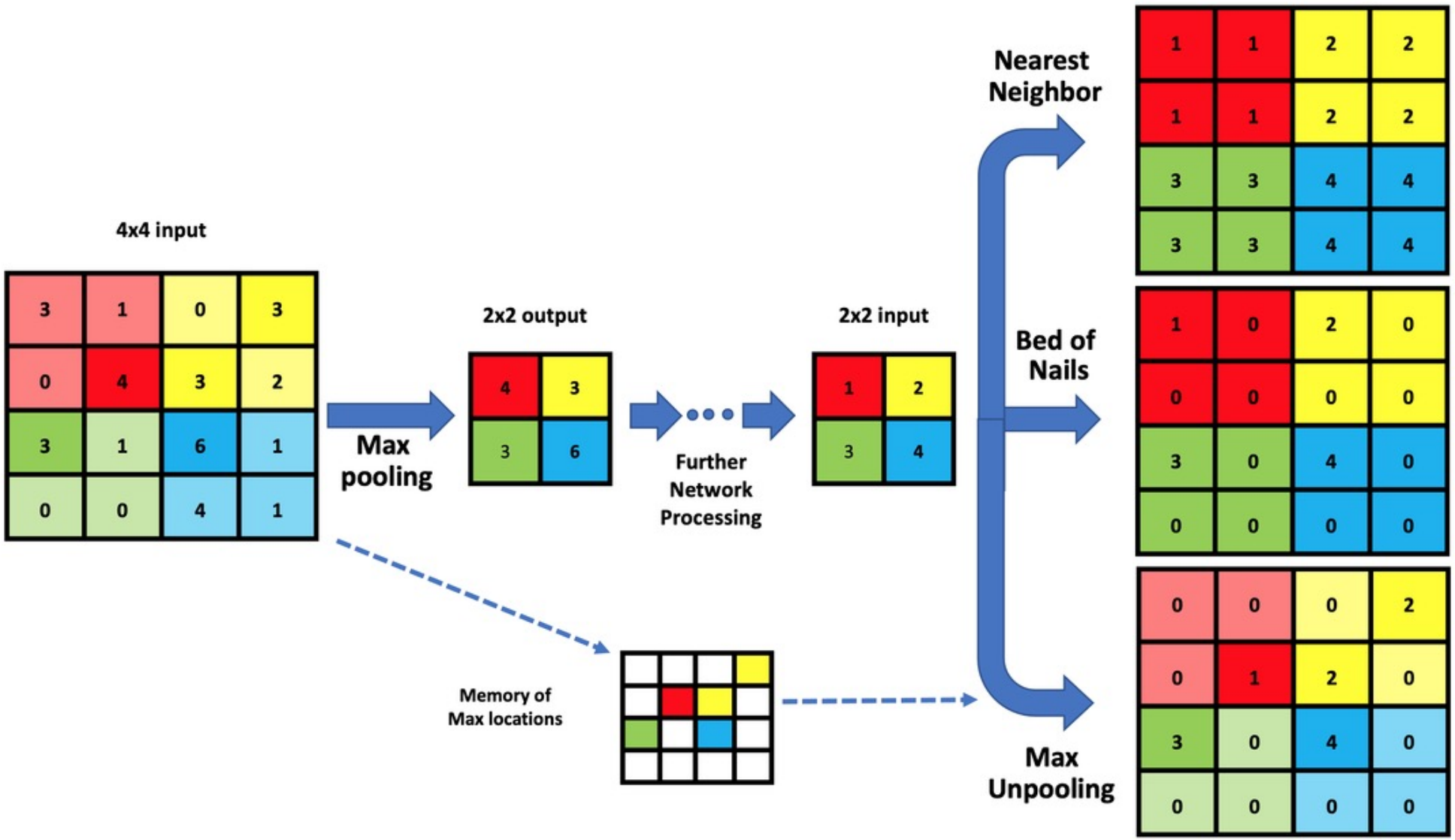
Recall: Convolutional Autoencoders



Semantic Segmentation: Convolutional Autoencoders



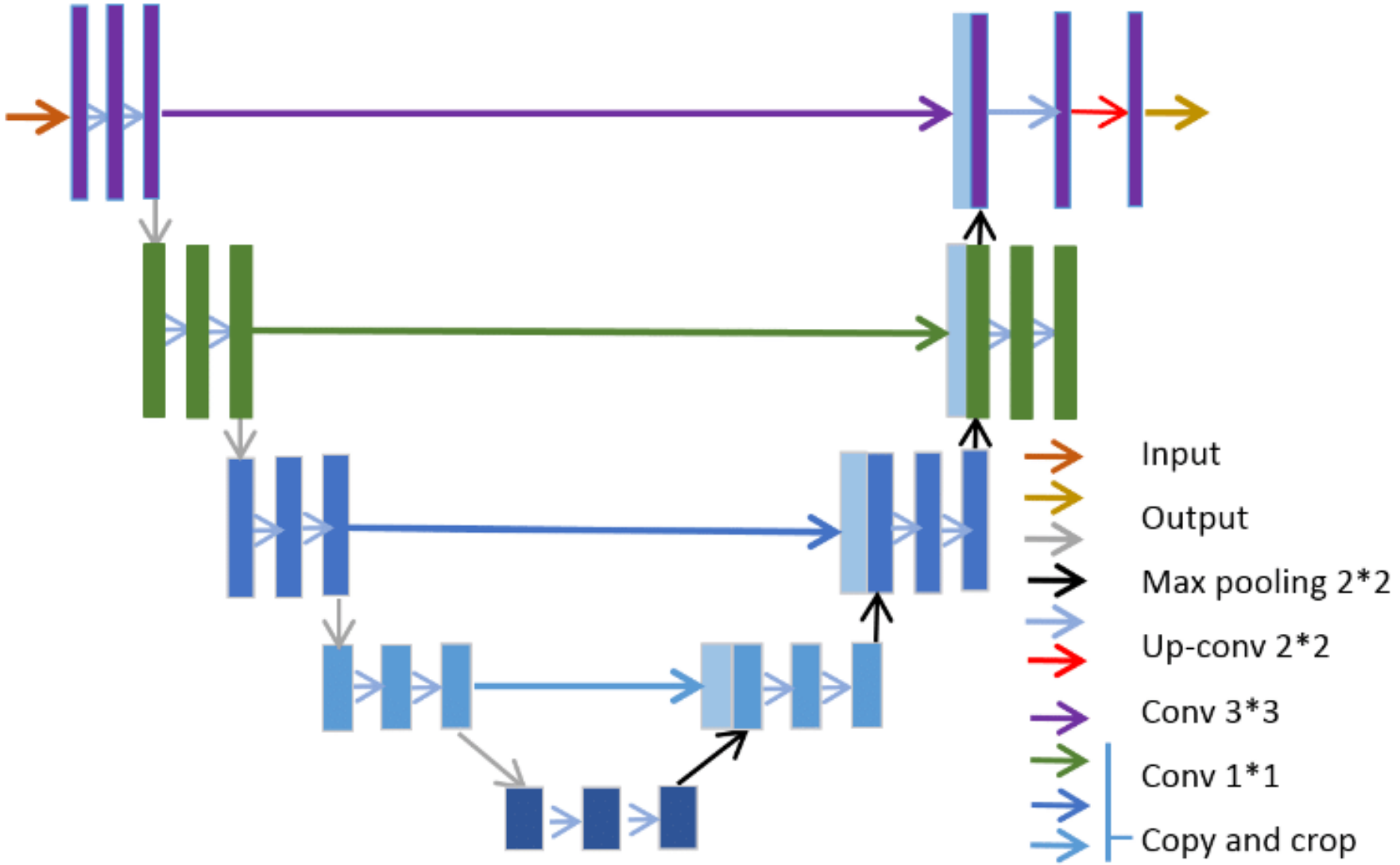
Upsampling in a CNN



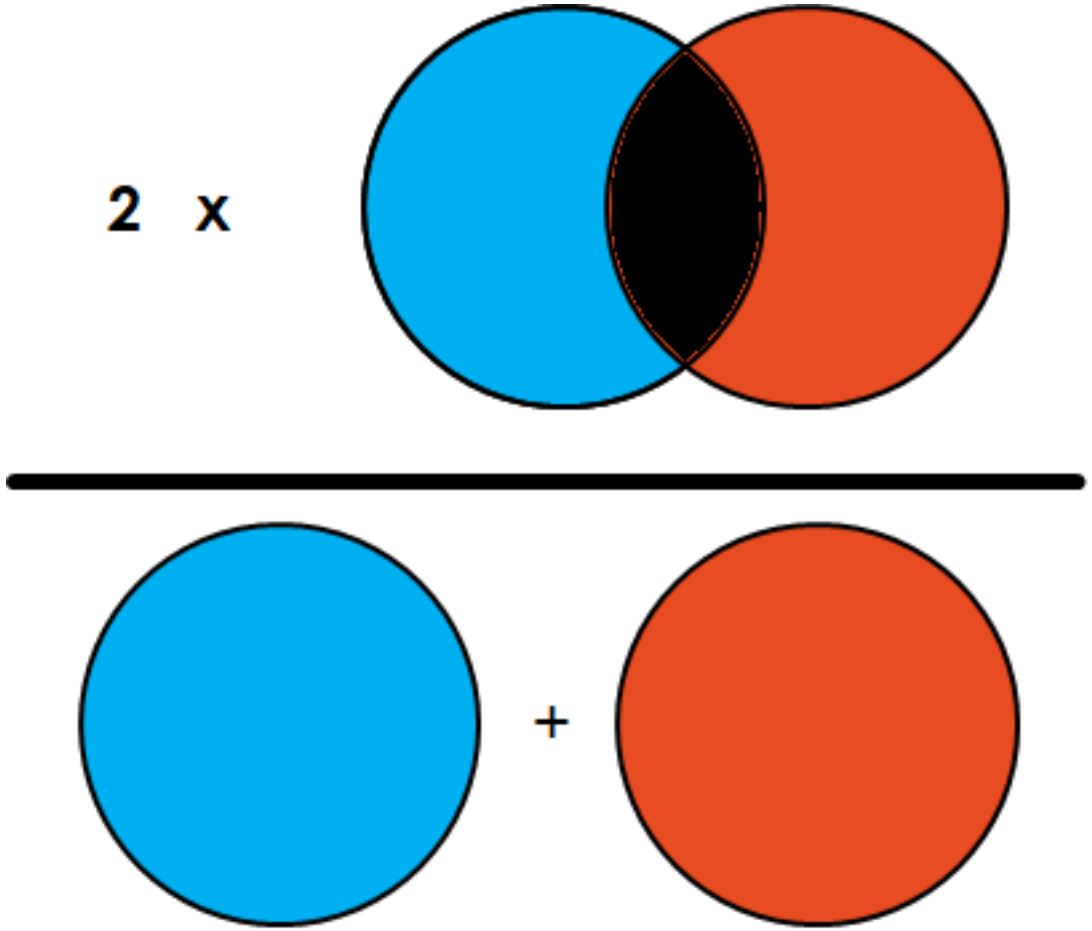
CNN Upsampling Visualization

https://github.com/vdumoulin/conv_arithmetic

Popular Segmentation Architecture: U-Net



Segmentation Loss Function: Dice Coefficient



$$\frac{2 * |T \cap P|}{|T| + |P|}$$

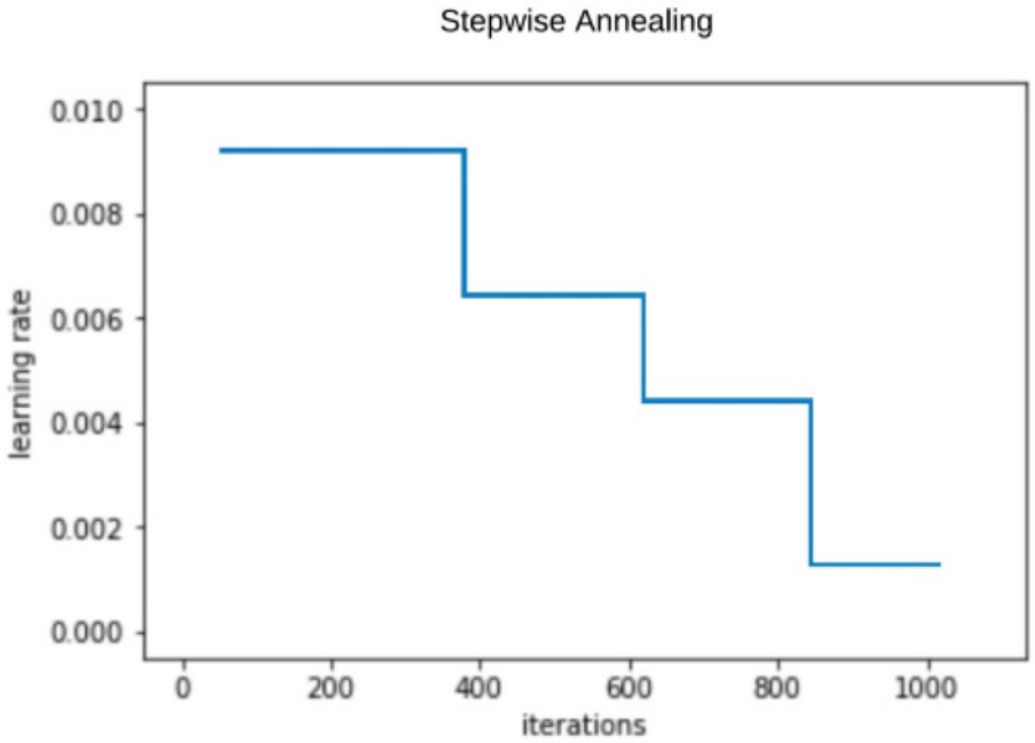
U-Net TensorFlow Code (from Segmentation notebook)

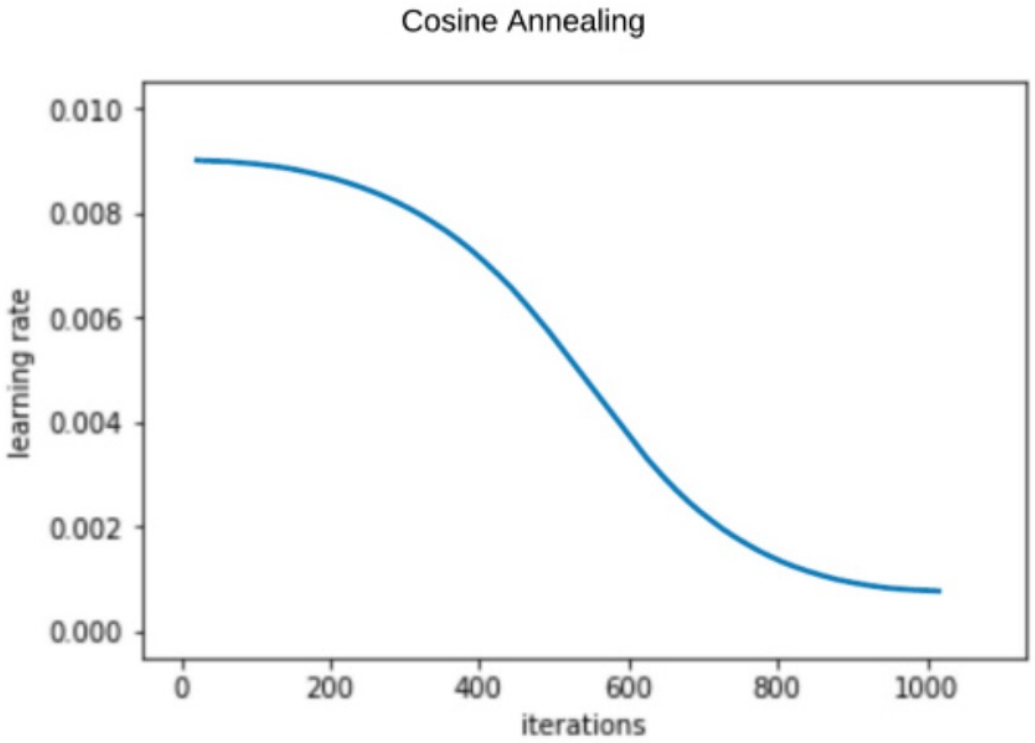
```
def section(conv1_filters, conv2_filters): #Represents one horizontal "section" of the U
    return tf.keras.Sequential(
        [
            Conv2D(conv1_filters, 3, padding="same", activation="relu"),
            Conv2D(conv2_filters, 3, padding="same", activation="relu"),
        ]
    )

class U_Net(tf.keras.Model):
    def __init__(self):
        super(U_Net, self).__init__() #What do the numbers represent?
        self.section1 = section(16, 16)
        self.section2 = section(32, 32)
        self.section3 = section(32, 64) #Bottom of the U!
        self.section4 = section(32, 32)
        self.section5 = section(16, 16)
        self.final_conv = Conv2D(1, 3, padding="same", activation="sigmoid")
        self.maxpool1, self.maxpool2 = MaxPool2D(2), MaxPool2D(2) #Why are there two of these?
        self.upsample1, self.upsample2 = UpSampling2D(2), UpSampling2D(2)

    def call(self, inputs):
        input1 = self.section1(inputs)
        input2 = self.section2(self.maxpool1(input1))
        input3 = self.section3(self.maxpool2(input2))
        input4 = self.section4(concatenate([input2, self.upsample1(input3)]))
        input5 = self.section5(concatenate([input1, self.upsample2(input4)]))
        output = self.final_conv(input5)
        return output
```

```
[ ] unet = U_Net()
```

This is an example of how to go about implementing more complex NN architectures in TensorFlow!

Learning Rate Annealing

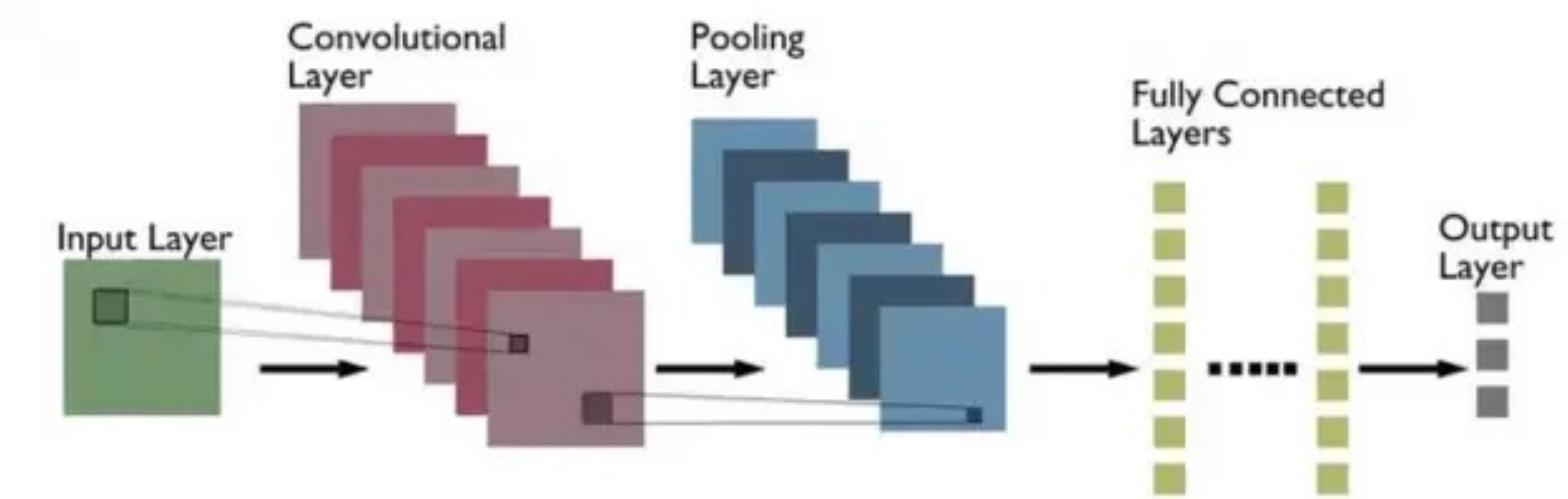




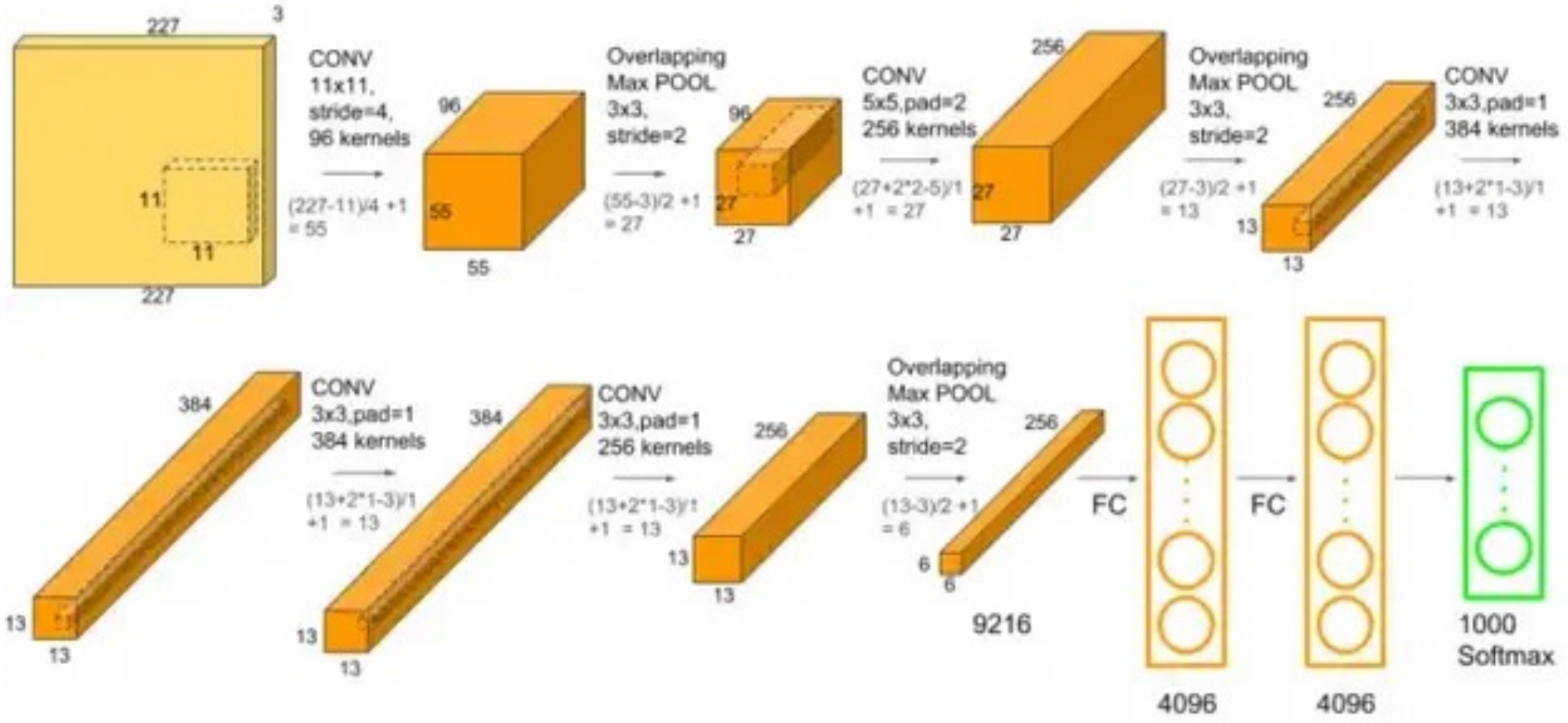
Why would we want to do this? Make sure you know.

CNN Architectures

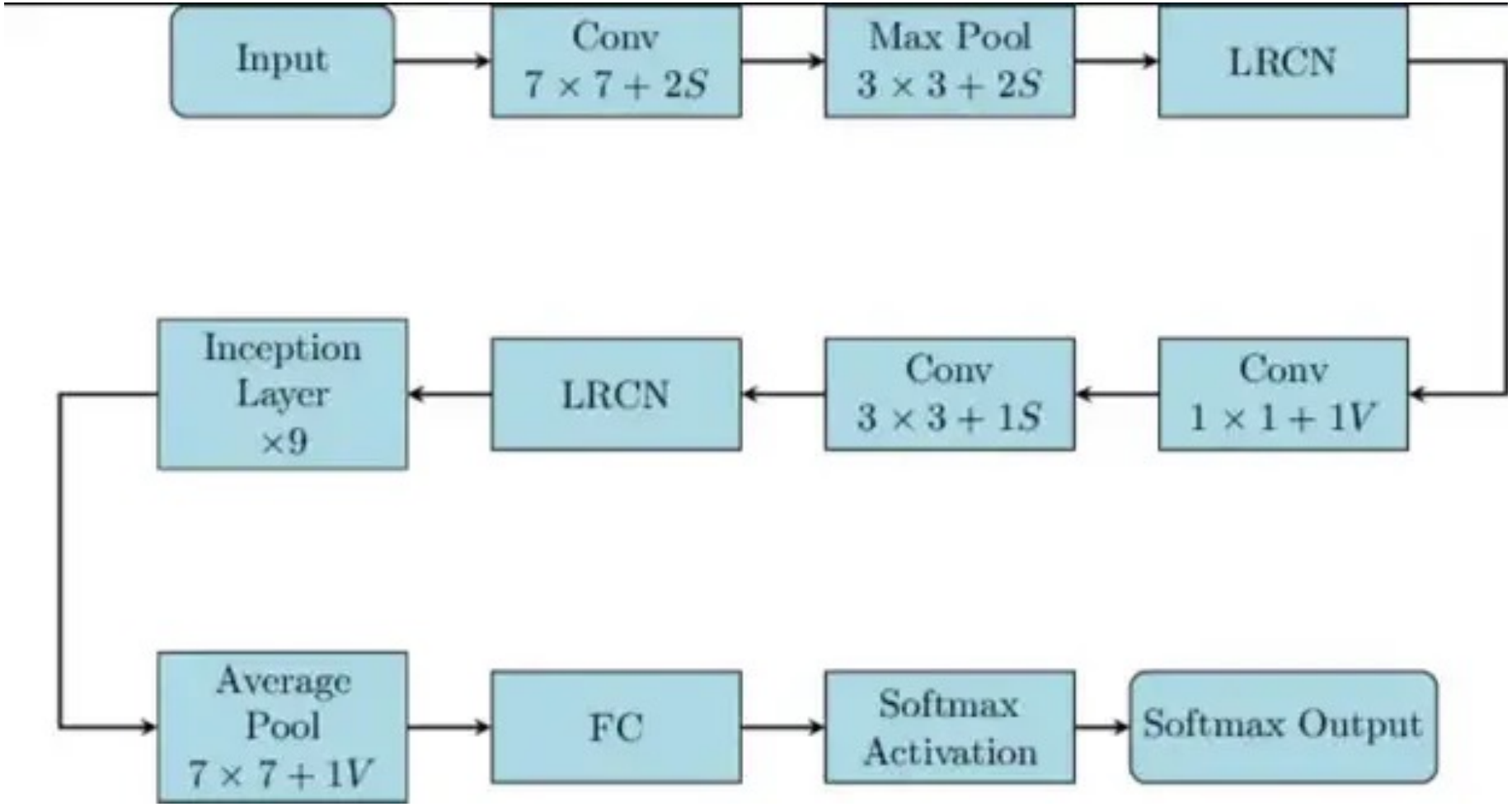
Prototypical CNN



AlexNet (2012)



GoogLeNet (2014)



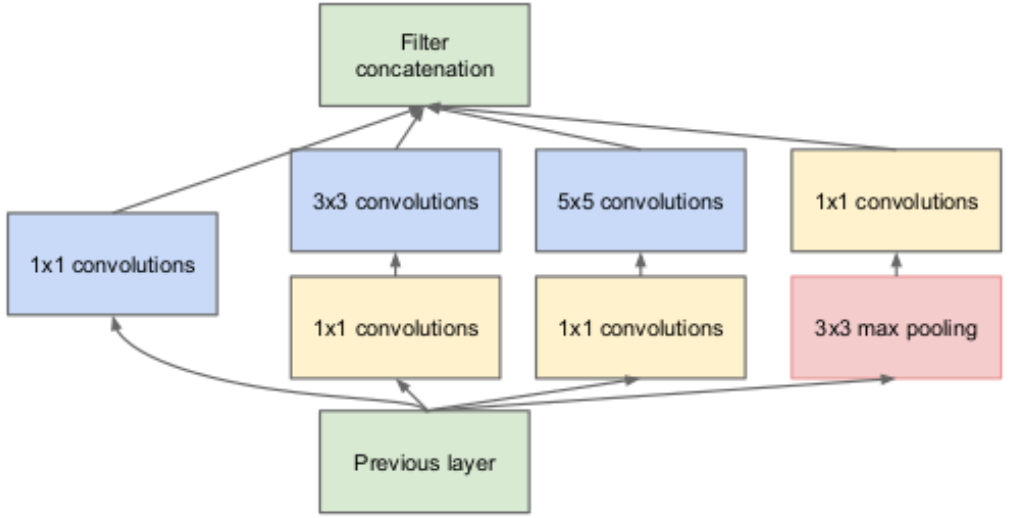
New Layers

Long-term Recurrent Convolutional (LRCN) layer

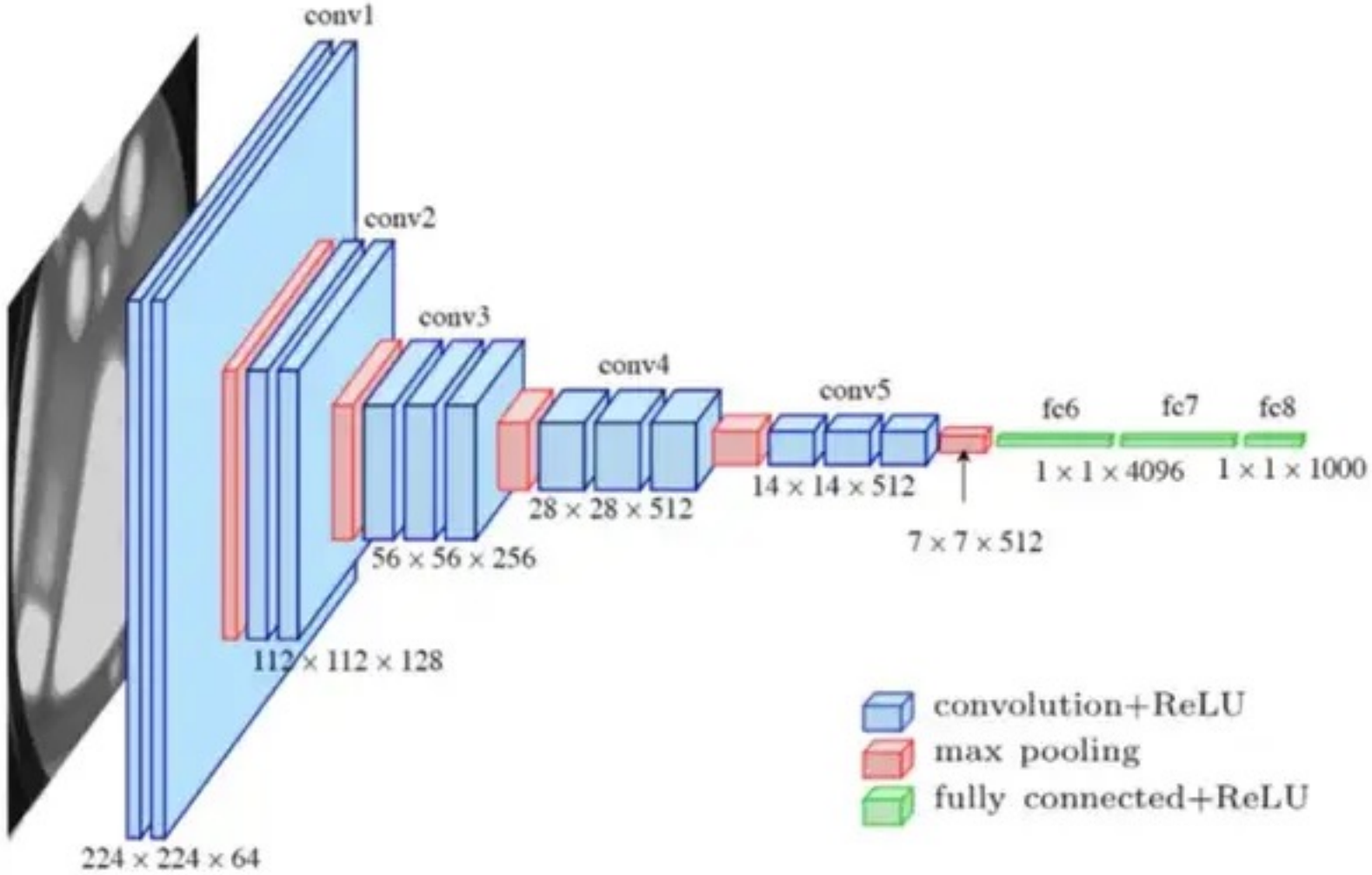
- Will make more sense after next lecture

Inception layer

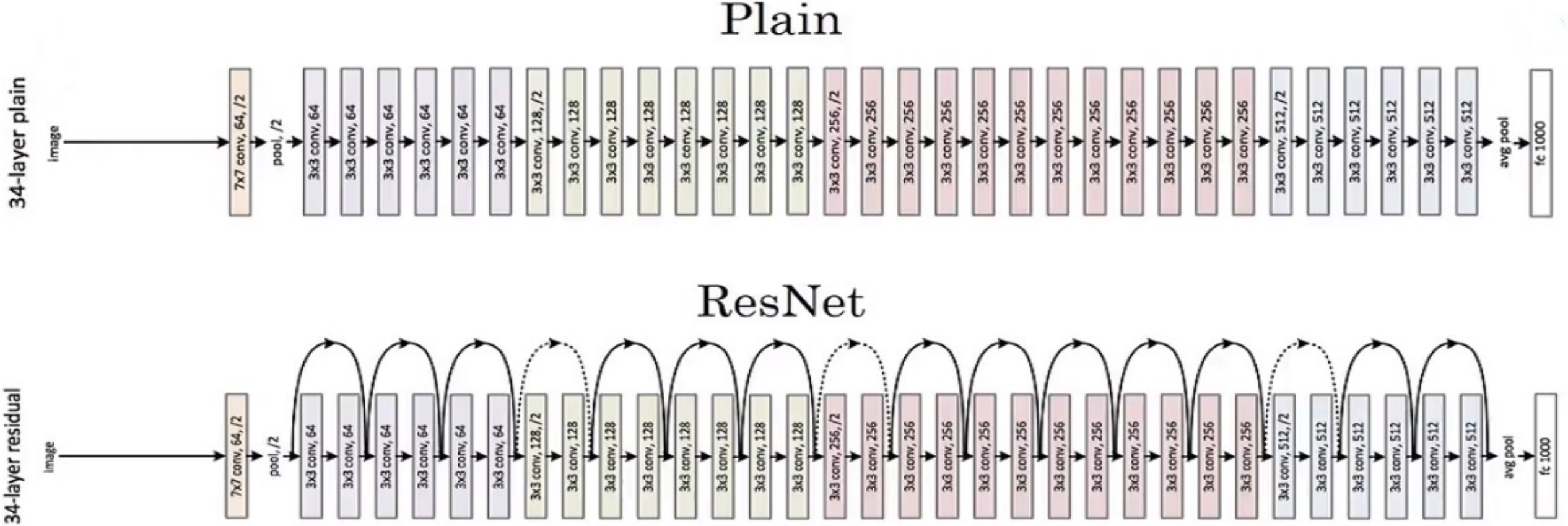
- allows the internal layers to pick and choose which filter size will be relevant to learn the required information



VGGNet (2014)


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
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

ResNet (2016-17)

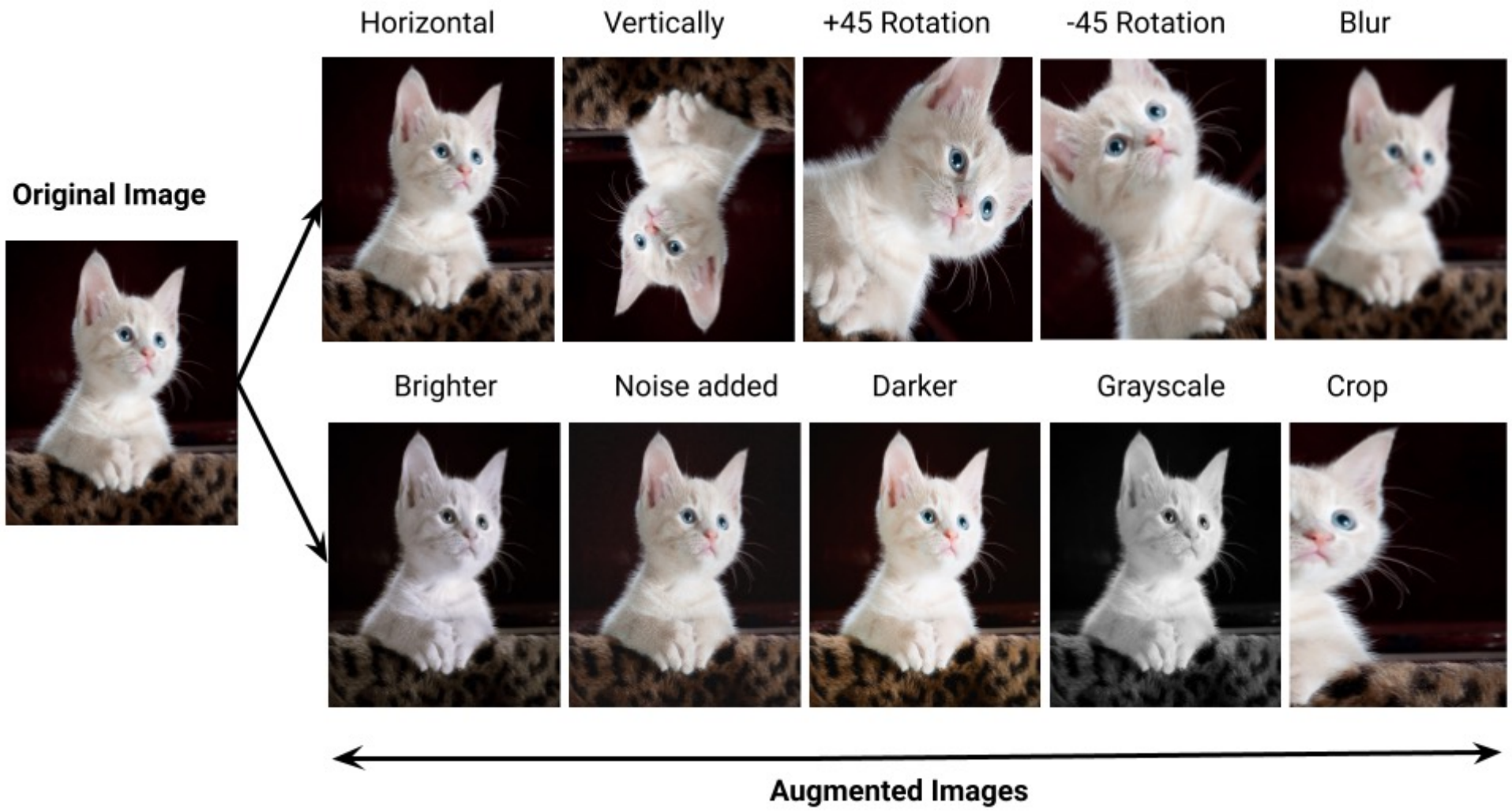


Data Augmentation

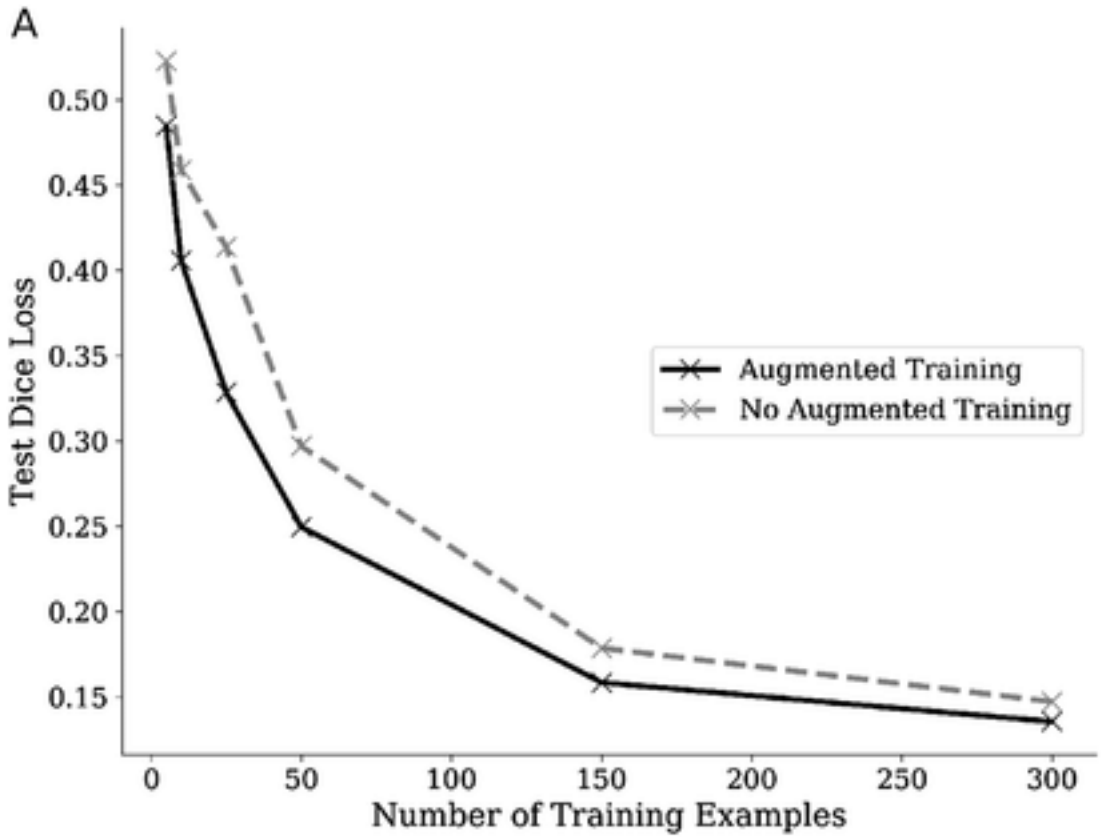
What is the issue if we train a parrot classifier using pictures of parrots taken at 1pm each day in the same location?

Data Augmentation: create many versions of the same image using data preprocessing

Common Augmentation in Computer Vision



Effect of Data Augmentation



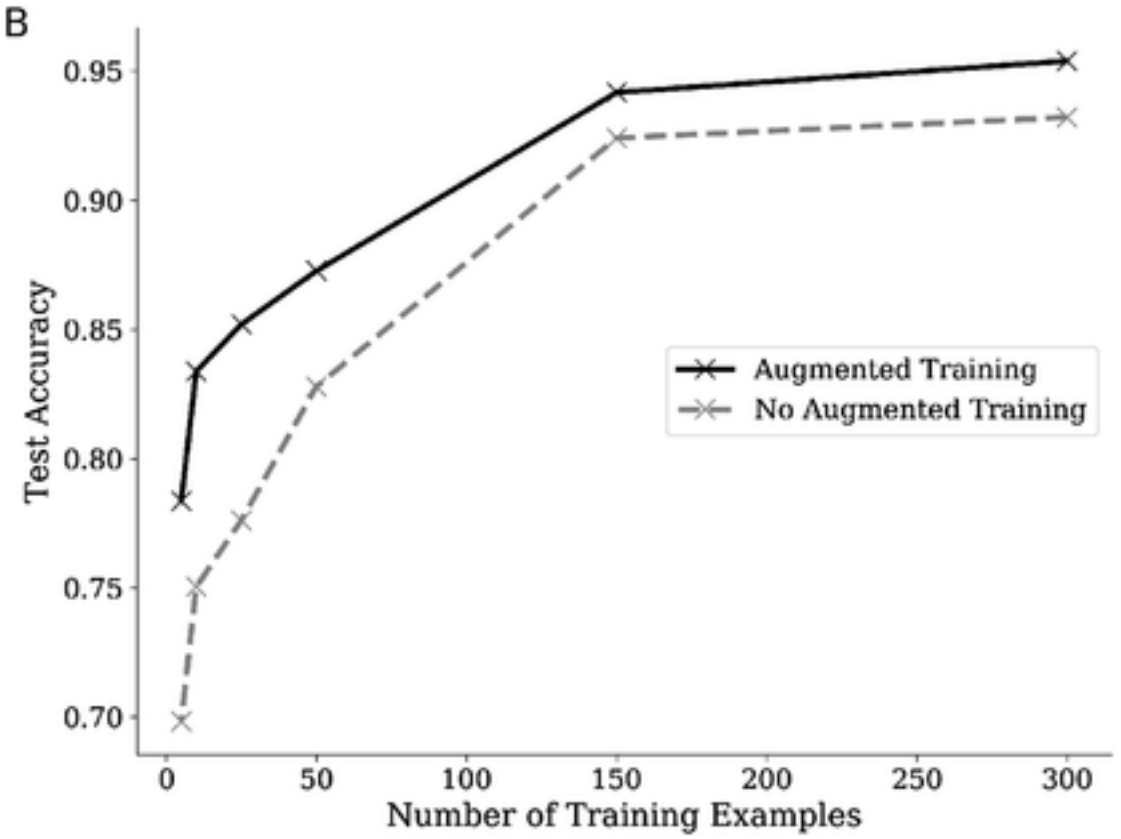


Image Data Augmentation in TensorFlow

```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=0,
    width_shift_range=0.0,
    height_shift_range=0.0,
    brightness_range=None,
    shear_range=0.0,
    zoom_range=0.0,
    channel_shift_range=0.0,
    fill_mode='nearest',
    cval=0.0,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=None,
    validation_split=0.0,
    interpolation_order=1,
    dtype=None
)
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = utils.to_categorical(y_train, num_classes)
y_test = utils.to_categorical(y_test, num_classes)
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)
# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)
datagen.fit(x_train)
# fits the model on batches with real-time data augmentation:
model.fit(datagen.flow(x_train, y_train, batch_size=32,
         subset='training'),
         validation_data=datagen.flow(x_train, y_train,
         batch_size=8, subset='validation'),
         steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

Image Data Augmentation in PyTorch

```
train_transforms = transforms.Compose([
                                transforms.Resize(256),
                                transforms.ColorJitter(),
                                transforms.RandomCrop(224),
                                transforms.RandomHorizontalFlip(),
                                transforms.Resize((32,32)),
                                transforms.ToTensor(),
                                transforms.Normalize(mean=(0.5,), std=(0.5,))])
```

```
class CatsDogsDataSet(Dataset):
    def __init__(self, train_dir, transform = train_transforms):
        self.train_dir = train_dir
        self.transform = transform
        self.images = []
        self.labels = []
        for fname in os.listdir(train_dir):
            self.images.append(fname)
            if 'cat' in fname.split('.')[0]:
                self.labels.append(1)
            else:
                self.labels.append(0)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img = Image.open(os.path.join(self.train_dir, self.images[idx]))
        if self.transform is not None:
            img = self.transform(img)
        else:
            img = np.array(img).astype('float32')
        return img, self.labels[idx]

    def split(self, start, end):
        return self.labels[start:end+1]
```

Deep Learning Implementation

Two Main Frameworks: TensorFlow and PyTorch

PyTorch

TensorFlow

Two Main Frameworks: TensorFlow and PyTorch

TensorFlow

- Developed by Google
- Most popular DL framework in industry
- (Tiny sample of) examples of users: Google, Airbnb, Coca Cola, GE Healthcare, Intel, Lenovo, Paypal, Qualcomm, Spotify, Texas Instruments, Twitter, Uber, JPMorgan Chase, Capital One, SeatGeek, WeightWatchers, Credit Karma, Grammarly, Dropbox, Upwork, Boeing, Liberty Mutual Insurance, Codeacademy, Box, McDonald's, Merck, McKinsey, Unity, DE Shaw, Tempus, Nike, DoorDash, Etsy, …

PyTorch

- Developed by Meta
- Most popular DL framework in research
- (Tiny sample of) examples of users: Meta (Facebook, Instagram), academic research labs, OpenAI, Microsoft, Toyota, Tesla, Lyft, Walmart, NVIDIA, Wells Fargo, Airbnb, Genentech, …

In practice, you can know either in terms of getting jobs (job requirements like "Development experience in deep learning frameworks such as PyTorch or TensorFlow")

Two Main Frameworks: TensorFlow and PyTorch

TensorFlow

- Developed by Google
- Most popular DL framework in industry
- (Tiny sample of) examples of users: Google, Airbnb, Coca Cola, GE Healthcare, Intel, Lenovo, Paypal, Qualcomm, Spotify, Texas Instruments, Twitter, Uber, JPMorgan Chase, Capital One, SeatGeek, WeightWatchers, Credit Karma, Grammarly, Dropbox, Upwork, Boeing, Liberty Mutual Insurance, Codeacademy, Box, McDonald's, Merck, McKinsey, Unity, DE Shaw, Tempus, Nike, DoorDash, Etsy, …

Will use TensorFlow for this class, but PyTorch is easy to learn if you already know deep learning fundamentals and TensorFlow.

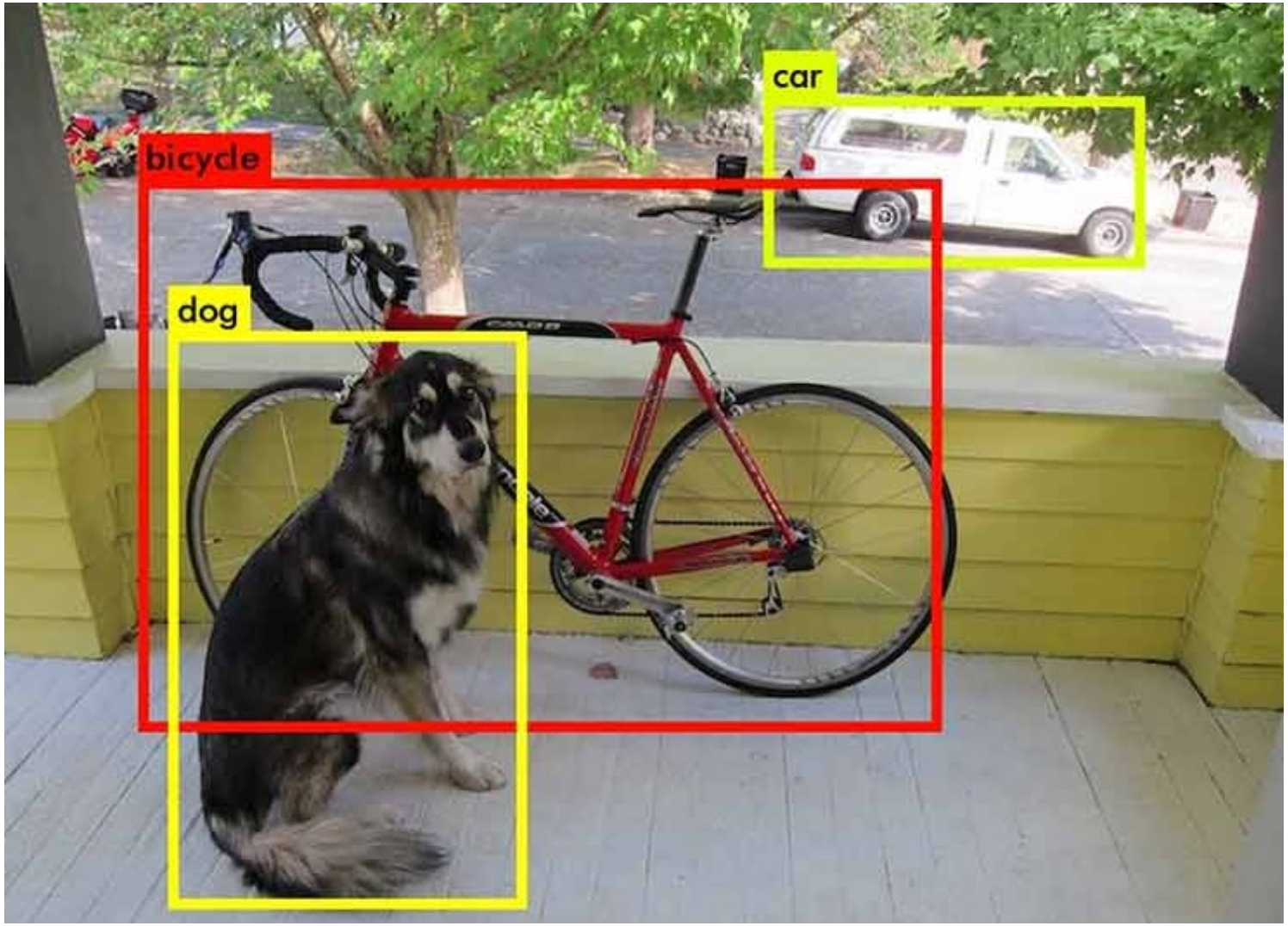
PyTorch

- Developed by Meta
- Most popular DL framework in research
- (Tiny sample of) examples of users: Meta (Facebook, Instagram), academic research labs, OpenAI, Microsoft, Toyota, Tesla, Lyft, Walmart, NVIDIA, Wells Fargo, Airbnb, Genentech, …

In practice, you can know either in terms of getting jobs (job requirements like "Development experience in deep learning frameworks such as PyTorch or TensorFlow")

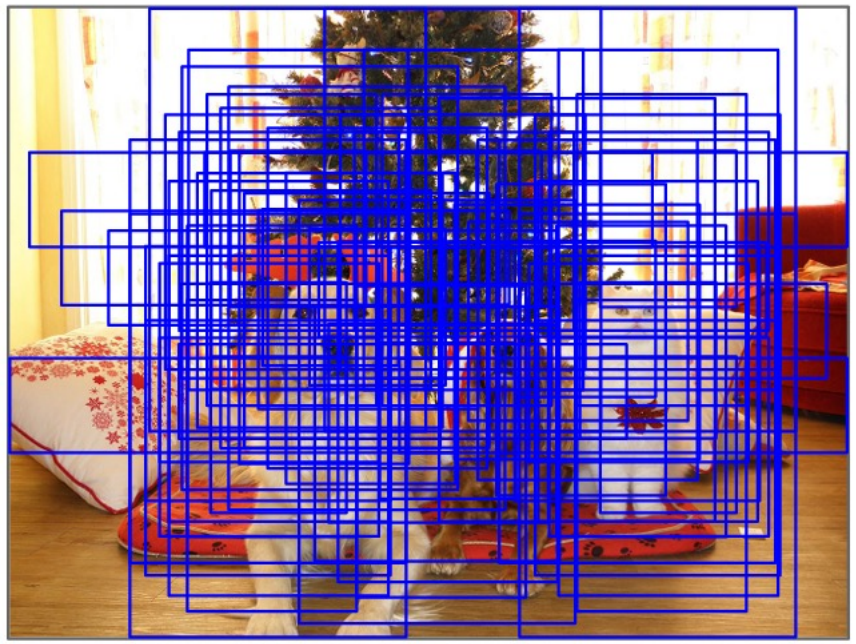
Object Detection

Goal of Object Detection



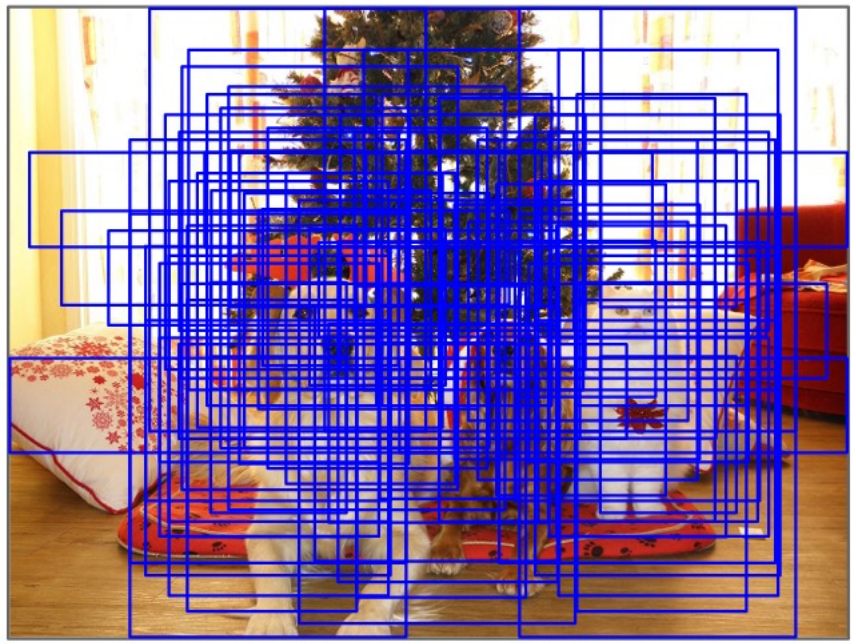
Idea #1: Brute Force

- Apply many crops of the image
- Run a CNN through each crop



Idea #1: Brute Force

- Apply many crops of the image
- Run a CNN through each crop

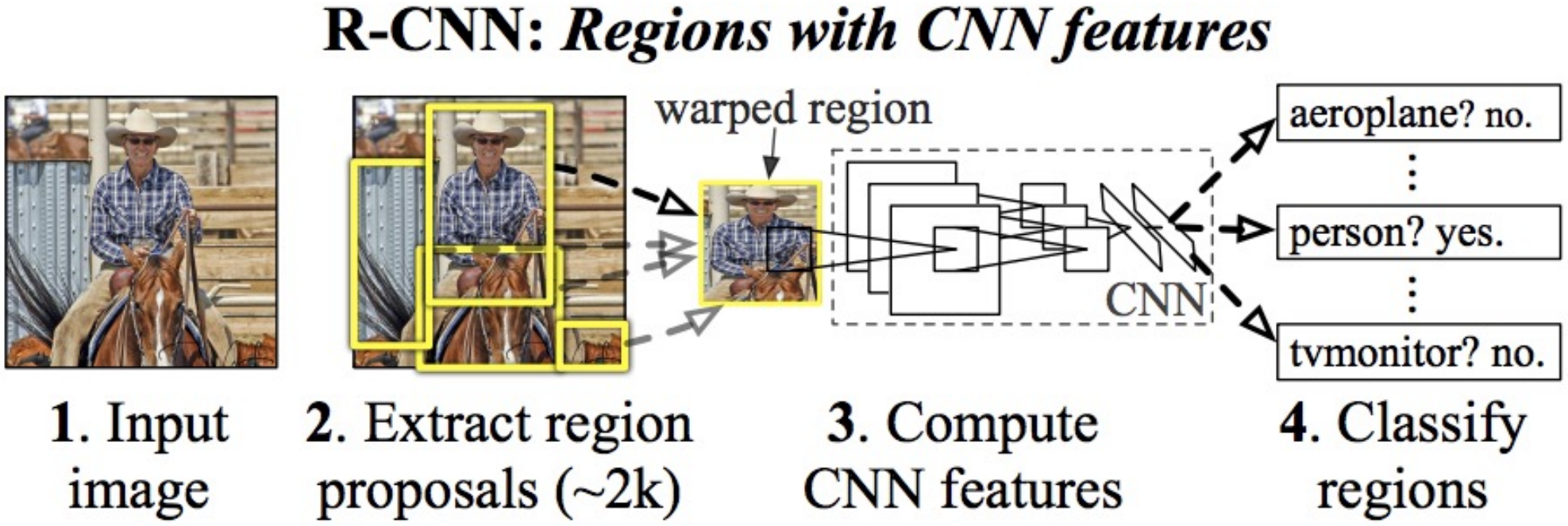


Issues:

- Need to apply CNN to huge number of locations, scales, and aspect ratios
- Very computationally expensive!

Idea #2: R-CNN (2014)

- Same as idea #1, except identify a small number of region proposals using a “selective search” algorithm (many possible algorithms)



Selective Search in R-CNNs

Algorithm 1: Hierarchical Grouping Algorithm

Input: (colour) image
Output: Set of object location hypotheses L

Obtain initial regions $R = \{r_1, \cdots, r_n\}$ using [13]
Initialise similarity set $S = \emptyset$
foreach *Neighbouring region pair* (r_i, r_j) **do**
  Calculate similarity $s(r_i, r_j)$
  $S = S \cup s(r_i, r_j)$

while $S \neq \emptyset$ **do**
  Get highest similarity $s(r_i, r_j) = \max(S)$
  Merge corresponding regions $r_t = r_i \cup r_j$
  Remove similarities regarding $r_i : S = S \setminus s(r_i, r_*)$
  Remove similarities regarding $r_j : S = S \setminus s(r_*, r_j)$
  Calculate similarity set S_t between r_t and its neighbours
  $S = S \cup S_t$
  $R = R \cup r_t$

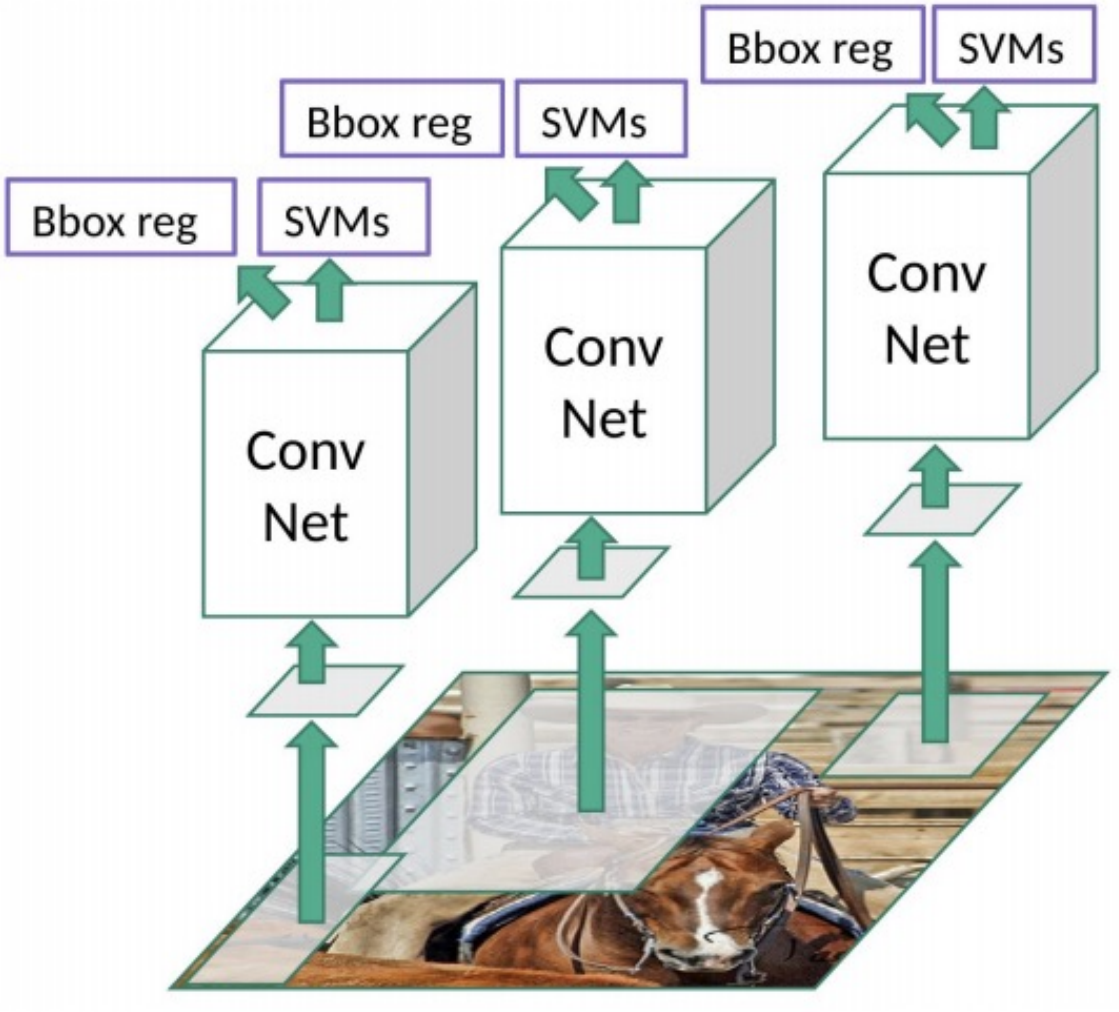
Extract object location boxes L from all regions in R

- Goal:
  - Capture all scales
  - Fast to compute
  - Diversification

- No need to memorize this algorithm; just know that there are <u>many</u> unsupervised ways of producing regions from an image. This is just one of them.

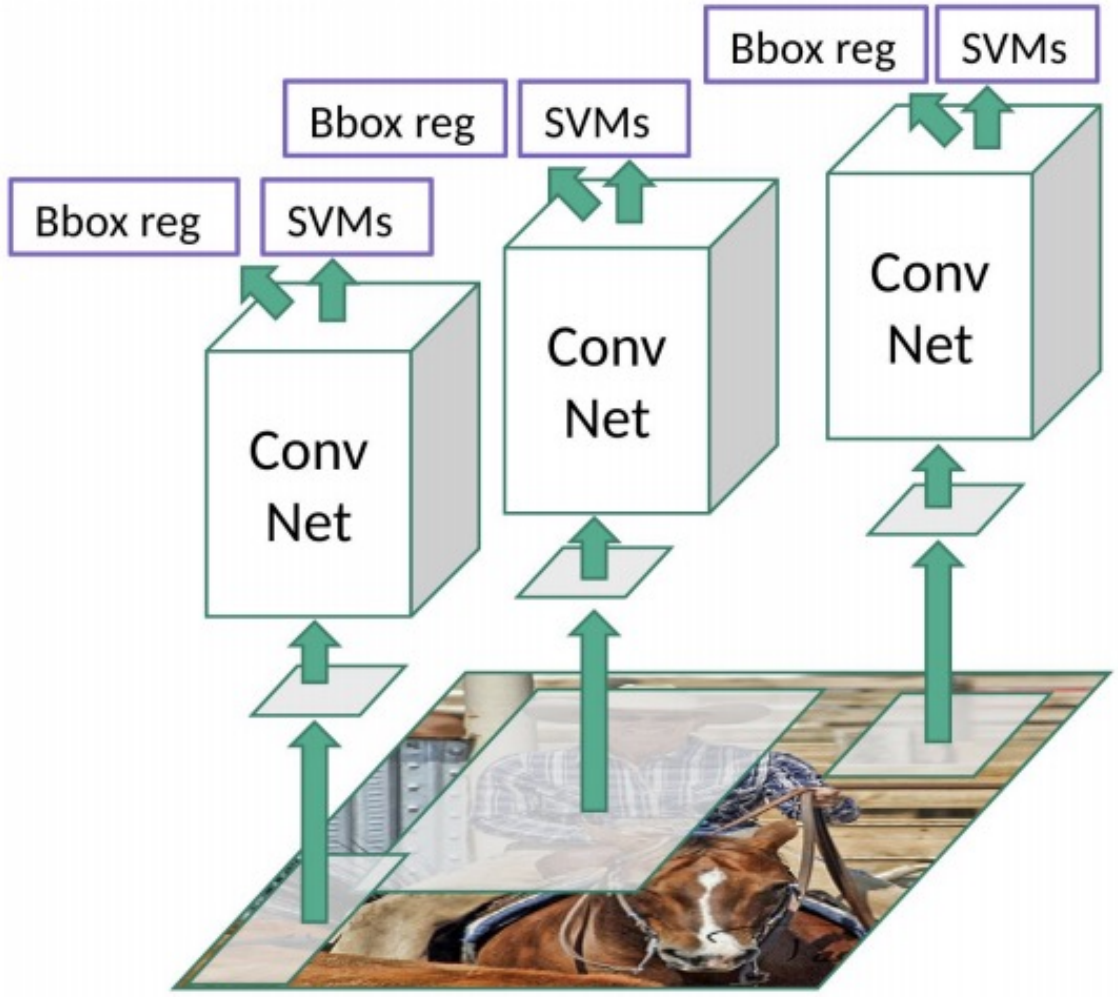
R-CNN continued

- Convolutional layers are used as a feature extractor
- An SVM is used to perform the final classification



R-CNN continued

- Convolutional layers are used as a feature extractor
- An SVM is used to perform the final classification

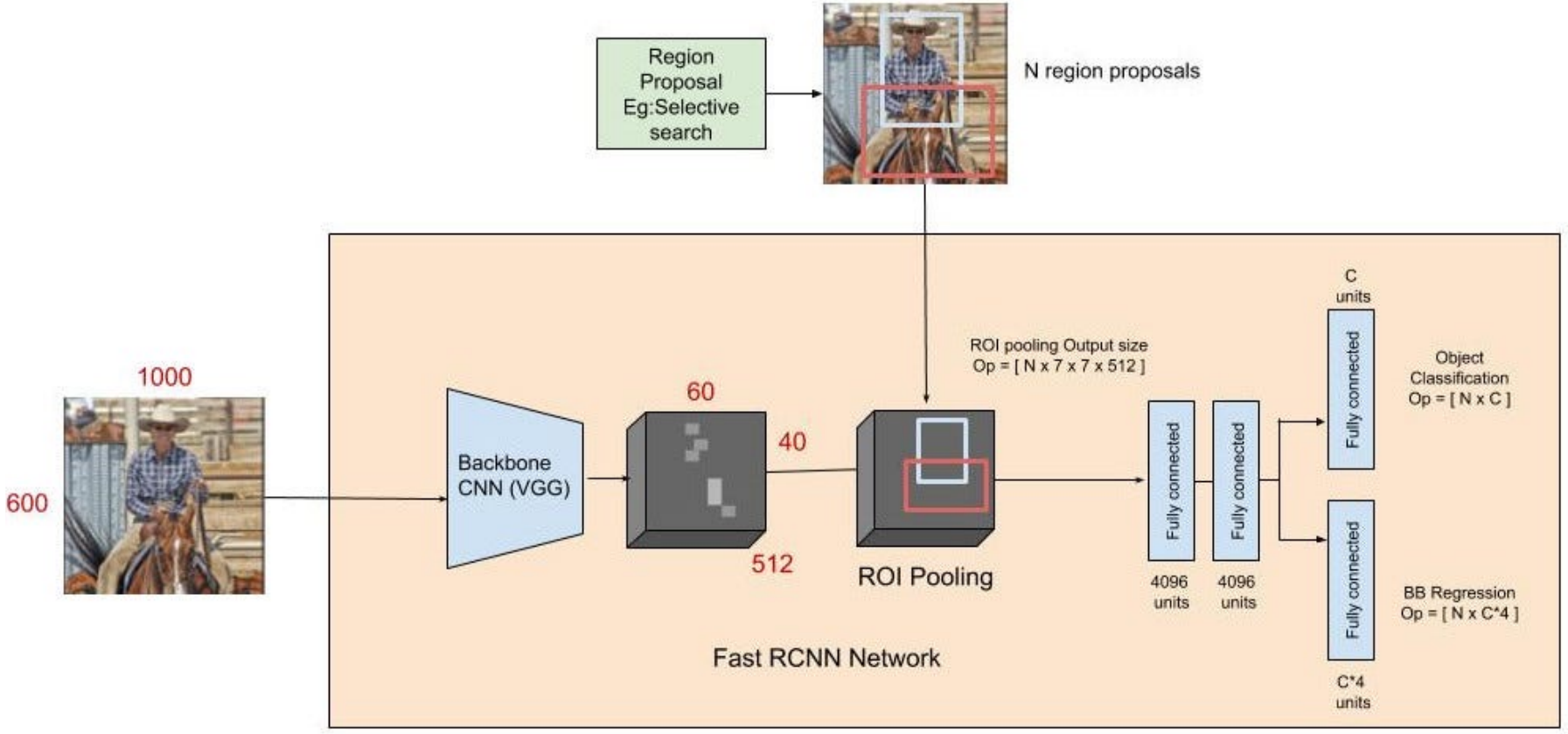


Issues:

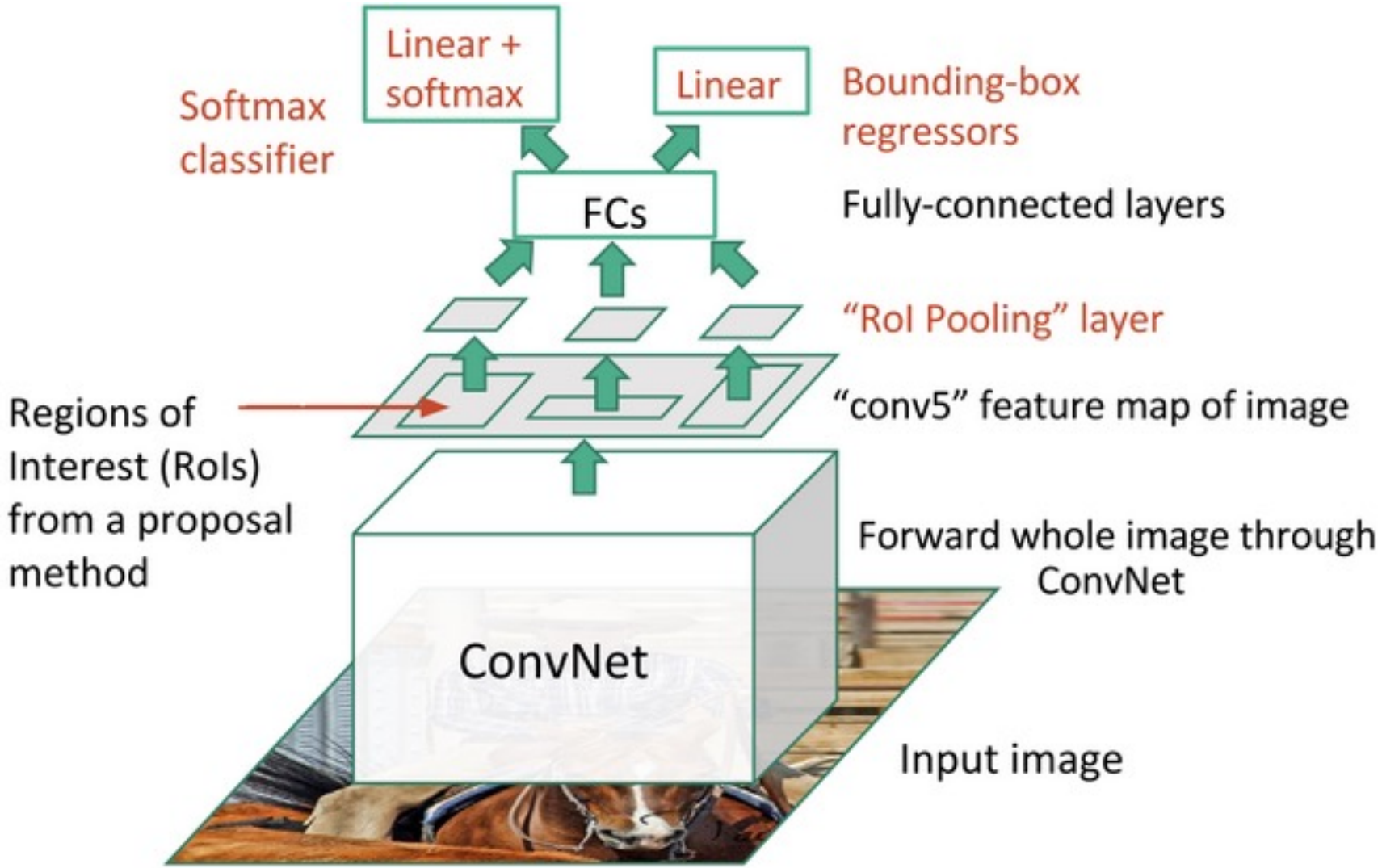
- Very slow: need to do ~2000 independent forward passes for each image
  - Takes about 47 seconds to run per image
- Selective Search is a fixed algorithm: no learning is involved. Can we do better?

Idea #3: Fast R-CNN (2015)

- Use a fixed region proposal method like in R-CNN, but…
- Feed whole image through a single CNN to make predictions per proposal, minimizing time from running separate CNN per region



Region of Interest (RoI) Pooling Layers



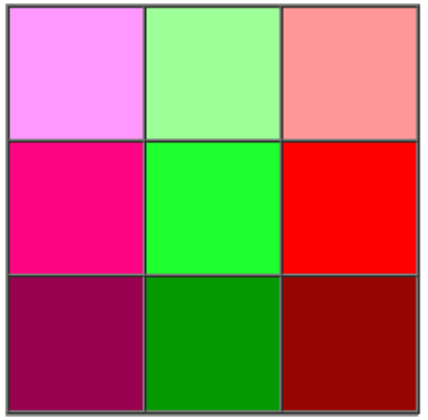
RoI Pooling

- Fixes issue of CNNs requiring a fixed input dimension

4x6 RoI

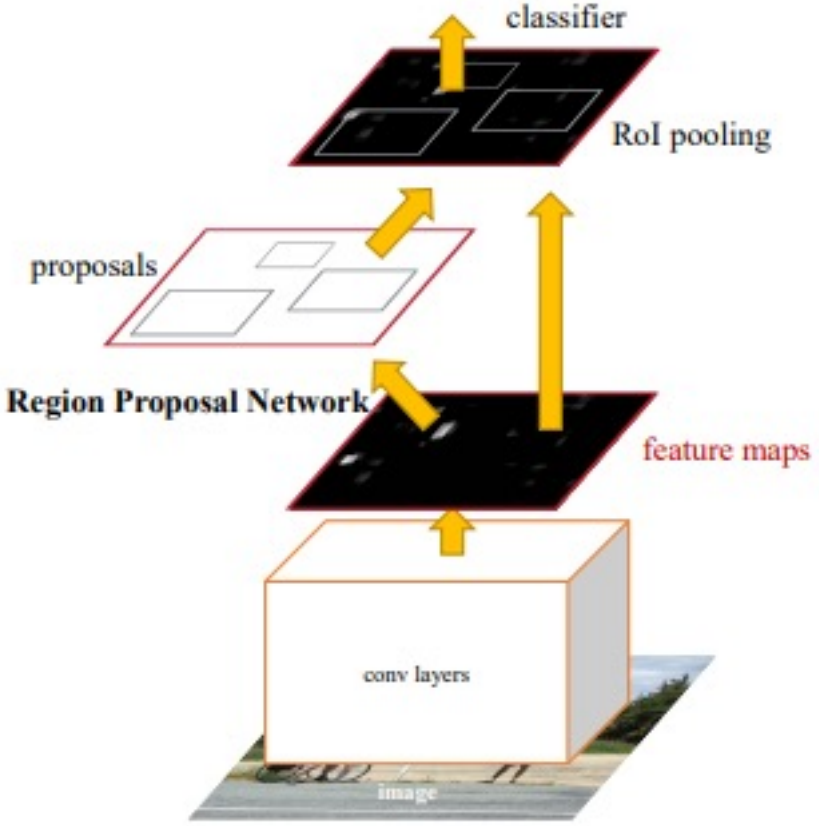
0.1	0.2	0.3	0.4	0.5	0.6
1	0.7	0.2	0.6	0.1	0.9
0.9	0.8	0.7	0.3	0.5	0.2

3x3 RoI Pooling



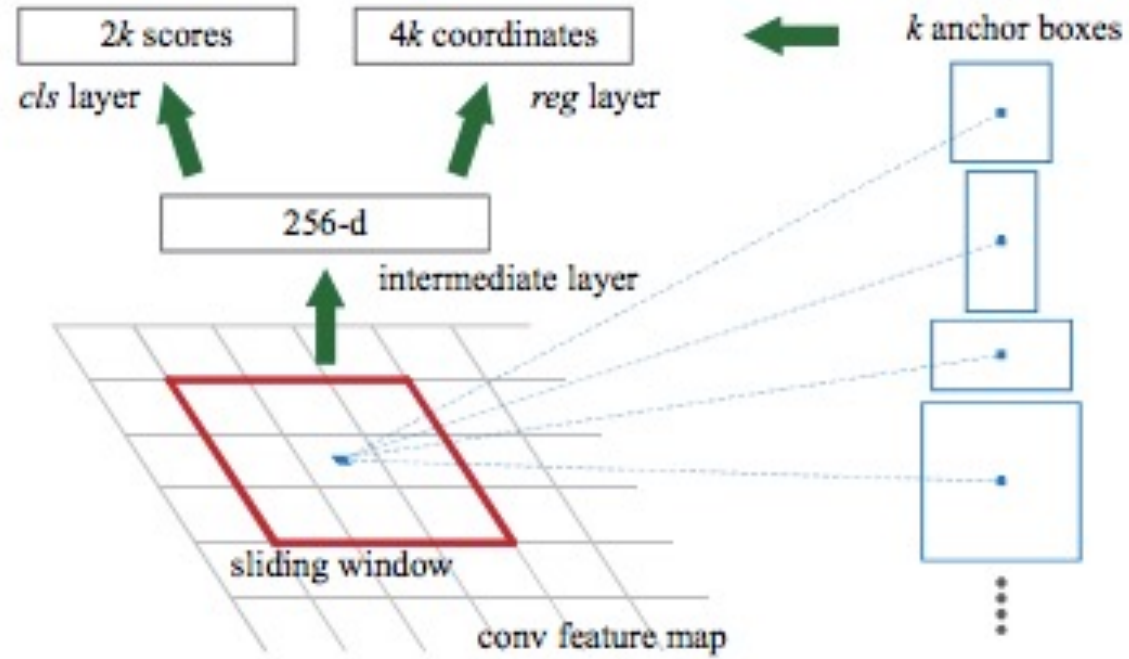
Idea #4: Faster R-CNN (2015)

- Same as Fast R-CNN, except a CNN does the ROI proposals
- This CNN is called a “Region Proposal Network”
- Notice a trend: increasingly use ML

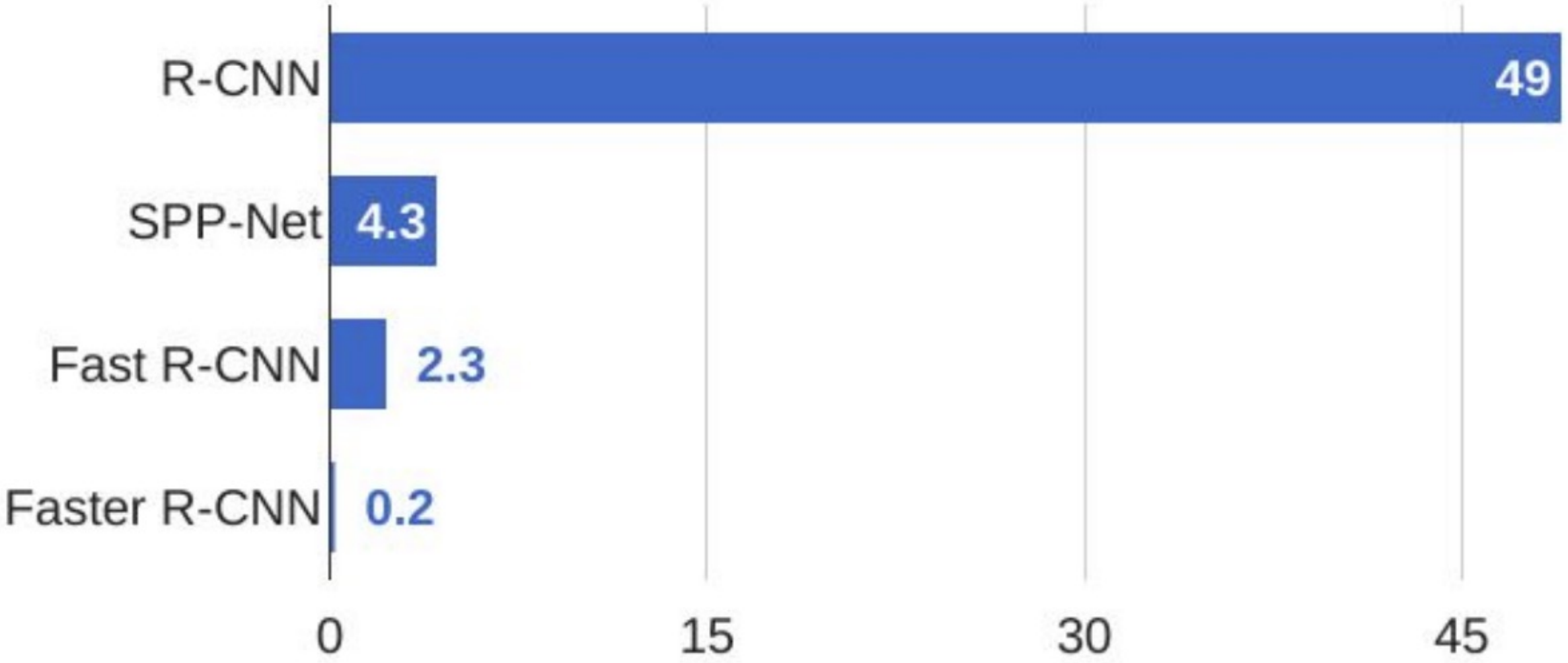


Region Proposal Network

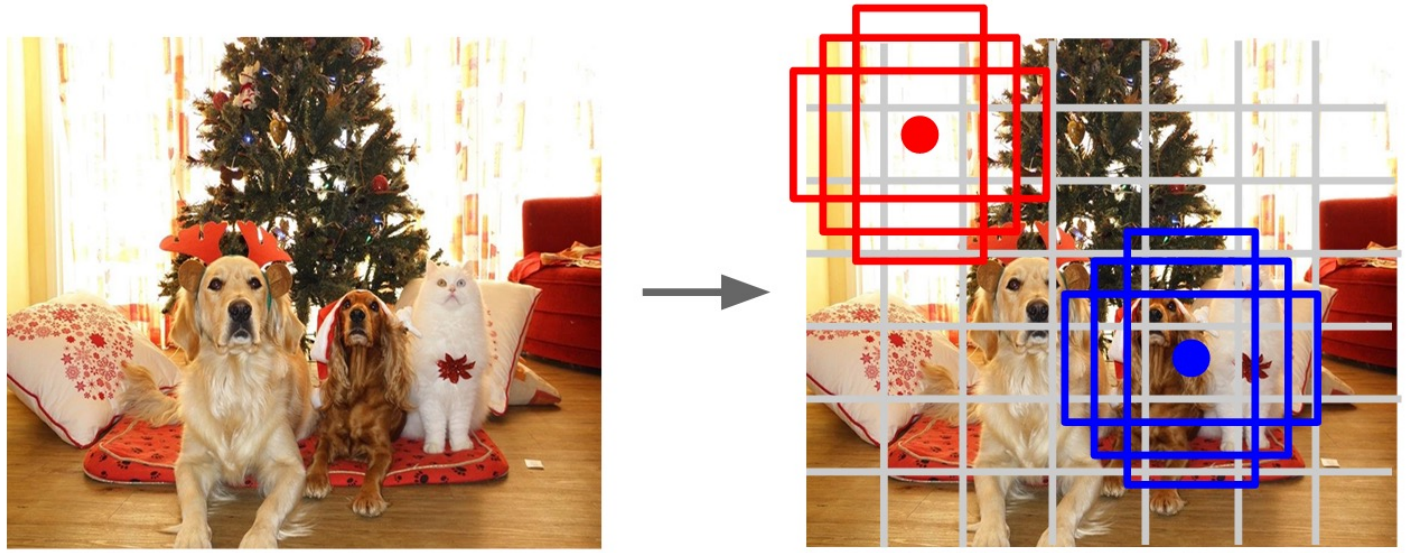
- Contains a classifier and a regressor
- Center of anchor boxes is center of sliding window
- Classifier determines the probability of a proposal having the target object
- Regressor regresses the coordinates of the proposals
- Details beyond scope of this class



Comparison of Detection Speeds (seconds)



Idea #5: You Only Look Once (YOLO) (2015-2023)



Divide image into a grid

Use a set of base boxes per grid

Within each grid cell:

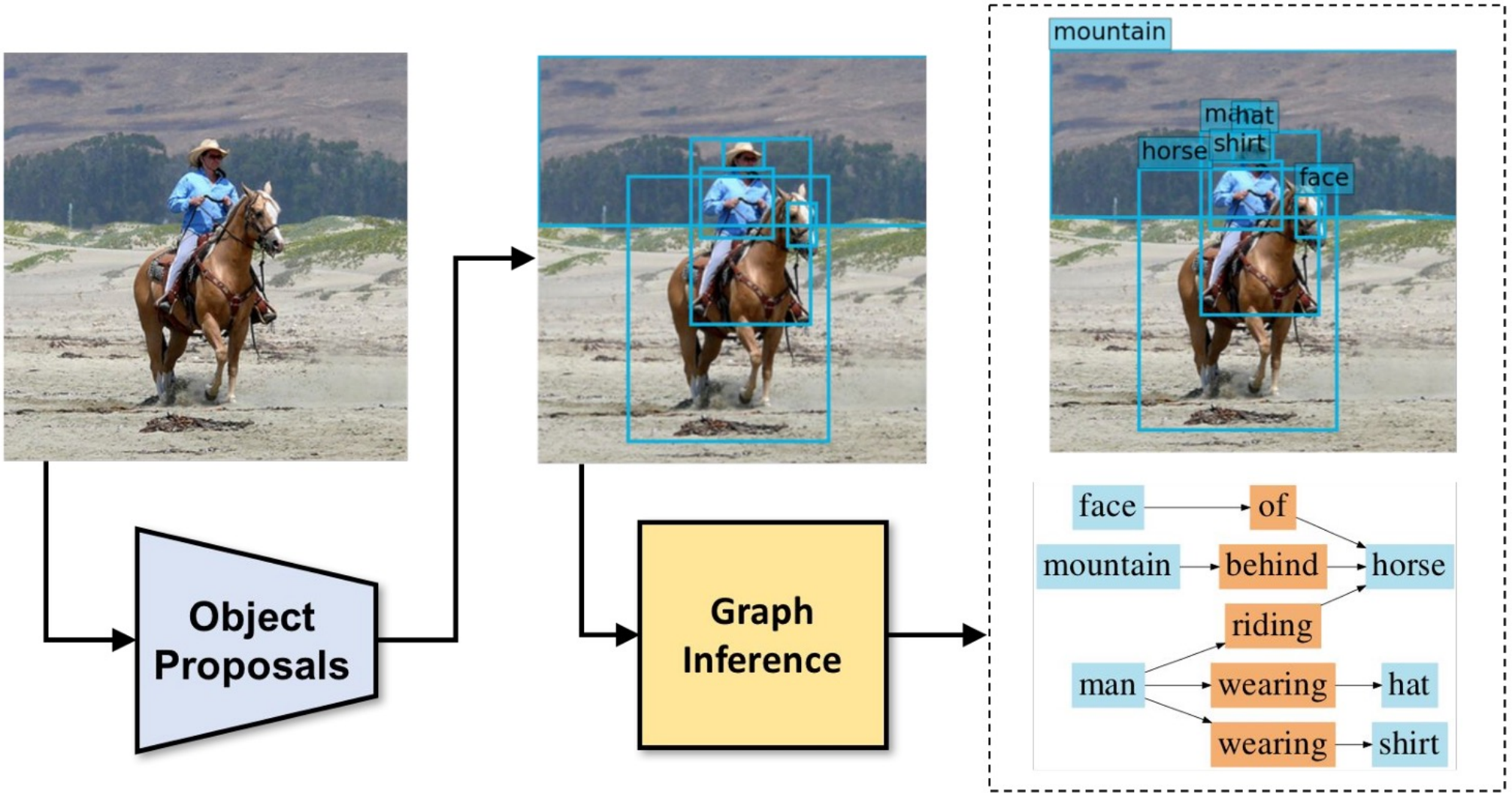
- Regress from each of the B base boxes to a final box with 5 numbers: box coordinates and confidence scores
- Predict scores for each of C classes (including background as a class)
- Looks a lot like a Region Proposal Network, but category-specific

Modifications to YOLO Over Time

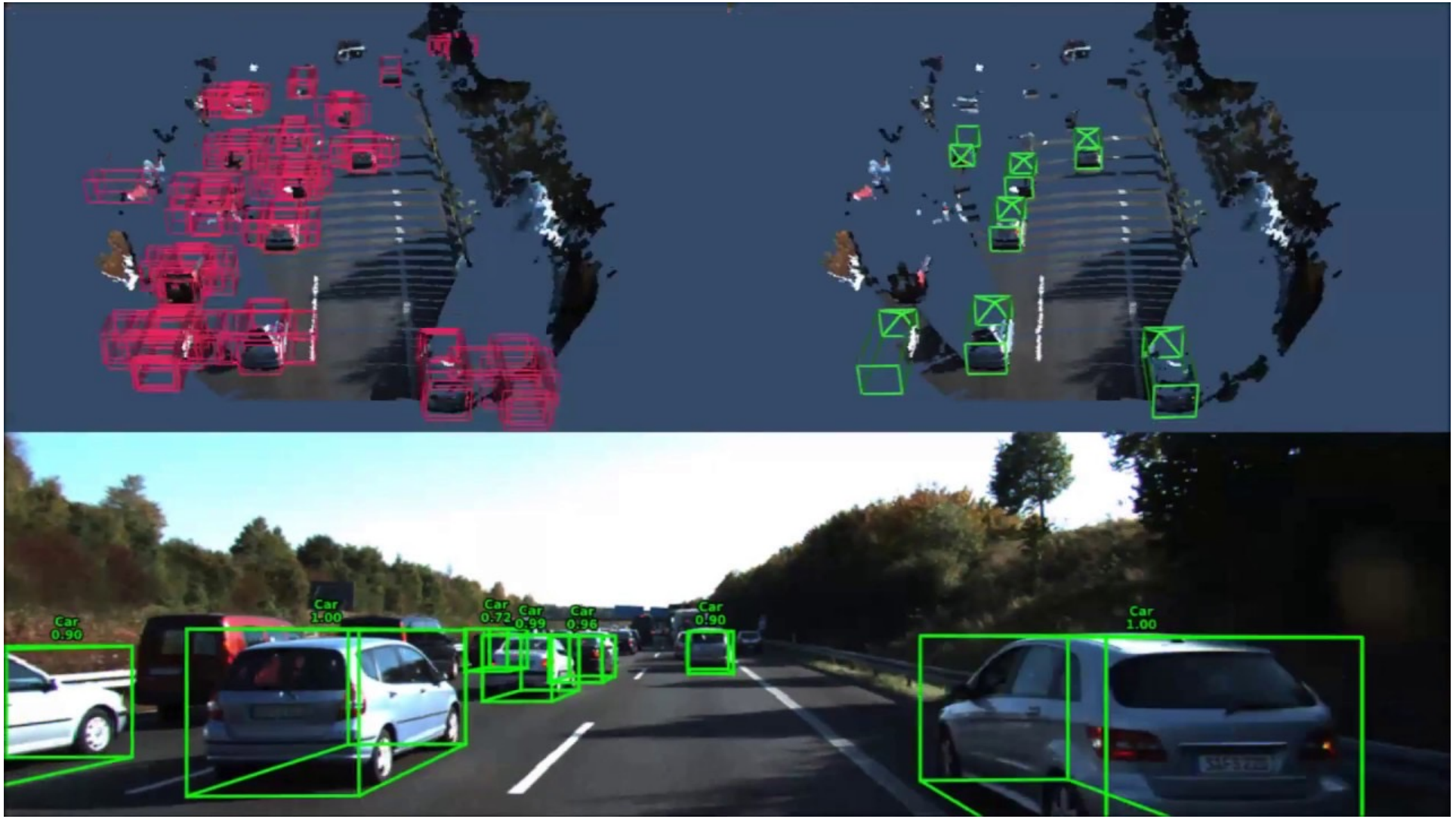
- YOLOv3 model, introduced by Redmon et al. in 2018
- YOLOv4 model, released by Bochkovskiy et al. in 2020,
- YOLOv4-tiny model, research published in 2021
- YOLOR (You Only Learn One Representation) model, published in 2021
- YOLOX model, published in 2021
- NanoDet-Plus model, published in 2021
- PP-YOLOE, an industrial object detector, published in 2022
- YOLOv5 model v6.1 published by Ultralytics in 2022
- YOLOv7, published in 2022

Related Deep Computer Vision Tasks

Knowledge Scene Graph Prediction



3D Object Detection



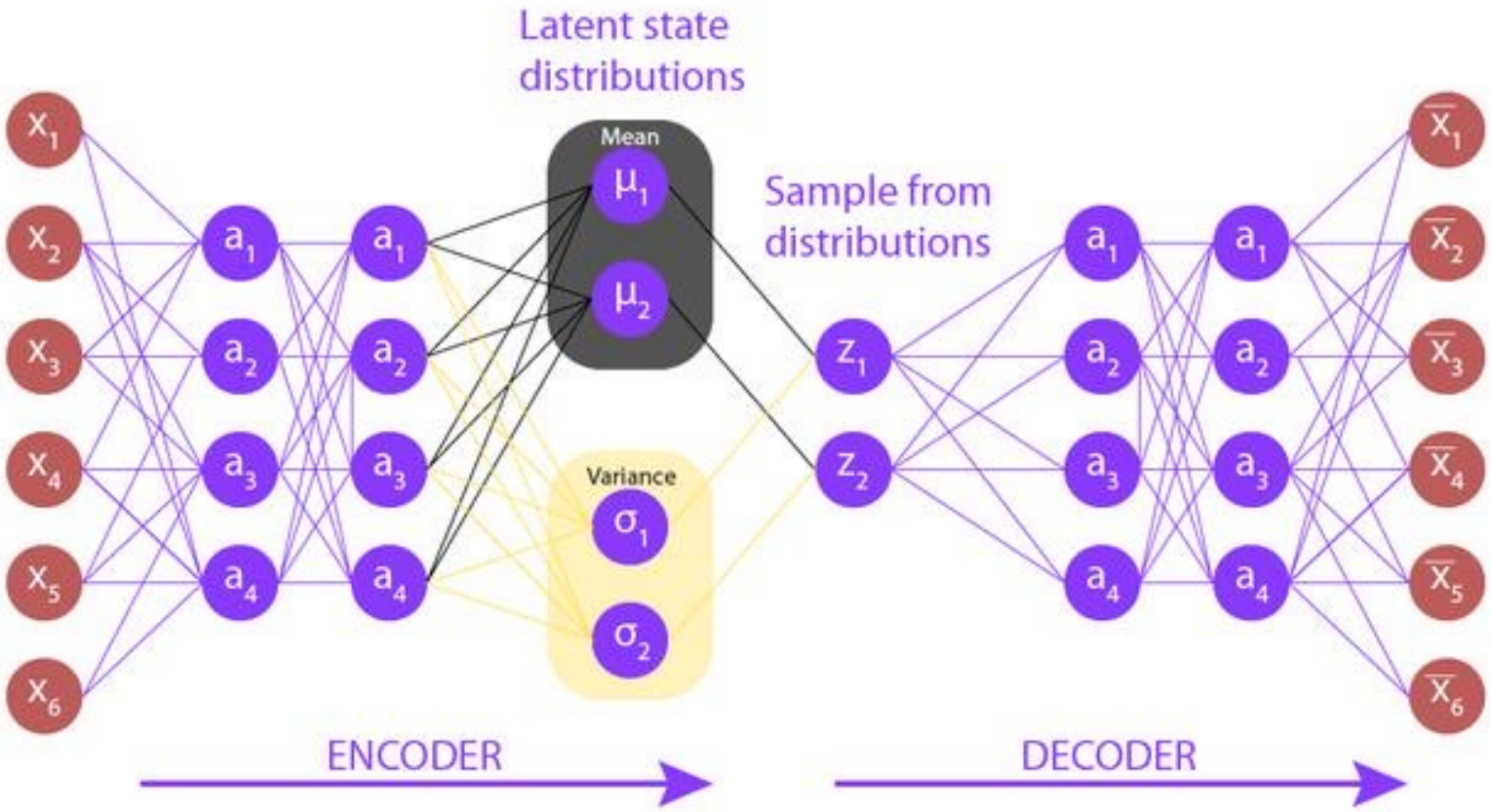
Pose Estimation

Object Tracking

Variational Autoencoders (VAEs)
(first proposed 2013,
good examples starting around 2017)

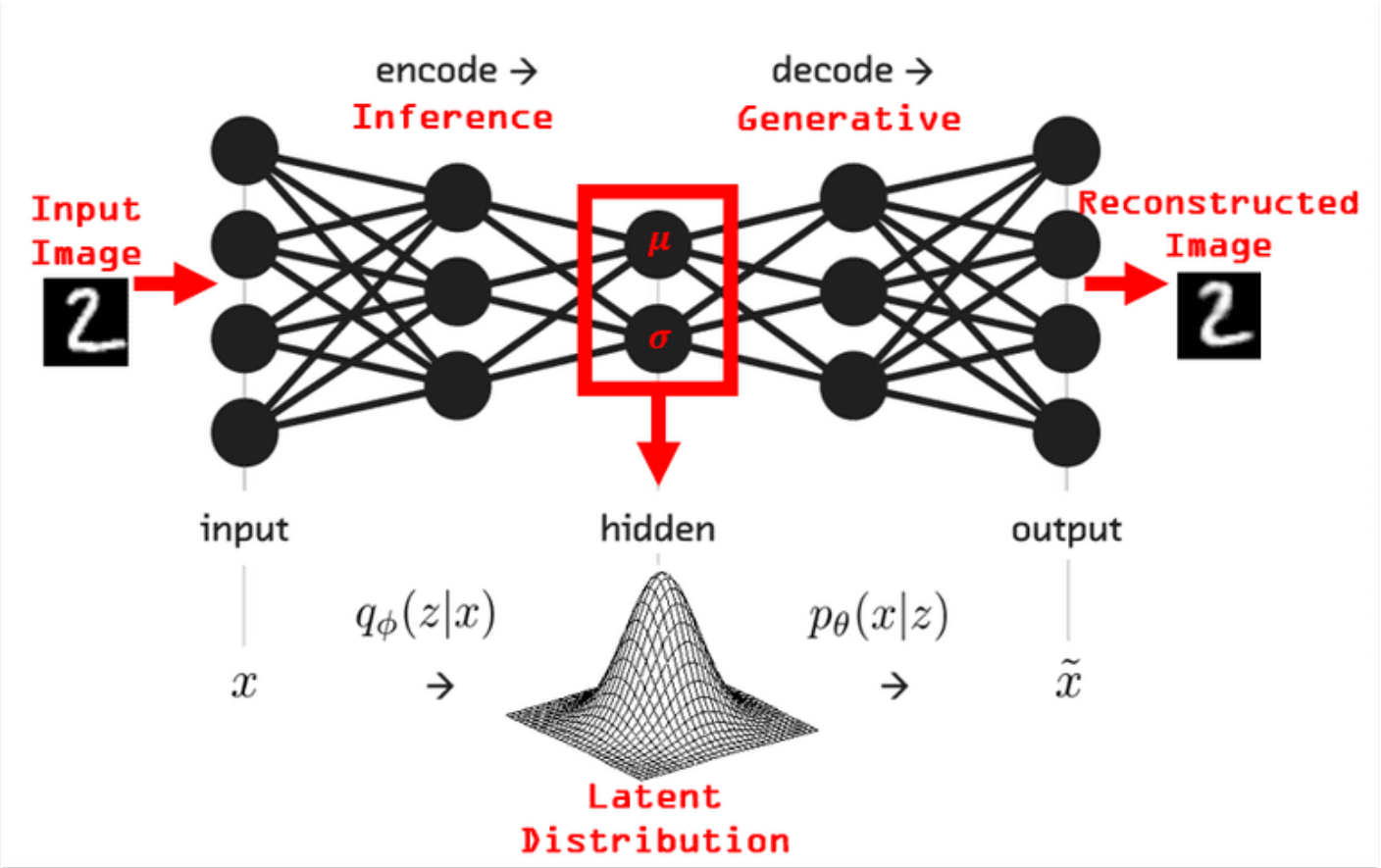
Variational Autoencoder (VAE)

Key idea: sample from the “latent space” generate data.



Variational Autoencoder (VAE)

Key idea: sample from the “latent space” generate data.



VAE Loss Function

$$|| x - d(z) ||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

VAE Loss Function

$$|| x - d(z) ||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Reconstruction Error term:

Same as a regular autoencoder

VAE Loss Function

$$|| x - d(z) ||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Reconstruction Error term:

Same as a regular autoencoder

"What is the distance between x and the resulting of running the decoder through the latent space z?"

VAE Loss Function

$$|| x - d(z) ||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Reconstruction Error term:

Same as a regular autoencoder

"What is the distance between x and the resulting of running the decoder through the latent space z?"

KL Divergence term:

If latent space distribution differs from a standard normal distribution, impose a penalty (otherwise, may learn to make variance 0, thus functioning as a plain autoencoder)

VAE Loss Function

(KL Divergence is a popular distance metric between probability distributions)

$$|| x - d(z) ||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Reconstruction Error term:

Same as a regular autoencoder

"What is the distance between x and the resulting of running the decoder through the latent space z?"

KL Divergence term:

If latent space distribution differs from a standard normal distribution, impose a penalty (otherwise, may learn to make variance 0, thus functioning as a plain autoencoder)

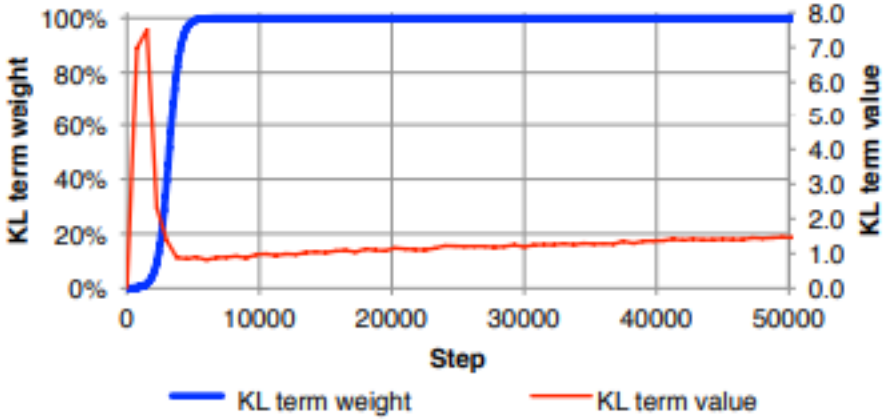
VAE Loss Function

In practice, use a hyperparameter to control how much to weight one term vs another (similar to λ in L1/L2 regularization).

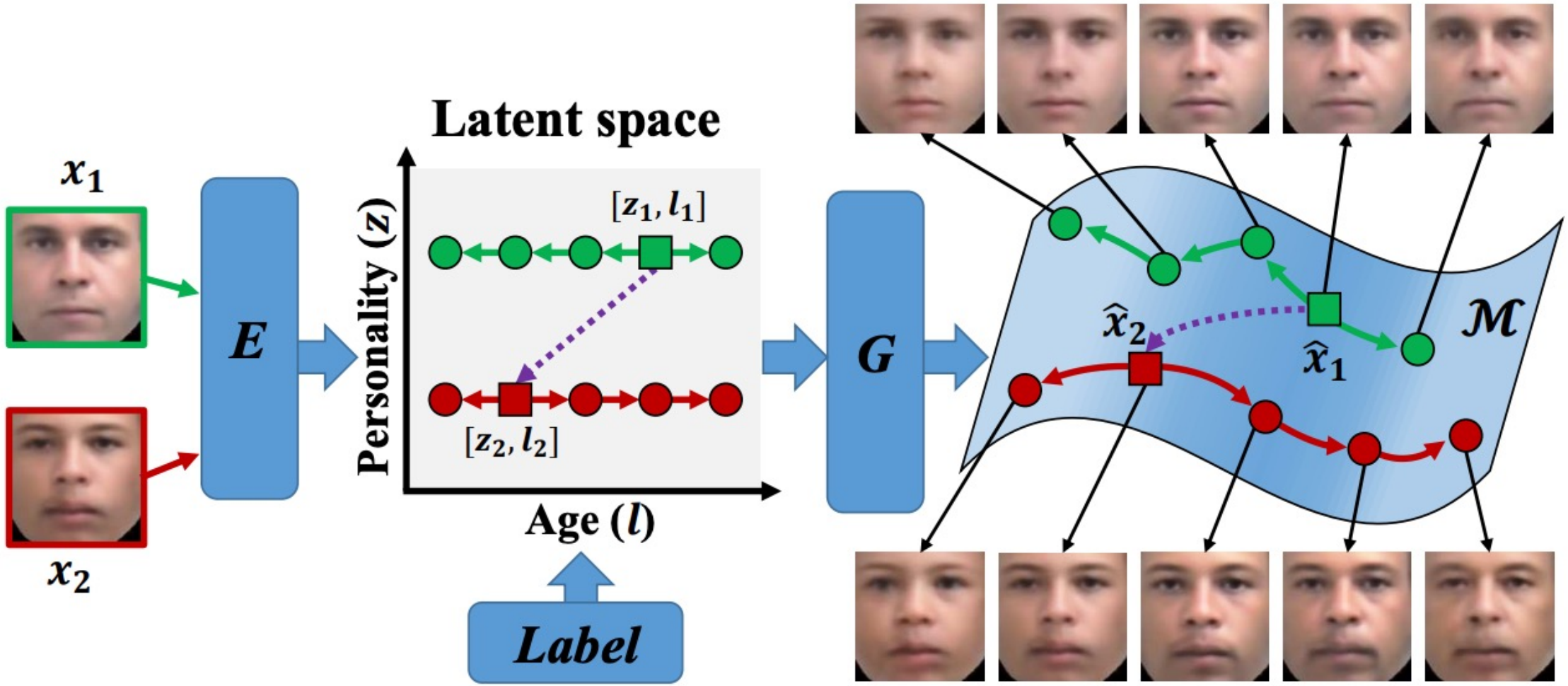
$$|| x - d(z) ||^2 + \lambda KL[N(\mu_x, \sigma_x), N(0, I)]$$

In practice, this hyperparameter is made larger with each consecutive epoch ("KL Annealing").

This prevents just optimizing the KL term at first without learning a useful representation, which happens in practice.



Applications of VAEs:
Examples of Sampling from the Latent Space



Zhang, Zhifei, Yang Song, and Hairong Qi. "Age progression/regression by conditional adversarial autoencoder." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

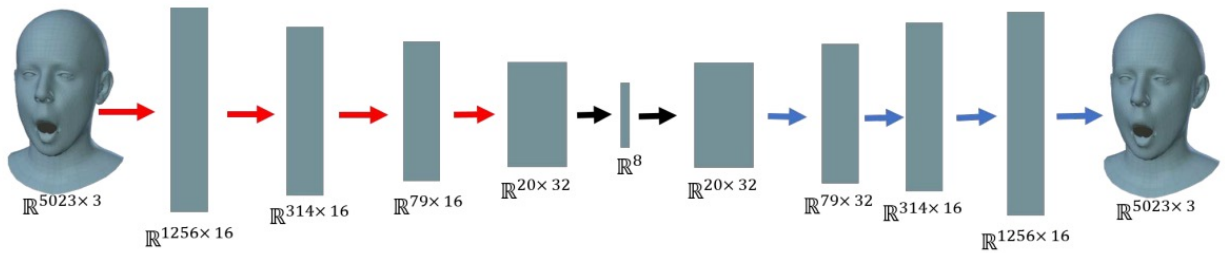


Fig. 2. Convolutional Mesh Autoencoder: The red and blue arrows indicate down-sampling and up-sampling layers respectively.

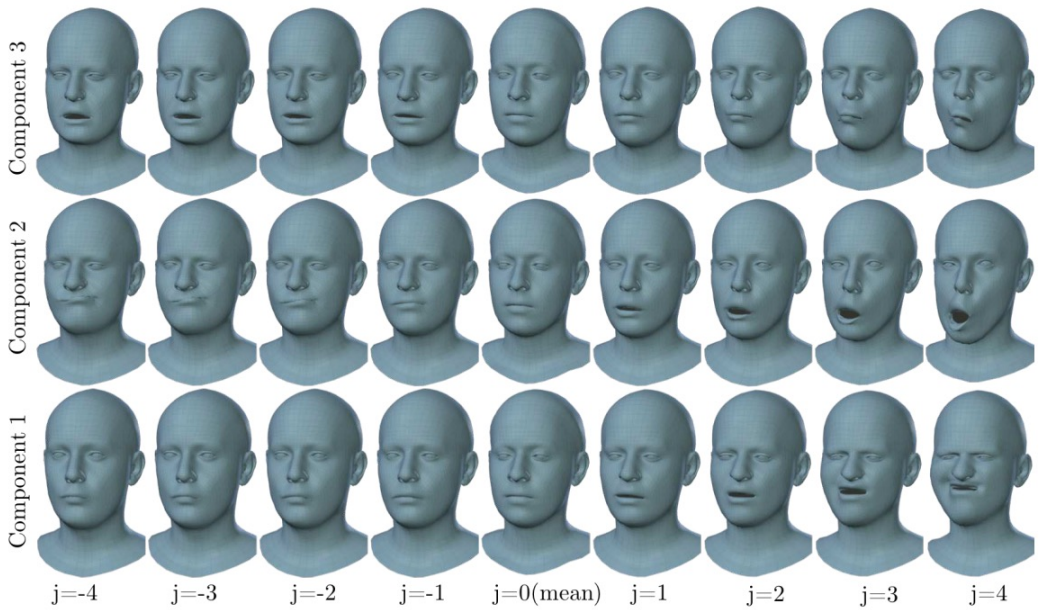


Fig. 3. Sampling from the latent space of the mesh autoencoder around the mean face $j = 0$ along 3 different components.

Ranjan, Anurag, et al. "Generating 3D faces using convolutional mesh autoencoders." *Proceedings of the European conference on computer vision (ECCV)*. 2018.

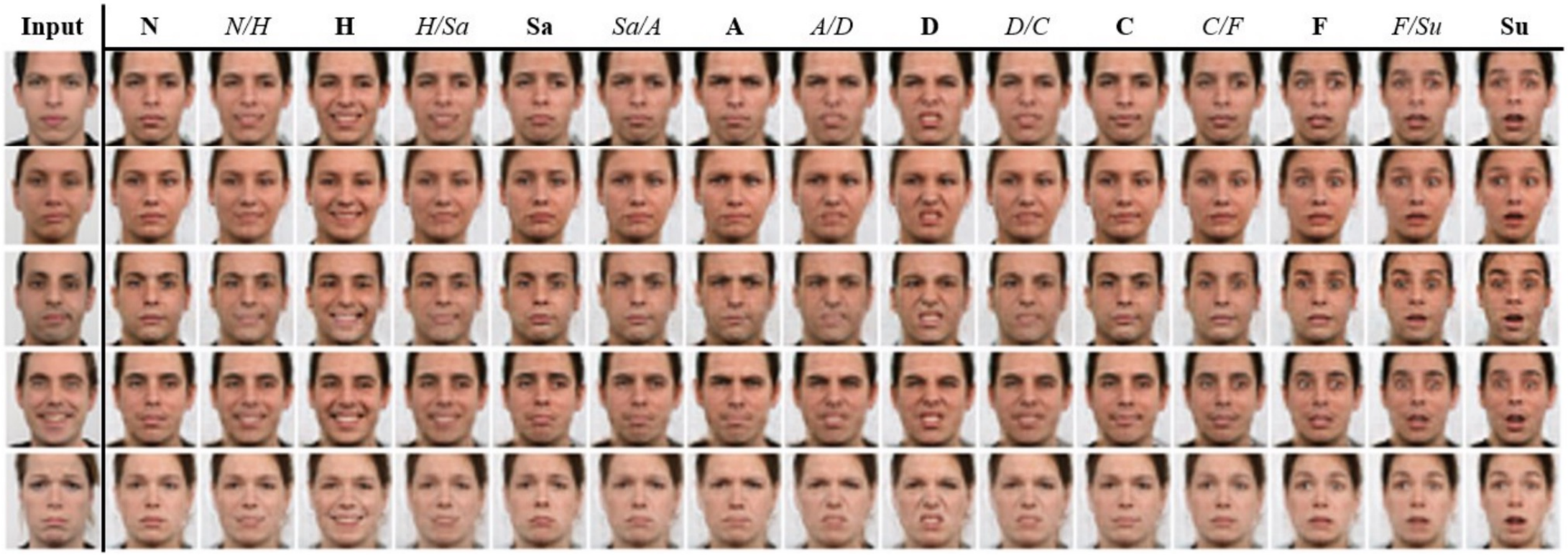
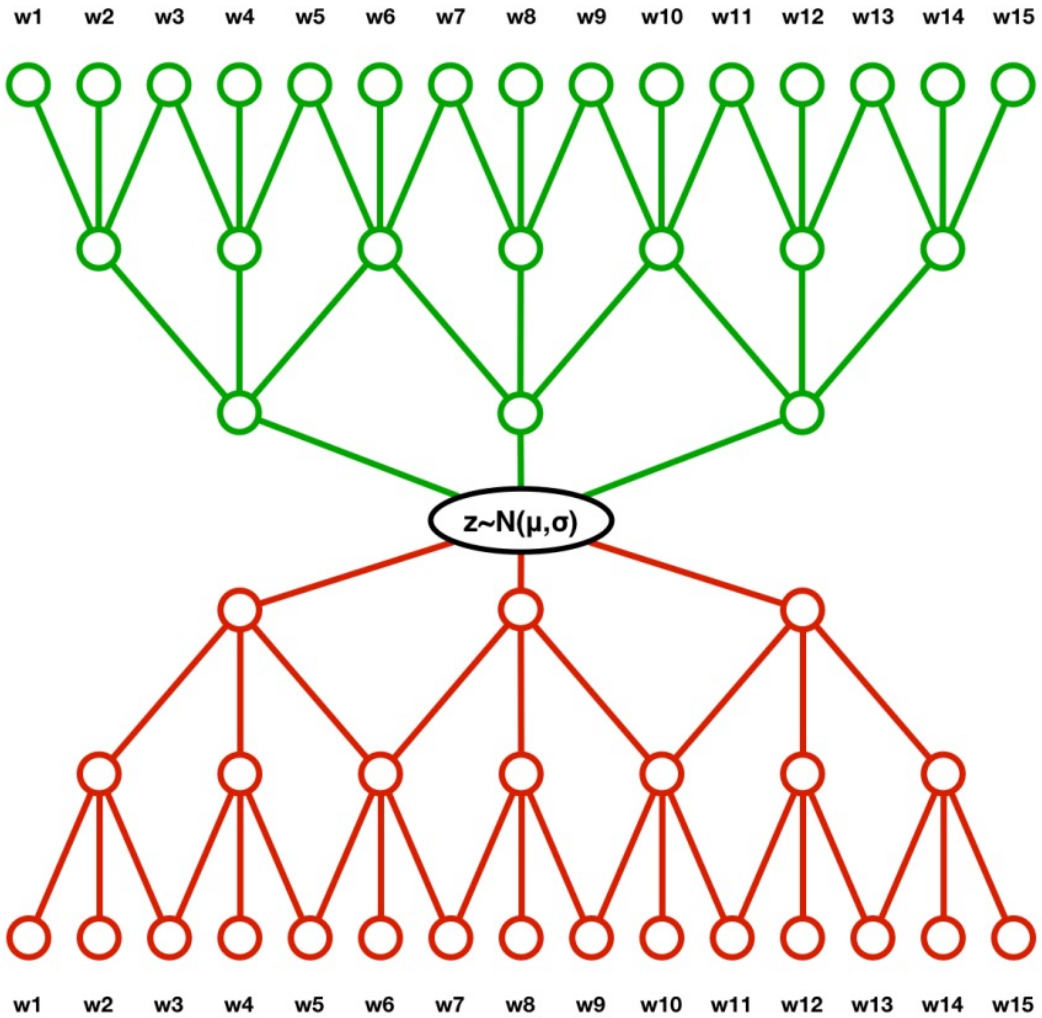
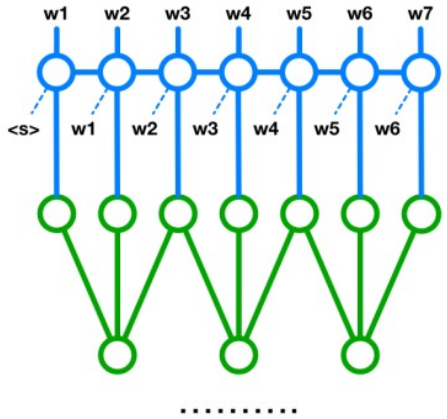


Fig.5. Synthesis results of all the emotion classes and their interpolation using the CDAAE (N2) network. The interpolation is obtained by setting the label values of the two emotions to 0.5. (**N**:neutral, **H**:happiness, **Sa**:sadness, **A**:anger, **D**:disgust, **C**:contempt, **F**:fear, **Su**:surprise)

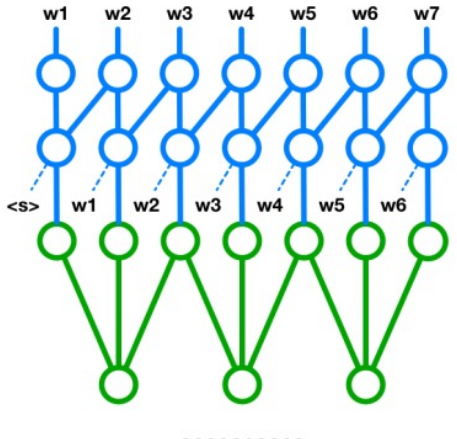
Zhou, Yuqian, and Bertram Emil Shi. "Photorealistic facial expression synthesis by the conditional difference adversarial autoencoder." *2017 seventh international conference on affective computing and intelligent interaction (ACII)*. IEEE, 2017.



(a) Fully feed-forward component of our VAE model



(b) Hybrid model with LSTM decoder



(c) Hybrid model with ByteNet decoder

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth. "A hybrid convolutional variational autoencoder for text generation." *arXiv preprint arXiv:1702.02390* (2017).

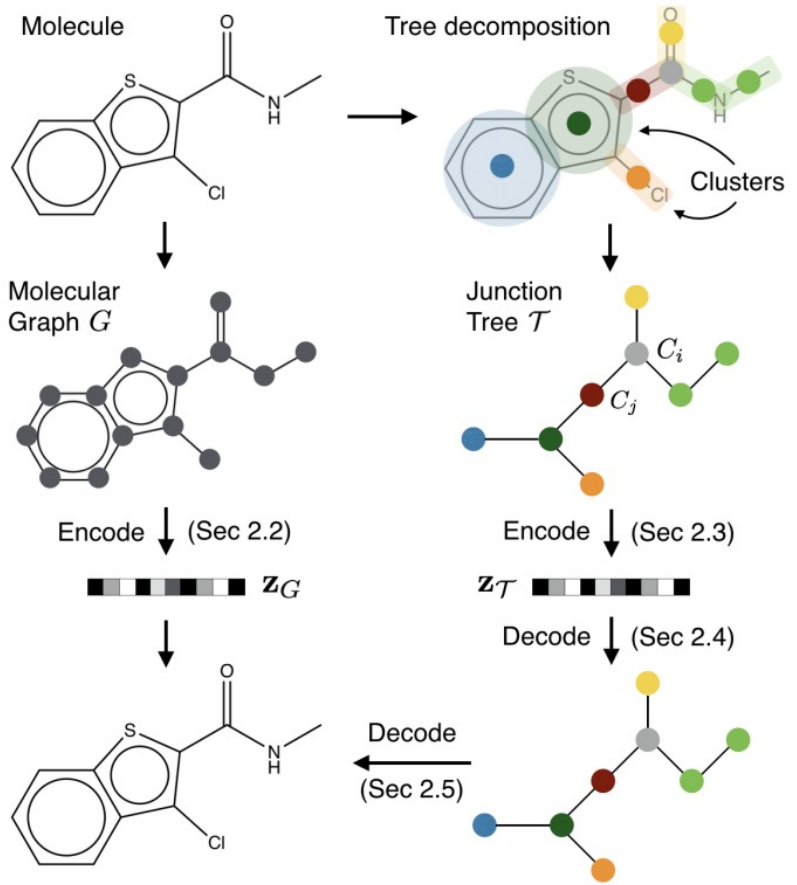
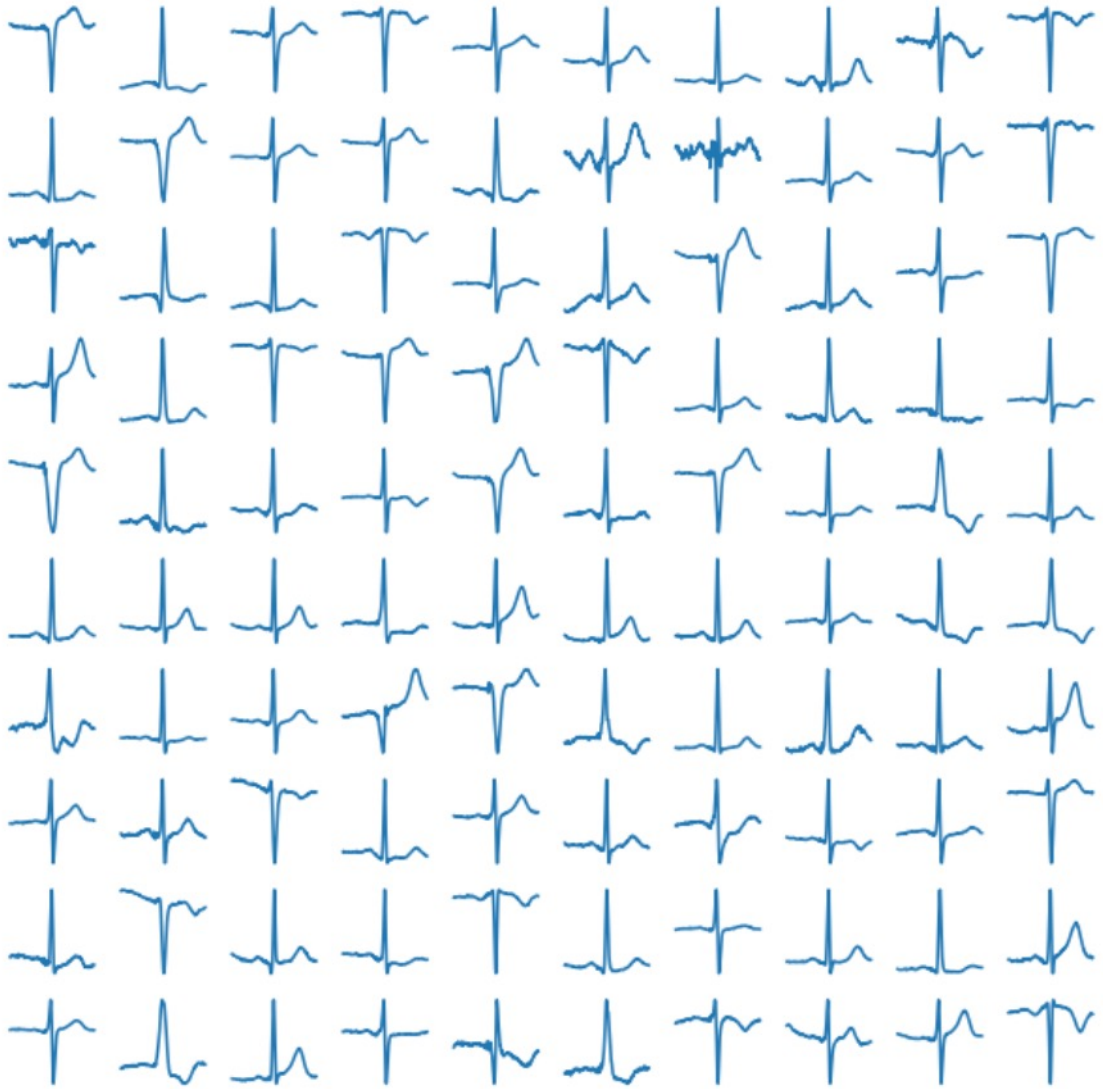


Figure 3. Overview of our method: A molecular graph G is first decomposed into its junction tree $\mathcal{T}_G$, where each colored node in the tree represents a substructure in the molecule. We then encode both the tree and graph into their latent embeddings $\mathbf{z}_\mathcal{T}$ and $\mathbf{z}_G$. To decode the molecule, we first reconstruct junction tree from $\mathbf{z}_\mathcal{T}$, and then assemble nodes in the tree back to the original molecule.

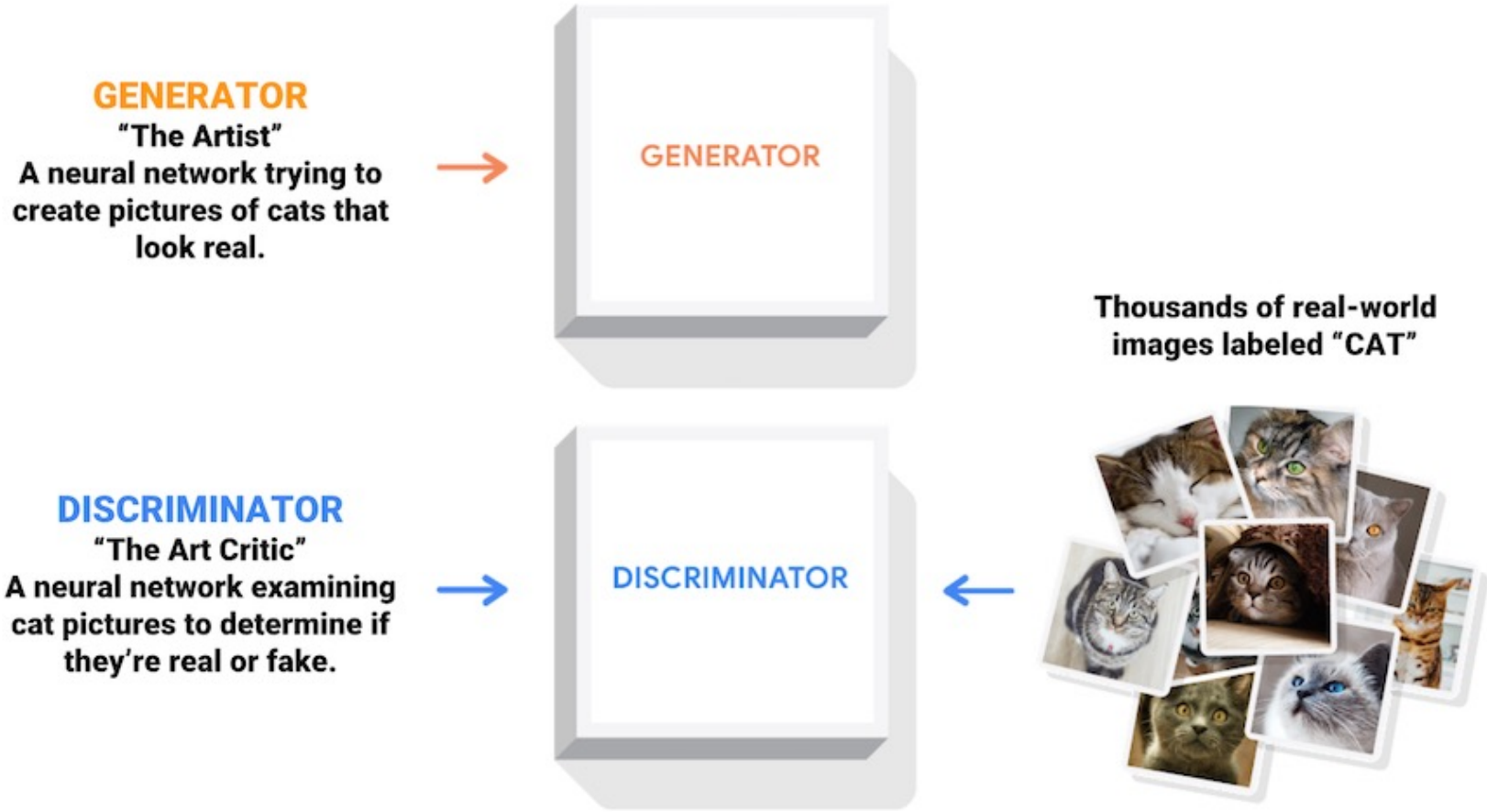
Jin, Wengong, Regina Barzilay, and Tommi Jaakkola. "Junction tree variational autoencoder for molecular graph generation." *International conference on machine learning*. PMLR, 2018.



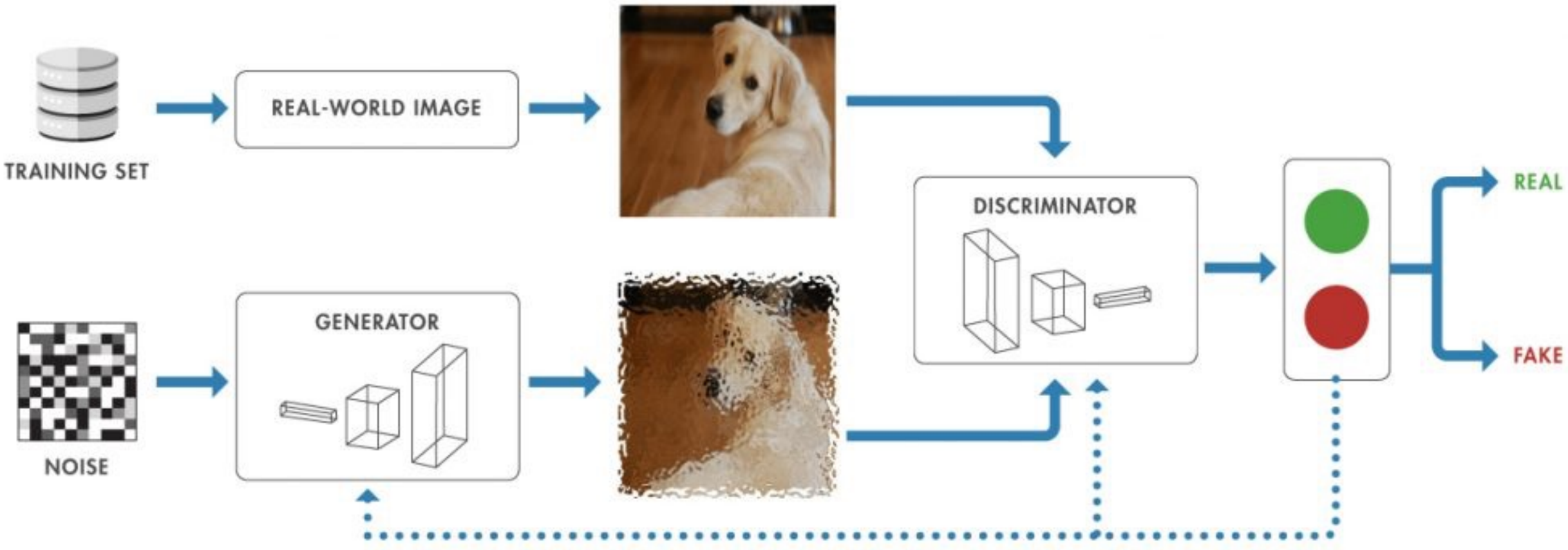
Kuznetsov, V. V., V. A. Moskalenko, and N. Yu Zolotykh. "Electrocardiogram generation and feature extraction using a variational autoencoder." *arXiv preprint arXiv:2002.00254* (2020).

Generative Adversarial Networks (GANs)
(first proposed 2014,
good examples starting around 2017)

Generative Adversarial Network (GAN)



Generative Adversarial Network (GAN)

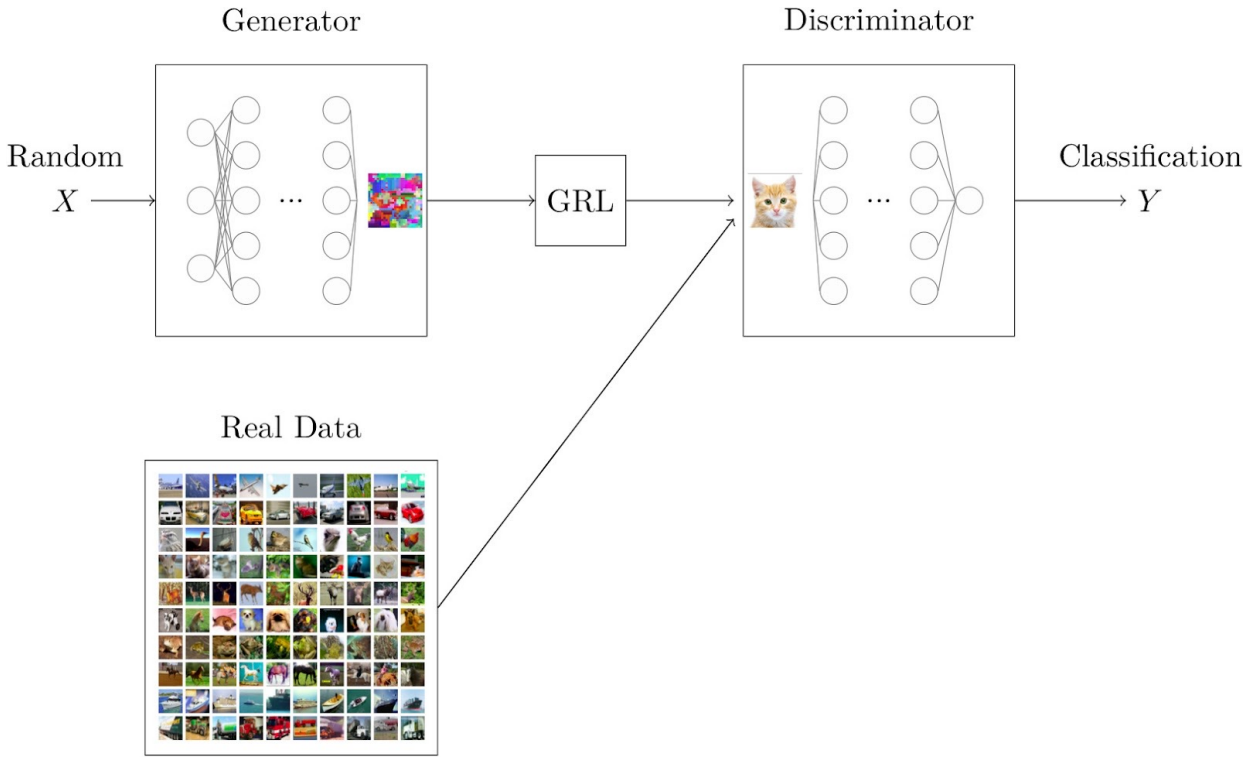


GAN Training Process

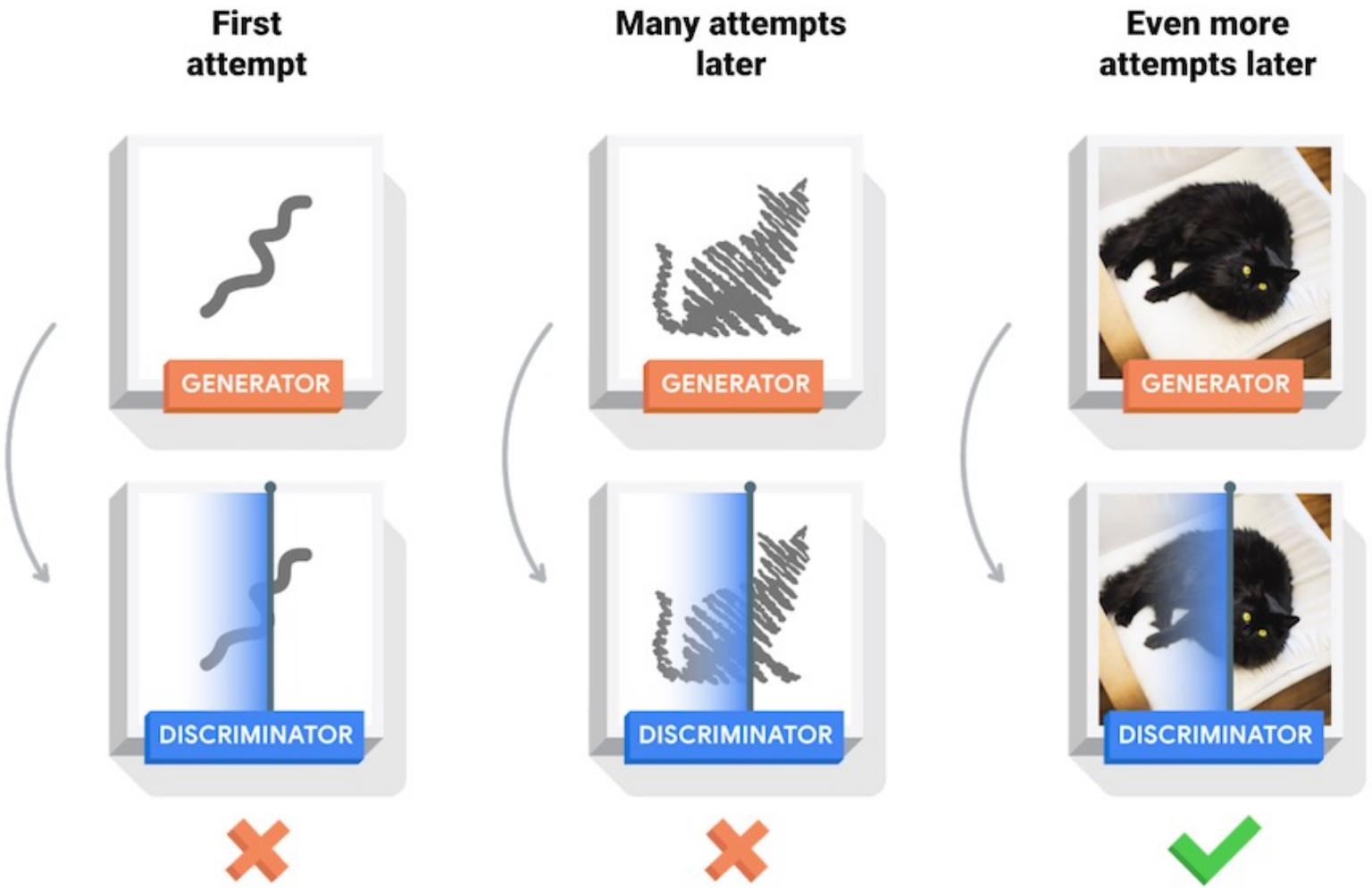
- Generator G produces noise
- Discriminator D learns to classify noise vs. real
- D tells G how to make noise look more real
- G starts generating real-looking images

While True:

- D gets confused, tries harder to distinguish real vs. fake images
- G gets better at generating fake images
- D gets better at identifying fake images



GAN Training Process



Multiple Interacting Neural Networks

This is the first time in this class where we build a single system with **multiple neural networks** which **interact with each other**

This is a recurring theme in many new areas of deep learning

GAN Loss Function

$$\min_G \max_D V(D,G)$$

Minimize loss for Generator; Maximize loss for Discriminator

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

Discriminator output for real data x

Discriminator output for generated fake data G(z)

GAN Loss Function

$$\min_G \max_D V(D,G)$$

For Discriminator:

Maximize to get D(X) as close to 1

Maximize to get D(G(z)) as close to 0

$$V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

Discriminator output for real data x: as close to 1 as possible

Discriminator output for generated fake data G(z): as close to 0 as possible

GAN Loss Function

For Generator (only cares about generated images):

Minimize to get D(G(z)) as close to 1

$$\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z))]$$

Discriminator output for generated fake data G(z): as close to 1 as possible

GAN Training Process

Alternate between:

- Gradient ascent for Discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

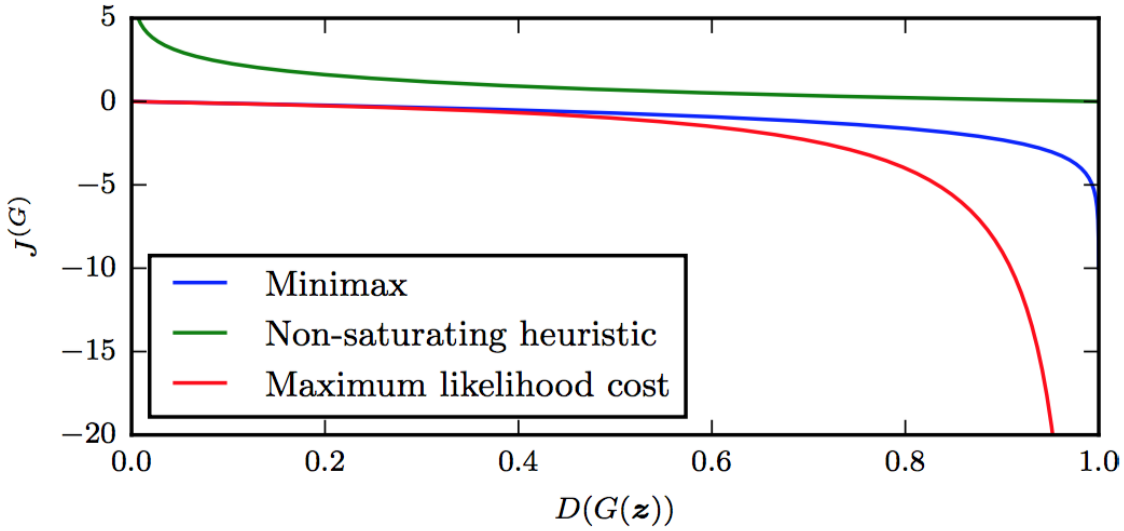
- Gradient descent for Generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Practical Consideration #1

In practice, gradient ascent to maximize likelihood of discriminator being wrong has a higher gradient signal than minimizing likelihood of discriminator being correct when the sample is likely fake, so this works better and is therefore what tends to be implemented

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$



There are several other practical things to figure out to successfully train a GAN

- The model parameters oscillate, destabilize and never converge
- The generator produces limited varieties of samples
- Gradients “vanish” and learn nothing
- Overfitting
- Highly sensitive to the hyperparameter selection
- … (the list goes on) …

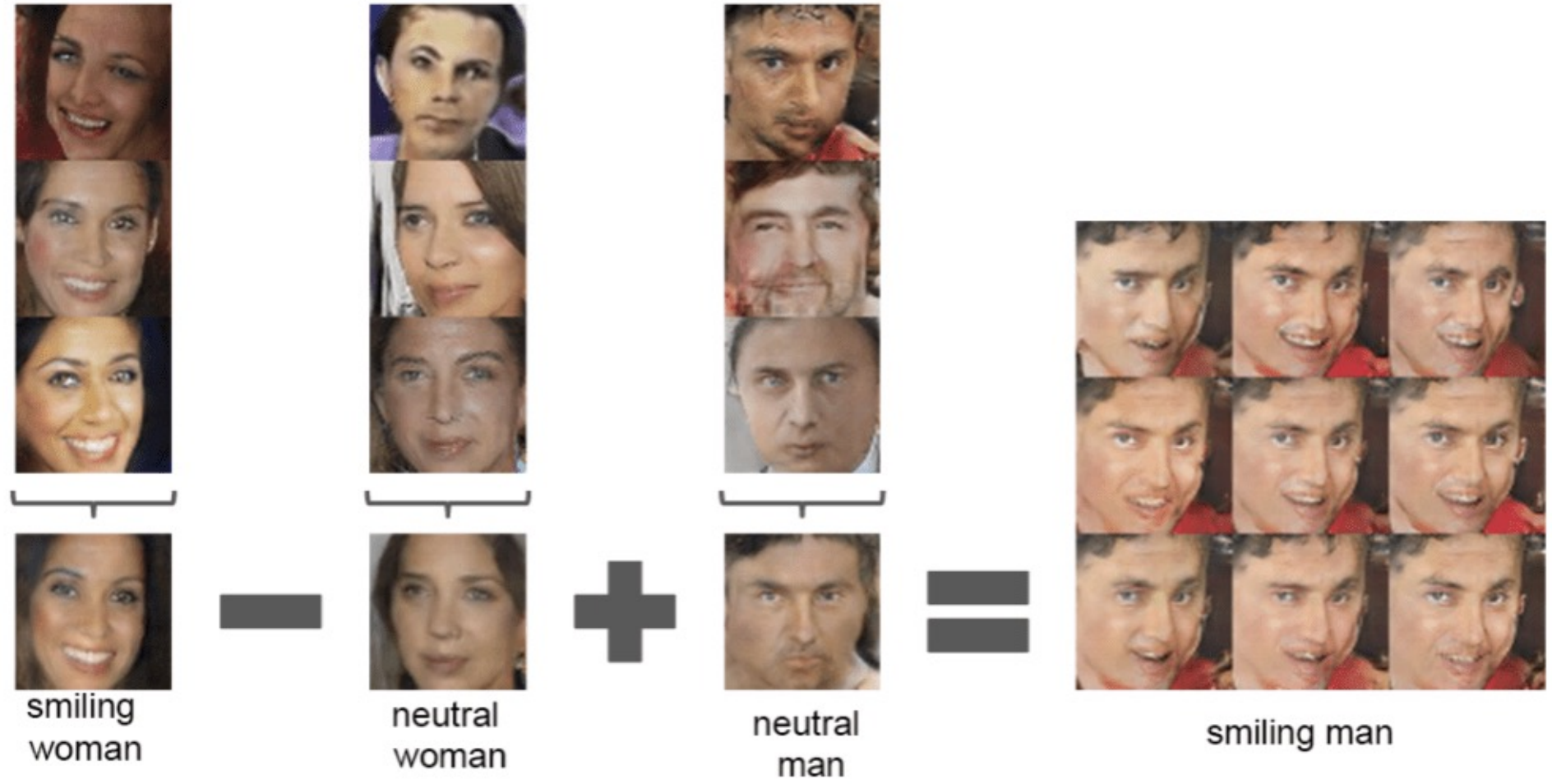
But if you eventually get it right, the applications can be (and have been) very powerful…

Applications of Generative Models

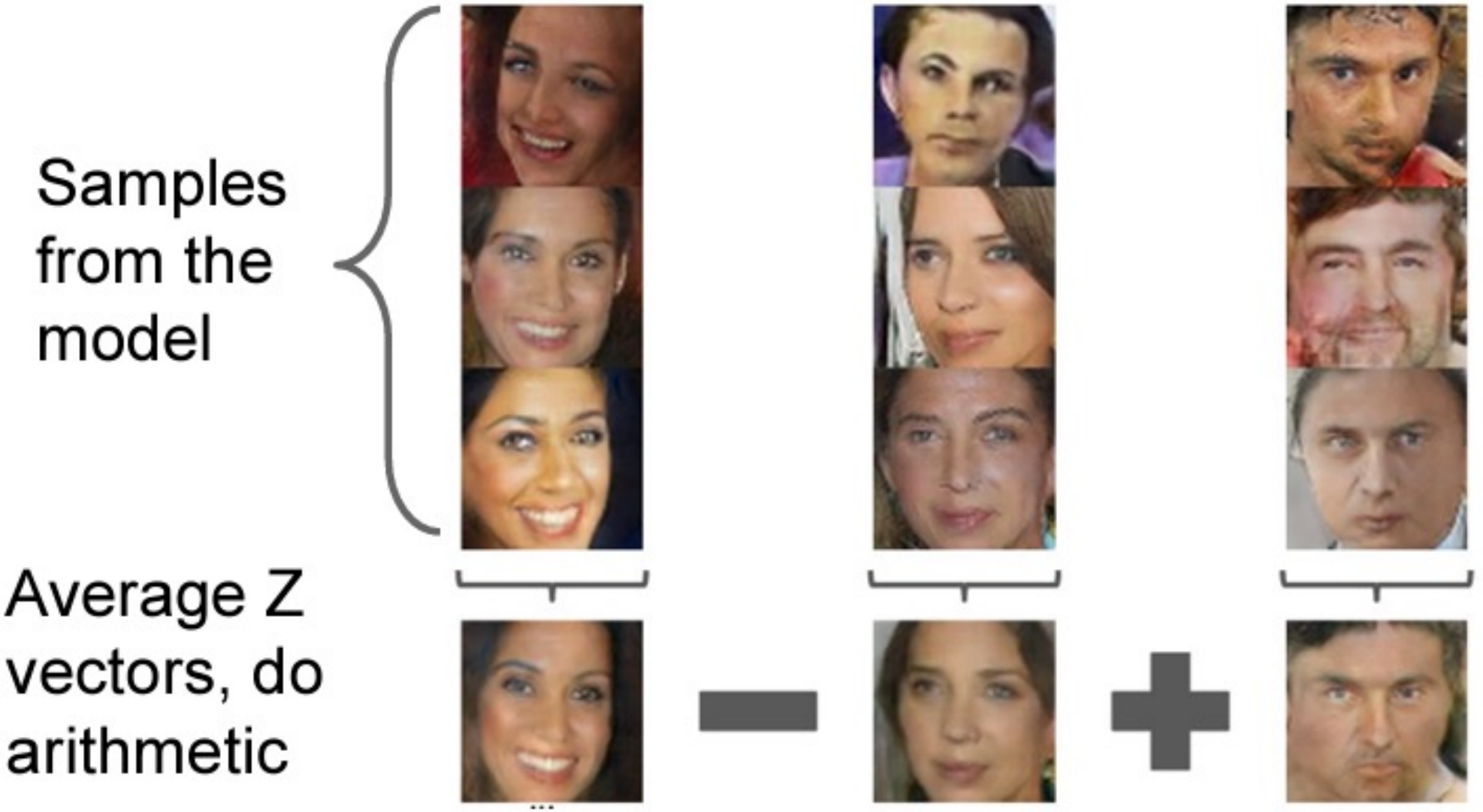
www.thispersondoesnotexist.com

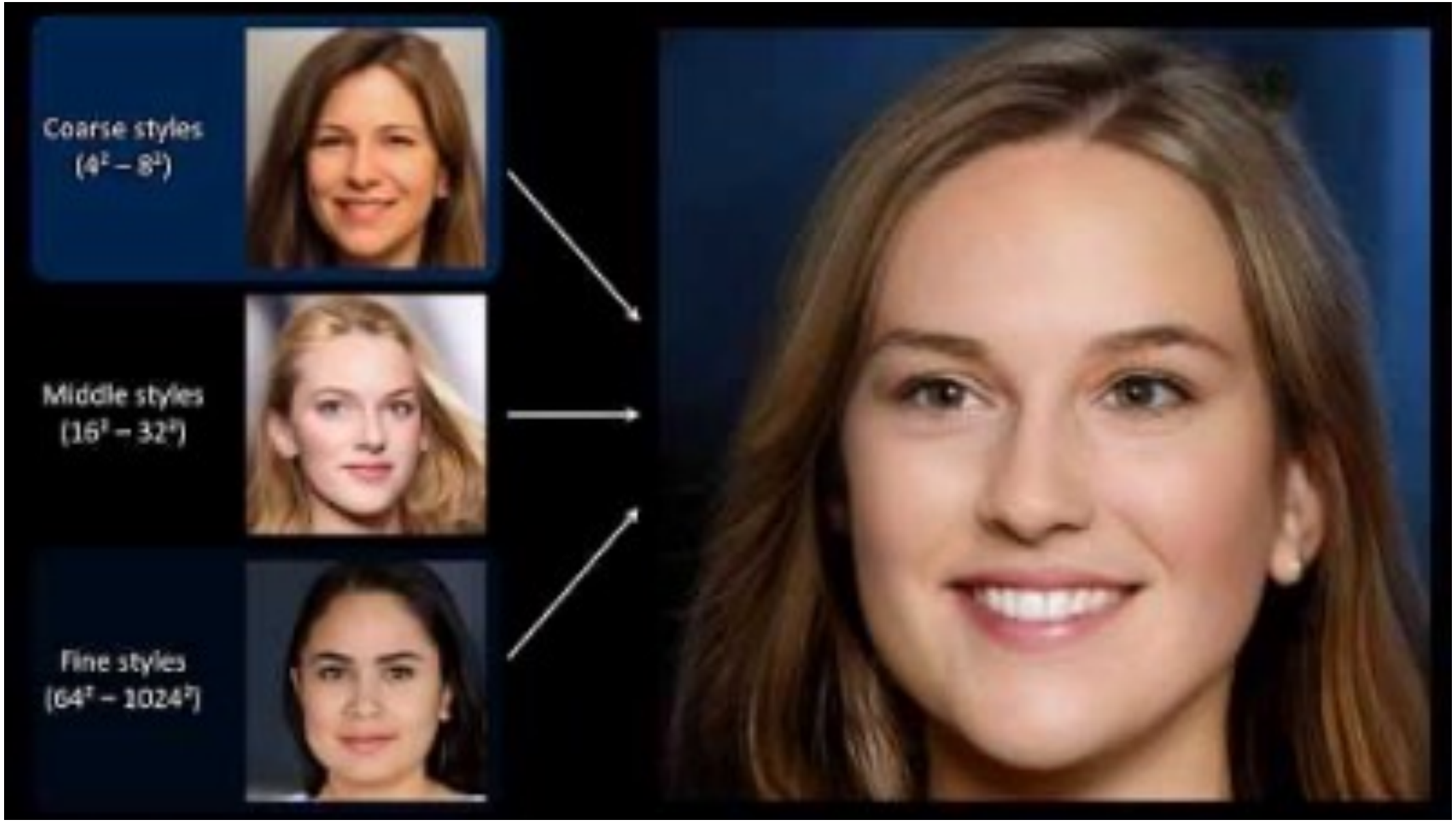
Interpolating between random points in latent space

Latent Space Math



Latent Space Math




Coarse styles
(4² – 8²)
Middle styles
(16² – 32²)
Fine styles
(64² – 1024²)

Other Applications of Generative Models

- Deep fakes
- Photograph editing
- Inspiration for music, art, designs, …
- Example generation
- Converting one modality to another (e.g., image to text)
- Data augmentation
  - For general performance increase
  - For creating fairer models
- …

Obama Deepfake (2018)

Volodymyr Zelenskyy Deepfake (2022)