# 3011979 Intro to Deep Learning for Medical Imaging

## L12: Neural network training & image segmentation architecture

Apr 23rd, 2021

Sira Sriswasdi, Ph.D.
Research Affairs, Faculty of Medicine
Chulalongkorn University

# Advanced ANN training techniques

# Representation

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| man | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| woman | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| boy | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| girl | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| prince | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| princess | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| queen | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| king | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| monarch | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Image from hackermoon.com

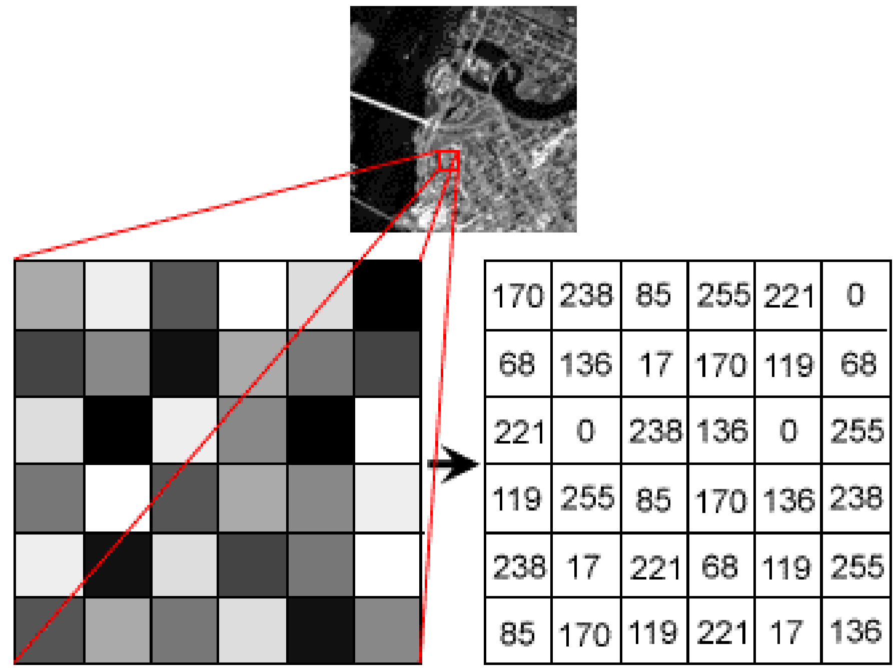



Image from naushardsblog.wordpress.com

- Representation = presentation of information contained in the data
- Simple representation is not useful for learning

# Good representation

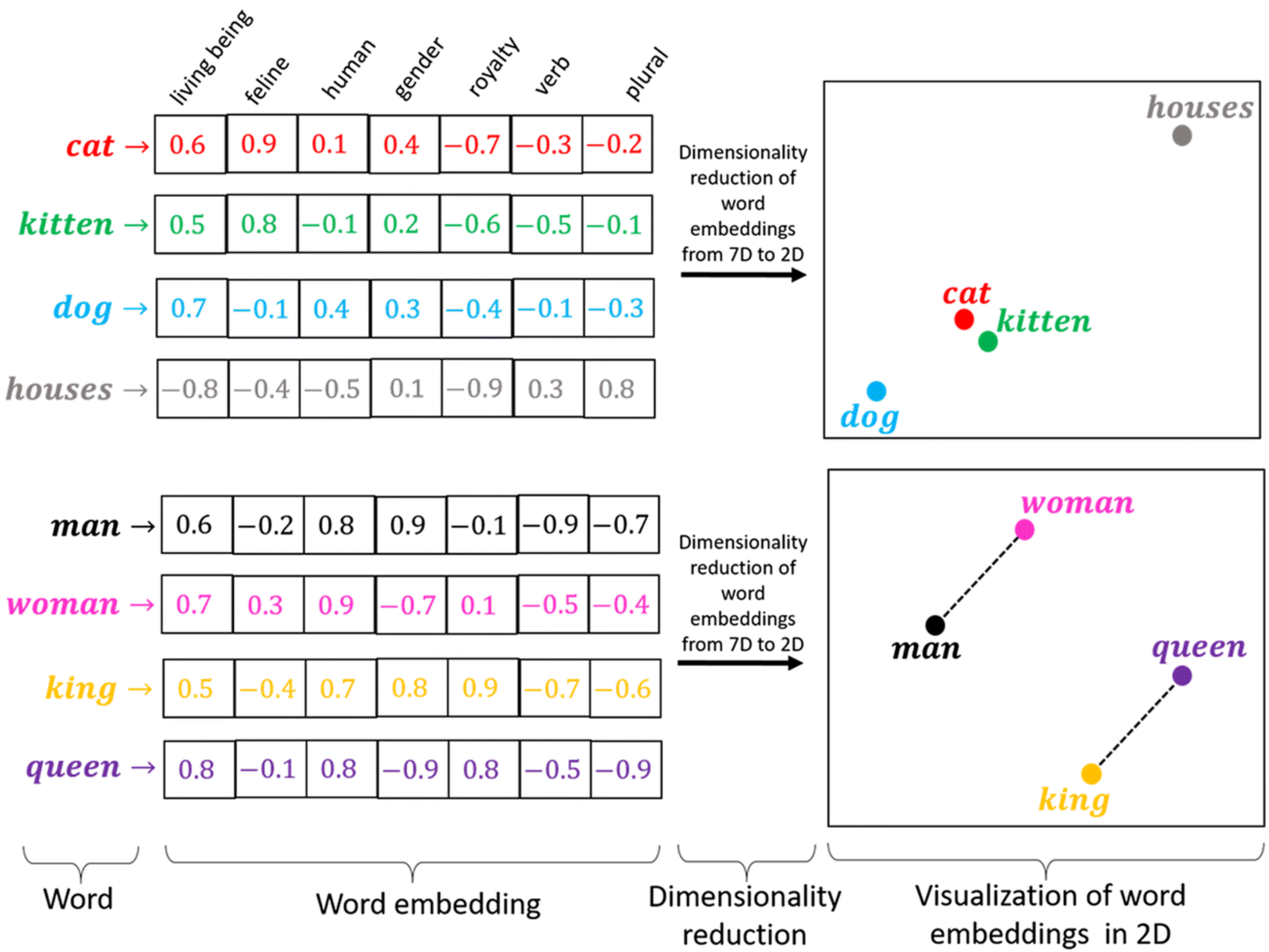
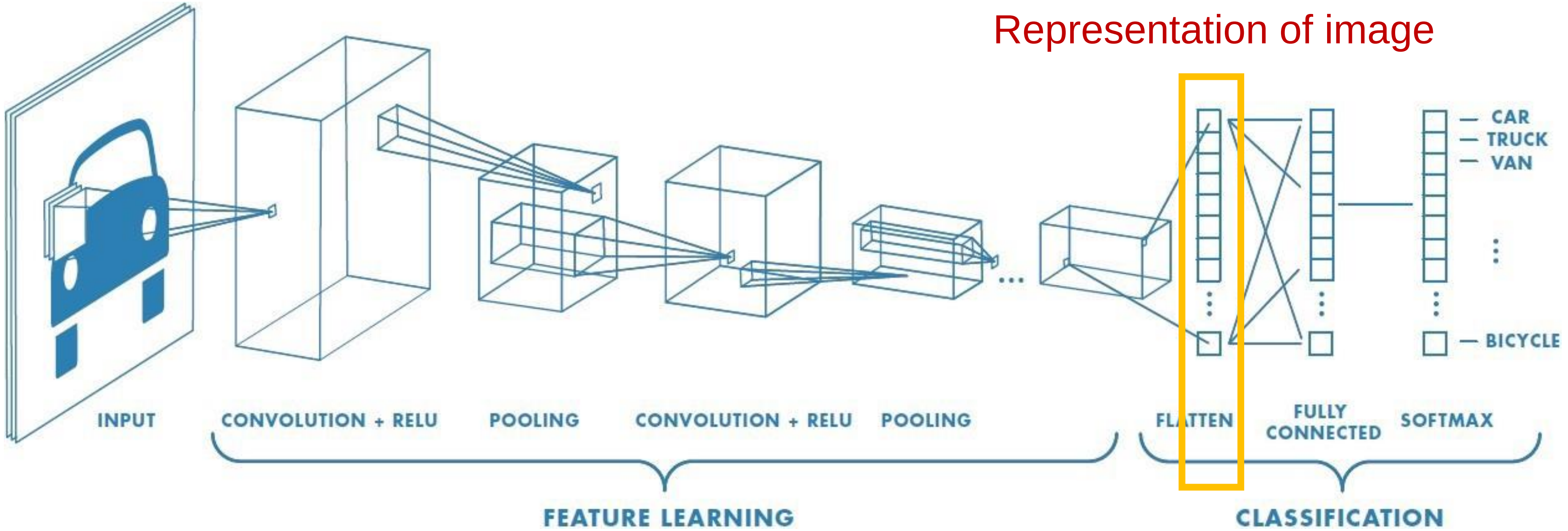



Image from medium.com

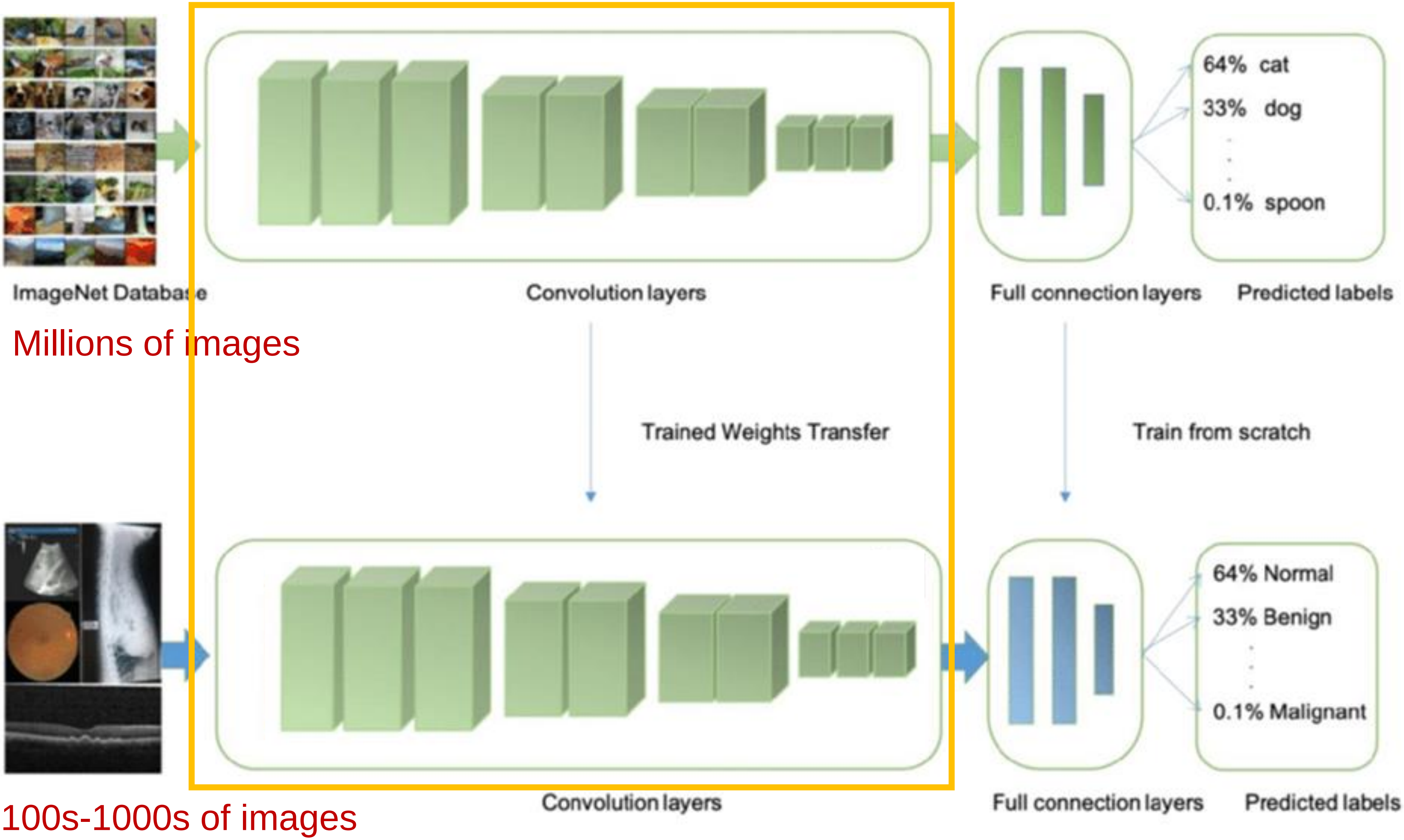
# Representation learning




Source: towardsdatascience.com by Saha, S.

- Representation learning = machine learning for obtaining a good representation of the data
  - High performance ←→ Good representation
- Convolution layers in a CNN

# Transfer learning




Source: Xu et al. Theranostics (2019)

# Tensorflow pre-defined models

TensorFlow > API > TensorFlow Core v2.4.1 > Python

## Module: tf.keras.applications

`densenet` module: DenseNet models for Keras.

`efficientnet` module: EfficientNet models for Keras.

`imagenet_utils` module: Utilities for ImageNet data preprocessing & prediction decoding.

`inception_resnet_v2` module: Inception-ResNet V2 model for Keras.

`inception_v3` module: Inception V3 model for Keras.

`mobilenet` module: MobileNet v1 models for Keras.

`mobilenet_v2` module: MobileNet v2 models for Keras.

`mobilenet_v3` module: MobileNet v3 models for Keras.

`nasnet` module: NASNet-A models for Keras.

`resnet` module: ResNet models for Keras.

# Using pre-trained models

## tf.keras.applications.DenseNet121

```
tf.keras.applications.DenseNet121(
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000
)
```

- include_top
  - Whether to include the fully connected layers
  - Set to False if to train fully connected layers from scratch
    - Remember to add new fully connected layers
- weights
  - Path to pretrained model weights
  - “imagenet” is provided by default

# More pre-trained models

## tf.keras.applications.ResNet50

```
tf.keras.applications.ResNet50(
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000, **kwargs
)
```

## tf.keras.applications.InceptionResNetV2

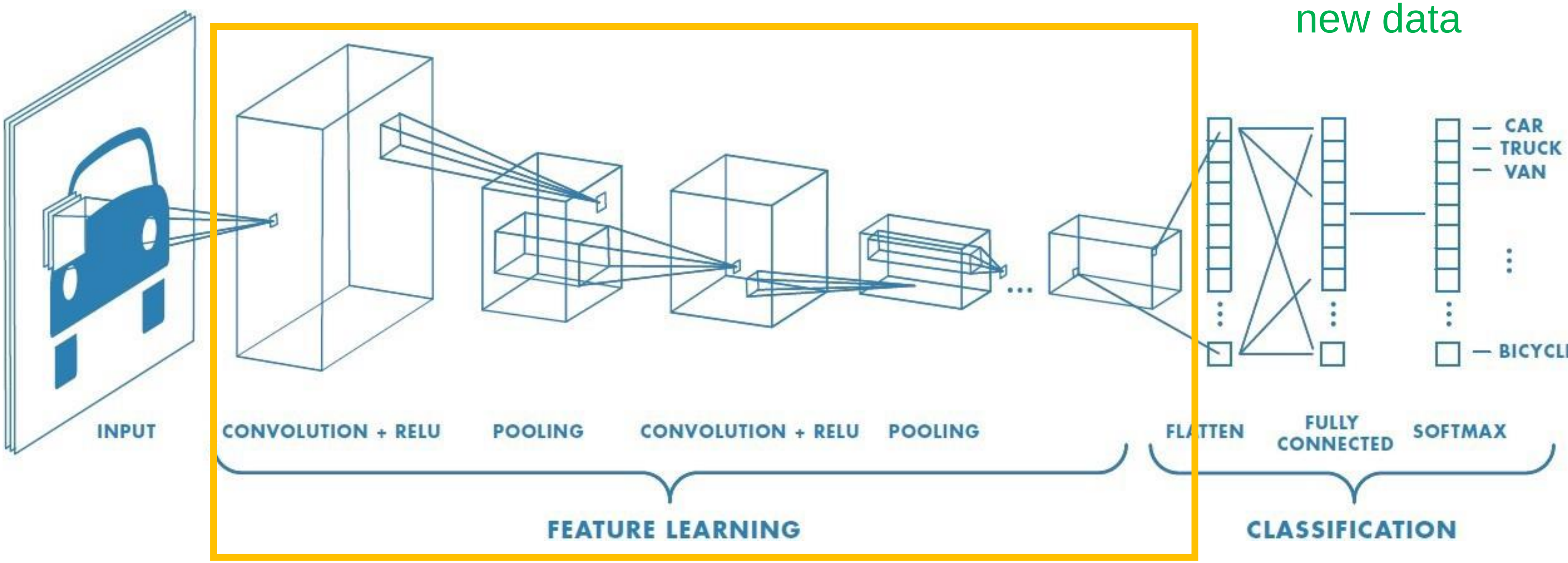
```
tf.keras.applications.InceptionResNetV2(
    include_top=True, weights='imagenet', input_tensor=None,
    input_shape=None, pooling=None, classes=1000,
    classifier_activation='softmax', **kwargs
)
```

# Transfer learning

New data

Load weights from ImageNet pretrained

Train using new data




- Fully connected part is untrained
- Convolution part is pretrained
  - Freeze its weights

# Layer freezing

```
base_model = keras.applications.Xception(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 3),
    include_top=False)  # Do not include the ImageNet classifier at the top.
```

```
base_model.trainable = False
```

```
inputs = keras.Input(shape=(150, 150, 3))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as you will
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = keras.layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
```

- https://www.tensorflow.org/guide/keras/transfer_learning

# Fine-tuning

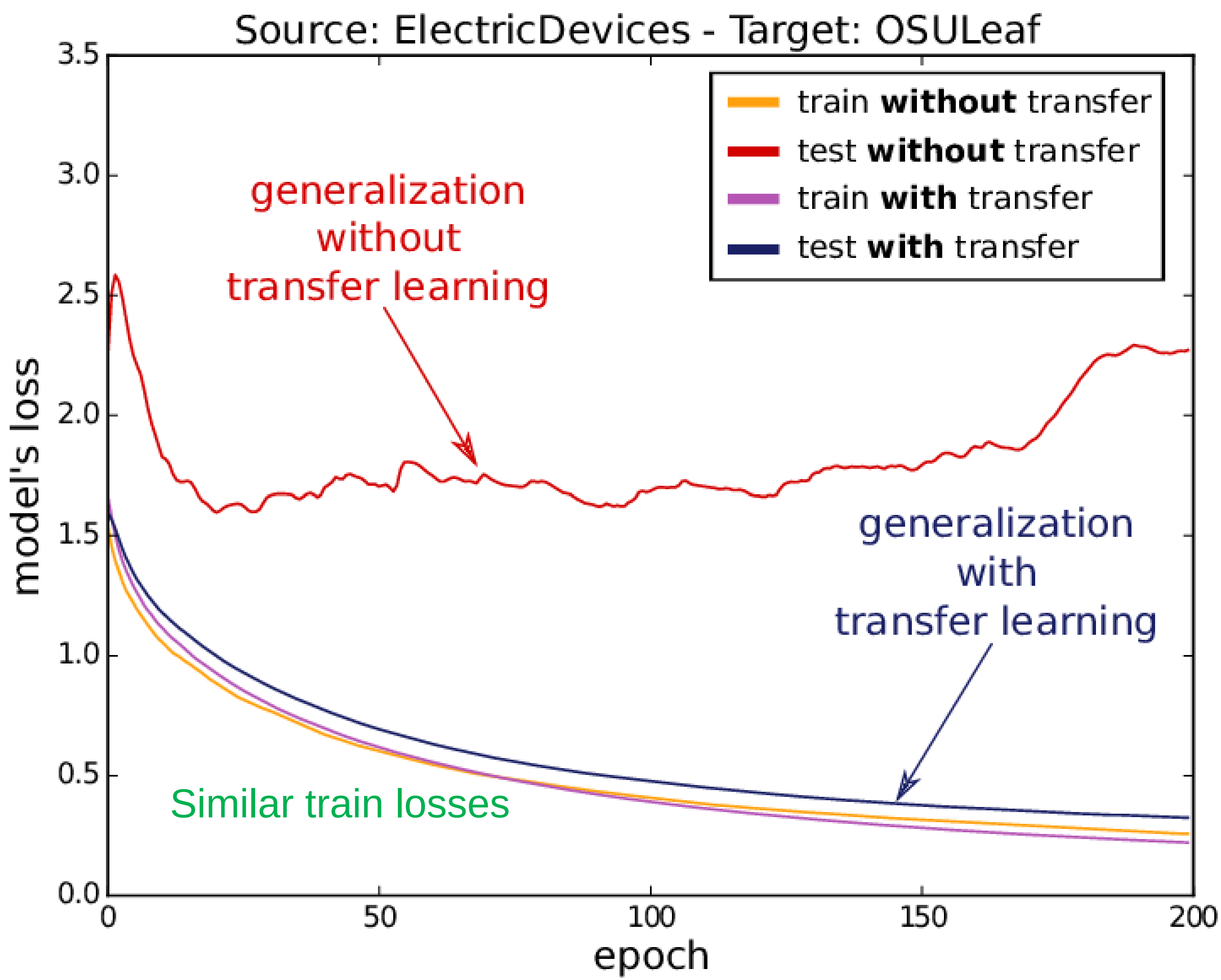
**Transfer learning**

- Load pretrained model
- Freeze pretrained layers
- Attach new fully connected layers
- Train model using new dataset

**Fine-tuning**

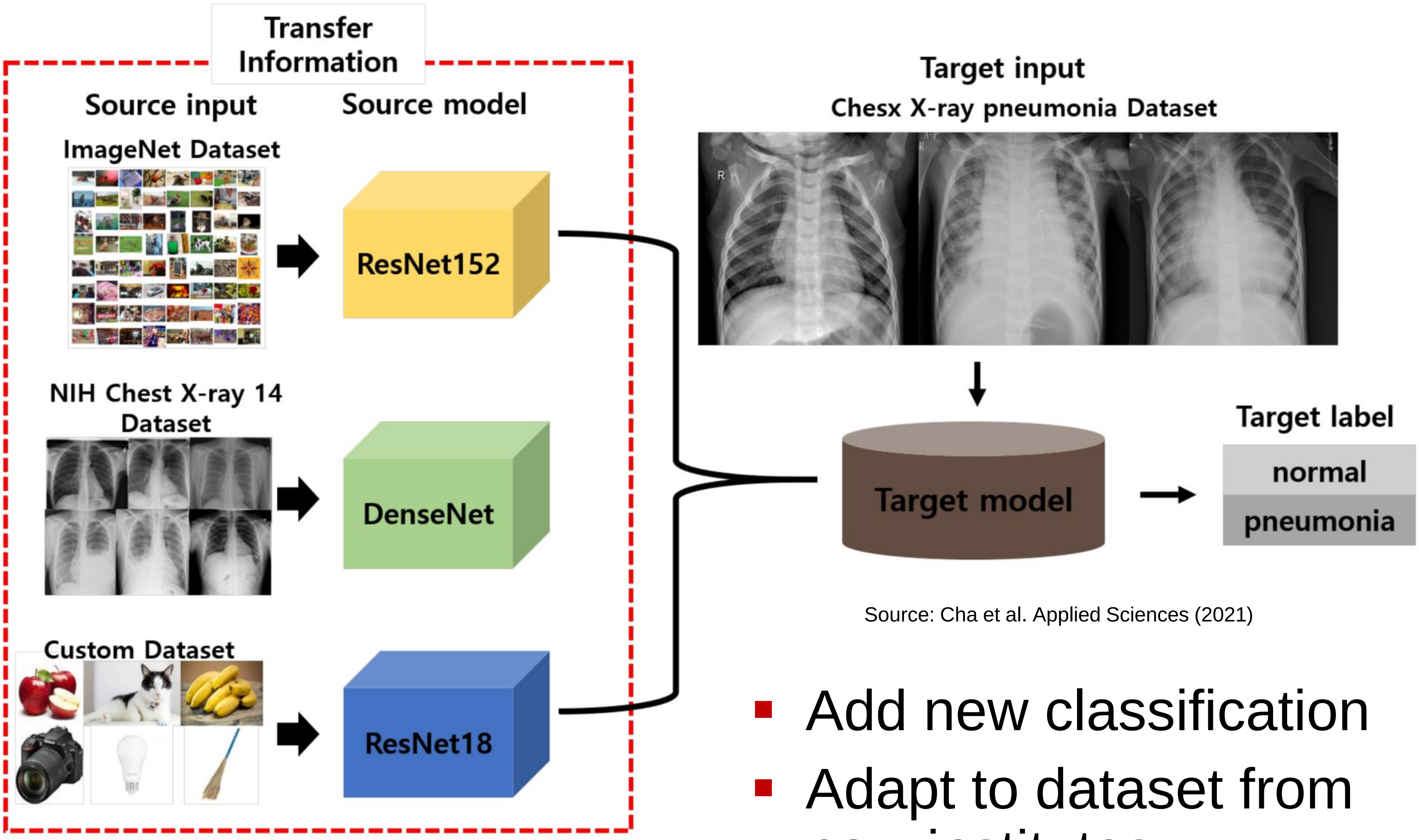
- Unfreeze pretrained layers
- Continue to train the model using new dataset

# Impact of transfer learning




Source: Fawaz et al. IEEE Big Data (2018)

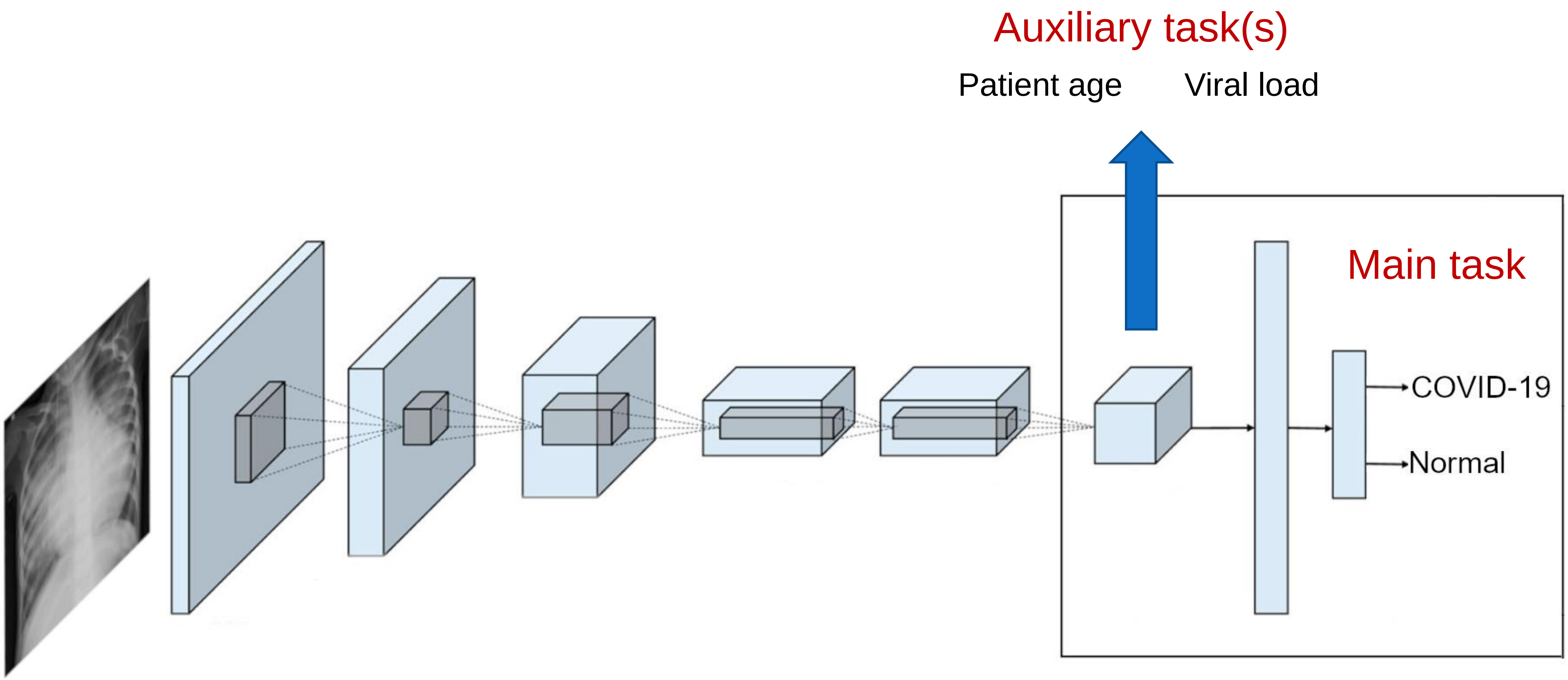
# Transfer learning application




Source: Cha et al. Applied Sciences (2021)

- Add new classification
- Adapt to dataset from new institutes

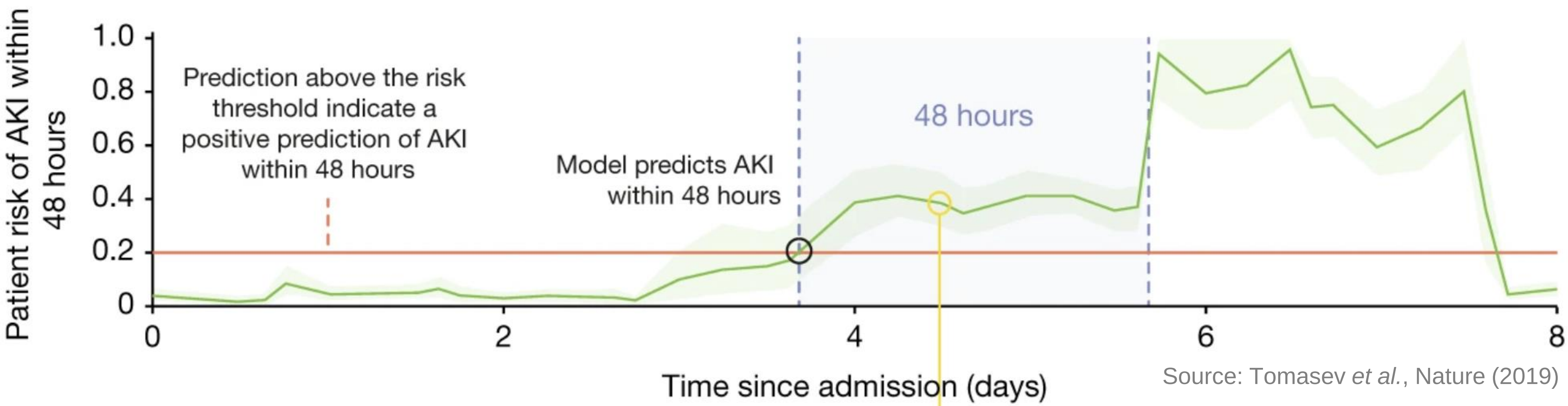
# Auxiliary task



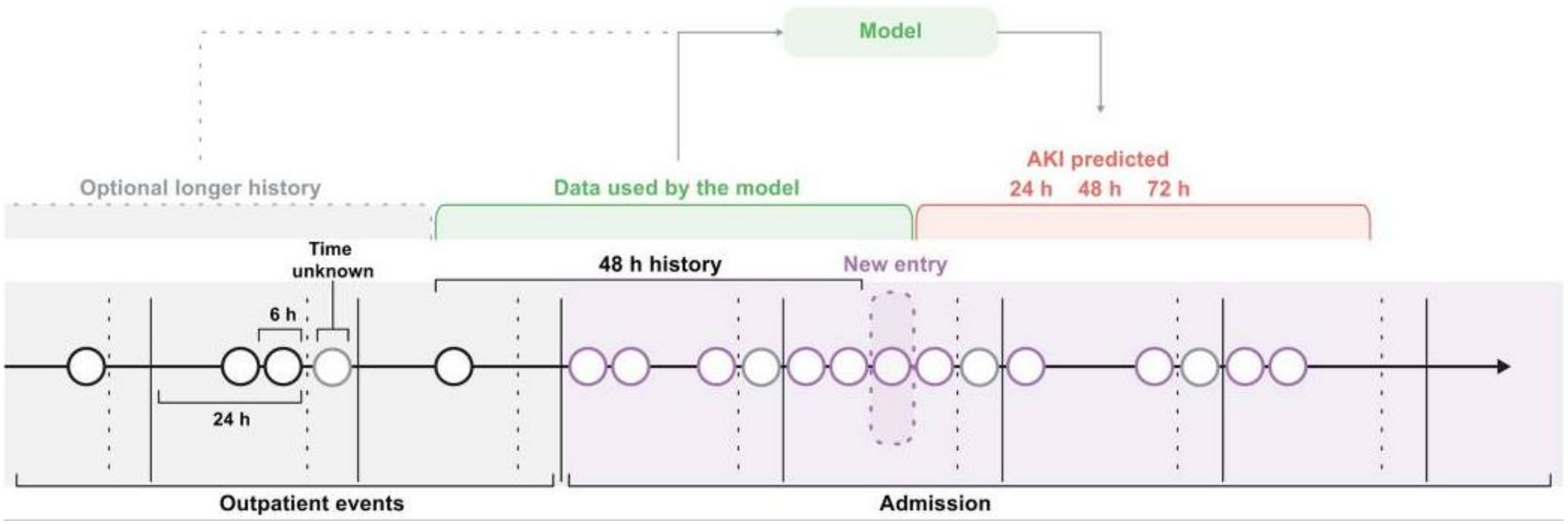

Source: Cortes and Sanchez. IEEE Latin America Transaction (2021)

- Good representation should capture multiple characteristics of the data
- Auxiliary task(s) guides the early layers toward good representation

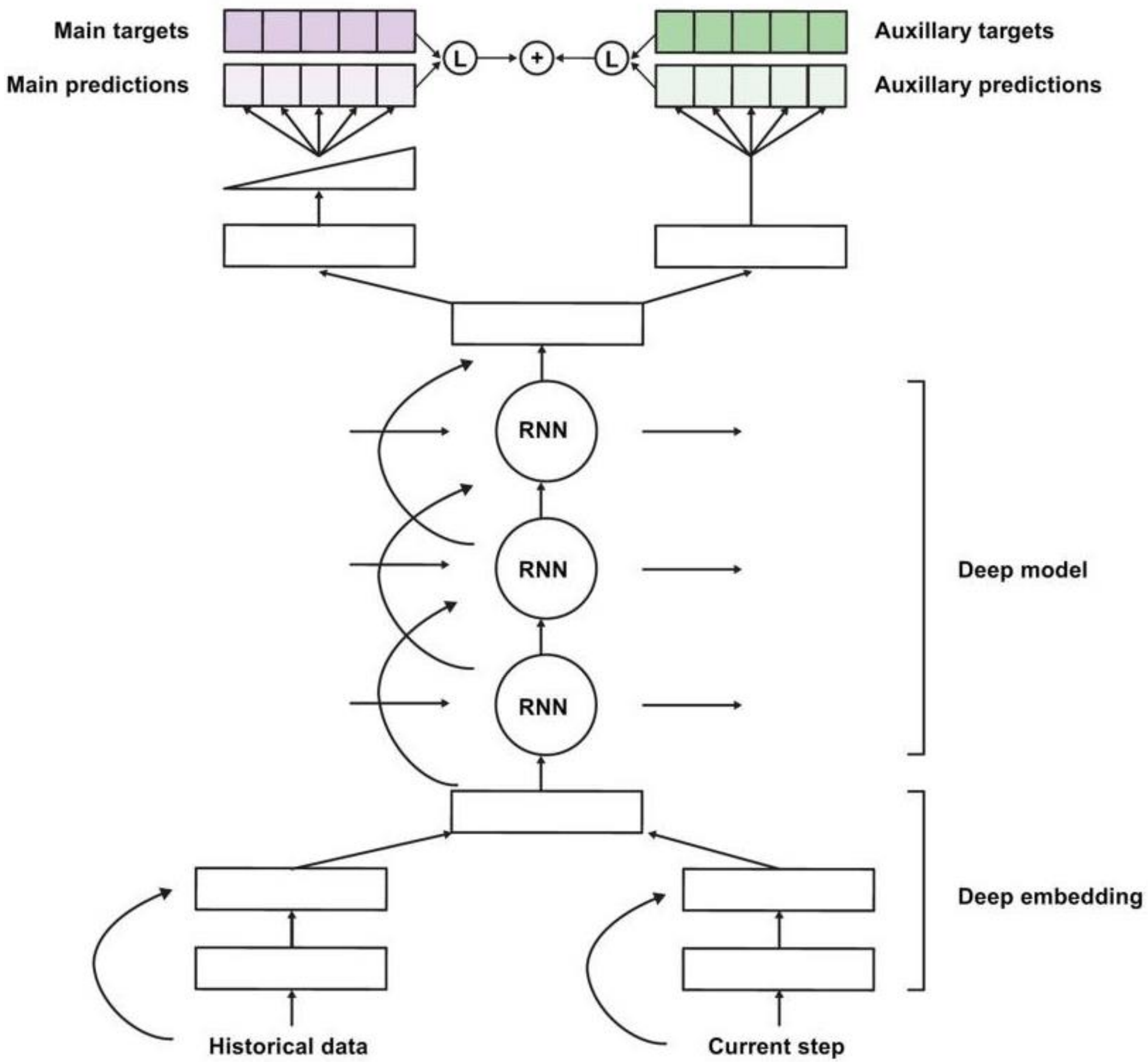
# Acute kidney injury prediction




Source: Tomasev *et al.*, Nature (2019)



# Auxiliary tasks




Source: Tomasev *et al.*, Nature (2019)

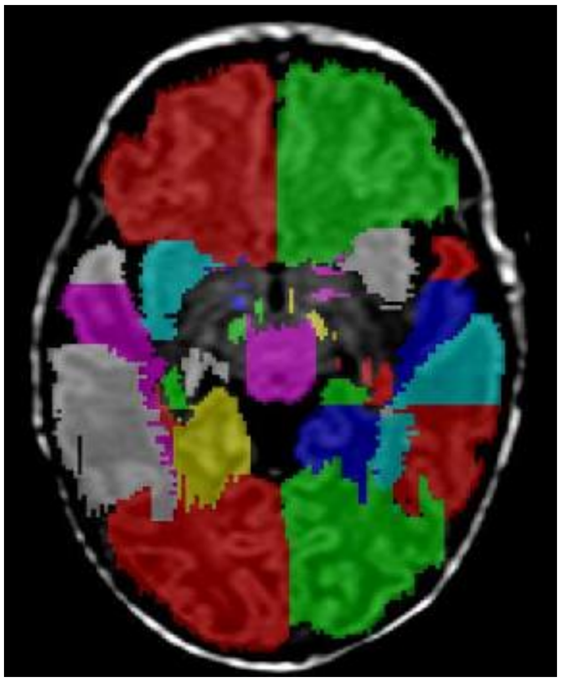
**Main task**: Predict occurrence of AKI within the next 48 hours

**Auxiliary tasks**: Predict maximum values of 7 laboratory features over the next 48 hours

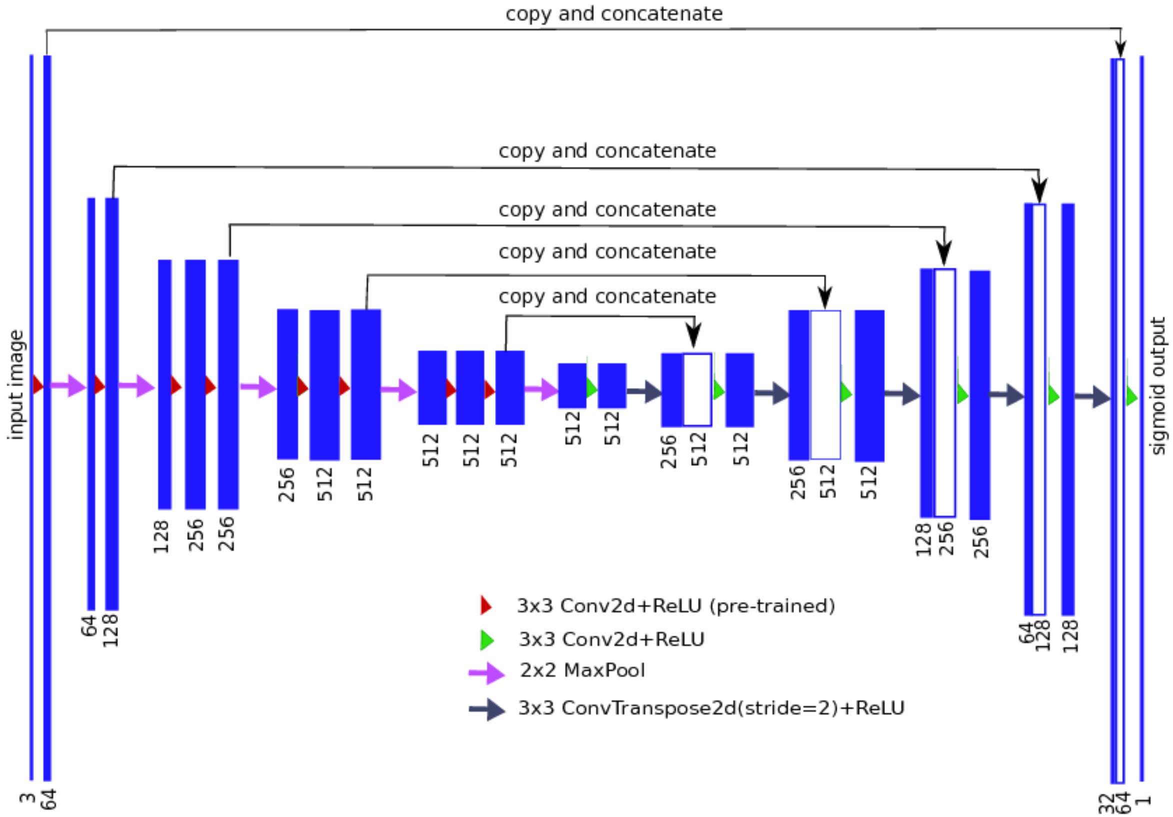
Addition of auxiliary task improve AUC by 3%

# Image segmentation networks

# Segmentation = pixel-level classification



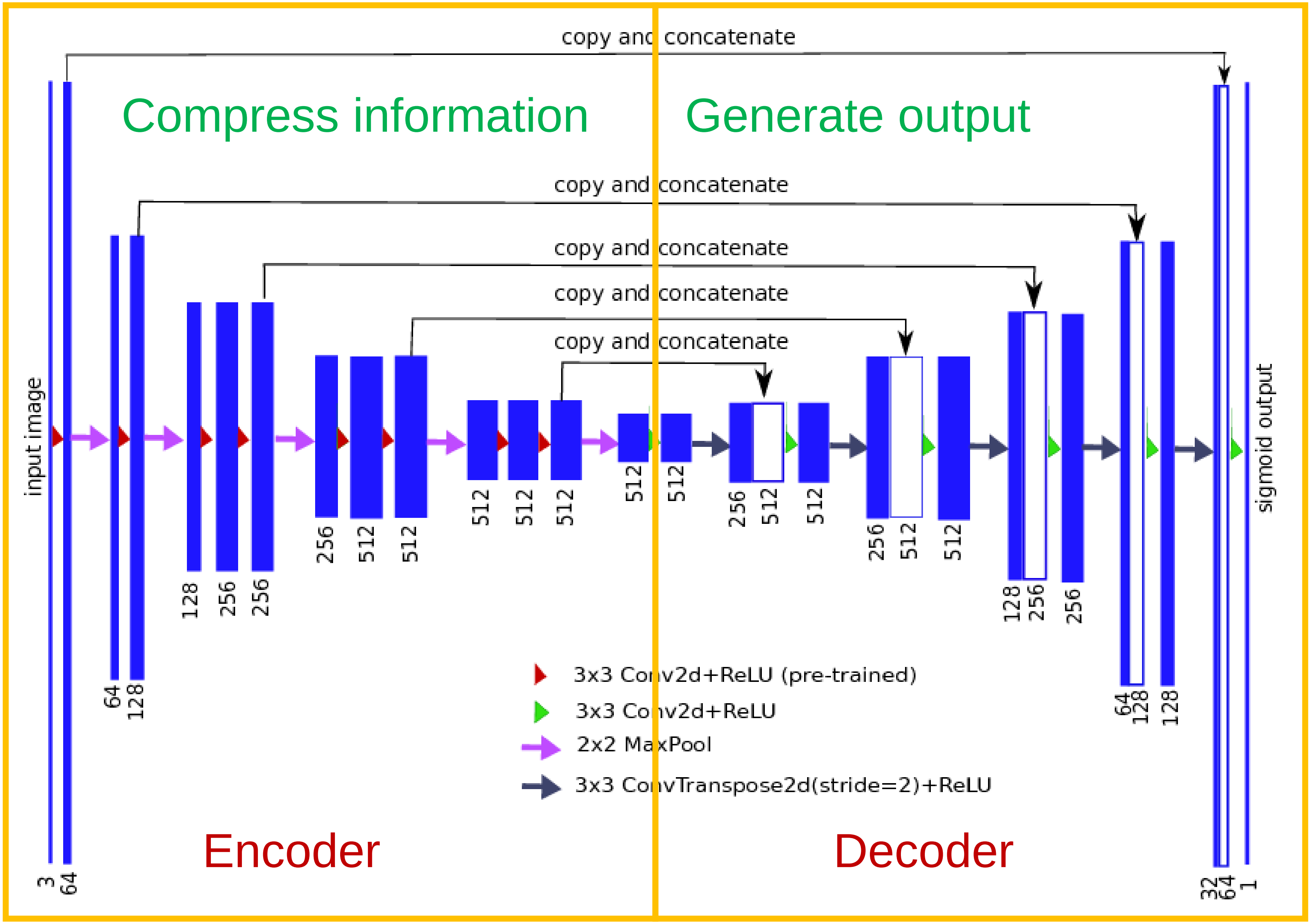
Makropoulos et al. IEEE Trans Med Imaging (2014)




Source: github.com/kaichoulyc/tgs-salts

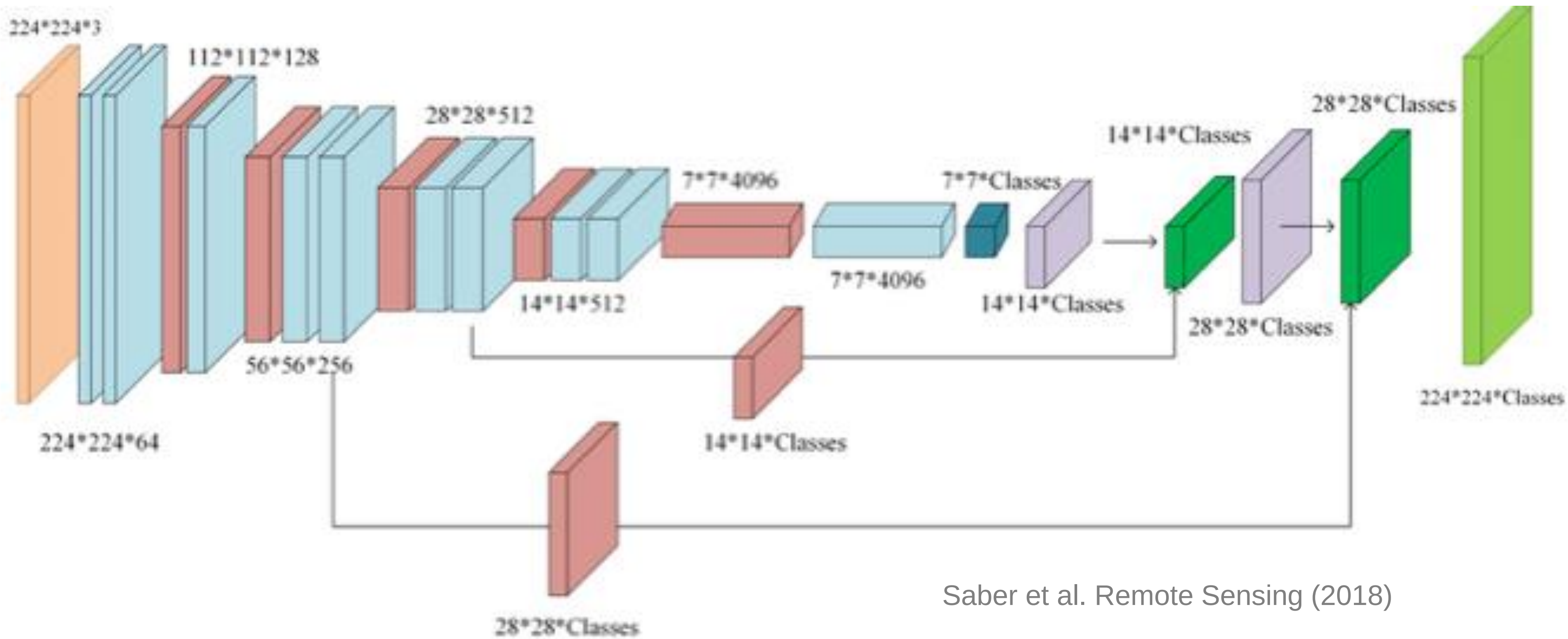
- Output dimension = input dimension!

# Encoder-decoder



# Fully convolutional network

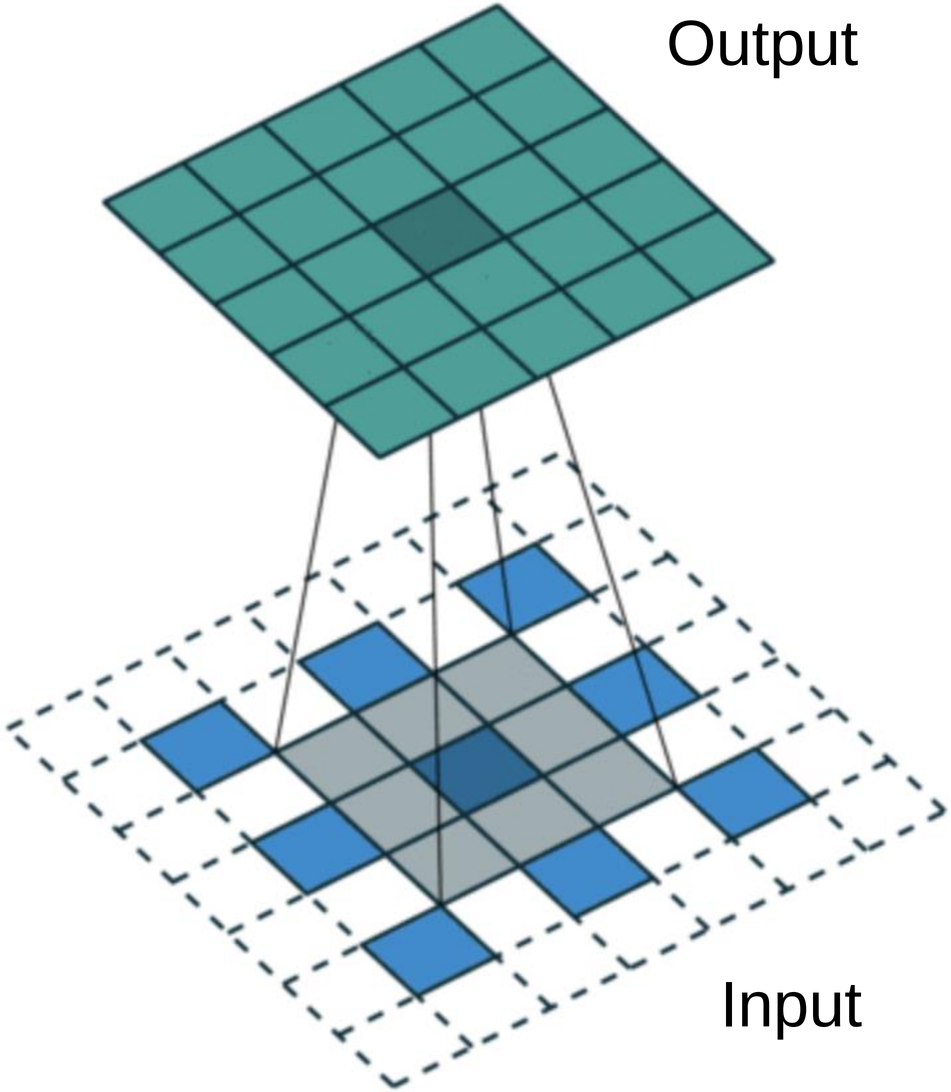
Up-sampling layers




Saber et al. Remote Sensing (2018)

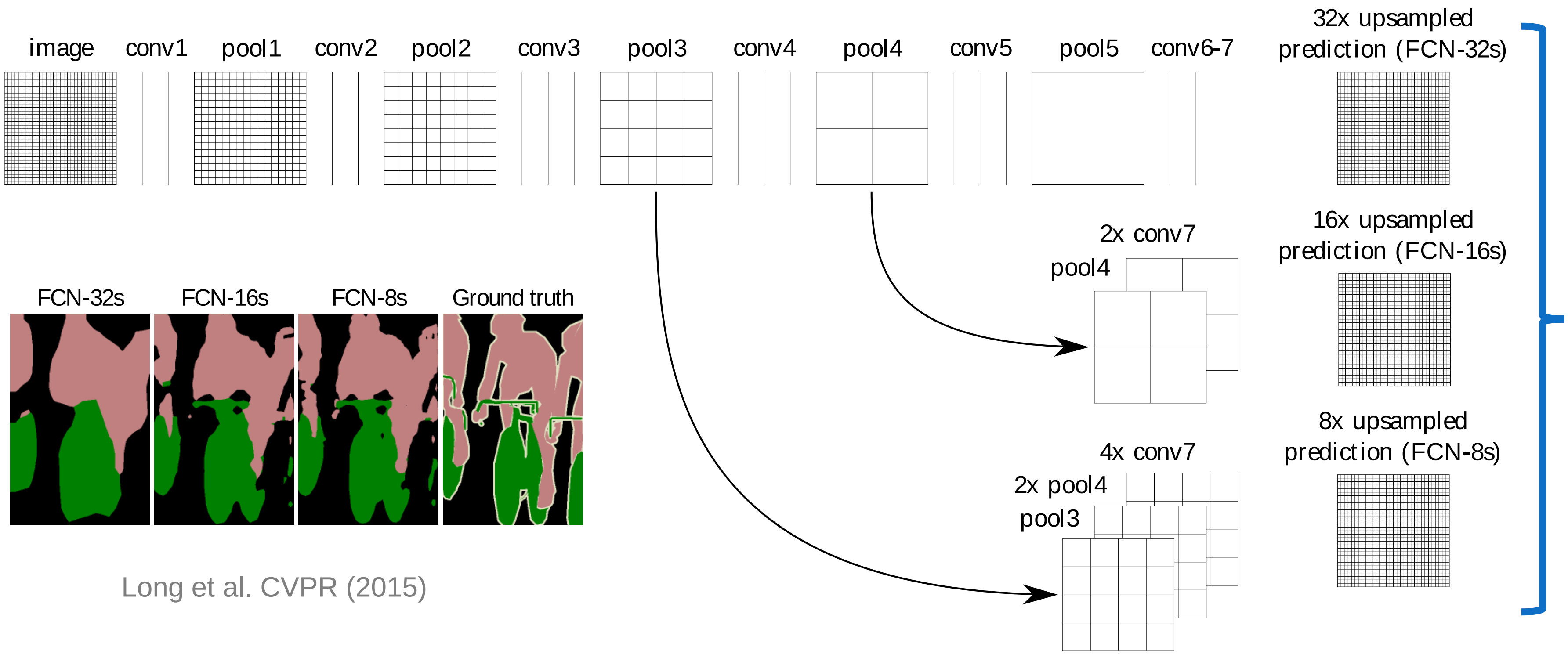
- Using fully connected layers lose spatial and context information

# Up-sampling operation




- Pad input (smaller) image with zeroes
- Apply convolution operation as usual
- Also called "deconvolution" or "transposed convolution"

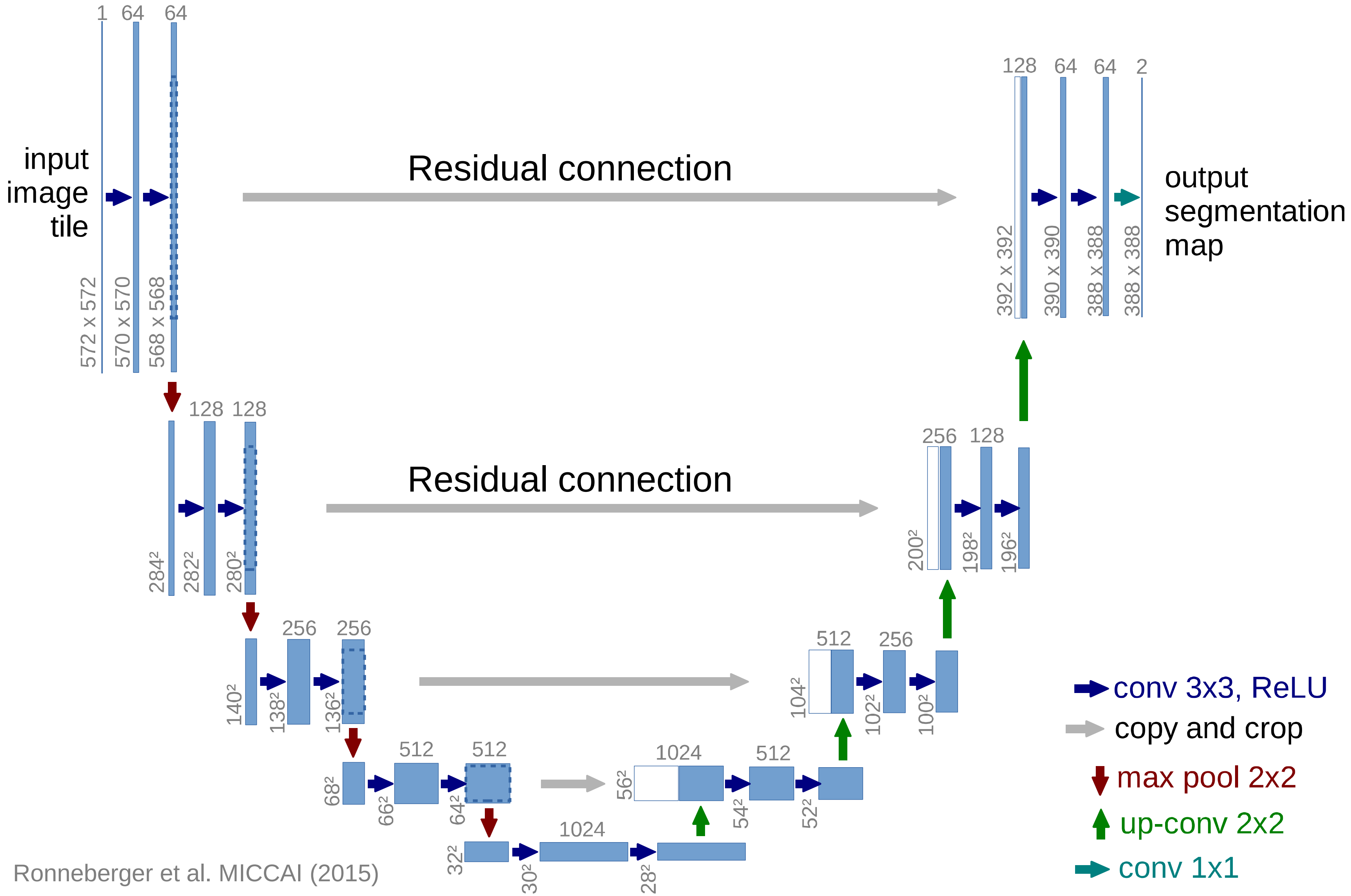
# Resolution vs feature complexity




Long et al. CVPR (2015)

- Each successive convolutional layer increase feature complexity but reduce resolution
- Final outputs = sum of up-sampled outputs from intermediate convolutional layers

# U-Net

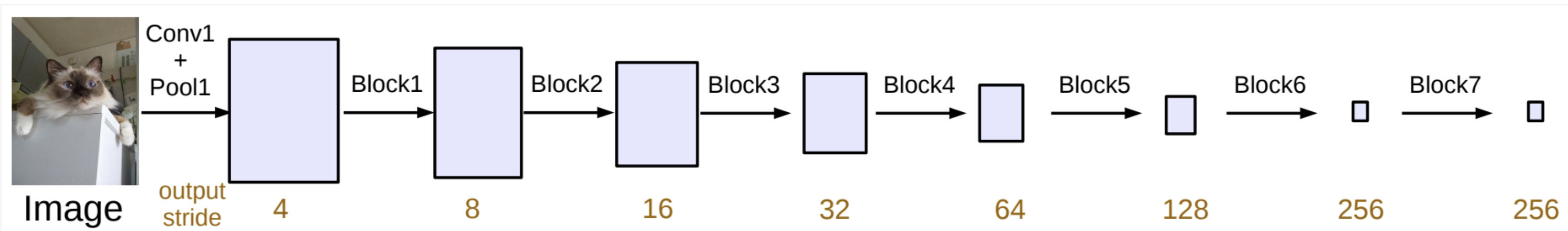



Ronneberger et al. MICCAI (2015)

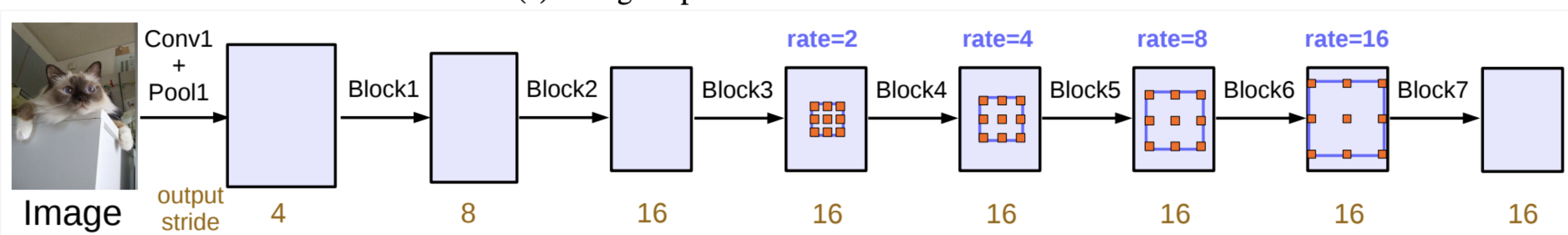
# DeepLab

Chen et al. CVPR (2017)

From left: A dog occupying most of the image, a dog occupying a part of it, and a dog occupying very little space (Images obtained from Unsplash).




(a) Going deeper without atrous convolution.

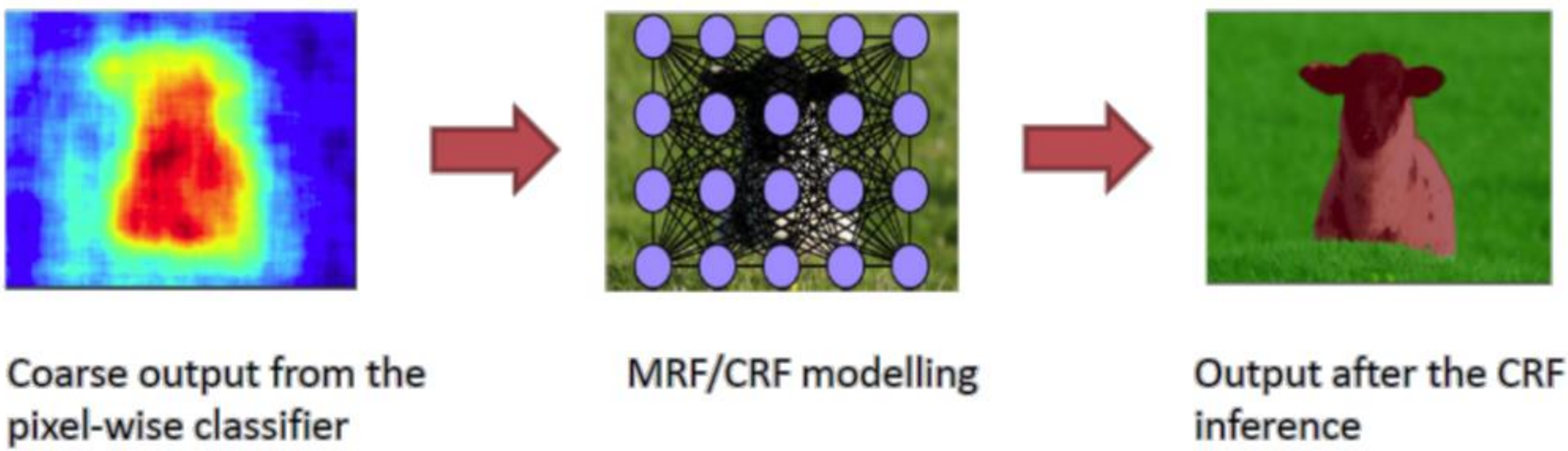



(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

Figure 3. Cascaded modules without and with atrous convolution.

- Handle multiple resolutions with atrous convolution
  - Same kernel size but cover pixels from broader area

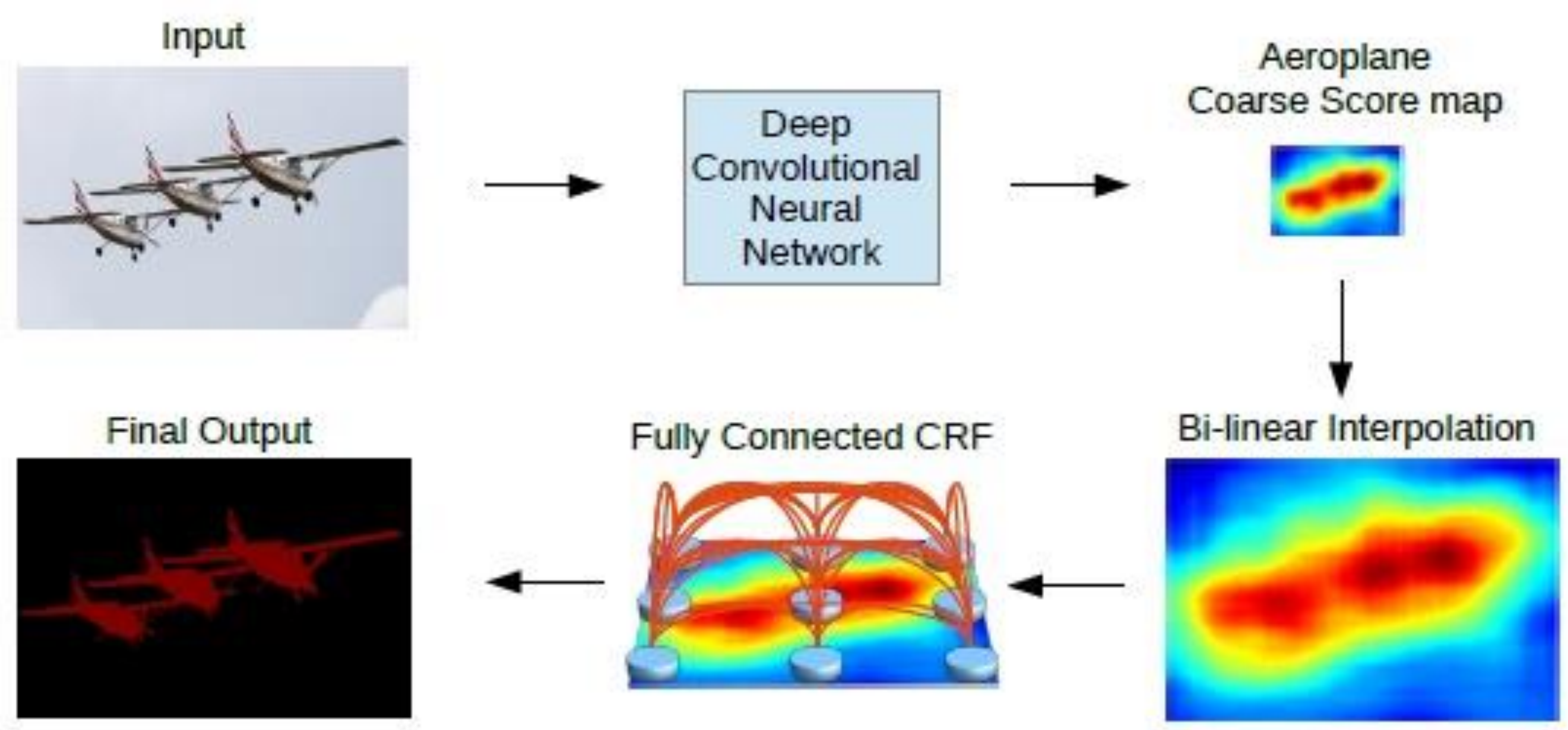
# Limitation of fully convolutional operation




Source: towardsdatascience.com

- Output from fully convolutional network does not take full advantage of information from surrounding pixels
- Additional post-processing with dense layer or conditional/Markov random field

# Conditional random field (CRF)

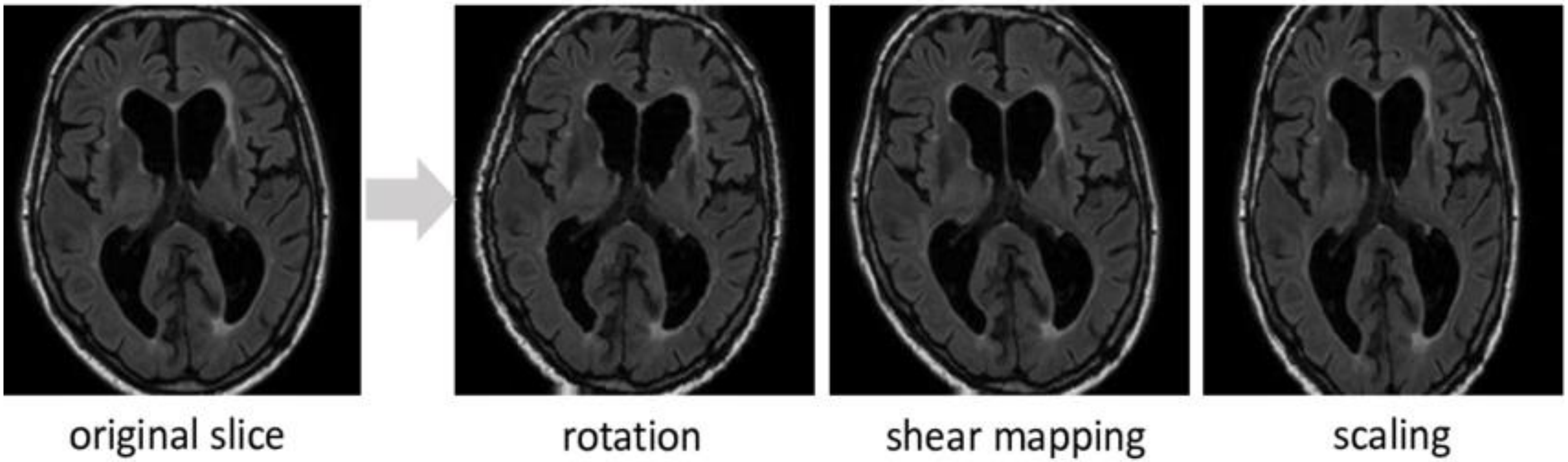



Chen et al. http://arxiv.org/pdf/1606.00915 (2016)

- Nearby pixels are conditionally dependent

# Data augmentation

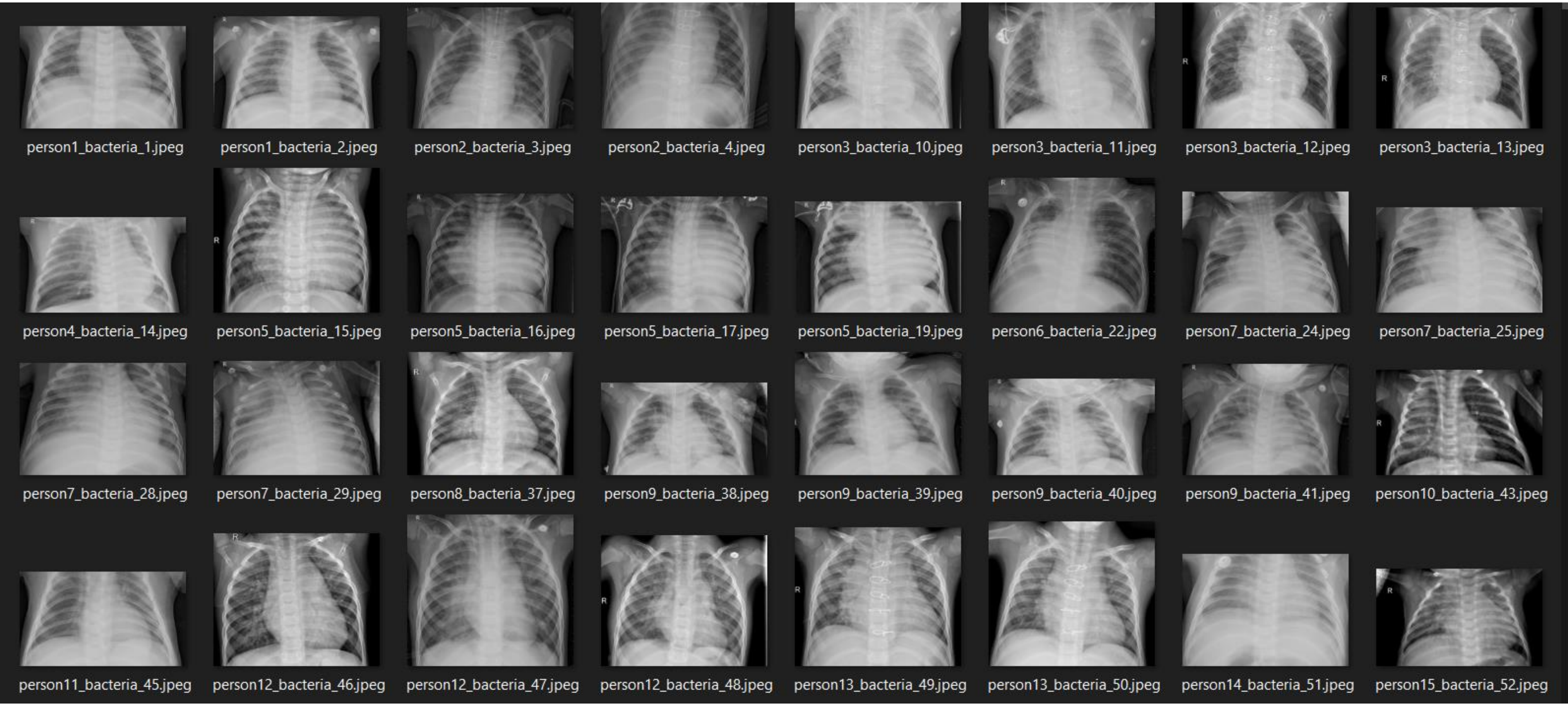
# Small change input → small change in output




Li et al. Neuroimage 183 (2018)

- Slight modifications of input shouldn't change the output by much
  - Same label for classification problem
- ANN should be a smooth function of input

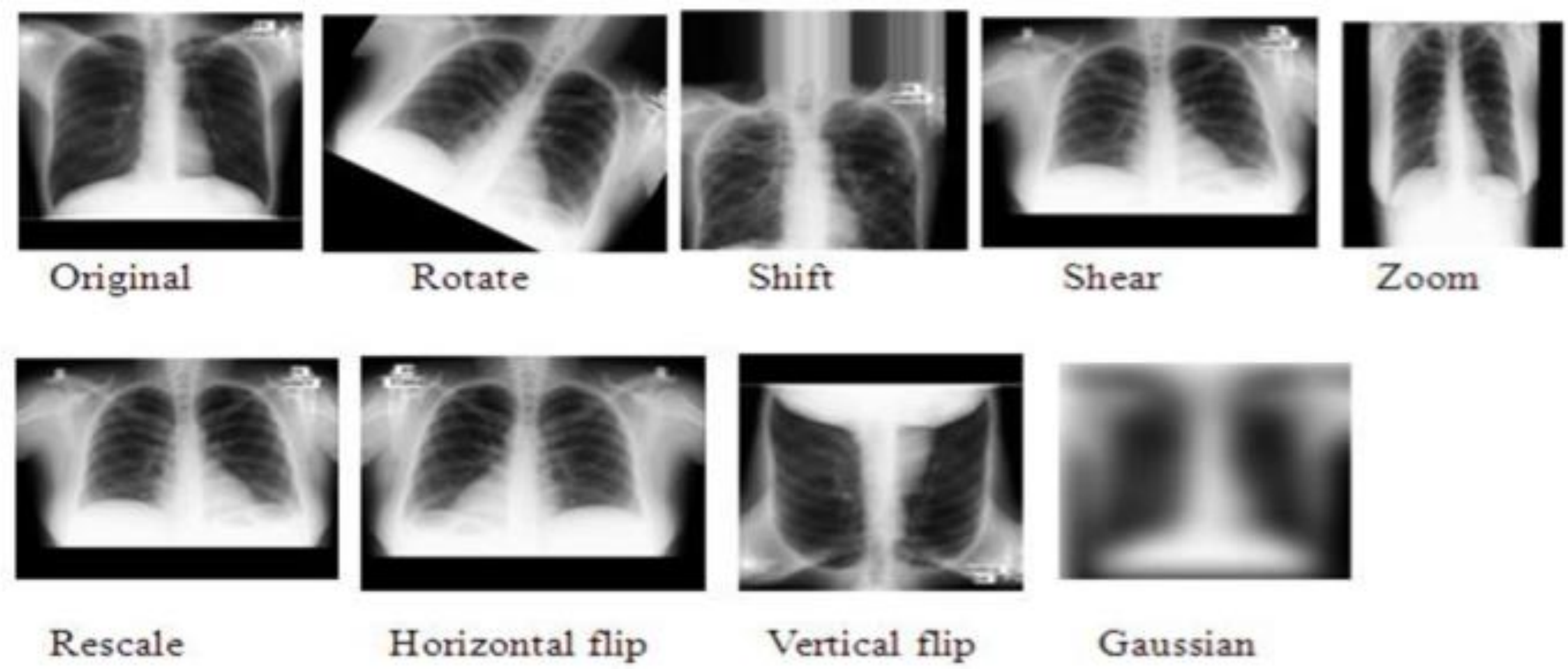
# Real data have variations




Source: analyticsvidhya.com

- ANN should be robust to data noises and variations

# Data augmentation

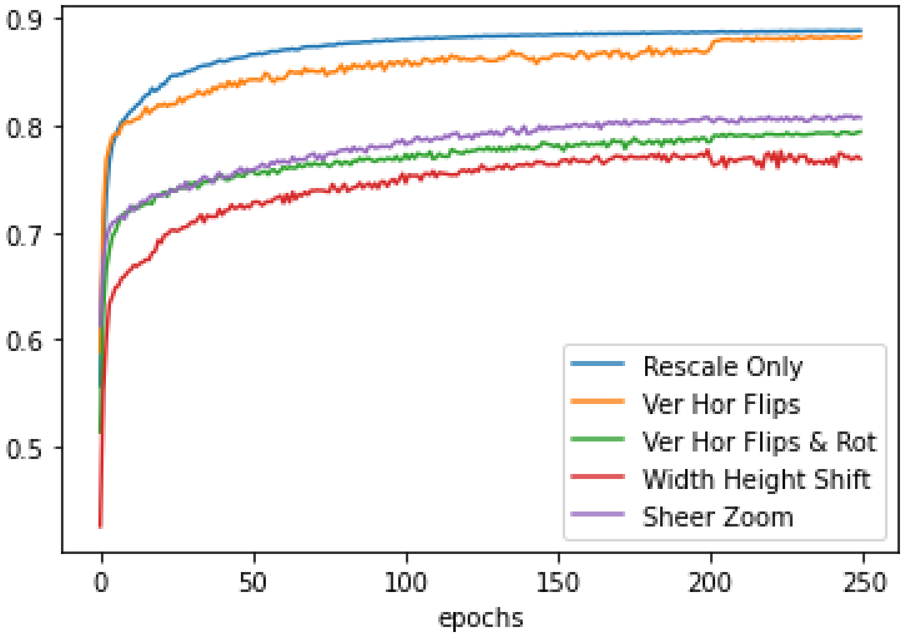



Rama et al. TIPCV (2019)

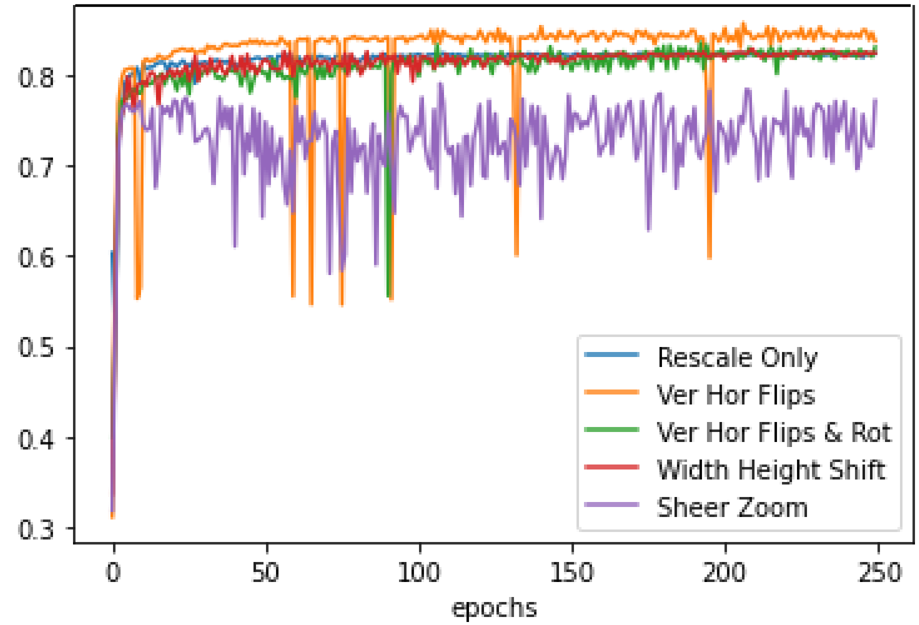
- Randomly perform on each batch of training data
- Help model learn features that are generalized

# Impact of data augmentation

Train accuracy




Validation accuracy




Source: towardsdatascience.com

- Too much augmentation → low training accuracy
- Proper augmentation → improved validation
  - Vertical + horizontal flip

# Augmentation in TF

```
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, dtype=None
)
```

```
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)
# 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),
          steps_per_epoch=len(x_train) / 32, epochs=epochs)
```

# Any question?