Deep Learning for Physics - DELPHYS 2025

IMAGE PROCESSING, NEURAL NETWORK AND DEEP LEARNING

Suraphong Yuma (MU)

OUTLINE

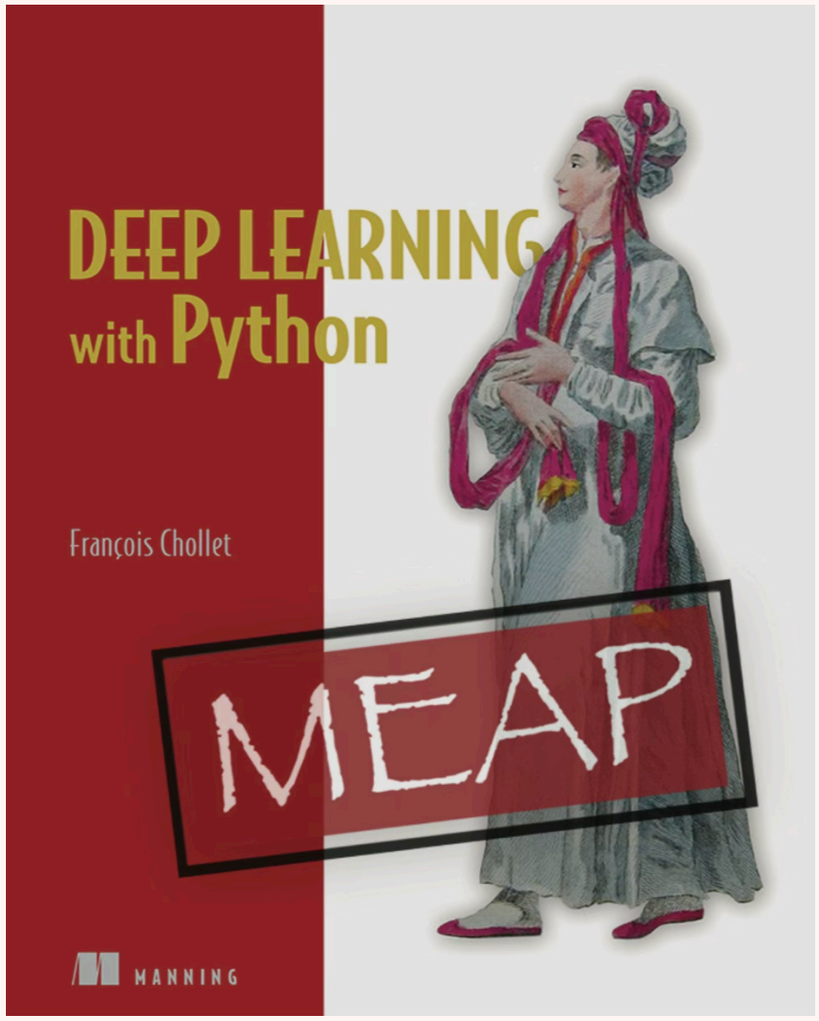
Image processing

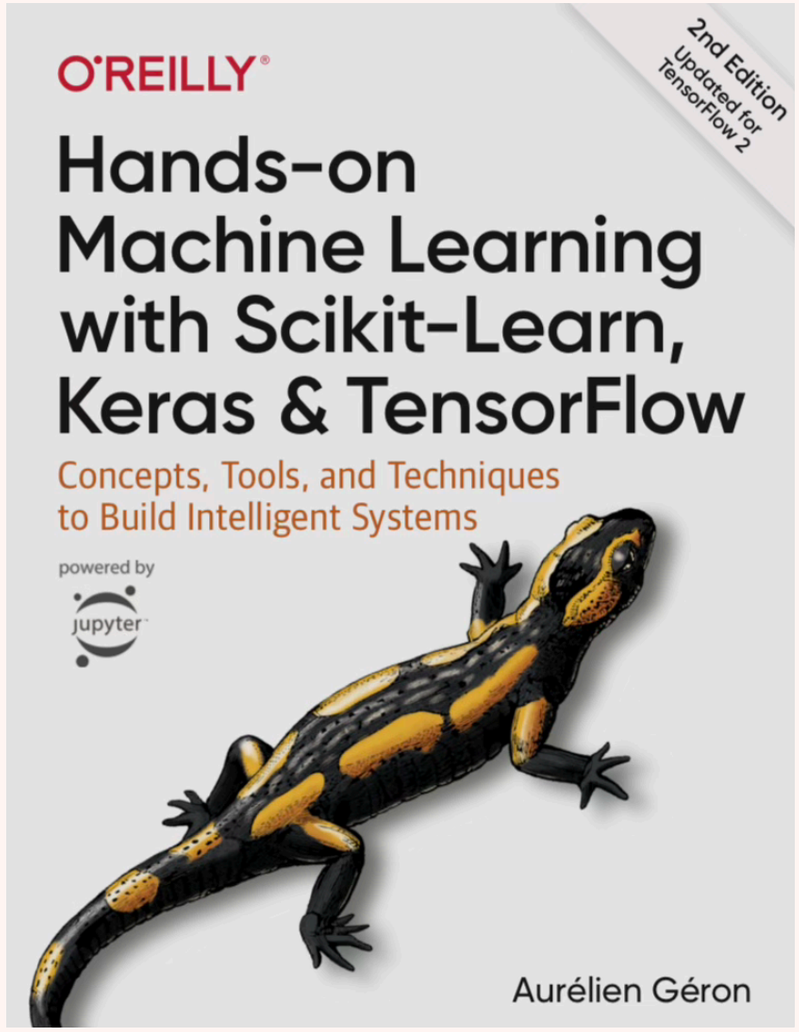
(Convolutional) Neural networks

Deep learning

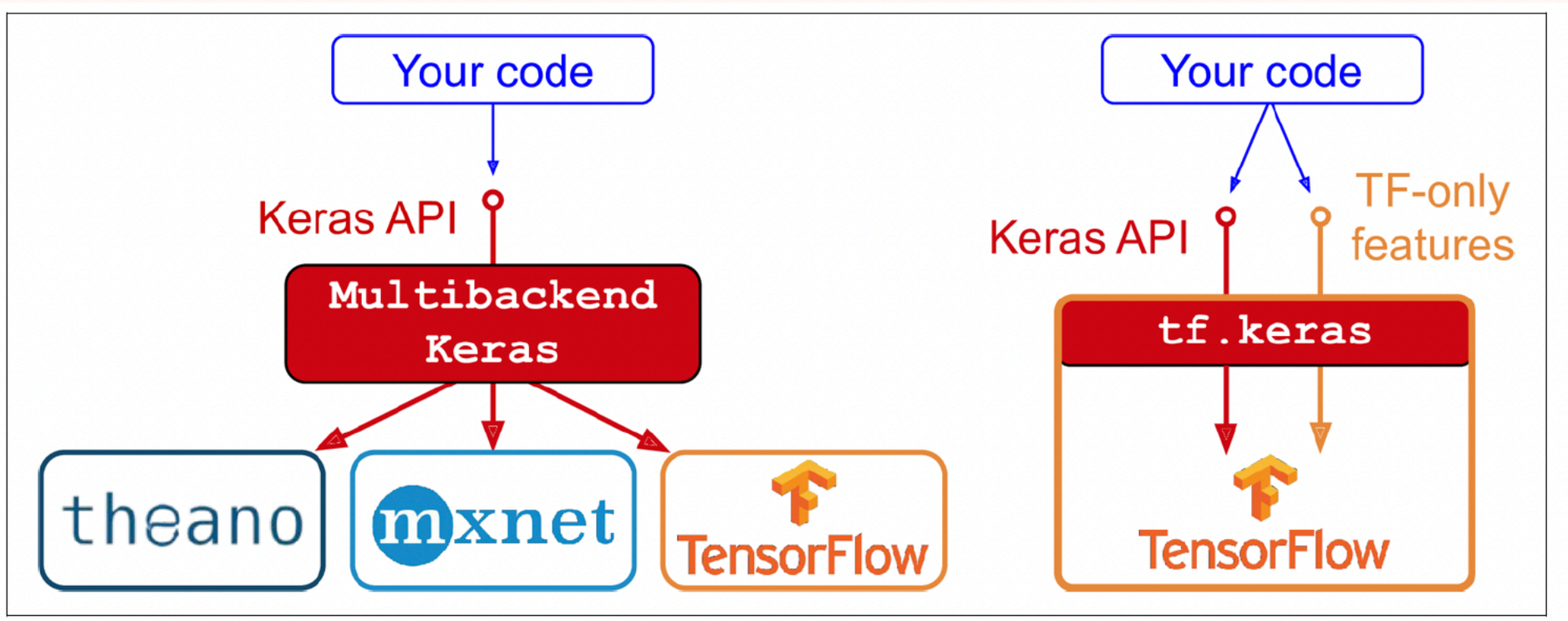
REFERENCED BOOKS

- **Deep learning with Python by Francois Chollet**
- **Hand-on machine learning with Scikit-Learn, Keras & Tensorflow by Aurélien Géron**





API: APPLICATION PROGRAMMING INTERFACE





MORE ADVANCED REFERENCE

Deep Learning in Astrophysics

Yuan-Sen Ting,[1,2]

[1]Department of Astronomy, The Ohio State University, Columbus, OH 43210, USA; email: ting.74@osu.edu
[2]Center for Cosmology and AstroParticle Physics (CCAPP), The Ohio State University, Columbus, OH 43210, USA

Annual Review of Astronomy and Astrophysics 2025. AA:1–57

https://doi.org/10.1146/

Copyright © 2025 by the author(s). All rights reserved

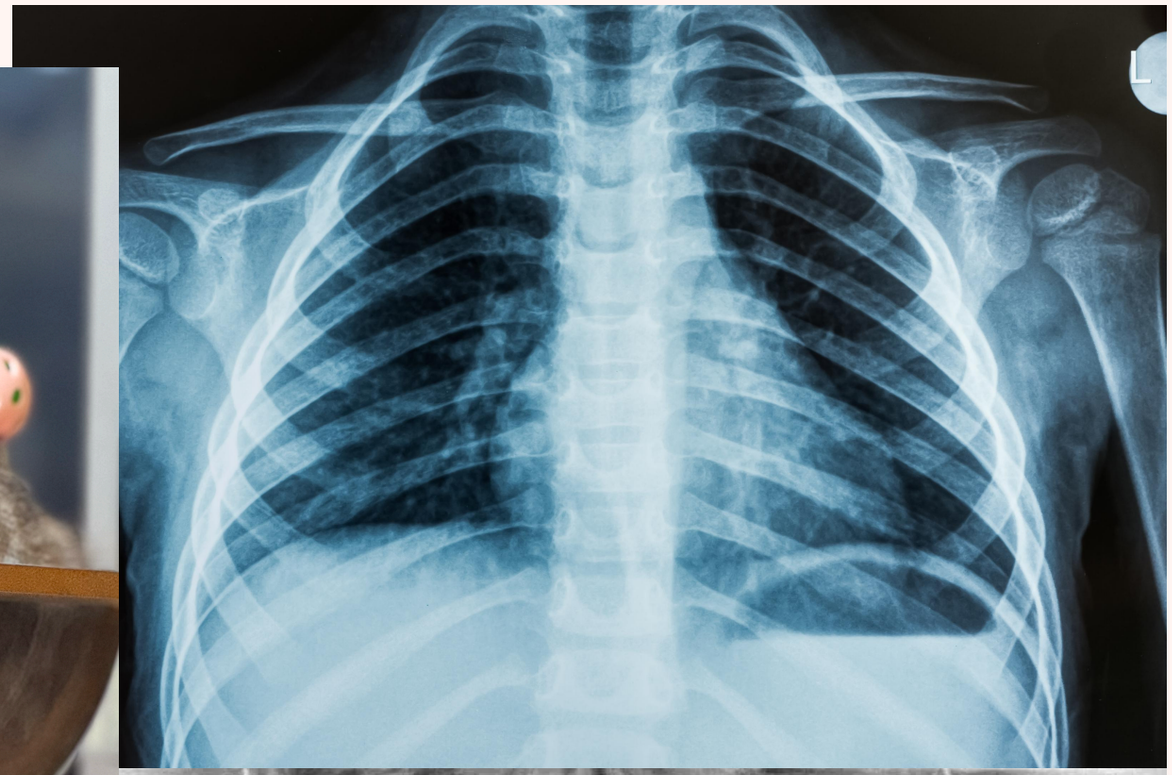
Keywords

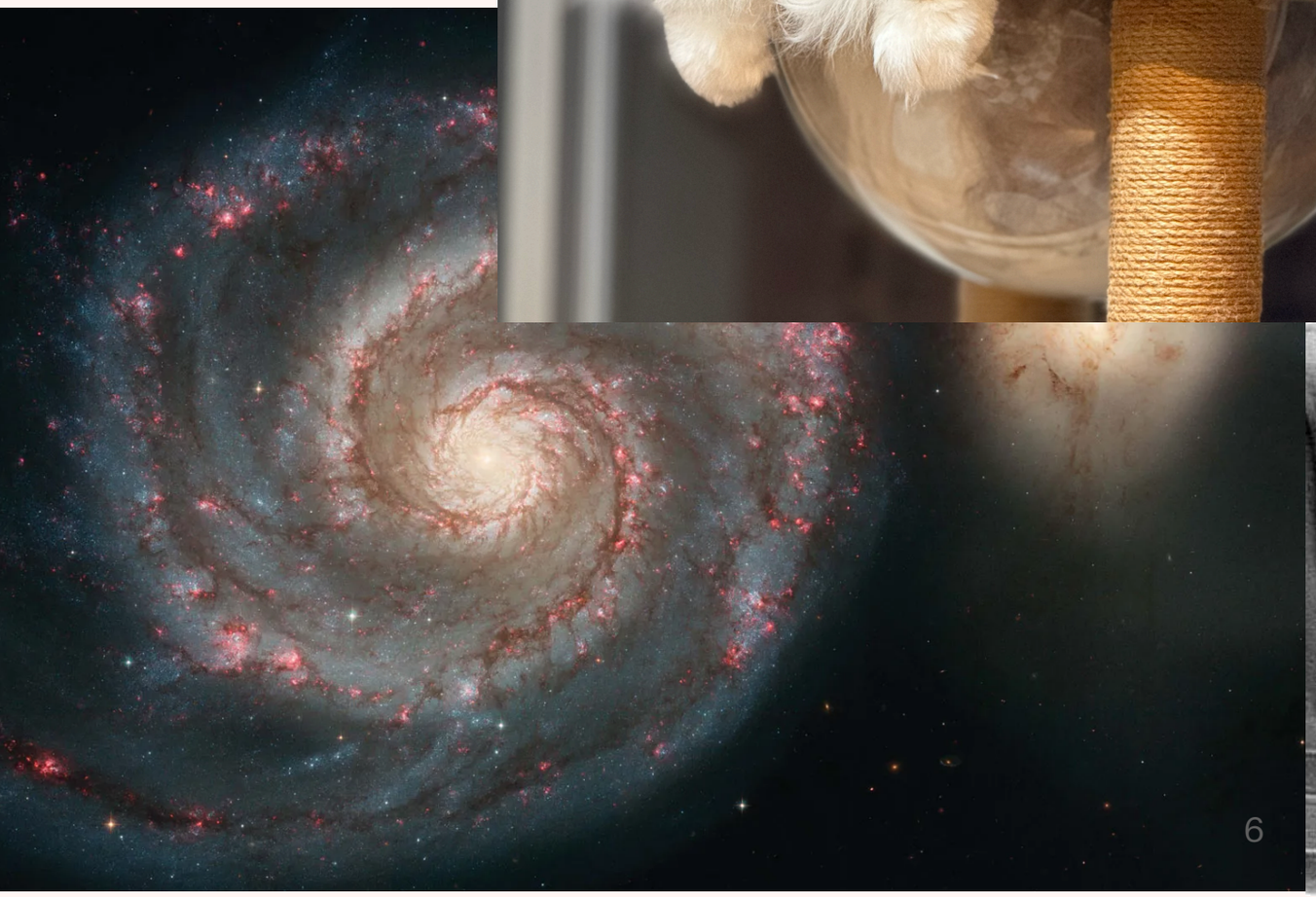
deep learning, inductive bias, physical symmetries, physics-informed neural networks, multi-scale modeling, simulation-based inference, anomaly detection, foundation models, reinforcement learning, large language models, astronomical surveys

Ting 2025, ARAA, (arXiv:2510.10713)

IMAGE > How different are they?





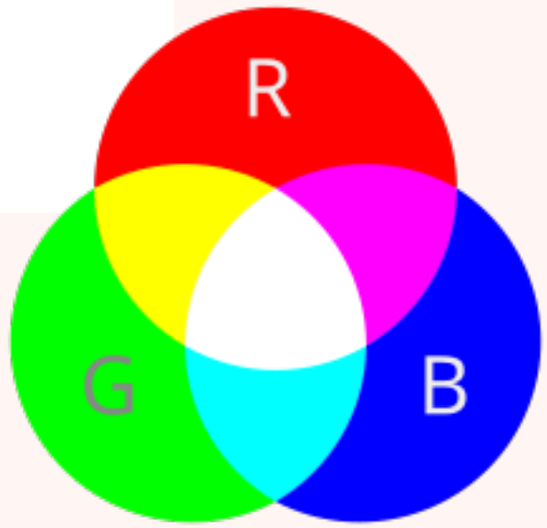




IMAGE

1. Raster Images (Bitmap Images)

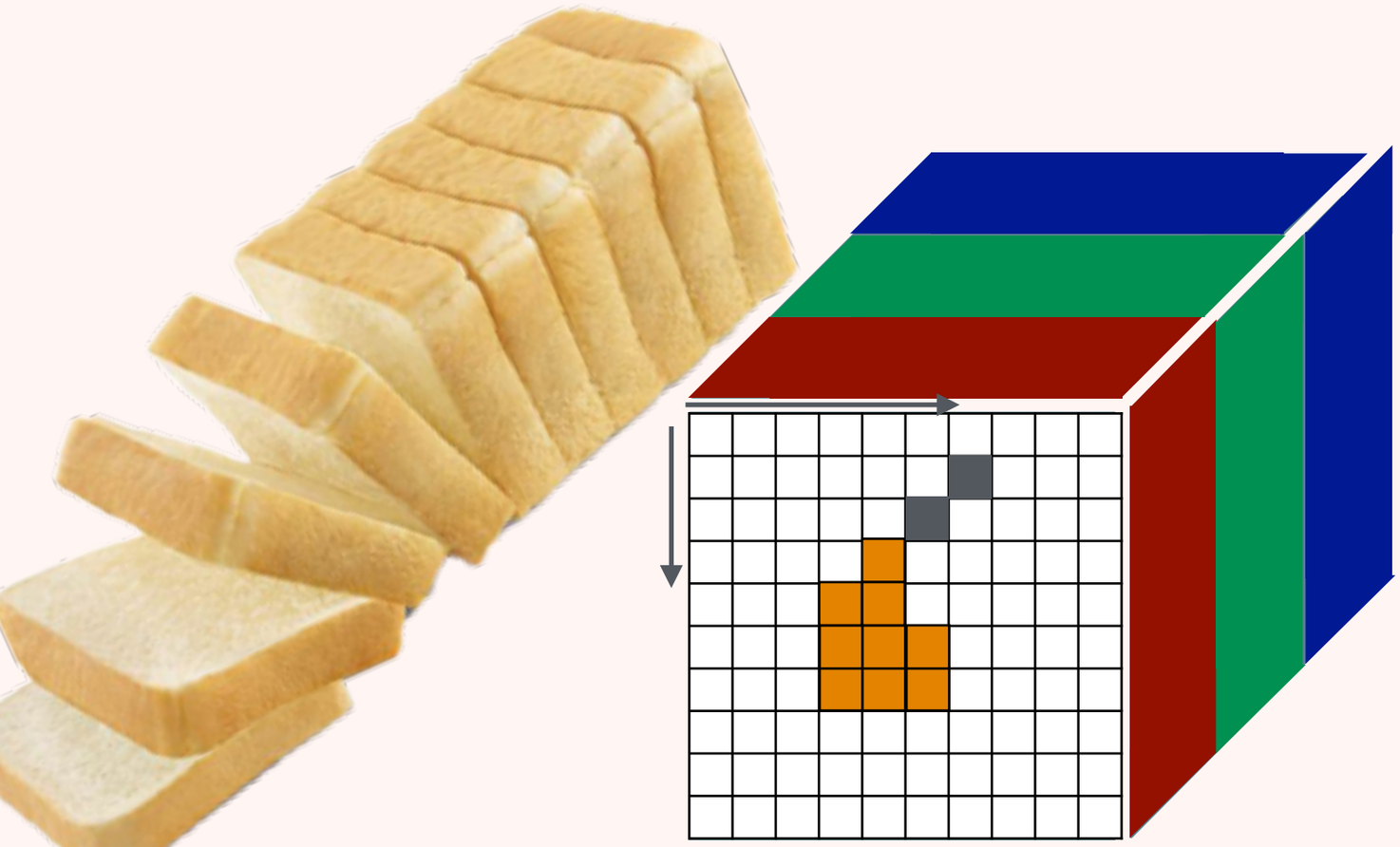
- Composed of a grid of pixels.
- Each pixel has color information stored in formats like RGB, RGBA, or grayscale.
- Resolution-dependent (scaling up can cause pixelation).
- Common formats: **JPEG, PNG, BMP, GIF, TIFF**.



2. Vector Images

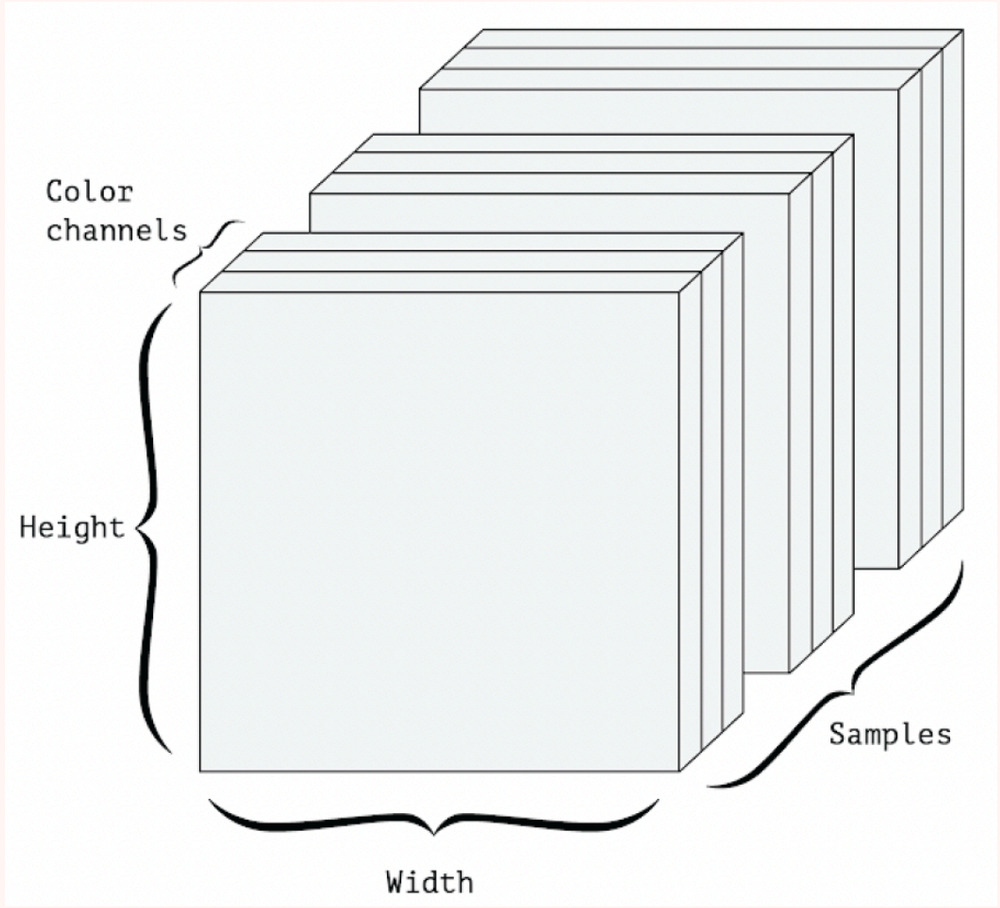
- Composed of mathematical formulas defining shapes, lines, and colors.
- Resolution-independent (scaling up does not cause pixelation).
- Best for logos, icons, and illustrations.
- Common formats: **SVG, EPS, PDF, AI**.

IMAGE



3D tensor for 1 image

(height, width, 3)



4D tensor for N images

(N, height, width, 3)

DEALING WITH IMAGE

OpenCV

Original

```
cv2_imshow(img)
```

Grayscale

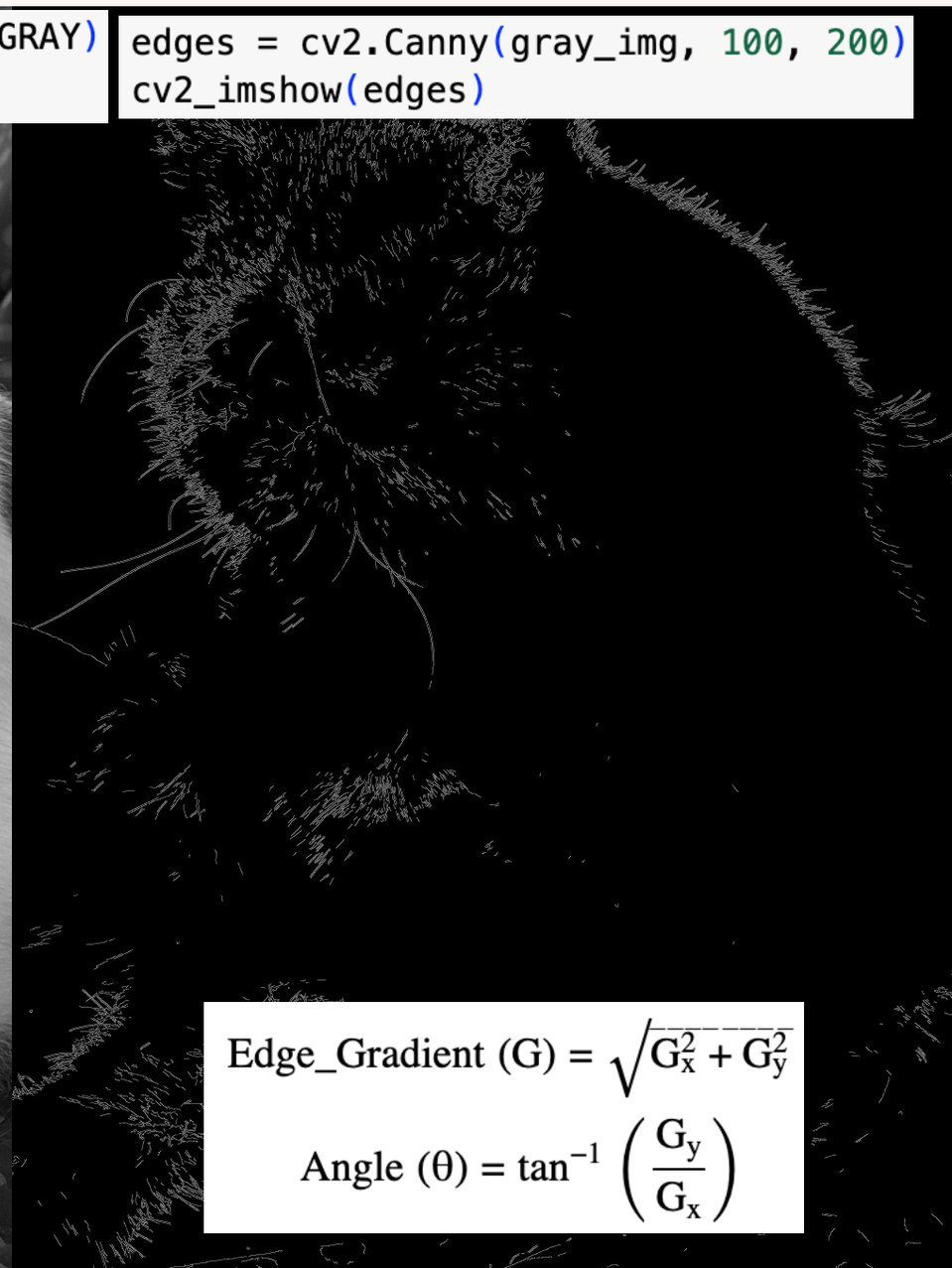
```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray_img)
```

Canny edge

```
edges = cv2.Canny(gray_img, 100, 200)
cv2_imshow(edges)
```





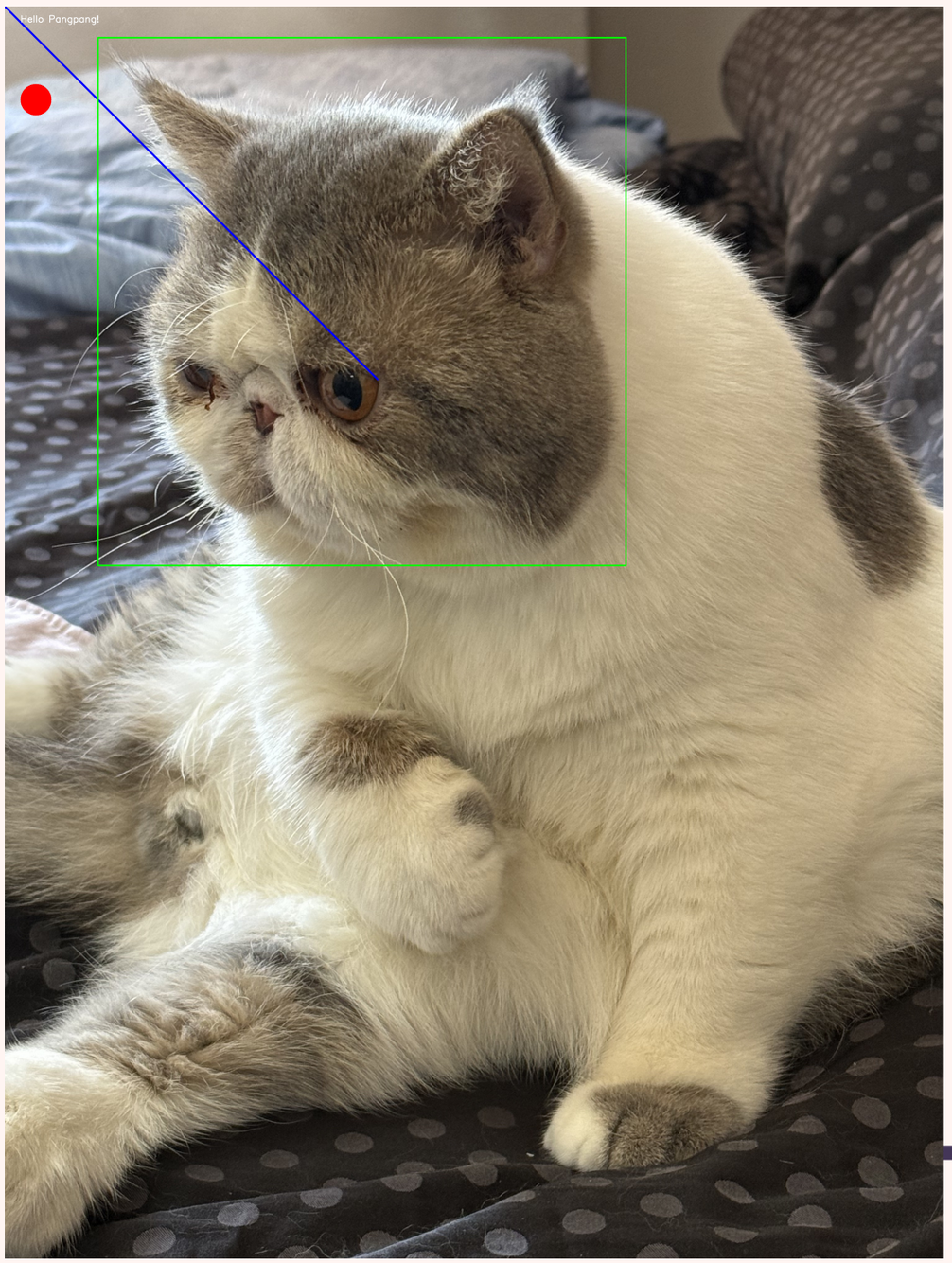


$$\text{Edge\_Gradient (G)} = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle } (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

DEALING WITH IMAGE

OpenCV



Resize



Blur



```
img_resized = cv2.resize(img, (200, 300))
cv2_imshow(img_resized)
```

```
img_blurred = cv2.GaussianBlur(img_resized, (5, 5), 0)
cv2_imshow(img_blurred)
```

ASTRONOMICAL IMAGE

FITS: Flexible Image Transport System

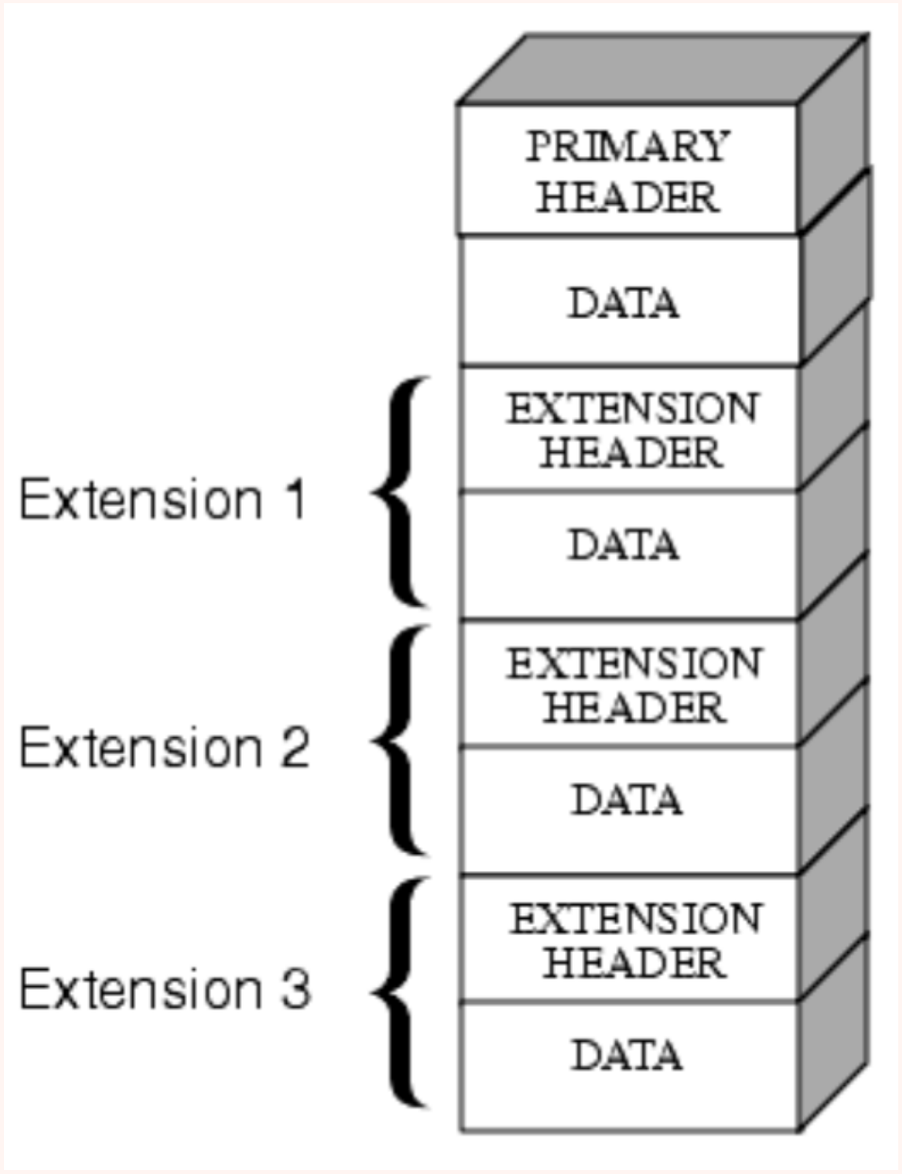
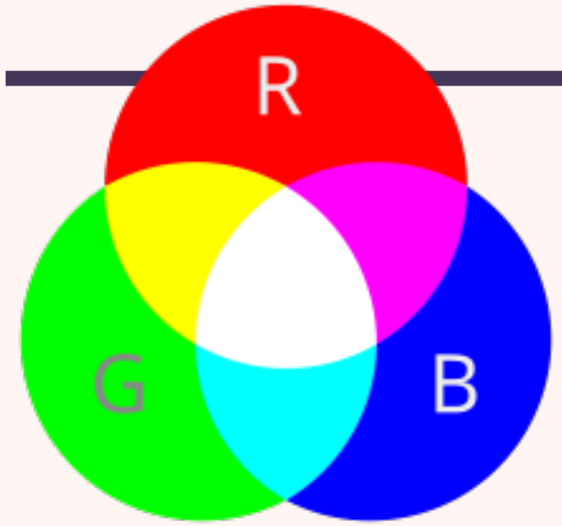


Image data

```
Filename: /Volumes/RGNS63SSD/JADES/images/img_goods-s/hlsp_jades_jwst_ni
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU     442   ()
  1  SCI           1 ImageHDU        83   (25000, 27700)   float32
  2  ERR           1 ImageHDU        83   (25000, 27700)   float32
  3  EXP           1 ImageHDU        83   (25000, 27700)   float32
  4  WHT           1 ImageHDU        83   (25000, 27700)   float32
```

Tabular data

```
Filename: /Volumes/RGNS63SSD/JADES/catalogs/hlsp_jades_jwst_nircam_good
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      13   ()
  1  FILTERS       1 BinTableHDU     33   23R x 12C   [6A, E, E, E, E,
  2  FLAG          1 BinTableHDU    161   94000R x 75C   [J, D, D, J, J
  3  SIZE          1 BinTableHDU    100   94000R x 43C   [J, D, D, E, D
  4  CIRC          1 BinTableHDU    856   94000R x 423C   [J, D, D, E,
  5  CIRC_BSUB     1 BinTableHDU    856   94000R x 423C   [J, D, D, E,
  6  CIRC_CONV     1 BinTableHDU    856   94000R x 423C   [J, D, D, E,
  7  KRON          1 BinTableHDU    376   94000R x 183C   [J, D, D, E,
  8  KRON_CONV     1 BinTableHDU    376   94000R x 183C   [J, D, D, E,
  9  PHOTOZ        1 BinTableHDU     29   94000R x 10C   [K, D, D, D, D
```



WHAT DIFFERENT?

Astronomical images

Red Green Blue

255

0

Grayscale

White

255

0

Black

Grayscale

White

$+\infty$

$-\infty$

Black

TURNING ASTRONOMICAL IMAGES TO RGB ONES

astropy
A Community Python Library for Astronomy

```python
import matplotlib.pyplot as plt
from astropy.visualization import make_lupton_rgb
from astropy.io import fits
from astropy.utils.data import get_pkg_data_filename

# Read in the three images downloaded from here:
g_name = get_pkg_data_filename('visualization/reprojected_sdss_g.fits.bz2')
r_name = get_pkg_data_filename('visualization/reprojected_sdss_r.fits.bz2')
i_name = get_pkg_data_filename('visualization/reprojected_sdss_i.fits.bz2')
g = fits.getdata(g_name)
r = fits.getdata(r_name)
i = fits.getdata(i_name)

rgb_default = make_lupton_rgb(i, r, g, filename="ngc6976-default.jpeg")
fig, ax = plt.subplots()
ax.imshow(rgb_default, origin='lower')
```

Preparing Red-Green-Blue Images from CCD Data

Robert Lupton,[1] Michael R. Blanton,[2] George Fekete,[3] David W. Hogg,[2] Wil O'Mullane,[3] Alex Szalay,[3] and Nicholas Wherry[2]

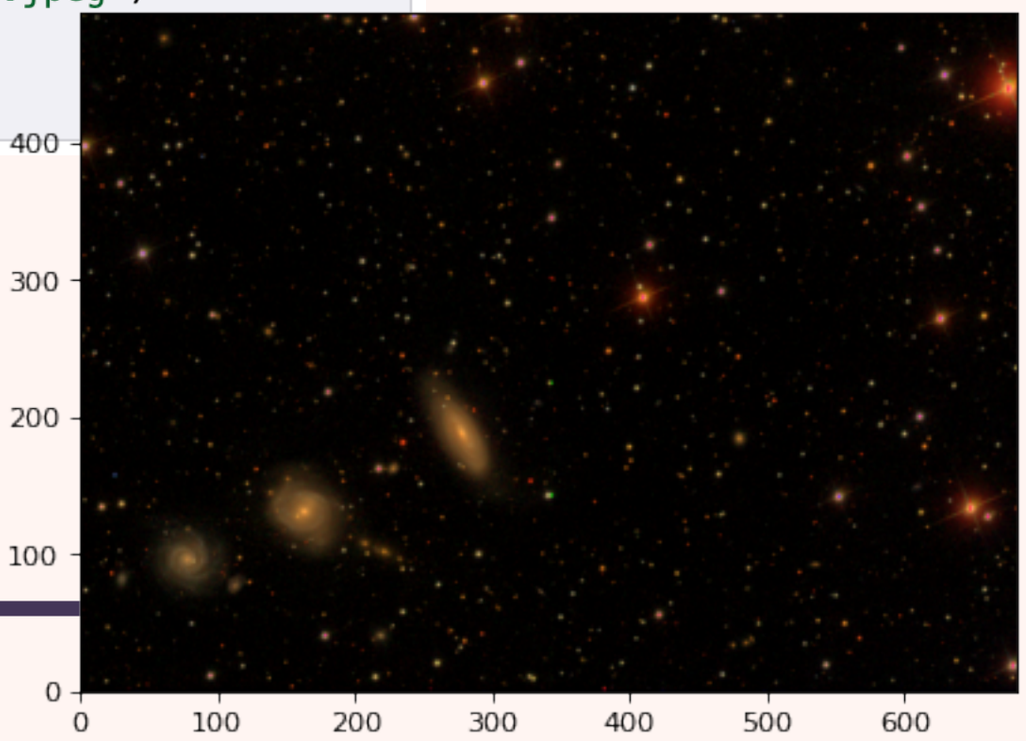
Received 2003 November 24; accepted 2003 December 24; published 2004 February 2

Publications of the Astronomical Society of the Pacific, **116**:133–137, 2004 February
© 2004. The Astronomical Society of the Pacific. All rights reserved. Printed in U.S.A.

$$\mathrm{R} = f(r),\ \mathrm{G} = f(g),\ \mathrm{B} = f(b), \tag{1}$$

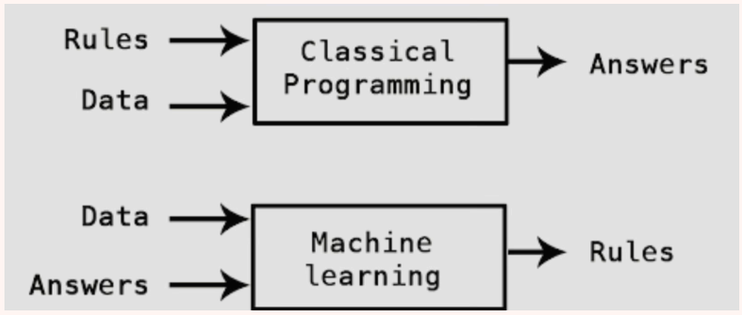
where

$$f(x) = \begin{cases} 0, & x < m, \\ F(x-m)/F(M-m), & m \le x \le M, \\ 1, & M < x, \end{cases}$$

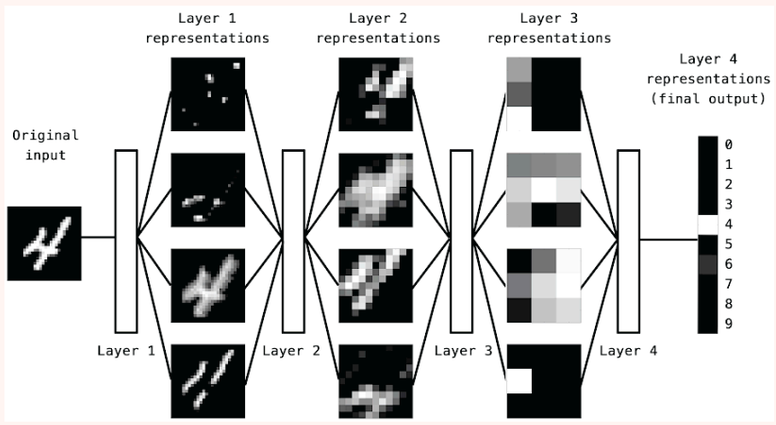


WHAT IS DEEP LEARNING?

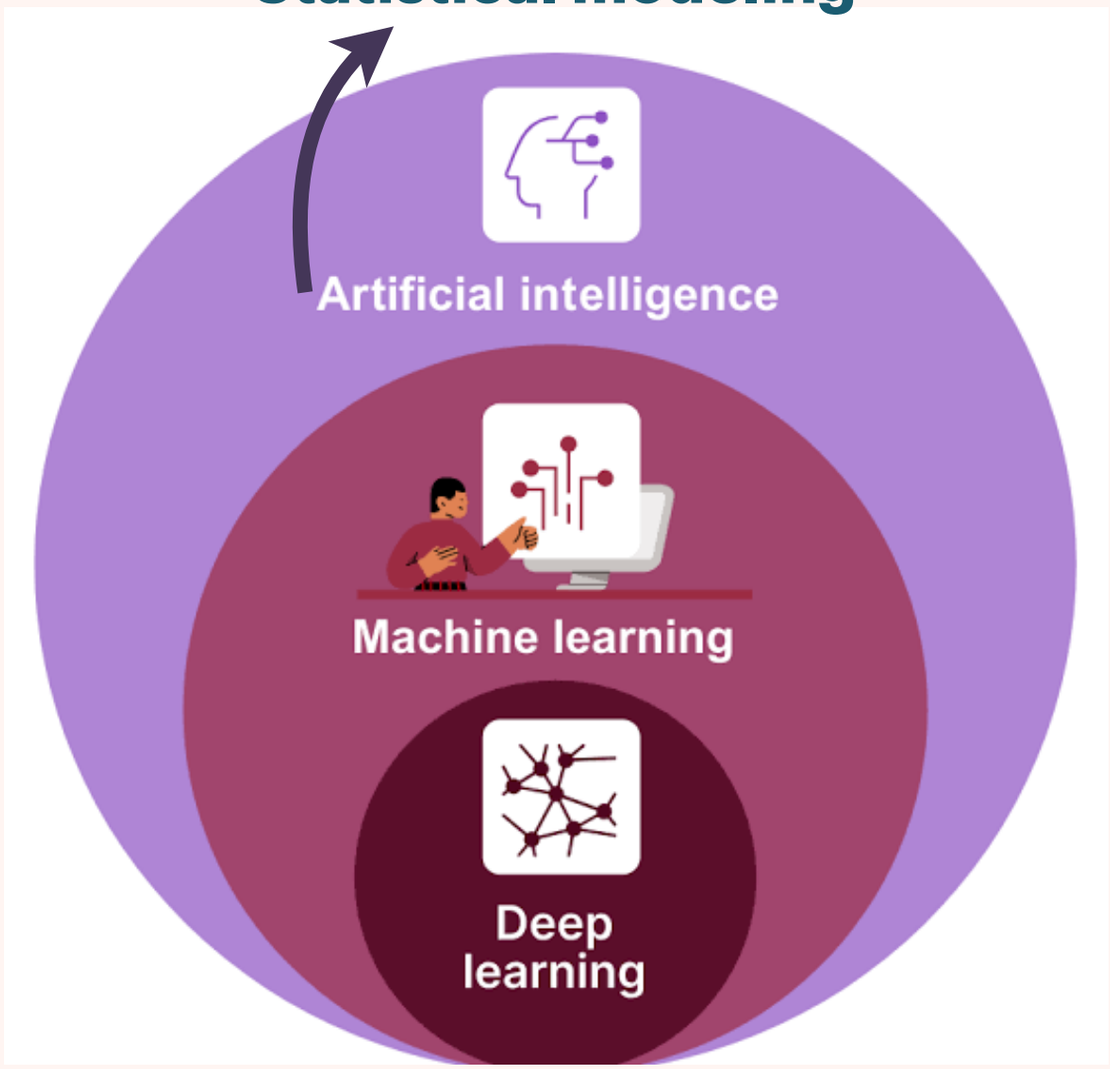
- **AI**
  - **“The effort to automate intellectual tasks that are usually done by humans.”**
- **Machine Learning**



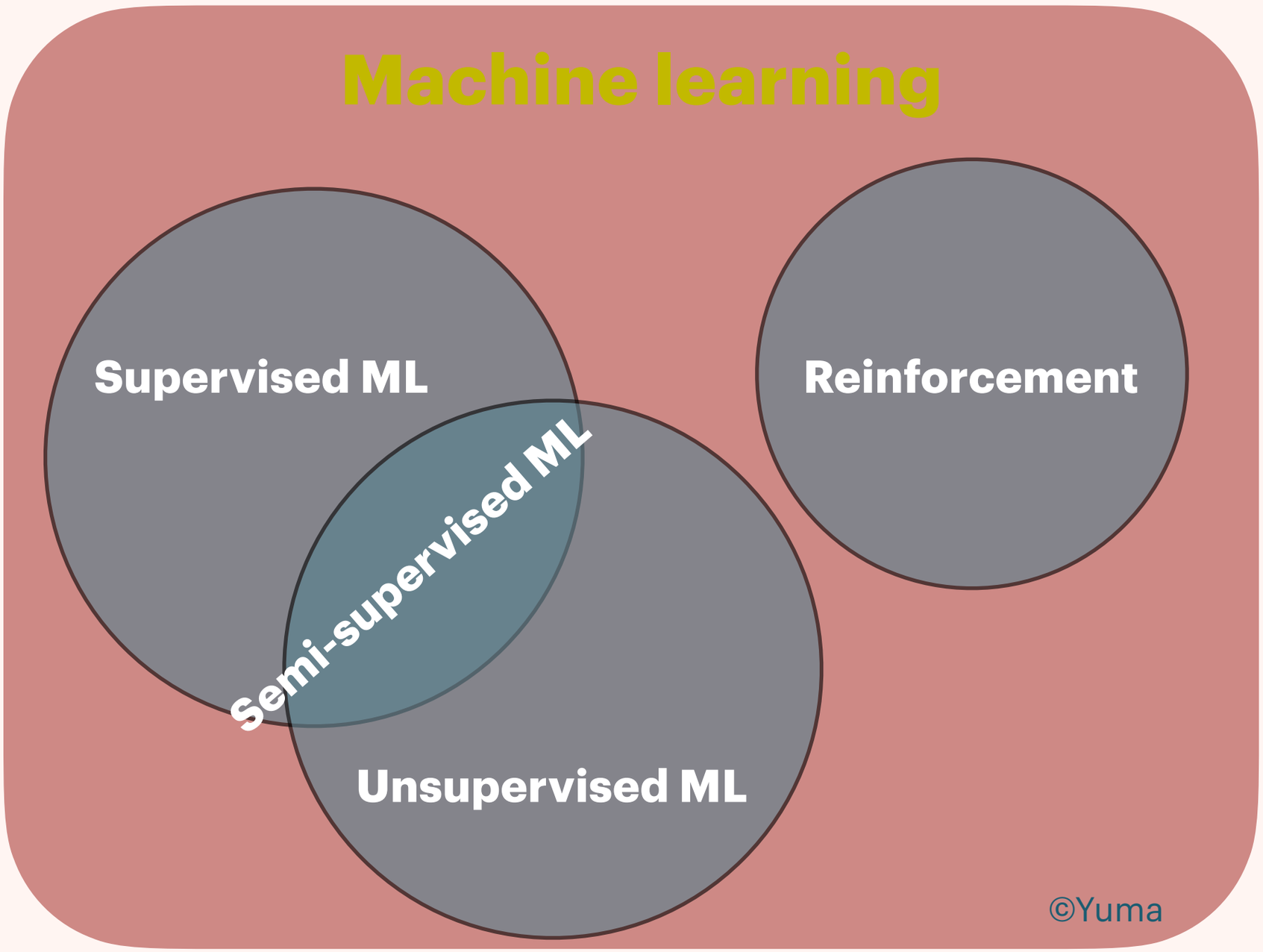
- **Deep Learning**



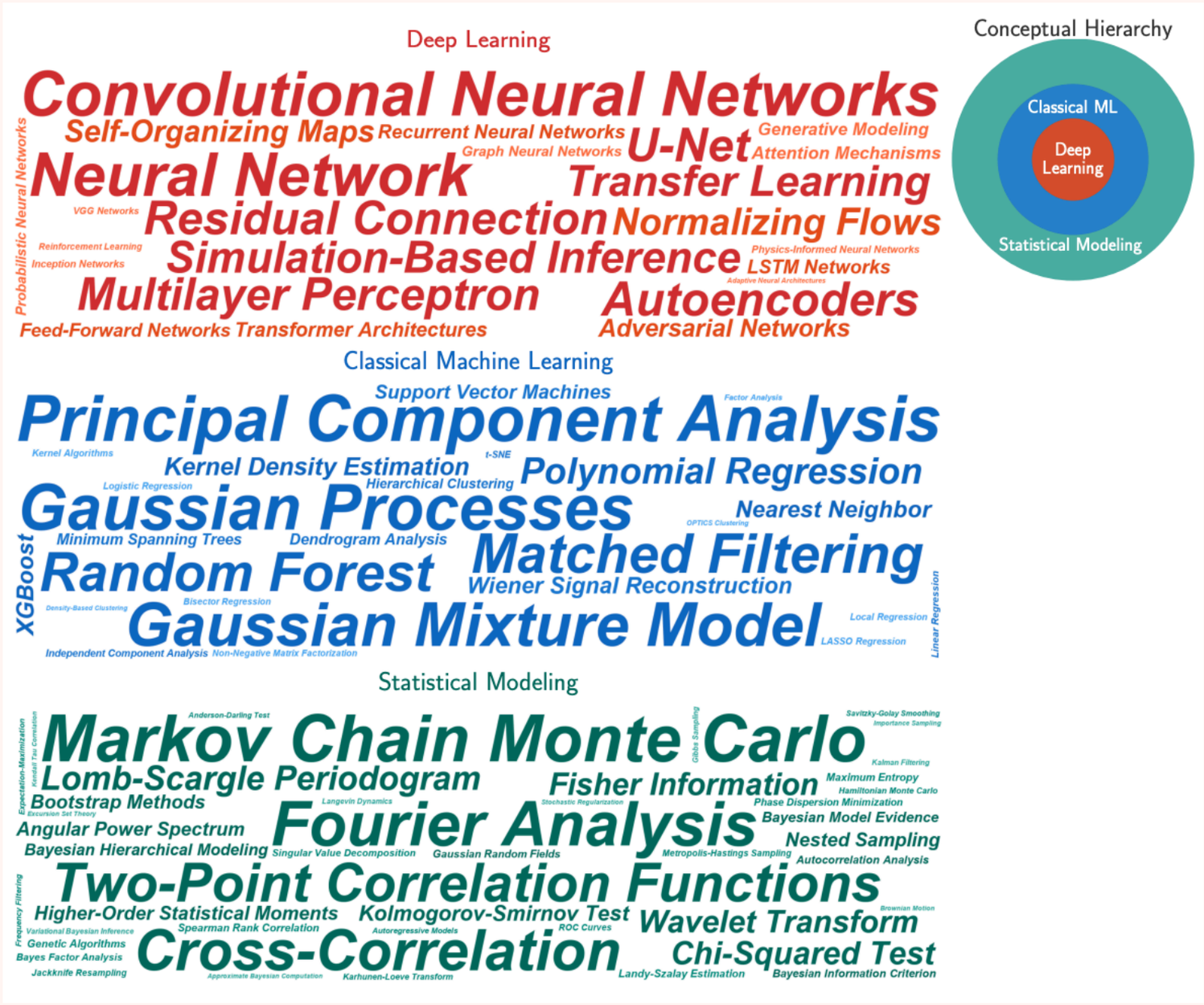
Statistical modeling



WHAT IS DEEP LEARNING?

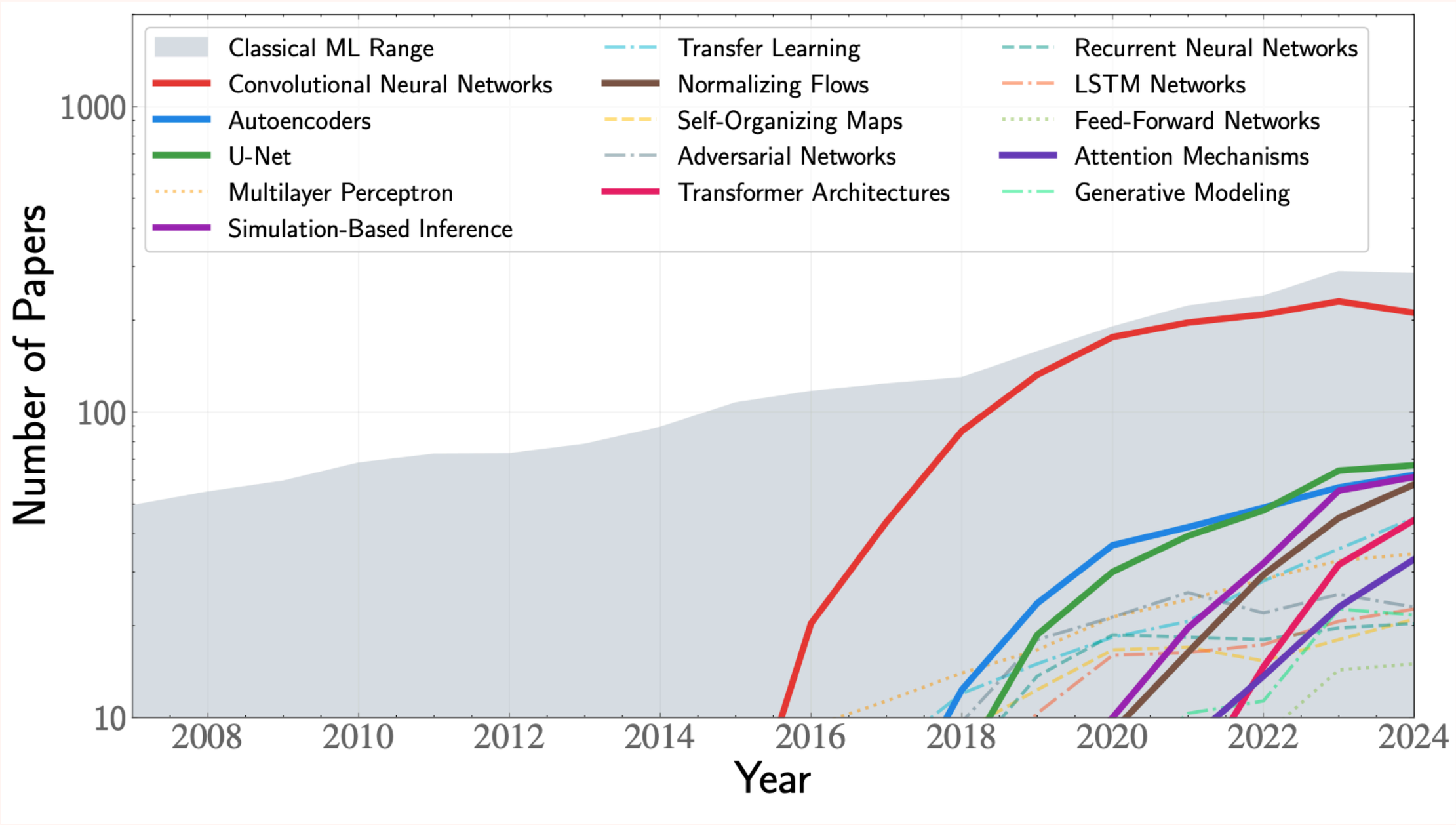


DL IN ASTROPHYSICS



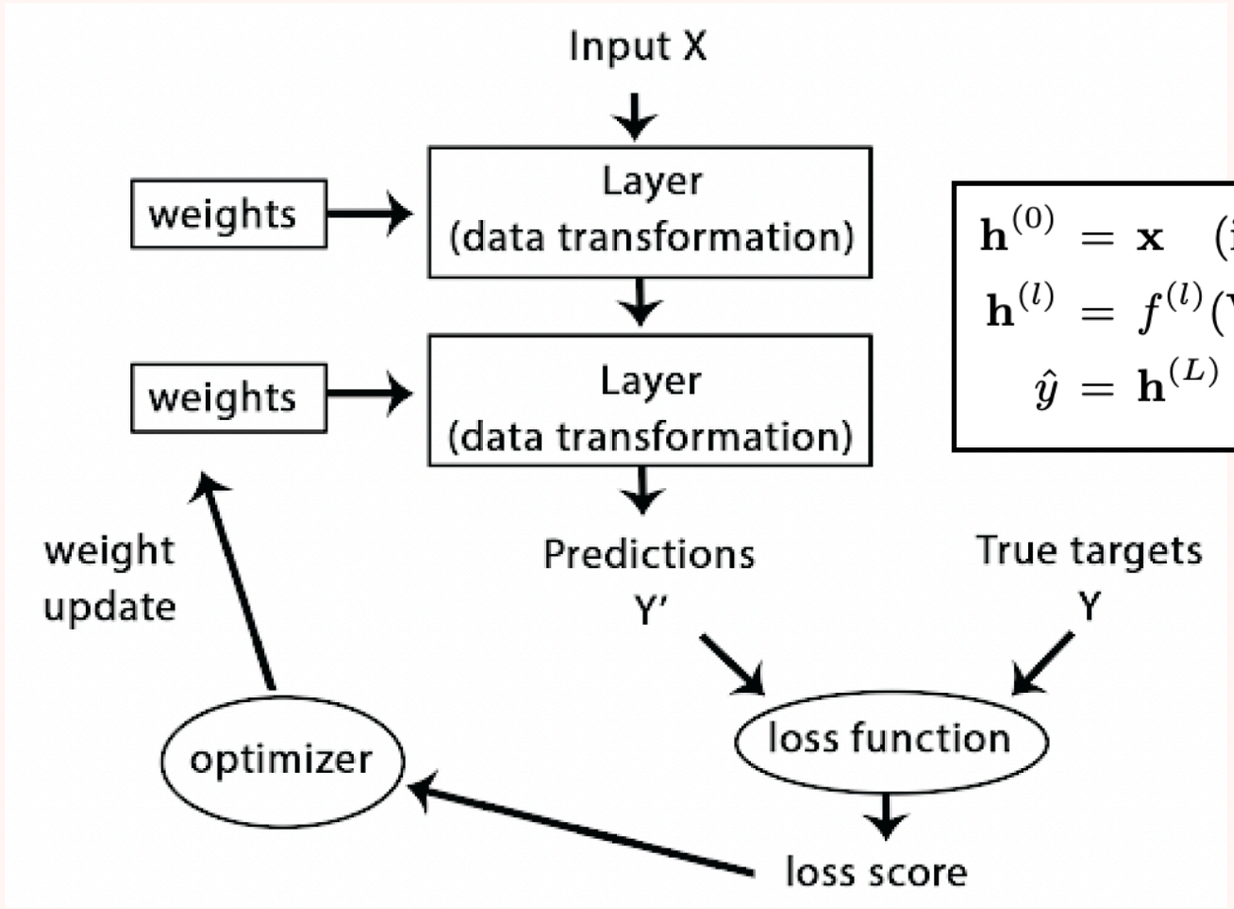
Ting 2025, ARAA, (arXiv:2510.10713)

DL IN ASTROPHYSICS



Ting 2025, ARAA, (arXiv:2510.10713)

HOW DL WORKS



Challot 2021, DL with Python

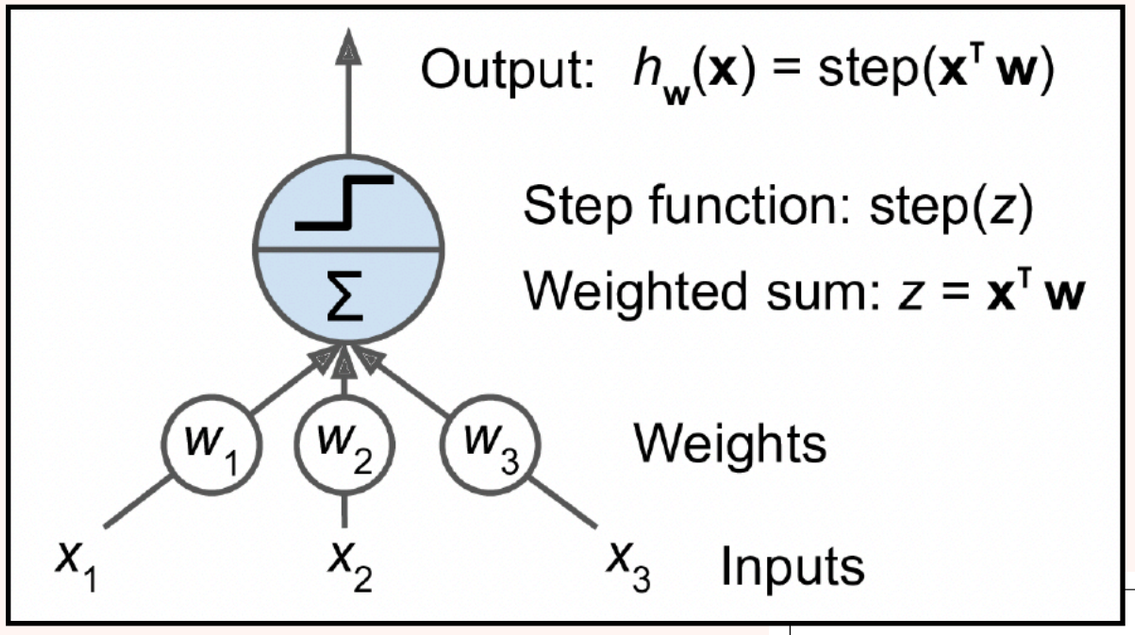
$$
\begin{aligned}
\mathbf{h}^{(0)} &= \mathbf{x} \quad \text{(input layer)} && 1. \\
\mathbf{h}^{(l)} &= f^{(l)}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}), \quad l = 1, \ldots, L && 2. \\
\hat{y} &= \mathbf{h}^{(L)} \quad \text{(output layer)} && 3.
\end{aligned}
$$

Ting 2025, ARAA, (arXiv:2510.10713)

3 compilation steps:

1. Optimizer
2. Loss function
3. Metrics

ARTIFICIAL NEURAL NETWORK



Deep neural networks

Desired output

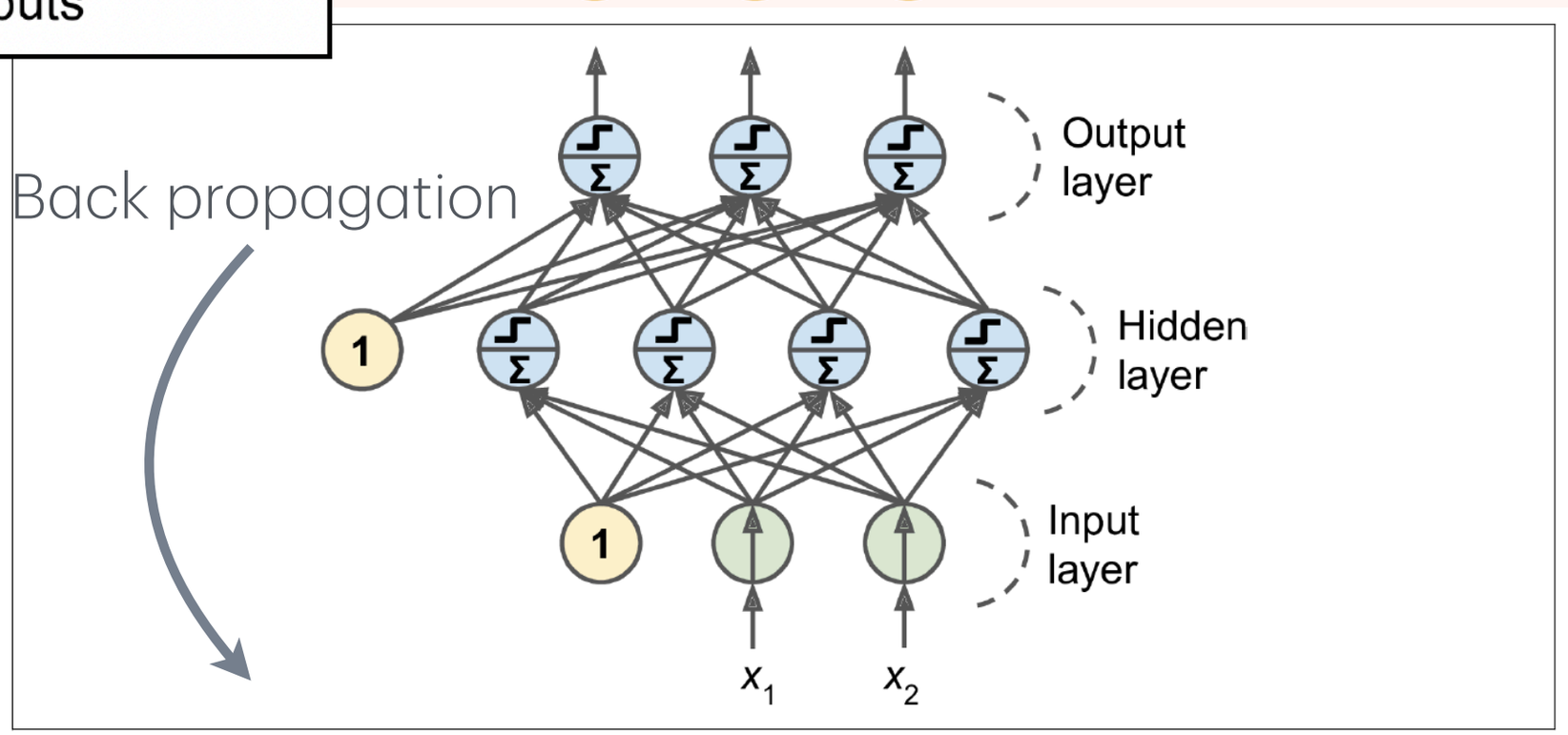
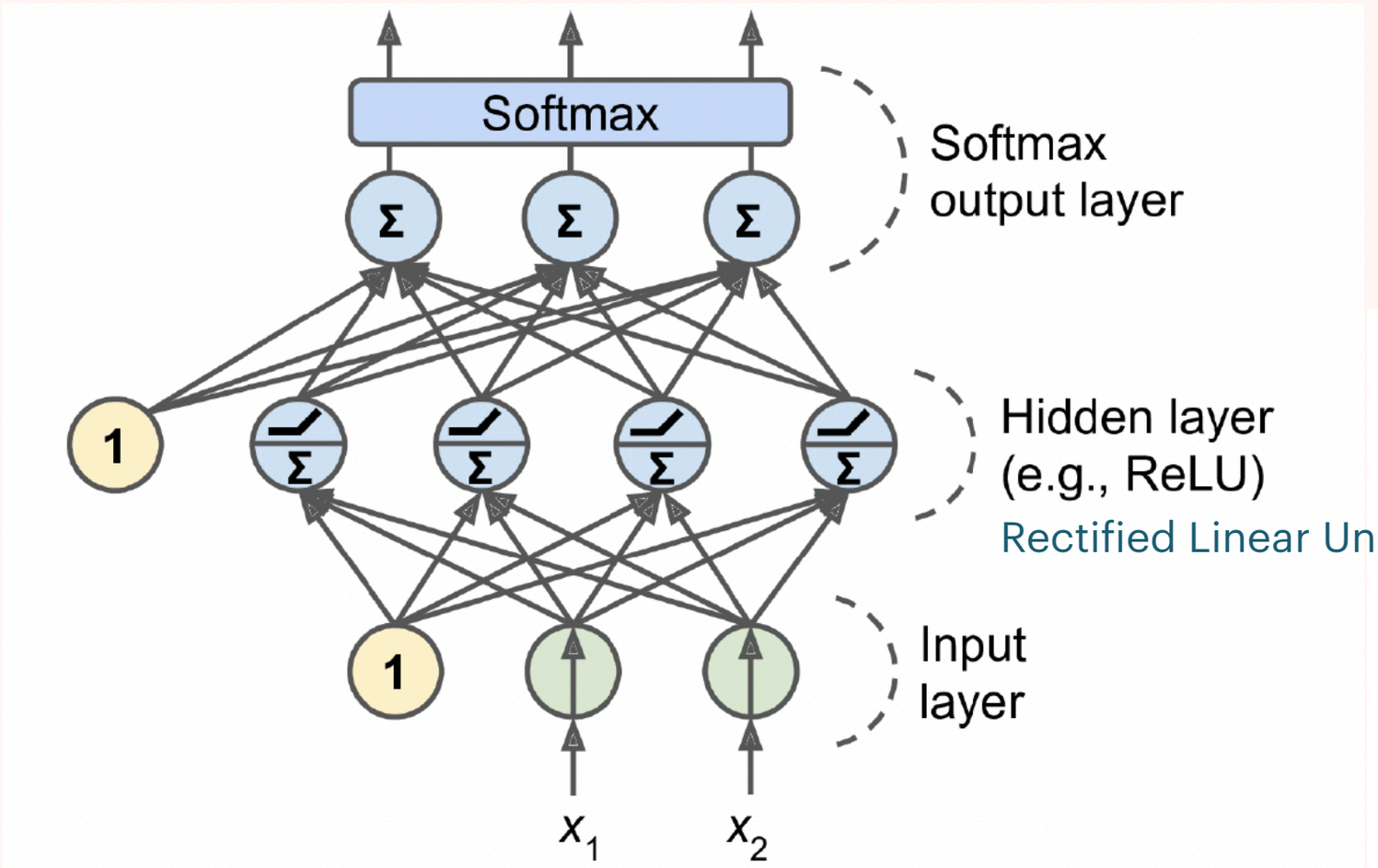


Figure 10-7. Architecture of a Multilayer Perceptron with two inputs, one hidden layer of four neurons, and three output neurons (the bias neurons are shown here, but usually they are implicit)

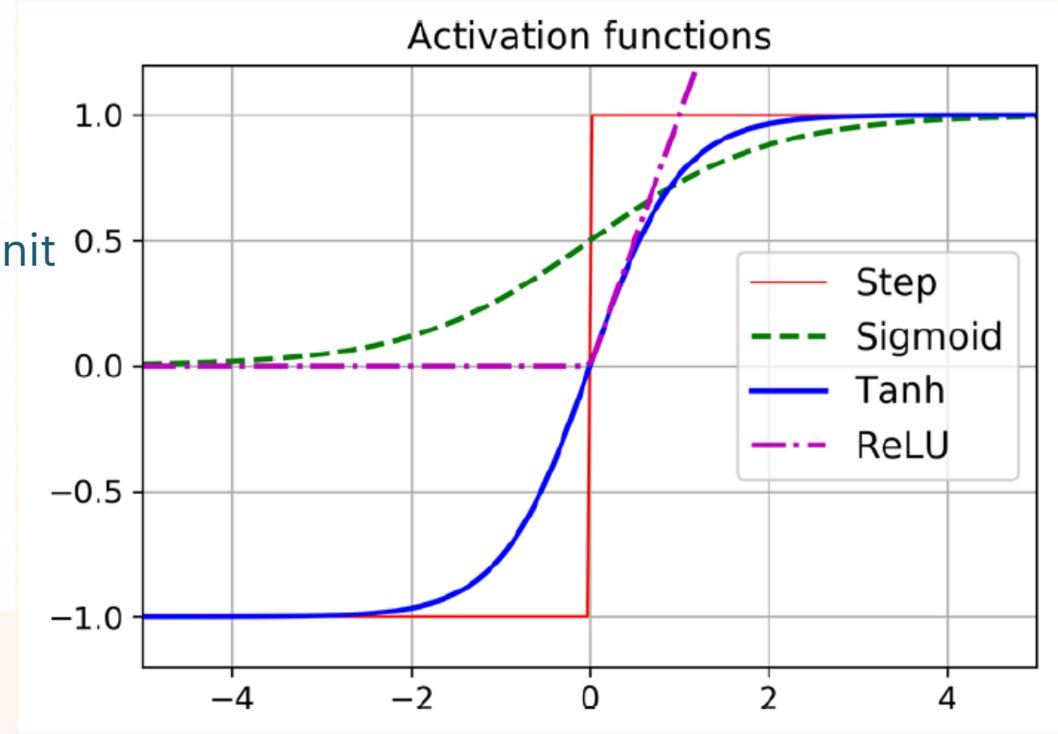
CLASSIFICATION WITH ANN



Softmax function

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}$$

Rectified Linear Unit



A FIRST LOOK AT NEURAL NETWORK

- **Classifying digits from MNIST data ->** Example_DL_MINSTdata.ipynb

Loading the MNIST data

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Exploring the training data

```
train_images.shape
```

```
(60000, 28, 28)
```

```
train_labels
#train_labels.shape
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
##transform into 2D tensor or 1D array
train_images = train_images.reshape((60000, 28 * 28))
## Normalizing the pixel values
train_images = train_images.astype('float32') / 255

## Do the same for test set
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

A FIRST LOOK AT NEURAL NETWORK

Classifying digits from MNIST data -> .ipynb file

Simple neural network architecture

```
from keras import models
from keras import layers
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

```
output = relu(dot(input, W) + b)
```

The compilation step

```
network.compile(optimizer='rmsprop',
  loss='categorical_crossentropy',
  metrics=['accuracy'])
```

$$L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

For each Parameter w^j

(j subscript dropped for clarity)

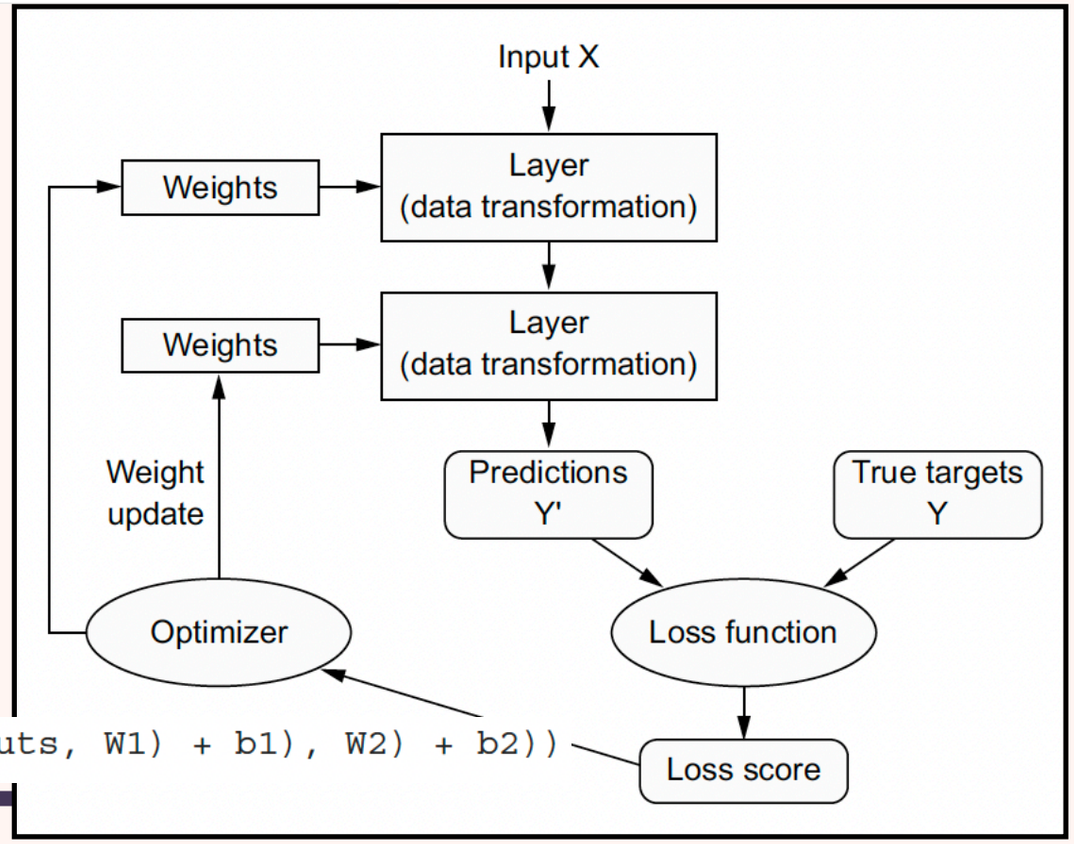
RMSProp

$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$

$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*
ν_t : *Exponential Average of squares of gradients*
g_t : *Gradient at time t along* ω^j




```
loss_value = loss(y_true, softmax(dot(relu(dot(inputs, W1) + b1), W2) + b2))
```

A FIRST LOOK AT NEURAL NETWORK

Training and evaluating the model

Preparing the labels

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
## Training the network
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.1660 - loss: 2.2803
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.5194 - loss: 1.8759
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7209 - loss: 1.1669
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7978 - loss: 0.8085
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.8306 - loss: 0.6360
<keras.src.callbacks.history.History at 0x7ef4e8ed8f80>
```

```
# Evaluating the network
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8301 - loss: 0.6048
test_acc: 0.8518000245094299
```

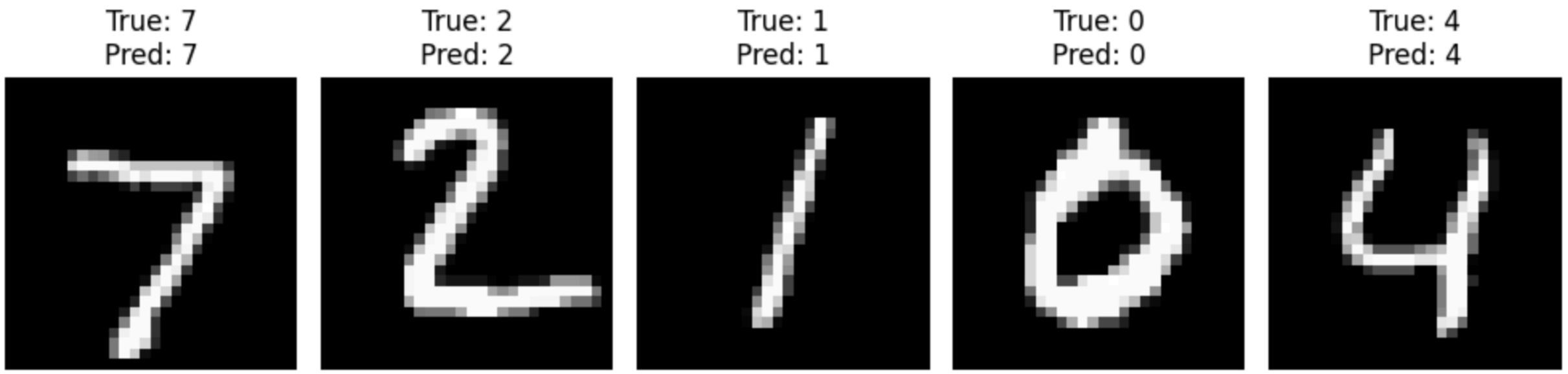
USING TRAINED MODEL TO PREDICT

```python
import matplotlib.pyplot as plt
import numpy as np

# Make predictions on the test images
predictions = network.predict(test_images)

# Select a few images to display
num_images = 5
plt.figure(figsize=(10, 5))
for i in range(num_images):
    plt.subplot(1, num_images, i + 1)
    plt.imshow(test_images[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {np.argmax(test_labels[i])}\nPred: {np.argmax(predictions[i])}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
```



FAMOUS NN FOR IMAGE

CNN: Convolutional Neural Network

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 128)	73,856
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 10)	11,530

Total params: 104,202 (407.04 KB)
Trainable params: 104,202 (407.04 KB)
Non-trainable params: 0 (0.00 B)

Conv2D: Feature map

(Height, width, channels)

bias

((3*3*1)+1)*32 =320

((3*3*32)+1)*64 =18,496

bias

(1152*10)+(1*10)=11,530

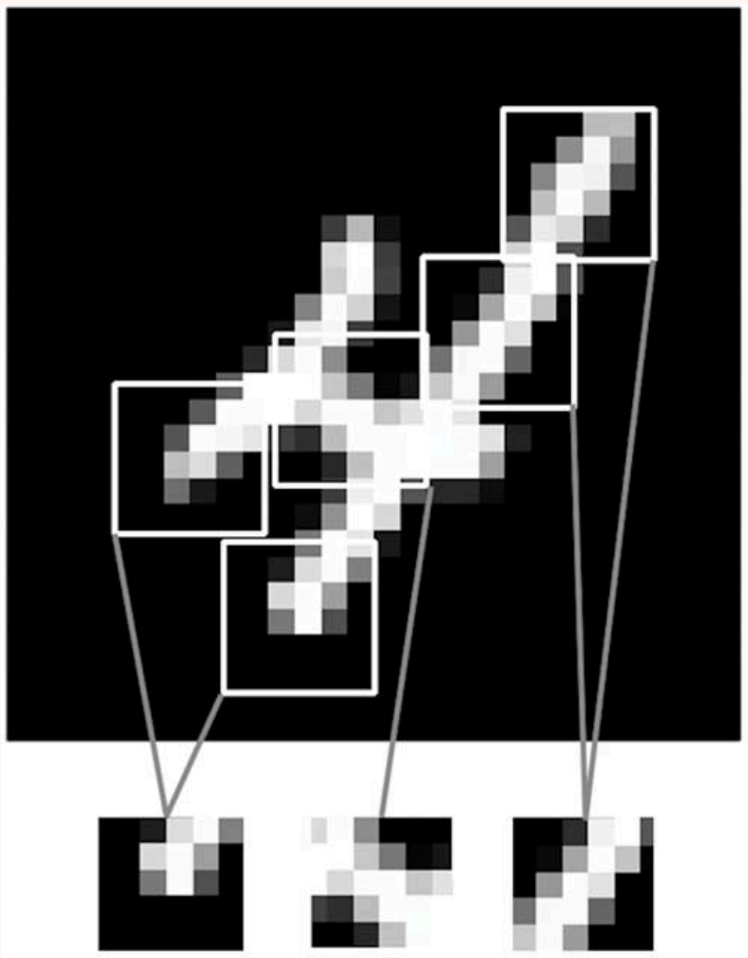
FAMOUS NN FOR IMAGE

CNN: Convolutional Neural Network

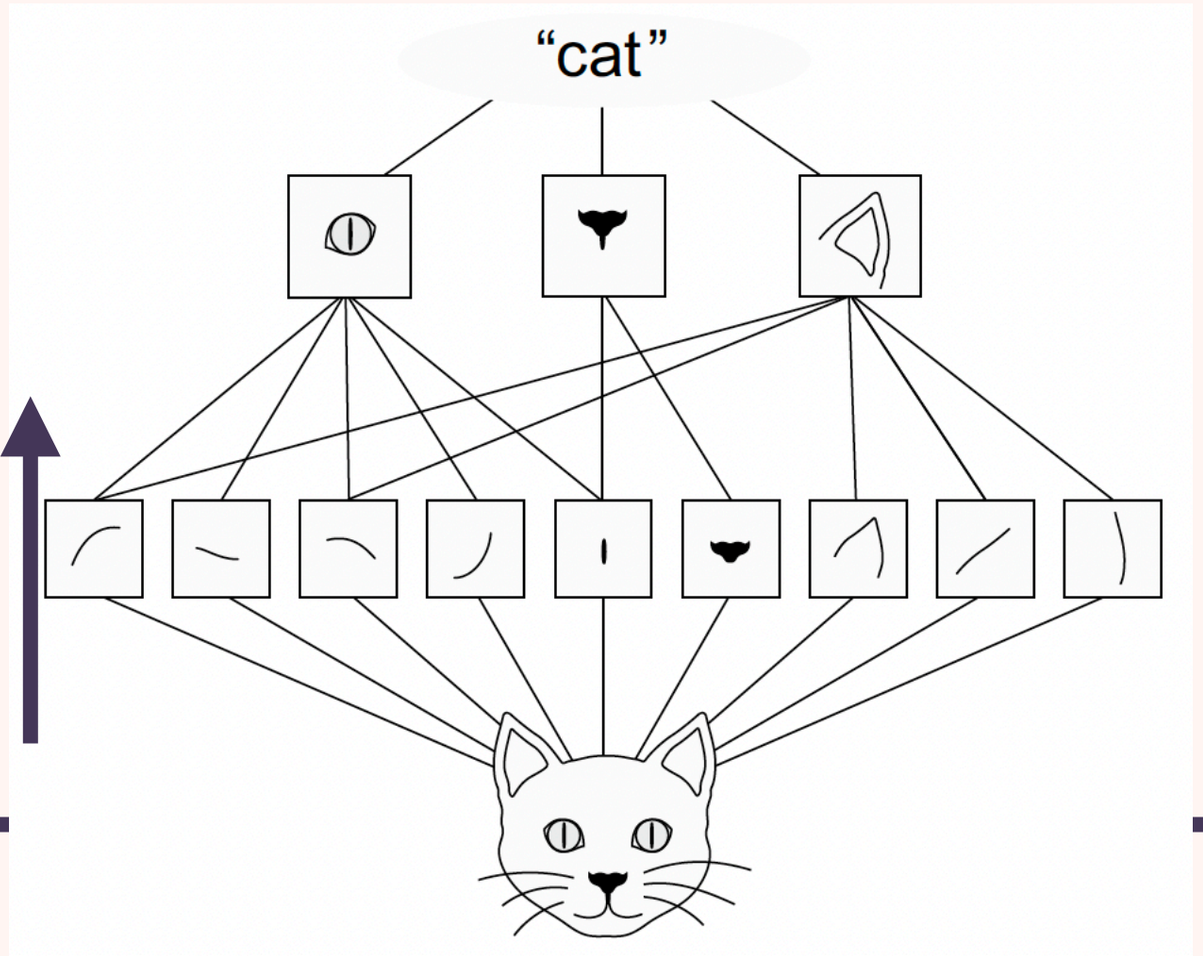
```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
model.compile(optimizer="rmsprop",
loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Epoch 1/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 56s 59ms/step - accuracy: 0.8853 - loss: 0.3567
Epoch 2/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 80s 57ms/step - accuracy: 0.9852 - loss: 0.0465
Epoch 3/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 79s 54ms/step - accuracy: 0.9909 - loss: 0.0299
Epoch 4/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 81s 54ms/step - accuracy: 0.9927 - loss: 0.0222
Epoch 5/5
938/938 ━━━━━━━━━━━━━━━━━━━━ 49s 53ms/step - accuracy: 0.9945 - loss: 0.0178
<keras.src.callbacks.history.History at 0x7b0c8a39f770>
```

CONVOLUTION OPERATION



- **Global (Dense) vs local (Convnet) pattern**
- **2 interesting properties of Convnet**
  1. The pattern they learn is translational invariant
     - Fewer training images
  2. They can learn spatial hierarchies of patterns



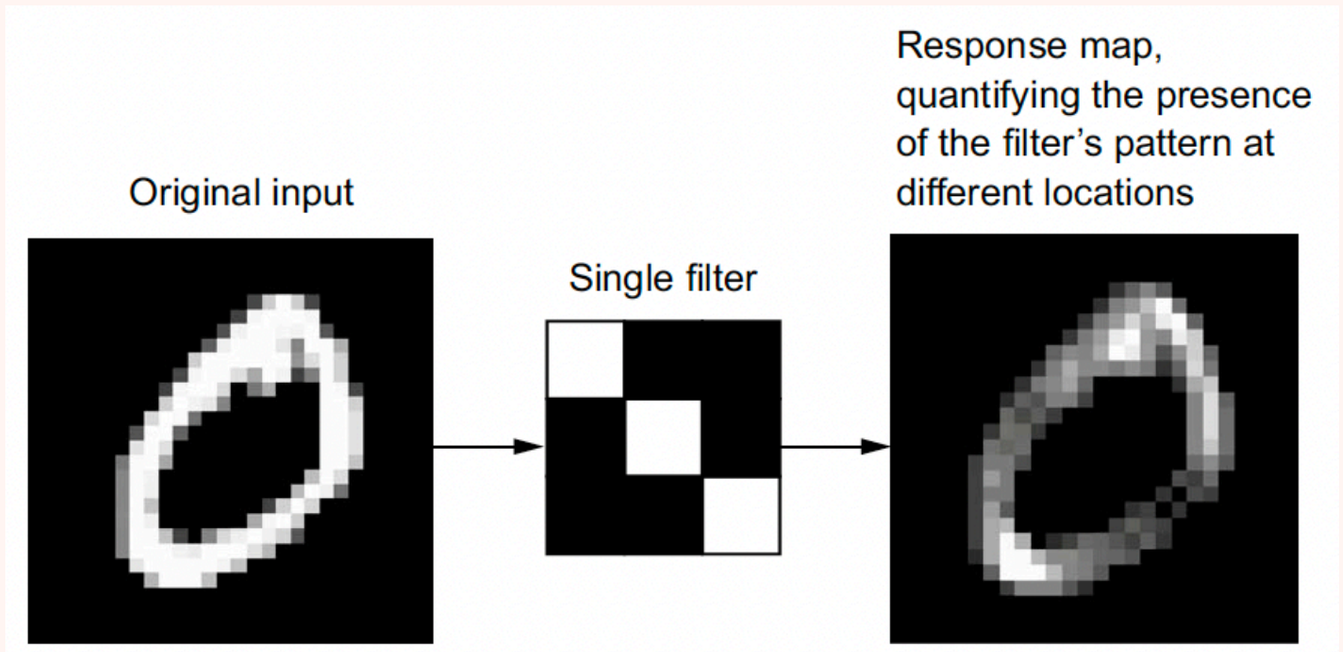
CONVOLUTION OPERATION

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320

Size of response map

Number of filters



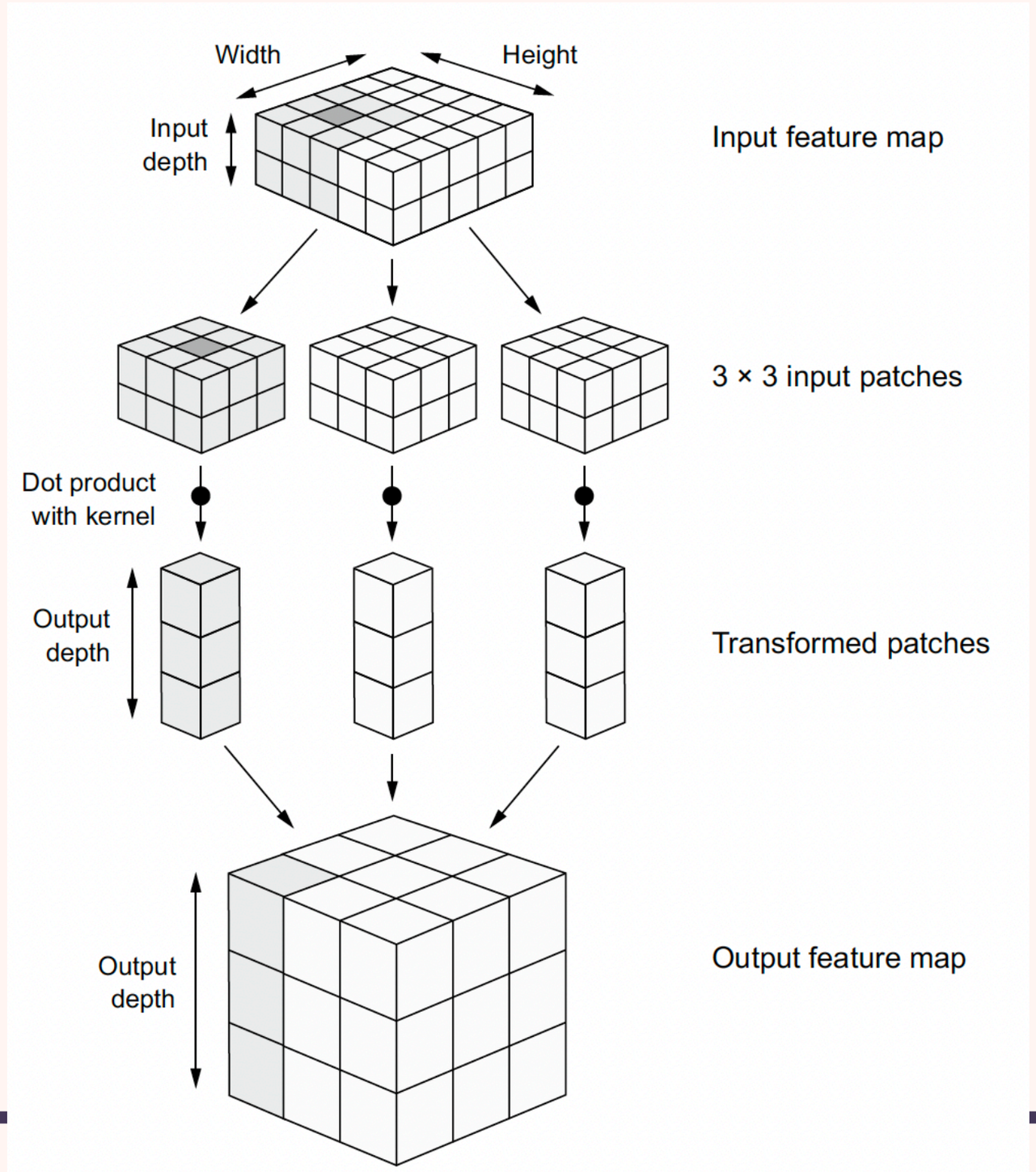
Response map = the response of that filter pattern at different location of the input

Feature map

Convolutions are defined by two key parameters:

- *Size of the patches extracted from the inputs*—These are typically 3 × 3 or 5 × 5. In the example, they were 3 × 3, which is a common choice.
- *Depth of the output feature map*—This is the number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.

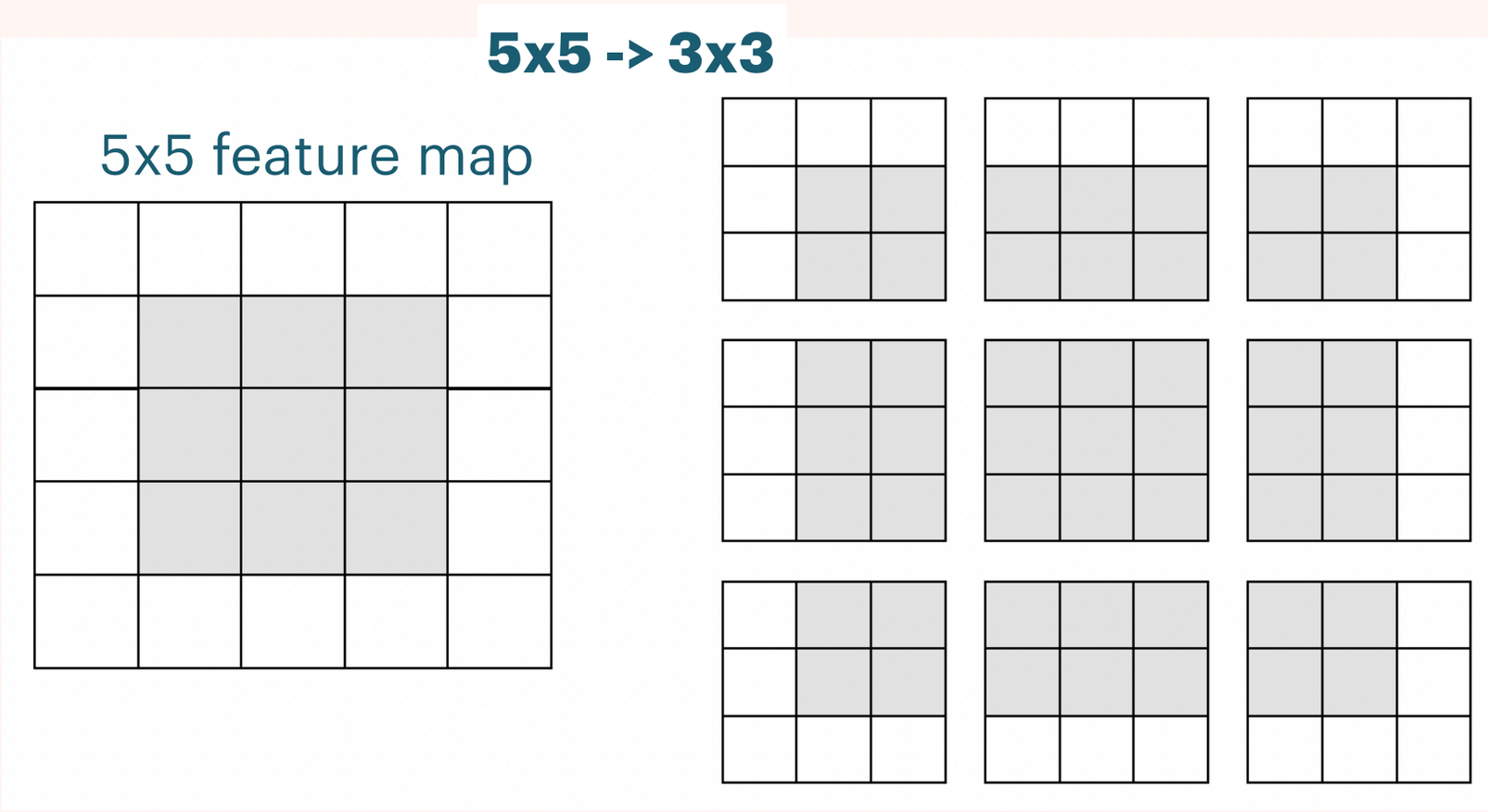
CONVOLUTION OPERATION



1. Cut the input feature map into small patches
2. Dot product with “convolutional kernel”, a learned weight matrix

CONVOLUTIONAL OPERATION

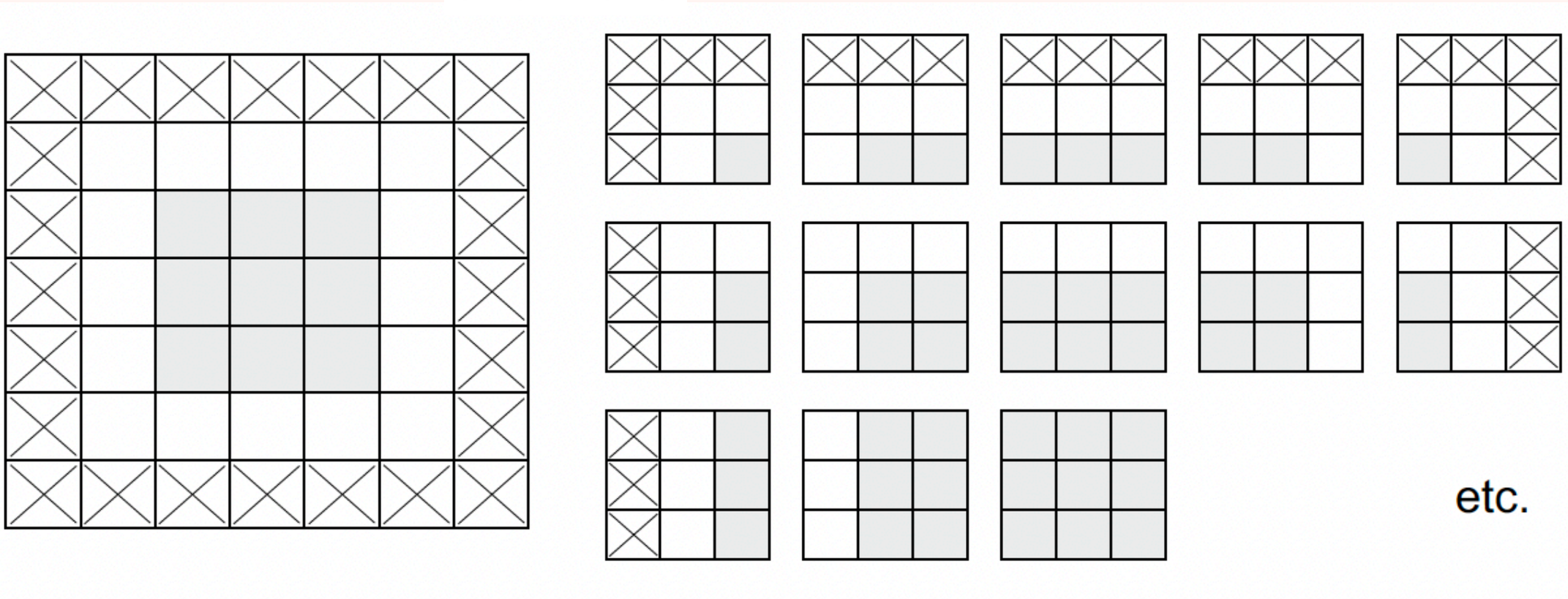
Border effect



CONVOLUTIONAL OPERATION

- **Want the same spatial dimensions? -> padding**
  - Adding columns and rows so that we can do centered convolution at every input tile

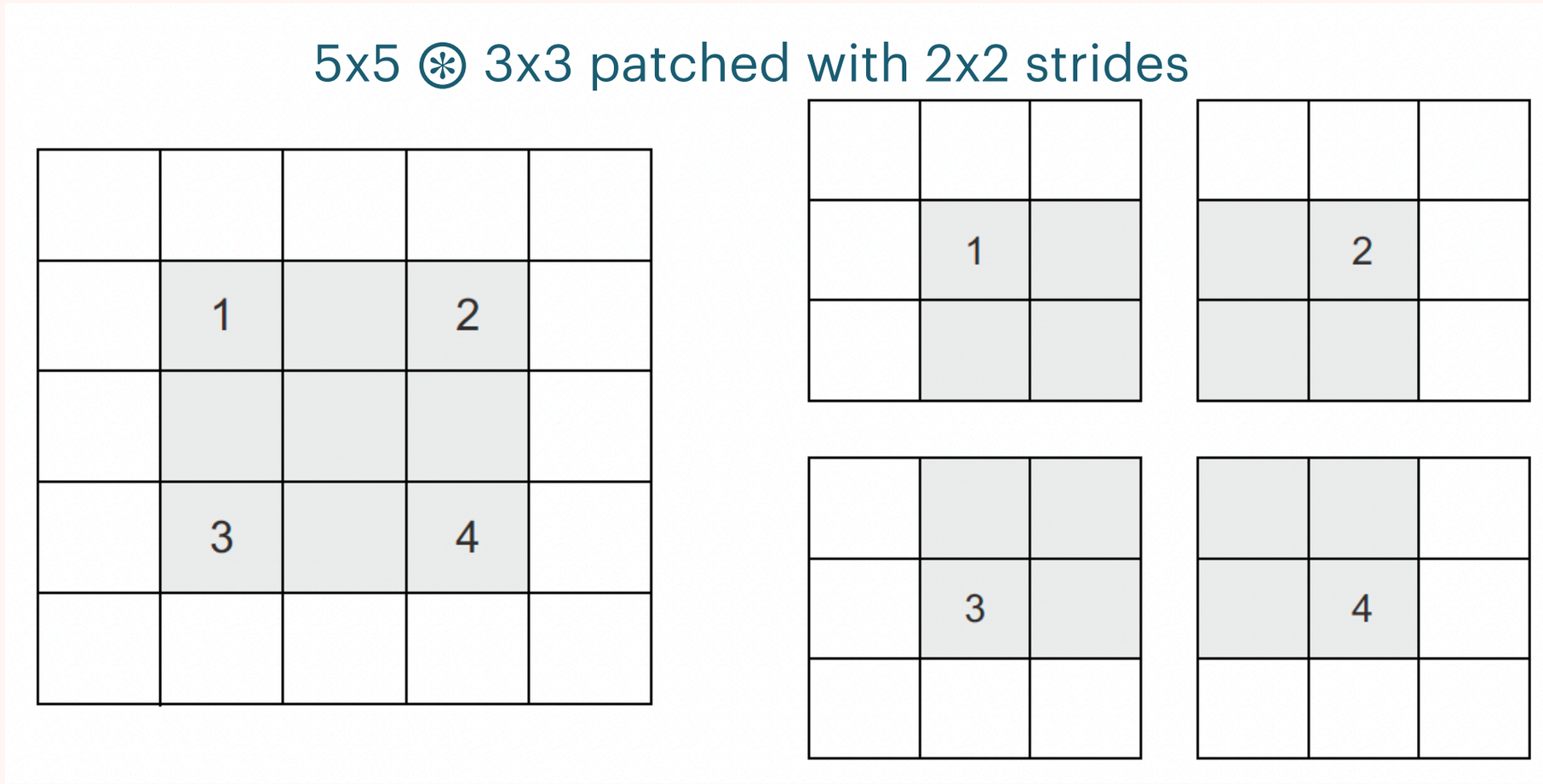
5x5 -> 5x5



In Conv2D layer, “padding” argument: **valid**=“no padding” or **same**=“same dimensions”

CONVOLUTIONAL OPERATION

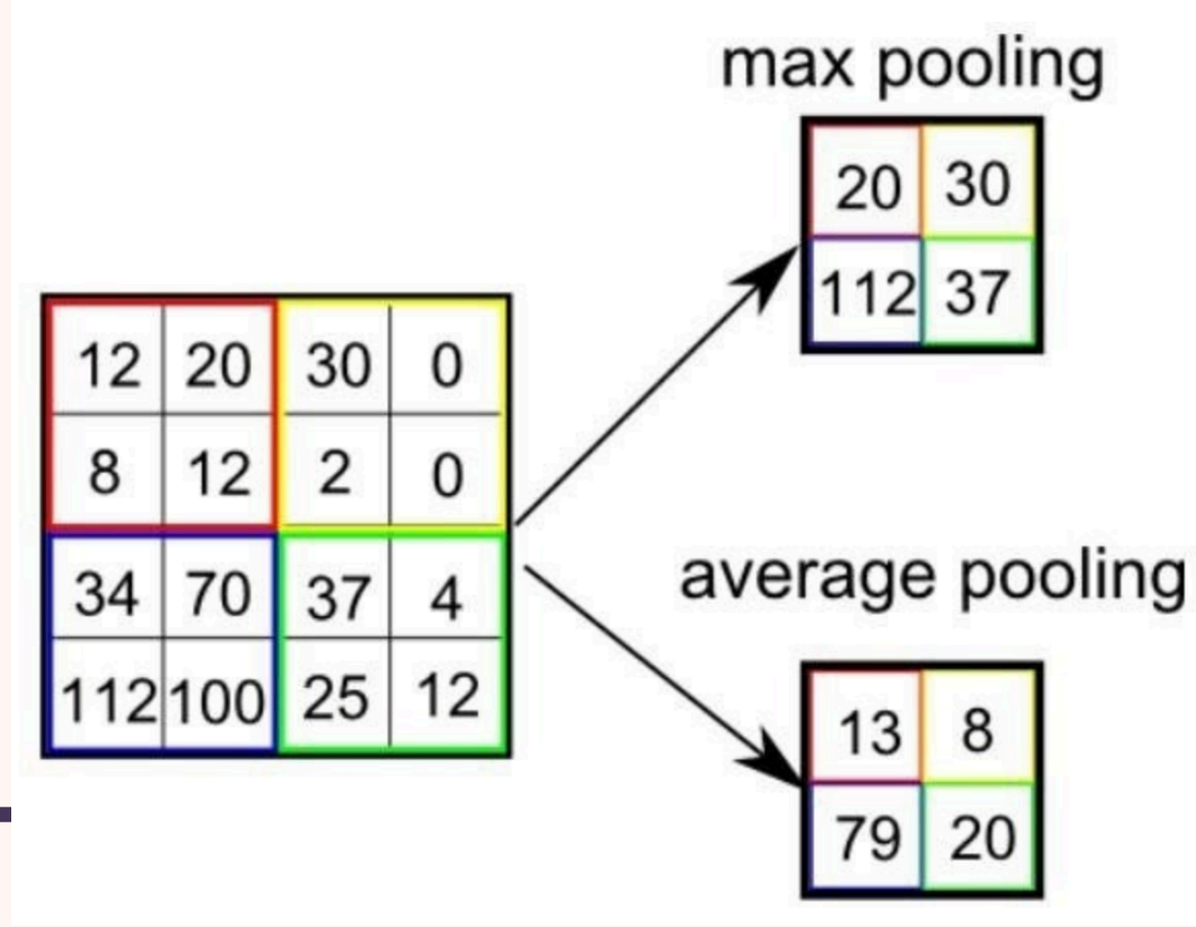
- **Convolution strides** (default = 1)



MAX POOLING

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0



LAST LAYERS

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

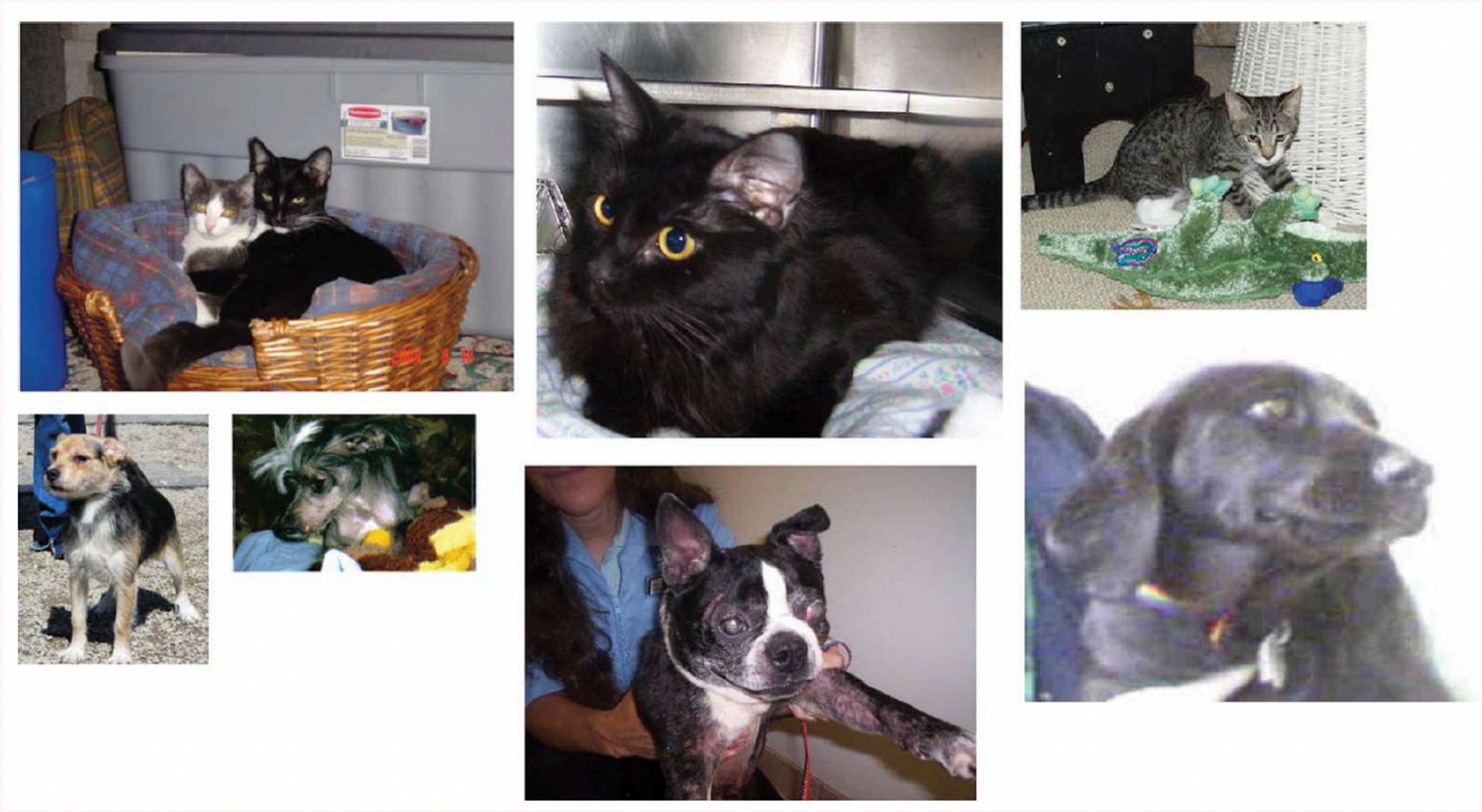
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 10)	11,530

Choosing the right last-layer activation and loss function for your model

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`

LET'S TRY WITH SMALL DATA

- **kaggle.com** Cat-dog dataset
  - 25,000 images of cats and dogs
    - **Try to use only 2,000 samples to train (1,000 images for each class)**



COPYING SMALL SET OF DATA

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")
```

Path to the directory where the original dataset was uncompressed

Directory where we will store our smaller dataset

```
def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg"
                  for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2500)
```

Utility function to copy cat (and dog) images from index start_index to index end_index to the subdirectory new_base_dir/{subset_name}/cat (and /dog). The "subset_name" will be either "train", "validation", or "test".

Create the training subset with the first 1,000 images of each category.

Create the validation subset with the next 500 images of each category.

Create the test subset with the next 1,000 images of each category.

CREATING MODEL

The model expects RGB images of size 180 × 180.

```
inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Rescale inputs to the [0, 1] range by dividing them by 255.

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

USE TENSORFLOW TO CREATE DATASETS

import image into dataset

```python
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

✓ 0.0s MagicPython

```
Found 1999 files belonging to 2 classes.
Found 1000 files belonging to 2 classes.
Found 2000 files belonging to 2 classes.
```

1. Read the picture files.
2. Decode the JPEG content to RGB grids of pixels.
3. Convert these into floating-point tensors.
4. Resize them to a shared size (we'll use 180 × 180).
5. Pack them into batches (we'll use batches of 32 images)

TRAIN MODEL

```python
callbacks = [
    keras.callbacks.ModelCheckpoint(
    filepath="convnet_from_scratch.keras",
    save_best_only=True,
    monitor="val_loss")
]
```

✓ 0.0s MagicPython

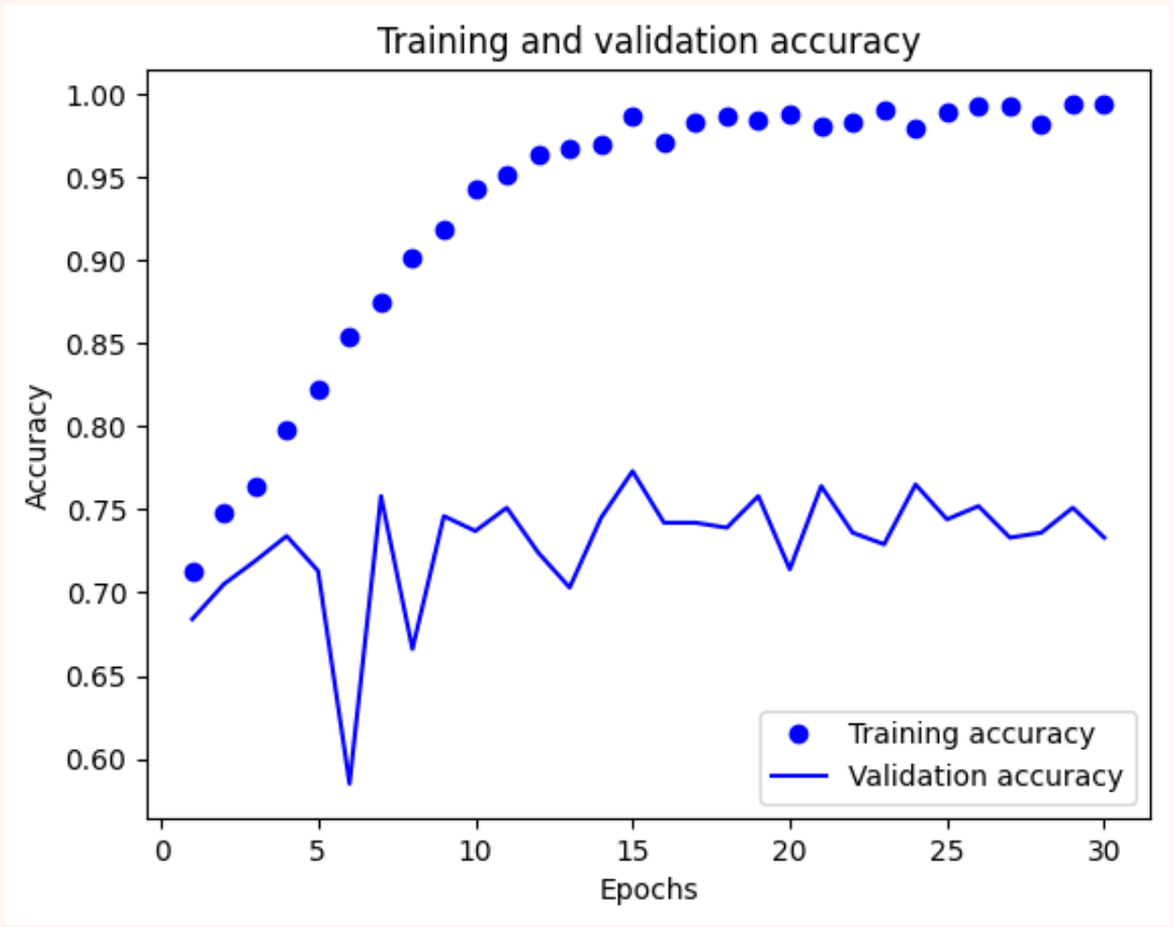
Save only the better version when val_loss is lower

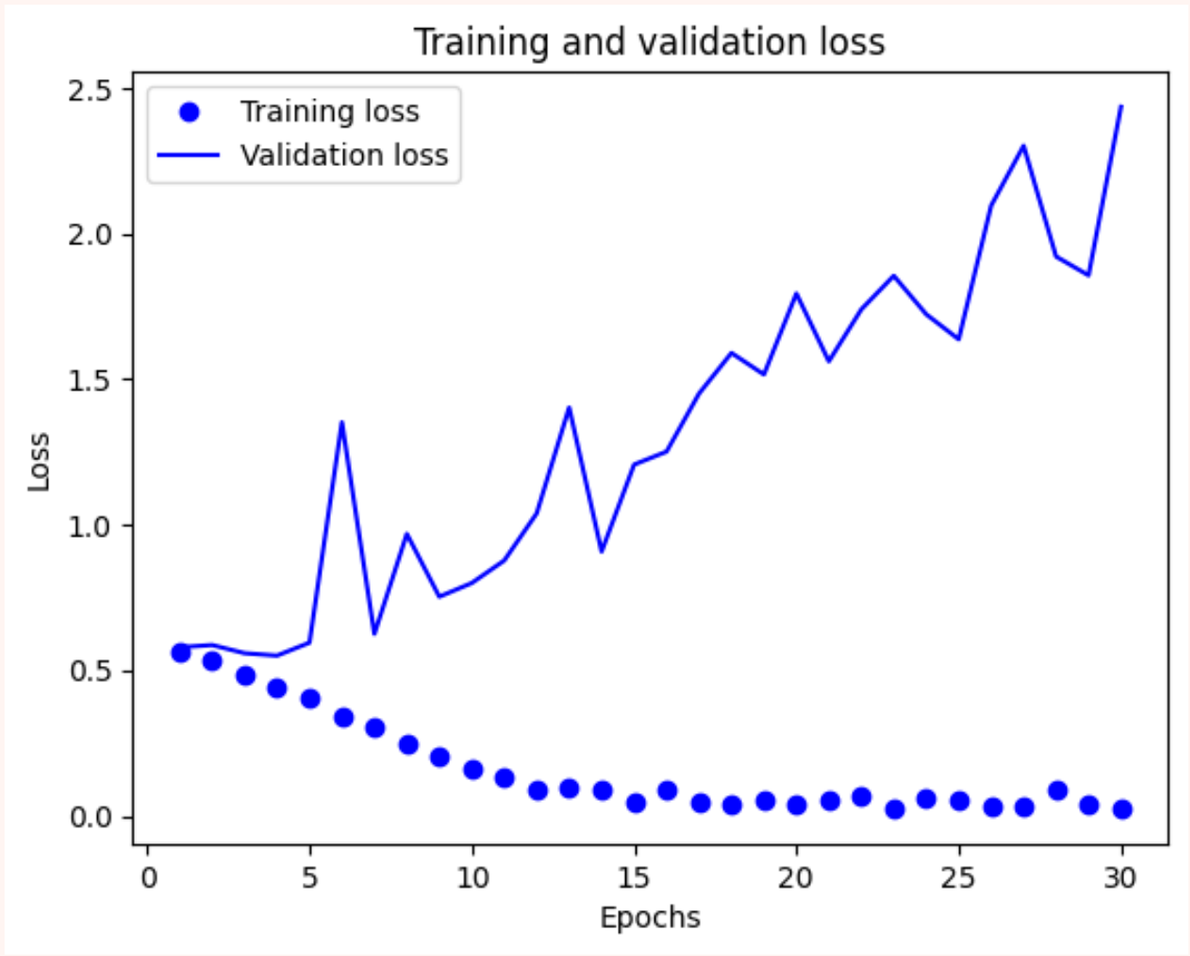
```python
history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

✓ 17m 26.9s

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 35s 556ms/step - accuracy: 0.9895 - loss: 0.0541 - val_accuracy: 0.7440 - val_loss: 1.6362
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 484ms/step - accuracy: 0.9972 - loss: 0.0153
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 35s 559ms/step - accuracy: 0.9930 - loss: 0.0347 - val_accuracy: 0.7520 - val_loss: 2.0952
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 500ms/step - accuracy: 0.9922 - loss: 0.0372
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 37s 585ms/step - accuracy: 0.9925 - loss: 0.0344 - val_accuracy: 0.7330 - val_loss: 2.3008
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 646ms/step - accuracy: 0.9825 - loss: 0.0939
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 46s 731ms/step - accuracy: 0.9815 - loss: 0.0869 - val_accuracy: 0.7360 - val_loss: 1.9204
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 488ms/step - accuracy: 0.9963 - loss: 0.0142
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 35s 561ms/step - accuracy: 0.9935 - loss: 0.0376 - val_accuracy: 0.7510 - val_loss: 1.8558
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 454ms/step - accuracy: 0.9958 - loss: 0.0216
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
63/63 ━━━━━━━━━━━━━━━━━━━━ 33s 524ms/step - accuracy: 0.9940 - loss: 0.0259 - val_accuracy: 0.7330 - val_loss: 2.4346
```

DISPLAY ACCURACY & LOSS

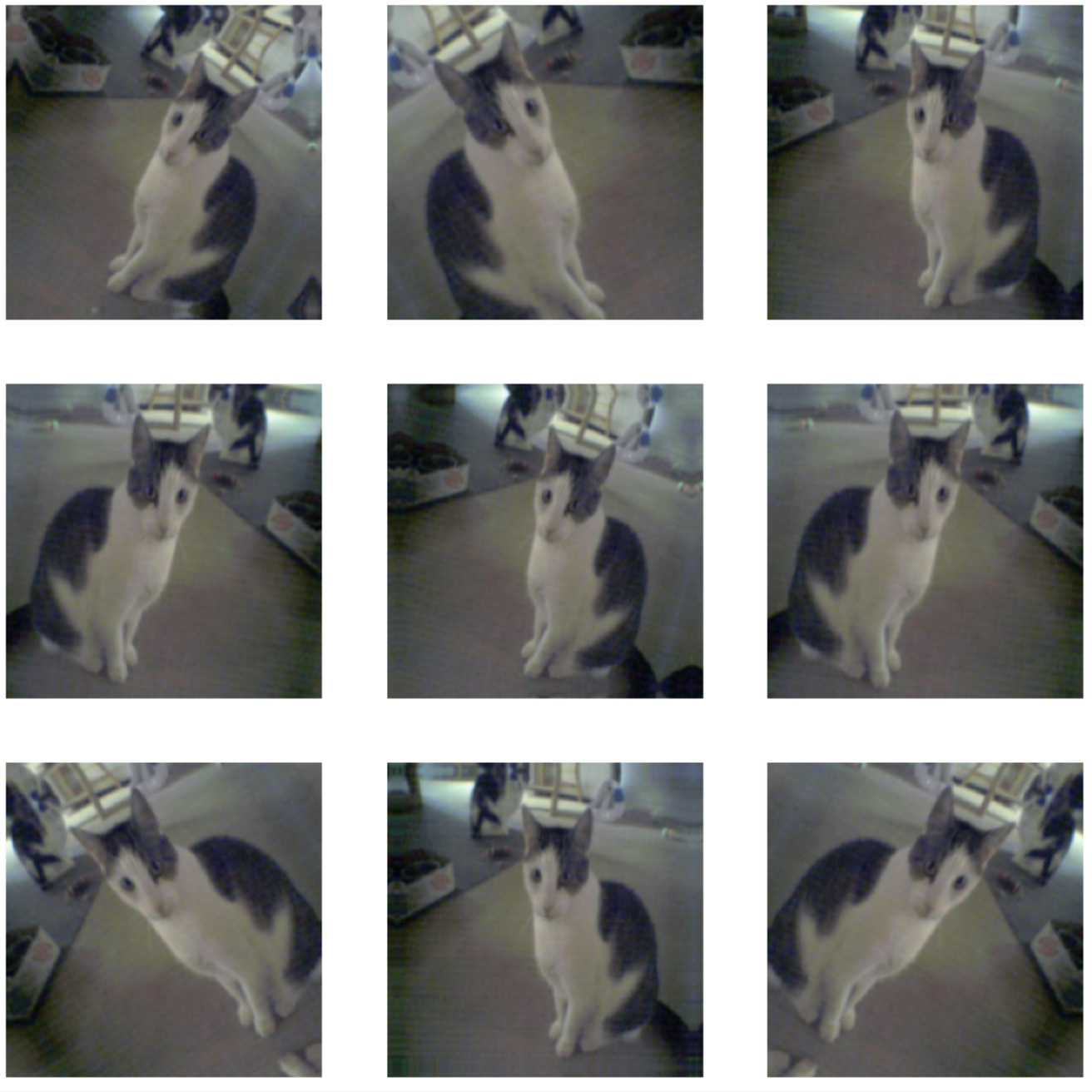




Clearly overfitting!

DATA AUGMENTATION

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```



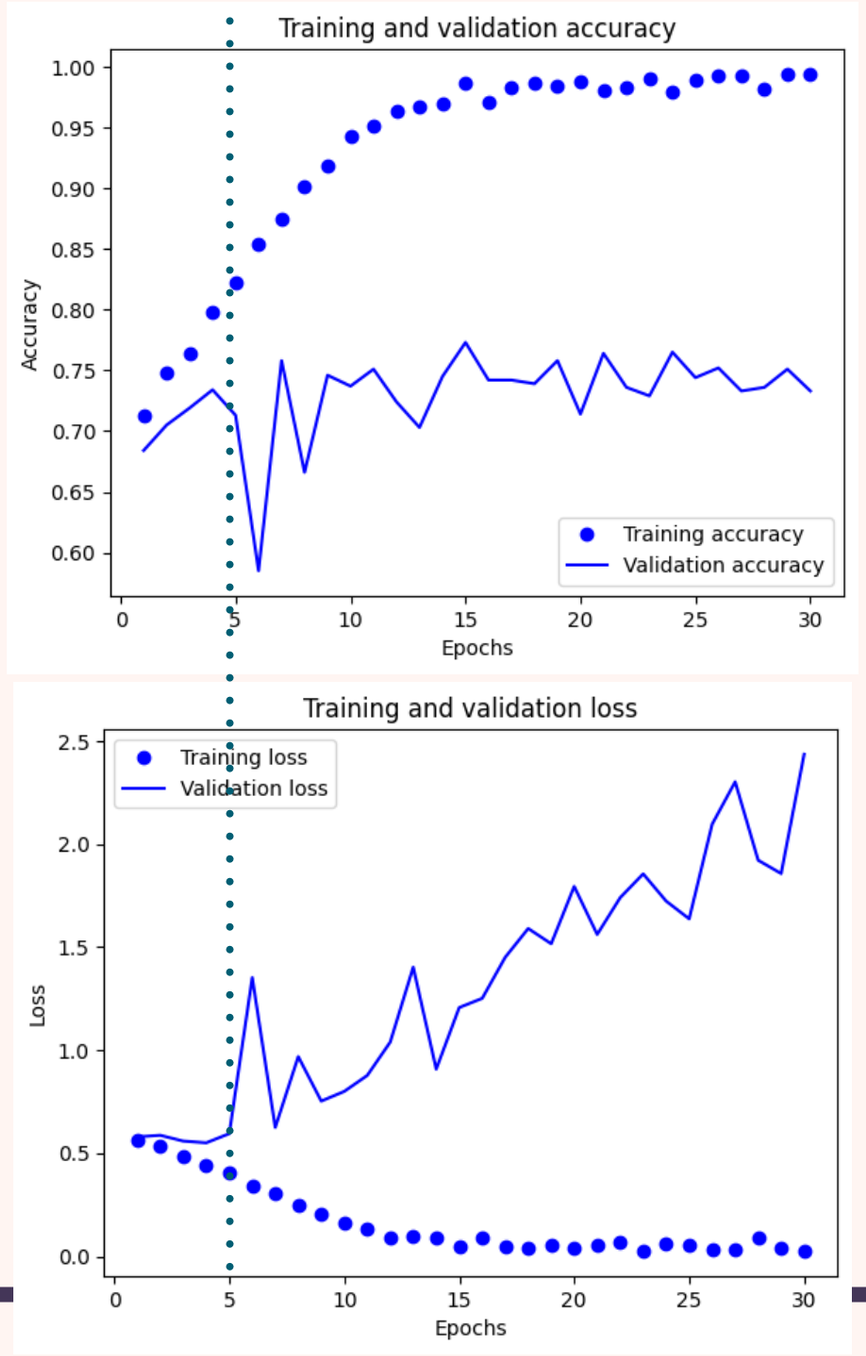
- **RandomFlip**: Horizontal vs vertical
- **RandomRotation(0.1)**
  - 10% rotation (+/- 36 degrees)
- **RandomZoom(0.2)**
  - +/- 20% zoom in-out

REBUILD MODEL

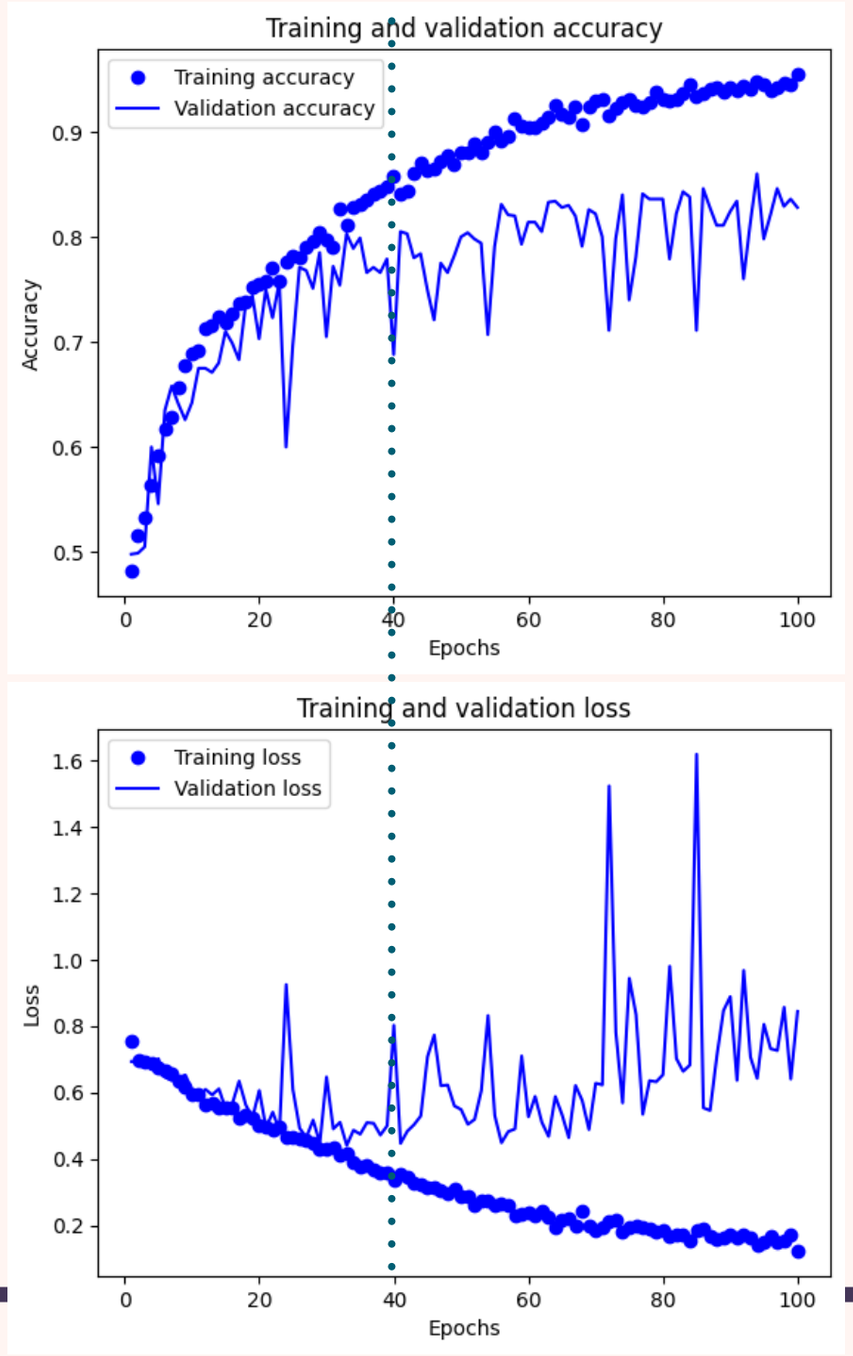
Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 180, 180, 3)	0
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_10 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_11 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_12 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_10 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_13 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_11 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_14 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_2 (Flatten)	(None, 12544)	0
dropout_1 (Dropout)	(None, 12544)	0
dense_2 (Dense)	(None, 1)	12,545

POWER OF DATA AUGMENTATION

Before

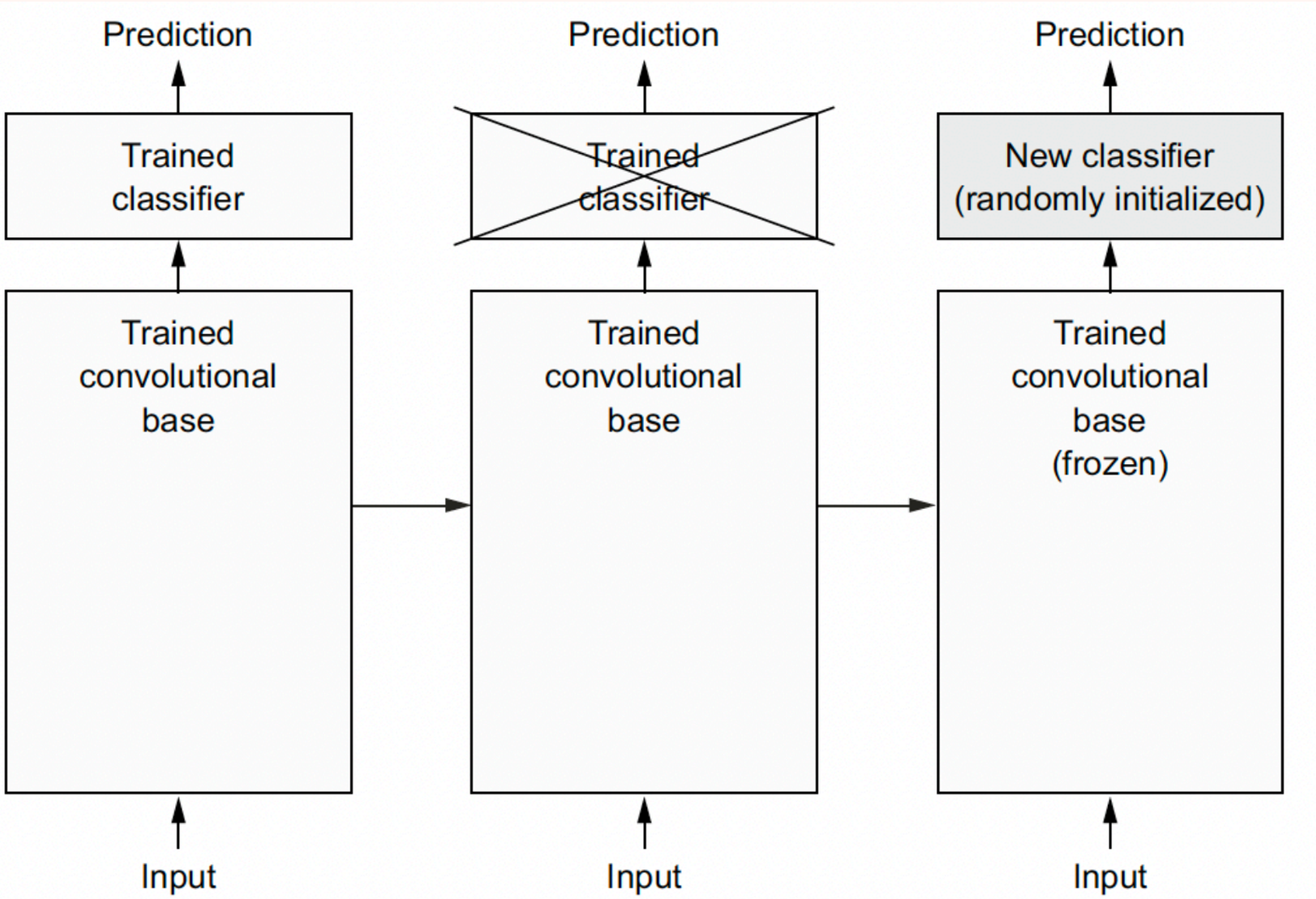


After



LEVERAGING THE PRETRAINED MODEL

- A pretrained model is a model that was previously trained with large datasets.

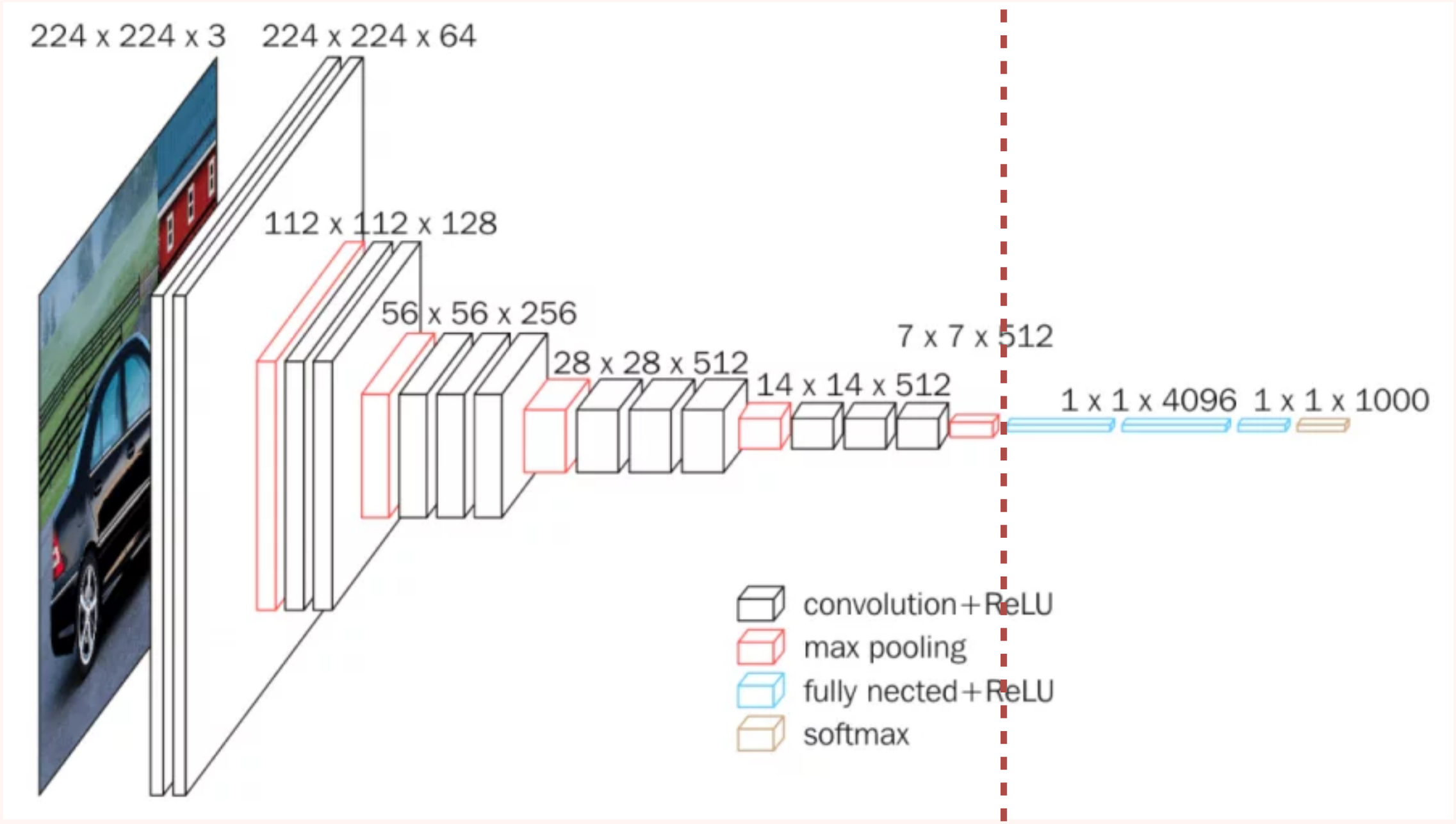


Frequently used pertained models

1. VGG16
2. Xception
3. ResNet
4. MobileNet
5. GoogLeNet
6. AlexNet

Etc.

EXAMPLE: VGG16



IMPLEMENT CONVOLUTIONAL VGG16

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 180, 180, 3)	0
block1_conv1 (Conv2D)	(None, 180, 180, 64)	1,792
block1_conv2 (Conv2D)	(None, 180, 180, 64)	36,928
block1_pool (MaxPooling2D)	(None, 90, 90, 64)	0
block2_conv1 (Conv2D)	(None, 90, 90, 128)	73,856
block2_conv2 (Conv2D)	(None, 90, 90, 128)	147,584
block2_pool (MaxPooling2D)	(None, 45, 45, 128)	0
block3_conv1 (Conv2D)	(None, 45, 45, 256)	295,168
block3_conv2 (Conv2D)	(None, 45, 45, 256)	590,080
block3_conv3 (Conv2D)	(None, 45, 45, 256)	590,080
block3_pool (MaxPooling2D)	(None, 22, 22, 256)	0
block4_conv1 (Conv2D)	(None, 22, 22, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 22, 22, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 11, 11, 512)	0
block5_conv1 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 11, 11, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 5, 5, 512)	0

2 WAYS OF IMPLEMENTING

1. Run convolutional base of the pretrained model -> Numpy array

    - Then use the Numpy array to train the densely connected classifier separately.

      Drawback: cannot use with data augmentation

2. Extend the model by adding the Dense layers and run the whole thing

   Drawback: computationally expensive

FIRST METHOD: SEPARATELY TRAIN

Without data augmentation

```
import numpy as np

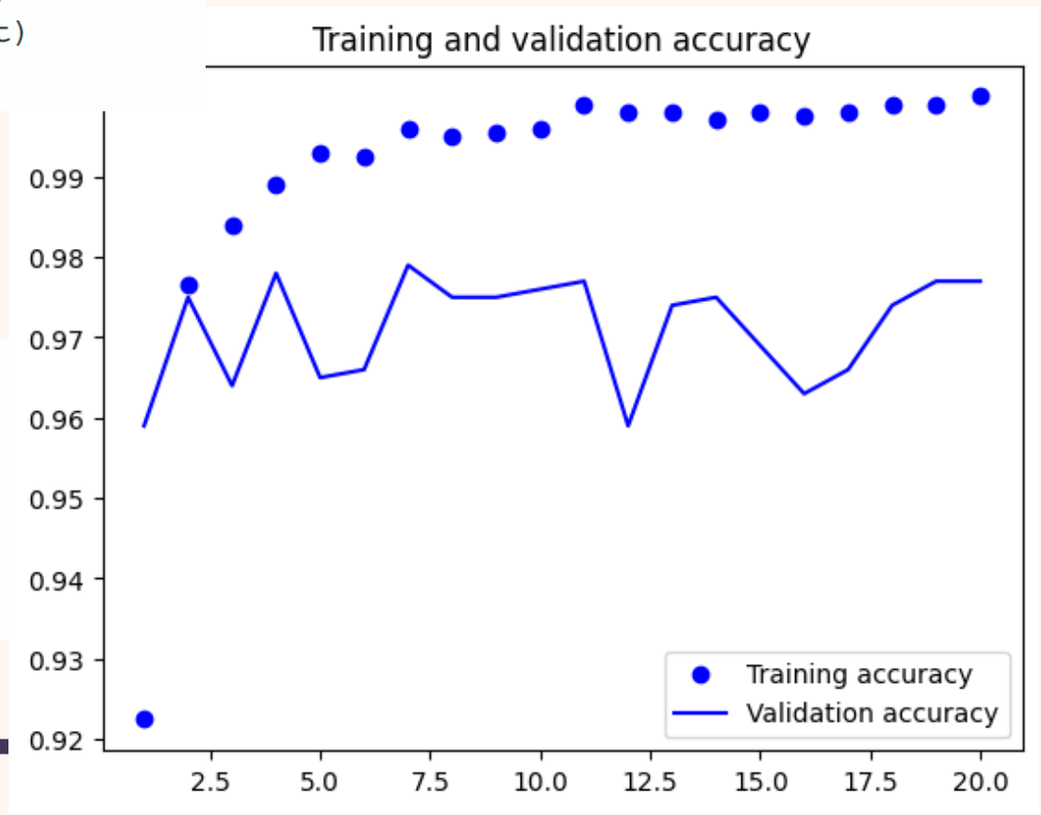
def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
test_features, test_labels =  get_features_and_labels(test_dataset)
```

Get feature from convolutional base of the pretrained model

```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
```

Note the use of the Flatten layer before passing the features to a Dense layer.



SECOND METHOD: EXTEND MODEL

- Adding the densely connected layers to the convolutional base
  - Don’t forget to freeze the convolutional base from updating the weights

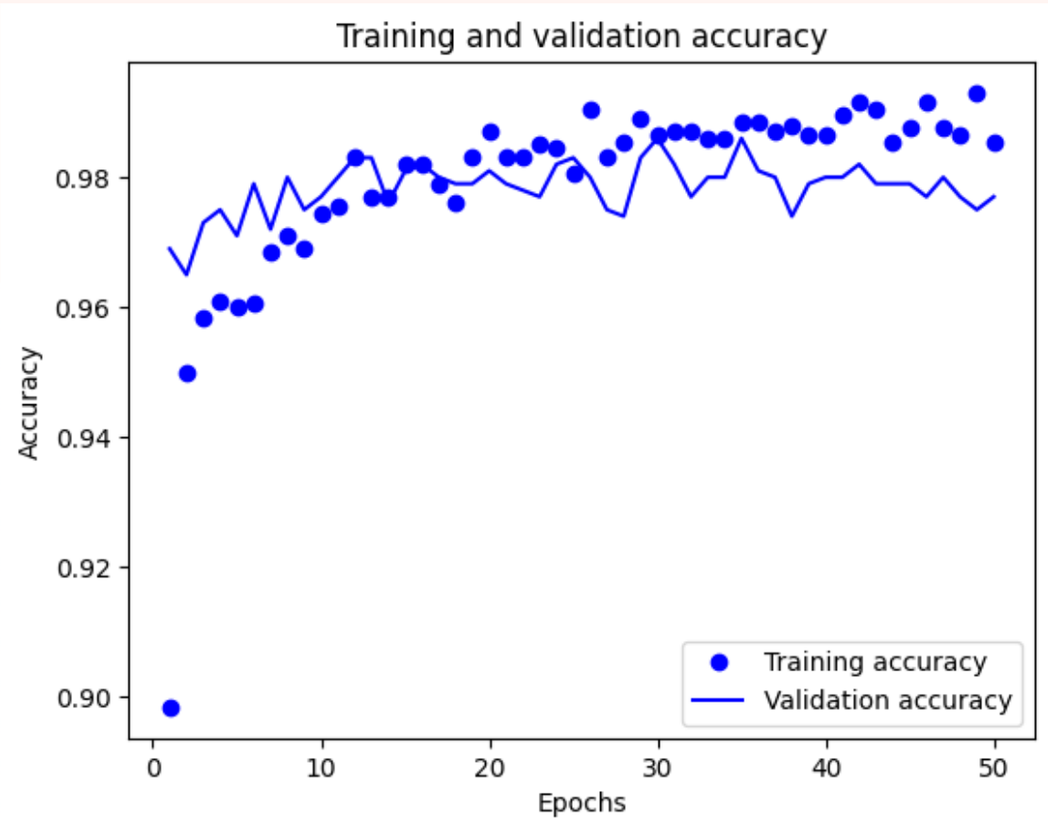
```
conv_base  = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False)
conv_base.trainable = False
```

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Apply data augmentation.

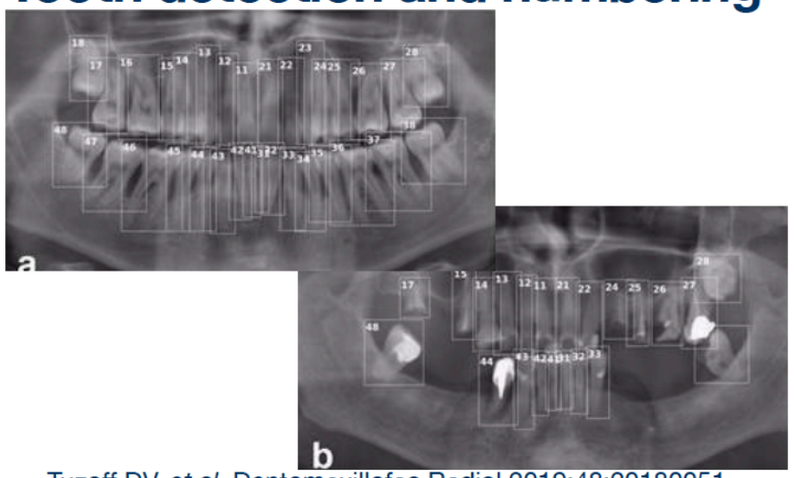
Apply input value scaling.



TASKS USING CNN

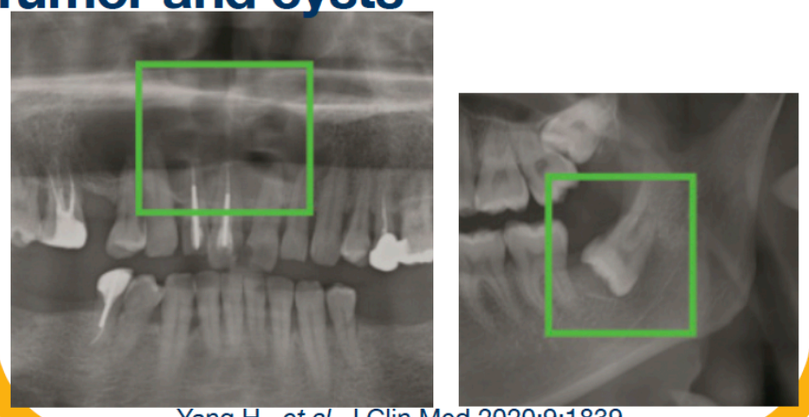
Detection

Tooth detection and numbering



Tuzoff DV.,*et al.* Dentomaxillofac Radiol.2019;48:20180051.

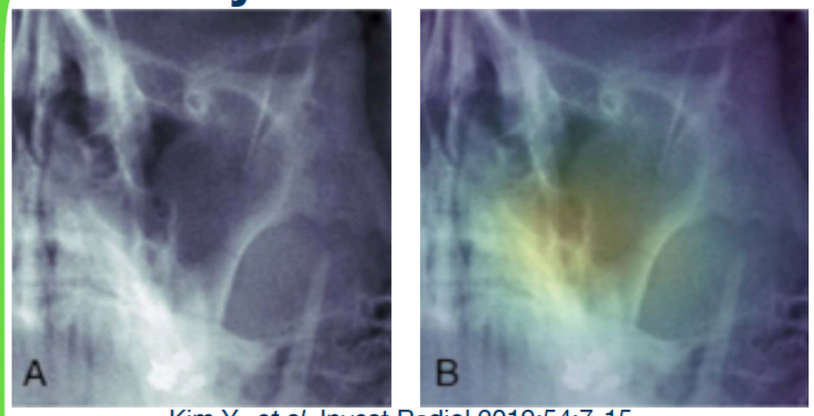
Tumor and cysts



Yang H., *et al.* J Clin Med.2020;9:1839.

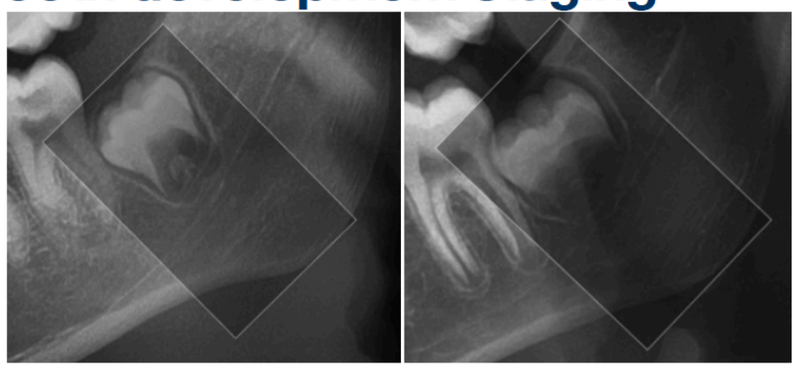
Classification

Maxillary sinusitis



Kim Y., *et al.* Invest Radiol.2019;54:7-15.

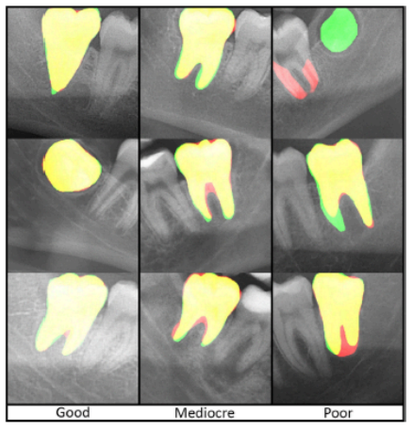
Tooth development staging

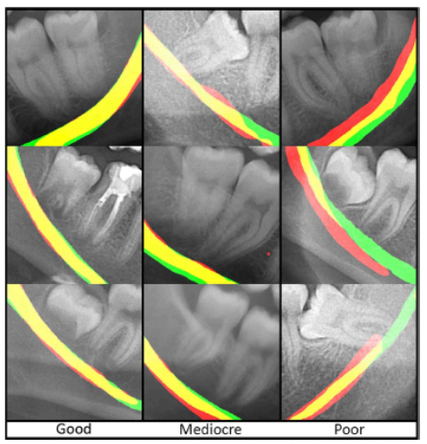


Merdieto Boedi R., *et al*. J Forensic Sci.2020;65:481-6.

Segmentation

Third molar & IAC





Vinayahalingam S., *et al*. Sci Rep.2019;9:9007.

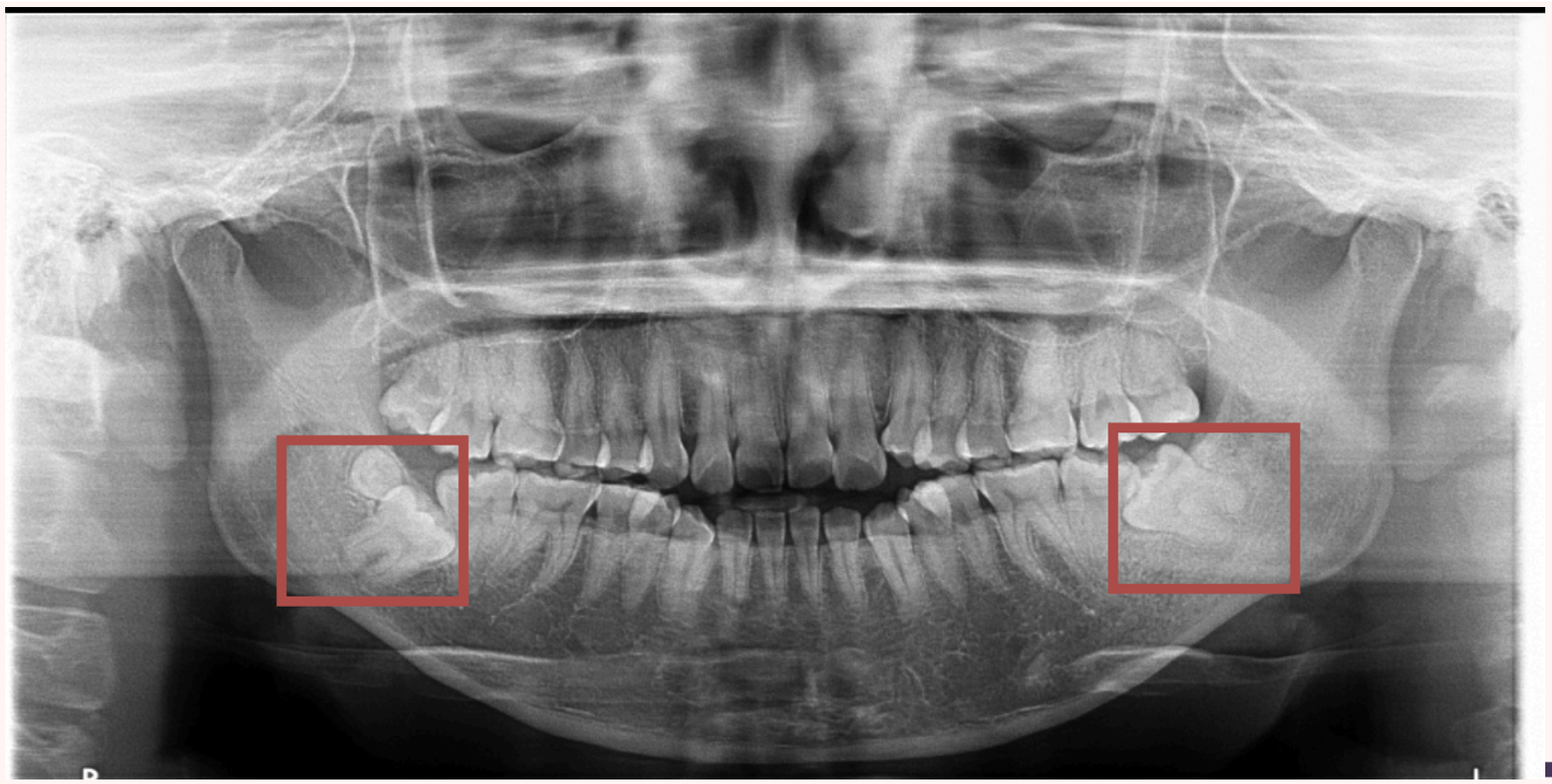
CLASSIFICATION PROBLEM

Clinical Oral Investigations
https://doi.org/10.1007/s00784-023-04992-6

RESEARCH

Investigation of the best effective fold of data augmentation for training deep learning models for recognition of contiguity between mandibular third molar and inferior alveolar canal on panoramic radiographs

Dhanaporn Papasratorn[1] · Suchaya Pornprasertsuk-Damrongsri[1] · Suraphong Yuma[2] · Warangkana Weerawanich[1]



Models		AUC[a]	*P*-value[b]
Pretrained CNN	Training set[c]		
AlexNet	Original	0.956	-
	5×	0.994	0.0174*
	10×	0.995	0.0152*
	15×	0.989	0.0343*
	20×	0.988	0.0386*
VGG-16	Original	0.981	-
	5×	0.990	0.3138
	10×	0.996	0.0817
	15×	0.992	0.2078
	20×	0.992	0.2113
GoogLeNet	Original	0.951	-
	5×	0.982	0.0151*
	10×	0.990	0.0091*
	15×	0.987	0.0026*
	20×	0.985	0.0040*

SEGMENTATION PROBLEM

A comparison of encoder architectures for proximal caries segmentation in bitewing radiographs

Nannapat Mitpothong[1], Nitchanan Jirachaiprasit[2], Suchaya Pornprasertsuk-Damrongsri[3], Warangkana Weerawanich[3], Chanita Tantipoj[2], Chatdanai Sawangwong[1], Charin Modchang[1], Naiyana Buranachad[2], Suraphong Yuma[1*]

To be submitted to Clinical Oral Investigation

