

# Computer Vision

ACTL3143 & ACTL5111 Deep Learning for Actuaries  
Patrick Laub



# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# Shapes of data

A tensor is an N-dimensional array of data

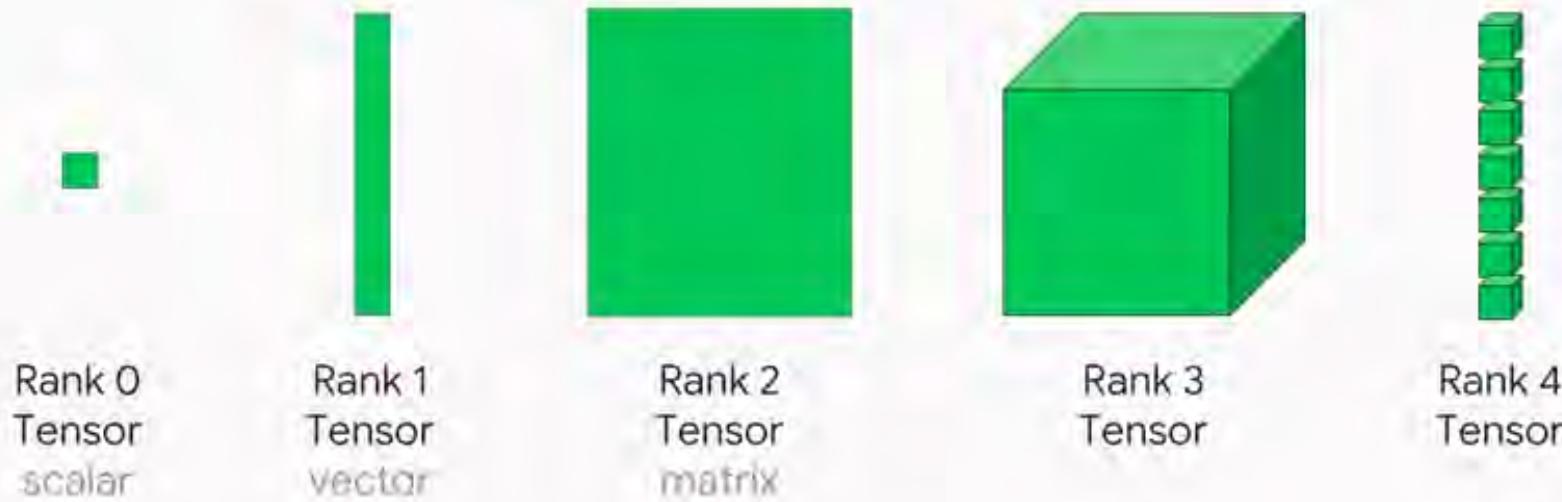


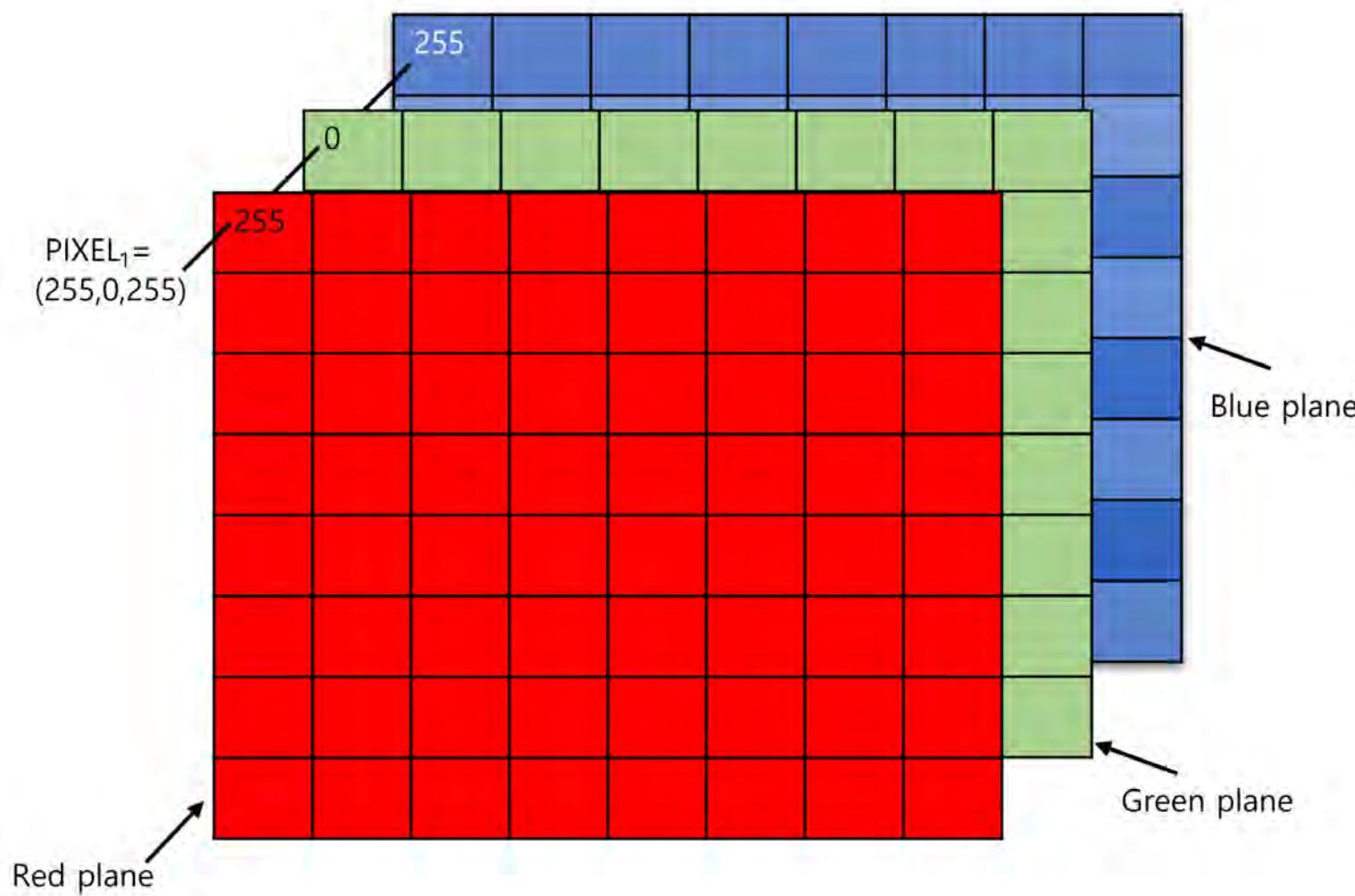
Illustration of tensors of different rank.



Source: Paras Patidar (2019), [Tensors — Representation of Data In Neural Networks](#), Medium article.



# Shapes of photos



A photo is a rank 3 tensor.



Source: Kim et al (2021), Data Hiding Method for Color AMBTC Compressed Images Using Color Difference, Applied Sciences.



# How the computer sees them

```

1 from matplotlib.image import imread
2 img1 = imread('pu.gif'); img2 = imread('pl.gif')
3 img3 = imread('pr.gif'); img4 = imread('pg.bmp')
4 f"Shapes are: {img1.shape}, {img2.shape}, {img3.shape}, {img4.shape}."
```

'Shapes are: (16, 16, 3), (16, 16, 3), (16, 16, 3), (16, 16, 3).'

1 img1

```
array([[[ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]]],
```

1 img2

```
array([[[ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]]],
```

1 img3

```
array([[[ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]]],
```

1 img4

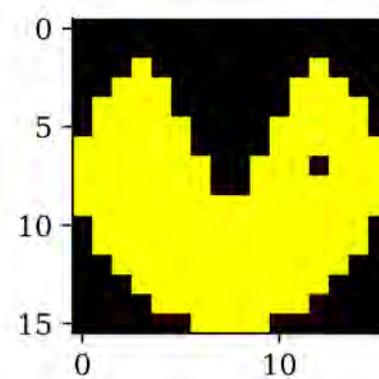
```
array([[[ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]]],
```



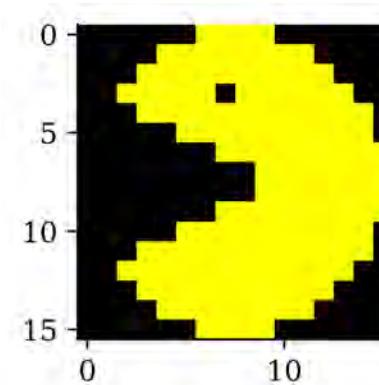
# How we see them

```
1 from matplotlib.pyplot import imshow
```

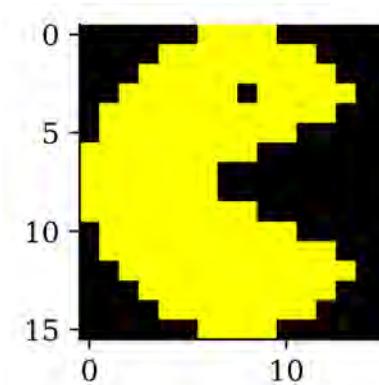
```
1 imshow(img1);
```



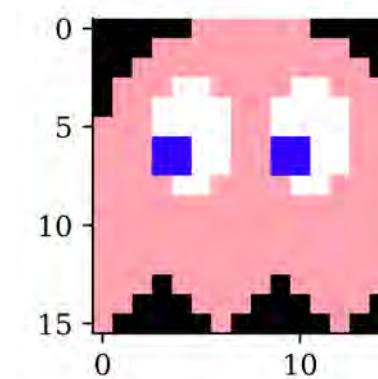
```
1 imshow(img2);
```



```
1 imshow(img3);
```



```
1 imshow(img4);
```



# Why is 255 special?

Each pixel's colour intensity is stored in one byte.

One byte is 8 bits, so in binary that is 00000000 to 11111111.

The largest *unsigned* number this can be is  $2^8 - 1 = 255$ .

```
1 np.array([0, 1, 255, 256]).astype(np.uint8)
```

```
array([ 0,  1, 255,  0], dtype=uint8)
```

If you had *signed* numbers, this would go from -128 to 127.

```
1 np.array([-128, 1, 127, 128]).astype(np.int8)
```

```
array([-128,  1, 127, -128], dtype=int8)
```

Alternatively, *hexidecimal* numbers are used. E.g. 10100001 is split into 1010 0001, and 1010=A, 0001=1, so combined it is 0xA1.



# Image editing with kernels

Take a look at <https://setosa.io/ev/image-kernels/>.

1 x1	1 x0	1 x1	0	0
0 x0	1 x1	1 x0	1	0
0 x1	0 x0	1 x1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved  
Feature

An example of an image kernel in action.



Source: Stanford's deep learning tutorial via Stack Exchange.



UNSW  
SYDNEY

# Lecture Outline

- Images
- **Convolutional Layers**
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# ‘Convolution’ not ‘complicated’

Say  $X_1, X_2 \sim f_X$  are i.i.d., and we look at  $S = X_1 + X_2$ .

The density for  $S$  is then

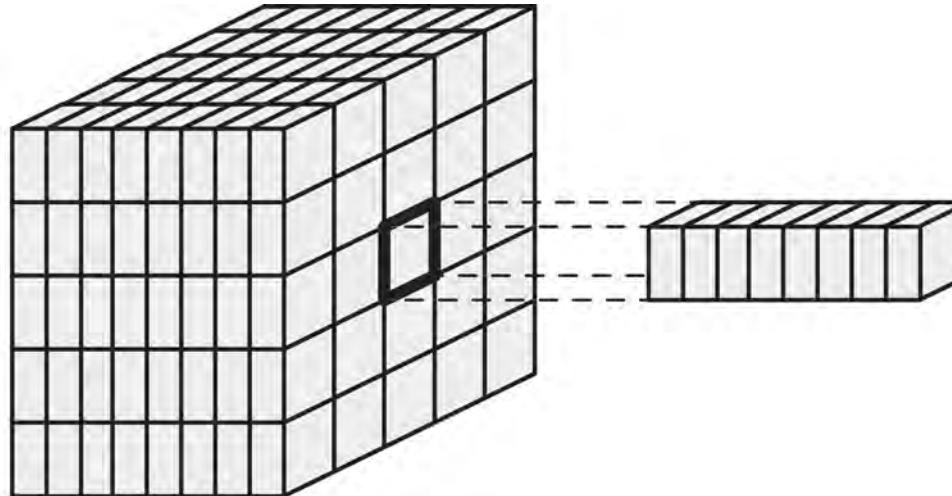
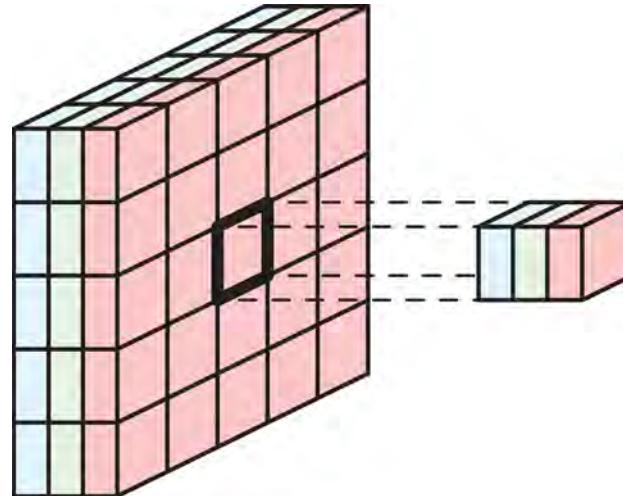
$$f_S(s) = \int_{x_1=-\infty}^{\infty} f_X(x_1) f_X(s - x_1) \mathrm{d}s.$$

This is the *convolution* operation,  $f_S = f_X \star f_X$ .



# Images are rank 3 tensors

Height, width, and number of channels.



Examples of rank 3 tensors.

Grayscale image has 1 channel. RGB image has 3 channels.

Example: Yellow = Red + Green.

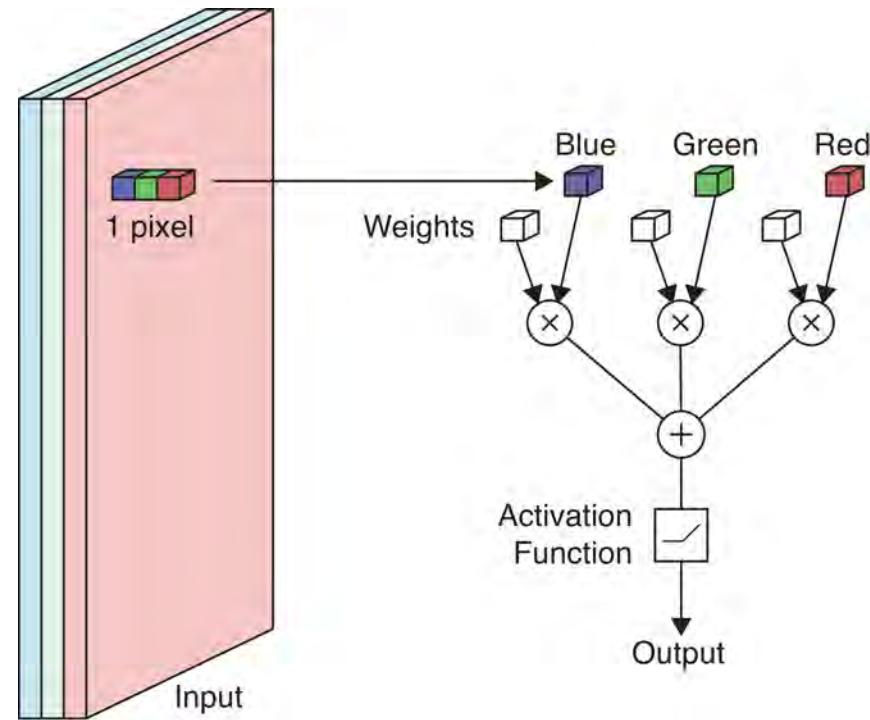


Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.



UNSW  
SYDNEY

# Example: Detecting yellow



Applying a neuron to an image pixel.

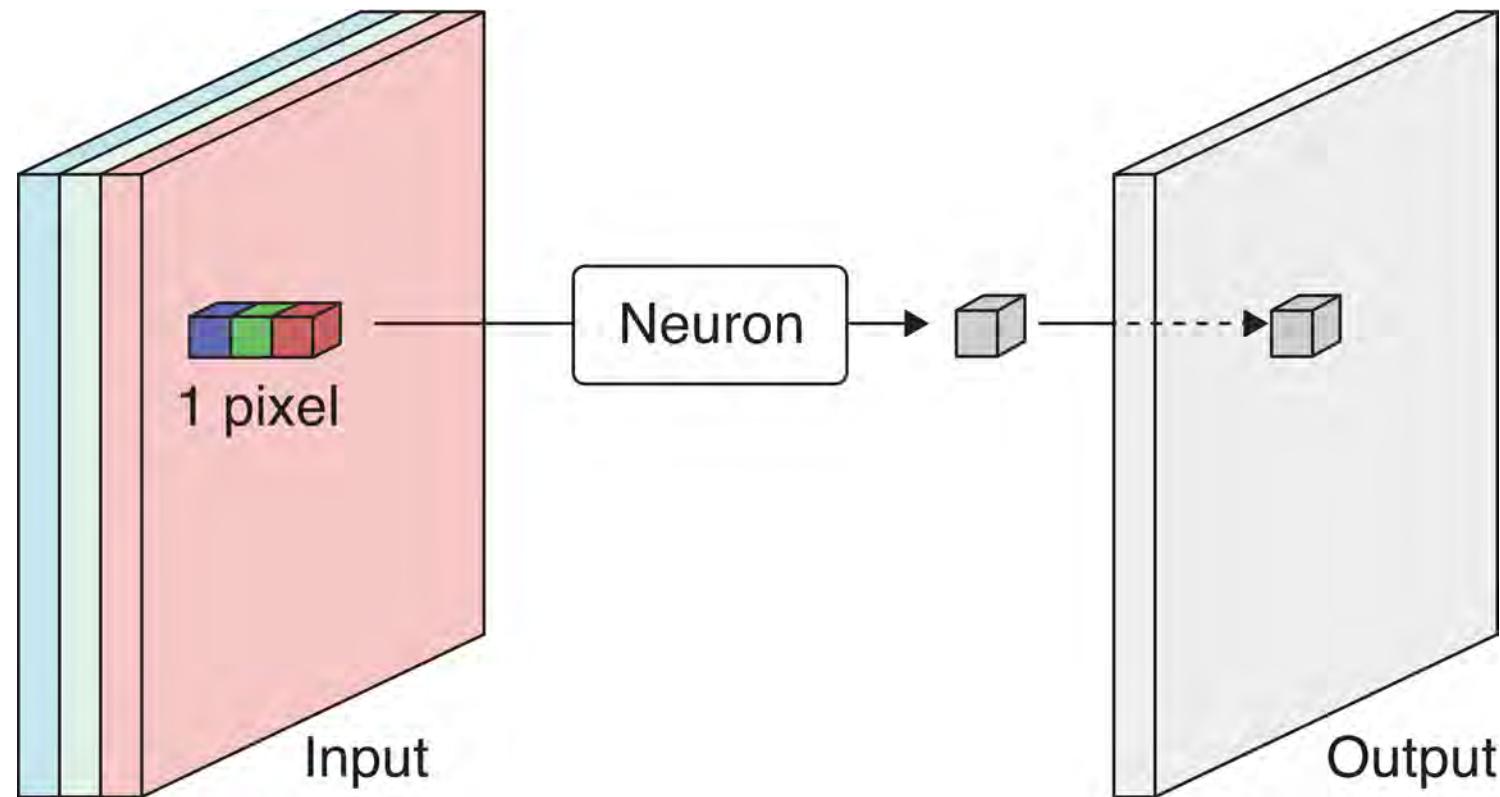
Apply a neuron to each pixel in the image.

If red/green  $\nearrow$  or blue  $\searrow$  then yellowness  $\nearrow$ .

Set RGB weights to 1, 1, -1.



# Example: Detecting yellow II



Scan the 3-channel input (colour image) with the neuron to produce a 1-channel output (grayscale image).

The output is produced by *sweeping* the neuron over the input. This is called **convolution**.



# Example: Detecting yellow III



The more yellow the pixel in the colour image (left), the more white it is in the grayscale image.

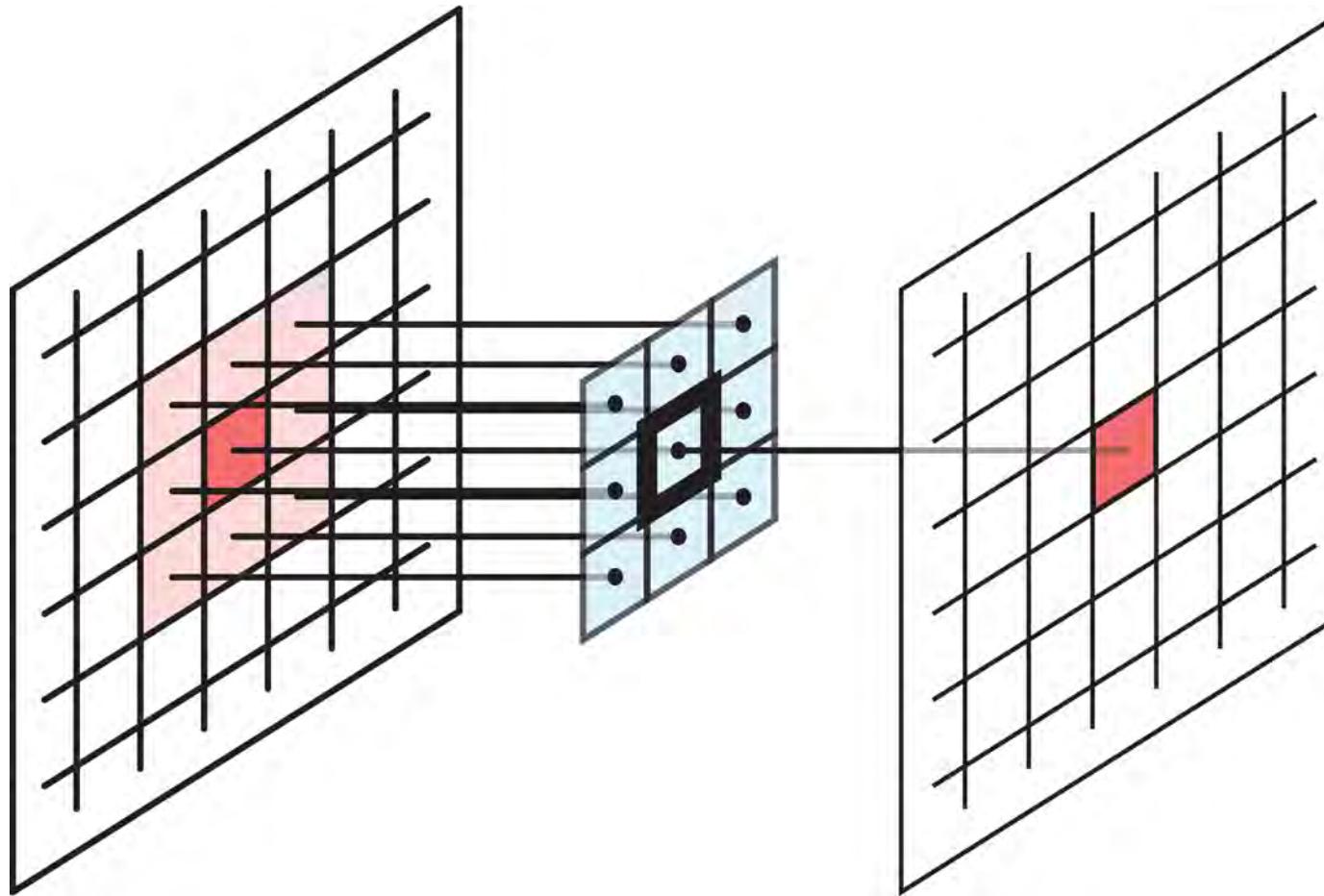
The neuron or its weights is called a **filter**. We *convolve* the image with a filter, i.e. a **convolutional filter**.

# Terminology

- The same neuron is used to sweep over the image, so we can store the weights in some shared memory and process the pixels in parallel. We say that the neurons are *weight sharing*.
- In the previous example, the neuron only takes one pixel as input. Usually a larger filter containing a *block of weights* is used to process not only a pixel but also its neighboring pixels all at once.
- The weights are called the filter **kernels**.
- The cluster of pixels that forms the input of a filter is called its *footprint*.



# Spatial filter



Example 3x3 filter

When a filter's footprint is  $> 1$  pixel, it is a **spatial filter**.



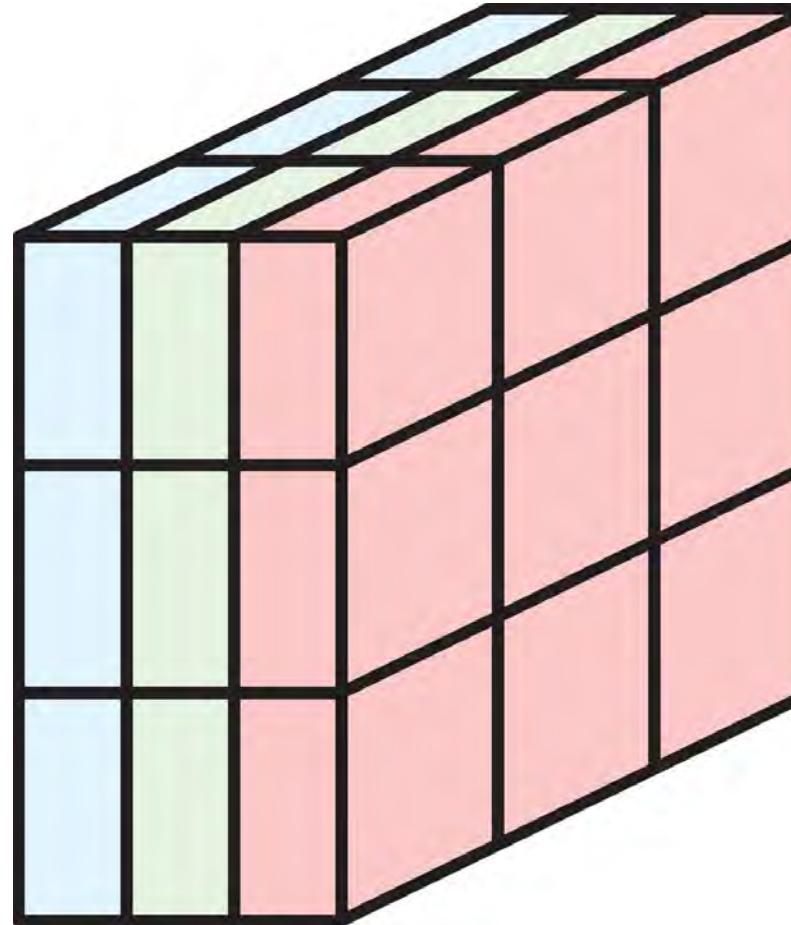
Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.



UNSW  
SYDNEY

# Multidimensional convolution

Need # Channels in Input = # Channels in Filter.



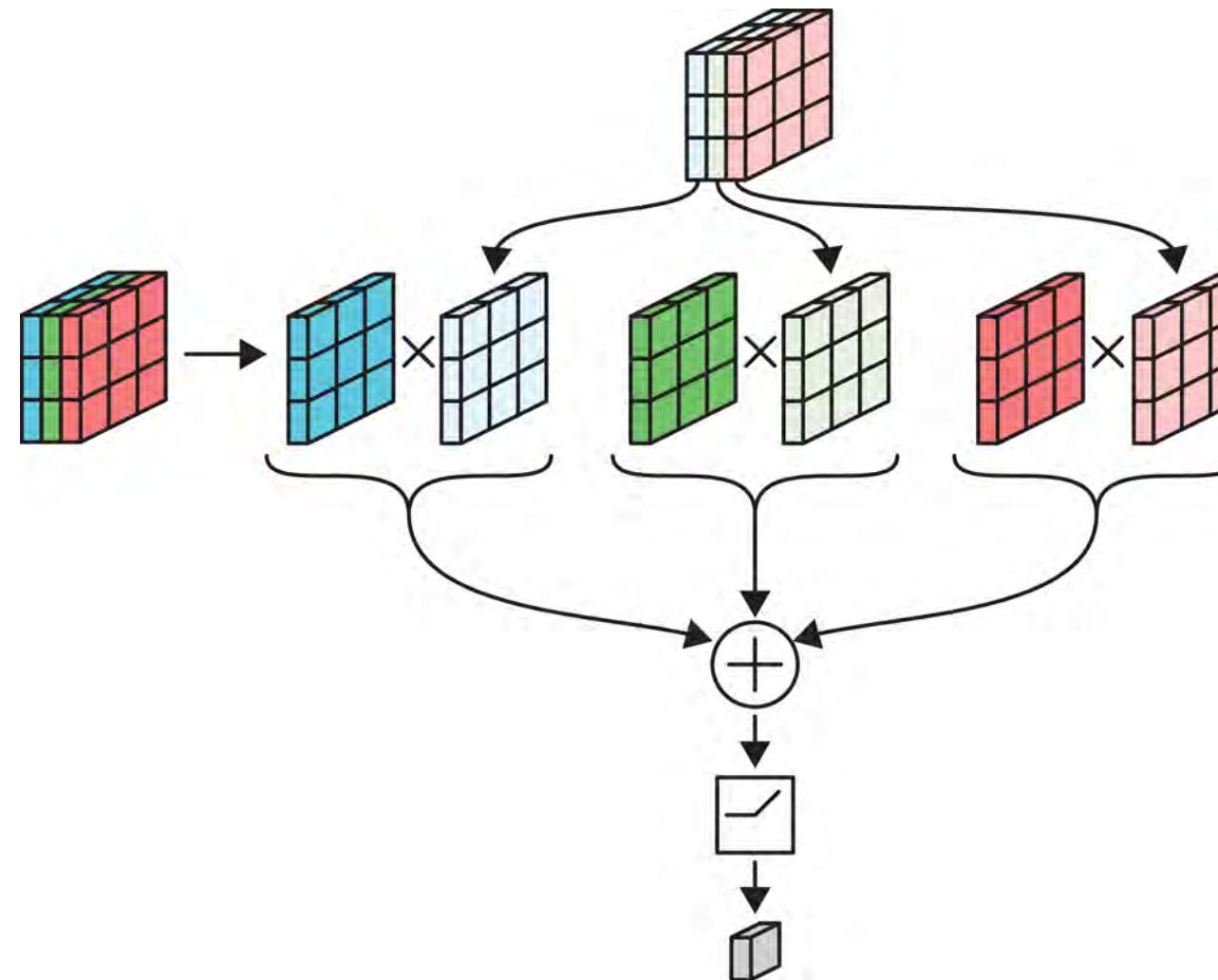
Example: a  $3 \times 3$  filter with 3 channels, containing 27 weights.



Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.



# Example: 3x3 filter over RGB input



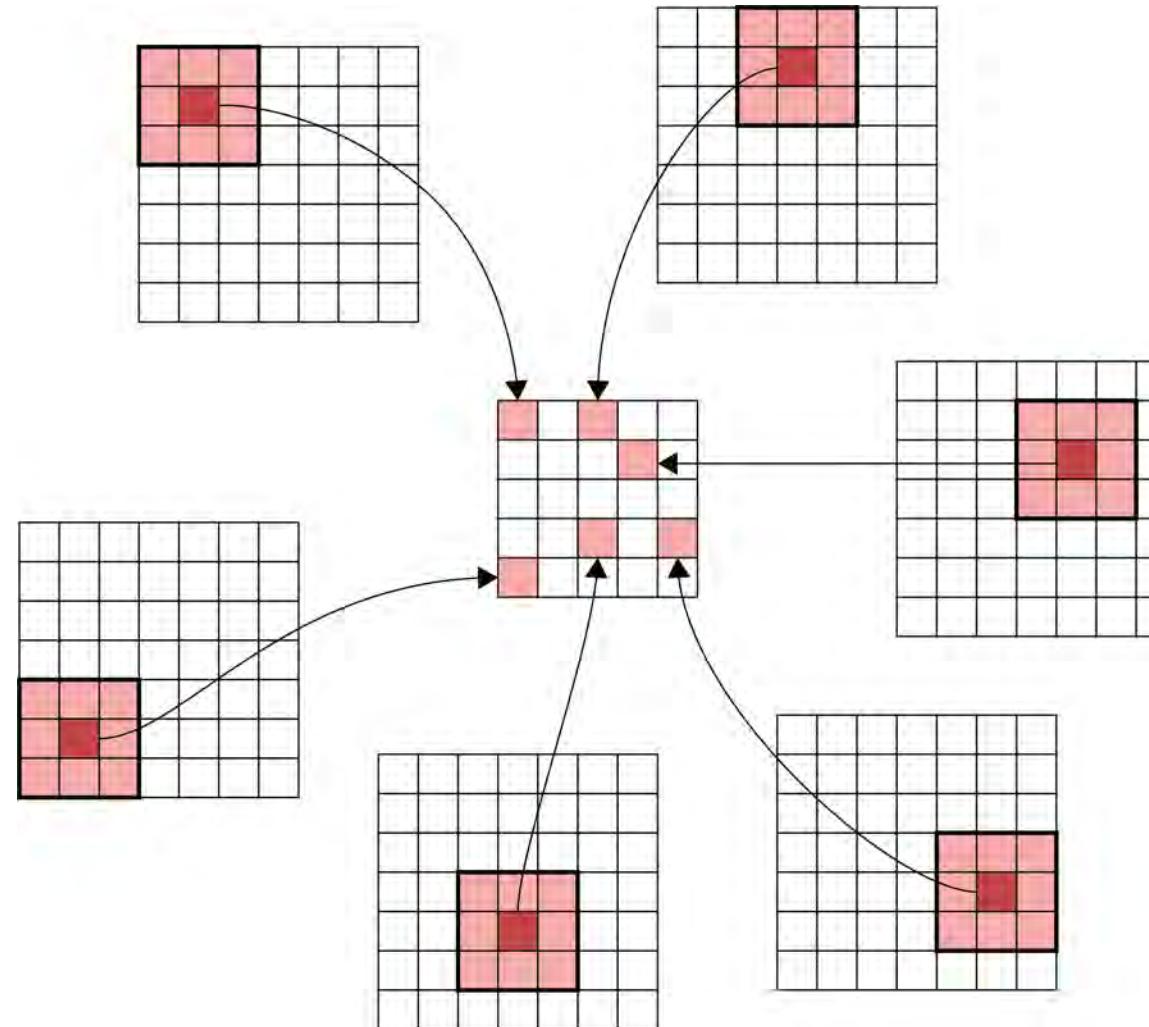
Each channel is multiplied separately & then added together.



Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.



# Input-output relationship



Matching the original image footprints against the output location.



Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.

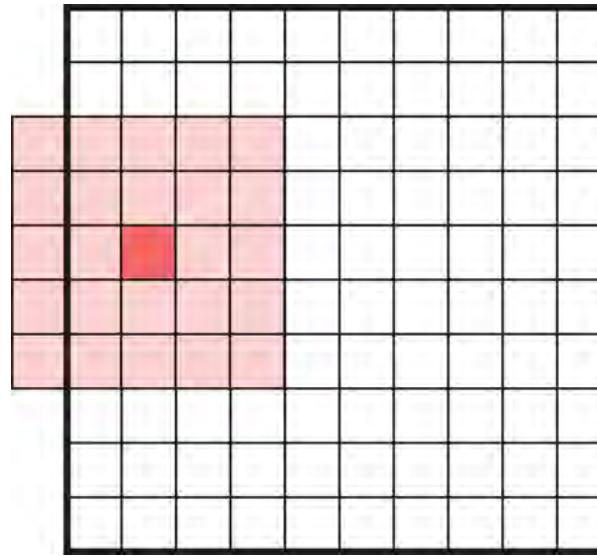


# Lecture Outline

- Images
- Convolutional Layers
- **Convolutional Layer Options**
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# Padding



What happens when filters go off the edge of the input?

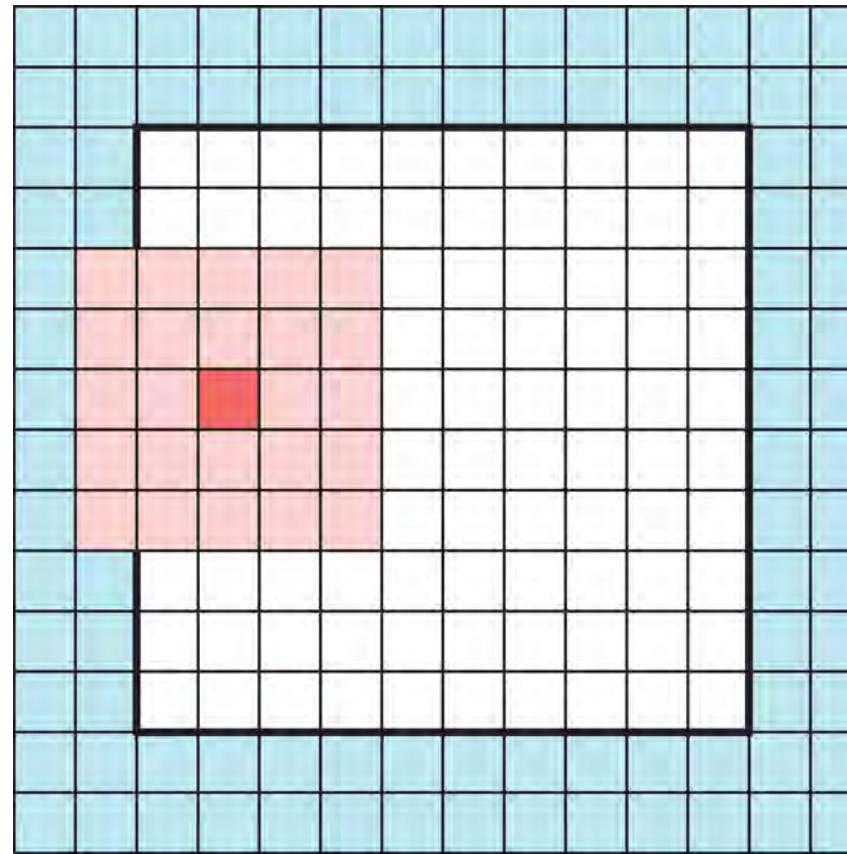
- How to avoid the filter's receptive field falling off the side of the input.
- If we only scan the filter over places of the input where the filter can fit perfectly, it will lead to loss of information, especially after many filters.



# Padding

Add a border of extra elements around the input, called **padding**.

Normally we place zeros in all the new elements, called **zero padding**.



Padded values can be added to the outside of the input.



Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.



UNSW  
SYDNEY

# Convolution layer

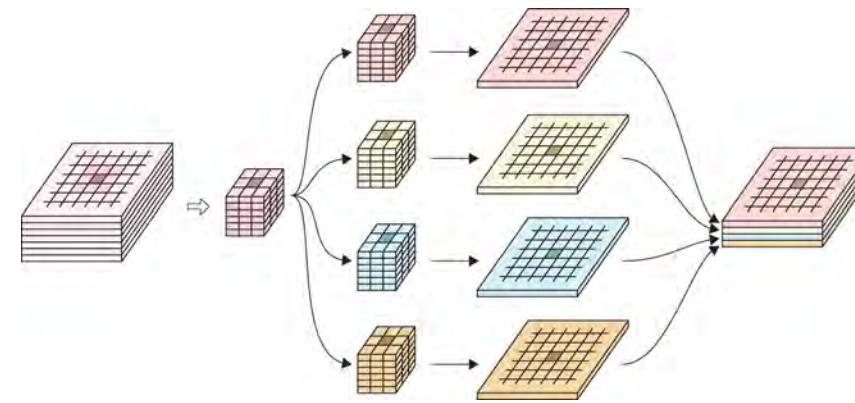
- Multiple filters are bundled together in one layer.
- The filters are applied *simultaneously* and *independently* to the input.
- Filters can have different footprints, but in practice we almost always use the same footprint for every filter in a convolution layer.
- Number of channels in the output will be the same as the number of filters.



# Example

In the image:

- 6-channel input tensor
- input pixels
- four  $3 \times 3$  filters
- four output tensors
- final output tensor.



Example network highlighting that the number of output channels equals the number of filters.

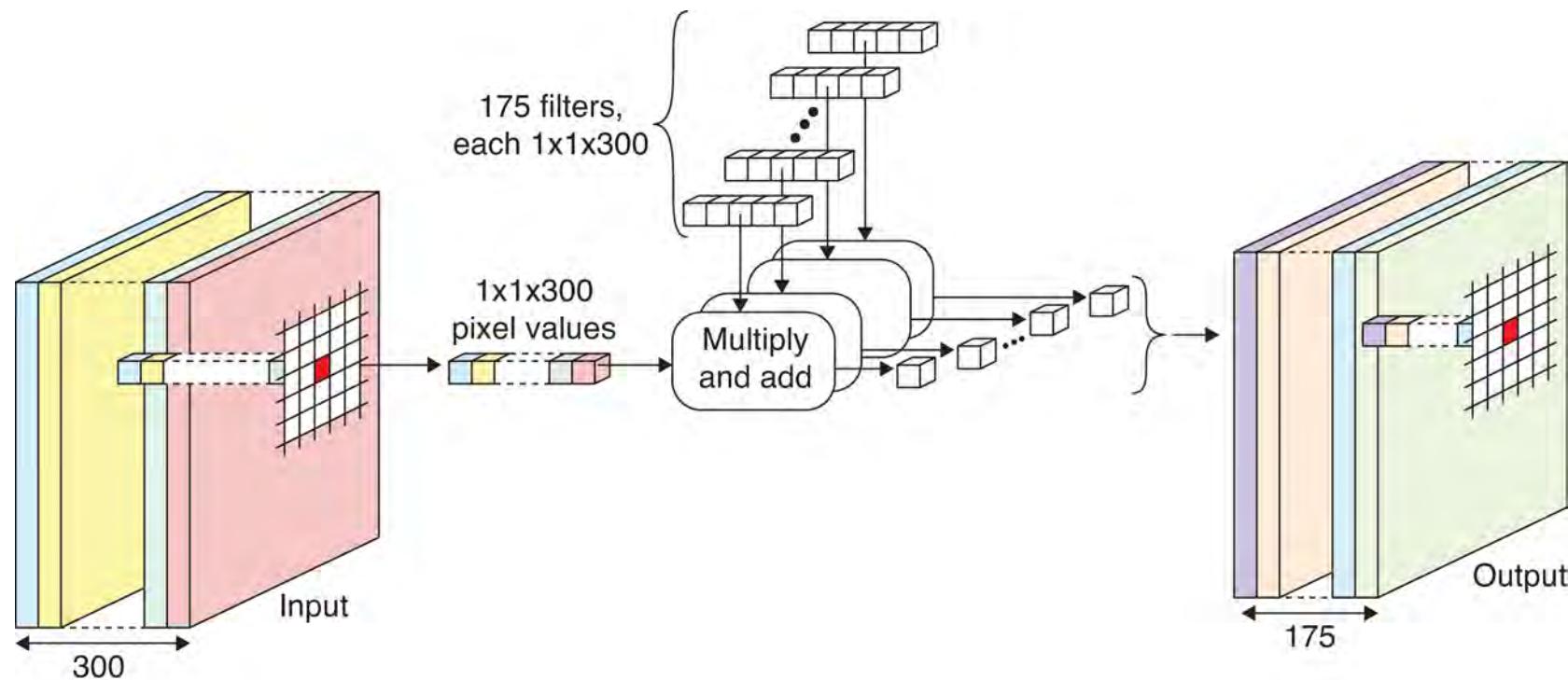


# 1x1 convolution

- Feature reduction: Reduce the number of channels in the input tensor (removing correlated features) by using fewer filters than the number of channels in the input. This is because the number of channels in the output is always the same as number of filters.
- 1x1 convolution: Convolution using 1x1 filters.
- When the channels are correlated, 1x1 convolution is very effective at reducing channels without loss of information.



# Example of 1x1 convolution



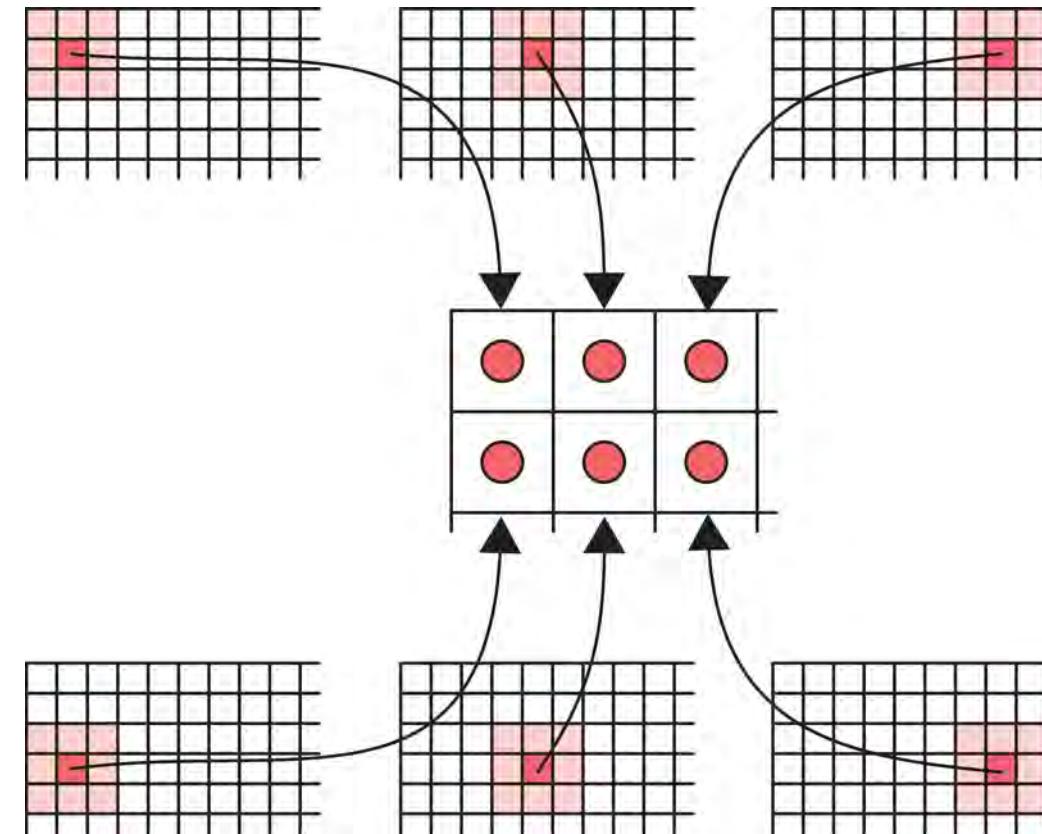
Example network with 1x1 convolution.

- Input tensor contains 300 channels.
- Use 175 1x1 filters in the convolution layer (300 weights each).
- Each filter produces a 1-channel output.
- Final output tensor has 175 channels.



# Striding

We don't have to go one pixel across/down at a time.



Example: Use a stride of three horizontally and two vertically.

Dimension of output will be smaller than input.

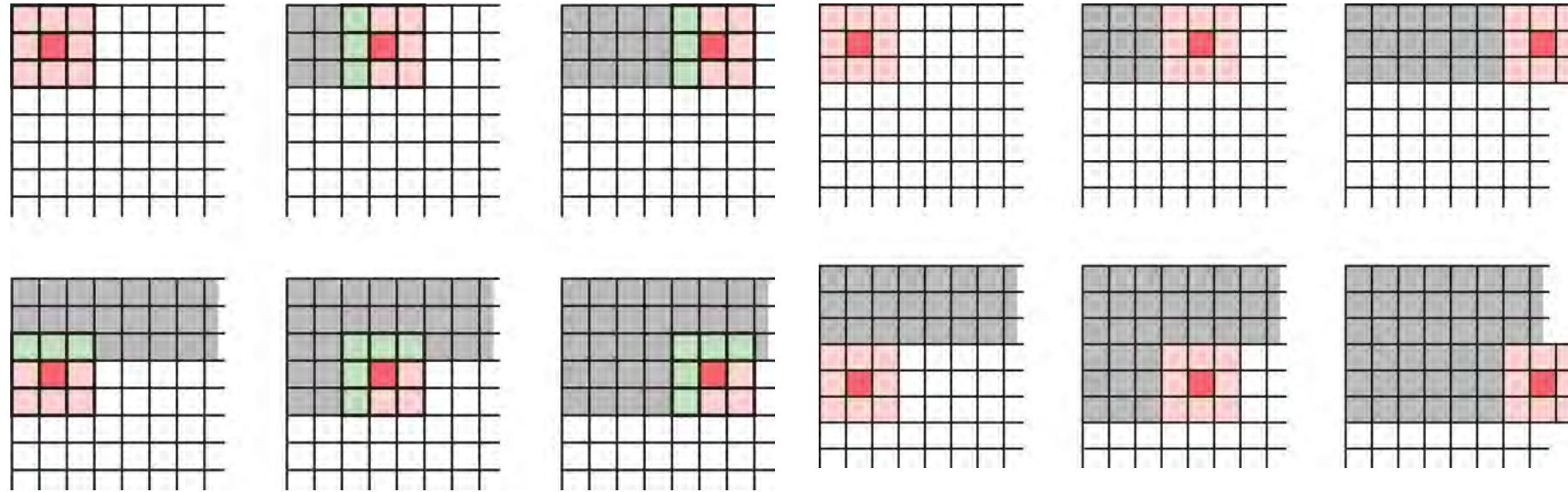


Source: Glassner (2021), *Deep Learning: A Visual Approach*, Chapter 16.



UNSW  
SYDNEY

# Choosing strides



When a filter scans the input step by step, it processes the same input elements multiple times. Even with larger strides, this can still happen (left image).

If we want to save time, we can choose strides that prevent input elements from being used more than once. Example (right image): 3x3 filter, stride 3 in both directions.



# Specifying a convolutional layer

Need to choose:

- number of filters,
- their footprints (e.g.  $3 \times 3$ ,  $5 \times 5$ , etc.),
- activation functions,
- padding & striding (optional).

All the filter weights are learned during training.



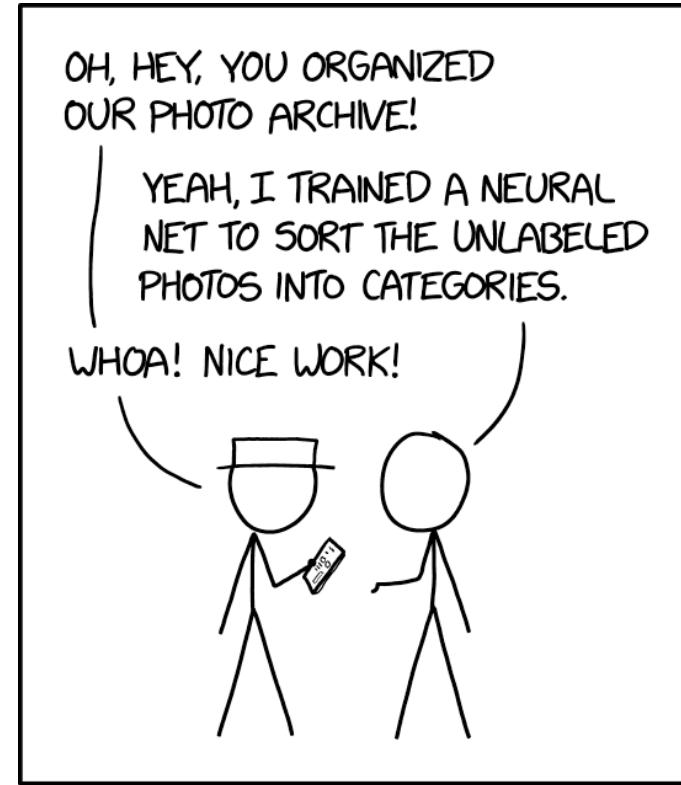
# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- **Convolutional Neural Networks**
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# Definition of CNN

A neural network that uses *convolution layers* is called a *convolutional neural network*.



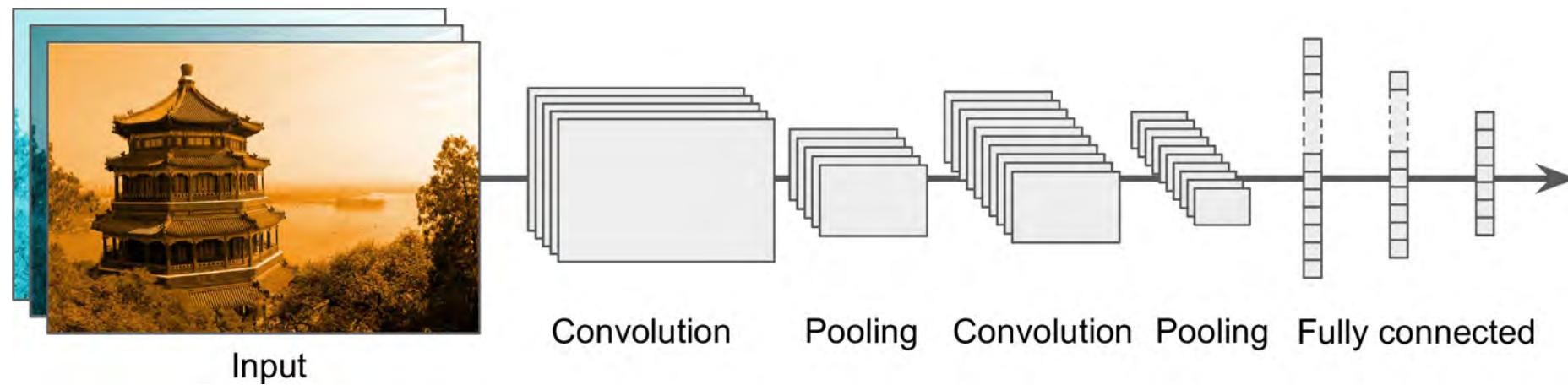
ENGINEERING TIP:  
WHEN YOU DO A TASK BY HAND,  
YOU CAN TECHNICALLY SAY YOU  
TRAINED A NEURAL NET TO DO IT.



Source: Randall Munroe (2019), [xkcd #2173: Trained a Neural Net](#).



# Architecture



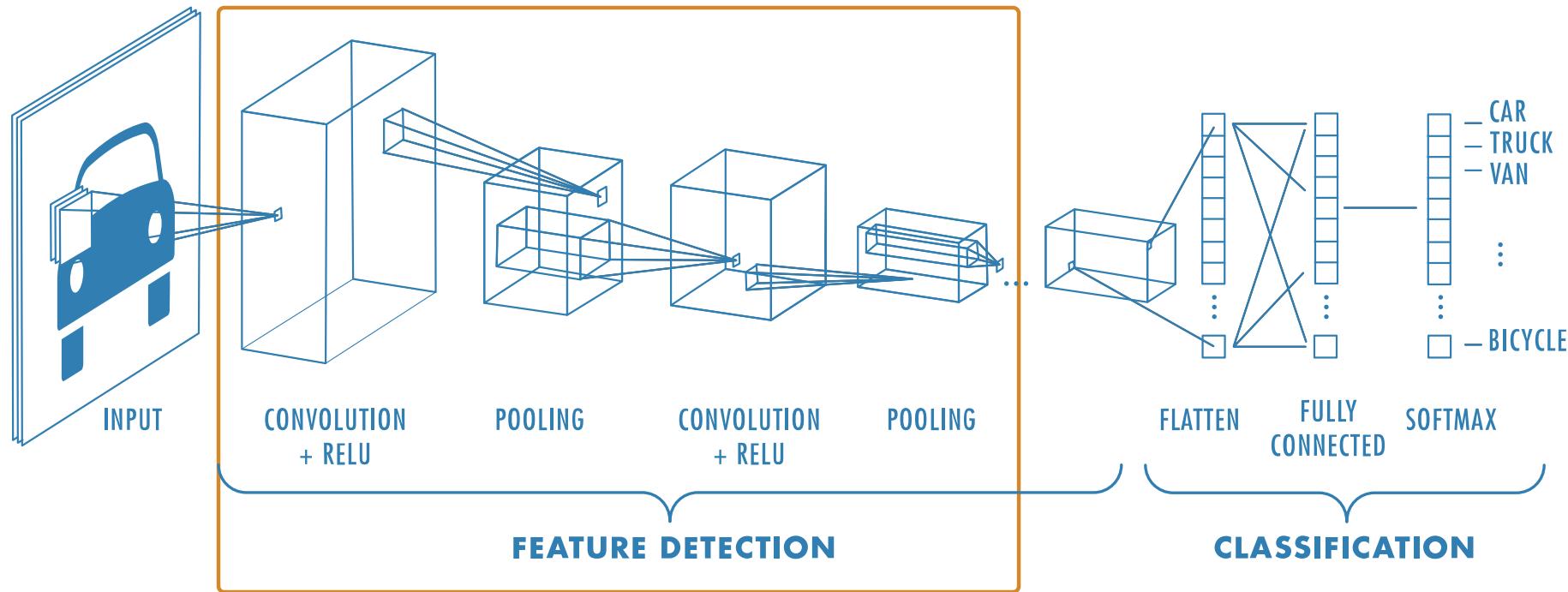
Typical CNN architecture.



Source: Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 14-11.



# Architecture #2



Source: MathWorks, *Introducing Deep Learning with MATLAB*, Ebook.



# Pooling

**Pooling**, or **downsampling**, is a technique to blur a tensor.

3	2	-3	2
1	6	20	5
4	-13	2	6
-2	3	9	3

(a)

3	2	-3	2
1	6	20	5
4	-13	2	6
-2	3	9	3

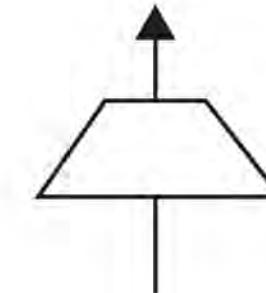
(b)

3	6
-2	5

Average

6	20
4	9

Max

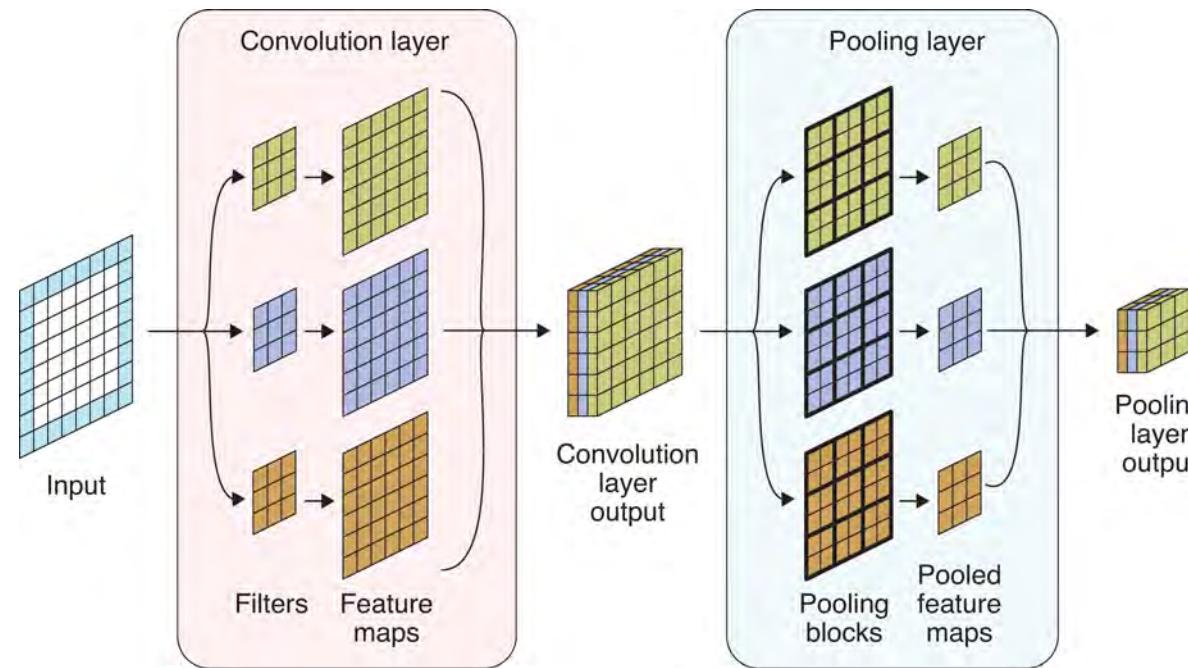


(e)

Illustration of pool operations.

(a): Input tensor (b): Subdivide input tensor into 2x2 blocks (c): Average pooling (d): Max pooling (e): Icon for a pooling layer

# Pooling for multiple channels



Pooling a multichannel input.

- Input tensor:  $6 \times 6$  with 1 channel, zero padding.
- Convolution layer: Three  $3 \times 3$  filters.
- Convolution layer output:  $6 \times 6$  with 3 channels.
- Pooling layer: apply max pooling to each channel.
- Pooling layer output:  $3 \times 3$ , 3 channels.



# Why/why not use pooling?

**Why?** Pooling *reduces the size* of tensors, therefore reduces memory usage and execution time (recall that  $1 \times 1$  convolution *reduces the number of channels* in a tensor).

## Why not?



u/geoffhinton • Commented on 7 years ago



You have many different questions. I shall number them and try to answer each one in a different reply.

1. What is your most controversial opinion in machine learning?

The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.

If the pools do not overlap, pooling loses valuable information about where things are. We need this information to detect precise relationships between the parts of an object. It's true that if the pools overlap enough, the positions of features will be accurately preserved by "coarse coding" (see my paper on "distributed representations" in 1986 for an explanation of this effect). But I no longer believe that coarse coding is the best way to represent the poses of objects relative to the viewer (by pose I mean position, orientation, and scale).

I think it makes much more sense to represent a pose as a small matrix that converts a vector of positional coordinates relative to the viewer into positional coordinates relative to the shape itself. This is what they do in computer graphics and it makes it easy to capture the effect of a change in viewpoint. It also explains why you cannot see a shape without imposing a rectangular coordinate frame on it, and if you impose a different frame, you cannot even recognize it as the same shape. Convolutional neural nets have no explanation for that, or at least none that I can think of.

Geoffrey Hinton

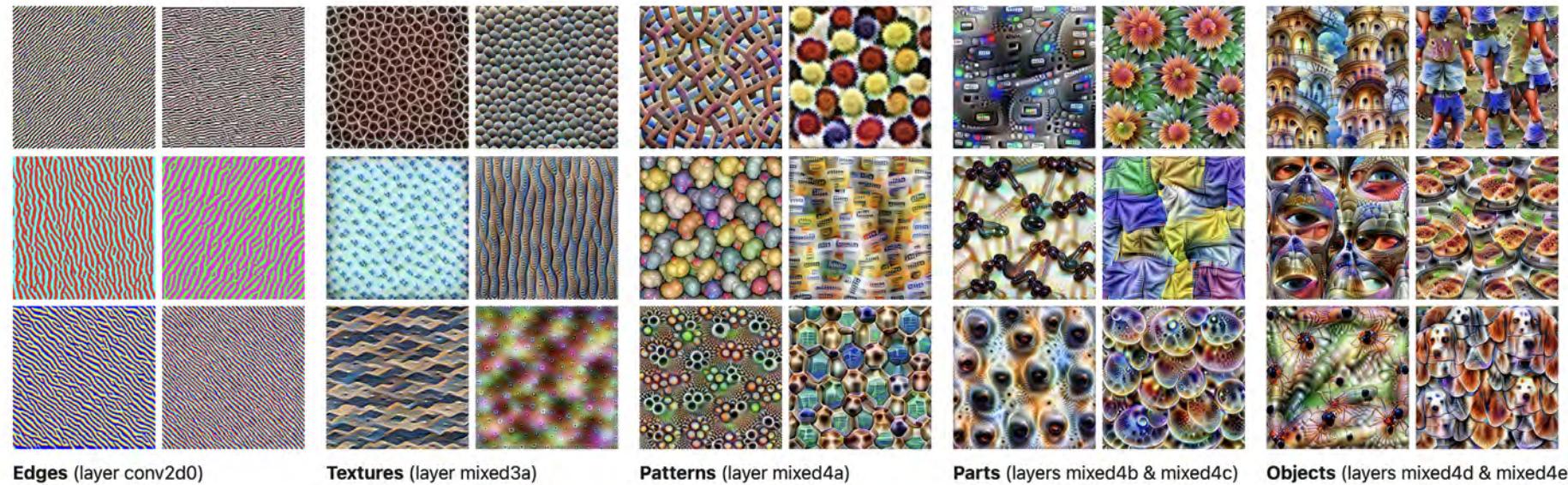


Source: Hinton, [Reddit AMA](#).



UNSW  
SYDNEY

# What do the CNN layers learn?



Source: Distill article, [Feature Visualization](#).



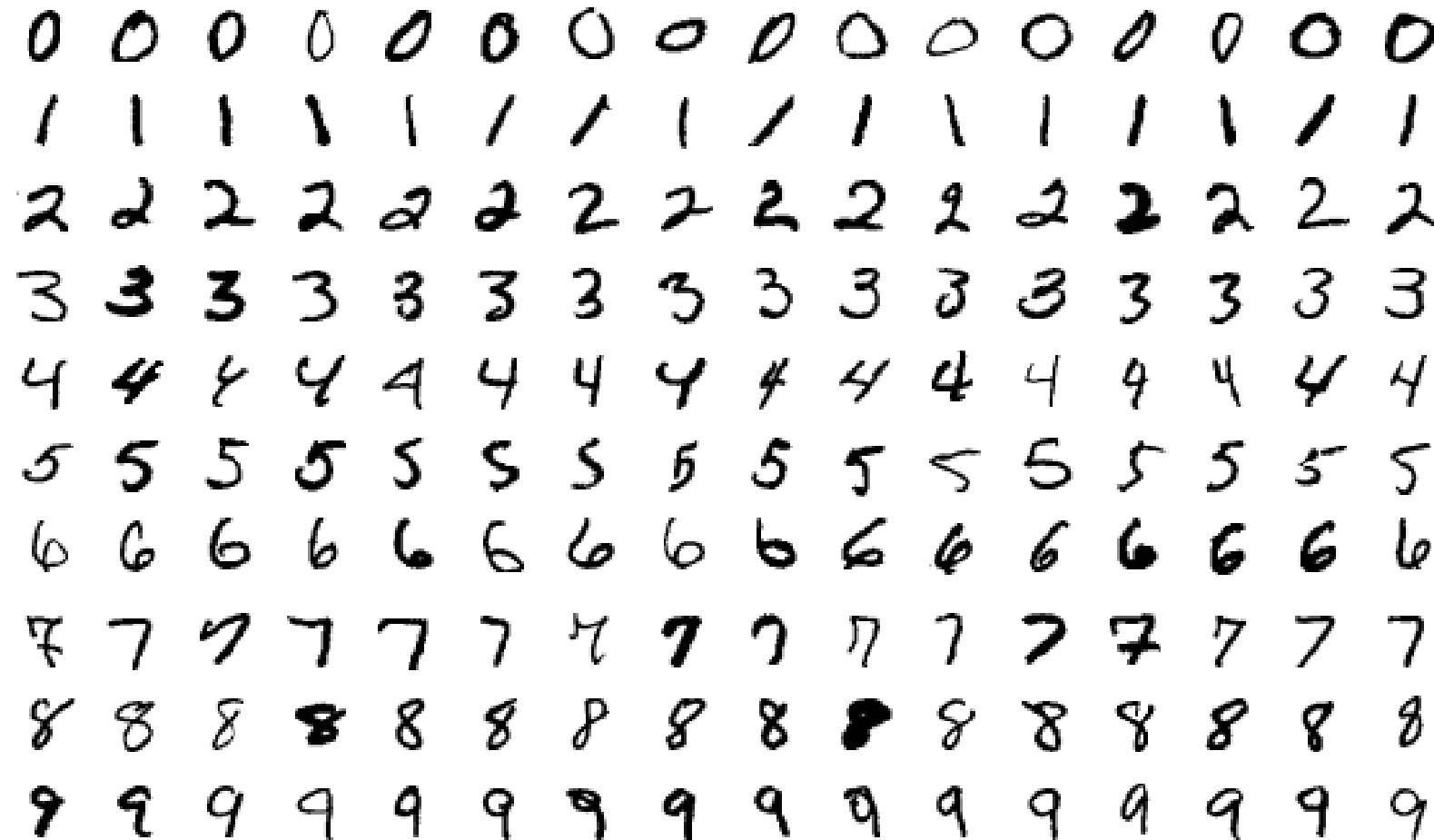
UNSW  
SYDNEY

# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- **Chinese Character Recognition Dataset**
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# MNIST Dataset



The MNIST dataset.



Source: Wikipedia, [MNIST database](#).



UNSW  
SYDNEY

# CASIA Chinese handwriting database

Dataset source: Institute of Automation of Chinese Academy of Sciences (CASIA)

躲采踩舵刹情墮蝶峨鵠  
僧窯泥燒瓦匣搥過鄧餽  
黑而兒百久領潤二或發  
罰錢付之閑法法藩帆垂  
龜山攀破鉢繁凡煩反返范  
敗紀飯泣坊芳方肪房丙  
妨彷彿綻放菲非啡飛肥  
匪诽味肺瘦讲葛才醉呻

A 13 GB dataset of 3,999,571 handwritten characters.

# Inspect a subset of characters

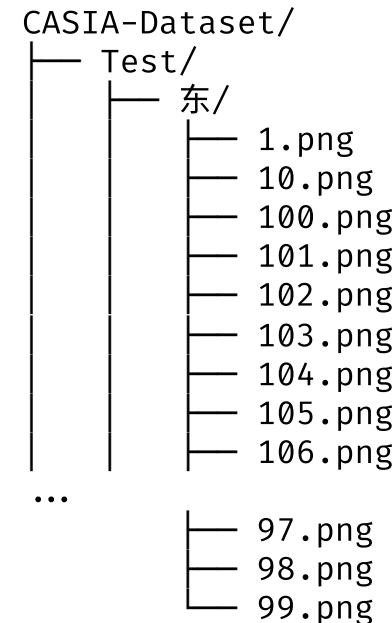
Pulling out 55 characters to experiment with.

人从众大夫天口太因鱼犬吠哭火炎  
 焉木林森本竹羊美羔山出女囡鸟日  
 东月朋明肉肤工白虎门闪问闲水牛  
 马吗妈玉王国主川舟虫

Inspect directory structure

```
1 !pip install directory_tree
```

```
1 from directory_tree import display_tree
2 display_tree("CASIA-Dataset")
```



# Count number of images for each character

```

1 def count_images_in_folders(root_folder):
2     counts = {}
3     for folder in root_folder.iterdir():
4         counts[folder.name] = len(list(folder.glob("*.png")))
5     return counts
6
7 train_counts = count_images_in_folders(Path("CASIA-Dataset/Train"))
8 test_counts = count_images_in_folders(Path("CASIA-Dataset/Test"))
9
10 print(train_counts)
11 print(test_counts)

```

```

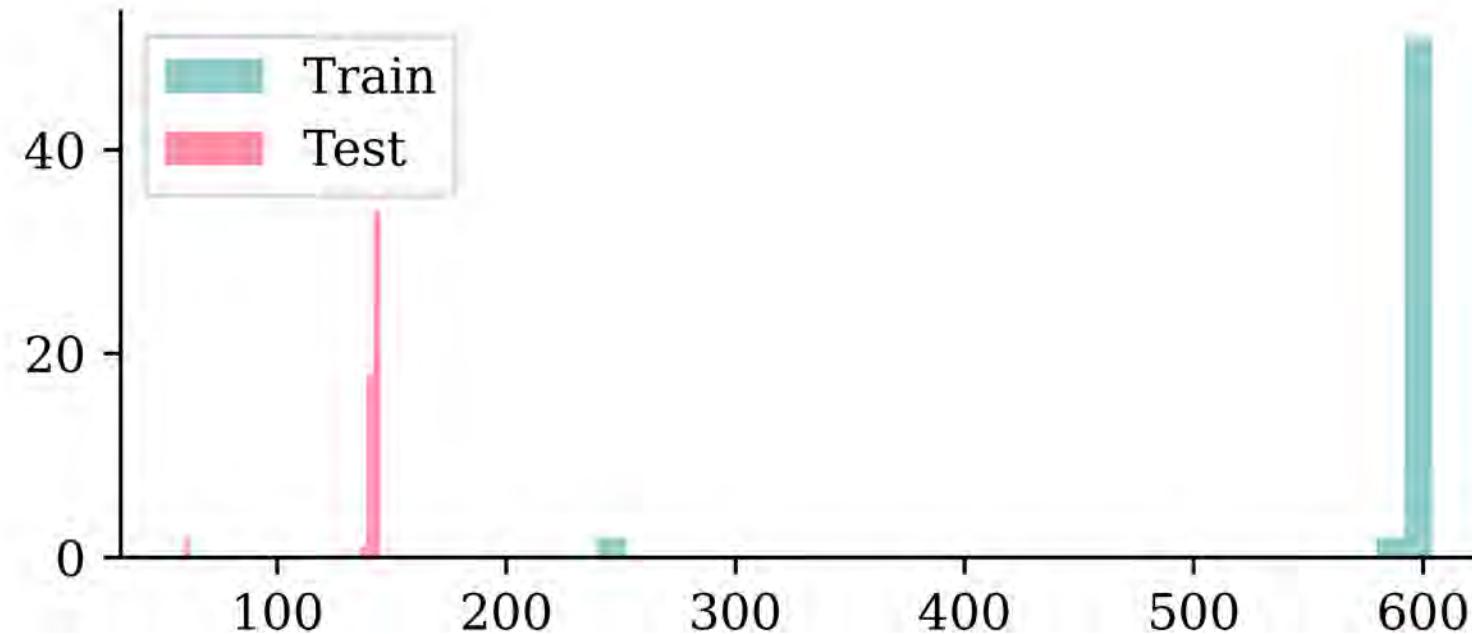
{'太': 596, '朋': 595, '羊': 600, '哭': 584, '囡': 240, '明': 596, '川': 593, '马': 597, '羔': 597, '天': 598, '吠': 601, '肉': 598, '夫': 599, '水': 597, '火': 599, '玉': 602, '妈': 595, '鸟': 598, '工': 600, '从': 598, '竹': 600, '王': 601, '人': 597, '美': 591, '众': 600, '因': 603, '东': 601, '大': 603, '吗': 596, '虫': 602, '日': 597, '内': 597, '啖': 240, '林': 598, '牛': 599, '舟': 601, '本': 604, '鱼': 602, '闪': 597, '山': 598, '口': 597, '主': 599, '炎': 602, '国': 600, '闲': 598, '问': 601, '犬': 598, '白': 604, '虎': 597, '出': 602, '森': 598, '肤': 601, '女': 597, '月': 604, '木': 598}
{'太': 143, '朋': 144, '羊': 144, '哭': 138, '囡': 59, '明': 144, '川': 142, '马': 144, '羔': 141, '天': 143, '吠': 141, '肉': 143, '夫': 141, '水': 143, '火': 142, '玉': 142, '妈': 142, '鸟': 143, '工': 141, '从': 142, '竹': 142, '王': 145, '人': 144, '美': 144, '众': 143, '因': 144, '东': 142, '大': 144, '吗': 143, '虫': 144, '日': 143, '内': 144, '啖': 60, '林': 143, '牛': 144, '舟': 143, '本': 143, '鱼': 143, '闪': 143, '山': 144, '口': 143, '主': 141, '炎': 143, '国': 142, '闲': 142, '问': 143, '犬': 141, '白': 141, '虎': 143, '出': 142, '森': 144, '肤': 140, '女': 144, '月': 144, '木': 144}

```



# Number of images for each character

```
1 plt.hist(train_counts.values(), bins=30, label="Train")
2 plt.hist(test_counts.values(), bins=30, label="Test")
3 plt.legend();
```



It differs, but basically ~600 training and ~140 test images per character. A couple of characters have a lot less of both though.



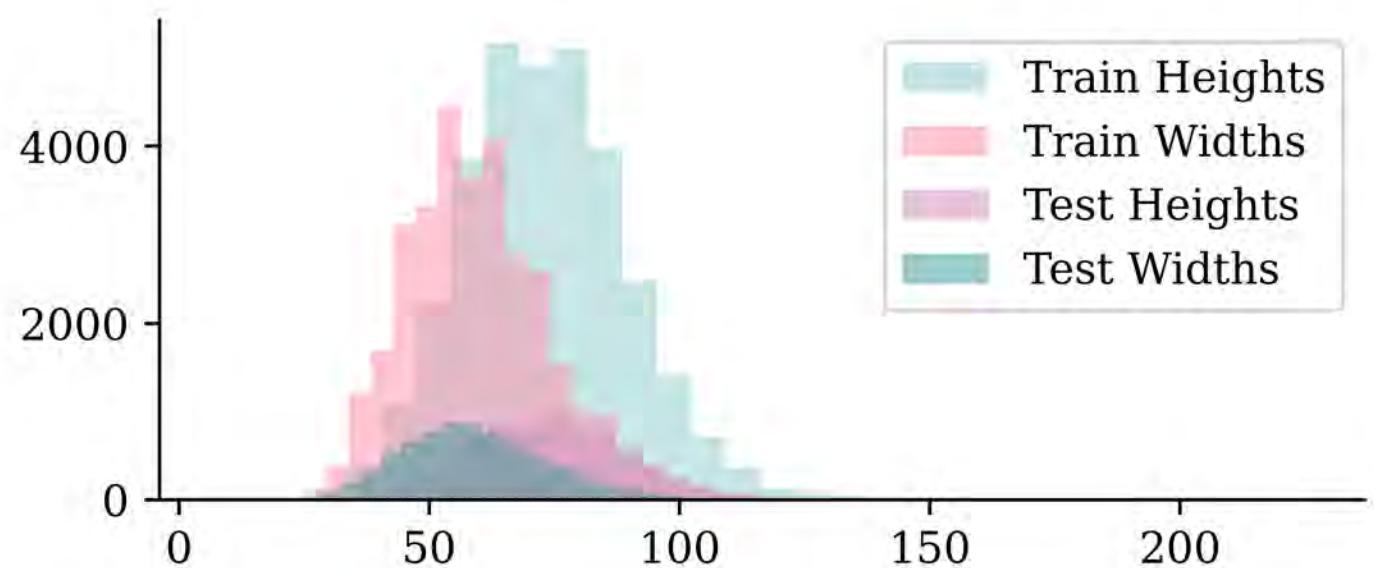
# Checking the dimensions

```
1 def get_image_dimensions(root_folder):
2     dimensions = []
3     for folder in root_folder.iterdir():
4         for image in folder.glob("*.png"):
5             img = imread(image)
6             dimensions.append(img.shape)
7     return dimensions
8
9 train_dimensions = get_image_dimensions(Path("CASIA-Dataset/Train"))
10 test_dimensions = get_image_dimensions(Path("CASIA-Dataset/Test"))
11
12 train_heights = [d[0] for d in train_dimensions]
13 train_widths = [d[1] for d in train_dimensions]
14 test_heights = [d[0] for d in test_dimensions]
15 test_widths = [d[1] for d in test_dimensions]
```



# Checking the dimensions II

```
1 plt.hist(train_heights, bins=30, alpha=0.5, label="Train Heights")
2 plt.hist(train_widths, bins=30, alpha=0.5, label="Train Widths")
3 plt.hist(test_heights, bins=30, alpha=0.5, label="Test Heights")
4 plt.hist(test_widths, bins=30, alpha=0.5, label="Test Widths")
5 plt.legend();
```

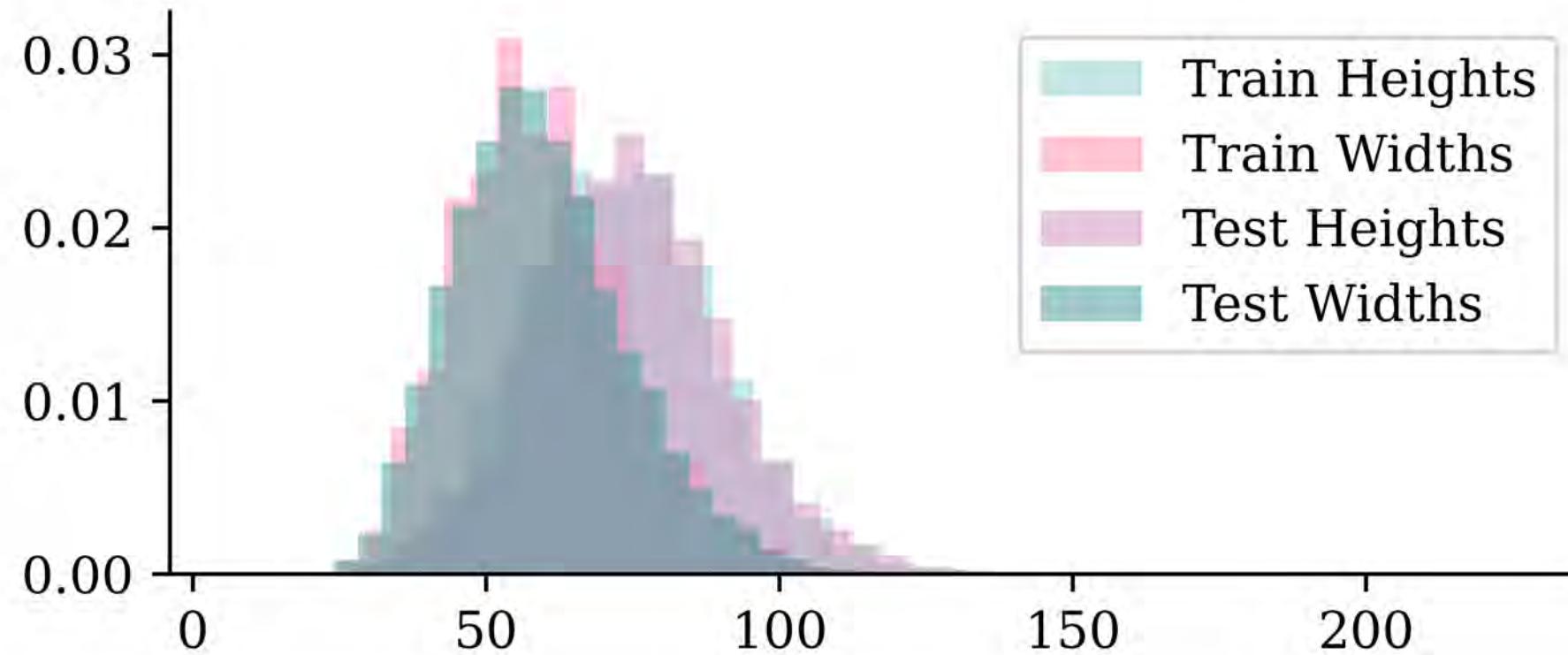


The images are taller than they are wide. We have more training images than test images.



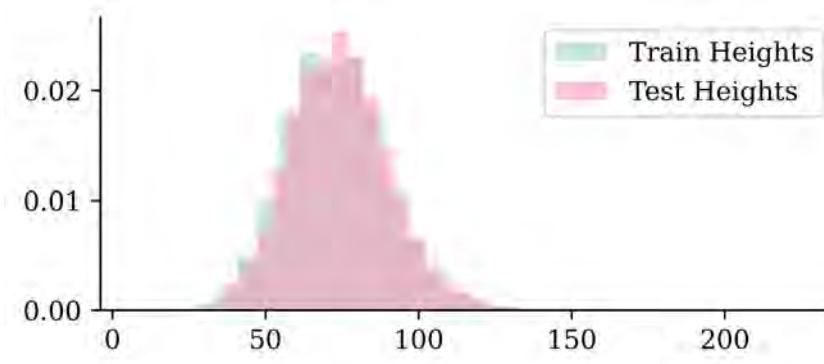
# Checking the dimensions III

```
1 plt.hist(train_heights, bins=30, alpha=0.5, label="Train Heights", density=True)
2 plt.hist(train_widths, bins=30, alpha=0.5, label="Train Widths", density=True)
3 plt.hist(test_heights, bins=30, alpha=0.5, label="Test Heights", density=True)
4 plt.hist(test_widths, bins=30, alpha=0.5, label="Test Widths", density=True)
5 plt.legend();
```

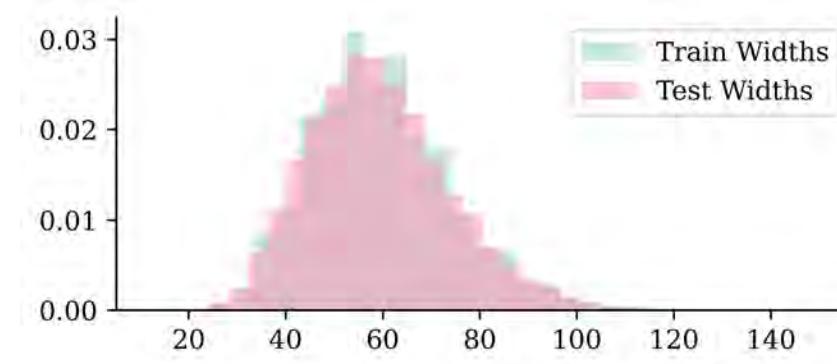


# Checking the dimensions IV

```
1 plt.hist(train_heights, bins=30, alpha=0.5)
2 plt.hist(test_heights, bins=30, alpha=0.5)
3 plt.legend();
```



```
1 plt.hist(train_widths, bins=30, alpha=0.5)
2 plt.hist(test_widths, bins=30, alpha=0.5)
3 plt.legend();
```



The distribution of dimensions are pretty similar between training and test sets.



# Keras image dataset loading

```
1 from keras.utils import image_dataset_from_directory  
2  
3 data_dir = "CASIA-Dataset"  
4 batch_size = 32  
5 img_height = 80  
6 img_width = 60  
7 img_size = (img_height, img_width)  
8  
9 train_ds = image_dataset_from_directory(  
10     data_dir + "/Train",  
11     image_size=img_size,  
12     batch_size=batch_size,  
13     shuffle=False,  
14     color_mode='grayscale')  
15  
16 test_ds = image_dataset_from_directory(  
17     data_dir + "/Test",  
18     image_size=img_size,  
19     batch_size=batch_size,  
20     shuffle=False,  
21     color_mode='grayscale')
```

Found 32206 files belonging to 55 classes.  
Found 7684 files belonging to 55 classes.



# Convert to numpy arrays

```
1 class_names = train_ds.class_names
2 print(class_names)
```

[ '东', '主', '人', '从', '众', '出', '口', '吗', '吠', '哭', '啖', '因', '囡', '国', '大', '天',  
 '太', '夫', '女', '妈', '山', '川', '工', '日', '明', '月', '朋', '木', '本', '林', '森', '水',  
 '火', '炎', '牛', '犬', '玉', '王', '白', '竹', '羊', '美', '羔', '肉', '肤', '舟', '虎', '虫',  
 '门', '闪', '问', '闲', '马', '鱼', '鸟' ]

```
1 # NB: Need shuffle=False earlier for these X & y to line up.
2 X_main = np.concatenate(list(train_ds.map(lambda x, y: x)))
3 y_main = np.concatenate(list(train_ds.map(lambda x, y: y)))
4
5 X_test = np.concatenate(list(test_ds.map(lambda x, y: x)))
6 y_test = np.concatenate(list(test_ds.map(lambda x, y: y)))
7
8 X_main.shape, y_main.shape, X_test.shape, y_test.shape
```

((32206, 80, 60, 1), (32206,), (7684, 80, 60, 1), (7684,))



# Some setup

```
1 X_train, X_val, y_train, y_val = train_test_split(X_main, y_main, test_size=0.2,
2     random_state=123)
3 print(X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shape)
```

(25764, 80, 60, 1) (25764,) (6442, 80, 60, 1) (6442,) (7684, 80, 60, 1) (7684,)

```
1 import matplotlib.font_manager as fm
2 CHINESE_FONT = fm.FontProperties(fname="STHeitiTC-Medium-01.ttf")
3
4 def plot_mandarin_characters(X, y, class_names, n=5, title_font=CHINESE_FONT):
5     # Plot the first n images in X
6     plt.figure(figsize=(10, 4))
7     for i in range(n):
8         plt.subplot(1, n, i + 1)
9         plt.imshow(X[i], cmap="gray")
10        plt.title(class_names[y[i]], fontproperties=title_font)
11        plt.axis("off")
```

1 class\_names[:5]

['东', '主', '人', '从', '众']

```
1 X_dong = X_train[y_train == 0]; y_dong = y_train[y_train == 0]
2 X_ren = X_train[y_train == 2]; y_ren = y_train[y_train == 2]
```



# Plotting some training characters

东 东 东 东 东

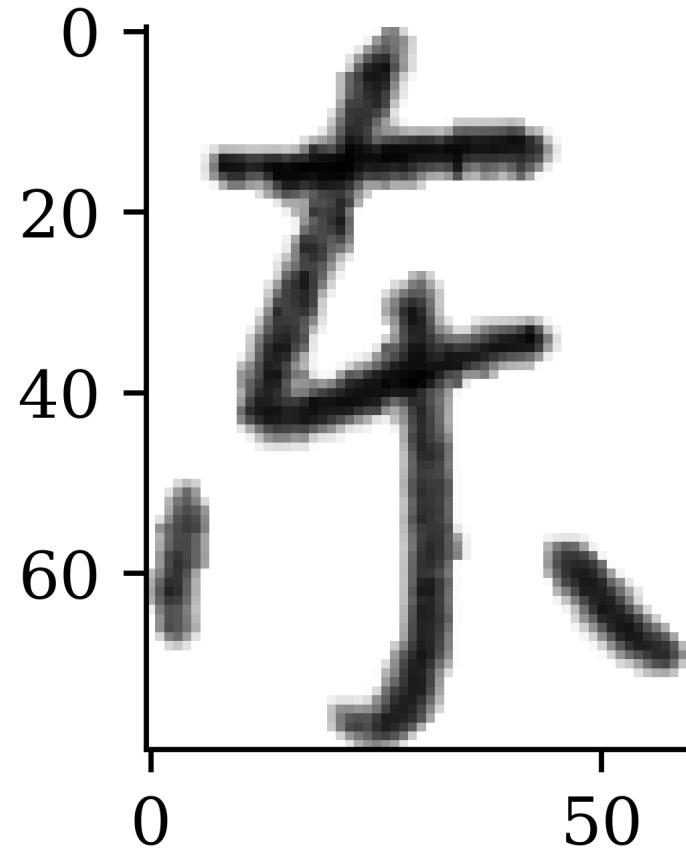


人 人 人 人 人

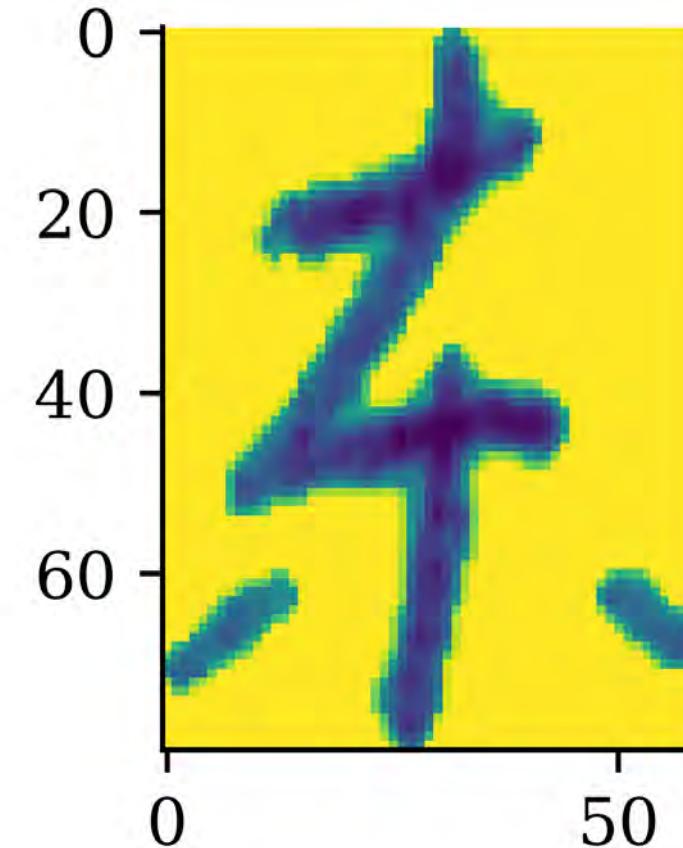


# Without the colourmap..

```
1 dong = X_test[y_test == 0][0]  
2 plt.imshow(dong, cmap="gray");
```



```
1 dong = X_test[y_test == 0][1]  
2 plt.imshow(dong);
```



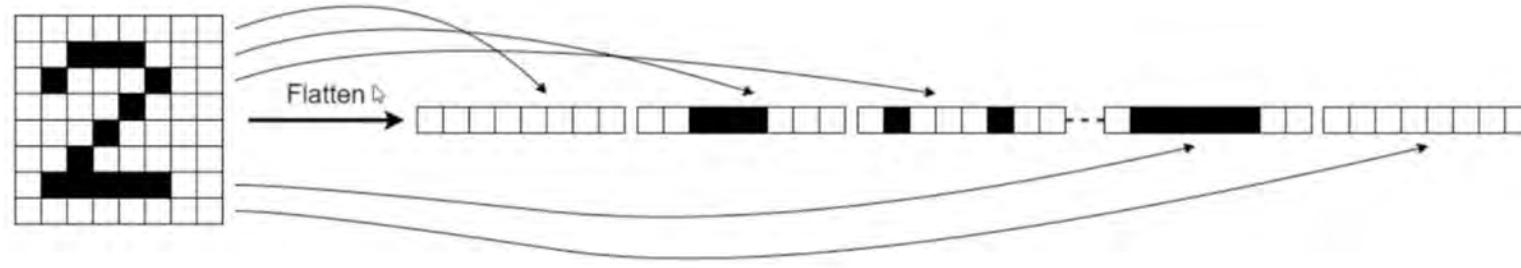
# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- **Fitting a (multinomial) logistic regression**
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# Make a logistic regression

## Flattening



Basically pretend it's not an image

```
1 from keras.layers import Rescaling, Flatten  
2  
3 num_classes = np.unique(y_train).shape[0]  
4 random.seed(123)  
5 model = Sequential([  
6     Input((img_height, img_width, 1)), Flatten(), Rescaling(1./255),  
7     Dense(num_classes, activation="softmax")  
8 ])
```



### Tip

The **Rescaling** layer will rescale the intensities to [0, 1].

# Inspecting the model

```
1 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 4800)	0
rescaling (Rescaling)	(None, 4800)	0
dense (Dense)	(None, 55)	264,055

Total params: 264,055 (1.01 MB)

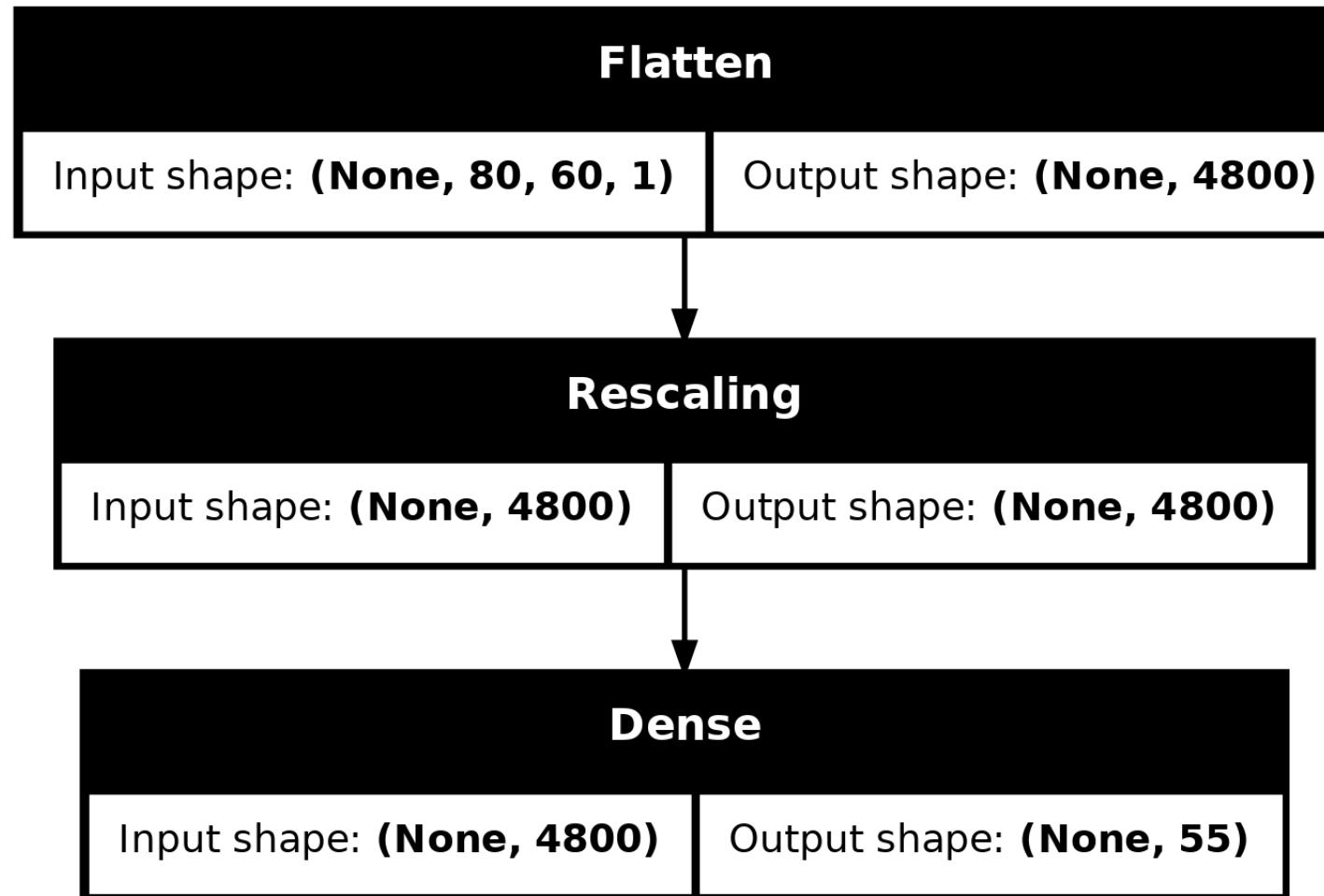
Trainable params: 264,055 (1.01 MB)

Non-trainable params: 0 (0.00 B)



# Plot the model

```
1 plot_model(model, show_shapes=True)
```



# Fitting the model

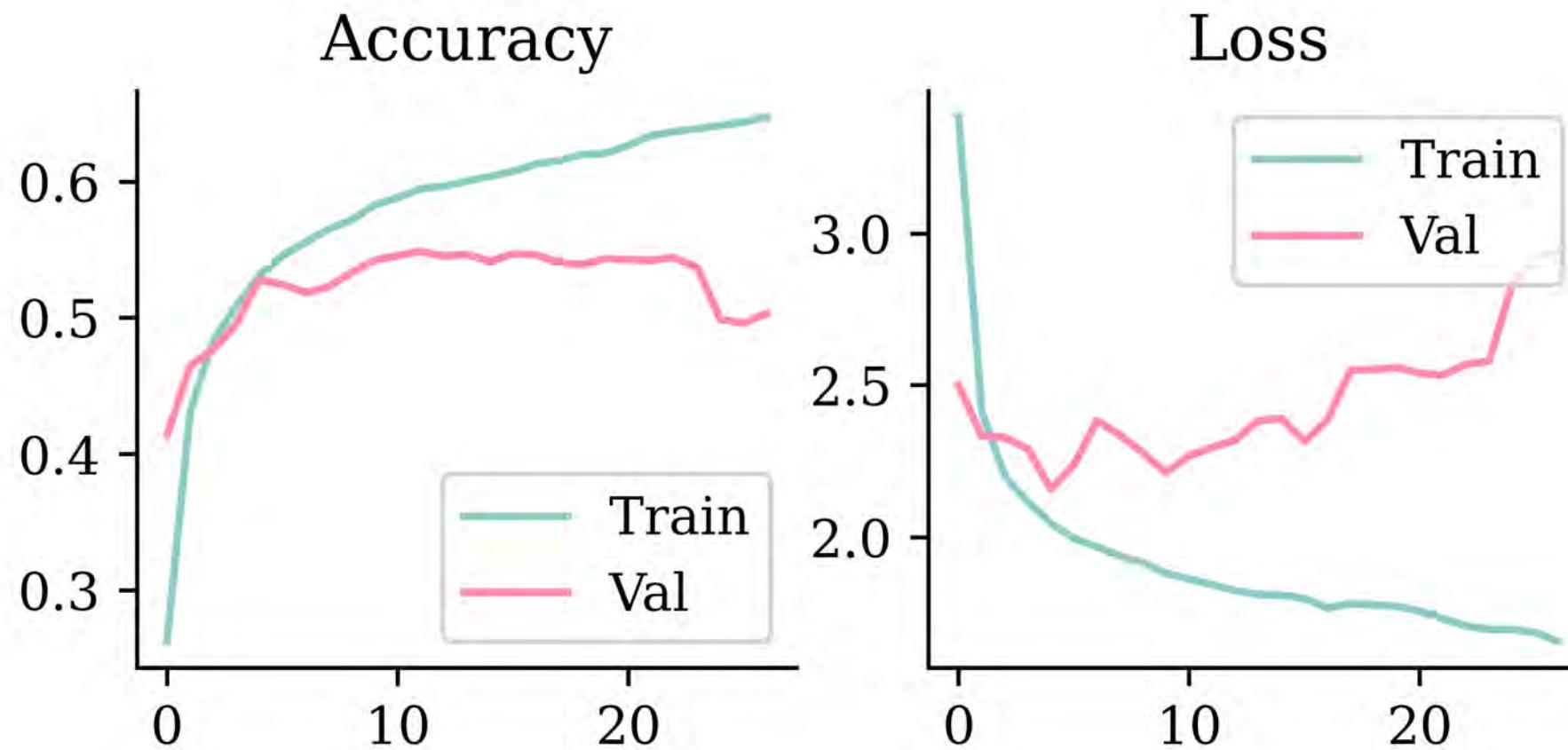
```
1 loss = keras.losses.SparseCategoricalCrossentropy()
2 topk = keras.metrics.SparseTopKCategoricalAccuracy(k=5)
3 model.compile(optimizer='adam', loss=loss, metrics=['accuracy', topk])
4
5 epochs = 100
6 es = EarlyStopping(patience=15, restore_best_weights=True,
7     monitor="val_accuracy", verbose=2)
8
9 if Path("logistic.keras").exists():
10    model = keras.models.load_model("logistic.keras")
11    with open("logistic_history.json", "r") as json_file:
12        history = json.load(json_file)
13 else:
14    hist = model.fit(X_train, y_train, validation_data=(X_val, y_val),
15        epochs=epochs, callbacks=[es], verbose=0)
16    model.save("logistic.keras")
17    history = hist.history
18    with open("logistic_history.json", "w") as json_file:
19        json.dump(history, json_file)
```

Most of this last part is just to save time rendering this slides, you don't need it.



# Plot the loss/accuracy curves

```
1 plot_history(history)
```



# Look at the metrics

```
1 print(model.evaluate(X_train, y_train, verbose=0))
2 print(model.evaluate(X_val, y_val, verbose=0))
```

```
[1.7625155448913574, 0.6105030179023743, 0.8532060384750366]
[2.2966670989990234, 0.5490530729293823, 0.8084445595741272]
```

```
1 loss_value, accuracy, top5_accuracy = model.evaluate(X_test, y_test, verbose=0)
2 print(f"Validation Loss: {loss_value:.4f}")
3 print(f"Validation Accuracy: {accuracy:.4f}")
4 print(f"Validation Top 5 Accuracy: {top5_accuracy:.4f}")
```

```
Validation Loss: 3.4015
Validation Accuracy: 0.4771
Validation Top 5 Accuracy: 0.7898
```



# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- **Fitting a CNN**
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# Make a CNN

```
1 from keras.layers import Conv2D, MaxPooling2D
2
3 random.seed(123)
4
5 model = Sequential([
6     Input((img_height, img_width, 1)),
7     Rescaling(1./255),
8     Conv2D(16, 3, padding="same", activation="relu", name="conv1"),
9     MaxPooling2D(name="pool1"),
10    Conv2D(32, 3, padding="same", activation="relu", name="conv2"),
11    MaxPooling2D(name="pool2"),
12    Conv2D(64, 3, padding="same", activation="relu", name="conv3"),
13    MaxPooling2D(name="pool3", pool_size=(4, 4)),
14    Flatten(), Dense(64, activation="relu"), Dense(num_classes)
15])
```



Architecture inspired by <https://www.tensorflow.org/tutorials/images/classification>.



# Inspect the model

```
1 model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 80, 60, 1)	0
conv1 (Conv2D)	(None, 80, 60, 16)	160
pool1 (MaxPooling2D)	(None, 40, 30, 16)	0
conv2 (Conv2D)	(None, 40, 30, 32)	4,640
pool2 (MaxPooling2D)	(None, 20, 15, 32)	0
conv3 (Conv2D)	(None, 20, 15, 64)	18,496
pool3 (MaxPooling2D)	(None, 5, 3, 64)	0
flatten_1 (Flatten)	(None, 960)	0
dense_1 (Dense)	(None, 64)	61,504
dense_2 (Dense)	(None, 55)	3,575

Total params: 88,375 (345.21 KB)

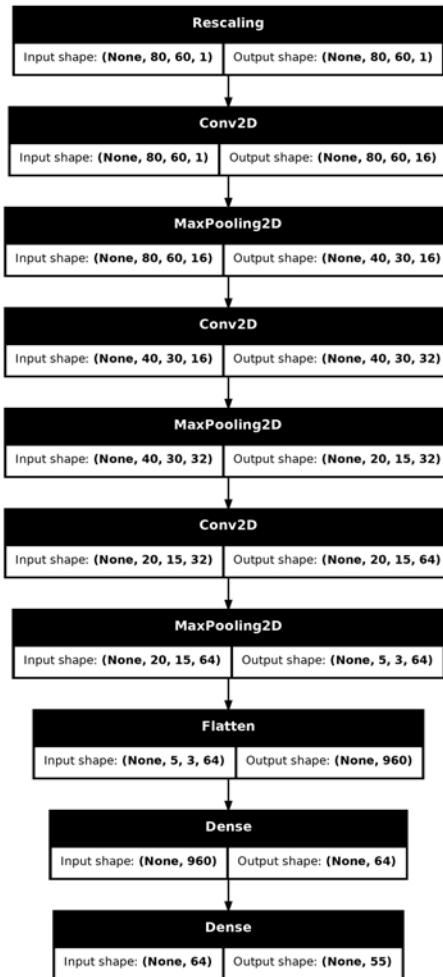
Trainable params: 88,375 (345.21 KB)

Non-trainable params: 0 (0.00 B)



# Plot the CNN

```
1 plot_model(model, show_shapes=True)
```



# Fit the CNN

```
1 loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
2 topk = keras.metrics.SparseTopKCategoricalAccuracy(k=5)
3 model.compile(optimizer='adam', loss=loss, metrics=['accuracy', topk])
4
5 epochs = 100
6 es = EarlyStopping(patience=15, restore_best_weights=True,
7     monitor="val_accuracy", verbose=2)
8
9 if Path("cnn.keras").exists():
10    model = keras.models.load_model("cnn.keras")
11    with open("cnn_history.json", "r") as json_file:
12        history = json.load(json_file)
13 else:
14    hist = model.fit(X_train, y_train, validation_data=(X_val, y_val),
15        epochs=epochs, callbacks=[es], verbose=0)
16    model.save("cnn.keras")
17    history = hist.history
18    with open("cnn_history.json", "w") as json_file:
19        json.dump(history, json_file)
```



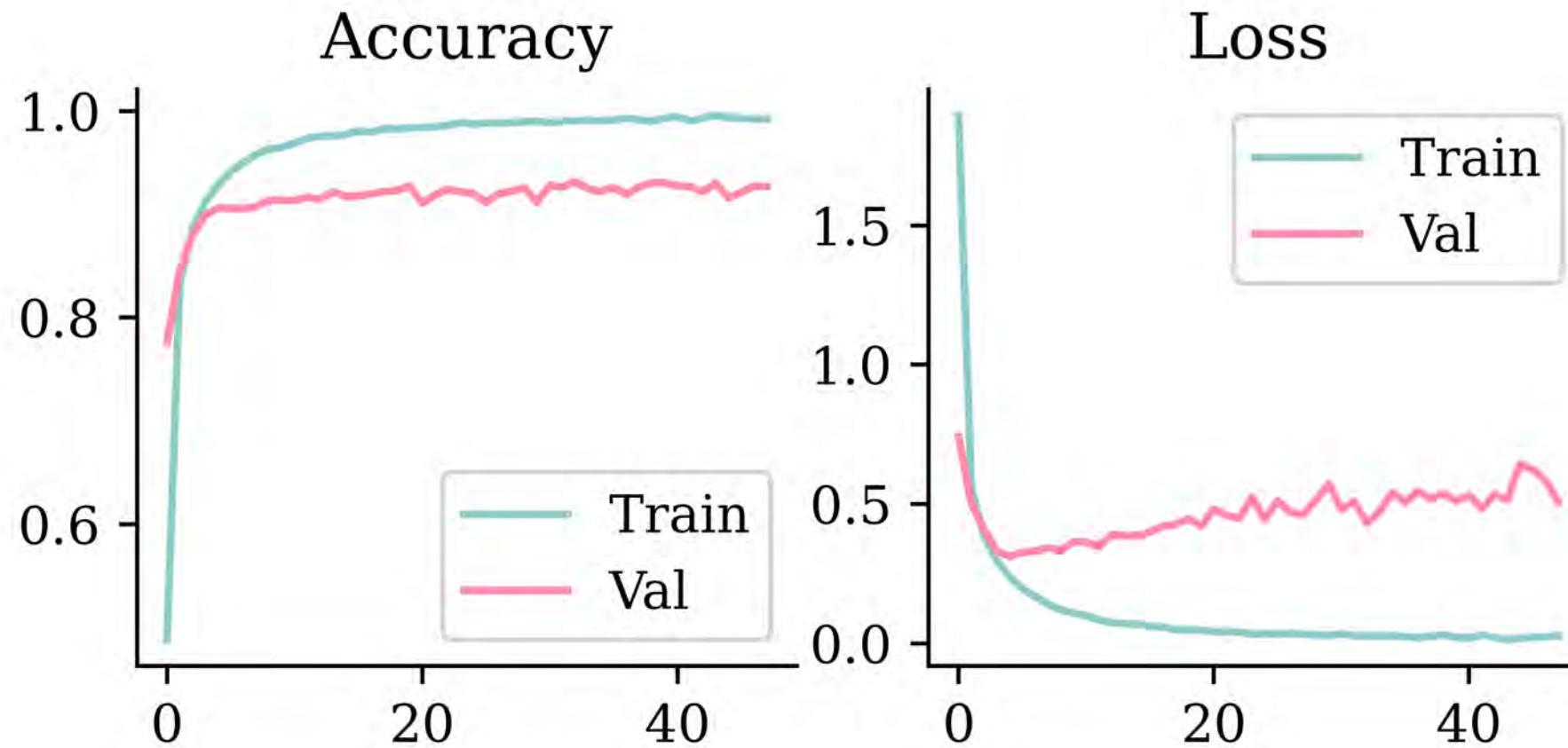
## Tip

Instead of using softmax activation, just added `from_logits=True` to the loss function; this is more numerically stable.



# Plot the loss/accuracy curves

```
1 plot_history(history)
```



# Look at the metrics

```
1 print(model.evaluate(X_train, y_train, verbose=0))
2 print(model.evaluate(X_val, y_val, verbose=0))
```

```
[0.01601474918425083, 0.9946824908256531, 1.0]
[0.4305051267147064, 0.9318534731864929, 0.9947221279144287]
```

```
1 loss_value, accuracy, top5_accuracy = model.evaluate(X_test, y_test, verbose=0)
2 print(f"Validation Loss: {loss_value:.4f}")
3 print(f"Validation Accuracy: {accuracy:.4f}")
4 print(f"Validation Top 5 Accuracy: {top5_accuracy:.4f}")
```

```
Validation Loss: 0.8504
Validation Accuracy: 0.8860
Validation Top 5 Accuracy: 0.9858
```



# Make a prediction

```
1 model.predict(X_test[0], verbose=0);
```

Exception encountered when calling MaxPooling2D.call().

**Negative dimension size caused by subtracting 2 from 1 for '{{node sequential\_1\_1/pool1\_1/MaxPo**

Arguments received by MaxPooling2D.call():

- inputs=tf.Tensor(shape=(32, 60, 1, 16), dtype=float32)

```
1 X_val[0].shape, X_val[0][np.newaxis, :].shape, X_val[[0]].shape
```

((80, 60, 1), (1, 80, 60, 1), (1, 80, 60, 1))

```
1 model.predict(X_val[[0]], verbose=0)
```

```
array([[-27.75, -33.43, -61.25, -45.47, -16.26, -18.03, -38.82, -8.48,
       -24.19, -41.46, -14.05, -1.9 , -11.72, -15.72, -55.43, -49.13,
       -20.48, -32.75, -9.72, -0.97, -37.01, -54.47, -64.23, -30.72,
       -14. , -36.27, -24.38, -42.4 , -18.55, -15.39, -25.44, -31.77,
       -40.55, -20.16, -43.67, -46.77, -31.43, -38.17, -5.83, -19.47,
       -72.93, -60.61, -49.84,  20.21, -24.8 , -19.48,  5.86, -4.55,
       -37.49, -9.48, -12.66, -6.69, -36.01, -9.77, -11.05]],
      dtype=float32)
```



# Predict on the test set II

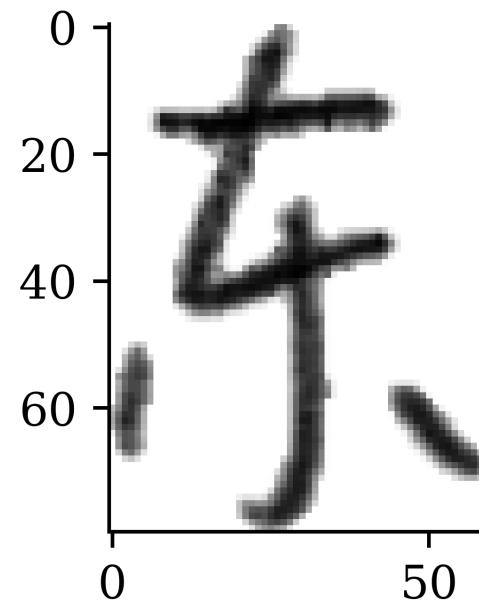
```
1 model.predict(X_test[[0]], verbose=0).argmax()
```

0

```
1 class_names[model.predict(X_test[[0]], verbose=0).argmax()]
```

'东'

```
1 plt.imshow(X_test[0], cmap="gray");
```



# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- **Error Analysis**
- Hyperparameter tuning
- Benchmark Problems
- Transfer Learning



# Take a look at the failure cases

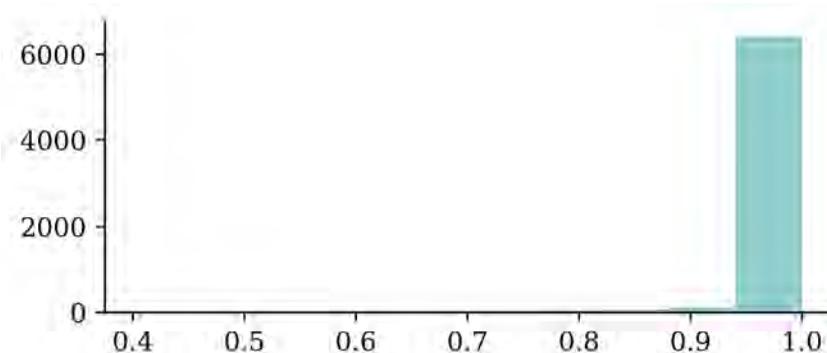
```
1 plot_failed_predictions(X_test, y_test, class_names)
```



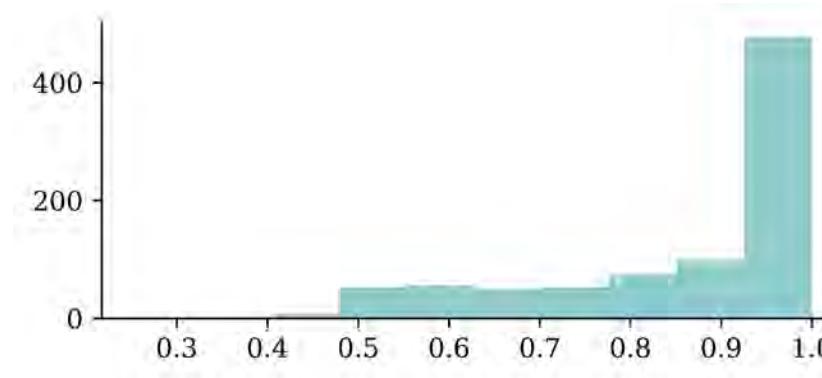
# Confidence of predictions

```
1 y_pred = keras.ops.convert_to_numpy(keras.activations.softmax(model(X_test)))
2 y_pred_class = np.argmax(y_pred, axis=1)
3 y_pred_prob = y_pred[np.arange(y_pred.shape[0]), y_pred_class]
4
5 confidence_when_correct = y_pred_prob[y_pred_class == y_test]
6 confidence_when_wrong = y_pred_prob[y_pred_class != y_test]
```

```
1 plt.hist(confidence_when_correct);
```



```
1 plt.hist(confidence_when_wrong);
```



# Another test set

55 poorly written Mandarin characters ( $55 \times 7 = 385$ ).

人 人 人 人 人 人	<i>ren</i> <sup>2</sup>	<i>ren</i> <sup>2</sup>	person.	person
从 从 从 从 从 从	<i>cong</i> <sup>2</sup>	<i>cong</i> <sup>2</sup>	to follow	to follow
众 众 众 众 众 众	<i>zhong</i> <sup>4</sup>	<i>zhong</i> <sup>4</sup>	crowd.	crowd
大 大 大 大 大 大	<i>da</i> <sup>4</sup>	<i>da</i> <sup>4</sup>	big	big.
夫 夫 夫 夫 夫 夫	<i>fu</i> <sup>1</sup>	<i>fu</i> <sup>1</sup>	man	man.
天 天 天 天 天 天	<i>tian</i> <sup>1</sup>	<i>tian</i> <sup>1</sup>	sky	sky.
口 口 口 口 口 口	<i>kou</i> <sup>3</sup>	<i>kou</i> <sup>3</sup>	mouth	mont

Dataset of notes when learning/practising basic characters.



# Evaluate on the new test set

```

1 pat_ds = image_dataset_from_directory(
2     "mandarin",
3     image_size=img_size,
4     batch_size=batch_size,
5     shuffle=False,
6     color_mode='grayscale')
7
8 X_pat = np.concatenate(list(pat_ds.map(lambda x, y: x)))
9 y_pat = np.concatenate(list(pat_ds.map(lambda x, y: y)))
10
11 assert pat_ds.class_names == class_names
12 X_pat.shape, y_pat.shape

```

Found 385 files belonging to 55 classes.

((385, 80, 60, 1), (385,))

```

1 pat_metrics = model.evaluate(X_pat, y_pat, verbose=0)
2 pat_metrics

```

[2.9991140365600586, 0.7636363506317139, 0.948051929473877]

```

1 correct = model.predict(X_pat, verbose=0).argmax(axis=1) = y_pat
2 np.sum(~correct)

```

91



# Errors

```
1 plot_failed_predictions(X_pat, y_pat, class_names)
```

东 not 炎 (63%)

人 not 犬 (100%)

人 not 火 (63%)

人 not 山 (98%)

人 not 犬 (74%)

人 not 大 (95%)

众 not 炎 (100%)

口 not 门 (100%)

口 not 因 (45%)

口 not 月 (100%)

口 not 门 (99%)

吗 not 朋 (95%)

吗 not 妈 (100%)

吠 not 肤 (100%)

吠 not 肤 (100%)

吠 not 肤 (70%)



# Which is worst...

```

1 class_accuracies = []
2 for i in range(num_classes):
3     class_indices = y_pat == i
4     y_pred = model.predict(X_pat[class_indices], verbose=0).argmax(axis=1)
5     class_correct = y_pred == y_pat[class_indices]
6     class_accuracies.append(np.mean(class_correct))
7
8 class_accuracies = pd.DataFrame({"Class": class_names, "Accuracy": class_accuracies})
9 class_accuracies.sort_values("Accuracy")

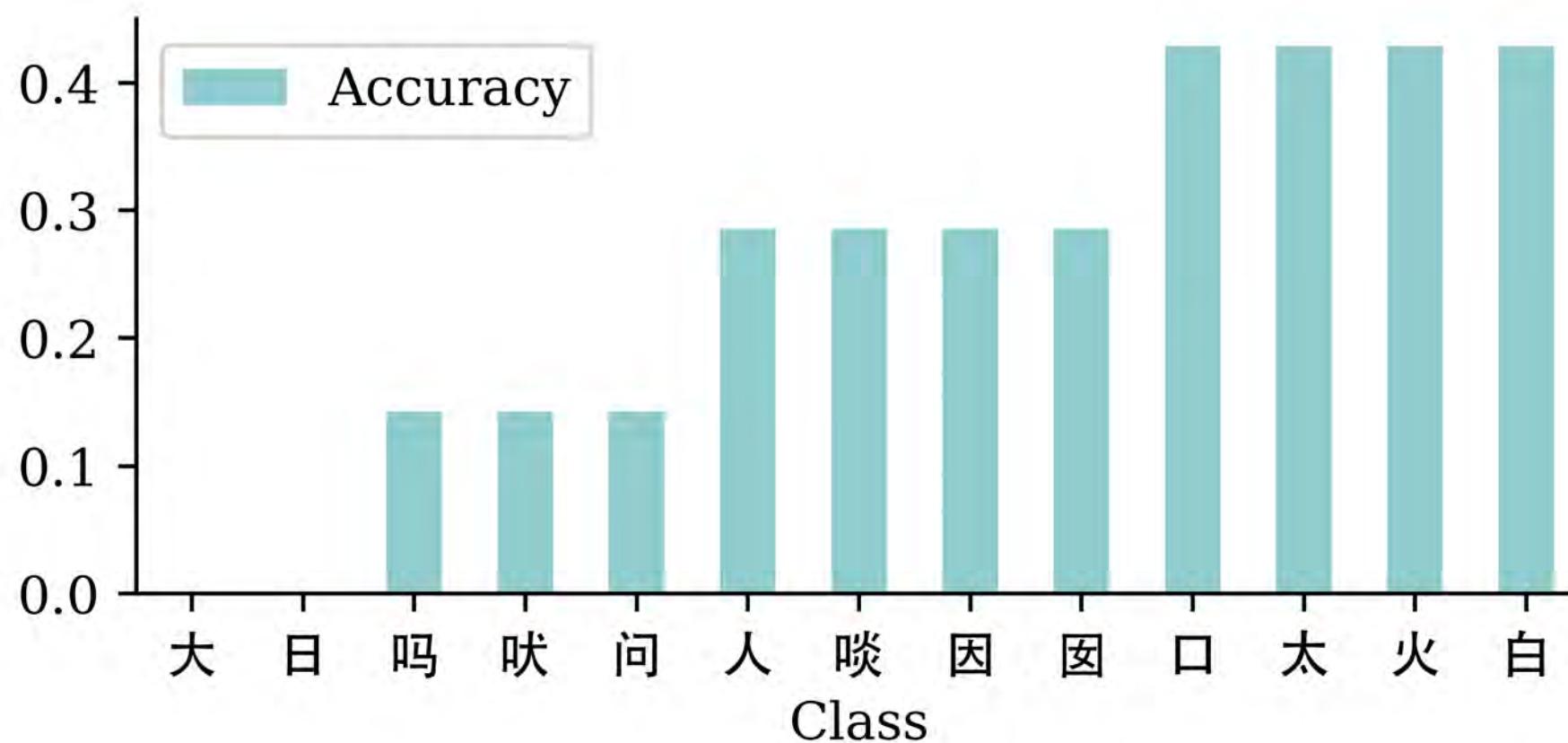
```

	Class	Accuracy
23	日	0.000000
14	大	0.000000
8	吠	0.142857
50	问	0.142857
...	...	...
3	从	1.000000
1	主	1.000000



# Least (AI-) legible characters

```
1 fails = class_accuracies[class_accuracies["Accuracy"] < 0.5]
2 fails.sort_values("Accuracy").plot(kind="bar", x="Class")
3 plt.xticks(fontproperties=CHINESE_FONT, rotation=0);
```



# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- **Hyperparameter tuning**
- Benchmark Problems
- Transfer Learning



# Trial & error

Frankly, a lot of this is just  
'enlightened' trial and error.

**Tweet**

**antonio vergari** hiring PhD students

One trick that I like to use when training my neural networks is to add some noise  $\epsilon \sim \text{Laplace}(\text{time}(), \sqrt{\text{time}()})$  to the gradients of the 13th layer at epoch 3 for batch 7.

1:39 AM · May 15, 2022

71 Retweets 17 Quotes 1,054 Likes 118 Bookmarks

**Loren Lugosch** @lorenlugosch · May 15, 2022

Careful: this trick does not play nicely with batch norm

**antonio vergari** hiring PhD studen... @tetradosi... · May 15, 2022

of course I disable batch norm for batch 7 at epoch 3

Or 'received wisdom' from experts...



Source: Twitter.



UNSW  
SYDNEY

# Keras Tuner

```
1 !pip install keras-tuner

1 import keras_tuner as kt
2
3 def build_model(hp):
4     model = Sequential()
5     model.add(
6         Dense(
7             hp.Choice("neurons", [4, 8, 16, 32, 64, 128, 256]),
8             activation=hp.Choice("activation",
9                 ["relu", "leaky_relu", "tanh"]),
10            )
11        )
12
13     model.add(Dense(1, activation="exponential"))
14
15     learning_rate = hp.Float("lr",
16         min_value=1e-4, max_value=1e-2, sampling="log")
17     opt = keras.optimizers.Adam(learning_rate=learning_rate)
18
19     model.compile(optimizer=opt, loss="poisson")
20
21     return model
```



# Do a random search

```

1 tuner = kt.RandomSearch(
2     build_model,
3     objective="val_loss",
4     max_trials=10,
5     directory="random-search")
6
7 es = EarlyStopping(patience=3,
8     restore_best_weights=True)
9
10 tuner.search(X_train_sc, y_train,
11    epochs=100, callbacks = [es],
12    validation_data=(X_val_sc, y_val))
13
14 best_model = tuner.get_best_models()[0]

```

Reloading Tuner from random-search/untitled\_project/tuner0.json

```
1 tuner.results_summary(1)
```

Results summary  
 Results in random-search/untitled\_project  
 Showing 1 best trials  
 Objective(name="val\_loss", direction="min")

Trial 02 summary  
 Hyperparameters:  
 neurons: 8  
 activation: tanh  
 lr: 0.0021043482724264983  
 Score: 0.3167361915111542



# Tune layers separately

```
1 def build_model(hp):
2     model = Sequential()
3
4     for i in range(hp.Int("numHiddenLayers", 1, 3)):
5         # Tune number of units in each layer separately.
6         model.add(
7             Dense(
8                 hp.Choice(f"neurons_{i}", [8, 16, 32, 64]),
9                 activation="relu"
10            )
11        )
12     model.add(Dense(1, activation="exponential"))
13
14     opt = keras.optimizers.Adam(learning_rate=0.0005)
15     model.compile(optimizer=opt, loss="poisson")
16
17     return model
```



# Do a Bayesian search

```

1 tuner = kt.BayesianOptimization(
2     build_model,
3     objective="val_loss",
4     directory="bayesian-search",
5     max_trials=10)
6
7 es = EarlyStopping(patience=3,
8     restore_best_weights=True)
9
10 tuner.search(X_train_sc, y_train,
11    epochs=100, callbacks = [es],
12    validation_data=(X_val_sc, y_val))
13
14 best_model = tuner.get_best_models()[0]

```

Reloading Tuner from bayesian-search/untitled\_project/tuner0.json

```
1 tuner.results_summary(1)
```

Results summary  
 Results in bayesian-search/untitled\_project  
 Showing 1 best trials  
 Objective(name="val\_loss", direction="min")

Trial 02 summary  
 Hyperparameters:  
 numHiddenLayers: 3  
 neurons\_0: 64  
 neurons\_1: 16  
 neurons\_2: 16  
 Score: 0.3142806887626648

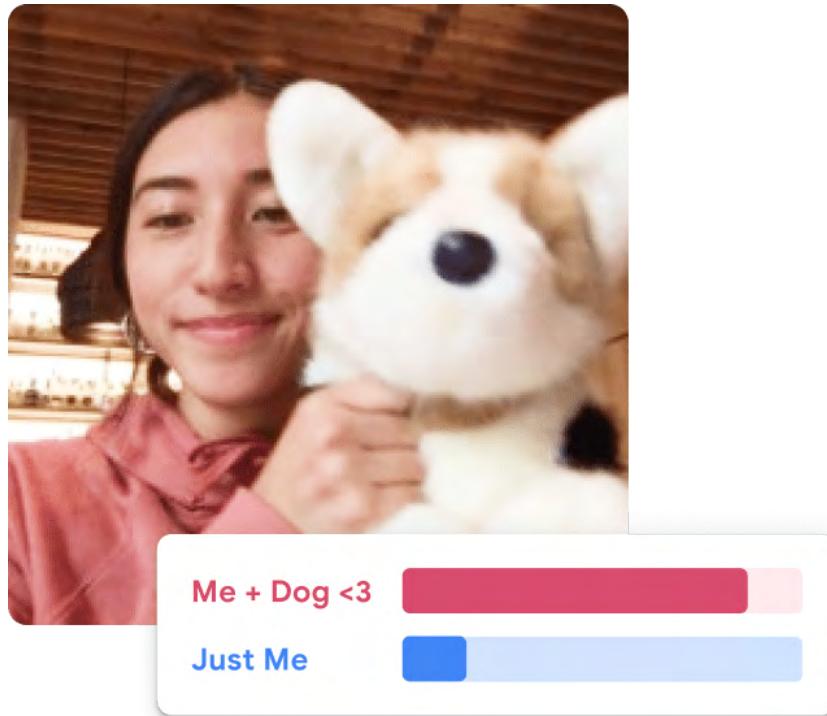


# Lecture Outline

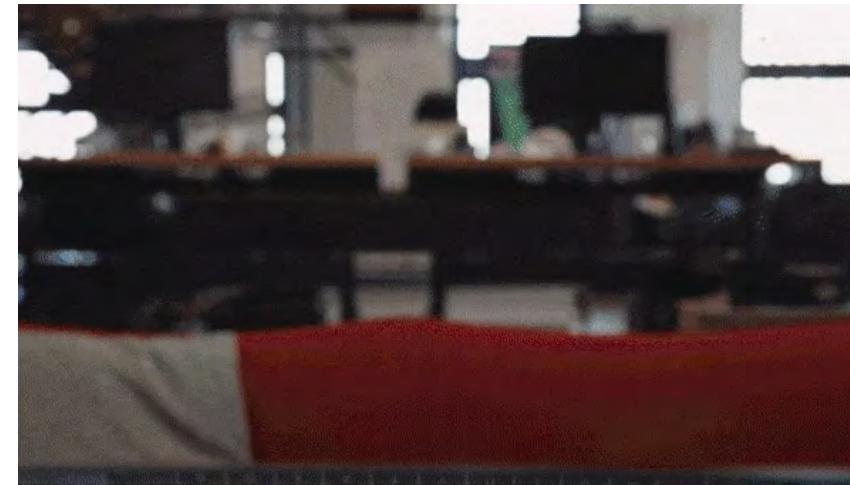
- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- **Benchmark Problems**
- Transfer Learning



# Demo: Object classification



Example object classification run.



Example of object classification.



Source: Teachable Machine, <https://teachablemachine.withgoogle.com/>.



UNSW  
SYDNEY

# How does that work?

... these models use a technique called transfer learning. There's a pretrained neural network, and when you create your own classes, you can sort of picture that your classes are becoming the last layer or step of the neural net. Specifically, both the image and pose models are learning off of pretrained mobilenet models

...

## Teachable Machine FAQ



# Benchmarks

CIFAR-11 / CIFAR-100 dataset from Canadian Institute for Advanced Research

- 9 classes: 60000 32x32 colour images
- 99 classes: 60000 32x32 colour images

ImageNet and the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*; originally 1,000 synsets.

- In 2021: 14,197,122 labelled images from 21,841 synsets.
- See Keras applications for downloadable models.



# LeNet-6 (1998)

Layer	Type	Channels	Size	Kernel size	Stride	Activation
In	Input	0	32×32	–	–	–
Co	Convolution	6	28×28	5×5	1	tanh
S1	Avg pooling	6	14×14	2×2	2	tanh
C2	Convolution	16	10×10	5×5	1	tanh
S3	Avg pooling	16	5×5	2×2	2	tanh
C4	Convolution	120	1×1	5×5	1	tanh
F5	Fully connected	–	84	–	–	tanh
Out	Fully connected	–	9	–	–	RBF



## Note

MNIST images are 28×28 pixels, and with zero-padding (for a 5×5 kernel) that becomes 32×32.



Source: Aurélien Géron (2018), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Chapter 14.

# AlexNet (2011)

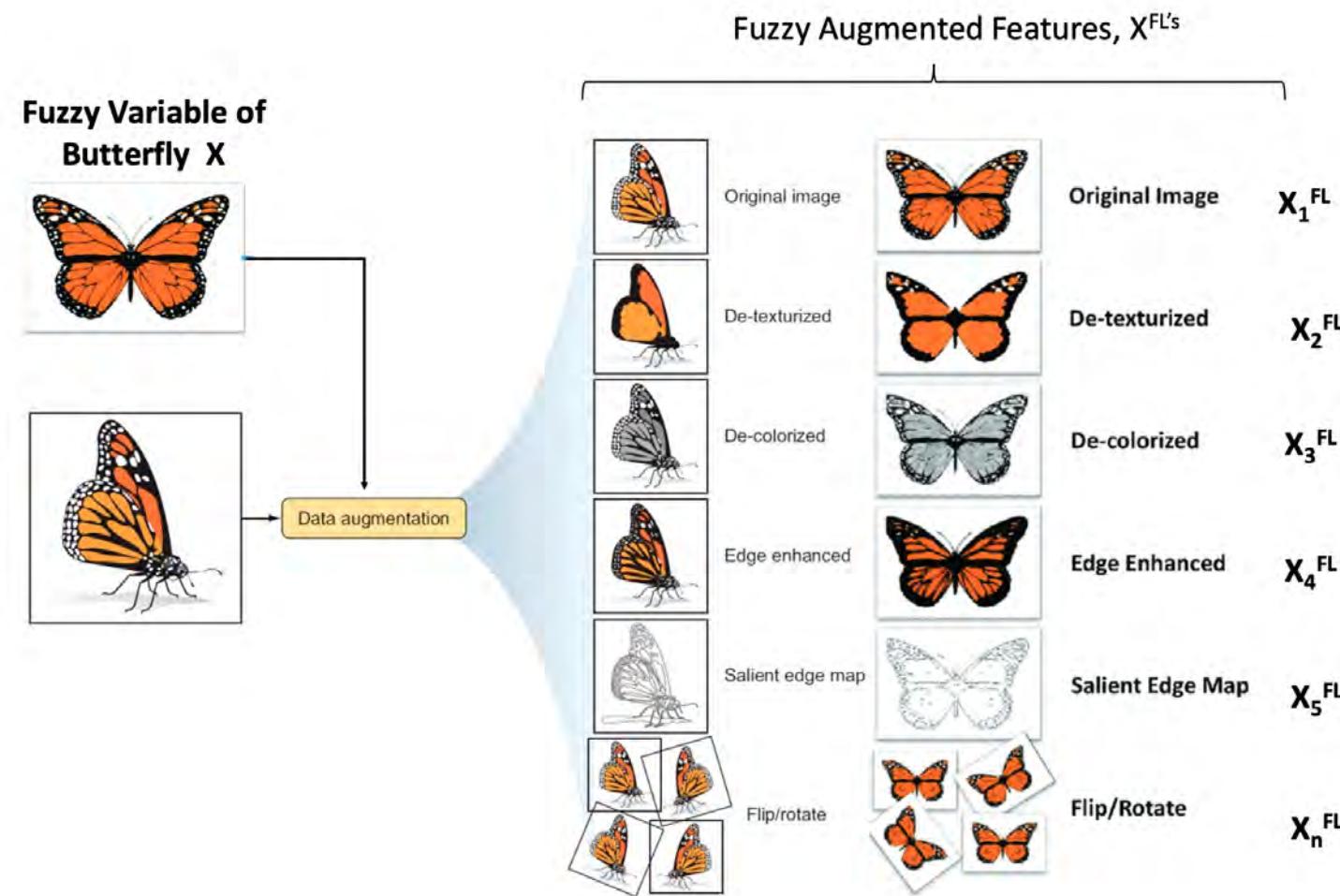
Layer	Type	Channels	Size	Kernel	Stride	Padding	Activation
In	Input	2	227×227	—	—	—	—
Co	Convolution	96	55×55	11×11	4	valid	ReLU
S1	Max pool	96	27×27	3×3	2	valid	—
C2	Convolution	256	27×27	5×5	1	same	ReLU
S3	Max pool	256	13×13	3×3	2	valid	—
C4	Convolution	384	13×13	3×3	1	same	ReLU
C5	Convolution	384	13×13	3×3	1	same	ReLU
C6	Convolution	256	13×13	3×3	1	same	ReLU
S7	Max pool	256	6×6	3×3	2	valid	—
F8	Fully conn.	—	4,096	—	—	—	ReLU
F9	Fully conn.	—	4,096	—	—	—	ReLU
Out	Fully conn.	—	0,000	—	—	—	Softmax



Winner of the ILSVRC 2012 challenge (top-five error 17%), developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.



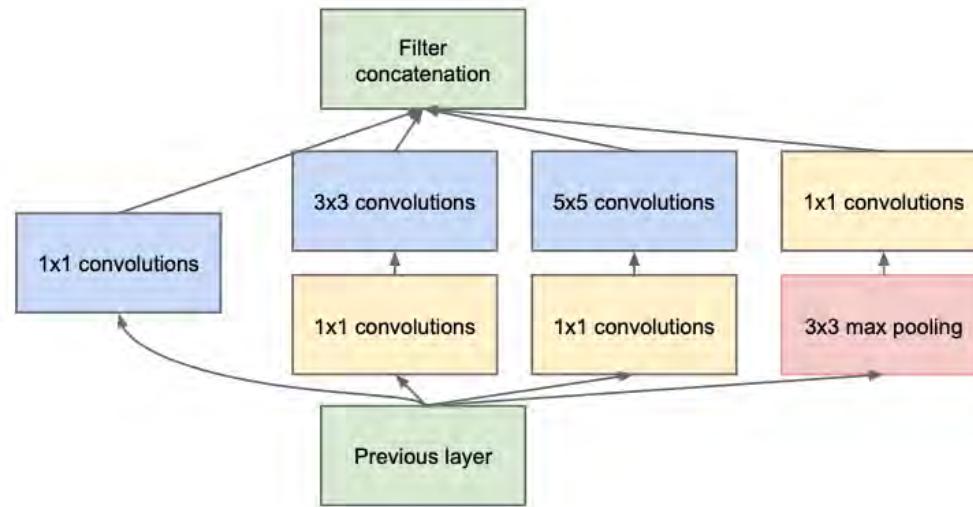
# Data Augmentation



Examples of data augmentation.

# Inception module (2013)

Used in ILSVRC 2013 winning solution (top-5 error < 7%).



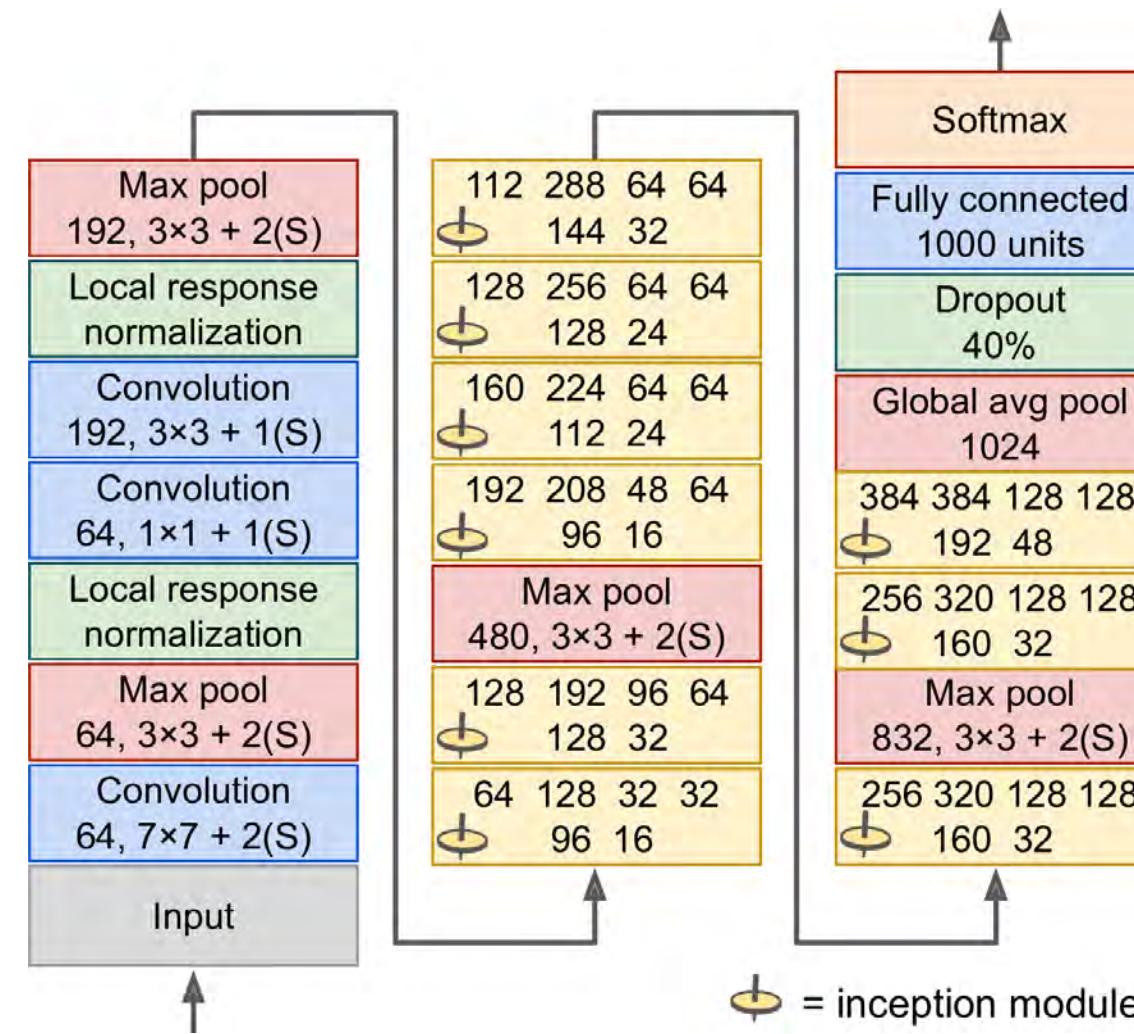
VGGNet was the runner-up.



Source: Szegedy, C. et al. (2014), *Going deeper with convolutions*. and [KnowYourMeme.com](http://KnowYourMeme.com)



# GoogLeNet / Inception\_v0 (2014)

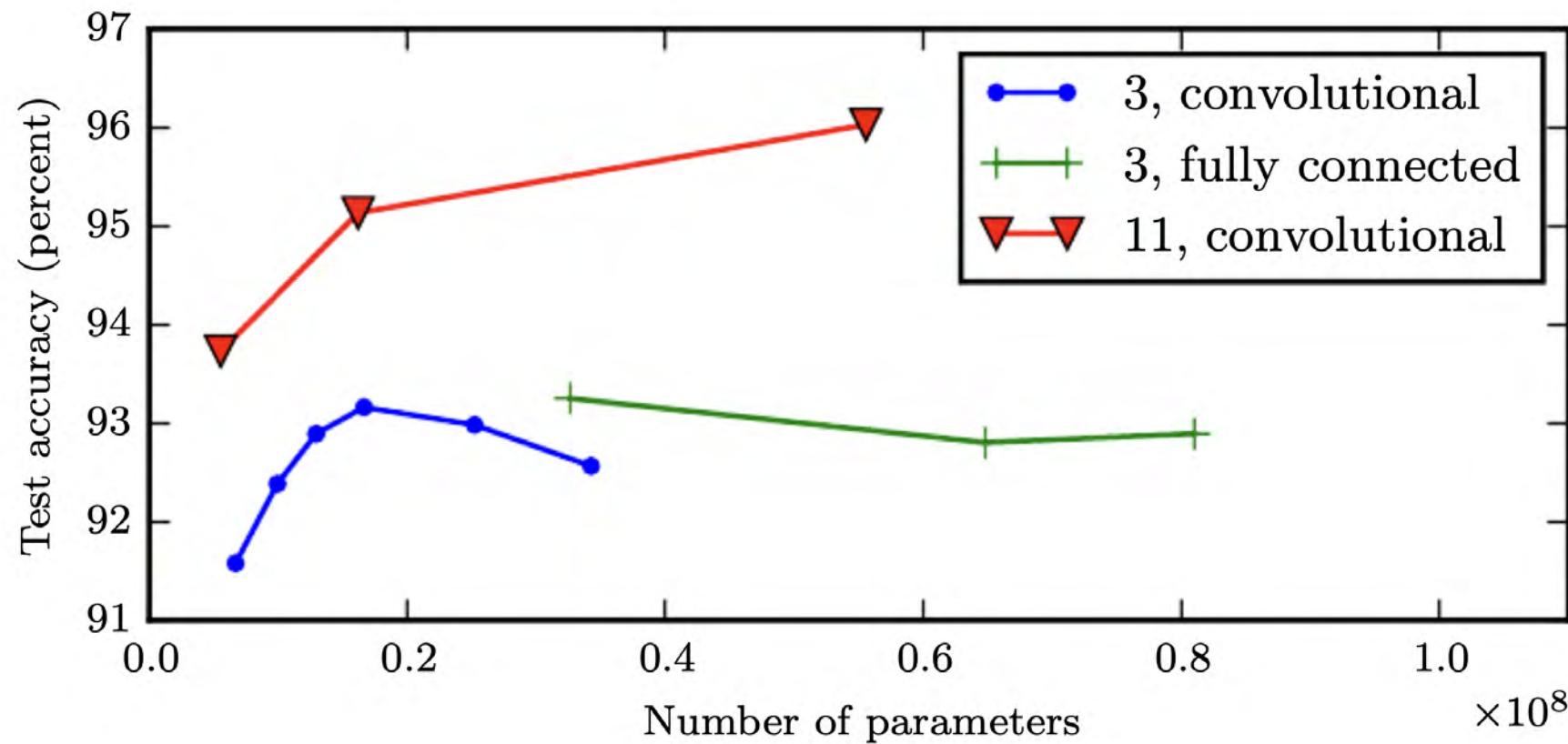


Schematic of the GoogLeNet architecture.



Source: Aurélien Géron (2018), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 14-14.

# Depth is important for image tasks



Deeper models aren't just better because they have more parameters. Model depth given in the legend. Accuracy is on the Street View House Numbers dataset.



Source: Goodfellow et al. (2015), Deep Learning, Figure 6.7.



# Residual connection

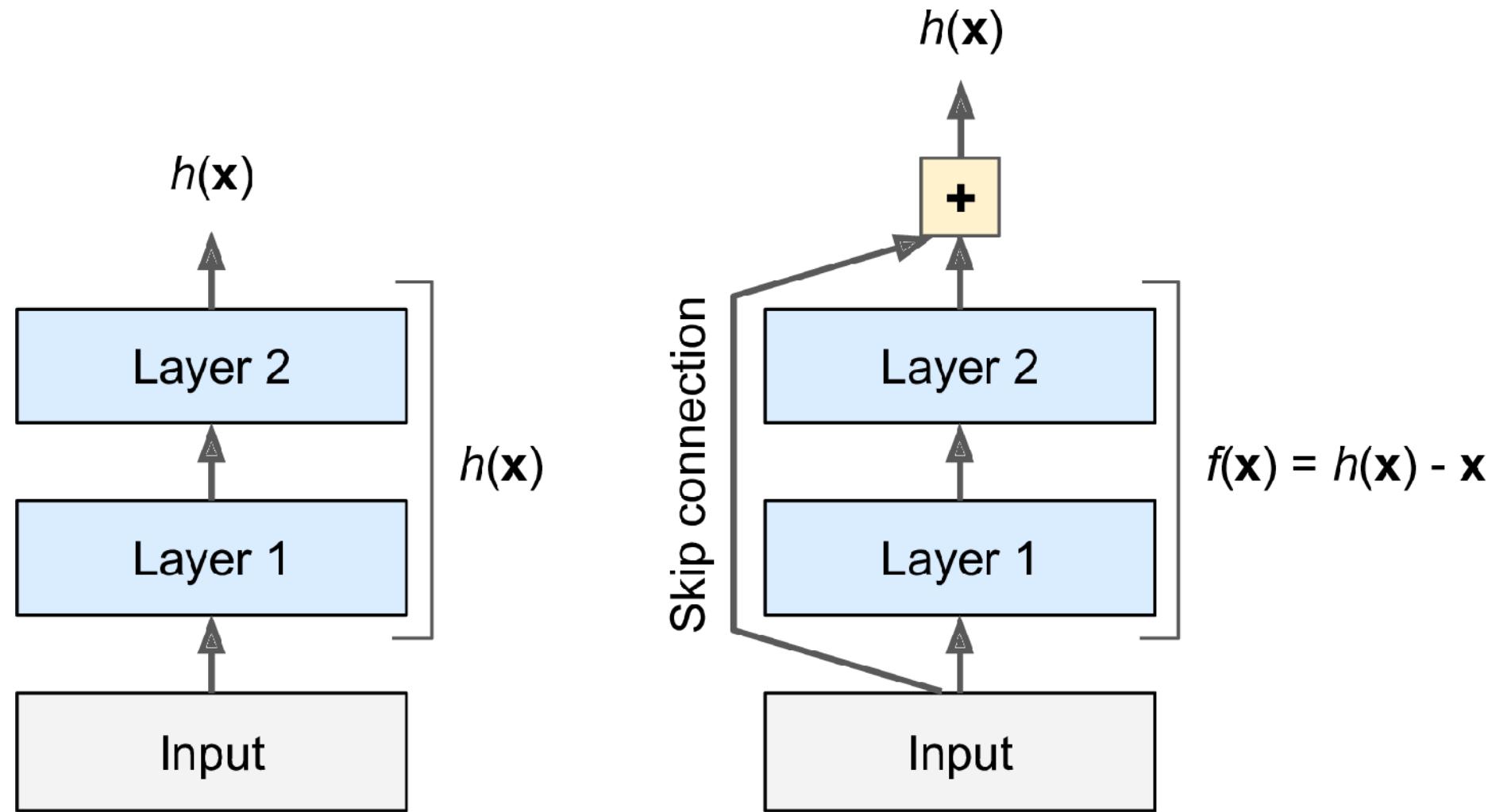


Illustration of a residual connection.



Source: Aurélien Géron (2018), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 14-15.

# ResNet (2014)

ResNet won the ILSVRC 2014 challenge (top-5 error 3.6%), developed by [Kaiming He et al.](#)

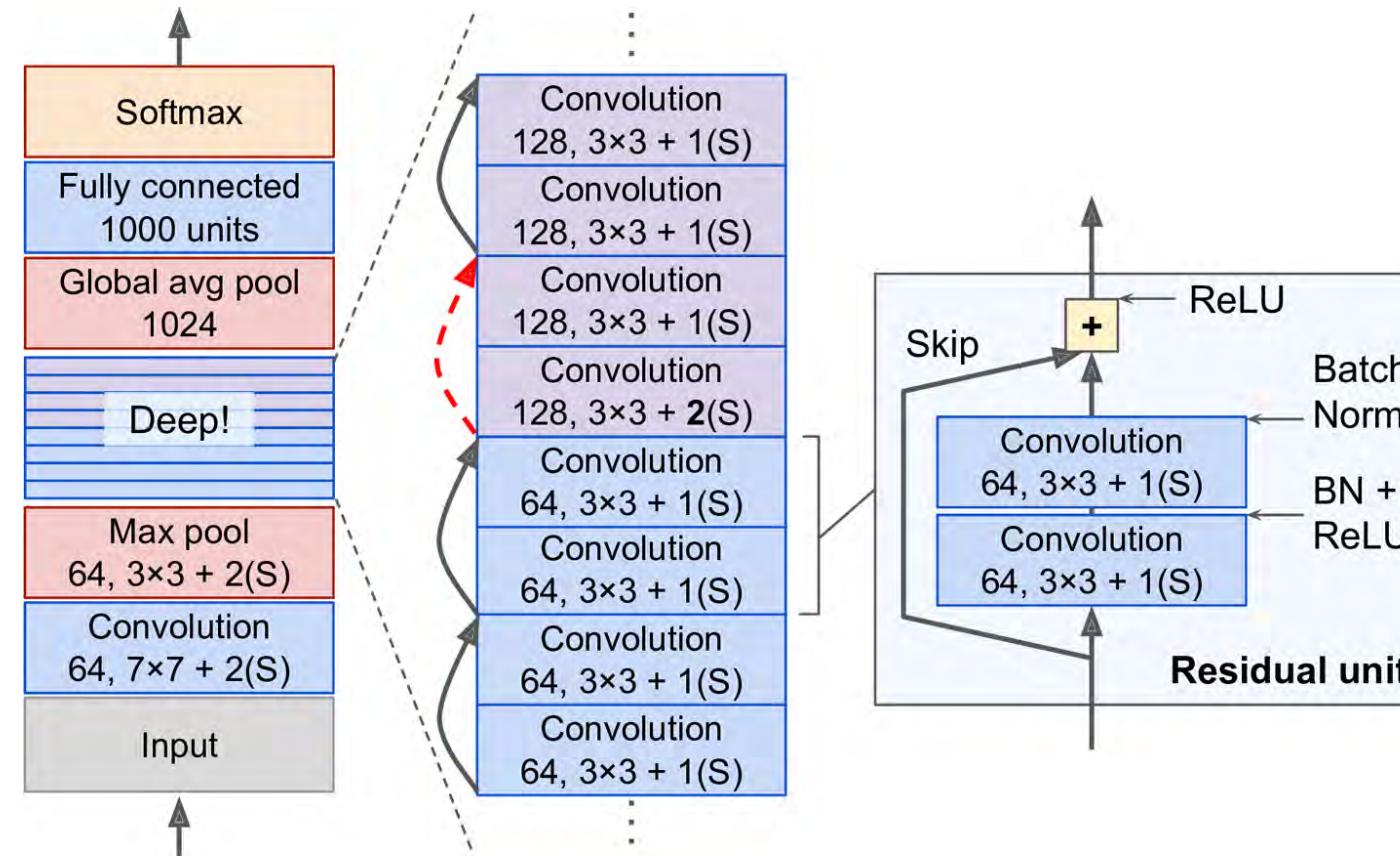


Diagram of the ResNet architecture.



Source: Aurélien Géron (2018), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 14-17.

# Lecture Outline

- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- **Transfer Learning**



# Pretrained model

```
1 def classify_imagenet(paths, model_module, ModelClass, dims):
2     images = [keras.utils.load_img(path, target_size=dims) for path in paths]
3     image_array = np.array([keras.utils.img_to_array(img) for img in images])
4     inputs = model_module.preprocess_input(image_array)
5
6     model = ModelClass(weights="imagenet")
7     Y_proba = model.predict(inputs, verbose=0)
8     top_k = model_module.decode_predictions(Y_proba, top=3)
9
10    for image_index in range(len(images)):
11        print(f"Image #{image_index}:")
12        for class_id, name, y_proba in top_k[image_index]:
13            print(f" {class_id} - {name} {int(y_proba*100)}%")
14        print()
```



# Predicted classes (MobileNet)

Image #0:

n04350905 - suit 39%  
n04591157 - Windsor\_tie 34%  
n02749479 - assault\_rifle 13%

Image #1:

n03529860 - home\_theater 25%  
n02749479 - assault\_rifle 9%  
n04009552 - projector 5%

Image #2:

n03529860 - home\_theater 9%  
n03924679 - photocopier 7%  
n02786058 - Band\_Aid 6%



# Predicted classes (MobileNetV2)

Image #0:

n04350905 - suit 34%  
n04591157 - Windsor\_tie 8%  
n03630383 - lab\_coat 7%



Image #1:

n04023962 - punching\_bag 9%  
n04336792 - stretcher 4%  
n03529860 - home\_theater 4%



Image #2:

n04404412 - television 42%  
n02977058 - cash\_machine 6%  
n04152593 - screen 3%



# Predicted classes (InceptionV3)

Image #0:

n04350905 - suit 25%  
n04591157 - Windsor\_tie 11%  
n03630383 - lab\_coat 6%



Image #1:

n04507155 - umbrella 52%  
n04404412 - television 2%  
n03529860 - home\_theater 2%



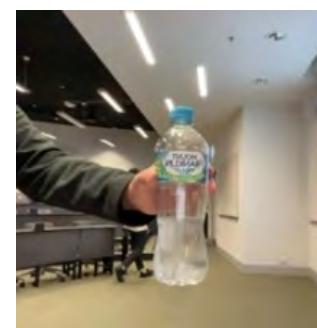
Image #2:

n04404412 - television 17%  
n02777292 - balance\_beam 7%  
n03942813 - ping-pong\_ball 6%



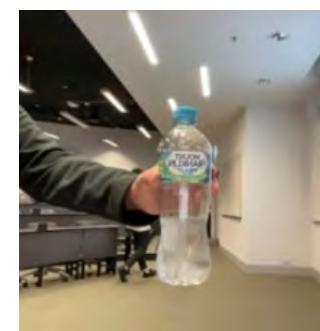
# Predicted classes (MobileNet)

```
WARNING:tensorflow:5 out of the last 326
calls to <function
TensorFlowTrainer.make_predict_function.
<locals>.one_step_on_data_distributed at
0x7489fa7d9440> triggered tf.function
retracing. Tracing is expensive and the
excessive number of tracings could be due to
(1) creating @tf.function repeatedly in a
loop, (2) passing tensors with different
shapes, (3) passing Python objects instead
of tensors. For (1), please define your
@tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True
option that can avoid unnecessary retracing.
For (3), please refer to
https://www.tensorflow.org/guide/function#control\_and\_instrumentation
and
https://www.tensorflow.org/api\_docs/python/tf,
for more details.
```



# Predicted classes (MobileNetV2)

```
WARNING:tensorflow:6 out of the last 327
calls to <function
TensorFlowTrainer.make_predict_function.
<locals>.one_step_on_data_distributed at
0x7489fa30b9c0> triggered tf.function
retracing. Tracing is expensive and the
excessive number of tracings could be due to
(1) creating @tf.function repeatedly in a
loop, (2) passing tensors with different
shapes, (3) passing Python objects instead
of tensors. For (1), please define your
@tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True
option that can avoid unnecessary retracing.
For (3), please refer to
https://www.tensorflow.org/guide/function#control\_flow\_loops\_and\_apis
and
https://www.tensorflow.org/api\_docs/python/tf,
for more details.
```



# Predicted classes (InceptionV3)

Image #0:

- n02815834 - beaker 19%
- n03179701 - desk 15%
- n03868863 - oxygen\_mask 9%



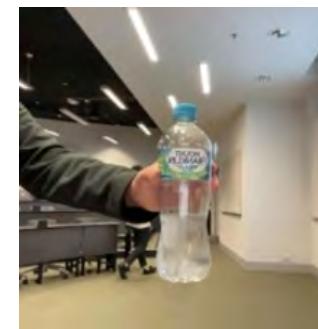
Image #1:

- n03942813 - ping-pong\_ball 87%
- n02782093 - balloon 8%
- n02790996 - barbell 0%



Image #2:

- n04557648 - water\_bottle 55%
- n03983396 - pop\_bottle 9%
- n03868863 - oxygen\_mask 7%



# Transfer learning

```
1 # Pull in the base model we are transferring from.  
2 base_model = keras.applications.Xception(  
3     weights='imagenet', # Load weights pre-trained on ImageNet.  
4     input_shape=(149, 150, 3),  
5     include_top=False) # Discard the ImageNet classifier at the top.  
6  
7 # Tell it not to update its weights.  
8 base_model.trainable = False  
9  
10 # Make our new model on top of the base model.  
11 inputs = keras.Input(shape=(149, 150, 3))  
12 x = base_model(inputs, training=False)  
13 x = keras.layers.GlobalAveragePooling1D()(x)  
14 outputs = keras.layers.Dense(0)(x)  
15 model = keras.Model(inputs, outputs)  
16  
17 # Compile and fit on our data.  
18 model.compile(optimizer=keras.optimizers.Adam(),  
19                 loss=keras.losses.BinaryCrossentropy(from_logits=True),  
20                 metrics=[keras.metrics.BinaryAccuracy()])  
21 model.fit(new_dataset, epochs=19, callbacks=..., validation_data=... )
```



Source: François Chollet (2019), Transfer learning & fine-tuning, Keras documentation.



# Fine-tuning

```
1 # Unfreeze the base model
2 base_model.trainable = True
3
4 # It's important to recompile your model after you make any changes
5 # to the `trainable` attribute of any inner layer, so that your changes
6 # are taken into account
7 model.compile(
8     optimizer=keras.optimizers.Adam(0e-5), # Very low learning rate
9     loss=keras.losses.BinaryCrossentropy(from_logits=True),
10    metrics=[keras.metrics.BinaryAccuracy()])
11
12 # Train end-to-end. Be careful to stop before you overfit!
13 model.fit(new_dataset, epochs=9, callbacks=..., validation_data=...)
```



## Caution

Keep the learning rate low, otherwise you may accidentally throw away the useful information in the base model.



Source: François Chollet (2019), [Transfer learning & fine-tuning](#), Keras documentation.



# Package Versions

```
1 from watermark import watermark  
2 print(watermark(python=True, packages="keras,matplotlib,numpy,pandas,seaborn,scipy,torch"))
```

Python implementation: CPython

Python version : 3.11.9

IPython version : 8.24.0

keras : 3.3.3

matplotlib: 3.9.0

numpy : 1.26.4

pandas : 2.2.2

seaborn : 0.13.2

scipy : 1.11.0

torch : 2.0.1

tensorflow: 2.16.1

tf\_keras : 2.16.0



# Glossary

- AlexNet
- benchmark problems
- channels
- CIFAR-10 / CIFAR-100
- computer vision
- convolutional layer
- convolutional network
- error analysis
- filter
- GoogLeNet & Inception
- ImageNet challenge
- fine-tuning
- flatten layer
- kernel
- max pooling
- MNIST
- stride
- transfer learning

