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

UNSW SYDNEY

Shapes of data

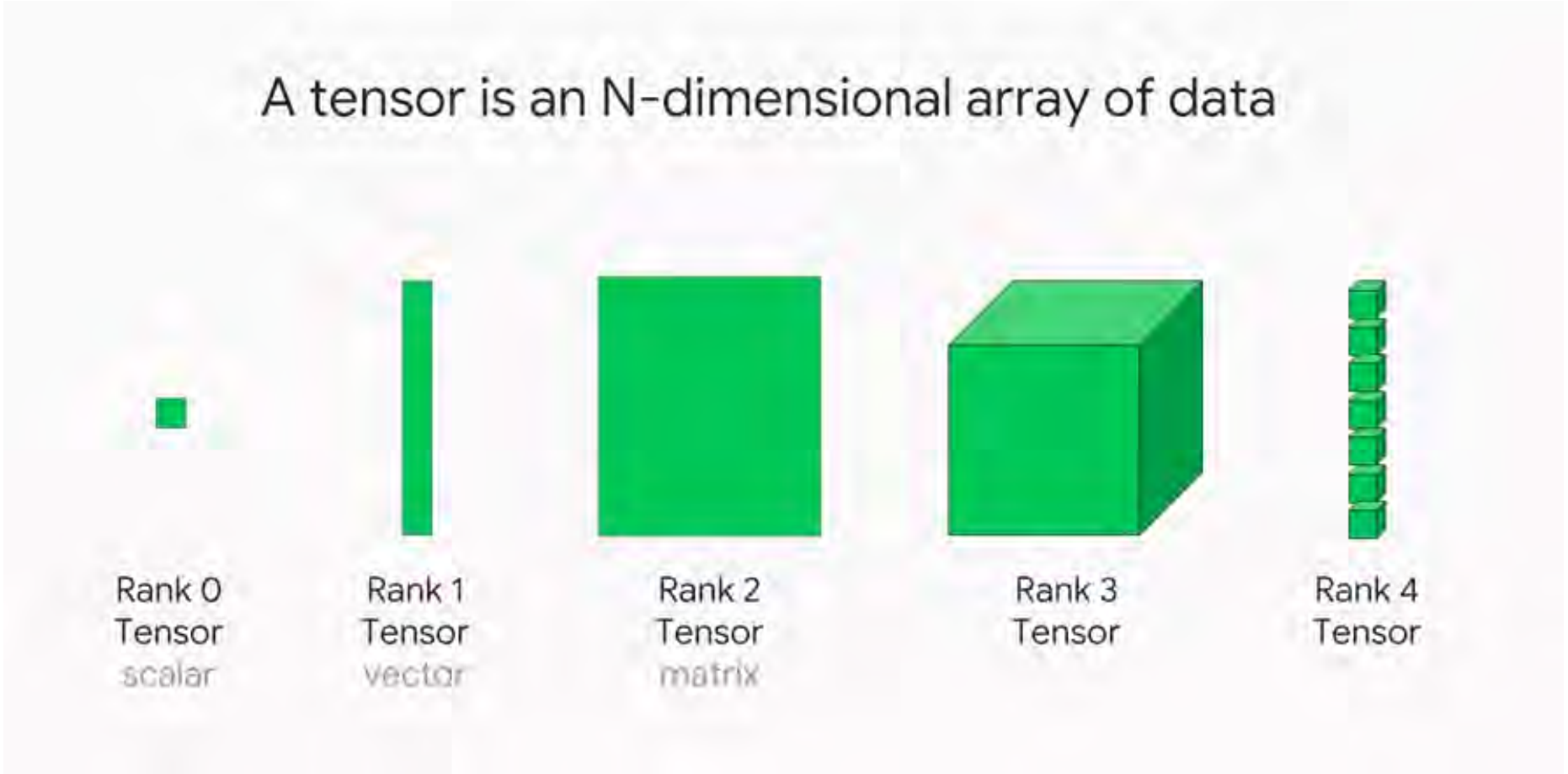
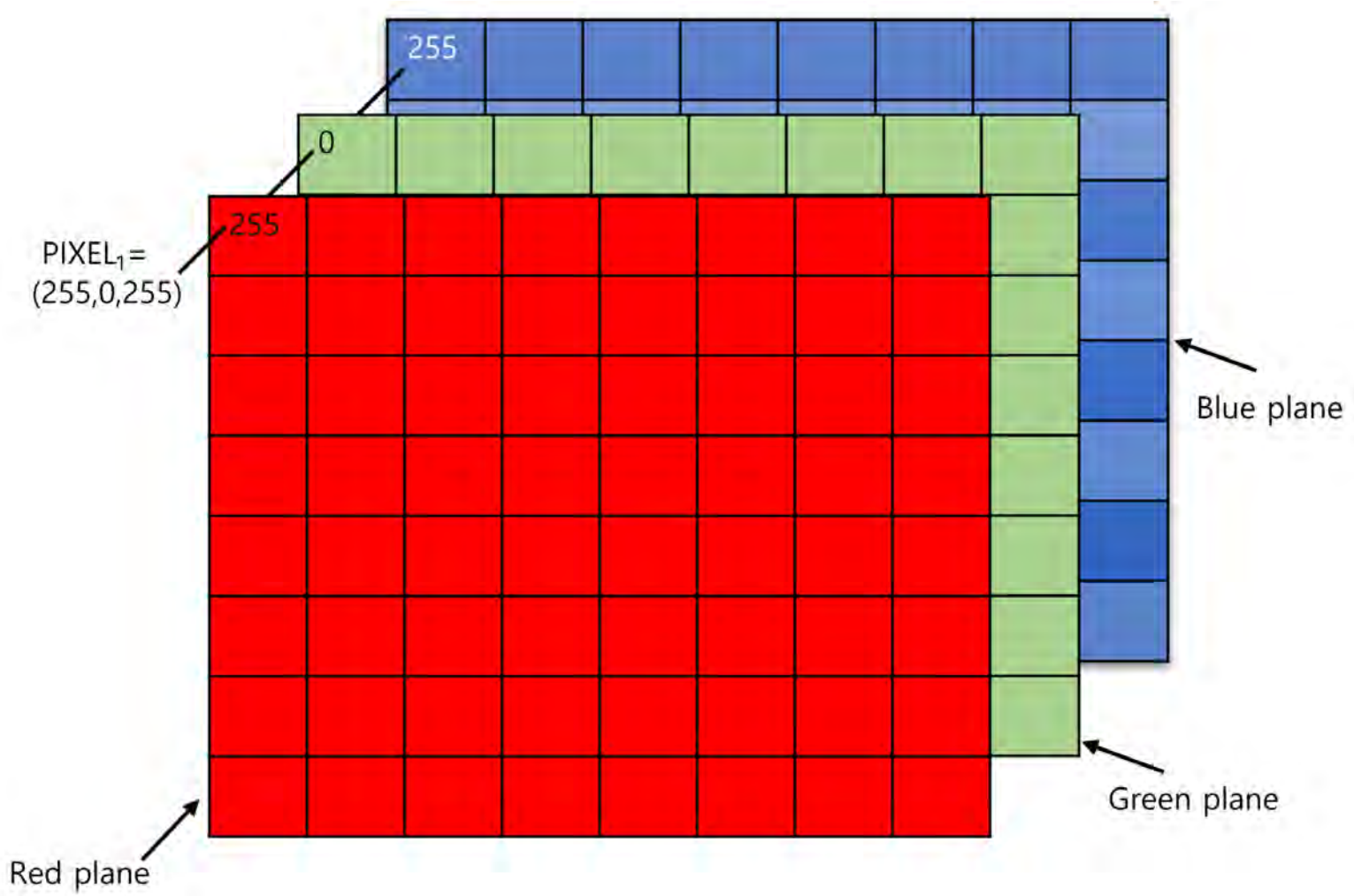


Illustration of tensors of different rank.

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],
        [  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],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   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],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0],
```

```
1 img3
```

```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   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],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0],
        [255, 255,   0],
```

```
1 img4
```

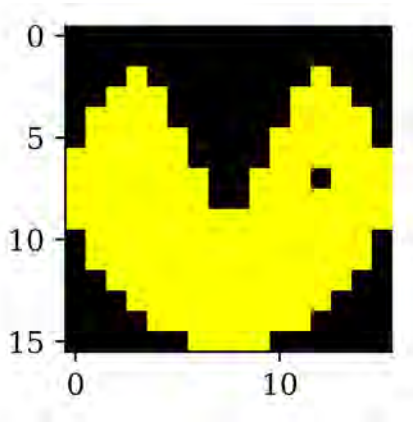
```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
        [255, 163, 177],
```

UNSW SYDNEY

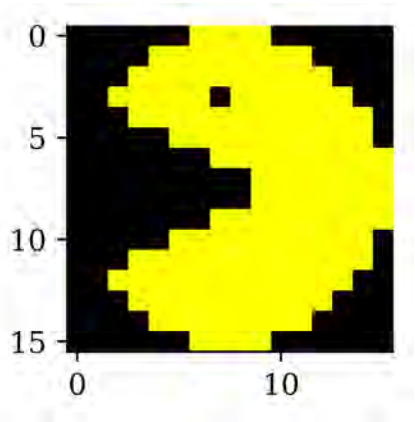
How we see them

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

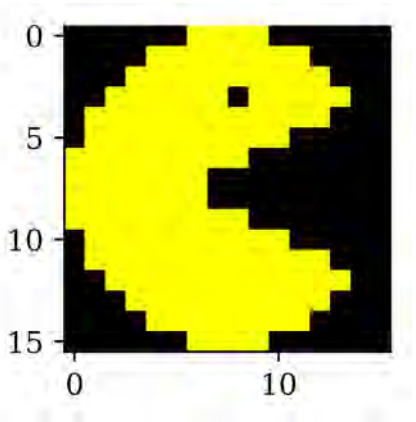
```
1 imshow(img1);
```



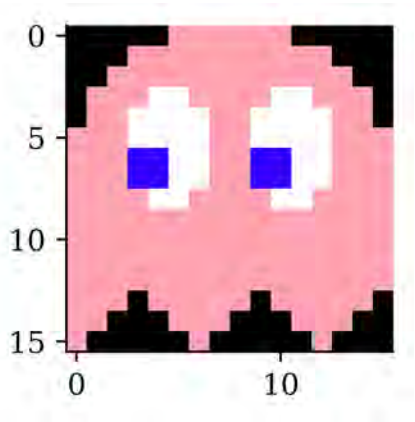

```
1 imshow(img2);
```




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




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



UNSW SYDNEY

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)
```

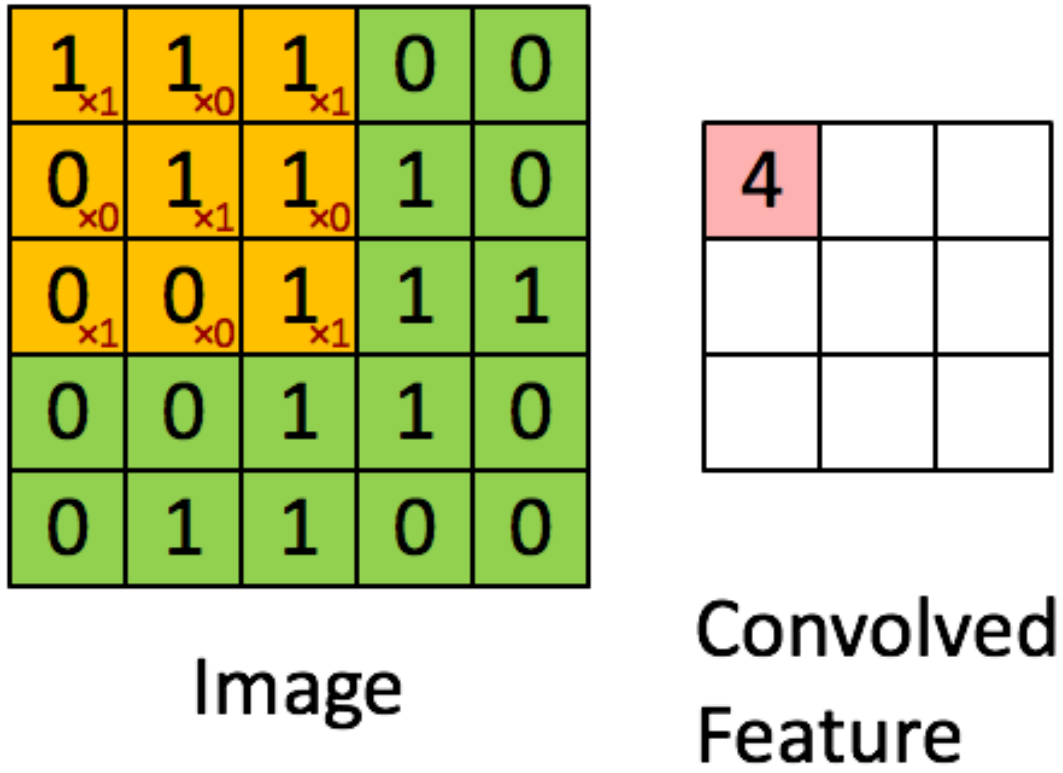
```
array([-128,    1,  127, -128], dtype=int8)
```

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

UNSW SYDNEY

Image editing with kernels

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



An example of an image kernel in action.

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

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

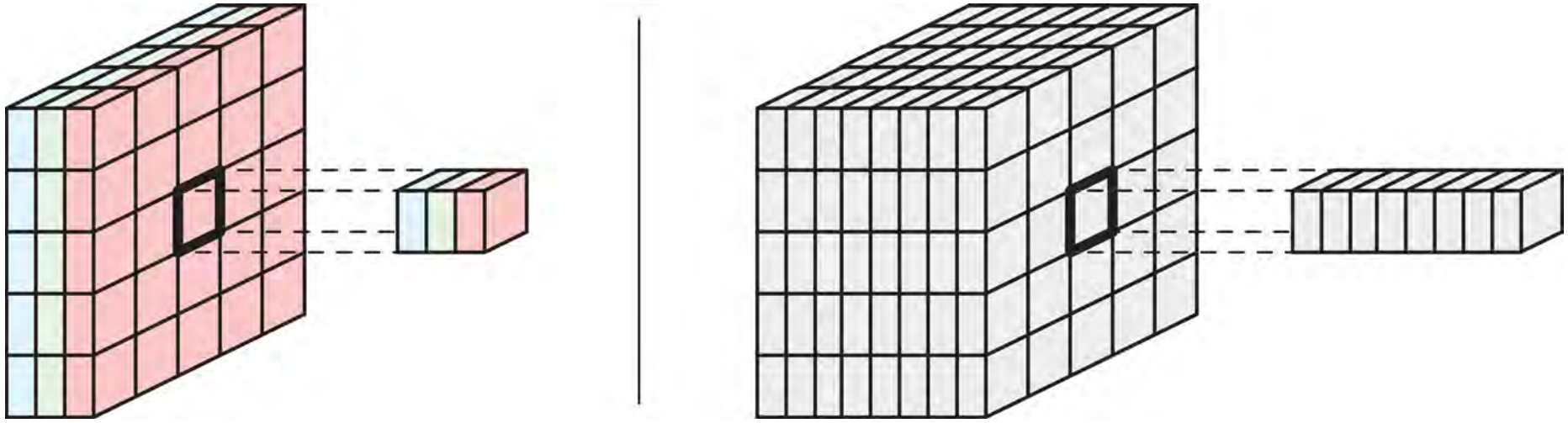
$$f_S(s) = \int_{x_1=-\infty}^{\infty} f_X(x_1)\, f_X(s - x_1)\, \mathrm{d}s.$$

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

UNSW SYDNEY

Images are rank 3 tensors

Height, width, and number of channels.



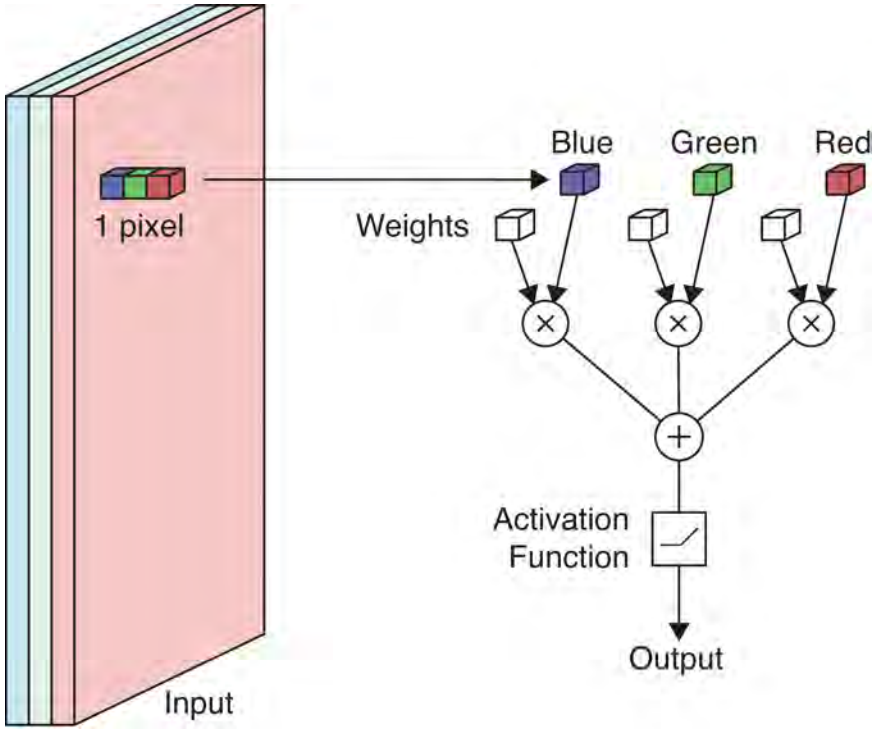
Examples of rank 3 tensors.

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

Example: Yellow = Red + Green.

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

Example: Detecting yellow



Applying a neuron to an image pixel.

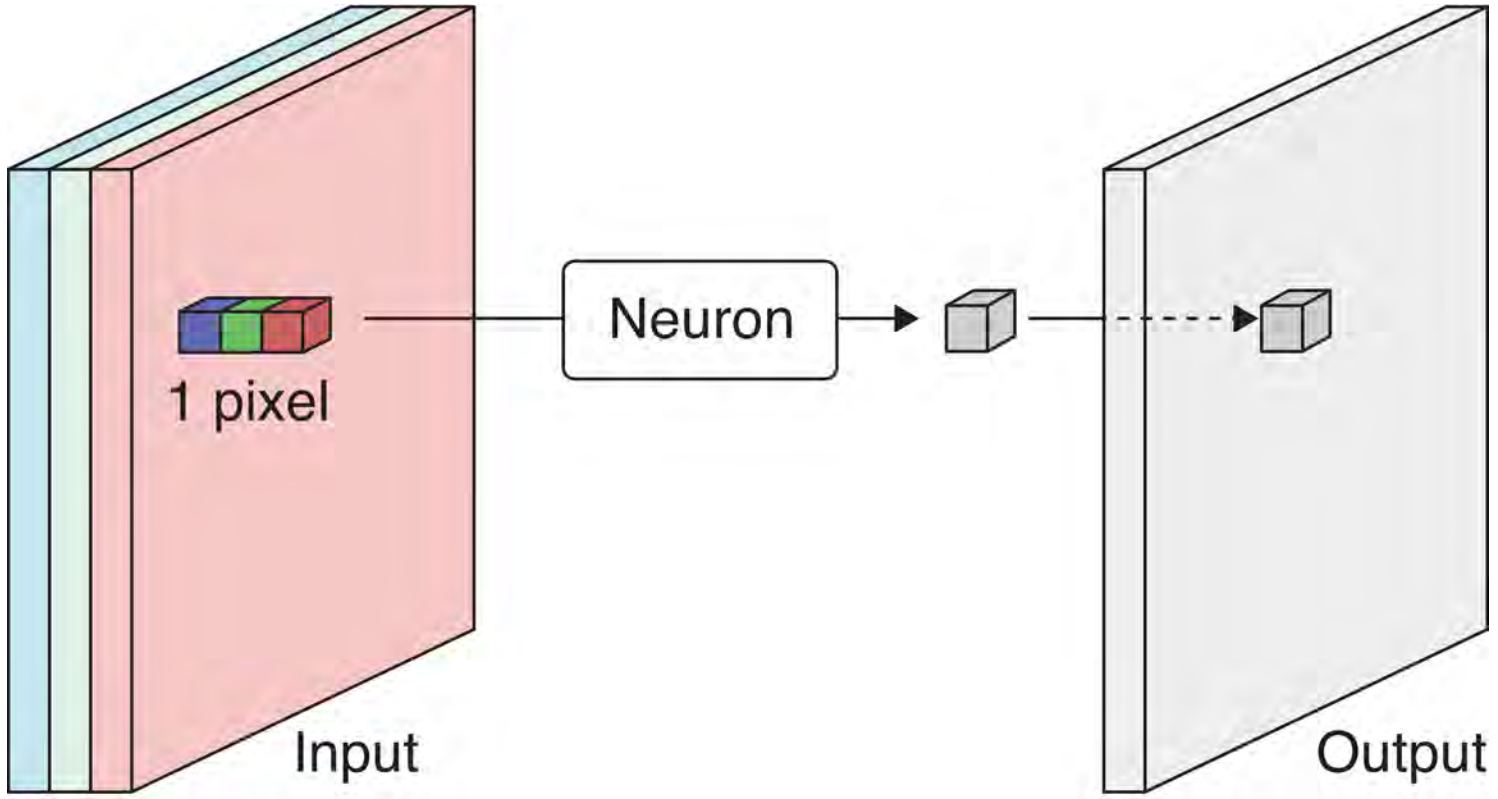
Apply a neuron to each pixel in the image.

If red/green ↗ or blue ↘ then yellowness ↗.

Set RGB weights to 1, 1, −1.

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

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**.

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

Example: Detecting yellow III

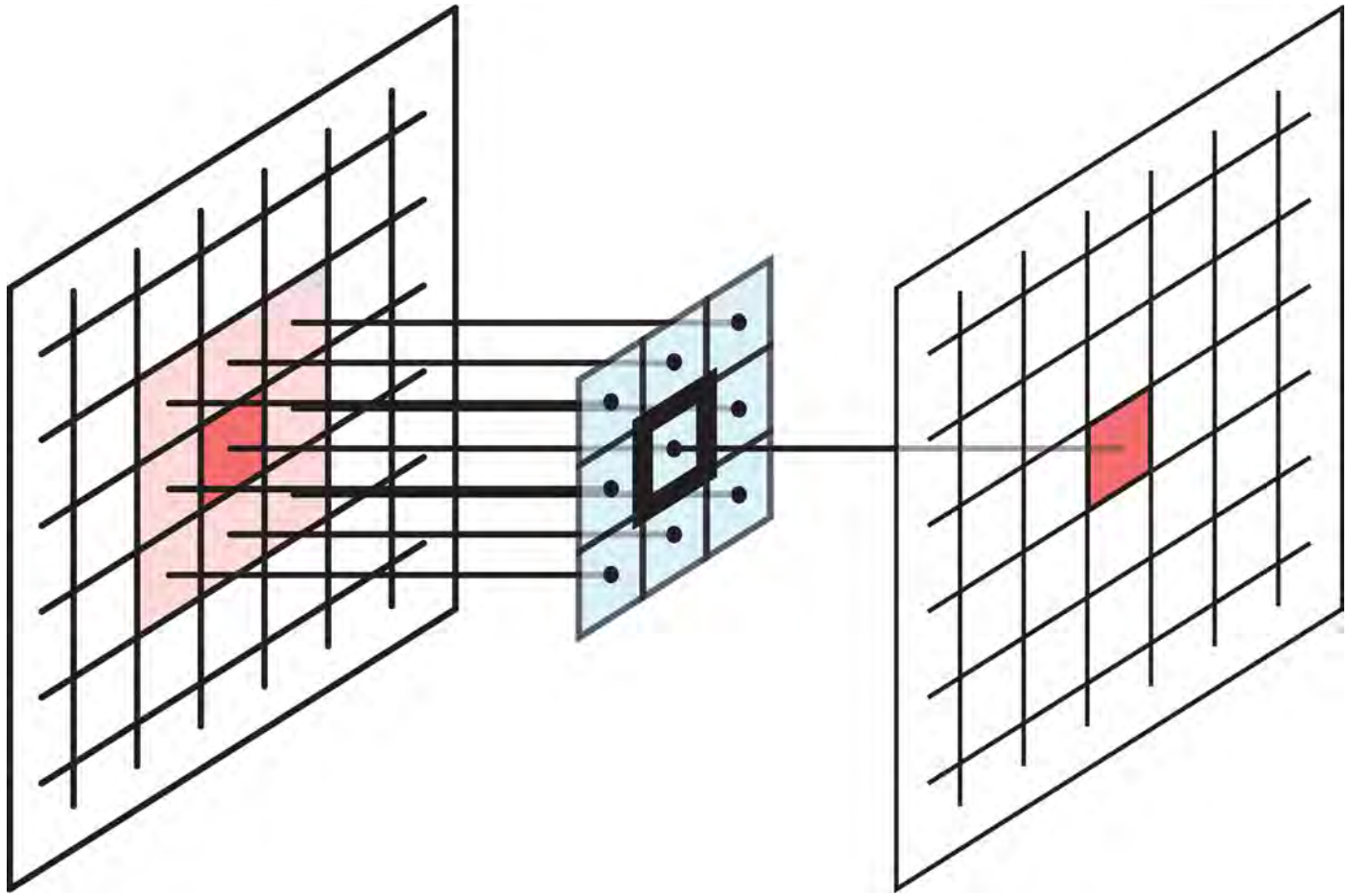
The more yellow the pixel in the colour image (left), the more white it is in the grayscale image.

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



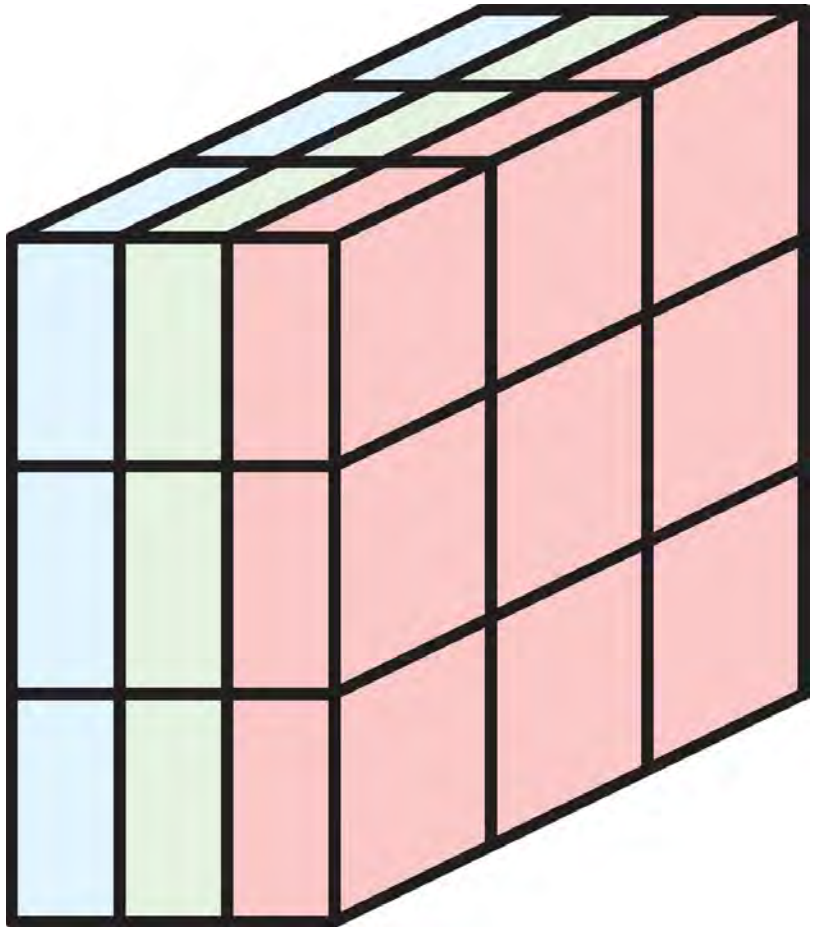
Example 3x3 filter

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

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

Multidimensional convolution

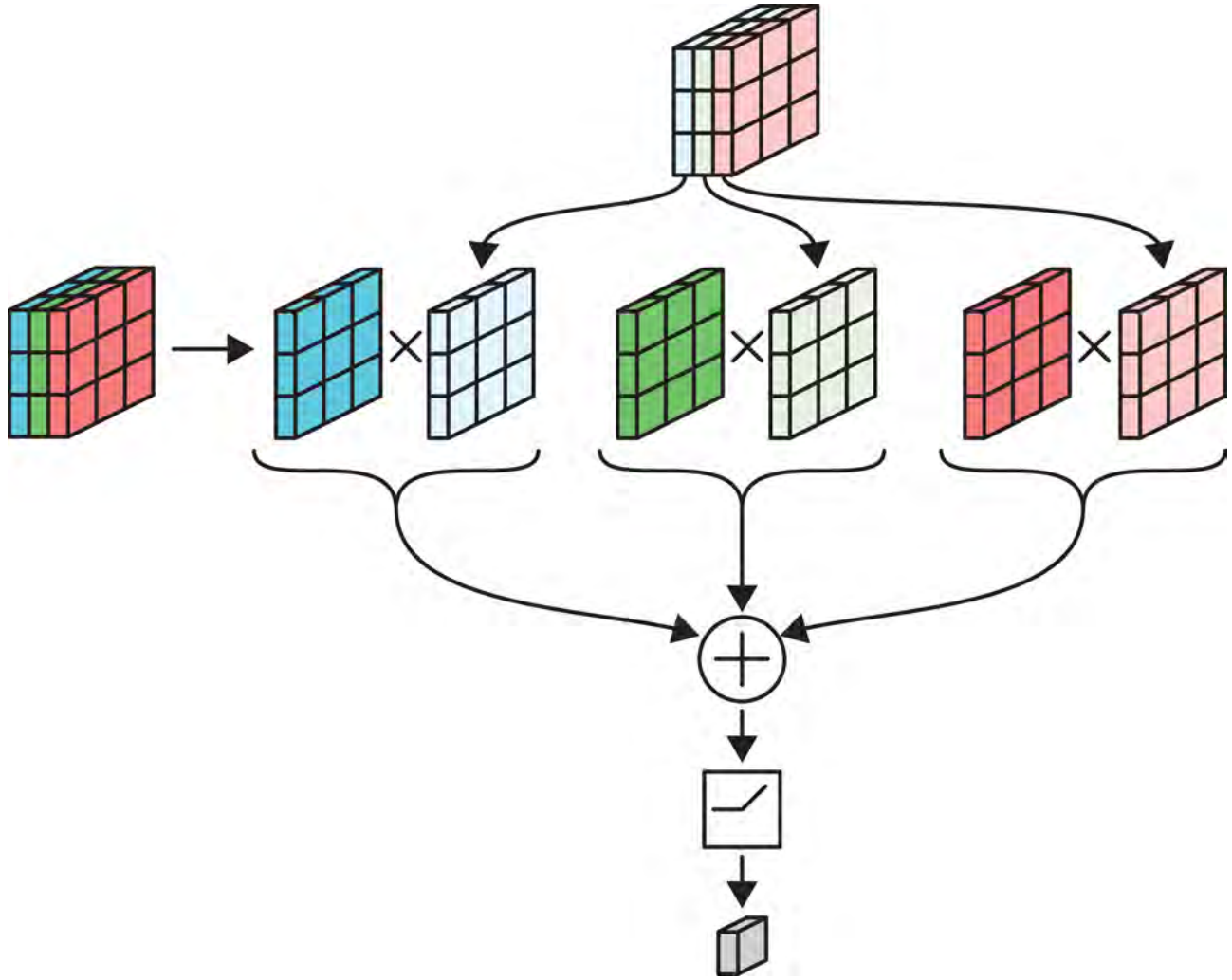
Need # Channels in Input = # Channels in Filter.



Example: a 3x3 filter with 3 channels, containing 27 weights.

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

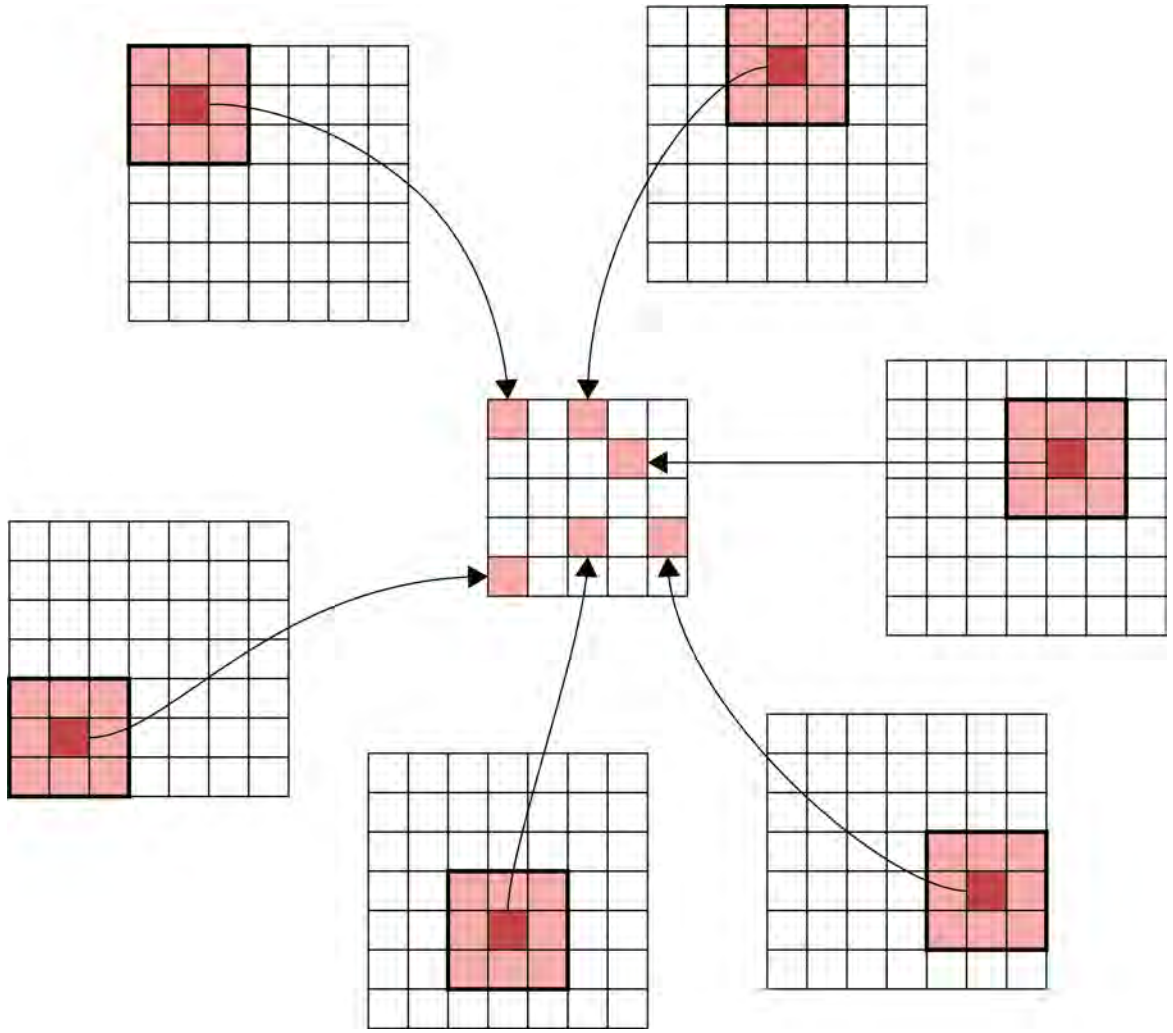
Example: 3x3 filter over RGB input



Each channel is multipled separately & then added together.

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

Input-output relationship



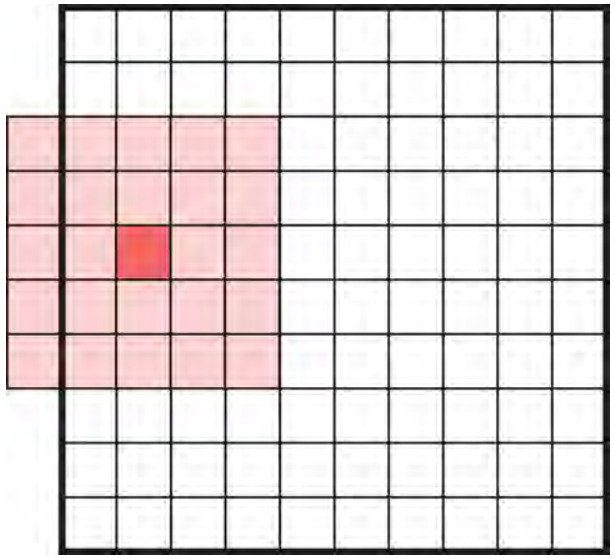
Matching the original image footprints against the output location.

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



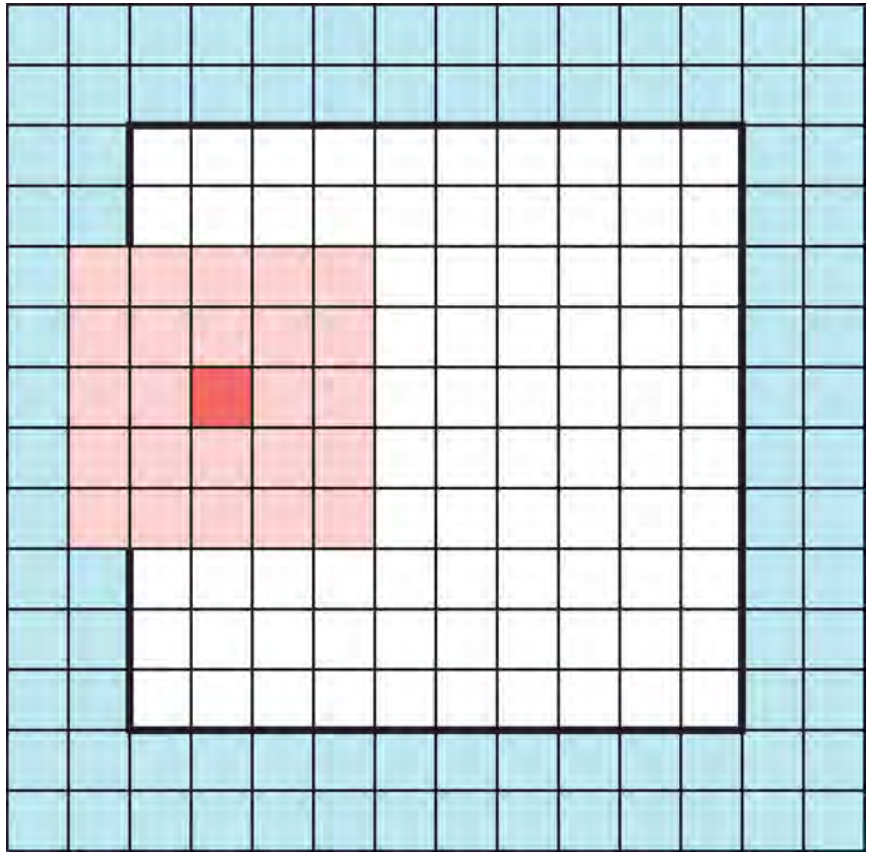
What happens when filters go off the edge of the input?

- 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.

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

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.

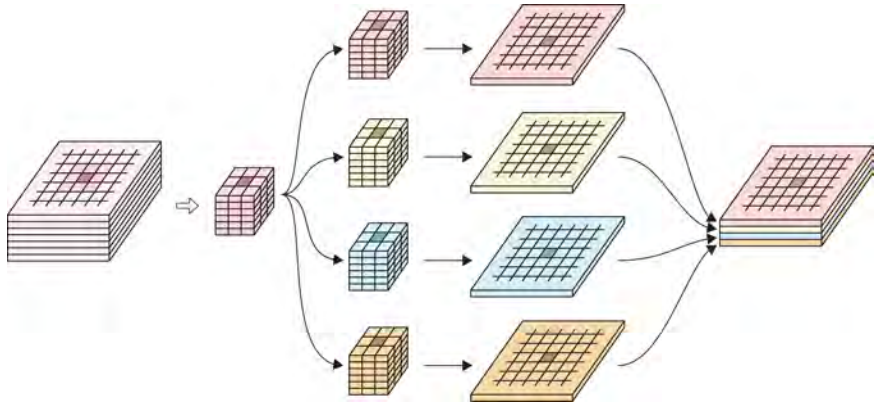
Convolution layer

- 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 3x3 filters
- four output tensors
- final output tensor.



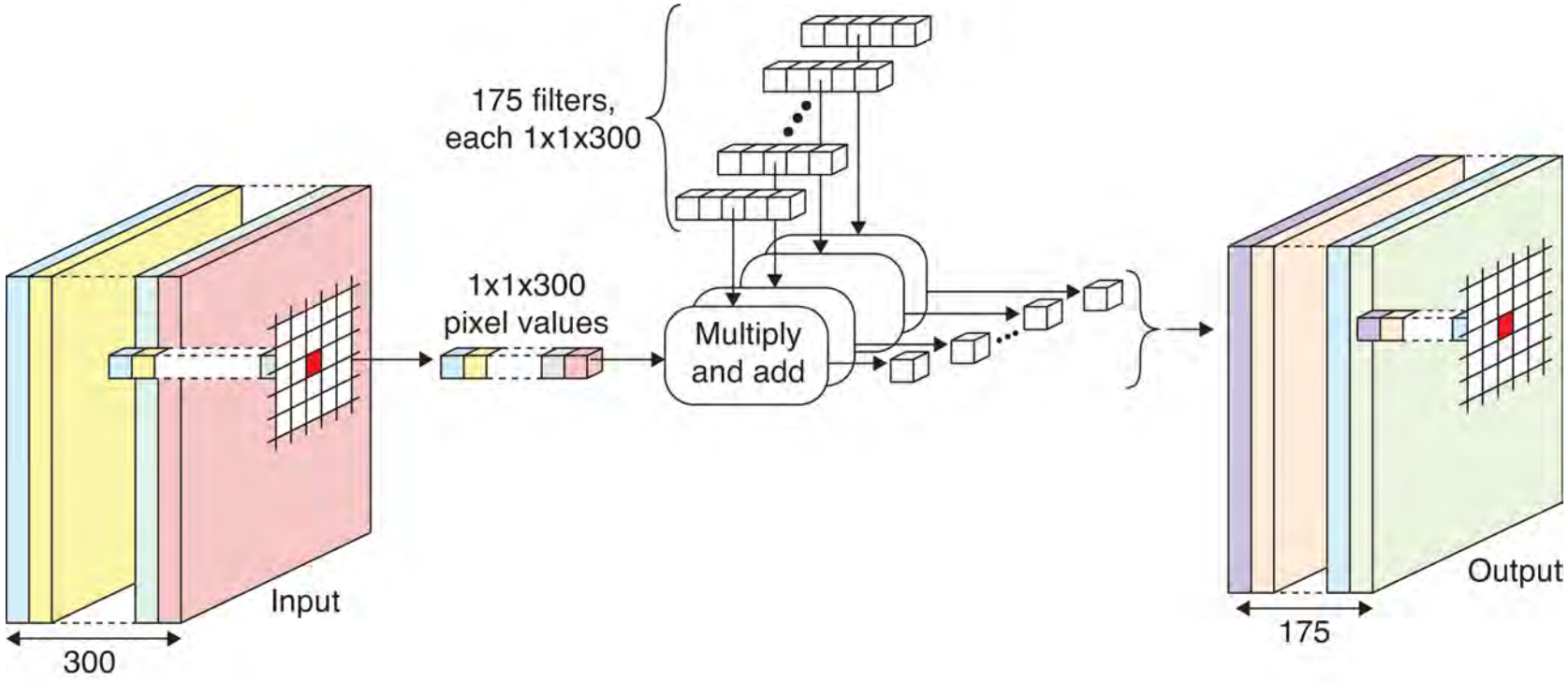
Example network highlighting that the number of output channels equals the number of filters.

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

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



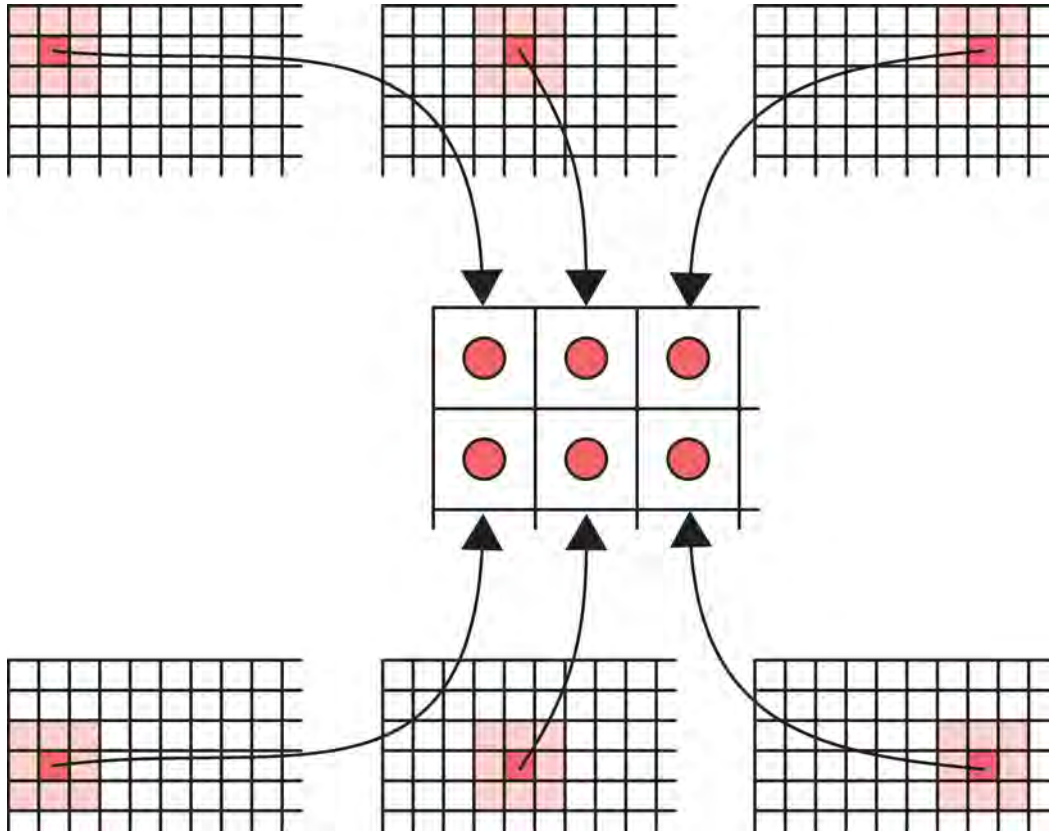
Example network with 1x1 convolution.

- 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.

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

Striding

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

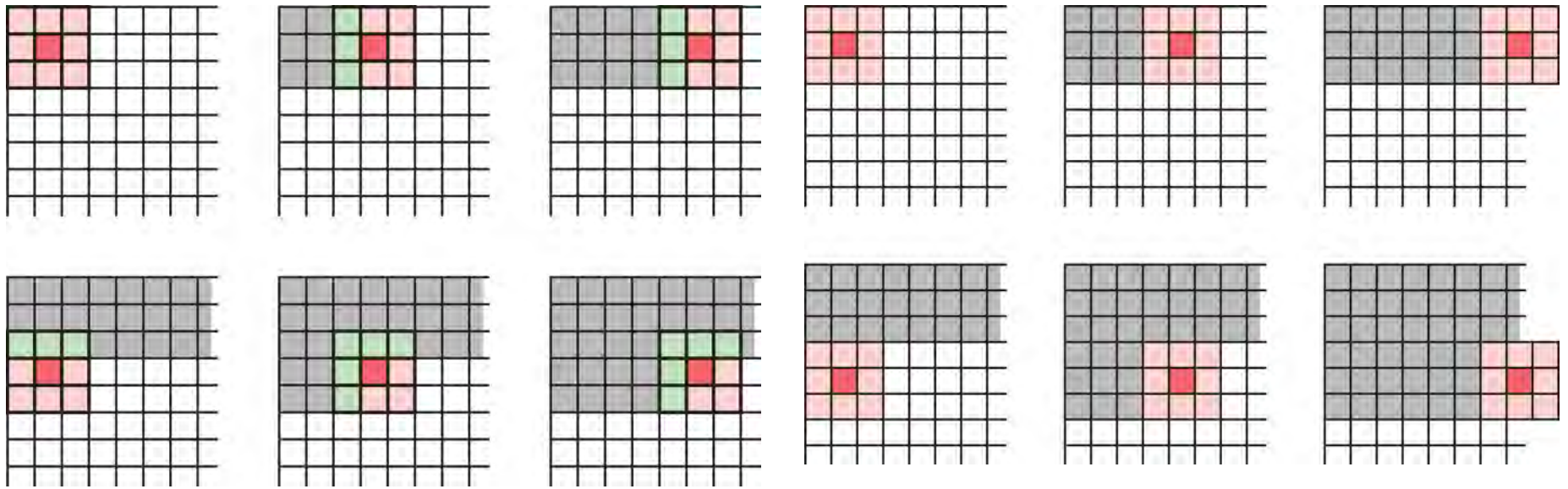


Example: Use a stride of three horizontally and two vertically.

Dimension of output will be smaller than input.

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

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 prevents input elements from being used more than once. Example (right image): 3x3 filter, stride 3 in both directions.

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

Specifying a convolutional layer

Need to choose:

- number of filters,
- their footprints (e.g. 3x3, 5x5, etc.),
- activation functions,
- padding & striding (optional).

All the filter weights are learned during training.

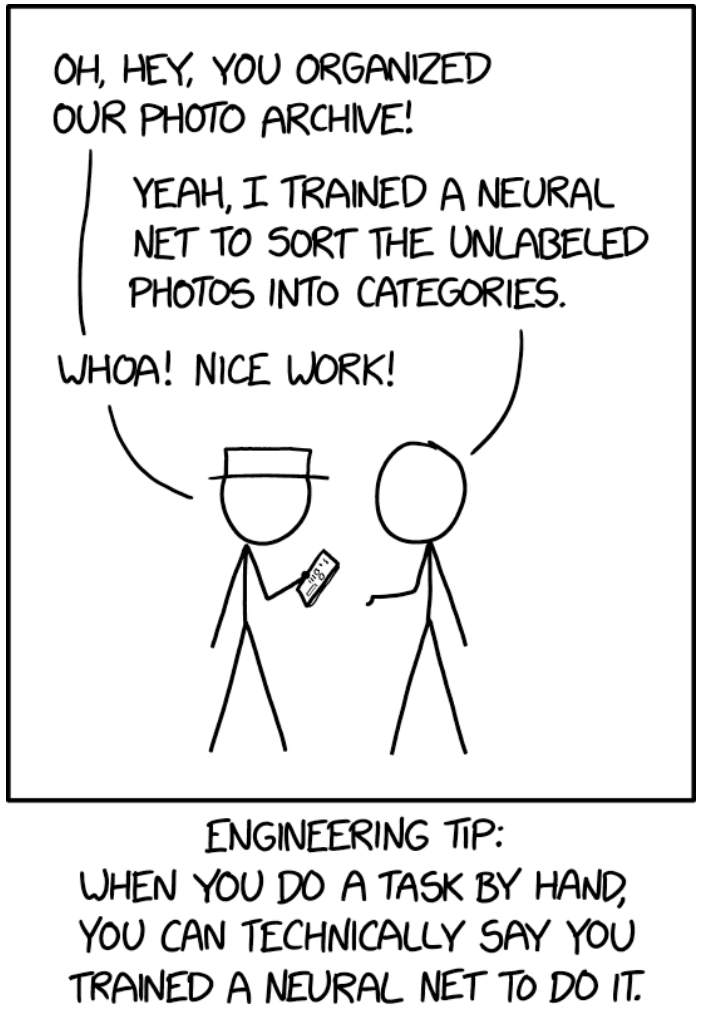
Lecture Outline

- 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

UNSW SYDNEY

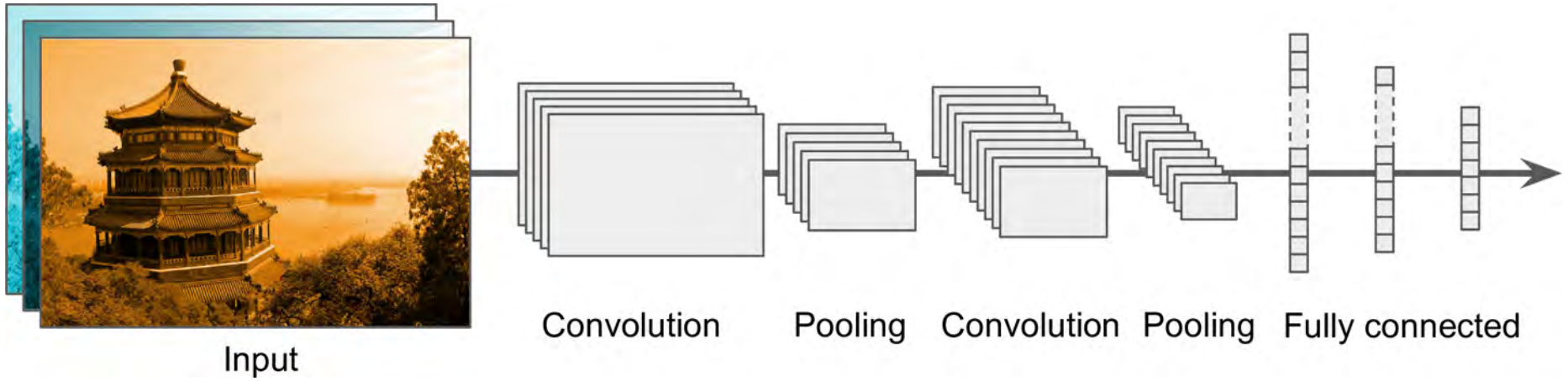
Definition of CNN

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



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

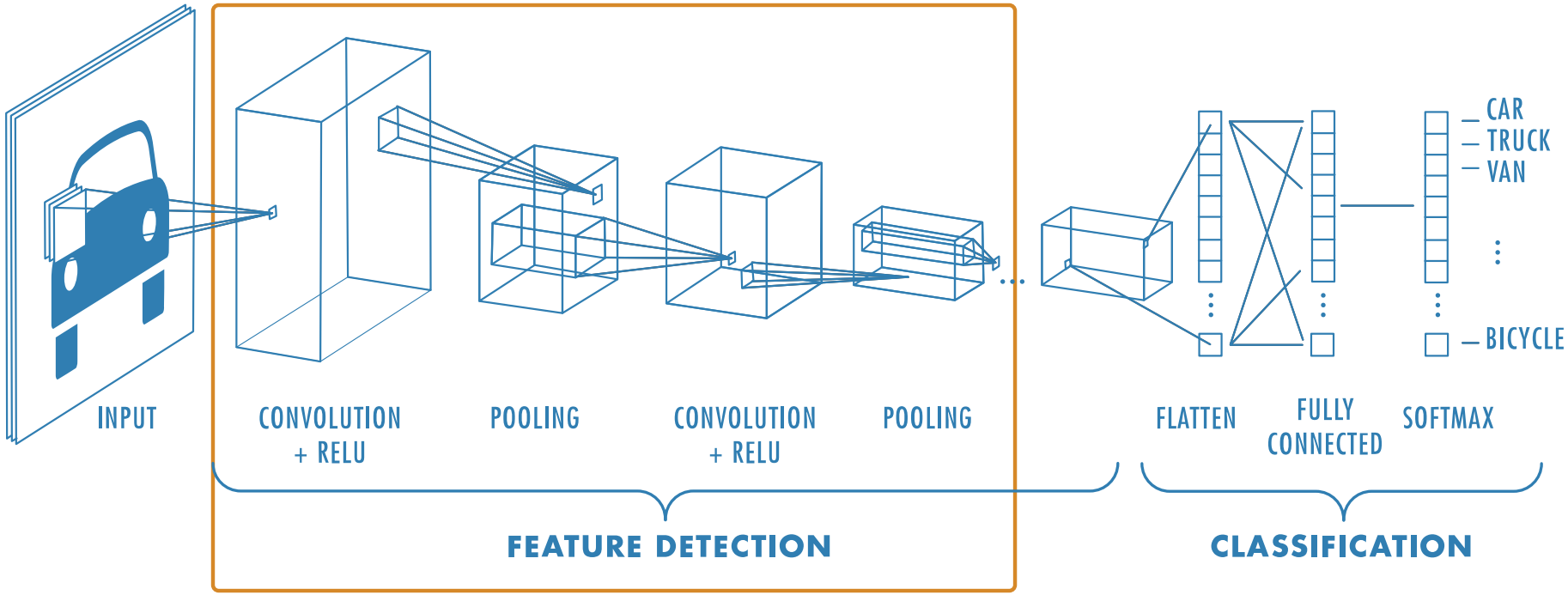
Architecture



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.

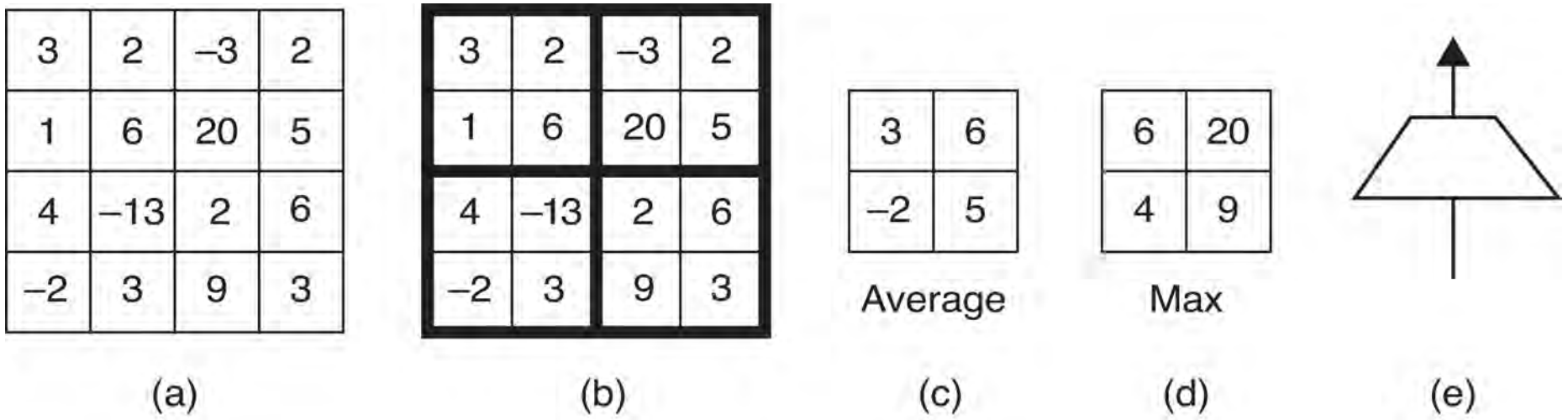
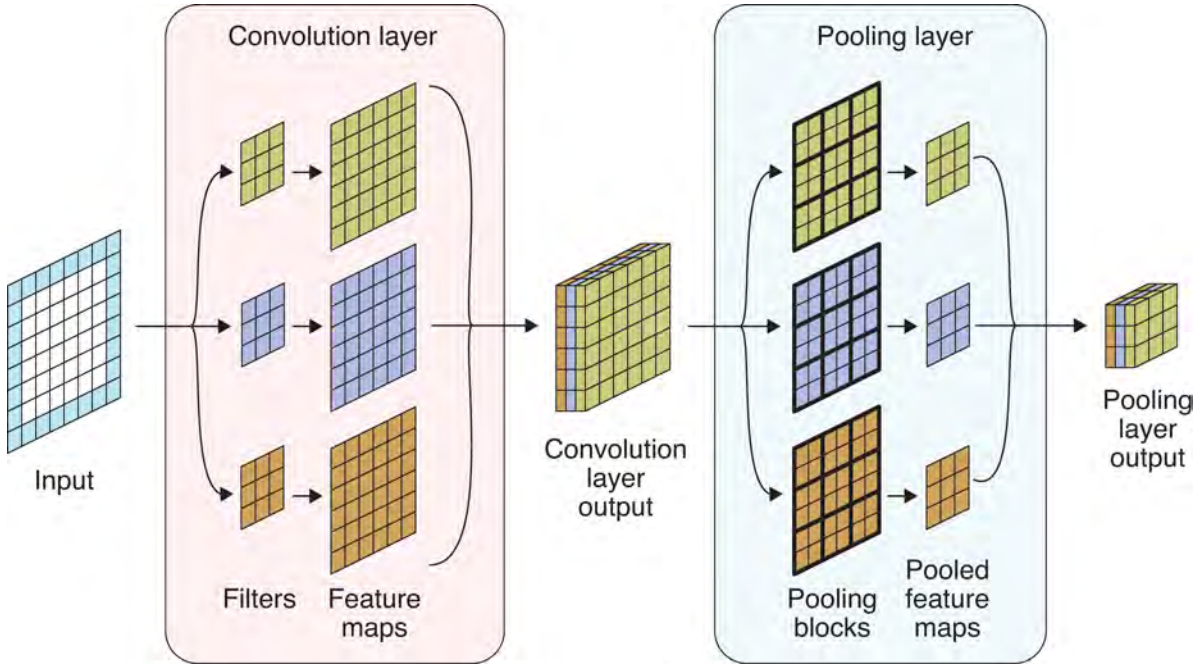


Illustration of pool operations.

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

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

Pooling for multiple channels



Pooling a multichannel input.

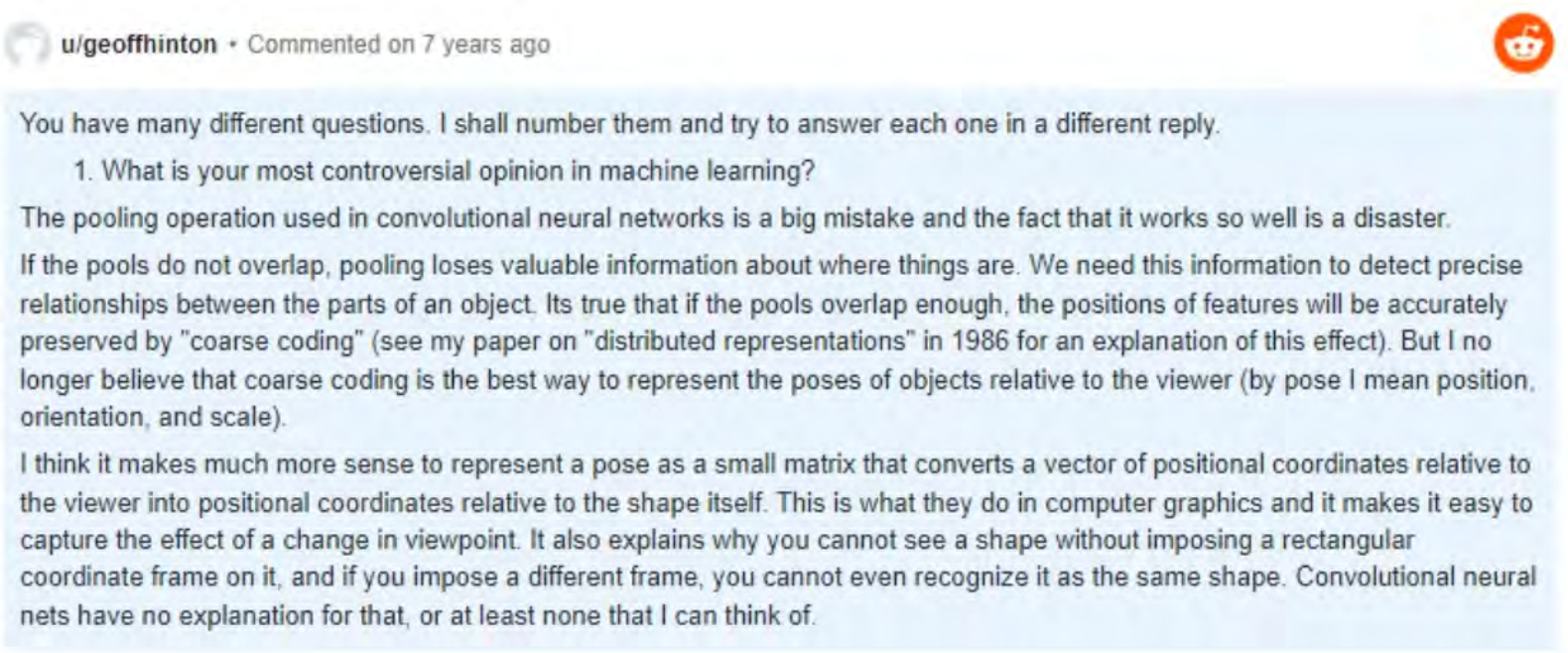
- Input tensor: 6x6 with 1 channel, zero padding.
- Convolution layer: Three 3x3 filters.
- Convolution layer output: 6x6 with 3 channels.
- Pooling layer: apply max pooling to each channel.
- Pooling layer output: 3x3, 3 channels.

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

Why/why not use pooling?

Why? Pooling *reduces the size* of tensors, therefore reduces memory usage and execution time (recall that 1x1 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.

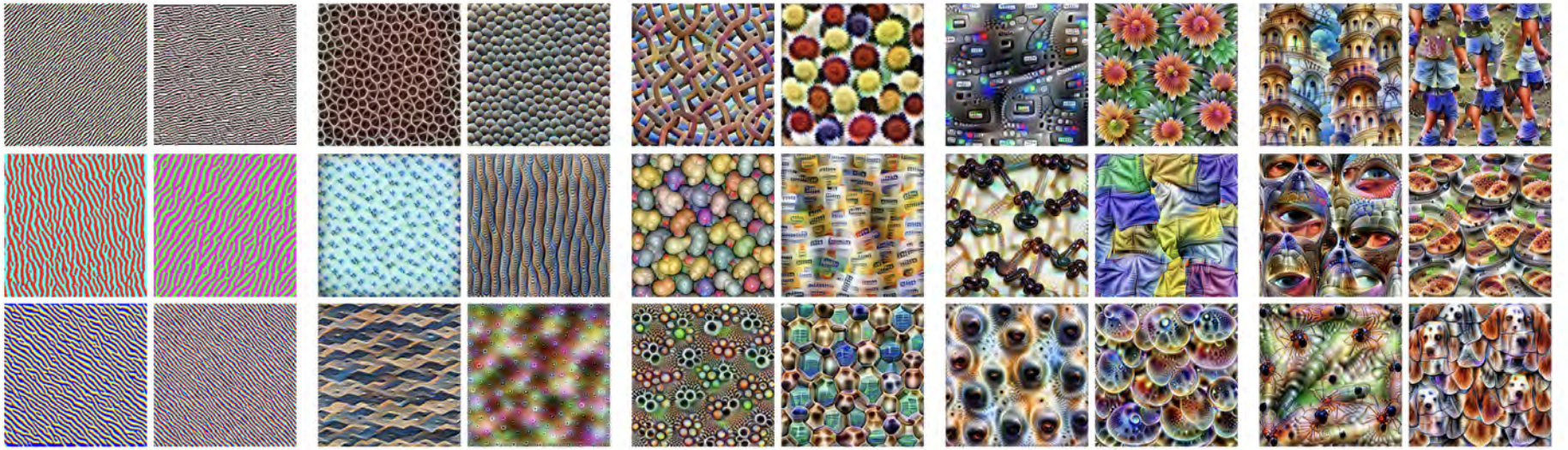
If the pools do not overlap, pooling loses valuable information about where things are. We need this information to detect precise relationships between the parts of an object. Its true that if the pools overlap enough, the positions of features will be accurately preserved by "coarse coding" (see my 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.

What do the CNN layers learn?



Source: Distill article, Feature Visualization.

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.

CASIA Chinese handwriting database

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

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

Source: Liu et al. (2011), CASIA online and offline Chinese handwriting databases, 2011 International Conference on Document Analysis and Recognition.

Inspect a subset of characters

Pulling out 55 characters to experiment with.

„ щ Ș , ʿ ~ ő ṫ ɭ ɟ ú
ψ ǩ Ɽ - ˊ S ķ ļ̃ ɛ Î
š Ì °C ˈ ʤ _ È ę
Ĝ ? « ĵ ť

Inspect directory structure

```
1 !pip install directory_tree
```

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

```
CASIA-Dataset/
├── Test/
│   ├── š /
│   │   ├── 1.png
│   │   ├── 10.png
│   │   ├── 100.png
│   │   ├── 101.png
│   │   ├── 102.png
│   │   ├── 103.png
│   │   ├── 104.png
│   │   ├── 105.png
│   │   ├── 106.png
...
│   │   ├── 97.png
│   │   ├── 98.png
│   │   └── 99.png
```

UNSW SYDNEY

Count number of images for each character

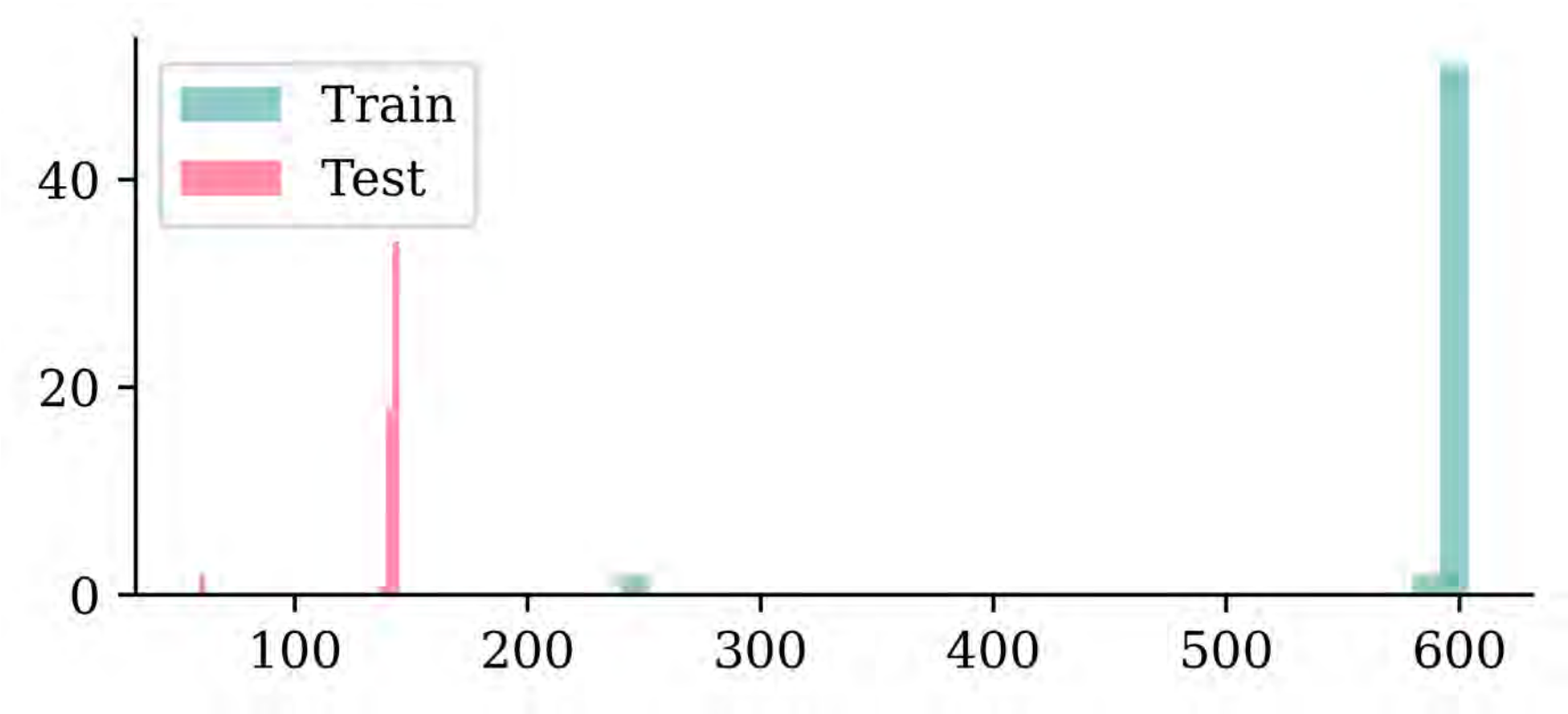
```python
def count_images_in_folders(root_folder):
    counts = {}
    for folder in root_folder.iterdir():
        counts[folder.name] = len(list(folder.glob("*.png")))
    return counts

train_counts = count_images_in_folders(Path("CASIA-Dataset/Train"))
test_counts = count_images_in_folders(Path("CASIA-Dataset/Test"))

print(train_counts)
print(test_counts)
```

```
{'  ': 584, '  ': 597, 'Ĝ': 597, '  ': 240, '  ': 240, '°C': 596, 'ŀ ': 596, 'Ɽ ': 598, '« ':
600, 'Ļ ': 597, '- ': 604, '‘ ': 599, 'ɭ ': 603, 'ň ': 598, 'Ì ': 604, 'ŕ ': 593, '  ': 599,
'ɟ ': 602, '  ': 602, '′ ': 600, 'ę ': 597, '  ': 598, '  ': 601, 'щ': 598, ́ ': 591, '  ':
597, 'ɛ ': 598, '  ': 598, 'š ': 601, '„ ': 597, 'È ': 601, '  ': 598, 'Î ': 597, '  ': 600,
'  ': 601, ̲ ': 597, '  ': 596, 'ψ': 598, '  ': 597, ', ': 603, ̃ ': 598, '  ': 595, '  ':
602, 'ʣ': 604, '  ': 595, 'ő ': 597, '  ': 601, 'S ': 598, '? ': 601, 'Ș ': 600, '  ': 600,
'  ': 602, 'ķ ': 602, 'ĵ ': 599, 'ú ': 599}
{'  ': 138, '  ': 143, 'Ĝ': 144, '  ': 60, '  ': 59, '°C': 144, 'ŀ ': 143, 'Ɽ ': 144, '« ':
142, 'Ļ ': 144, '- ': 143, '‘ ': 141, 'ɭ ': 144, 'ň ': 143, 'Ì ': 144, 'ŕ ': 142, '  ': 144,
'ɟ ': 143, '  ': 142, '′ ': 141, 'ę ': 143, '  ': 141, '  ': 140, 'щ': 142, ́ ': 144, '  ':
141, 'ɛ ': 143, '  ': 143, 'š ': 142, '„ ': 144, 'È ': 143, '  ': 142, 'Î ': 143, '  ': 142,
'  ': 141, ̲ ': 144, '  ': 143, 'ψ': 144, '  ': 143, ', ': 144, ̃ ': 143, '  ': 142, '  ':
144, 'ʣ': 141, '  ': 144, 'ő ': 143, '  ': 143, 'S ': 144, '? ': 145, 'Ș ': 143, '  ': 144,
'  ': 143, 'ķ ': 142, 'ĵ ': 141, 'ú ': 142}
```

UNSW SYDNEY

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();
```



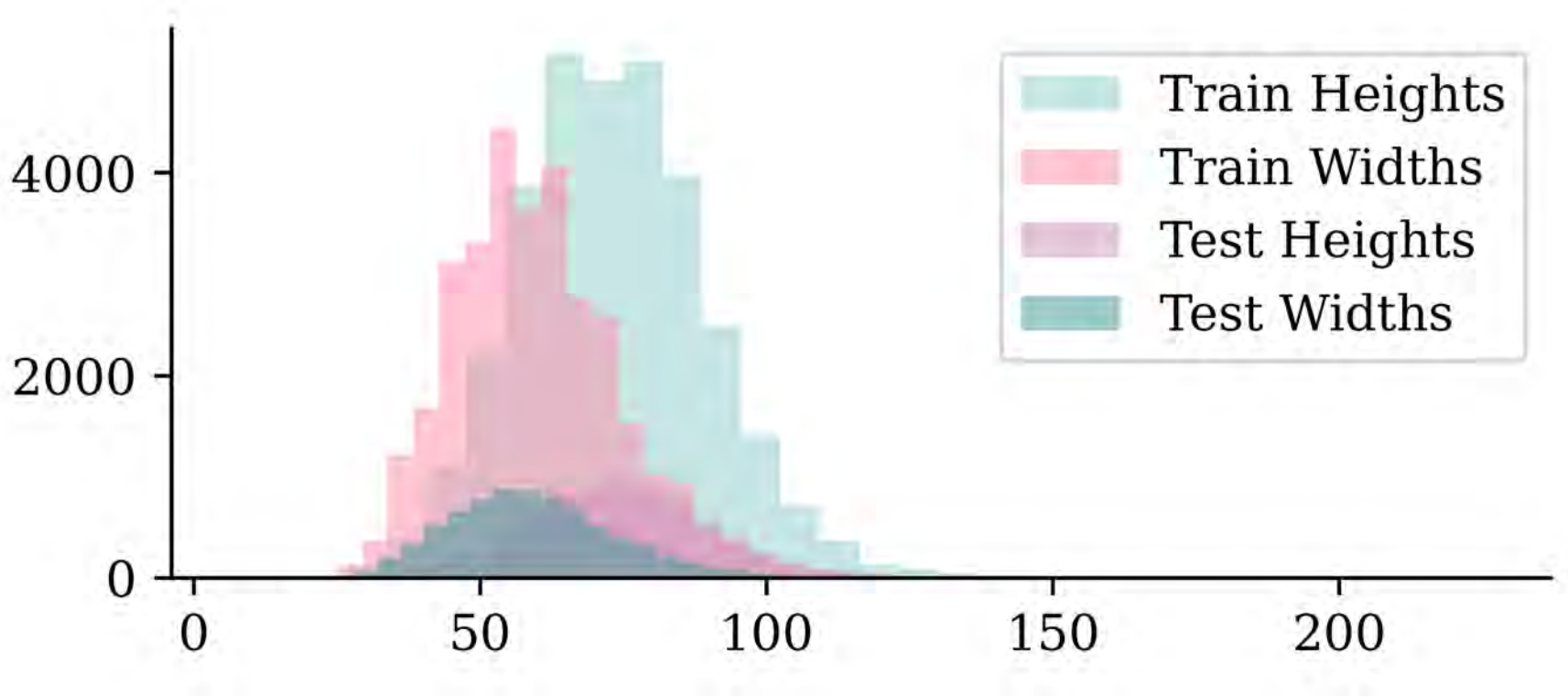
It differs, but basically ~600 training and ~140 test images per character. A couple of characters have a lot less of both though.

Checking the dimensions

```python
def get_image_dimensions(root_folder):
    dimensions = []
    for folder in root_folder.iterdir():
        for image in folder.glob("*.png"):
            img = imread(image)
            dimensions.append(img.shape)
    return dimensions

train_dimensions = get_image_dimensions(Path("CASIA-Dataset/Train"))
test_dimensions = get_image_dimensions(Path("CASIA-Dataset/Test"))

train_heights = [d[0] for d in train_dimensions]
train_widths = [d[1] for d in train_dimensions]
test_heights = [d[0] for d in test_dimensions]
test_widths = [d[1] for d in test_dimensions]
```

Checking the dimensions II

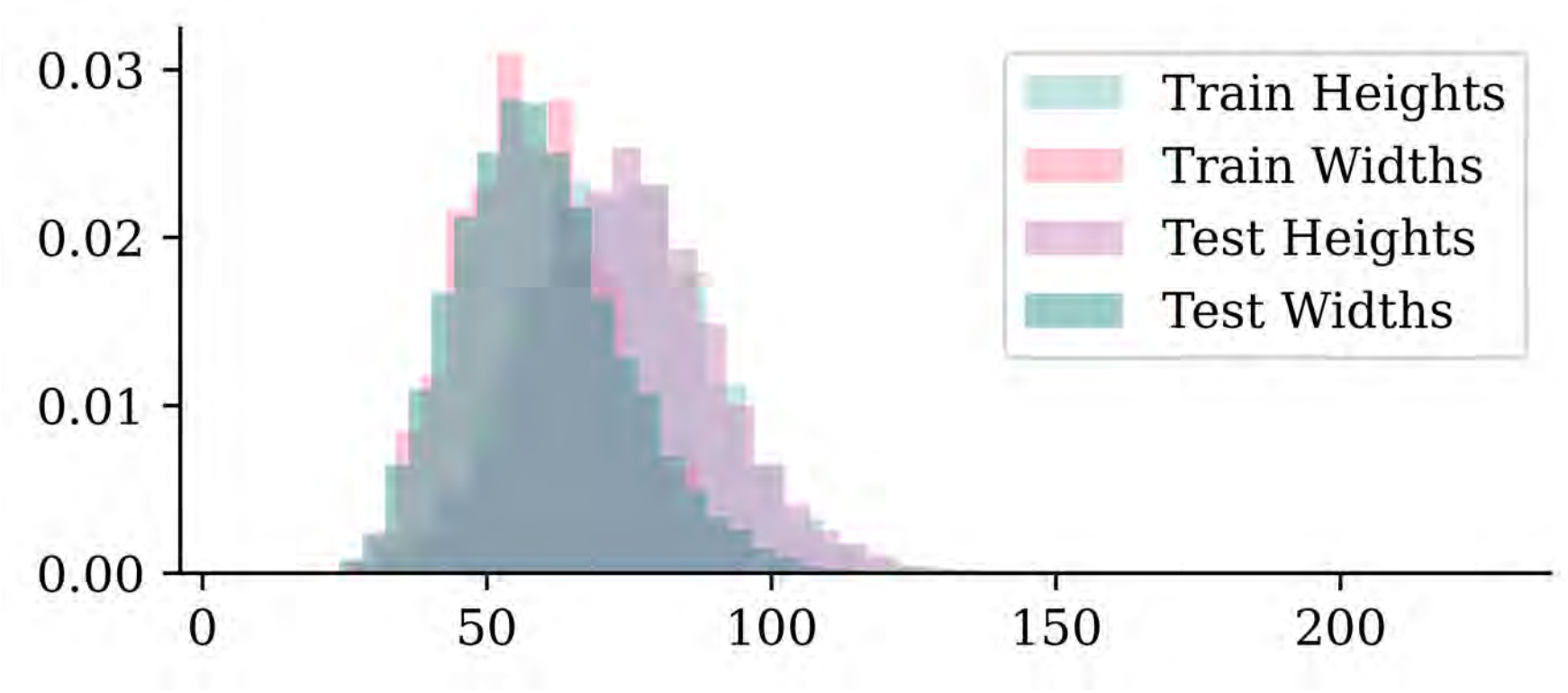
```
1 plt.hist(train_heights, bins=30, alpha=0.5, label="Train Heights")
2 plt.hist(train_widths, bins=30, alpha=0.5, label="Train Widths")
3 plt.hist(test_heights, bins=30, alpha=0.5, label="Test Heights")
4 plt.hist(test_widths, bins=30, alpha=0.5, label="Test Widths")
5 plt.legend();
```



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

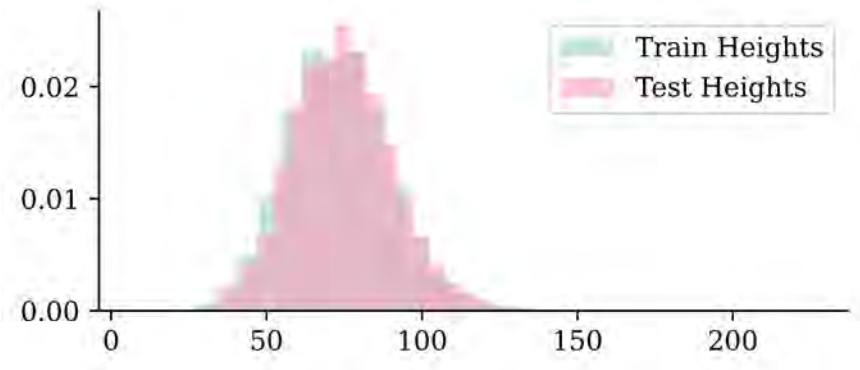
Checking the dimensions III

```
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();
```

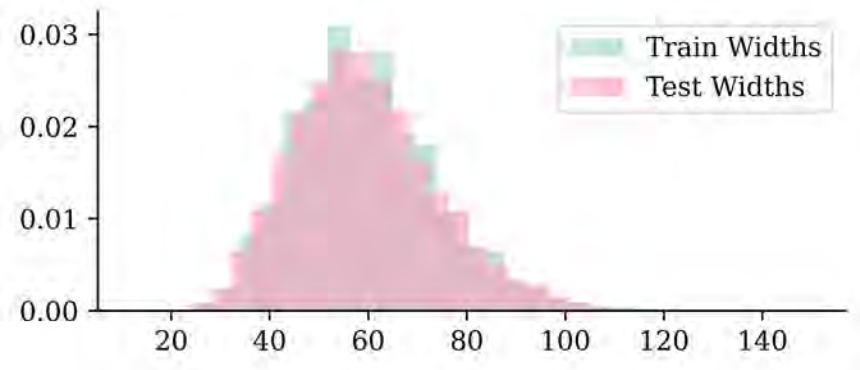


Checking the dimensions IV

```
1 plt.hist(train_heights, bins=30, alpha=0
2 plt.hist(test_heights, bins=30, alpha=0.
3 plt.legend();
```




```
1 plt.hist(train_widths, bins=30, alpha=0.
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

Normally we'd used `keras.utils.image_dataset_from_directory` but the Chinese characters breaks it on Windows. I made an image loading function just for this demo.

```
data_dir = Path("CASIA-Dataset")
img_height, img_width = 80, 60  # Target image size

# Load 'training' and test datasets
X_main, y_main, class_names = load_images_from_directory(data_dir / "Train", img_height,
X_test, y_test, _ = load_images_from_directory(data_dir / "Test", img_height, img_width)

# Verify dataset shape
print(f"Train: X={X_main.shape}, y={y_main.shape}")
print(f"Test: X={X_test.shape}, y={y_test.shape}")
print("Class Names:", class_names)
```

```
Train: X=(32206, 80, 60, 1), y=(32206,)
Test: X=(7684, 80, 60, 1), y=(7684,)
Class Names: ['š ', 'ĵ ', '„ ', 'щ ', 'Ș ', 'ķ ', 'ő ', '  ', '  ', '  ', '  ', '  ', 'ļ ', '  ', '« ',
' ', '˜ ', 'ŧ ', '‘ ', 'ļ ', '  ', 'S ', 'ť ', '′ ', 'Î ', '°C', 'Ì ', '  ', 'ψ ', '- ', 'ň ',
'Γ ', 'ę ', 'ú ', '  ', '  ', '  ', '  ', '  ', '? ', 'ʤ', '  ', '  ', '  ', '  ', '  ', '  ',
'  ', '  ', '  ', '_ ', '  ', '  ', 'È ', '  ', 'Ĝ ', 'ŧ ', 'ɛ ']
```

UNSW SYDNEY

Some setup

```
X_train, X_val, y_train, y_val = train_test_split(X_main, y_main, test_size=0.2,
    random_state=123)
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,)
```

```
import matplotlib.font_manager as fm
CHINESE_FONT = fm.FontProperties(fname="STHeitiTC-Medium-01.ttf")

def plot_mandarin_characters(X, y, class_names, n=5, title_font=CHINESE_FONT):
    # Plot the first n images in X
    plt.figure(figsize=(10, 4))
    for i in range(n):
        plt.subplot(1, n, i + 1)
        plt.imshow(X[i], cmap="gray")
        plt.title(class_names[y[i]], fontproperties=title_font)
        plt.axis("off")
```

```
class_names[:5]
```

```
['š ', 'ĵ ', '„ ', 'щ', 'Ṣ ']
```

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

UNSW SYDNEY

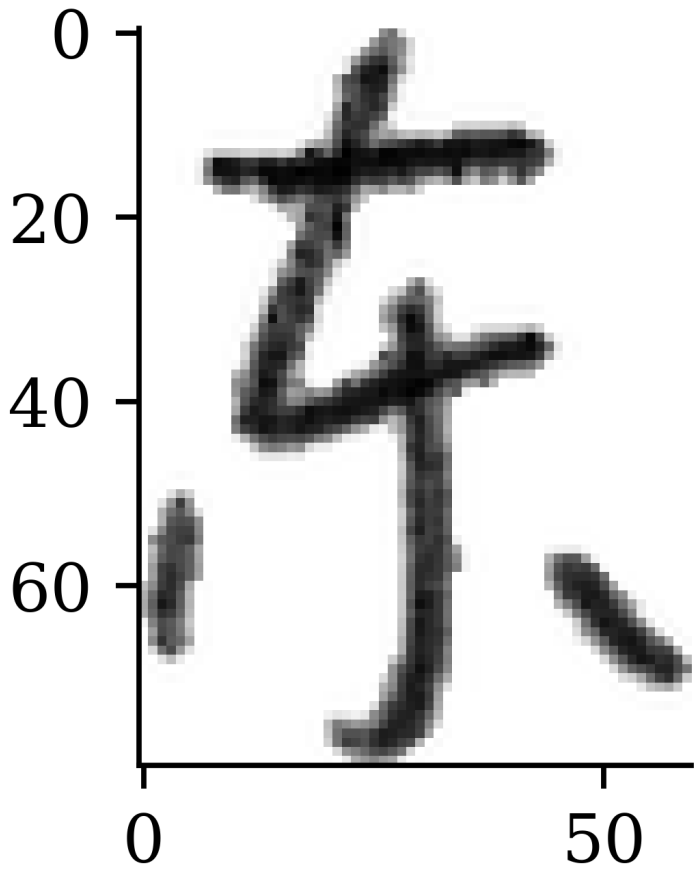
Plotting some training characters

东 东 东 东 东

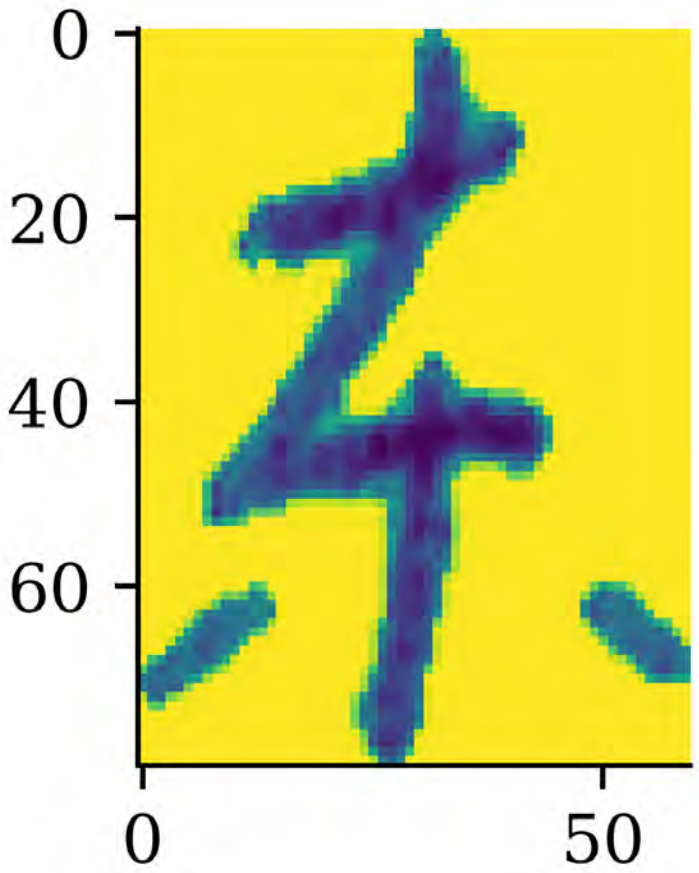
人 人 人 人 人

Without the colourmap..

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



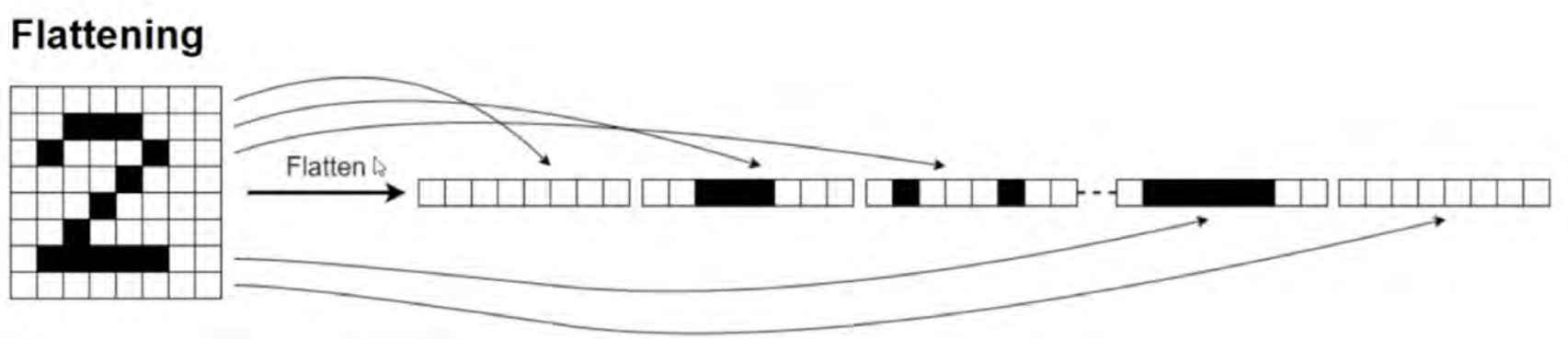

```
dong = X_test[y_test == 0][1]
plt.imshow(dong);
```



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



Basically pretend it's not an image

```
from keras.layers import Rescaling, Flatten

num_classes = np.unique(y_train).shape[0]
random.seed(123)
model = Sequential([
  Input((img_height, img_width, 1)), Flatten(), Rescaling(1./255),
  Dense(num_classes, activation="softmax")
])
```

Tip

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

Inspecting the model

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

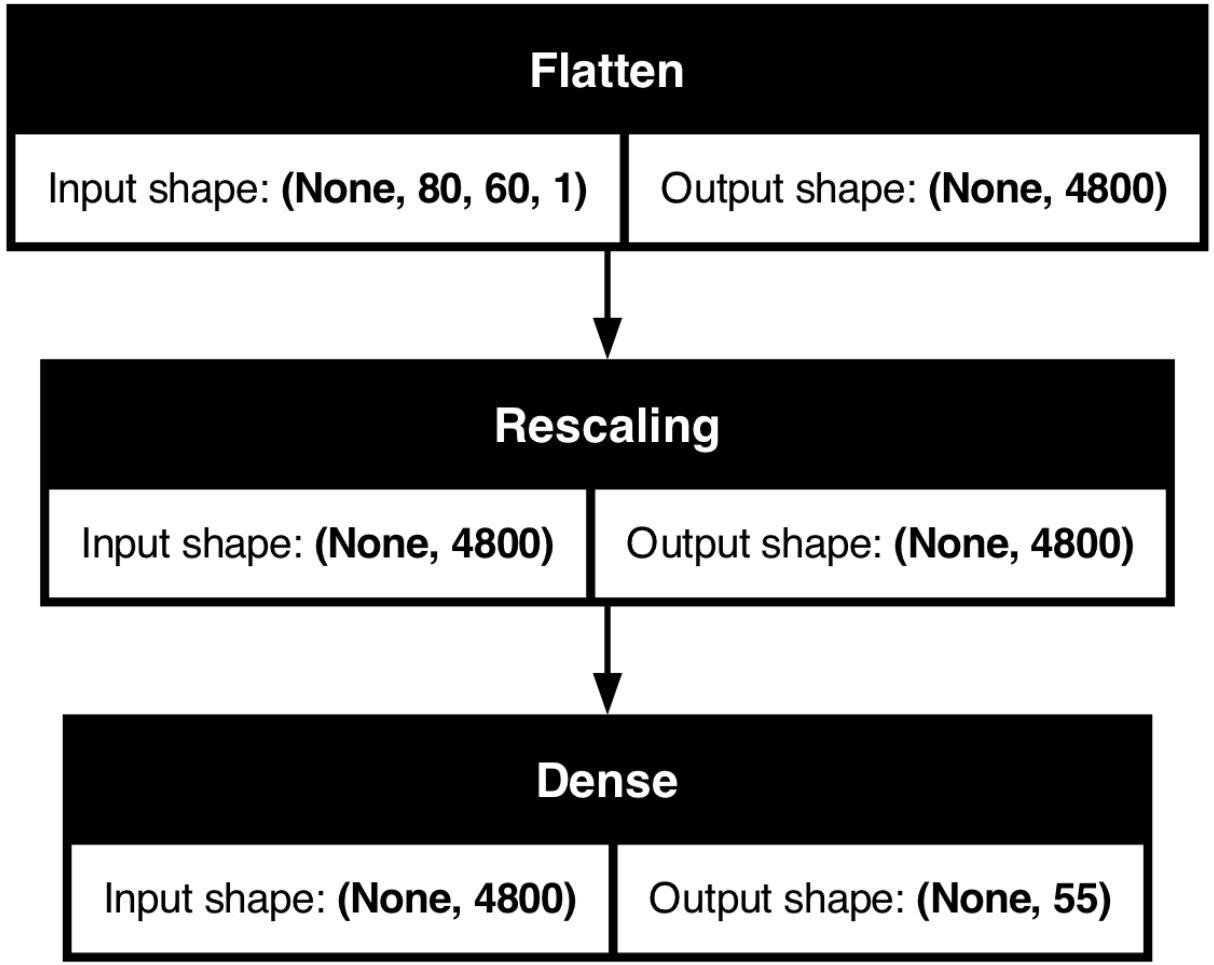
Model: "sequential"

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

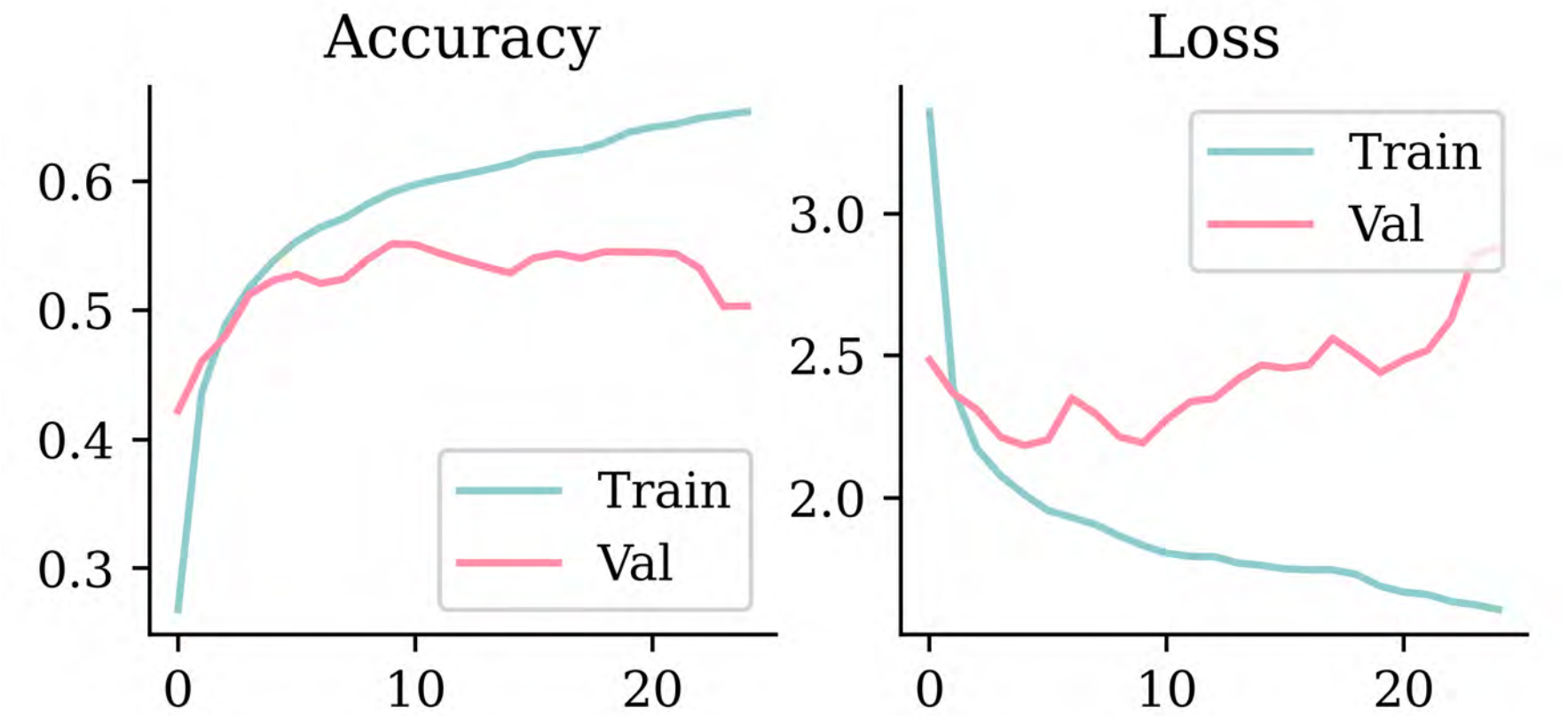
```
loss = keras.losses.SparseCategoricalCrossentropy()
topk = keras.metrics.SparseTopKCategoricalAccuracy(k=5)
model.compile(optimizer='adam', loss=loss, metrics=['accuracy', topk])

epochs = 100
es = EarlyStopping(patience=15, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)

if Path("logistic.keras").exists():
    model = keras.models.load_model("logistic.keras")
    with open("logistic_history.json", "r") as json_file:
        history = json.load(json_file)
else:
    hist = model.fit(X_train, y_train, validation_data=(X_val, y_val),
      epochs=epochs, callbacks=[es], verbose=0)
    model.save("logistic.keras")
    history = hist.history
    with open("logistic_history.json", "w") as json_file:
        json.dump(history, json_file)
```

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

UNSW SYDNEY

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))
```

```
[1.699511170387268, 0.6174119114875793, 0.8565052151679993]
[2.1941583156585693, 0.5513815879821777, 0.8146538138389587]
```

```
1 loss_value, accuracy, top5_accuracy = model.evaluate(X_val, y_val, 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: 2.1942
Validation Accuracy: 0.5514
Validation Top 5 Accuracy: 0.8147
```

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

```
from keras.layers import Conv2D, MaxPooling2D

random.seed(123)

model = Sequential([
  Input((img_height, img_width, 1)),
  Rescaling(1./255),
  Conv2D(16, 3, padding="same", activation="relu", name="conv1"),
  MaxPooling2D(name="pool1"),
  Conv2D(32, 3, padding="same", activation="relu", name="conv2"),
  MaxPooling2D(name="pool2"),
  Conv2D(64, 3, padding="same", activation="relu", name="conv3"),
  MaxPooling2D(name="pool3", pool_size=(4, 4)),
  Flatten(), Dense(64, activation="relu"), Dense(num_classes)
])
```

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

Inspect the model

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

Model: "sequential_1"

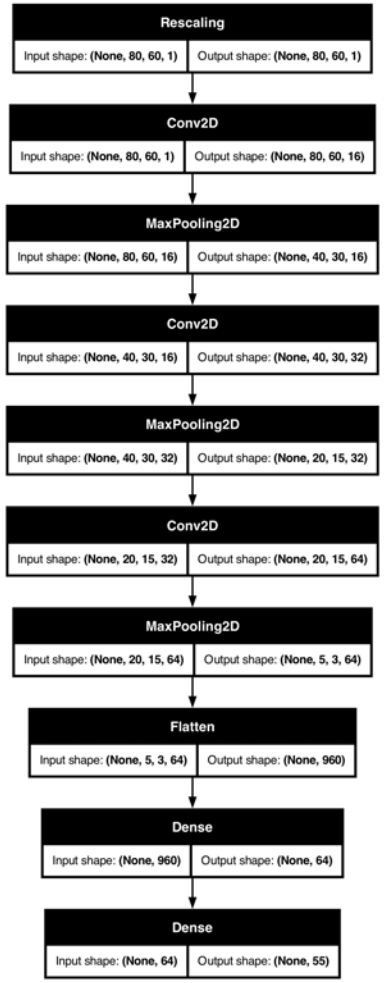
Layer (type)	Output Shape	Param #
rescaling_1 (Rescaling)	(None, 80, 60, 1)	0
conv1 (Conv2D)	(None, 80, 60, 16)	160
pool1 (MaxPooling2D)	(None, 40, 30, 16)	0
conv2 (Conv2D)	(None, 40, 30, 32)	4,640
pool2 (MaxPooling2D)	(None, 20, 15, 32)	0
conv3 (Conv2D)	(None, 20, 15, 64)	18,496
pool3 (MaxPooling2D)	(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)

UNSW SYDNEY

Plot the CNN

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



UNSW SYDNEY

Fit the CNN

```
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
topk = keras.metrics.SparseTopKCategoricalAccuracy(k=5)
model.compile(optimizer='adam', loss=loss, metrics=['accuracy', topk])

epochs = 100
es = EarlyStopping(patience=15, restore_best_weights=True,
    monitor="val_accuracy", verbose=2)

if Path("cnn.keras").exists():
    model = keras.models.load_model("cnn.keras")
    with open("cnn_history.json", "r") as json_file:
        history = json.load(json_file)
else:
    hist = model.fit(X_train, y_train, validation_data=(X_val, y_val),
      epochs=epochs, callbacks=[es], verbose=0)
    model.save("cnn.keras")
    history = hist.history
    with open("cnn_history.json", "w") as json_file:
        json.dump(history, json_file)
```

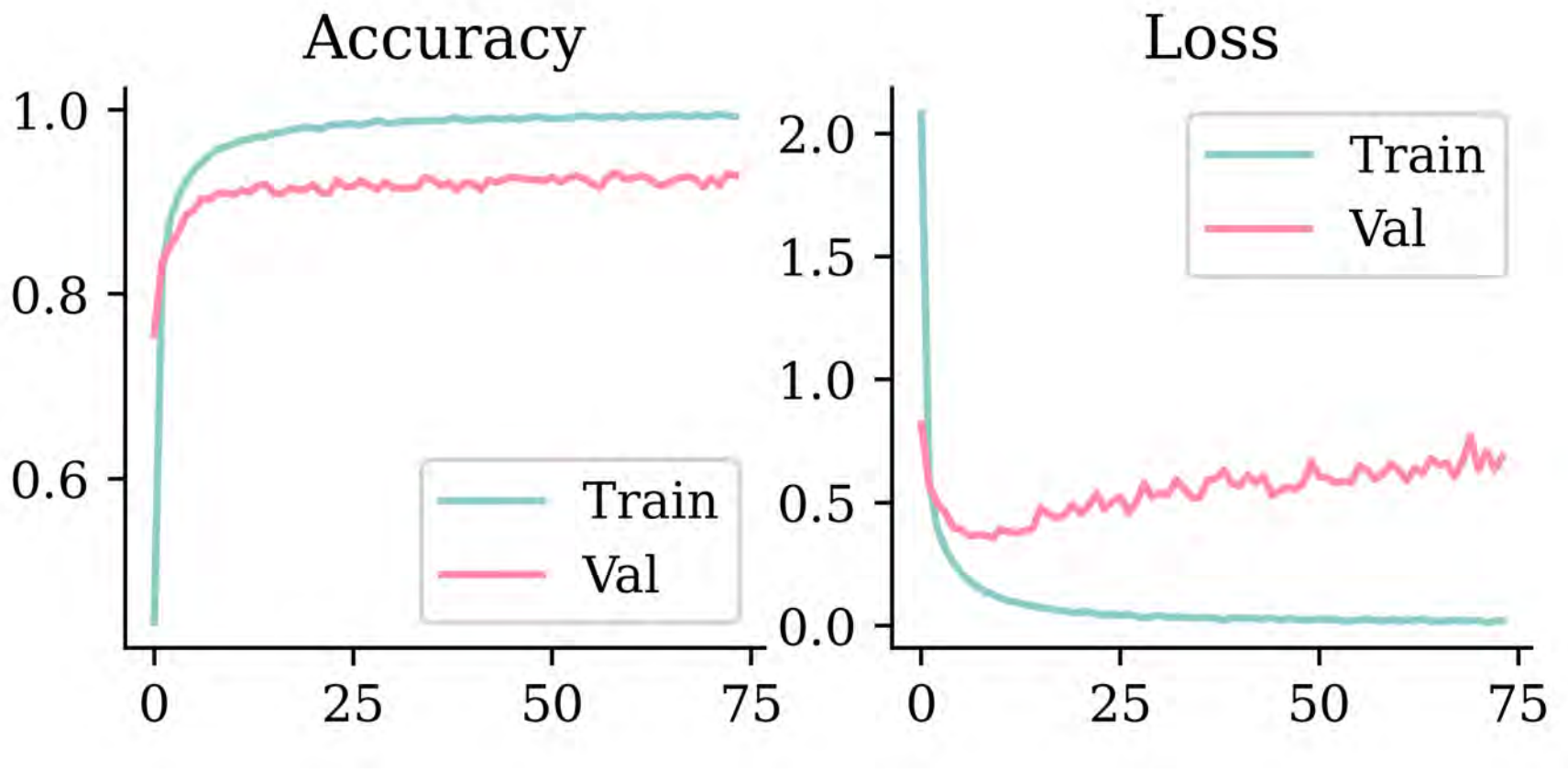
Tip

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

UNSW SYDNEY

Plot the loss/accuracy curves

```
1 plot_history(history)
```



Look at the metrics

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

```
[0.017913011834025383, 0.994255542755127, 1.0]
[0.6190589070320129, 0.932319164276123, 0.9928593635559082]
```

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

```
Test Loss: 1.1932
Test Accuracy: 0.8826
Test Top 5 Accuracy: 0.9845
```

UNSW SYDNEY

Make a prediction

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

```
ValueError                                Traceback (most recent call last)
Cell In[51], line 1
----> 1 model.predict(X_test[0], verbose=0);

File ~/anaconda3/envs/ai2025/lib/python3.11/site-packages/keras/src/utils/traceback_utils.py:12
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
    120     # To get the full stack trace, call:
    121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
    123 finally:
    124     del filtered_tb

File ~/anaconda3/envs/ai2025/lib/python3.11/site-packages/keras/src/utils/traceback_utils.py:12
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
    120     # To get the full stack trace, call:
    121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
    123 finally:
    124     del filtered_tb

ValueError: 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():
```

UNSW SYDNEY

Predict on the test set II

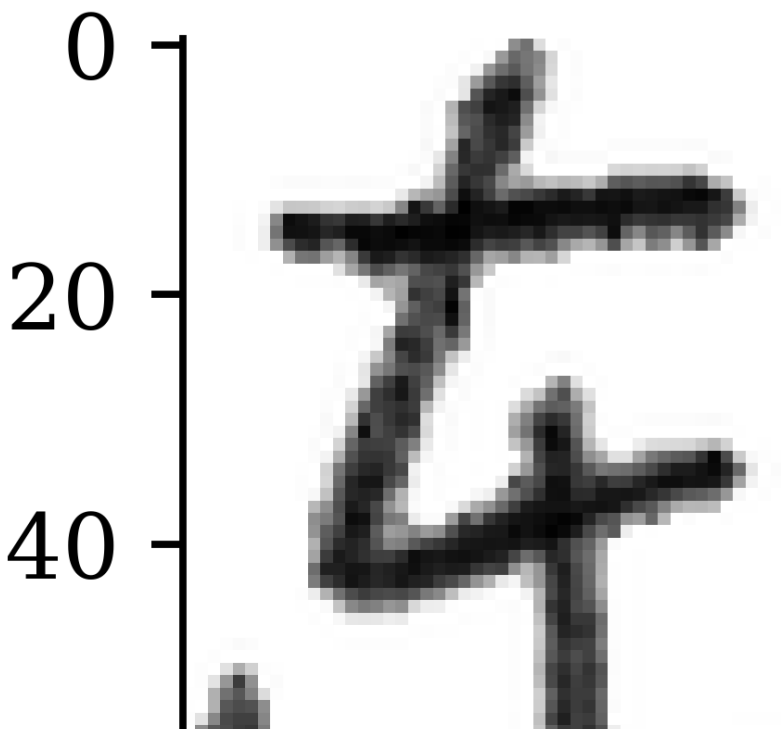
```
1 model.predict(X_test[[0]], verbose=0).argmax()
```

```
0
```

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

```
'š '
```

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

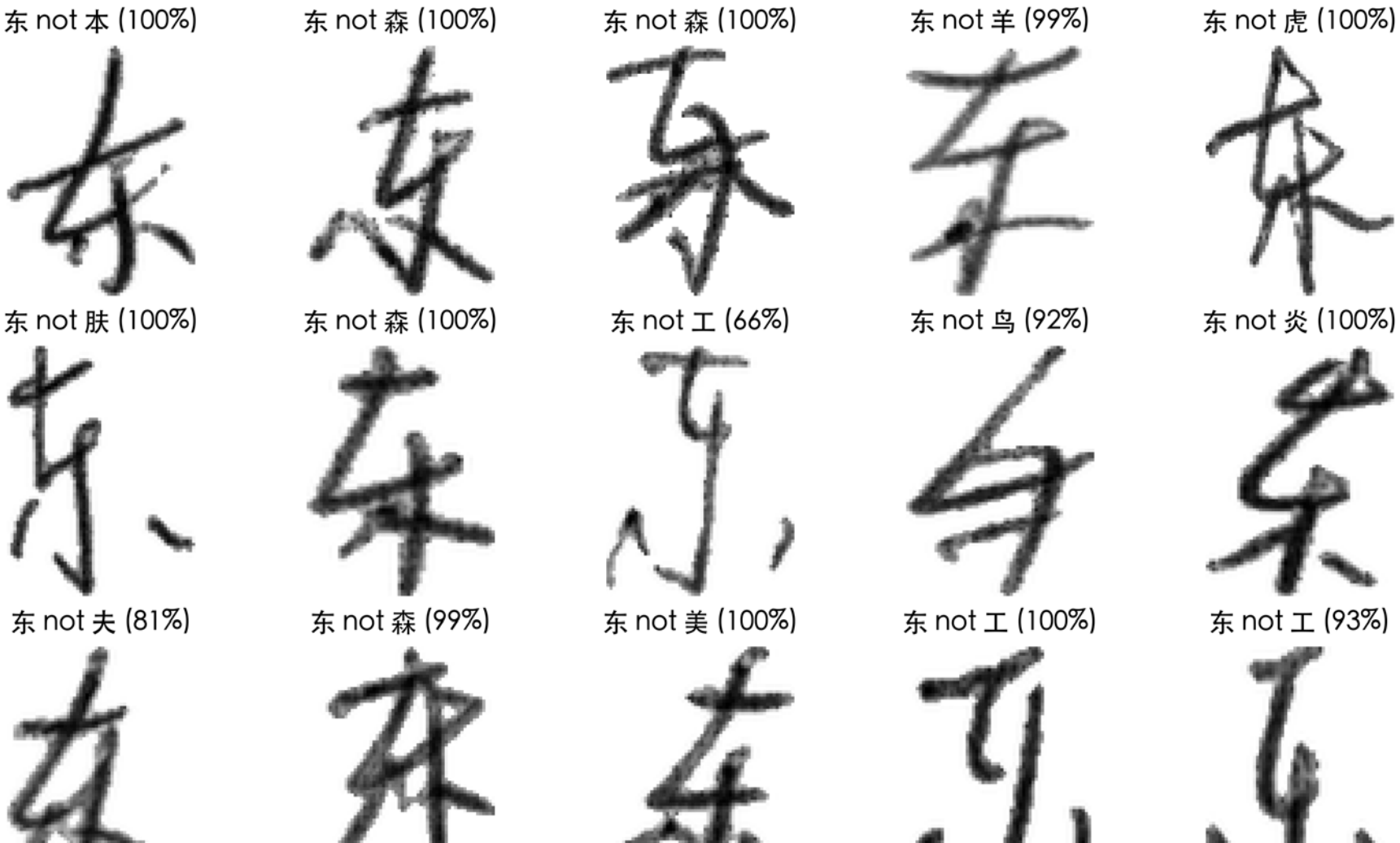


Lecture Outline

- 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

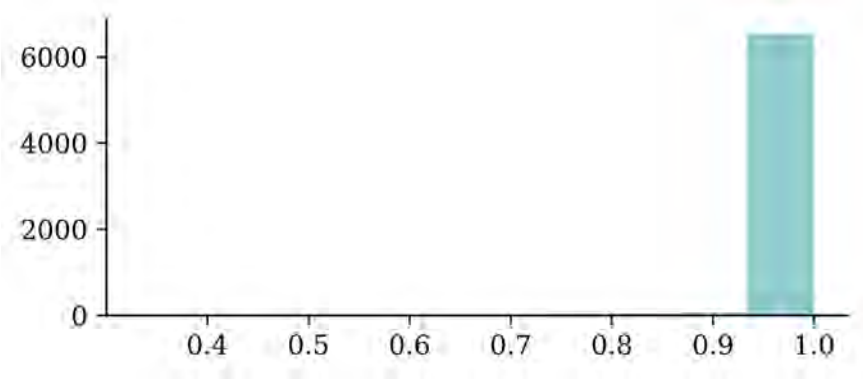
```
1 plot_failed_predictions(X_test, y_test, class_names)
```



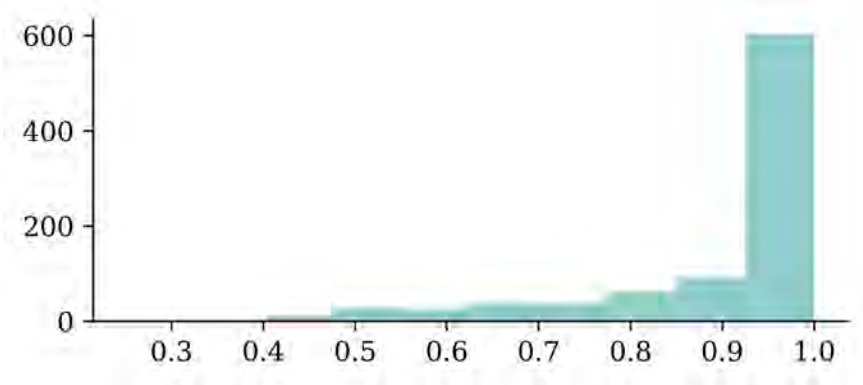
Confidence of predictions

```
y_log = model.predict(X_test, verbose=0)
y_pred = keras.ops.convert_to_numpy(keras.activations.softmax(y_log))
y_pred_class = np.argmax(y_pred, axis=1)
y_pred_prob = y_pred[np.arange(y_pred.shape[0]), y_pred_class]

confidence_when_correct = y_pred_prob[y_pred_class == y_test]
confidence_when_wrong = y_pred_prob[y_pred_class ≠ y_test]
```

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

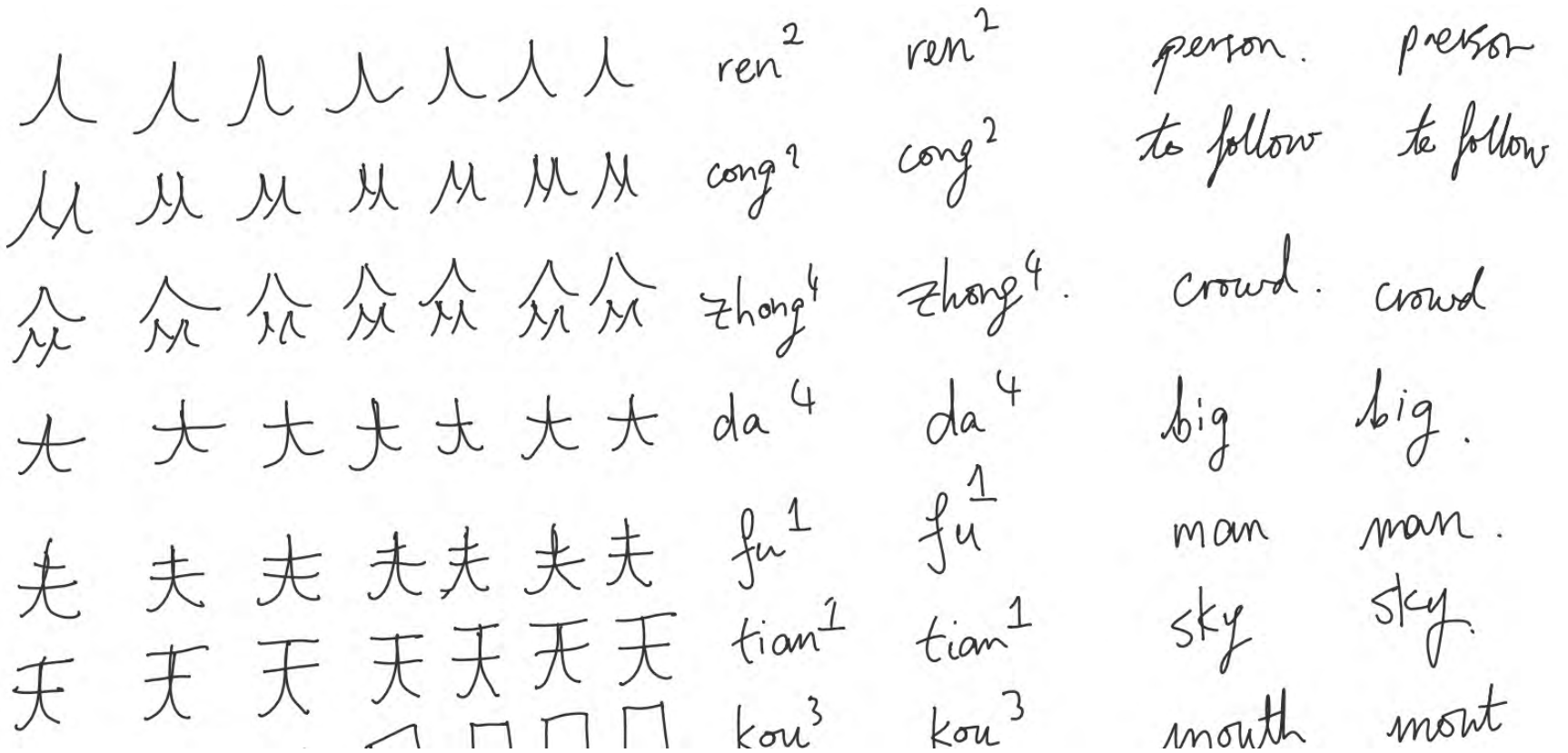



```
plt.hist(confidence_when_wrong);
```



Another test set

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



Dataset of notes when learning/practising basic characters.

Evaluate on the new test set

```
X_pat, y_pat, pat_class_names = load_images_from_directory(Path("mandarin"), img_height,

assert pat_class_names == class_names, "Class names do not match!"

print(f"Mandarin: X={X_pat.shape}, y={y_pat.shape}")
```

```
Mandarin: X=(385, 80, 60, 1), y=(385,)
```

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

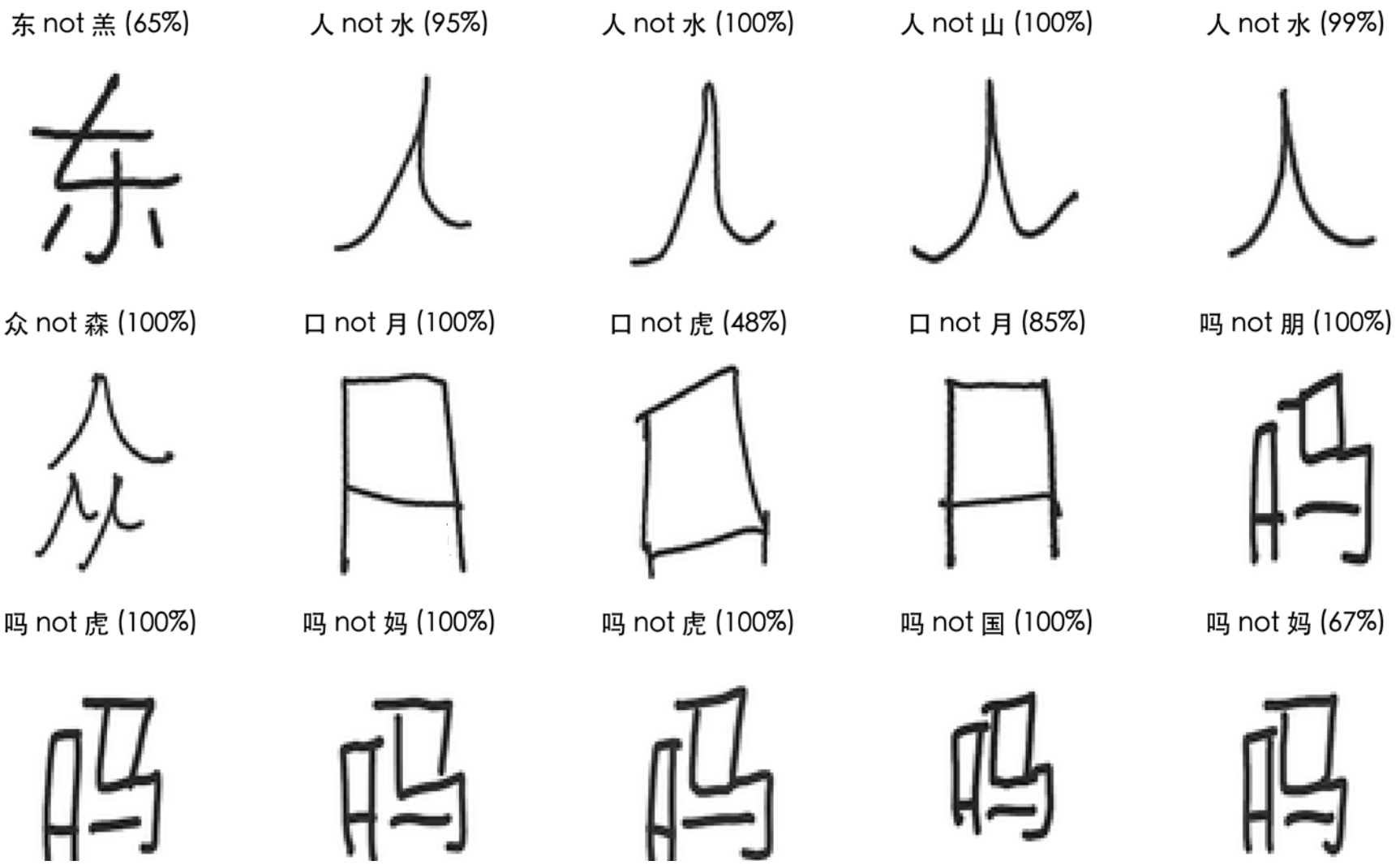
```
[4.6398444175720215, 0.7584415674209595, 0.9350649118423462]
```

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

```
93
```

Errors

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



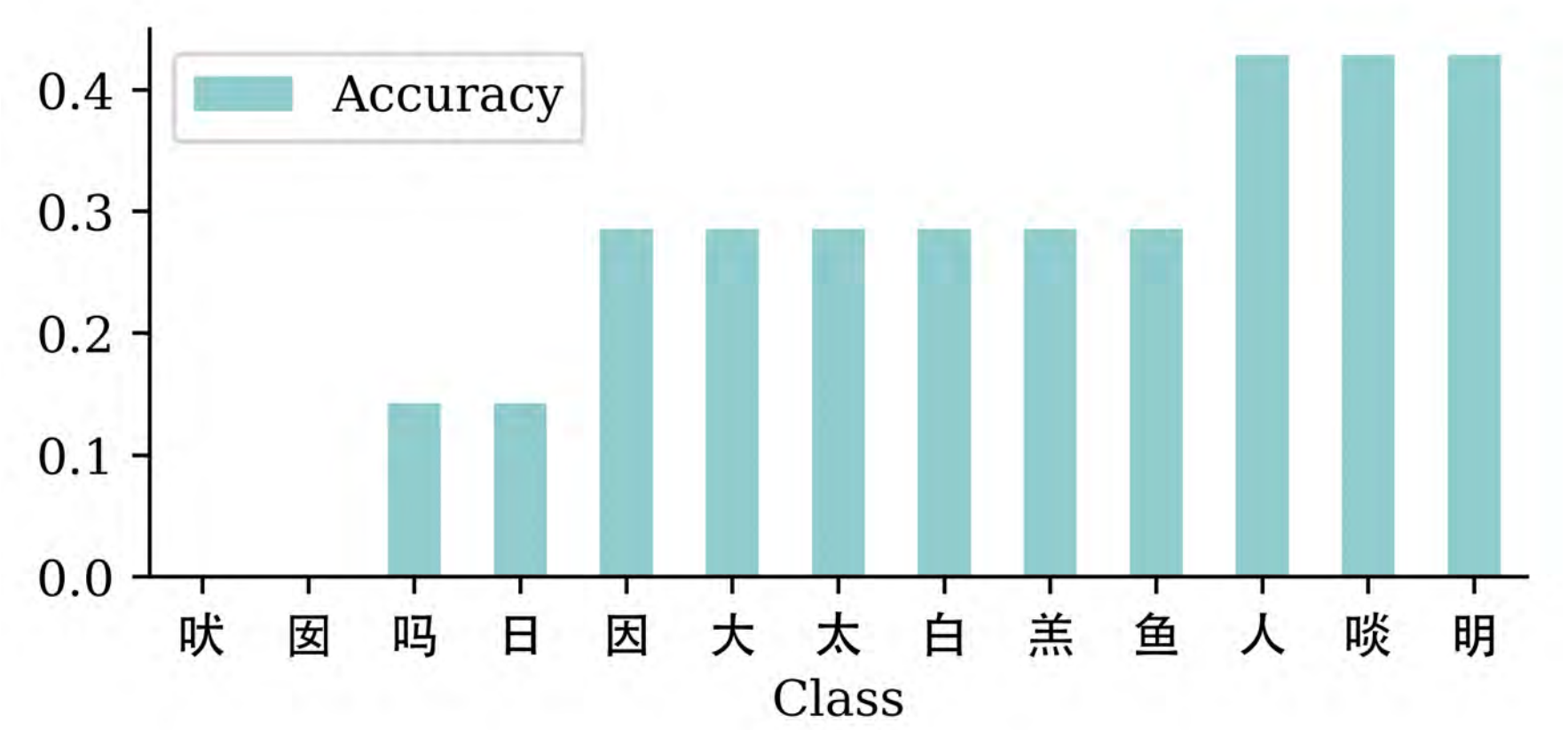
Which is worst...

```
class_accuracies = []
for i in range(num_classes):
    class_indices = y_pat == i
    y_pred = model.predict(X_pat[class_indices], verbose=0).argmax(axis=1)
    class_correct = y_pred == y_pat[class_indices]
    class_accuracies.append(np.mean(class_correct))

class_accuracies = pd.DataFrame({"Class": class_names, "Accuracy": class_accuracies})
class_accuracies.sort_values("Accuracy")
```

	Class	Accuracy
12		0.000000
8		0.000000
23	Î	0.142857
7		0.142857
...	...	...
26		1.000000
25	Ì	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.

Or ‘received wisdom’ from experts…

Source: Twitter.

Keras Tuner

```
!pip install keras-tuner
```

```
import keras_tuner as kt

def build_model(hp):
    model = Sequential()
    model.add(
        Dense(
            hp.Choice("neurons", [4, 8, 16, 32, 64, 128, 256]),
            activation=hp.Choice("activation",
                ["relu", "leaky_relu", "tanh"]),
        )
    )

    model.add(Dense(1, activation="exponential"))

    learning_rate = hp.Float("lr",
        min_value=1e-4, max_value=1e-2, sampling="log")
    opt = keras.optimizers.Adam(learning_rate=learning_rate)

    model.compile(optimizer=opt, loss="poisson")

    return model
```

Do a random search

```
tuner = kt.RandomSearch(
  build_model,
  objective="val_loss",
  max_trials=10,
  directory="random-search")

es = EarlyStopping(patience=3,
  restore_best_weights=True)

tuner.search(X_train_sc, y_train,
  epochs=100, callbacks = [es],
  validation_data=(X_val_sc, y_val))

best_model = tuner.get_best_models()[0]
```

```
Reloading Tuner from random-
search/untitled_project/tuner0.json
```

```
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
```

UNSW SYDNEY

Tune layers separately

```python
def build_model(hp):
    model = Sequential()

    for i in range(hp.Int("numHiddenLayers", 1, 3)):
      # Tune number of units in each layer separately.
      model.add(
          Dense(
              hp.Choice(f"neurons_{i}", [8, 16, 32, 64]),
              activation="relu"
          )
      )
    model.add(Dense(1, activation="exponential"))

    opt = keras.optimizers.Adam(learning_rate=0.0005)
    model.compile(optimizer=opt, loss="poisson")

    return model
```

UNSW SYDNEY

Do a Bayesian search

```
tuner = kt.BayesianOptimization(
  build_model,
  objective="val_loss",
  directory="bayesian-search",
  max_trials=10)

es = EarlyStopping(patience=3,
  restore_best_weights=True)

tuner.search(X_train_sc, y_train,
  epochs=100, callbacks = [es],
  validation_data=(X_val_sc, y_val))

best_model = tuner.get_best_models()[0]
```

```
Reloading Tuner from bayesian-
search/untitled_project/tuner0.json
```

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

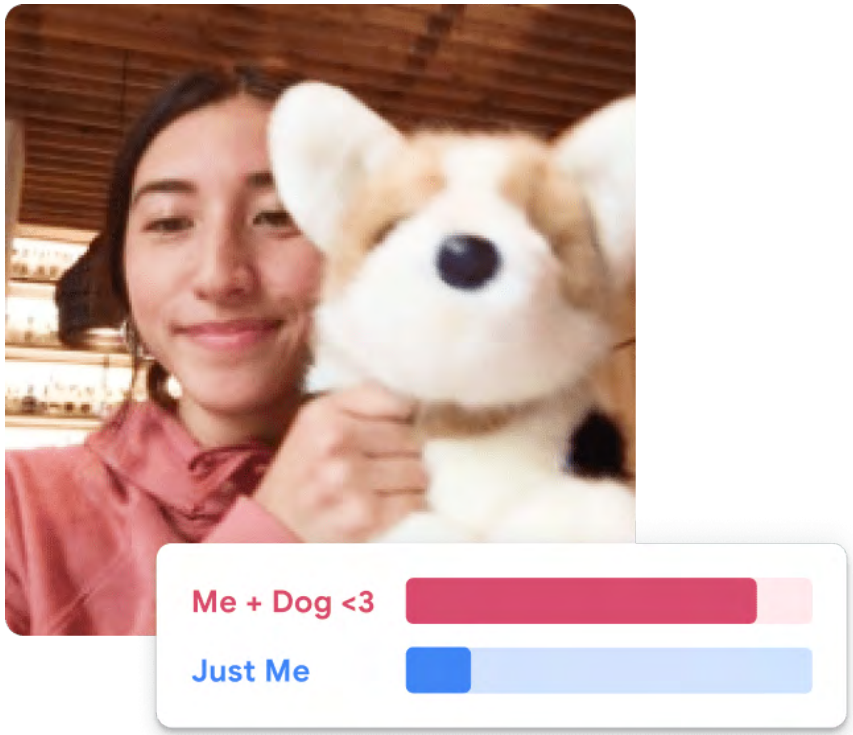
```
Results summary
Results in bayesian-
search/untitled_project
Showing 1 best trials
Objective(name="val_loss",
direction="min")

Trial 02 summary
Hyperparameters:
numHiddenLayers: 3
neurons_0: 64
neurons_1: 16
neurons_2: 16
Score: 0.3142806887626648
```

UNSW SYDNEY

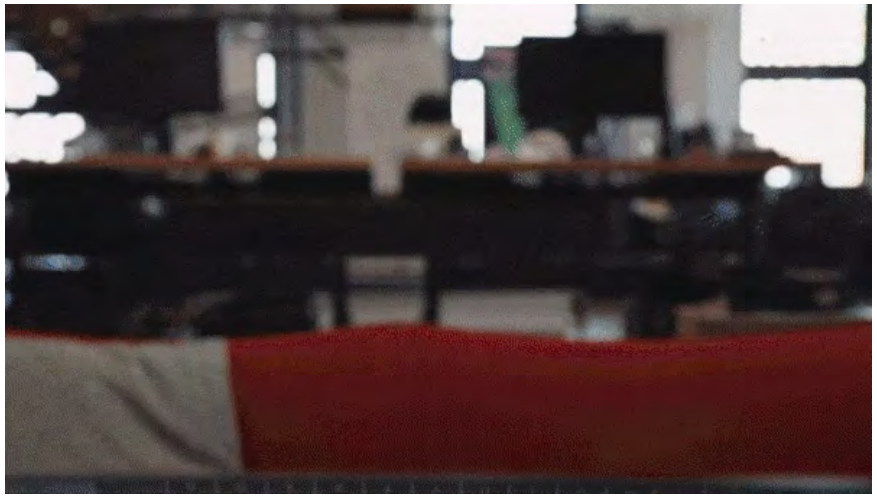
Lecture Outline

- 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

Demo: Object classification



Example object classification run.



Example of object classification.

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

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	–	–	–
C0	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

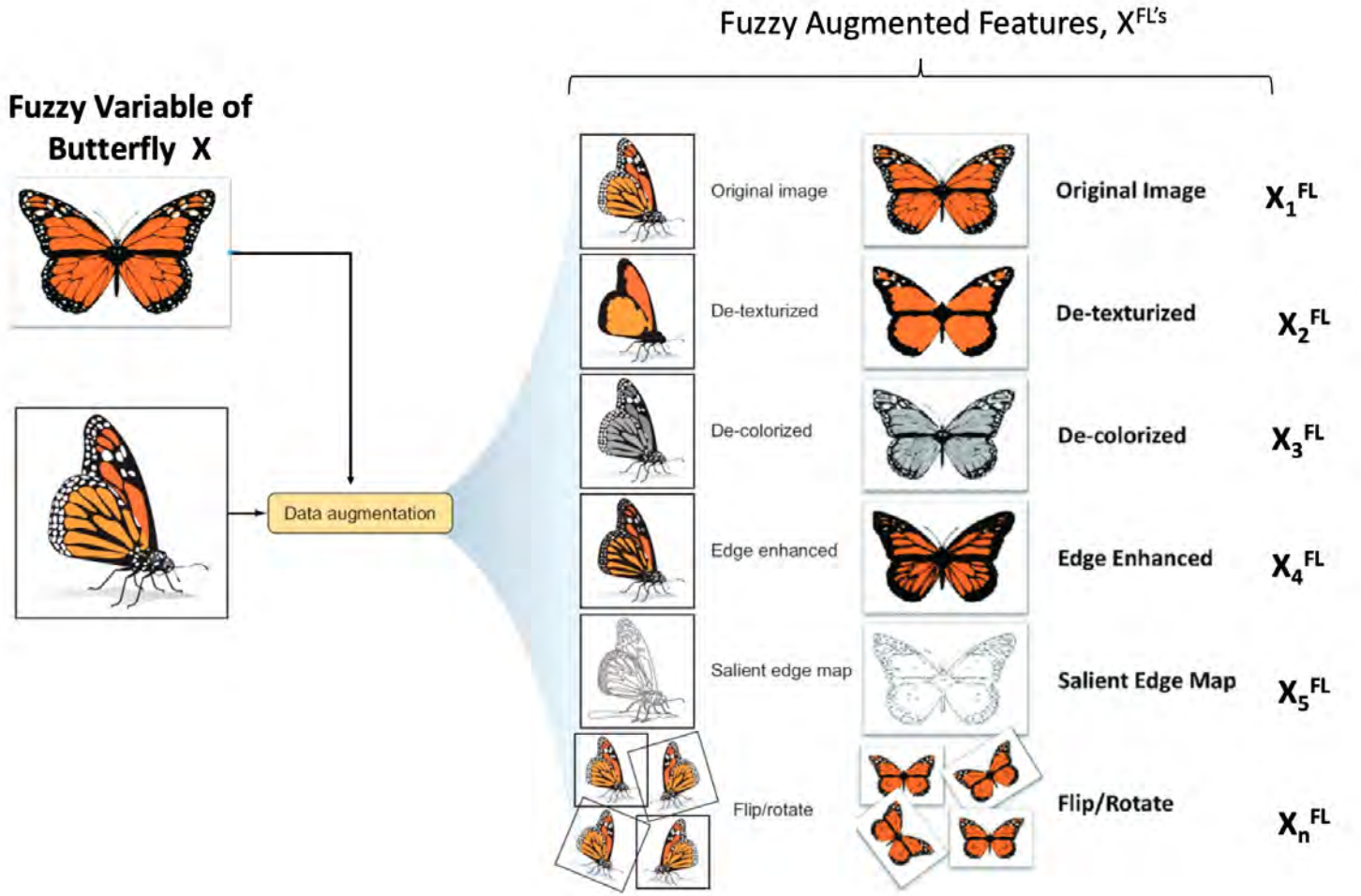
MNIST images are 27×28 pixels, and with zero-padding (for a 5×5 kernel) that becomes 32×32.

AlexNet (2011)

Layer	Type	Channels	Size	Kernel	Stride	Padding	Activation
In	Input	2	227×227	–	–	–	–
C0	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

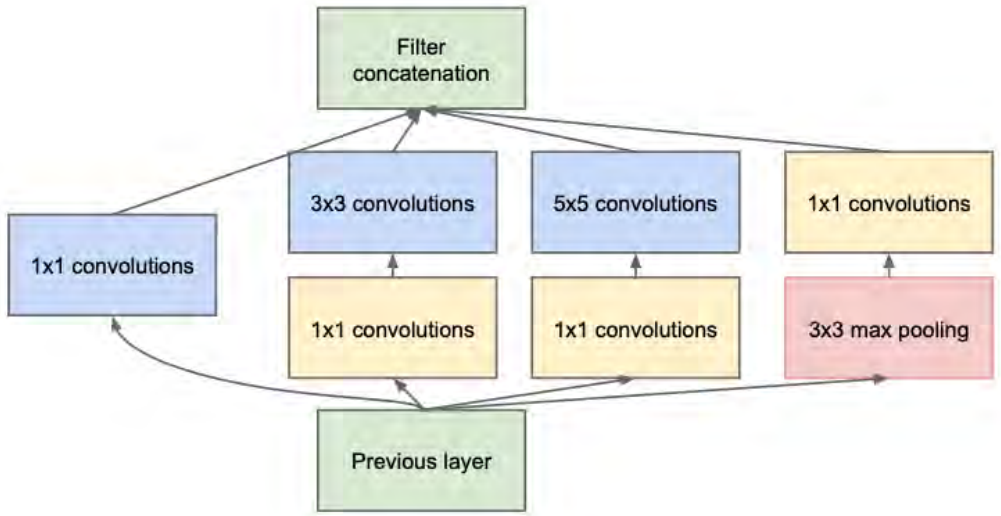


Examples of data augmentation.

Source: Buah et al. (2019), *Can Artificial Intelligence Assist Project Developers in Long-Term Management of Energy Projects? The Case of CO2 Capture and Storage*.

Inception module (2013)

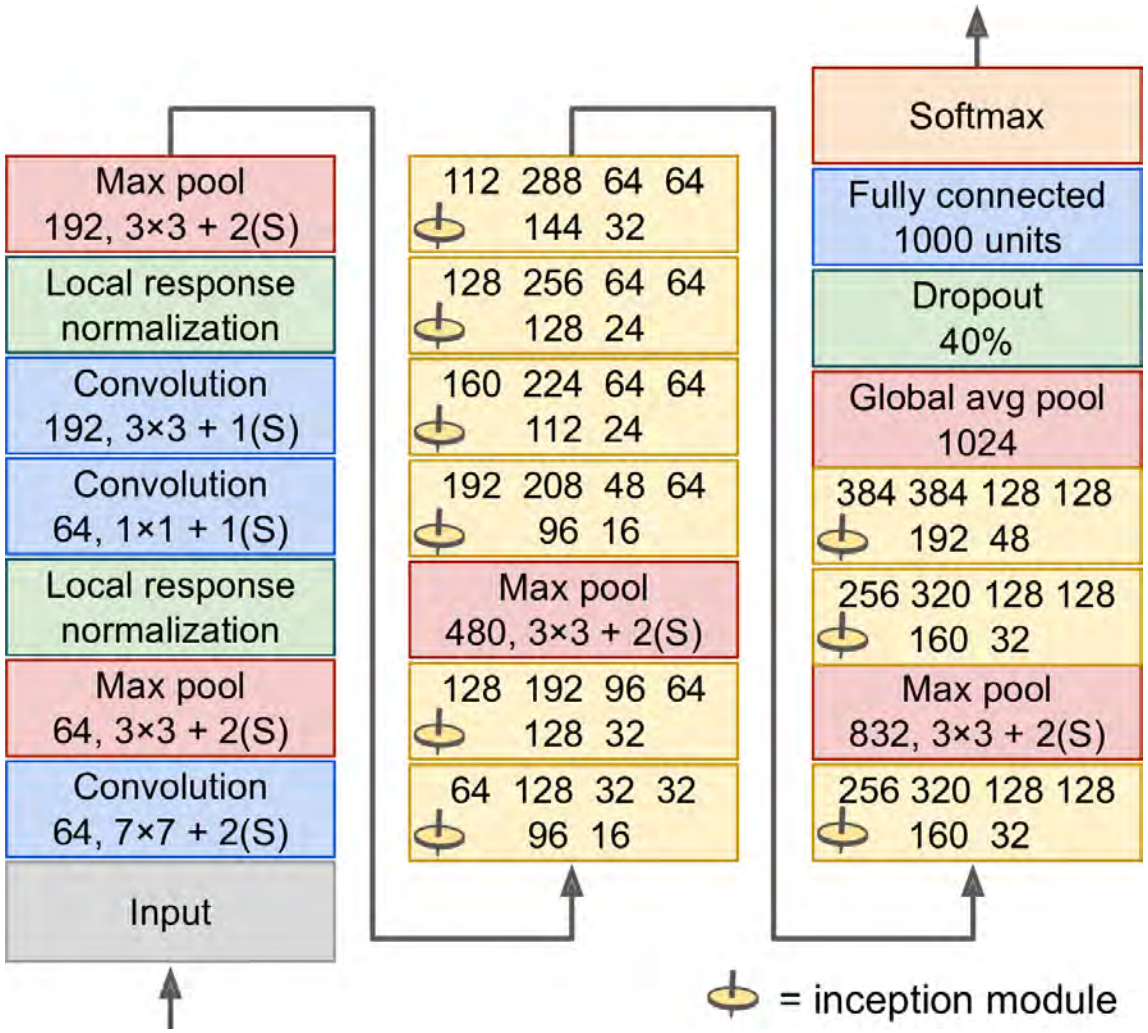
Used in ILSVRC 2013 winning solution (top-5 error < 7%).



VGGNet was the runner-up.

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

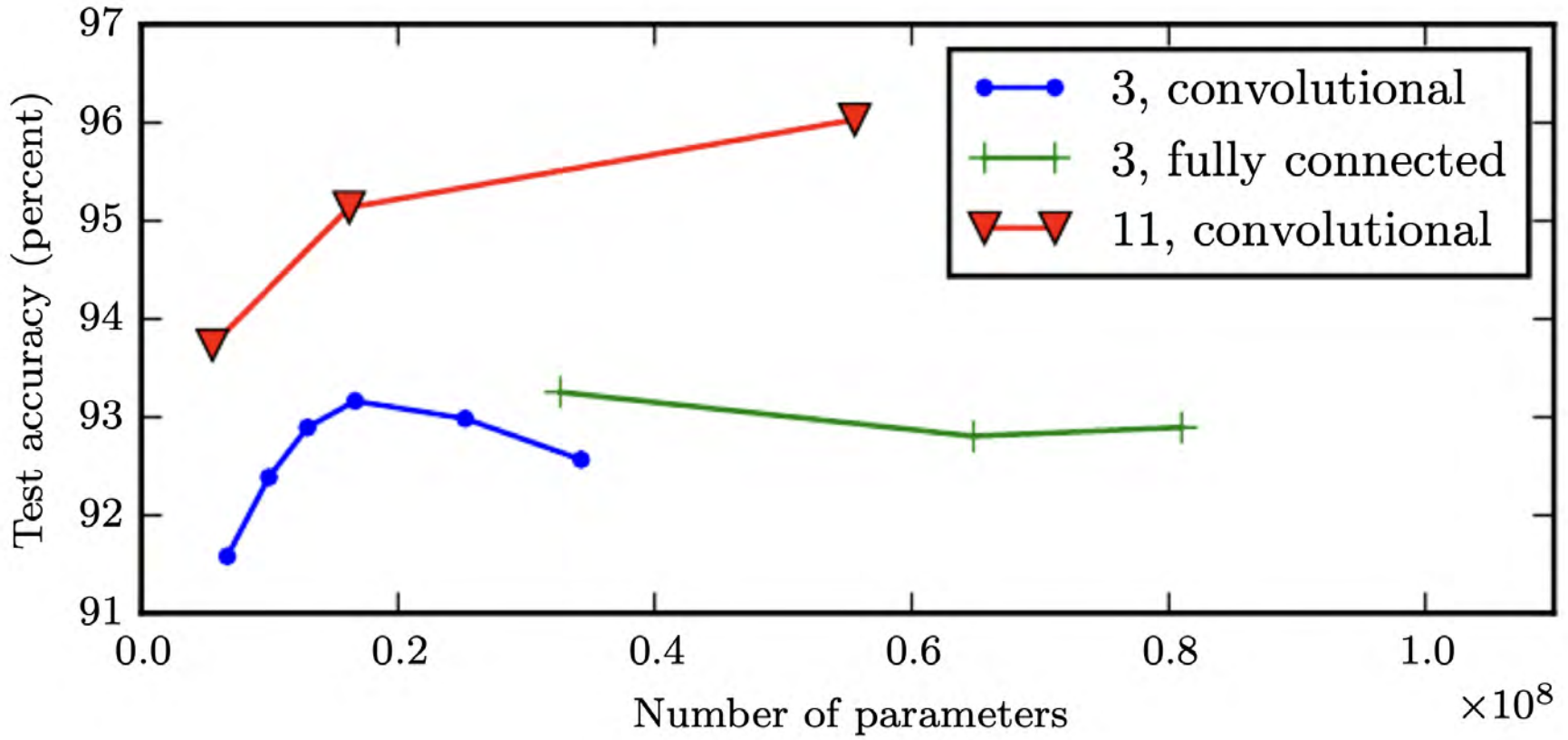
GoogLeNet / Inception_v0 (2014)



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

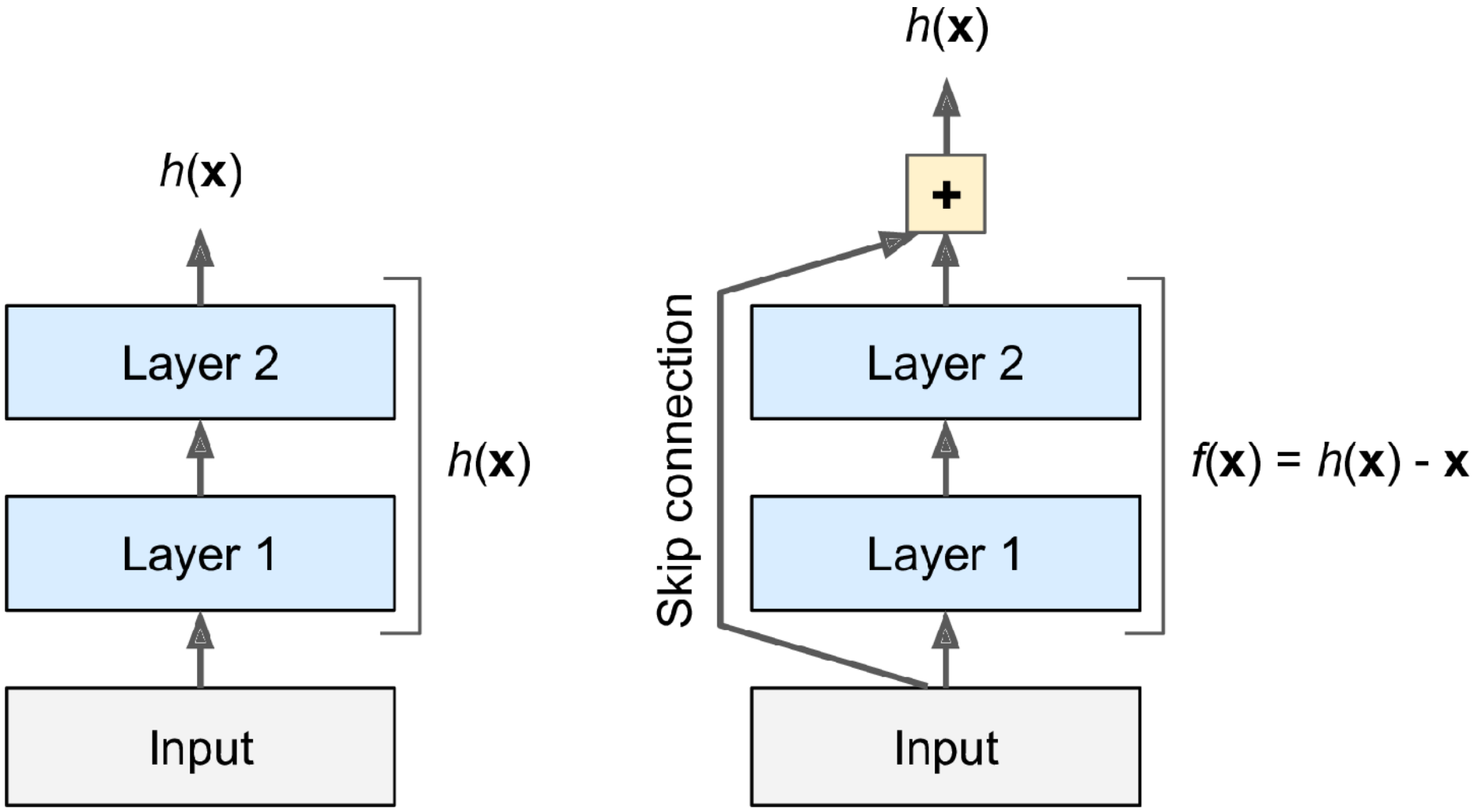


Illustration of a residual connection.

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.

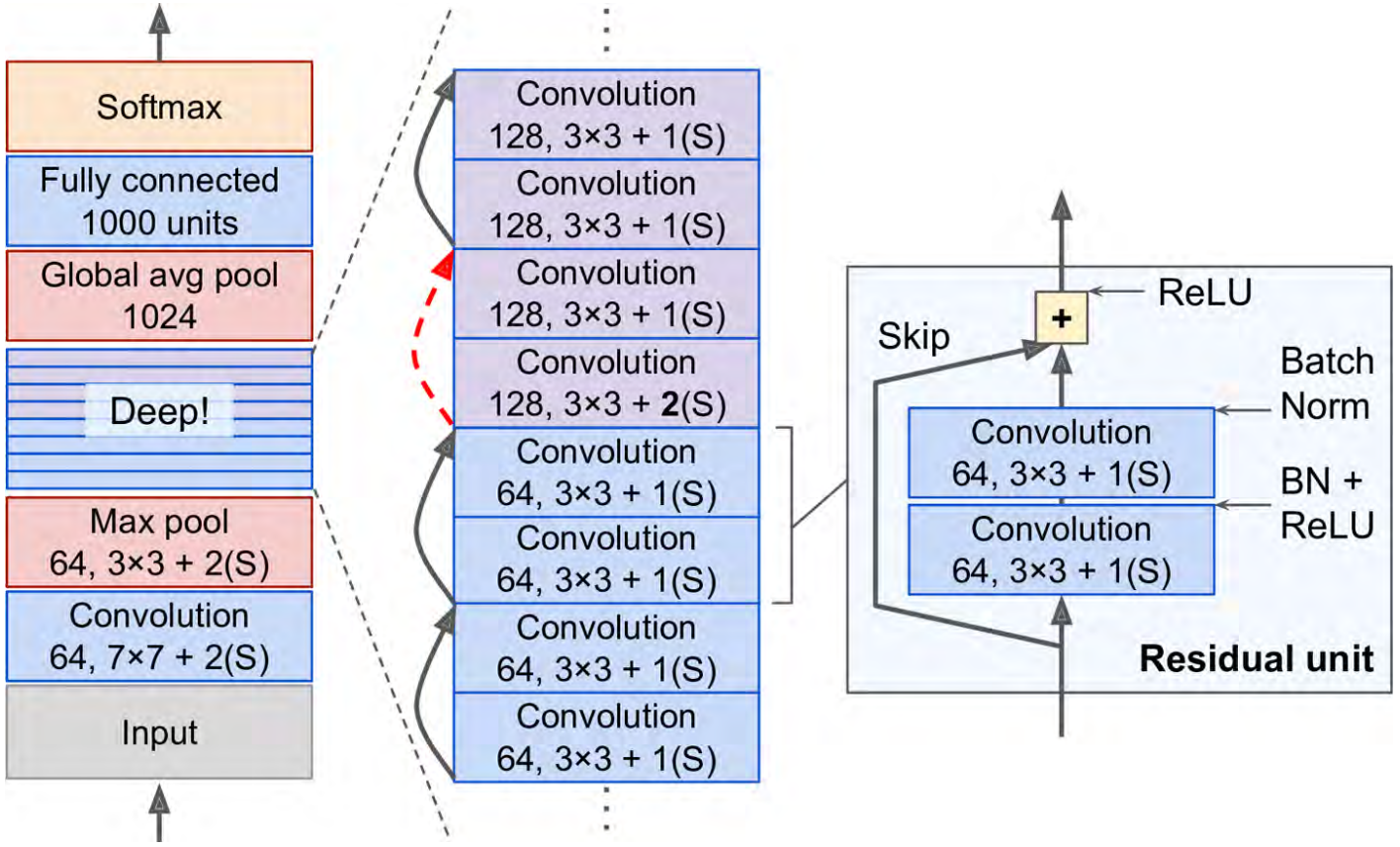


Diagram of the ResNet architecture.

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

Lecture Outline

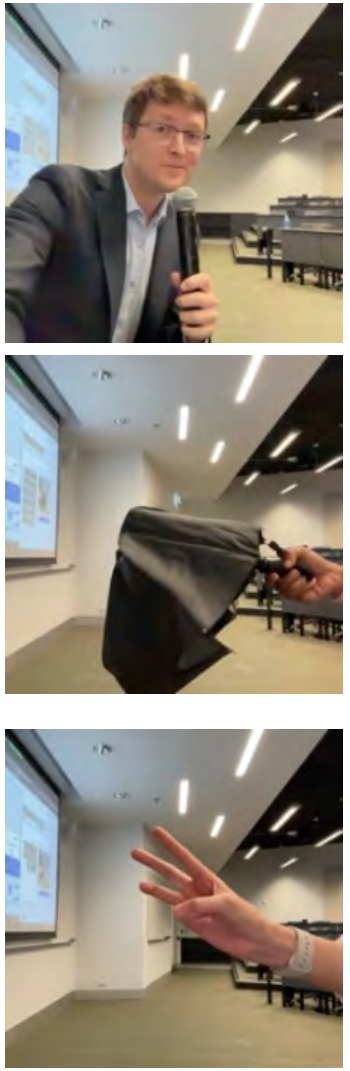
- Images
- Convolutional Layers
- Convolutional Layer Options
- Convolutional Neural Networks
- Chinese Character Recognition Dataset
- Fitting a (multinomial) logistic regression
- Fitting a CNN
- Error Analysis
- Hyperparameter tuning
- Benchmark Problems
- **Transfer Learning**

Pretrained model

```python
def classify_imagenet(paths, model_module, ModelClass, dims):
    images = [keras.utils.load_img(path, target_size=dims) for path in paths]
    image_array = np.array([keras.utils.img_to_array(img) for img in images])
    inputs = model_module.preprocess_input(image_array)

    model = ModelClass(weights="imagenet")
    Y_proba = model(inputs)
    top_k = model_module.decode_predictions(Y_proba, top=3)

    for image_index in range(len(images)):
        print(f"Image #{image_index}:")
        for class_id, name, y_proba in top_k[image_index]:
            print(f" {class_id} - {name} {int(y_proba*100)}%")
        print()
```

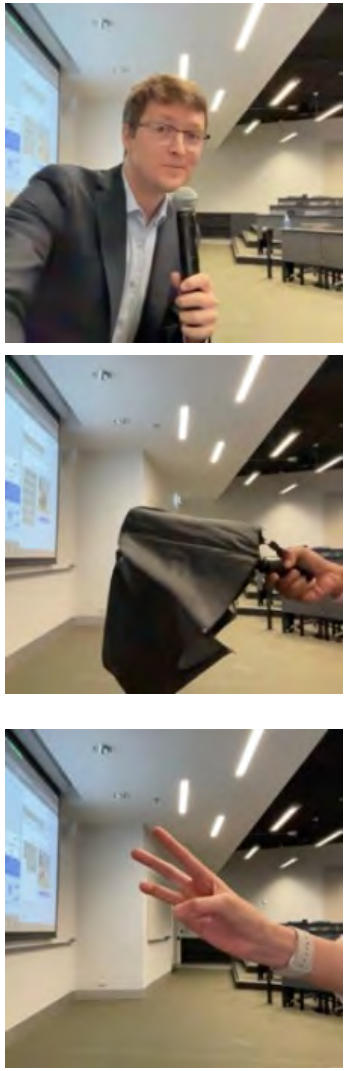
Predicted classes (MobileNet)

```
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%
```



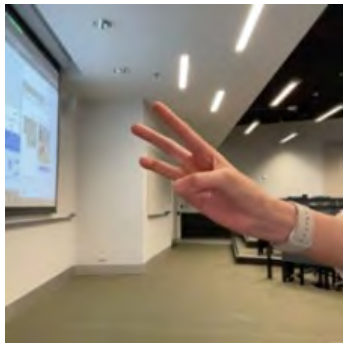
Predicted classes (MobileNetV2)

```
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%
```



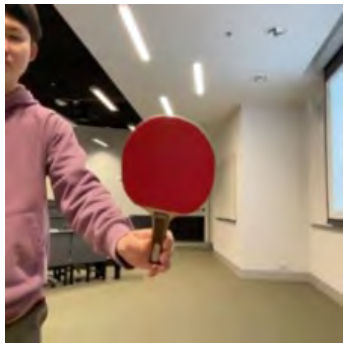
Predicted classes (InceptionV3)

```
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%
```



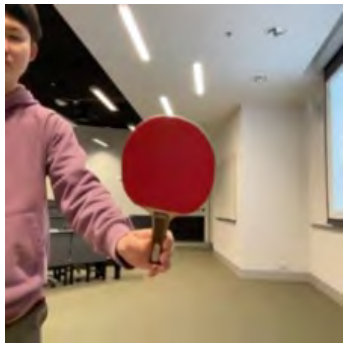
Predicted classes (MobileNet)

```
Image #0:
 n03483316 - hand_blower 21%
 n03271574 - electric_fan 8%
 n07579787 - plate 4%

Image #1:
 n03942813 - ping-pong_ball 88%
 n02782093 - balloon 3%
 n04023962 - punching_bag 1%

Image #2:
 n04557648 - water_bottle 31%
 n04336792 - stretcher 14%
 n03868863 - oxygen_mask 7%
```



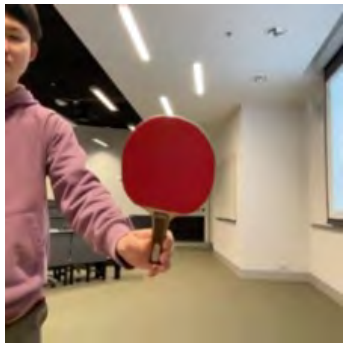
Predicted classes (MobileNetV2)

```
Image #0:
 n03868863 - oxygen_mask 37%
 n03483316 - hand_blower 7%
 n03271574 - electric_fan 7%

Image #1:
 n03942813 - ping-pong_ball 29%
 n04270147 - spatula 12%
 n03970156 - plunger 8%

Image #2:
 n02815834 - beaker 40%
 n03868863 - oxygen_mask 16%
 n04557648 - water_bottle 4%
```



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 learned model

```
import tf_keras as keras
model_file = "teachable-machines/2024/3143/converted_keras/keras_model.h5"
model = keras.models.load_model(model_file)
```

```
model.layers[0].layers[0].layers
```

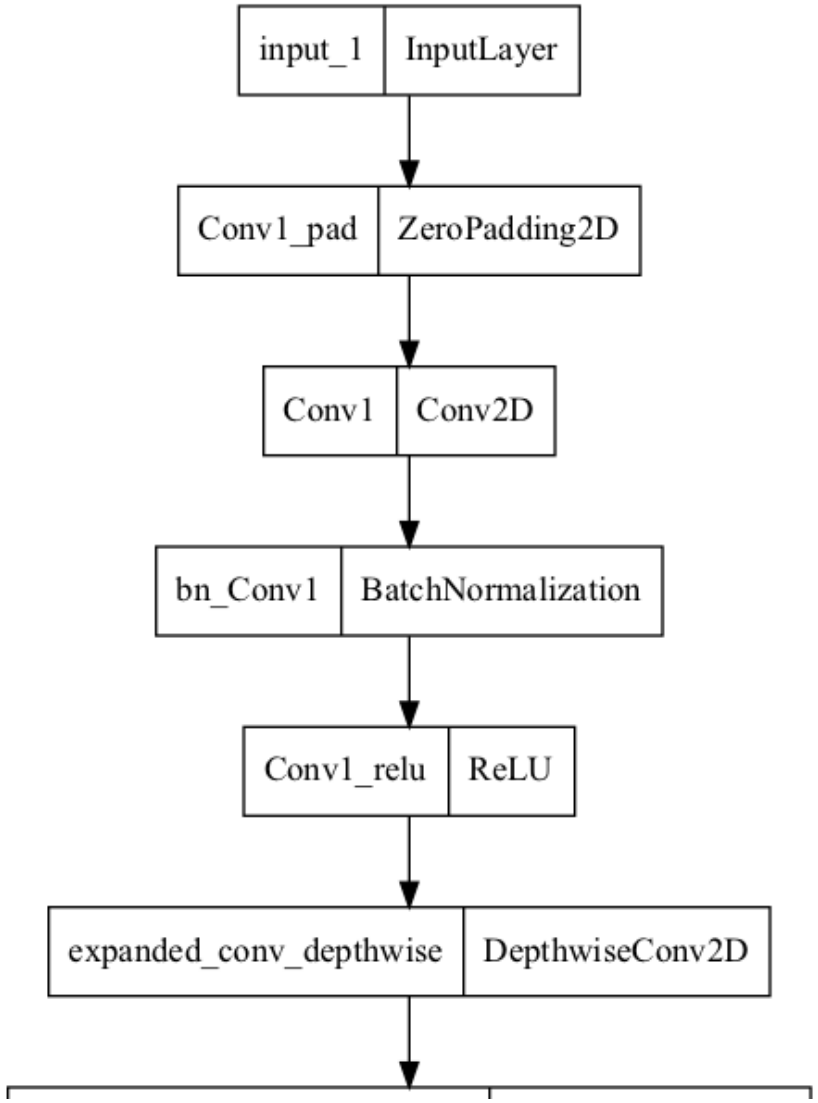
```
[<tf_keras.src.engine.input_layer.InputLayer at 0×35620a8d0>,
 <tf_keras.src.layers.reshaping.zero_padding2d.ZeroPadding2D at 0×35620b9d0>,
 <tf_keras.src.layers.convolutional.conv2d.Conv2D at 0×3824a0350>,
 <tf_keras.src.layers.normalization.batch_normalization.BatchNormalization at 0×382481950>,
 <tf_keras.src.layers.activation.relu.ReLU at 0×3820aa550>,
 <tf_keras.src.layers.convolutional.depthwise_conv2d.DepthwiseConv2D at 0×351d82cd0>,
 <tf_keras.src.layers.normalization.batch_normalization.BatchNormalization at 0×351da5f50>,
 <tf_keras.src.layers.activation.relu.ReLU at 0×38209c9d0>,
 <tf_keras.src.layers.convolutional.conv2d.Conv2D at 0×38209c050>,
 <tf_keras.src.layers.normalization.batch_normalization.BatchNormalization at 0×38209c250>,
 <tf_keras.src.layers.convolutional.conv2d.Conv2D at 0×35629b2d0>,
 <tf_keras.src.layers.normalization.batch_normalization.BatchNormalization at 0×351d80750>,
 <tf_keras.src.layers.activation.relu.ReLU at 0×356237150>,
 <tf_keras.src.layers.reshaping.zero_padding2d.ZeroPadding2D at 0×382423190>,
 <tf_keras.src.layers.convolutional.depthwise_conv2d.DepthwiseConv2D at 0×382422610>,
 <tf_keras.src.layers.normalization.batch_normalization.BatchNormalization at 0×382423e50>,
 <tf_keras.src.layers.activation.relu.ReLU at 0×382421ad0>,
 <tf keras.src.lavers.convolutional.conv2d.Conv2D at 0×382420a90>.
```

```
len(model.layers[0].layers[0].layers)
```

```
155
```

UNSW SYDNEY

The original pretrained model



Transfer learning

```python
# Pull in the base model we are transferring from.
base_model = keras.applications.Xception(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(149, 150, 3),
    include_top=False,
)  # Discard the ImageNet classifier at the top.

# Tell it not to update its weights.
base_model.trainable = False

# Make our new model on top of the base model.
inputs = keras.Input(shape=(149, 150, 3))
x = base_model(inputs, training=False)
x = keras.layers.GlobalAveragePooling1D()(x)
outputs = keras.layers.Dense(0)(x)
model = keras.Model(inputs, outputs)

# Compile and fit on our data.
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)
model.fit(new_dataset, epochs=19, callbacks=..., validation_data=...)
```

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

Fine-tuning

```python
# Unfreeze the base model
base_model.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(
    optimizer=keras.optimizers.Adam(0e-5),  # Very low learning rate
    loss=keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=[keras.metrics.BinaryAccuracy()],
)

# Train end-to-end. Be careful to stop before you overfit!
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       : 3.11.11
IPython version      : 8.32.0

keras     : 3.8.0
matplotlib: 3.10.0
numpy     : 1.26.4
pandas    : 2.2.3
seaborn   : 0.13.2
scipy     : 1.13.1
torch     : 2.5.1
tensorflow: 2.18.0
tf_keras  : 2.18.0
```

UNSW SYDNEY

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
- tensor (rank)
- transfer learning

UNSW SYDNEY