Image Processing

Daniel Hardt

CBS

Copenhagen
Business School
HANDELSHØJSKOLEN

Outline

```
# this only needs to be done once!
#!pip install transformers

##############################################################################################
# bertIntro.py
# this program takes 3 optional args:
#  inputfile (default is https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/train.tsv')
#  Features: default BoW, or BERT embeddings
#  Size: number of lines of input file
#  Program assumes input file contains text in first column and labels in second column
#
#  Sample invocations:
#  python3 bertIntro.py  -s 500 -f "BERT"
#  (first 500 lines of default input file, using BERT features)
#
#  python3 bertIntro.py -i eng.tsv -s 5000
#  (first 5000 lines of eng.tsv file, using default BoW features)
#
##############################################################################################
```

```
def bowFeatures(texts):
    print("BoW model...")
    cv = CountVectorizer()
    X_bow = cv.fit_transform(texts)
    return(X_bow)
```

```
def BertFeatures(texts):
    print("loading BERT...")
    # For DistilBERT:
    model_class, tokenizer_class, pretrained_weights = (ppb.DistilBertModel, ppb.DistilBertTokenizer, 'distilbert-base-uncased')

    ## Want BERT instead of distilBERT? Uncomment the following line:
#    model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer, 'bert-base-uncased', use_cached_eval_features=True)

    # Load pretrained model/tokenizer
    tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
    model = model_class.from_pretrained(pretrained_weights)

    tokenized = texts.apply((lambda x: tokenizer.encode(x, add_special_tokens=True)))

    max_len = 0
    for i in tokenized.values:
        if len(i) > max_len:
            max_len = len(i)

    padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized.values])
    np.array(padded).shape

    attention_mask = np.where(padded != 0, 1, 0)
    attention_mask.shape

    input_ids = torch.tensor(padded)
    attention_mask = torch.tensor(attention_mask)

    print("Getting model encodings...")
    with torch.no_grad():
        last_hidden_states = model(input_ids, attention_mask=attention_mask)


    features = last_hidden_states[0][:,0,:].numpy()
    return features
```

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("-i", "--input_file_path", default='https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/train.tsv')
    parser.add_argument("-f", "--features", default="BOW")
    parser.add_argument("-s", "--size", default=1000, type=int)
    main(parser.parse_args())
```

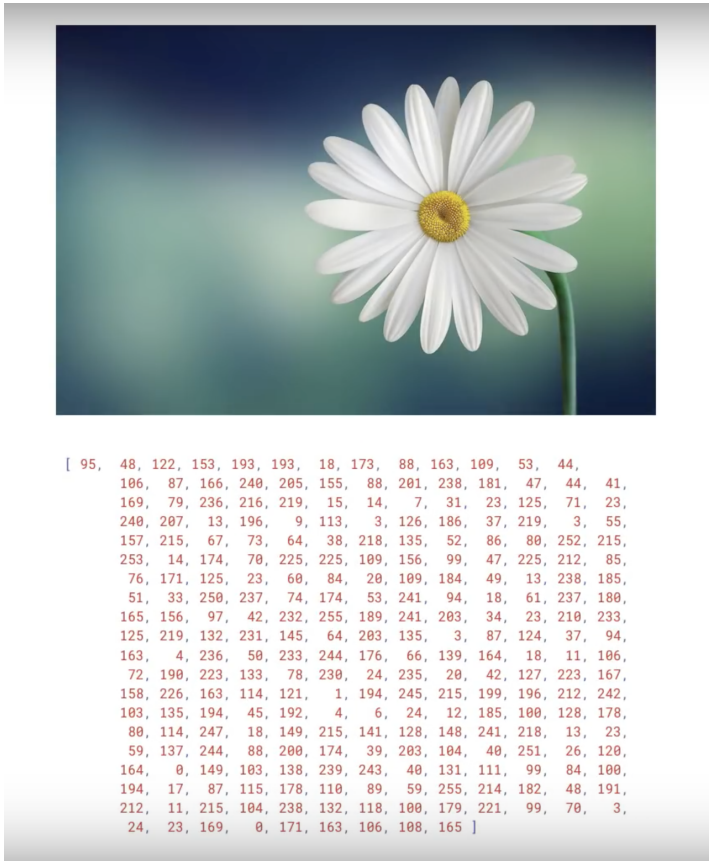
```
def main(args):
    print("input_file_path", args.input_file_path)
    print("Features", args.features)
    print("Size", args.size)

    INPUT_FILE_PATH= args.input_file_path
    FEATURES= args.features
    SIZE= args.size


    # Can use BoW or BERT features
    #ModelFeatures = ["BOW"]
    #ModelFeatures = ["BERT"]

    print("Reading file...")

    #  Can use eng.tsv file or SST2 (sentiment file)

    #df = pd.read_csv('https://github.com/clairett/pytorch-sentiment-classification/raw/master/data/SST2/train.tsv', delimiter='\t', header=None)

    #df = pd.read_csv("eng.csv",header=None)
    df = pd.read_csv(INPUT_FILE_PATH,delimiter='\t', header=None)

    # this determines size of dataset
    batch_1 = df[:SIZE]

    print("value counts",batch_1[1].value_counts())


    if FEATURES == "BERT":
        features = BertFeatures(batch_1[0])
    elif FEATURES == "BOW":
        features = bowFeatures(batch_1[0])
    else:
        features = bowFeatures(batch_1[0])  # just do BoW by default

    labels = batch_1[1]
```

```
train_features, test_features, train_labels, test_labels = train_test_split(features, labels)
print("LogisticRegression for ", FEATURES, "features...")
lr_clf = LogisticRegression()

lr_clf.fit(train_features, train_labels)

print("Train score",lr_clf.score(train_features, train_labels))
print("Test score",lr_clf.score(test_features, test_labels))

test_predictions = lr_clf.predict(test_features)

print(classification_report(test_labels, test_predictions))
```

Computer Vision

Computers gaining high-level understanding from digital images or videos

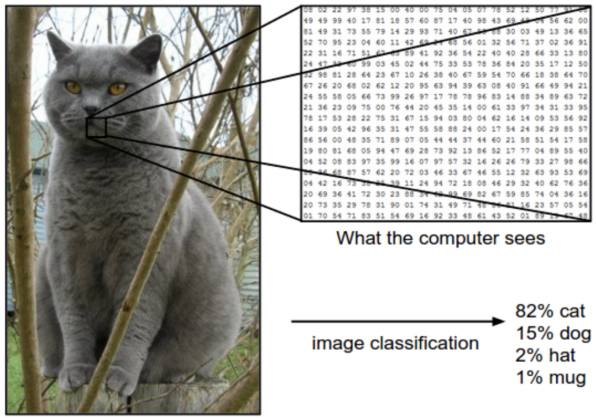
Images as Digital Arrays



```
[ 95,  48, 122, 153, 193, 193,  18, 173,  88, 163, 109,  53,  44,
       106,  87, 166, 240, 205, 155,  88, 201, 238, 181,  47,  44,  41,
       169,  79, 236, 216, 219,  15,  14,   7,  31,  23, 125,  71,  23,
       240, 207,  13, 196,   9, 113,   3, 126, 186,  37, 219,   3,  55,
       157, 215,  67,  73,  64,  38, 218, 135,  52,  86,  80, 252, 215,
       253,  14, 174,  70, 225, 225, 109, 156,  99,  47, 225, 212,  85,
        76, 171, 125,  23,  60,  84,  20, 109, 184,  49,  13, 238, 185,
        51,  33, 250, 237,  74, 174,  53, 241,  94,  18,  61, 237, 180,
       165, 156,  97,  42, 232, 255, 189, 241, 203,  34,  23, 210, 233,
       125, 219, 132, 231, 145,  64, 203, 135,   3,  87, 124,  37,  94,
       163,   4, 236,  50, 233, 244, 176,  66, 139, 164,  18,  11, 106,
        72, 190, 223, 133,  78, 230,  24, 235,  20,  42, 127, 223, 167,
       158, 226, 163, 114, 121,   1, 194, 245, 215, 199, 196, 212, 242,
       103, 135, 194,  45, 192,   4,   6,  24,  12, 185, 100, 128, 178,
        80, 114, 247,  18, 149, 215, 141, 128, 148, 241, 218,  13,  23,
        59, 137, 244,  88, 200, 174,  39, 203, 104,  40, 251,  26, 120,
       164,   0, 149, 103, 138, 239, 243,  40, 131, 111,  99,  84, 100,
       194,  17,  87, 115, 178, 110,  89,  59, 255, 214, 182,  48, 191,
       212,  11, 215, 104, 238, 132, 118, 100, 179, 221,  99,  70,   3,
        24,  23, 169,   0, 171, 163, 106, 108, 165 ]
```

Copenhagen
Business School
HANDELSHØJSKOLEN

Image Classification



(From: Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition)

Nearest-Neighbor for Images

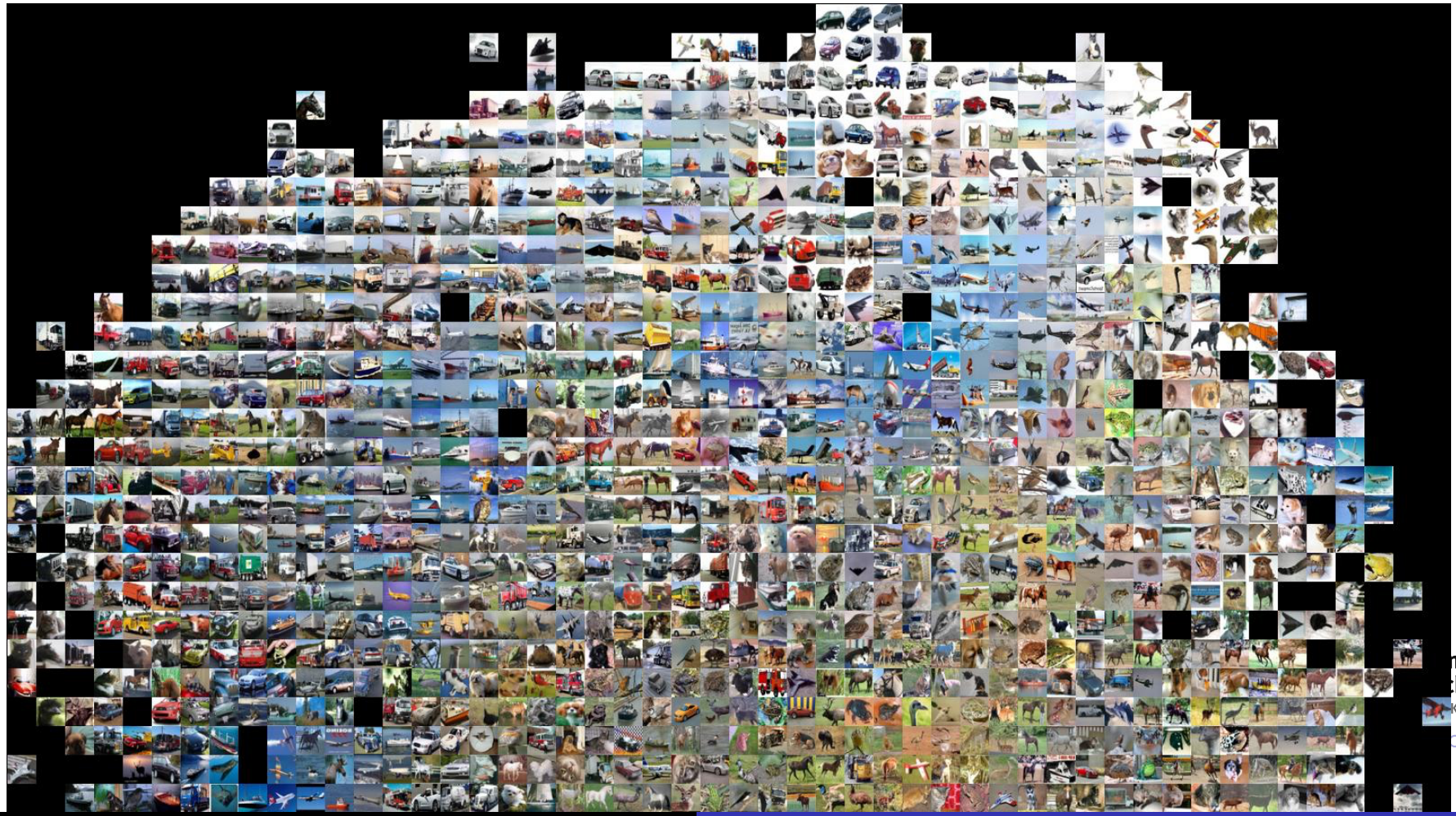
test image					training image					pixel-wise absolute value differences				
56	32	10	18	-	10	20	24	17	=	46	12	14	1	→ 456
90	23	128	133		8	10	89	100		82	13	39	33	
24	26	178	200		12	16	178	170		12	10	0	30	
2	0	255	220		4	32	233	112		2	32	22	108	

Nearest-Neighbor for Images

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

Take difference of each element of the pixel array.

kNN and Images



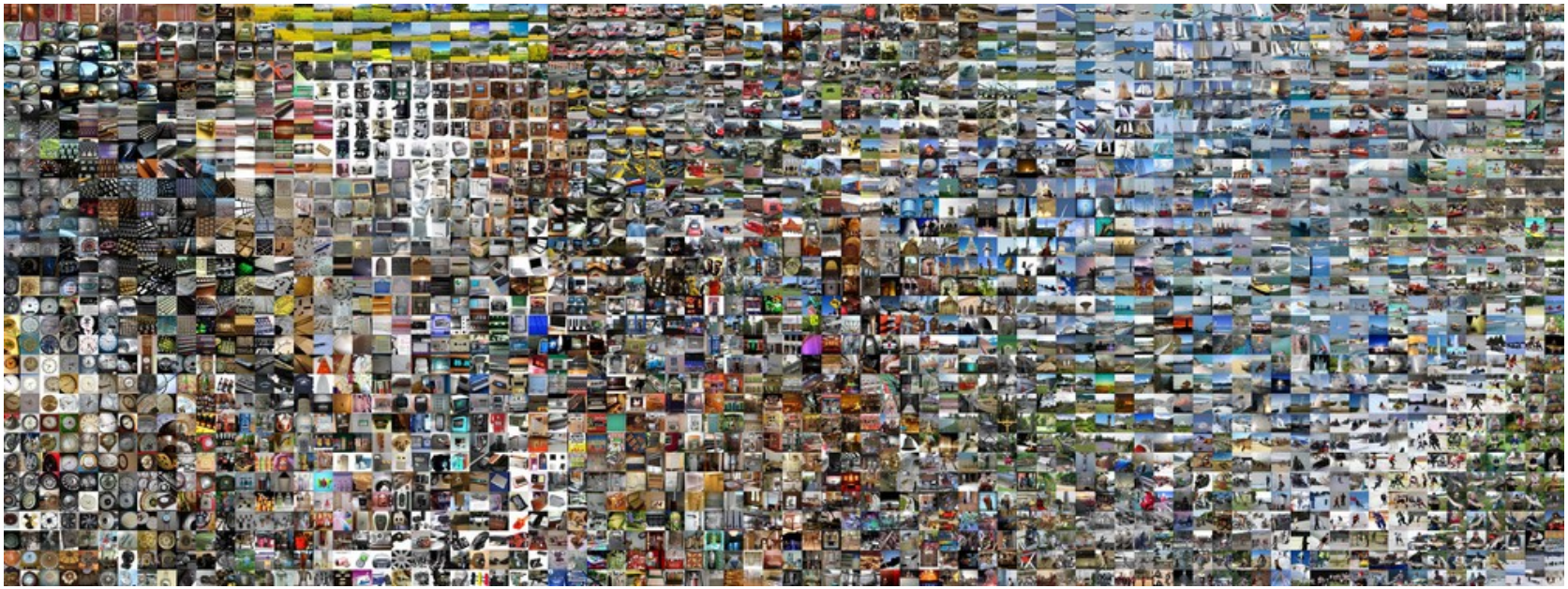
kNN and Images

kNN and Images

"Images that are nearby each other are much more a function of the general color distribution of the images, or the type of background rather than their semantic identity. For example, a dog can be seen very near a frog since both happen to be on white background."

(From: Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition)

CNN and Images

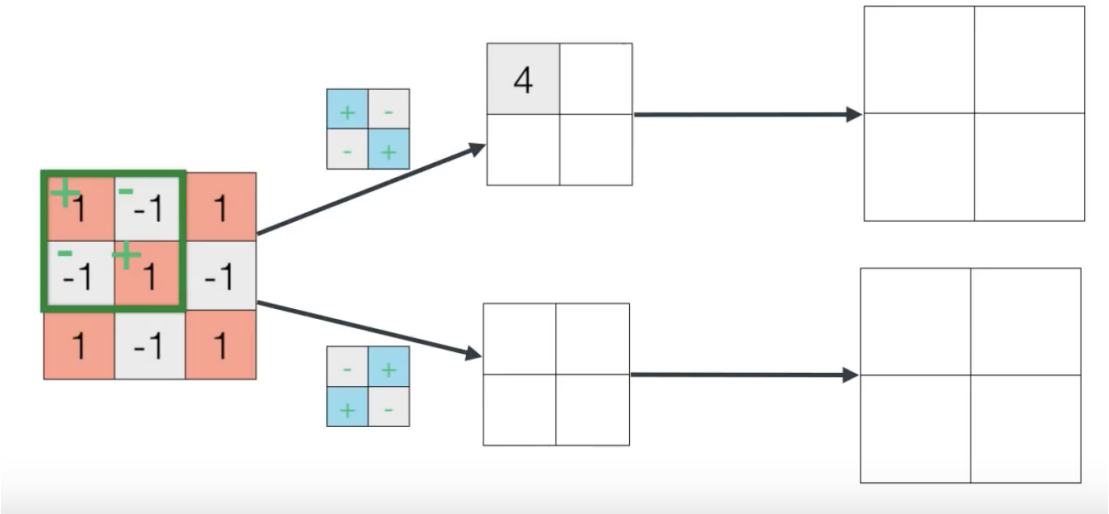


CNN and Images

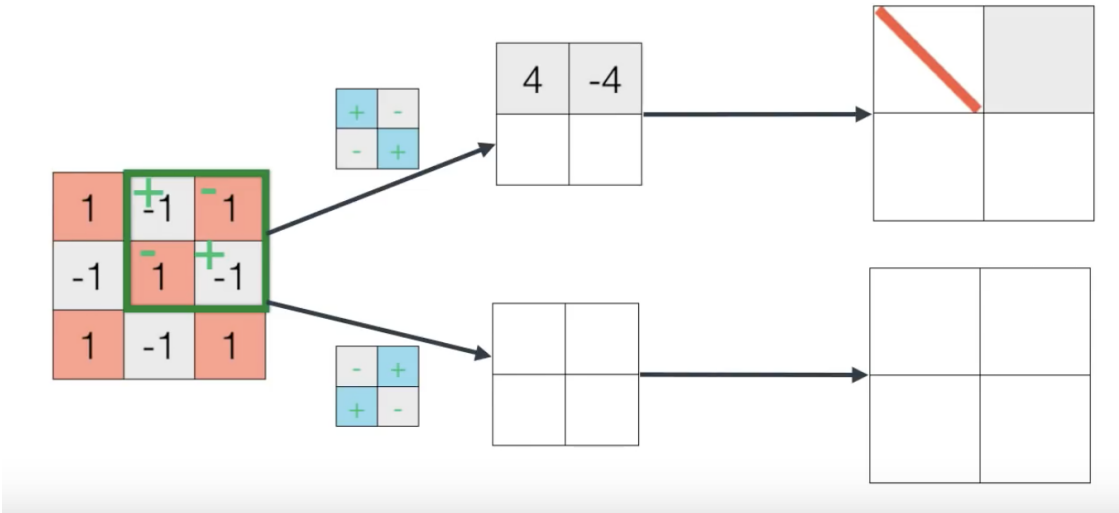
"Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based."

(From: Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition)

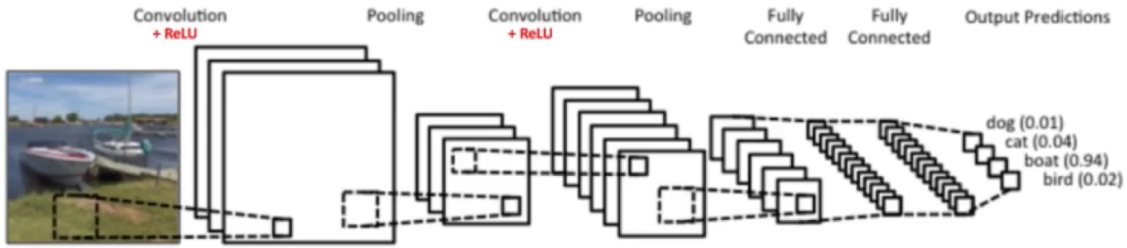
CNN – Filters



CNN – Filters



CNN – Image Recognition



Computer Vision Applications

- Automotive
  - Scene analysis, automated lane detection, automated road sign reading
- Media
  - Ebay: users can visually search for photos
- Health Care
  - Analyze medical images

ImageNet

IMAGENET

14,197,122 images, 21841 synsets indexed

Explore Download Challenges Publications CoolStuff About

Not logged in. Login | Signup

ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently we have an average of over five hundred images per node. We hope ImageNet will become a useful resource for researchers, educators, students and all of you who share our passion for pictures.
Click here to learn more about ImageNet, Click here to join the ImageNet mailing list.

ImageNet

- Over 14 million annotated images
- Over 20,000 categories
- Based on WordNet categories
- Crowdsourced annotation – in 2012, was world's largest academic user of Mechanical Turk
- Supports transfer learning for different image recognition tasks

WordNet

WordNet Search - 3.1

- WordNet home page - Glossary - Help

Word to search for: read Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

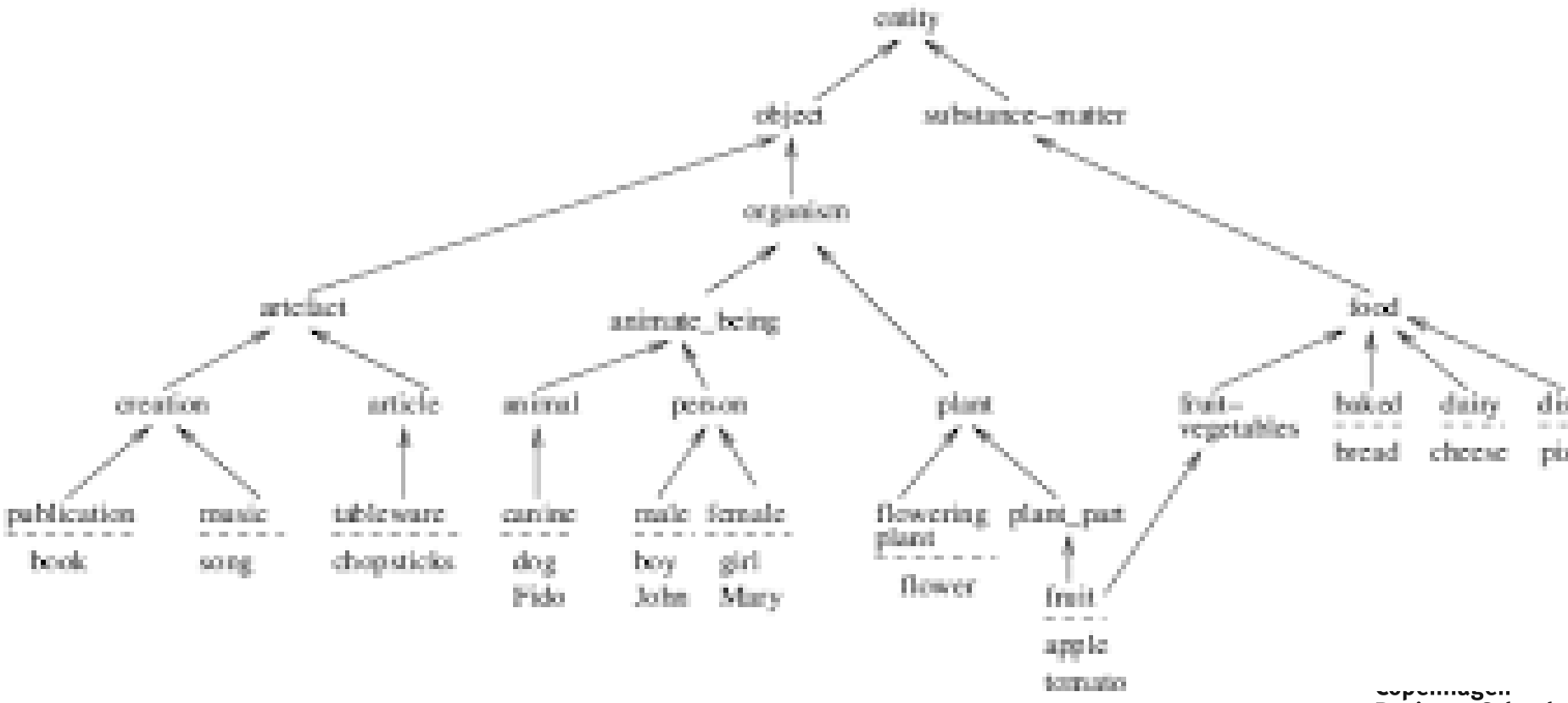
Noun

- S: (n) **read** (something that is read) *"the article was a very good read"*

Verb

- S: (v) **read** (interpret something that is written or printed) *"read the advertisement"; "Have you read Salman Rushdie?"*
- S: (v) **read**, say (have or contain a certain wording or form) *"The passage reads as follows"; "What does the law say?"*
- S: (v) **read** (look at, interpret, and say out loud something that is written or printed) *"The King will read the proclamation at noon"*
- S: (v) **read**, scan (obtain data from magnetic tapes or other digital sources) *"This dictionary can be read by the computer"*
- S: (v) **read** (interpret the significance of, as of palms, tea leaves, intestines, the sky; also of human behavior) *"She read the sky and predicted rain"; "I can't read his strange behavior"; "The fortune teller read his fate in the crystal ball"*
- S: (v) take, **read** (interpret something in a certain way; convey a particular meaning or impression) *"I read this address as a satire"; "How should I take this message?"*

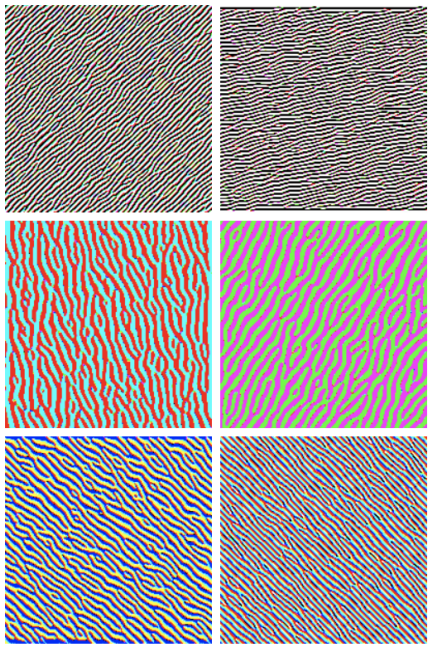
WordNet Structure



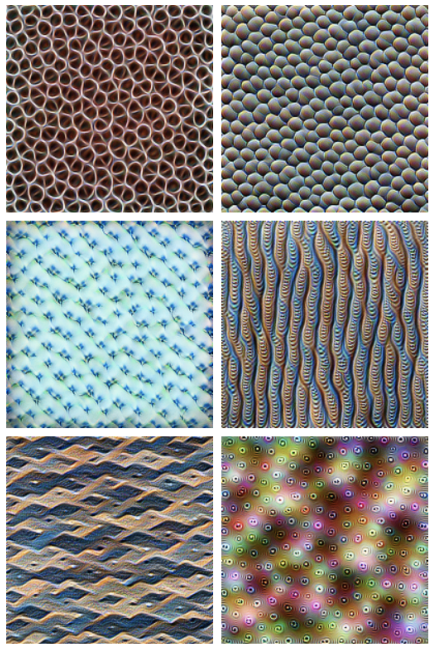
Transfer Learning for Image Recognition

- Features of deep neural networks trained on ImageNet transition from general to task-specific from the first to the last layer
- lower layers learn to model low-level features such as edges
- higher layers model higher-level concepts such as patterns and entire parts or objects
- Knowledge of edges, structures, and the visual composition of objects is relevant for many CV tasks
- ImageNet-like dataset is thus to encourage a model to learn features that will likely generalize to new tasks in the problem domain.

Transfer Learning for Image Recognition

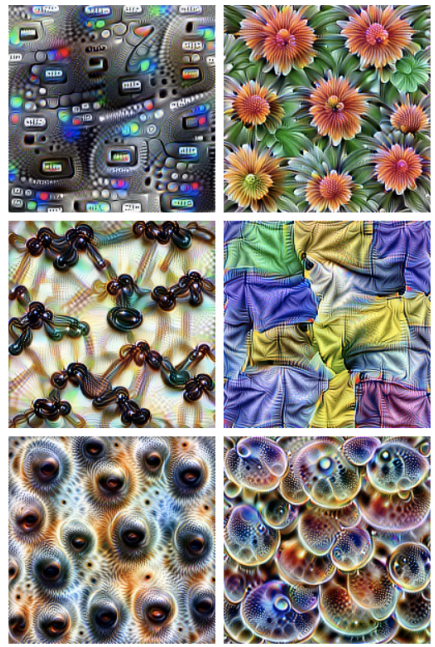


Edges (layer conv2d0)



Textures (layer mixed3a)

Patterns (layer mixed4a)

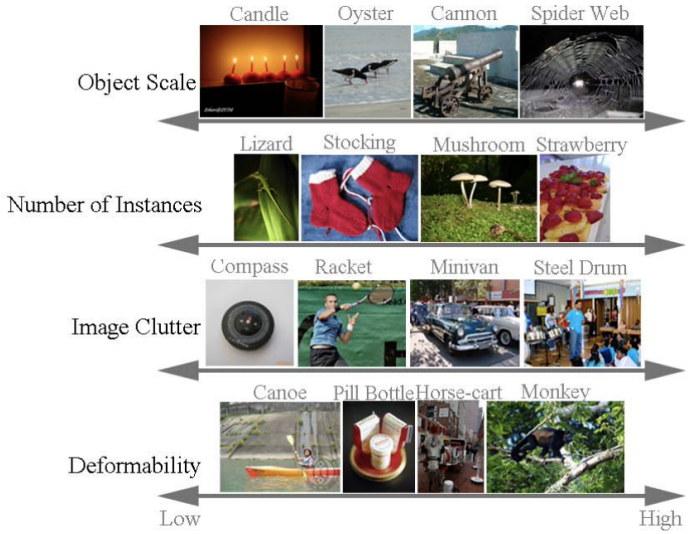


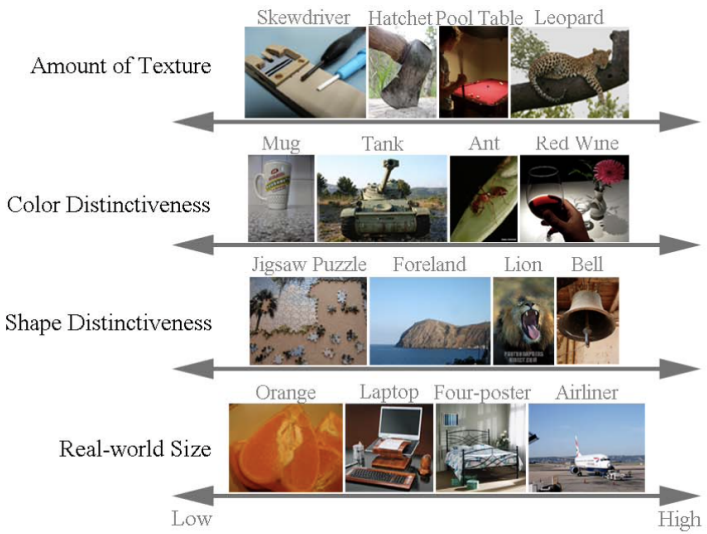
Parts (layers mixed4b & mixed4c)

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- 1000 Object Categories

diversity of ILSVRC



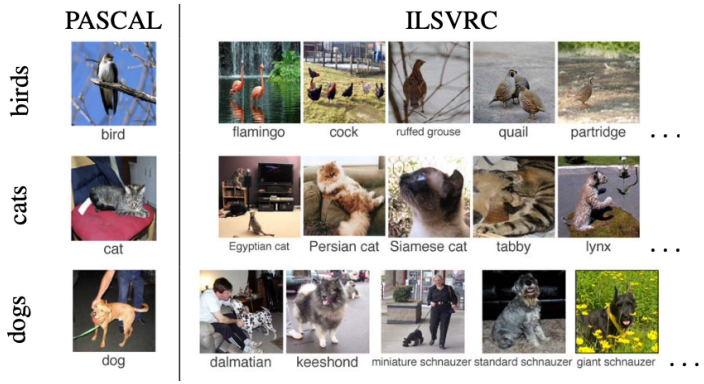


Copenhagen
Business School
HANDELSHØJSKOLEN

Analysis

- systems strong at classifying basic-level object categories
- amount of clutter in images is a strong predictor of detection accuracy
- Man-made objects are significantly more challenging to detect than natural objects
- Objects with distinctive textures are easier than untextured objects

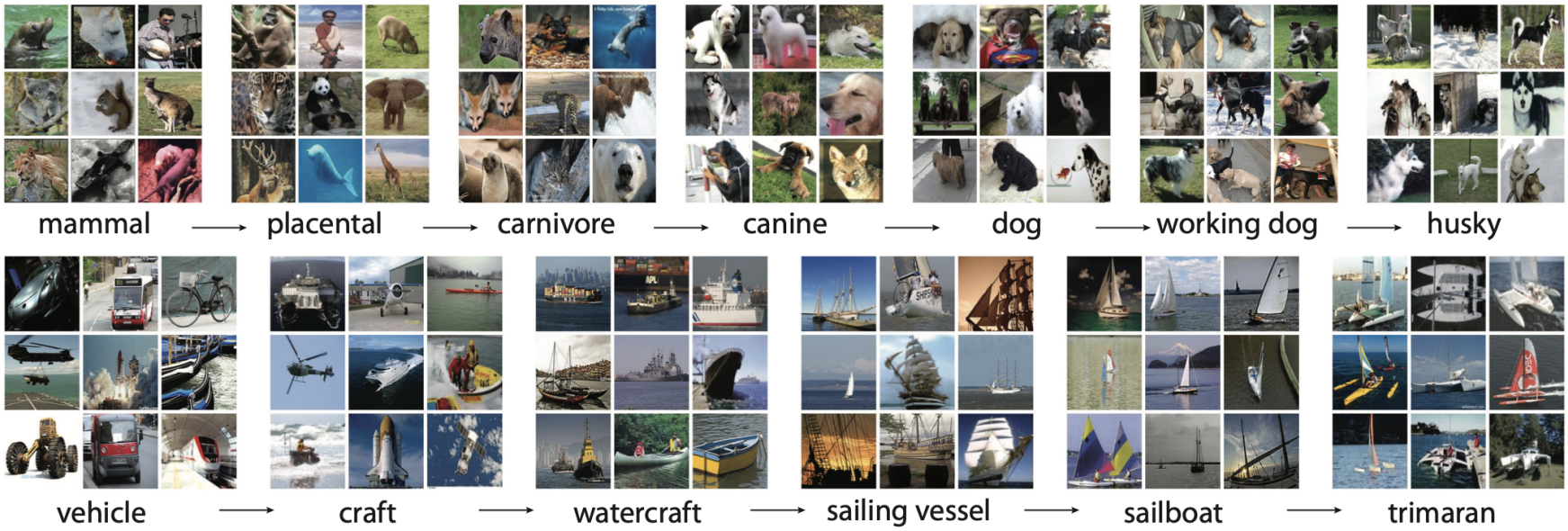
Fine-Grained Classes



ImageNet

- Concept categories from WordNet
- More than 21,000 concepts

ImageNet


mammal
placental
carnivore
canine
dog
working dog
husky
vehicle
craft
watercraft
sailing vessel
sailboat
trimaran

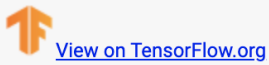
ILSVRC

- subset of ImageNet data – 1000 classes

ILSVRC

- many fine-grained classes – three schnauzer breeds
- 1.2 million images for training
- 50K images for validation
- 100K new images for testing.

Basic classification: Classify images of clothing

 View on TensorFlow.org  Run in Google Colab  View source on GitHub  Download notebook

This guide trains a neural network model to classify images of clothing, like sneakers and shirts. It's okay if you don't understand all the details; this is a fast-paced overview of a complete TensorFlow program with the details explained as you go.

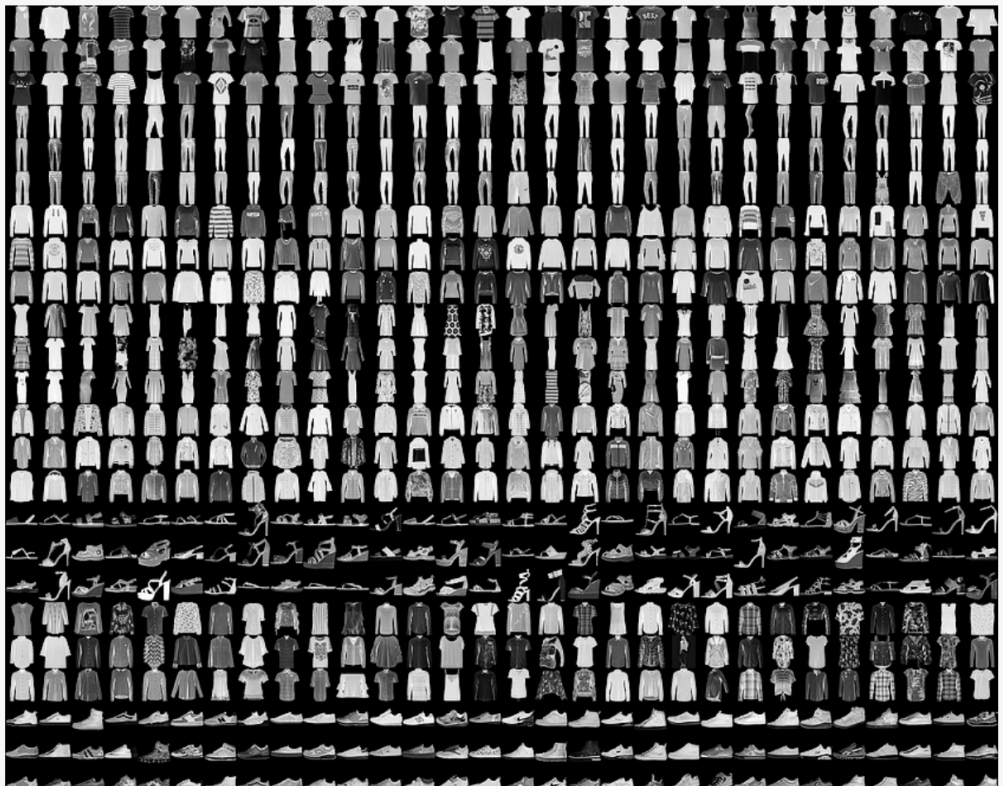
This guide uses tf.keras, a high-level API to build and train models in TensorFlow.

```
[ ]  # TensorFlow and tf.keras
     import tensorflow as tf

     # Helper libraries
     import numpy as np
     import matplotlib.pyplot as plt

     print(tf.__version__)
```

```
2.8.0
```

Import the Fashion MNIST dataset

This guide uses the Fashion MNIST dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels), as seen here:



Fashion MNIST is intended as a drop-in replacement for the classic MNIST dataset—often used as the "Hello, World" of machine learning programs for computer vision. The MNIST dataset contains images of handwritten digits (0, 1, 2, etc.) in a format identical to that of the articles of clothing you'll use here.

This guide uses Fashion MNIST for variety, and because it's a slightly more challenging problem than regular MNIST. Both datasets are relatively small and are used to verify that an algorithm works as expected. They're good starting points to test and debug code.

Here, 60,000 images are used to train the network and 10,000 images to evaluate how accurately the network learned to classify images. You can access the Fashion MNIST directly from TensorFlow. Import and load the Fashion MNIST data directly from TensorFlow:

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-ic
32768/29515 [==============================] - 0s 0us/step
40960/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-ic
```

Loading the dataset returns four NumPy arrays:

- The `train_images` and `train_labels` arrays are the *training set*—the data the model uses to learn.
- The model is tested against the *test set*, the `test_images`, and `test_labels` arrays.

The images are 28x28 NumPy arrays, with pixel values ranging from 0 to 255. The *labels* are an array of integers, ranging from 0 to 9. These correspond to the *class* of clothing the image represents:

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Each image is mapped to a single label. Since the *class names* are not included with the dataset, store them here to use later when plotting the images:

```
[ ]  class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                    'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Explore the data

Let's explore the format of the dataset before training the model. The following shows there are 60,000 images in the training set, with each image represented as 28 x 28 pixels:

```
[ ] train_images.shape
```

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

Likewise, there are 60,000 labels in the training set:

```
[ ] len(train_labels)
```

```
60000
```

Each label is an integer between 0 and 9:

```
[ ] train_labels
```

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

There are 10,000 images in the test set. Again, each image is represented as 28 x 28 pixels:

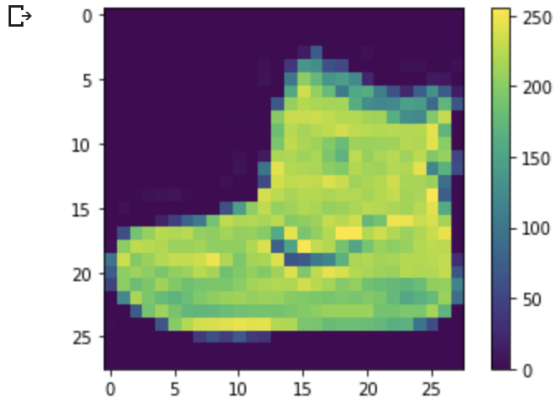
```
[ ] test_images.shape
```

```
(10000, 28, 28)
```

Preprocess the data

The data must be preprocessed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255:

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



```
train_images[0]
```

```
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,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,
          0,   0,  13,  73,   0,   0,   1,   4,   0,   0,   0,   0,   1,
          1,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
          0,  36, 136, 127,  62,  54,   0,   0,   0,   1,   3,   4,   0,
          0,   3],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   6,
          0, 102, 204, 176, 134, 144, 123,  23,   0,   0,   0,   0,  12,
         10,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0, 155, 236, 207, 178, 107, 156, 161, 109,  64,  23,  77, 130,
         72,  15],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
         69, 207, 223, 218, 216, 216, 163, 127, 121, 122, 146, 141,  88,
        172,  66],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   1,   1,   0,
        200, 232, 232, 233, 229, 223, 223, 215, 213, 164, 127, 123, 196,
        229,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        183, 225, 216, 223, 228, 235, 227, 224, 222, 224, 221, 223, 245,
        173,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        193, 228, 218, 213, 198, 180, 212, 210, 211, 213, 223, 220, 243,
        202,   0],
```

Copenhagen
Business School
HANDELSHØJSKOLEN

Scale these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It's important that the *training set* and the *testing set* be preprocessed in the same way:

```
[ ]  train_images = train_images / 255.0

     test_images = test_images / 255.0
```

To verify that the data is in the correct format and that you're ready to build and train the network, let's display the first 25 images from the *training set* and display the class name below each image.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Build the model

Building the neural network requires configuring the layers of the model, then compiling the model.

Set up the layers

The basic building block of a neural network is the *layer*. Layers extract representations from the data fed into them. Hopefully, these representations are meaningful for the problem at hand.

Most of deep learning consists of chaining together simple layers. Most layers, such as `tf.keras.layers.Dense`, have parameters that are learned during training.

```
[ ]  model = tf.keras.Sequential([
         tf.keras.layers.Flatten(input_shape=(28, 28)),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dense(10)
     ])
```

The first layer in this network, `tf.keras.layers.Flatten`, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data.

After the pixels are flattened, the network consists of a sequence of two `tf.keras.layers.Dense` layers. These are densely connected, or fully connected, neural layers. The first `Dense` layer has 128 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes.

Train the model

Training the neural network model requires the following steps:

1. Feed the training data to the model. In this example, the training data is in the `train_images` and `train_labels` arrays.
2. The model learns to associate images and labels.
3. You ask the model to make predictions about a test set—in this example, the `test_images` array.
4. Verify that the predictions match the labels from the `test_labels` array.

Feed the model

To start training, call the `model.fit` method—so called because it "fits" the model to the training data:

```
model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5071 - accuracy: 0.8241
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3742 - accuracy: 0.8653
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3386 - accuracy: 0.8773
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3135 - accuracy: 0.8856
Epoch 5/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2949 - accuracy: 0.8908
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2817 - accuracy: 0.8963
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2700 - accuracy: 0.9000
```

Evaluate accuracy

Next, compare how the model performs on the test dataset:

```
[ ]  test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

     print('\nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 0.3333 - accuracy: 0.8831 - 540ms/epoch - 2ms/step

Test accuracy: 0.8830999732017517
```