Preparing images for deep learning

Normalization / Standardization / Feature Scaling

- Standardization (most widely used)
  - Works well for populations that are normally distributed
  - Output can be -ve (bad for methods that expect positive inputs, e.g. RBMs)

$$x' = \frac{x - \bar{x}}{\sigma}$$

- Rescaling (min-max normalization)
  - Very fast

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Mean normalization

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

- Rescaling by dividing by maximum (most common for images)
- Why data normalization or standardization?
  - In stochastic gradient descent, feature scaling can improve the convergence speed of the algorithm
  - In support vector machines, it can reduce the time to find support vectors

5.2.4 Data preprocessing

- We should load the data into memory to do the training
  - Here is what we have been doing:

In future, we will have X but not Y!

$\underbrace{\text{Input (X)}}$ $\underbrace{\text{Output (Y)}}$

Available data

Fixed acidity	Volatile acidity	Citric acid	Residual sugar	…	Sulphates	Alcohol	Quality
7.4	0.70	0.00	1.90		0.6	9.4	5
7.8	0.88	0.00	2.60		0.7	9.8	5
7.8	0.76	0.04	2.30		0.7	9.8	5
11.2	0.28	0.56	1.90		0.6	9.8	6
7.4	0.70	0.00	1.90		0.6	9.4	5
…	…	…	…		…	…	…
7.9	0.60	0.06	1.60		0.5	9.4	5
7.3	0.65	0.00	1.20		0.5	10.0	7
7.8	0.58	0.02	2.00		0.6	9.5	7
7.5	0.50	0.36	6.10		0.8	10.5	5
6.7	0.58	0.08	1.80		0.5	9.2	5

Split for training and validation →

XTRAIN / YTRAIN / XVALID / YVALID

Fixed acidity	Volatile acidity	Citric acid	Residual sugar	…	Sulphates	Alcohol	Quality
7.4	0.70	0.00	1.90		0.6	9.4	5
7.8	0.88	0.00	2.60		0.7	9.8	5
7.8	0.76	0.04	2.30		0.7	9.8	5
11.2	0.28	0.56	1.90		0.6	9.8	6
7.4	0.70	0.00	1.90		0.6	9.4	5
…	…	…	…		…	…	…
7.9	0.60	0.06	1.60		0.5	9.4	5
7.3	0.65	0.00	1.20		0.5	10.0	7
7.8	0.58	0.02	2.00		0.6	9.5	7
7.5	0.50	0.36	6.10		0.8	10.5	5
6.7	0.58	0.08	1.80		0.5	9.2	5

```
history = model.fit(XTRAIN, YTRAIN, validation_data=(XVALIDATION, YVALIDATION), epochs=256)
```

- If we have Terabytes of data, can we load the entire dataset into XTRAIN and YTRAIN? Why?

THINK PAIR SHARE

An ideal way of loading the data & training

- As the training starts:
    1. Read a picture file
    2. Decode the JPEG content to RGB grids of pixels
    3. Convert these into floating-point tensors
    4. Rescale the pixel values (between 0 and 255) to the [0, 1] interval
    5. Add it to the pool of training/validation dataset

- Keras has utilities to take care of these steps automatically
    - contains the class `ImageDataGenerator` which lets you quickly set up Python generators that can automatically turn image files on disk into batches of preprocessed tensors

Python generators

- Python has a ‘yield’ operator that you can use in place of ‘return’
- A Python generator is an object that acts as an iterator
    - You can use it with the for … in operator

```python
def generator():
    i = 0
    while True:
        i += 1
        yield i
```

```python
for x in generator():
    print(x)
    if x > 5:
        break
```

5.7 Data generator for generating images

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150)
        batch_size=20,
        class_mode='binary')
```

Rescales all images by 1/255

Target directory

Resizes all images to 150 × 150

Because you use binary_crossentropy loss, you need binary labels.

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=10,
      validation_data=validation_generator,
      validation_steps=50)
```

5.2.5 Data augmentation

- The approach of generating more training data from existing training samples
  - by augmenting the samples via a number of random transformations that yield believable-looking images
- The goal is: “at training time, the model will never see the exact same picture twice”
  - This helps expose the model to more aspects of the data and generalize better
- If we read images using ImageDataGenerator instance (in Keras)
  - a number of random transformations can be performed

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150)
        batch_size=20,
        class_mode='binary')
```

Rescales all images by 1/255

Target directory

Resizes all images to 150 × 150

Because you use binary_crossentropy loss, you need binary labels.

Previous Code

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

Augmentation using `ImageDataGenerator`

- `rotation_range`

  a value in degrees (0–180), a range within which to randomly rotate pictures
- `width_shift` and `height_shift`

  ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally
- `shear_range`

  randomly applying shearing transformations.
- `zoom_range`

  randomly zooming inside pictures.
- `horizontal_flip`

  is for randomly flipping half the images horizontally—relevant when there are no assumptions of horizontal asymmetry (for example, real-world pictures)
- `fill_mode`

  the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift

… (there are more)

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```