Quantitative Big Imaging - Building and Augmenting Datasets

Anders Kaestner

Mar 04, 2021

CONTENTS

Quantitative Big Imaging ETHZ: 227-0966-00L

CHAPTER

ONE

LET'S LOAD SOME MODULES FOR THE NOTEBOOK

```
import matplotlib.pyplot as plt
import matplotlib    as mpl
import numpy         as np
import skimage       as ski
import skimage.io    as io
from skimage.morphology import disk
import scipy.ndimage as ndimage

%matplotlib inline
mpl.rcParams['figure.dpi'] = 300
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-f8507326f08a> in <module>
      2 import matplotlib    as mpl
      3 import numpy         as np
----> 4 import skimage       as ski
      5 import skimage.io    as io
      6 from skimage.morphology import disk

ModuleNotFoundError: No module named 'skimage'
```

CHAPTER

TWO

TODAY'S LECTURE

Creating Datasets

- Famous Datasets
- Types of Datasets
- What makes a good dataet?
- Building your own
- "scrape, mine, move, annotate, review, and preprocess" - Kathy Scott
- tools to use
- simulation

Augmentation

- How can you artifically increase the size of your dataset?
- What are the limits of these increases

Baselines

- What is a baseline?
- Example: Nearest Neighbor

CHAPTER

THREE

REFERENCES

- Revisiting **Unreasonable Effectiveness of Data** in Deep Learning Era: https://arxiv.org/abs/1707.02968
- Building Datasets
  - Python Machine Learning 2nd Edition by Sebastian Raschka, Packt Publishing Ltd. 2017
  - Chapter 2: Building Good Datasets: https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch04/ch04.ipynb
  - A Standardised Approach for Preparing Imaging Data for Machine Learning Tasks in Radiology https://doi.org/10.1007/978-3-319-94878-2_6
- Creating Datasets / Crowdsourcing
- Mindcontrol: A web application for brain segmentation quality control: https://www.sciencedirect.com/science/article/pii/S1053811917302707
- Combining citizen science and deep learning to amplify expertise in neuroimaging: https://www.biorxiv.org/content/10.1101/363382v1.abstract
- Augmentation
- https://github.com/aleju/imgaug
- https://github.com/mdbloice/Augmentor

CHAPTER

FOUR

MOTIVATION

Most of you taking this class are rightfully excited to learn about new tools and algorithms to analyzing *your* data.

This lecture is a bit of an anomaly and perhaps disappointment because it doesn't cover any algorithms, or tools.

- So you might ask, why are we spending so much time on datasets?
- You already collected data (sometimes lots of it) that is why you took this class?!

. . . let's see what some other people say

4.1 Sean Taylor (Research Scientist at Facebook)

4.2 Andrej Karpathy (Director of AI at Tesla)

4.3 Kathy Scott (Image Analytics Lead at Planet Labs)

4.4 Data is important

It probably isn't the *new* oil, but it forms an essential component for building modern tools today.

- Testing good algorithms *requires* good data
- If you don't know what to expect how do you know your algorithm worked?
- If you have dozens of edge cases how can you make sure it works on each one?
- If a new algorithm is developed every few hours, how can you be confident they actually work better (facebook's site has a new version multiple times per day and their app every other day)
- For machine learning, even building requires good data
- If you count cells maybe you can write your own algorithm,
- but if you are trying to detect subtle changes in cell structure that indicate cancer you probably can't write a list of simple mathematical rules yourself.

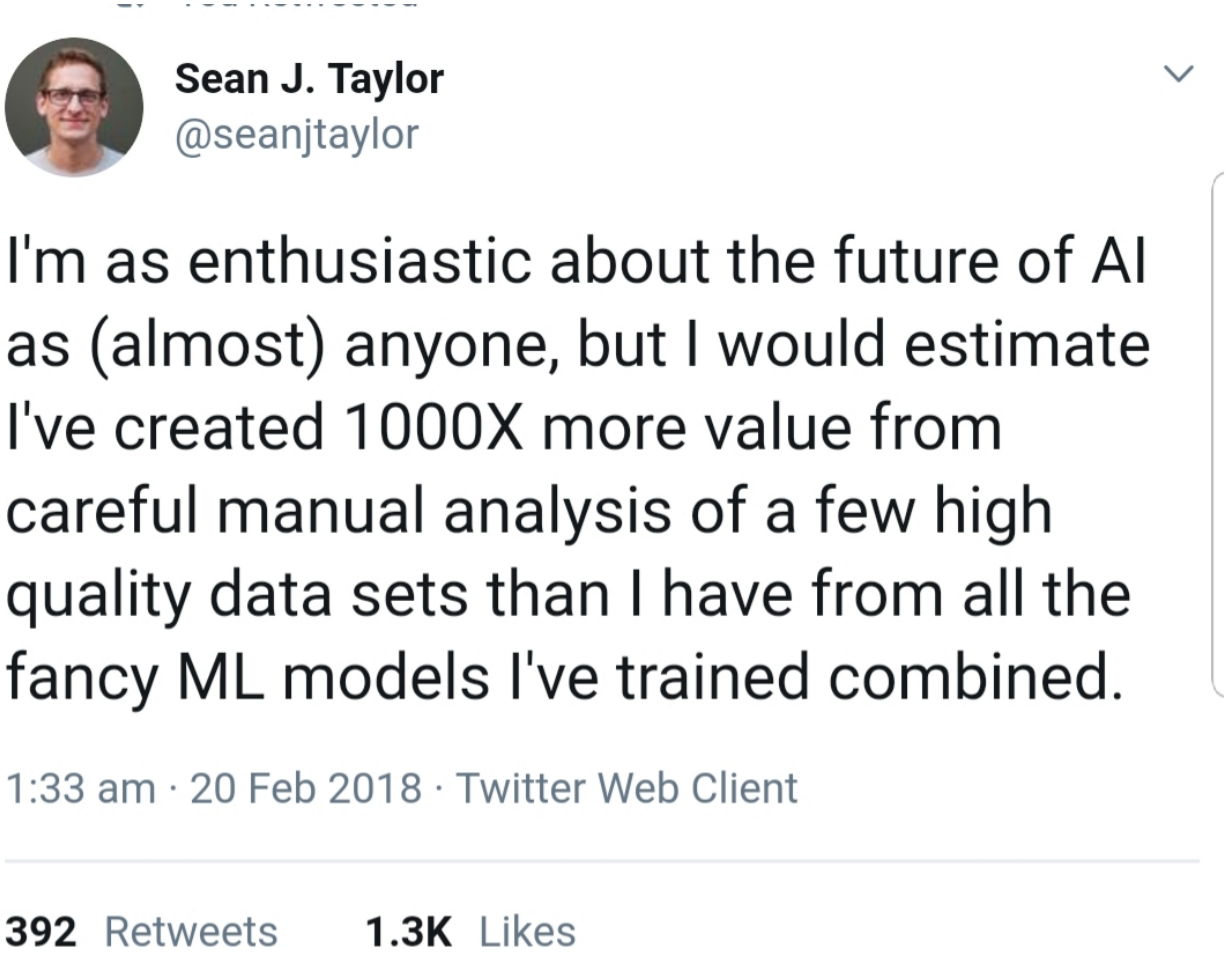


Fig. 4.1: Realistic thoughts about AI.

Why you need to improve your training data, and how to do it

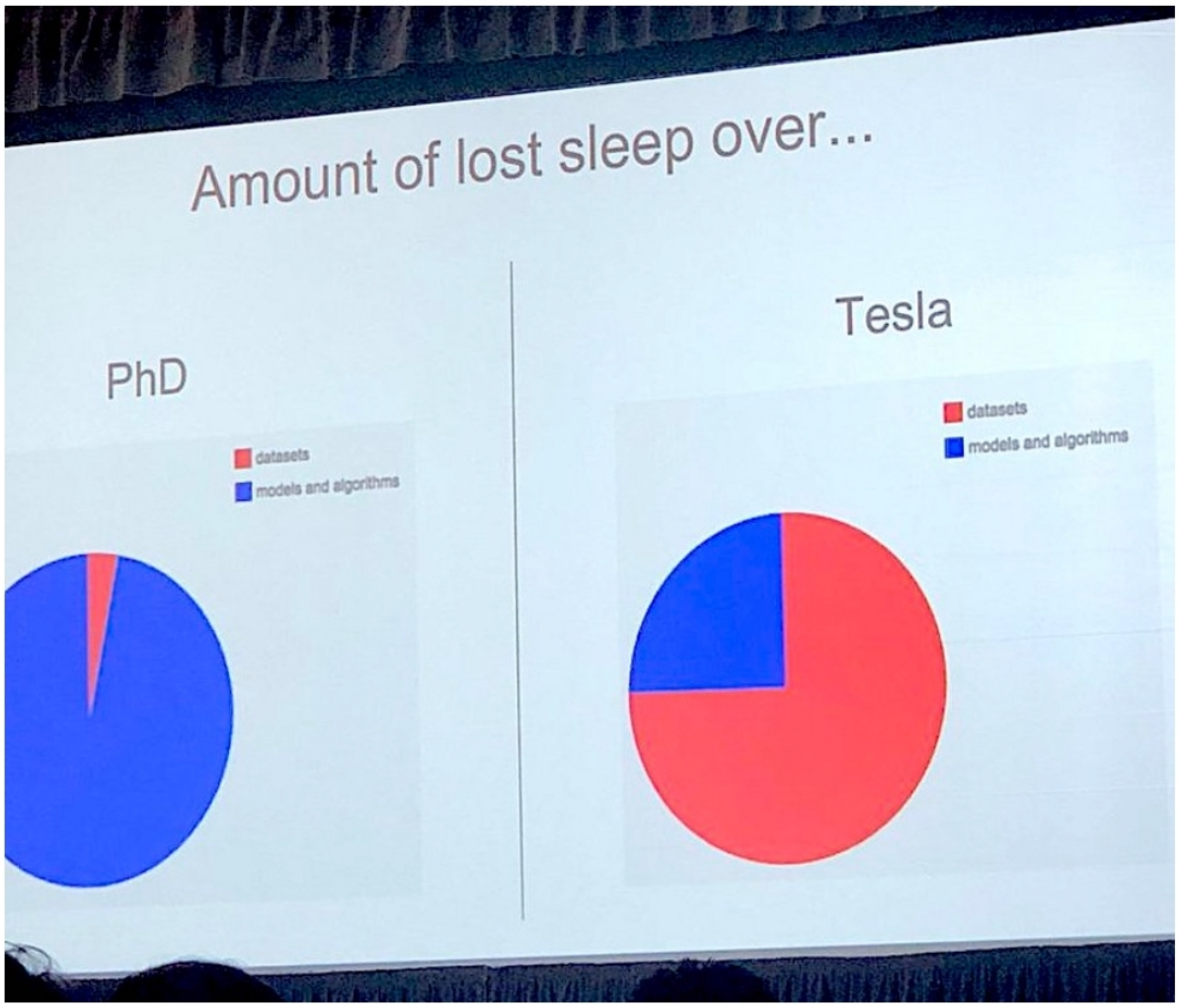


Photo by Lisha Li

Andrej Karpathy showed this slide as part of his talk at Train AI and I loved it! It captures the difference between deep learning

Fig. 4.2: Time spent on different tasks.

4.5 Data is reusable

- Well organized and structured data is very easy to reuse.
- Another project can easily combine your data with their data in order to get even better results.
- Algorithms are often messy, complicated, poorly written, ... (especially so if written by students trying to graduate on time)

4.6 Data recycling saves time and improves performance

CHAPTER

FIVE

FAMOUS DATASETS

The primary success of datasets has been shown through the most famous datasets collected.

Here I show

- Two of the most famous general datasets
  - MNIST Digits
  - ImageNET
- and one of the most famous medical datasets.
  - BRATS

The famous datasets are important for basic network training.

5.1 MNIST Digits

Modified NIST (National Institute of Standards and Technology) created a list of handwritten digits

5.2 ImageNet

- ImageNet is an image database
  - organized according to the WordNet hierarchy (currently only the nouns),
  - each node of the hierarchy is depicted by hundreds and thousands of images.
- 1000 different categories and >1M images.
- Not just dog/cat, but wolf vs german shepard,

CNN architectures

5.3 BRATS

Segmenting Tumors in Multimodal MRI Brain Images.

5.4 What story did these datasets tell?

Each of these datasets is very different from images with fewer than 1000 pixels to images with more than 100MPx, but what they have in common is how their analysis has changed.

5.4.1 Hand-crafted features

All of these datasets used to be analyzed by domain experts with hand-crafted features.

- A handwriting expert using graph topology to assign images to digits
- A computer vision expert using gradients common in faces to identify people in ImageNet
- A biomedical engineer using knowledge of different modalities to fuse them together and cluster healthy and tumorous tissue

5.4.2 Machine Learning / Deep Learning

Starting in the early 2010s, the approaches of deep learning began to improve and become more computationally efficient. With these techniques groups with **absolutely no domain knowledge** could begin building algorithms and winning contests based on these datasets

5.5 So Deep Learning always wins?

No, that isn't the point of this lecture.

Even if you aren't using deep learning the point of these stories is having

- well-labeled,
- structured,
- and organized datasets

makes your problem *a lot more accessible* for other groups and enables a variety of different approaches to be tried.

Ultimately it enables better solutions to be made and you to be confident that the solutions are in fact better

5.6 Other Datasets

- Grand-Challenge.org a large number of challenges in the biomedical area
- Kaggle Datasets
- Google Dataset Search

5.7 What makes a good dataset?

5.7.1 Lots of images

- Small datasets can be useful but here the bigger the better
- Particularly if you have
  - Complicated problems
  - Very subtle differences (ie a lung tumor looks mostly like normal lung tissue but it is in a place it shouldn't be)
  - Class imbalance

5.8 What makes a good dataset?

5.8.1 Lots of diversity

- Is it what data 'in the wild' really looks like?
- Lots of different
  - Scanners/reconstruction algorithms,
  - noise levels,
  - illumination types,
  - rotation,
  - colors, ...
- Many examples from different categories
  - *if you only have one male with breast cancer it will be hard to generalize exactly what that looks like*

5.9 What makes a good dataset?

5.9.1 Meaningful labels

- Clear task or question
- Unambiguous (would multiple different labelers come to the same conclusion)
- Able to be derived from the image alone
  - *A label that someone cannot afford insurance is interesting but it would be nearly impossible to determine that from an X-ray of their lungs*
- Quantiative!
- Non-obvious
  - *A label saying an image is bright is not a helpful label because you could look at the histogram and say that*

CHAPTER

SIX

PURPOSE OF DIFFERENT TYPES OF DATASETS

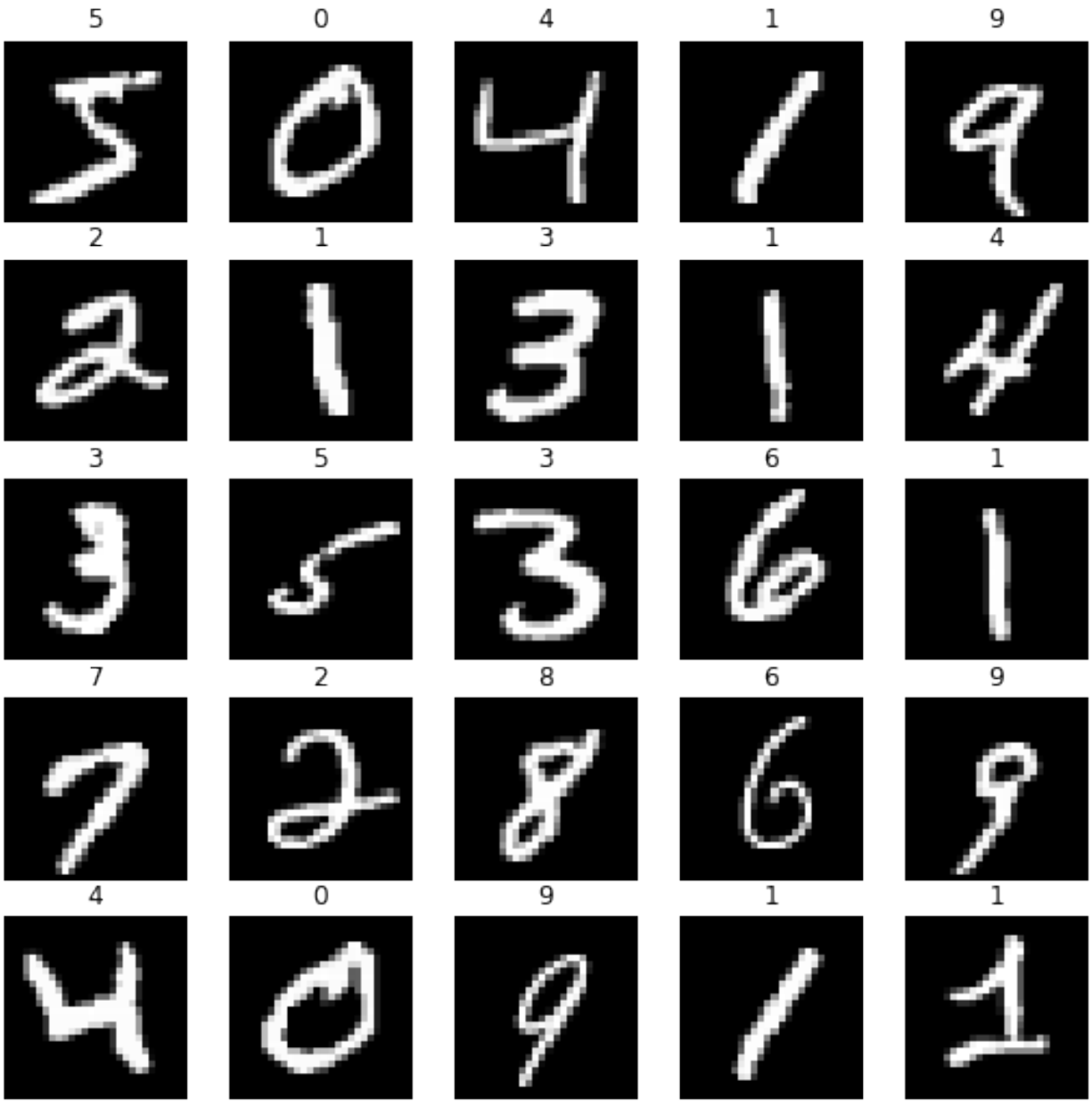
- Classification
- Regression
- Segmentation
- Detection
- Other

6.1 Classification

- Taking an image and putting it into a category
- Each image should have exactly one category
- The categories should be non-ordered
- Example:
- Cat vs Dog
- Cancer vs Healthy

6.1.1 Classification example

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from skimage.util import montage as montage2d
%matplotlib inline
(img, label), _ = mnist.load_data()
fig, m_axs = plt.subplots(5, 5, figsize=(9, 9))
for c_ax, c_img, c_label in zip(m_axs.flatten(), img, label):
    c_ax.imshow(c_img, cmap='gray')
    c_ax.set_title(c_label)
    c_ax.axis('off')
```



6.2 Regression

- Taking an image and predicting one (or more) decimal values
- Examples:
- Value of a house from the picture taken by owner
- Risk of hurricane from satellite image

6.2.1 Regression example Age from X-Rays

More details

6.3 Segmentation

- Taking an image and predicting one (or more) values for each pixel
- Every pixel needs a label (and a pixel cannot have multiple labels)
- Typically limited to a few (<20) different types of objects
- Examples:
- Where a tumor is from an image of the lungs
- Where streets are from satellite images of a neighborhood

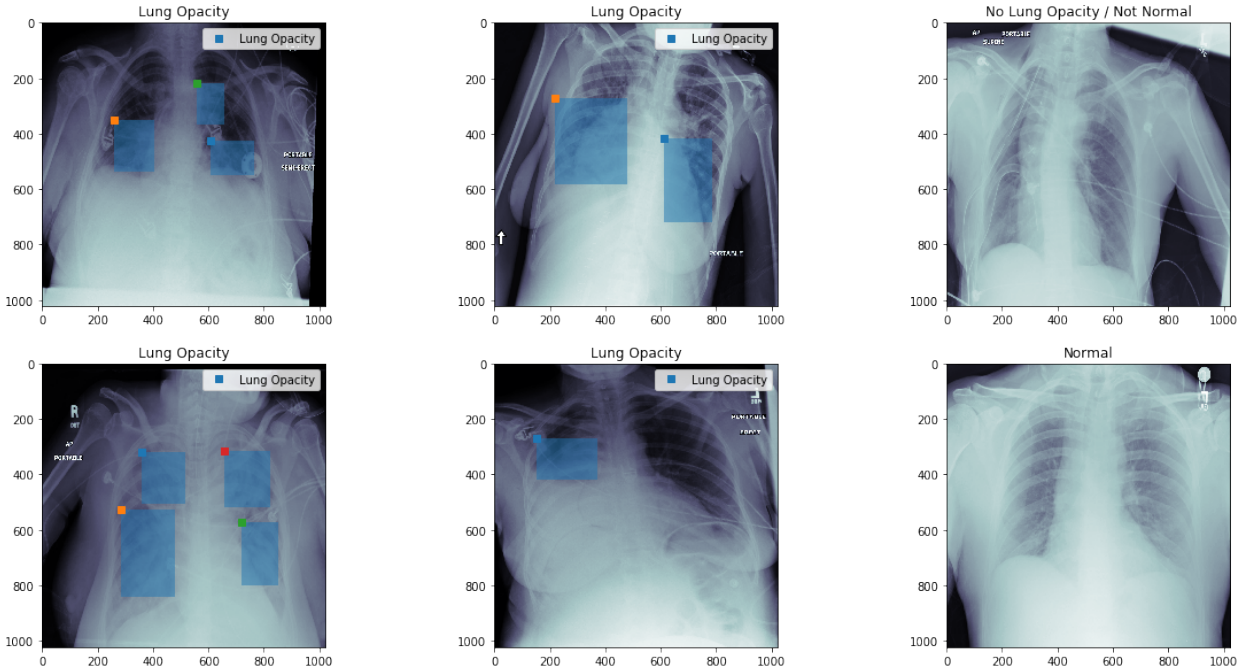
6.3.1 Segmentation example: Nuclei in Microscope Images

More details on Kaggle

6.4 Detection

- Taking an image and predicting where and which type of objects appear
- Generally bounding box rather then specific pixels
- Multiple objects can overlap

6.4.1 Detection example: Opaque Regions in X-Rays



More details on Kaggle

6.5 Other

- Unlimited possibilities here
- Horses to Zebras

6.5.1 Image Enhancement

- Denoising Learning to See in the Dark
- Super-resolution

CHAPTER

SEVEN

BUILDING YOUR OWN DATA SETS

- Very time consuming
- Not a lot of great tools
- Very problem specific

7.1 Code-free

7.1.1 Classification

- Organize images into folders

7.1.2 Regression

- Create an excel file (first column image name, next columns to regress)

7.1.3 Segmentation / Object Detection

- Take FIJI and manually draw region to be identified and save it as a grayscale image

7.2 Software for data labelling

7.2.1 Free tools

- Classification / Segmentation: https://github.com/Labelbox/Labelbox
- Classification/ Object Detection: http://labelme.csail.mit.edu/Release3.0/
- Classification: https://github.com/janfreyberg/superintendent: https://www.youtube.com/watch?v=fMg0mPYiEx0
- Classification/ Detection: https://github.com/chestrays/jupyanno: https://www.youtube.com/watch?v=XDIJU5Beg_w
- Classification (Tinder for Brain MRI): https://braindr.us/#/

7.2.2 Commercial Approaches

- https://www.figure-eight.com/
- MightyAI / Spare5: https://mighty.ai/ https://app.spare5.com/fives/sign_in

7.3 Simulations

Another way to enhance or expand your dataset is to use simulations

- already incorporate realistic data (game engines, 3D rendering, physics models)
- 100% accurate ground truth (original models)
- unlimited, risk-free playability (driving cars in the world is more dangerous)

7.3.1 Examples

- P. Fuchs et al. - Generating Meaningful Synthetic Ground Truth for Pore Detection in Cast Aluminum Parts, iCT 2019, Padova
- https://download.visinf.tu-darmstadt.de/data/from_games/
- https://pythonprogramming.net/self-driving-car-neural-network-training-data-python-plays-gta-v/
- https://towardsdatascience.com/learning-from-simulated-data-ff4be63ac89c

CHAPTER

EIGHT

DATASET PROBLEMS

Some of the issues which can come up with datasets are

- imbalance
- too few examples
- too homogenous
- and other possible problems

These lead to problems with the algorithms built on top of them.

8.1 Bias

8.1.1 The solution was to remove Gorilla from the category

8.2 Better solution would be to have training sets with more diverse people

- IBM Diverse Face Dataset

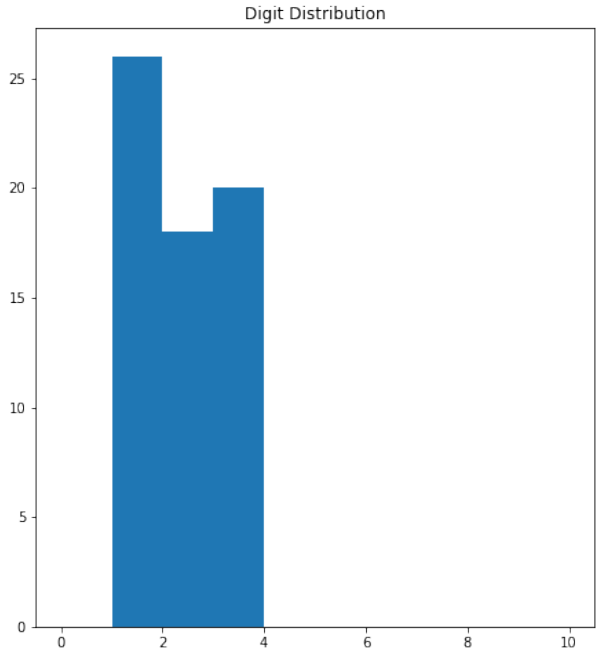
CHAPTER

NINE

EXAMPLE OF IMAGE DATA AND LABELS

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from skimage.util import montage as montage2d
%matplotlib inline
(img, label), _ = mnist.load_data()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))
d_subset = np.where(np.in1d(label, [1, 2, 3]))[0]
ax1.imshow(montage2d(img[d_subset[:64]]), cmap='gray')
ax1.set_title('Images')
ax1.axis('off')
ax2.hist(label[d_subset[:64]], np.arange(11))
ax2.set_title('Digit Distribution');
```



CHAPTER

TEN

AUGMENTATION

- Most groups have too little well-labeled data and labeling new examples can be very expensive.
- Additionally there might not be very many cases of specific classes.
- In medicine this is particularly problematic, because some diseases might only happen a few times in a given hospital and you still want to be able to recognize the disease and not that particular person.

10.1 Handling limited data

10.1.1 Transformations

- Shift
- Zoom
- Rotation
- Intensity
- Normalization
- Scaling
- Color
- Shear

10.1.2 Further modifications

- Add noise
- Blurring

10.2 Some augmentation examples

Retial images from DRIVE prepared by Gian Guido Parenza.

10.3 Limitations of augmentation

- What transformations are normal in the images?
- CT images usually do not get flipped (the head is always on the top)
- The values in CT images have a physical meaning (Hounsfield unit), → scaling them changes the image
- How much distortion is too much?
- Can you still recognize the features?

10.4 Keras ImageDataGenerator

```
ImageDataGenerator(
    ['featurewise_center=False', 'samplewise_center=False', 'featurewise_std_
↪normalization=False', 'samplewise_std_normalization=False', 'zca_whitening=False',
↪'zca_epsilon=1e-06', 'rotation_range=0.0', 'width_shift_range=0.0', 'height_shift_
↪range=0.0', 'shear_range=0.0', 'zoom_range=0.0', 'channel_shift_range=0.0', "fill_
↪mode='nearest'", 'cval=0.0', 'horizontal_flip=False', 'vertical_flip=False',
↪'rescale=None', 'preprocessing_function=None', 'data_format=None'],
)
Docstring:
Generate minibatches of image data with real-time data augmentation.

# Arguments
    featurewise_center: set input mean to 0 over the dataset.
    samplewise_center: set each sample mean to 0.
    featurewise_std_normalization: divide inputs by std of the dataset.
    samplewise_std_normalization: divide each input by its std.
    zca_whitening: apply ZCA whitening.
    zca_epsilon: epsilon for ZCA whitening. Default is 1e-6.
    rotation_range: degrees (0 to 180).
    width_shift_range: fraction of total width, if < 1, or pixels if >= 1.
    height_shift_range: fraction of total height, if < 1, or pixels if >= 1.
    shear_range: shear intensity (shear angle in degrees).
    zoom_range: amount of zoom. if scalar z, zoom will be randomly picked
        in the range [1-z, 1+z]. A sequence of two can be passed instead
        to select this range.
    channel_shift_range: shift range for each channel.
    fill_mode: points outside the boundaries are filled according to the
        given mode ('constant', 'nearest', 'reflect' or 'wrap'). Default
        is 'nearest'.
        Points outside the boundaries of the input are filled according to the given␣
↪mode:
            'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k)
            'nearest':  aaaaaaaa|abcd|dddddddd
            'reflect':  abcddcba|abcd|dcbaabcd
            'wrap':  abcdabcd|abcd|abcdabcd
    cval: value used for points outside the boundaries when fill_mode is
```

(continues on next page)

(continued from previous page)

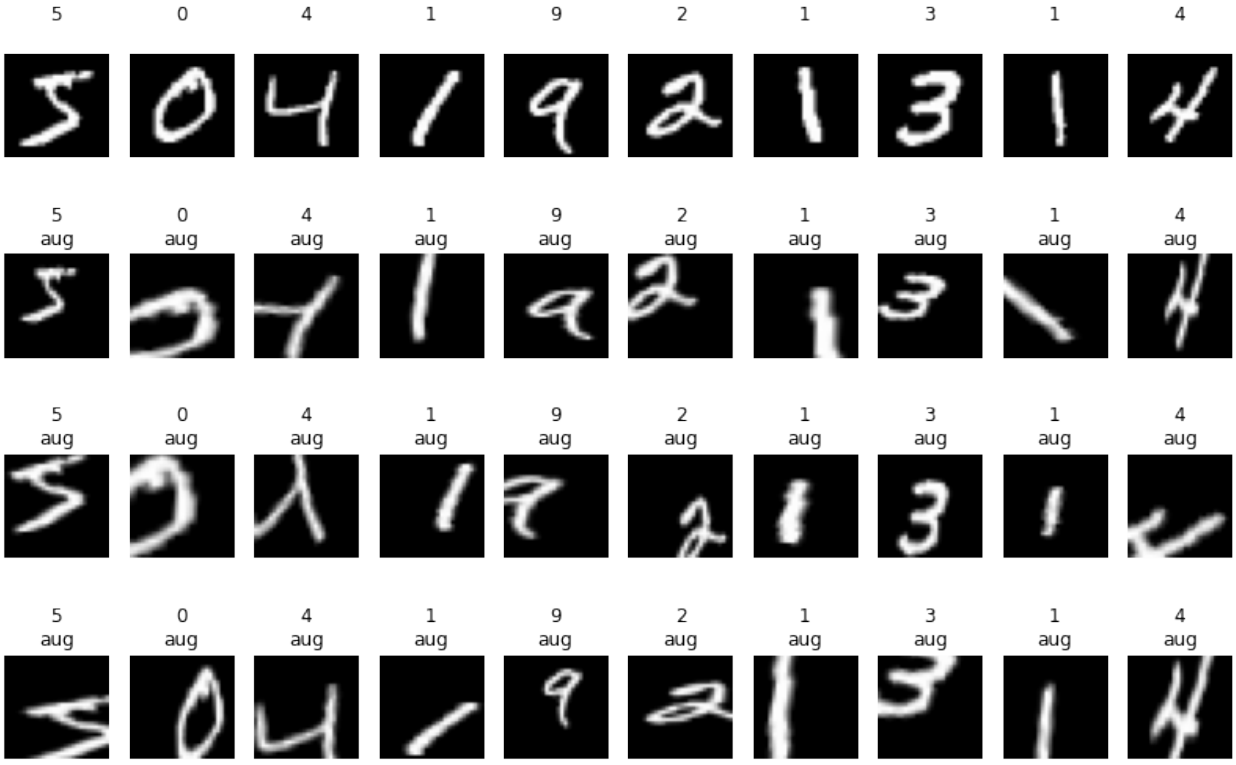
```
        'constant'. Default is 0.
    horizontal_flip: whether to randomly flip images horizontally.
    vertical_flip: whether to randomly flip images vertically.
    rescale: rescaling factor. If None or 0, no rescaling is applied,
        otherwise we multiply the data by the value provided. This is
        applied after the `preprocessing_function` (if any provided)
        but before any other transformation.
    preprocessing_function: function that will be implied on each input.
        The function will run before any other modification on it.
        The function should take one argument:
        one image (Numpy tensor with rank 3),
```

```
from keras.preprocessing.image import ImageDataGenerator
img_aug = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=30.0,
    width_shift_range=0.25,
    height_shift_range=0.25,
    shear_range=0.25,
    zoom_range=0.5,
    fill_mode='nearest',
    horizontal_flip=False,
    vertical_flip=False
)
```

10.5 MNIST

Even something as simple as labeling digits can be very time consuming (maybe 1-2 per second).

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
%matplotlib inline
(img, label), _ = mnist.load_data()
img = np.expand_dims(img, -1)
fig, m_axs = plt.subplots(4, 10, figsize=(14, 9))
# setup augmentation
img_aug.fit(img)
real_aug = img_aug.flow(img[:10], label[:10], shuffle=False)
for c_axs, do_augmentation in zip(m_axs, [False, True, True, True]):
    if do_augmentation:
        img_batch, label_batch = next(real_aug)
    else:
        img_batch, label_batch = img, label
    for c_ax, c_img, c_label in zip(c_axs, img_batch, label_batch):
        c_ax.imshow(c_img[:, :, 0], cmap='gray', vmin=0, vmax=255)
        c_ax.set_title('{}\n{}'.format( c_label, 'aug' if do_augmentation else ''))
        c_ax.axis('off')
```


5 0 4 1 9 2 1 3 1 4
5 aug 0 aug 4 aug 1 aug 9 aug 2 aug 1 aug 3 aug 1 aug 4 aug
5 aug 0 aug 4 aug 1 aug 9 aug 2 aug 1 aug 3 aug 1 aug 4 aug
5 aug 0 aug 4 aug 1 aug 9 aug 2 aug 1 aug 3 aug 1 aug 4 aug

CHAPTER

ELEVEN

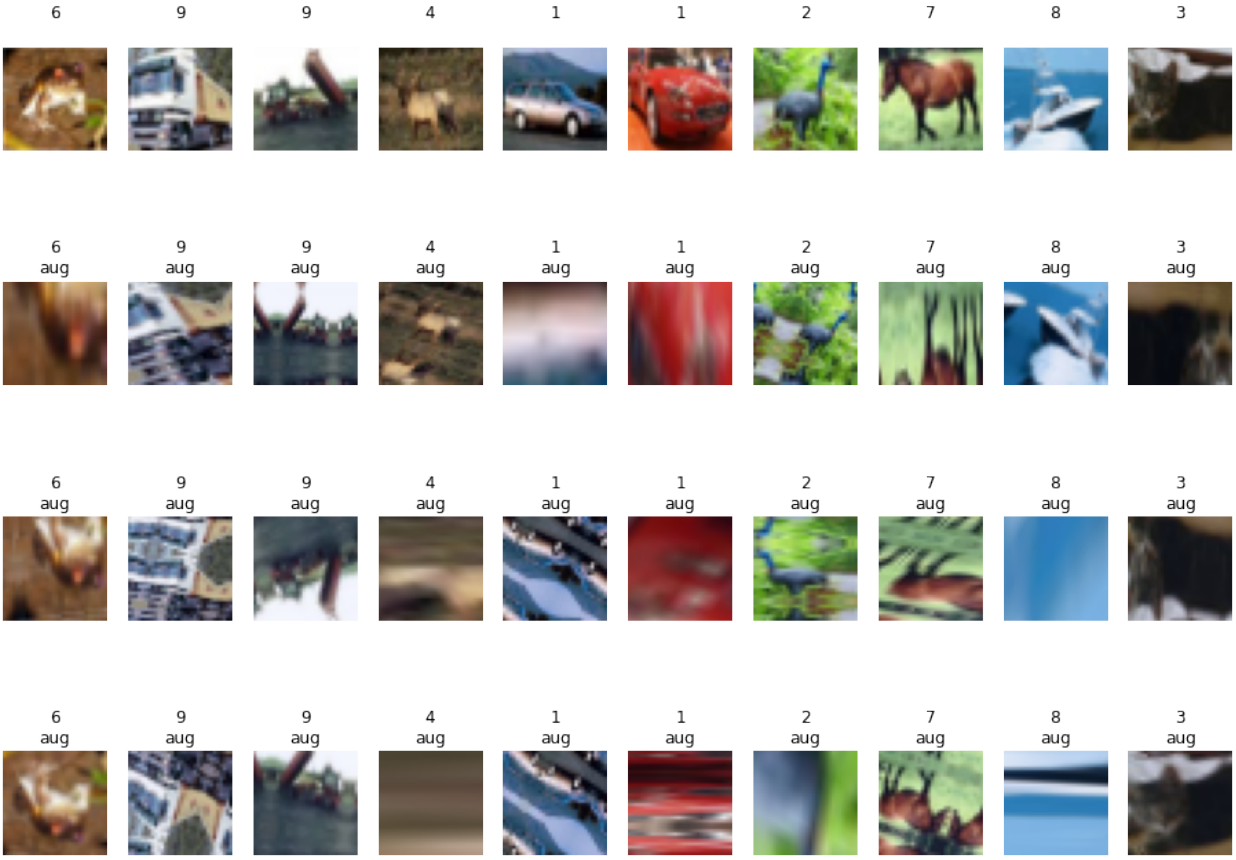
A LARGER OPEN DATA SET

CIFAR10 We can use a more exciting dataset to try some of the other features in augmentation

```
from keras.datasets import cifar10
(img, label), _ = cifar10.load_data()
```

```
img_aug = ImageDataGenerator(
    featurewise_center=True,
    samplewise_center=False,
    zca_whitening=False,
    zca_epsilon=1e-06,
    rotation_range=30.0,
    width_shift_range=0.25,
    height_shift_range=0.25,
    channel_shift_range=0.25,
    shear_range=0.25,
    zoom_range=1,
    fill_mode='reflect',
    horizontal_flip=True,
    vertical_flip=True
)
```

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
%matplotlib inline
fig, m_axs = plt.subplots(4, 10, figsize=(16, 12))
# setup augmentation
img_aug.fit(img)
real_aug = img_aug.flow(img[:10], label[:10], shuffle=False)
for c_axs, do_augmentation in zip(m_axs, [False, True, True, True]):
    if do_augmentation:
        img_batch, label_batch = next(real_aug)
        img_batch -= img_batch.min()
        img_batch = np.clip(img_batch/img_batch.max() *
                            255, 0, 255).astype('uint8')
    else:
        img_batch, label_batch = img, label
    for c_ax, c_img, c_label in zip(c_axs, img_batch, label_batch):
        c_ax.imshow(c_img)
        c_ax.set_title('{}\n{}'.format(
            c_label[0], 'aug' if do_augmentation else ''))
        c_ax.axis('off')
```


6
9
9
4
1
1
2
7
8
3
6 aug
9 aug
9 aug
4 aug
1 aug
1 aug
2 aug
7 aug
8 aug
3 aug
6 aug
9 aug
9 aug
4 aug
1 aug
1 aug
2 aug
7 aug
8 aug
3 aug
6 aug
9 aug
9 aug
4 aug
1 aug
1 aug
2 aug
7 aug
8 aug
3 aug

CHAPTER

TWELVE

BASELINES

- A baseline is a simple, easily implemented and understood model that illustrates the problem and the 'worst-case scenario' for a model that learns nothing (some models will do worse, but these are especially useless).
- Why is this important?

12.1 Baseline model example

I have a a model that is >99% accurate for predicting breast cancer

$$\text{DoIHaveBreastCancer}(\text{Age}, \text{Weight}, \text{Race}) = \text{No!}$$

12.2 The dummy classifier

```
from sklearn.dummy import DummyClassifier
dc = DummyClassifier(strategy='most_frequent')
dc.fit([0, 1, 2, 3],
       ['Healthy', 'Healthy', 'Healthy', 'Cancer'])
```

```
DummyClassifier(strategy='most_frequent')
```

```
dc.predict([0]), dc.predict([1]), dc.predict([3]), dc.predict([100])
```

```
(array(['Healthy'], dtype='<U7'),
 array(['Healthy'], dtype='<U7'),
 array(['Healthy'], dtype='<U7'),
 array(['Healthy'], dtype='<U7'))
```

12.3 Try dummy classifier on MNIST data

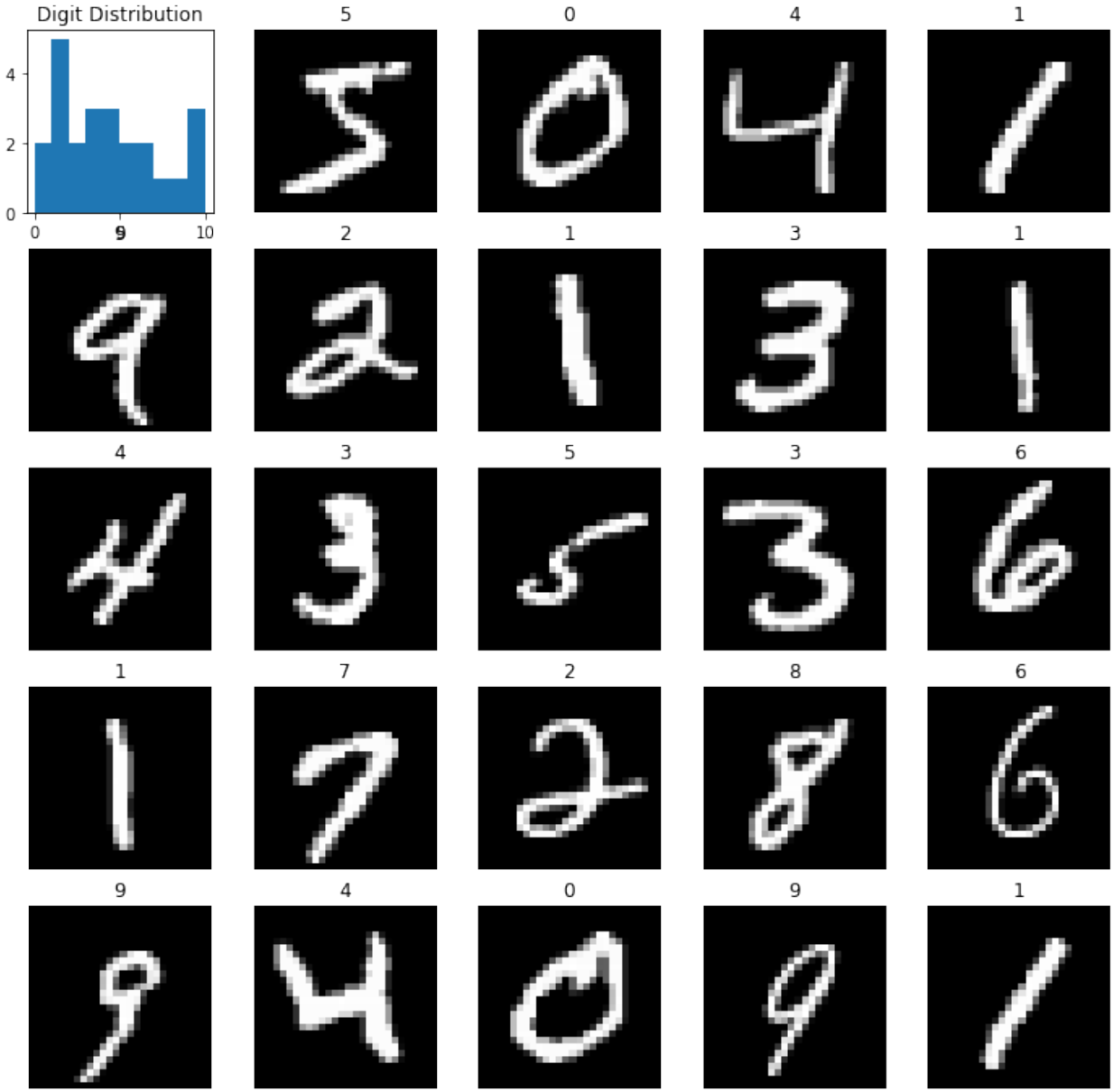
First we load the data...

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from skimage.util import montage as montage2d
```

(continues on next page)

(continued from previous page)

```
%matplotlib inline
(img, label), _ = mnist.load_data()
fig, m_axs = plt.subplots(5, 5, figsize=(12, 12))
m_axs[0, 0].hist(label[:24], np.arange(11))
m_axs[0, 0].set_title('Digit Distribution')
for i, c_ax in enumerate(m_axs.flatten()[1:]):
    c_ax.imshow(img[i], cmap='gray')
    c_ax.set_title(label[i])
    c_ax.axis('off')
```



12.4 Let's train the model...

```
dc = DummyClassifier(strategy='most_frequent')
dc.fit(img[:24], label[:24])
```

```
DummyClassifier(strategy='most_frequent')
```

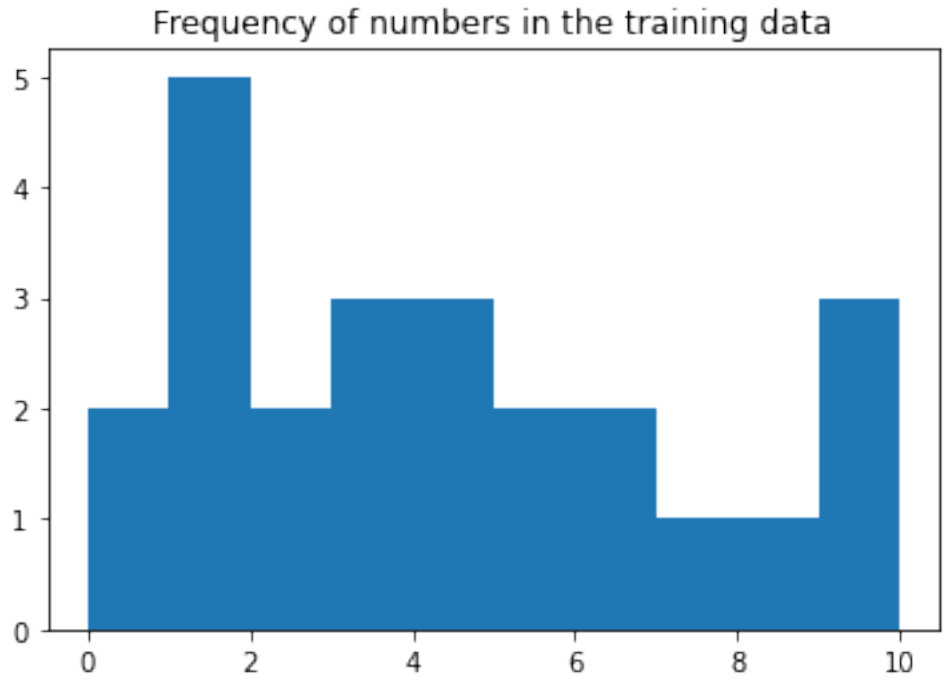
A basic test

```
dc.predict(img[0:10])
```

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

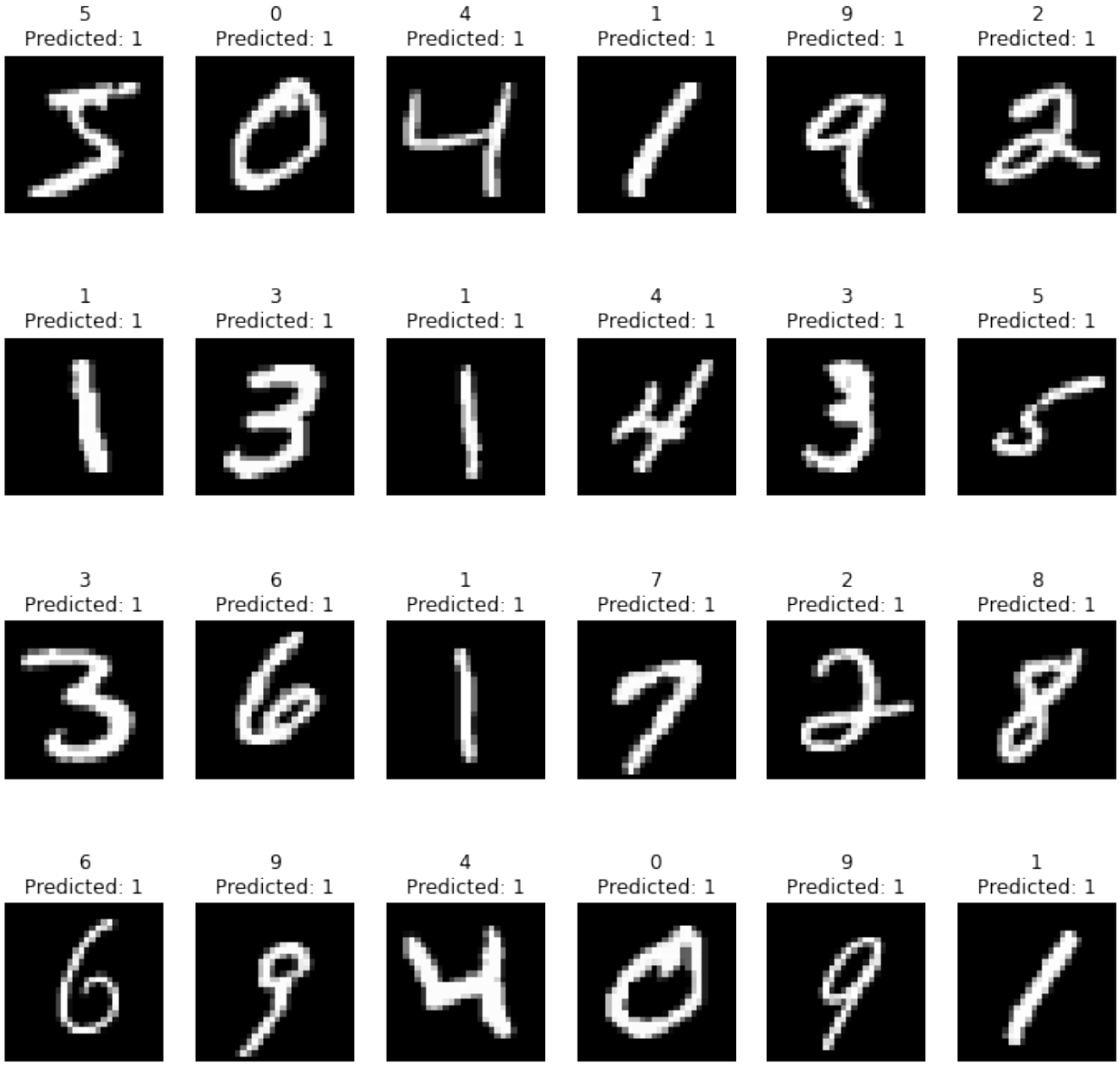
12.4.1 ... why are all predictions = 1?

```
plt.hist(label[:24], np.arange(11)); plt.title('Frequency of numbers in the training␣
↪data');
```



12.4.2 Test on the images

```
fig, m_axs = plt.subplots(4, 6, figsize=(12, 12))
for i, c_ax in enumerate(m_axs.flatten()):
    c_ax.imshow(img[i], cmap='gray')
    c_ax.set_title('{}\nPredicted: {}'.format(label[i], dc.predict(img[i])[0]))
    c_ax.axis('off')
```



12.5 Nearest Neighbor

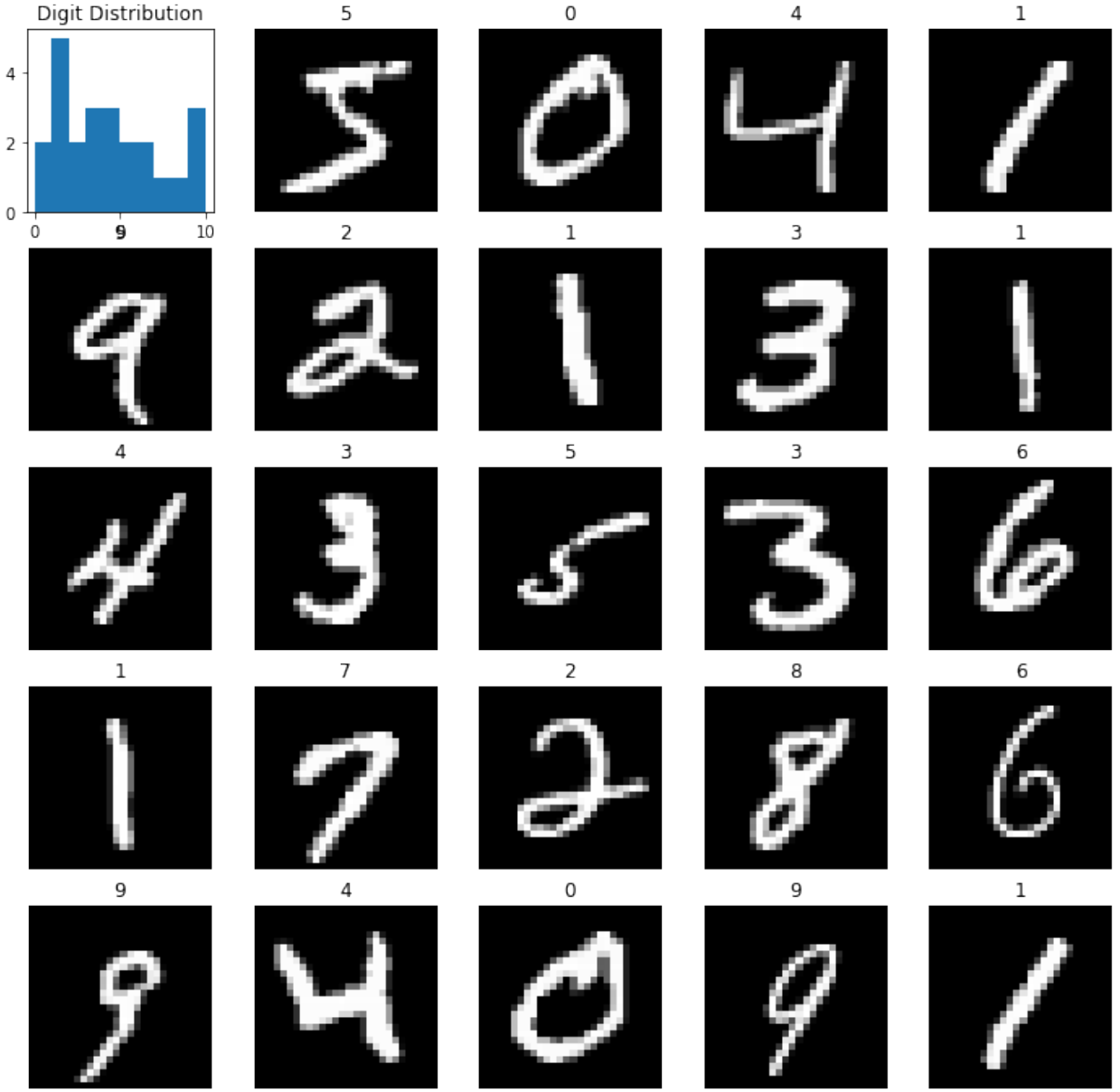
This isn't a machine learning class and so we won't dive deeply into other methods, but nearest neighbor is often a very good baseline (that is also very easy to understand). You basically take the element from the original set that is closest to the image you show.

Figure from J. Russ, Image Processing Handbook

You can make the method more robust by using more than one nearest neighbor (hence K nearest neighbors), but that we will cover in the supervised methods lecture

12.5.1 Let's load the data again...

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from skimage.util import montage as montage2d
%matplotlib inline
(img, label), _ = mnist.load_data()
fig, m_axs = plt.subplots(5, 5, figsize=(12, 12))
m_axs[0, 0].hist(label[:24], np.arange(11))
m_axs[0, 0].set_title('Digit Distribution')
for i, c_ax in enumerate(m_axs.flatten()[1:]):
    c_ax.imshow(img[i], cmap='gray')
    c_ax.set_title(label[i])
    c_ax.axis('off')
```



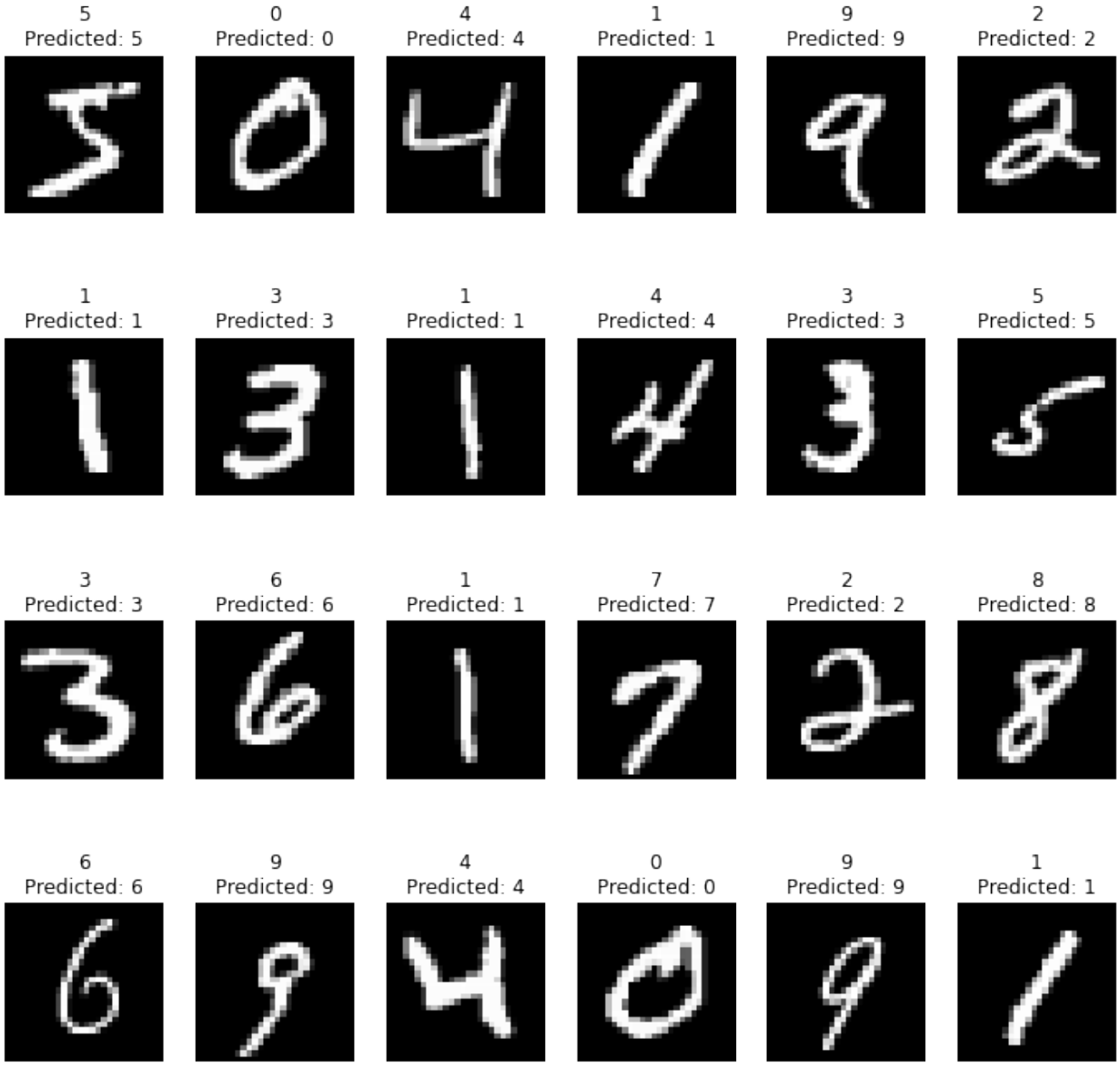
```
from sklearn.neighbors import KNeighborsClassifier
neigh_class = KNeighborsClassifier(n_neighbors=1)
neigh_class.fit(img[:24].reshape((24, -1)), label[:24])
```

```
KNeighborsClassifier(n_neighbors=1)
```

```
# predict on a few images
neigh_class.predict(img[0:10].reshape((10, -1)))
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

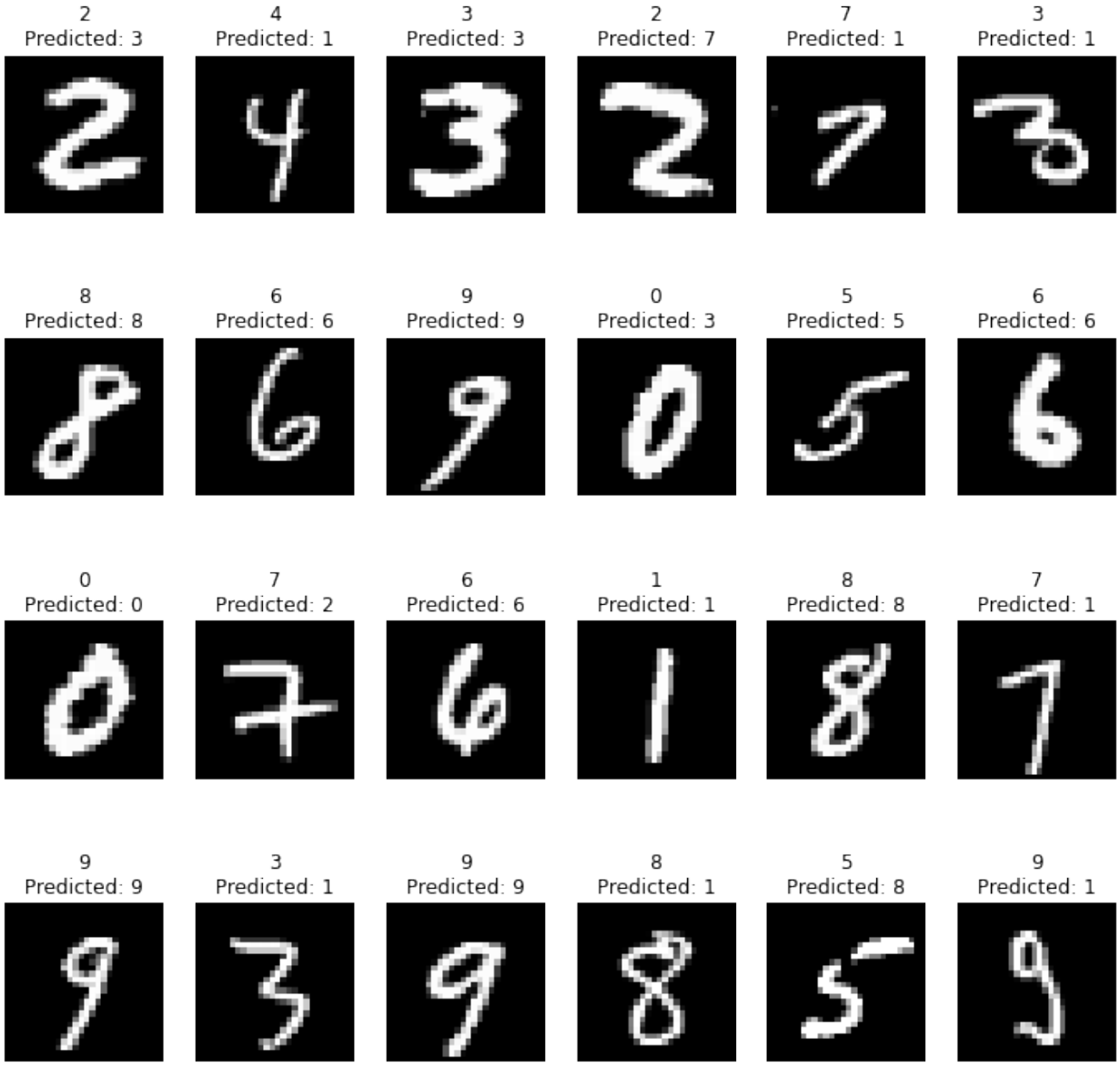
```
fig, m_axs = plt.subplots(4, 6, figsize=(12, 12))
for i, c_ax in enumerate(m_axs.flatten()):
    c_ax.imshow(img[i], cmap='gray')
    c_ax.set_title('{}\nPredicted: {}'.format(label[i],
                                              neigh_class.predict(img[i].reshape((1, -
↪1)))[0]))
    c_ax.axis('off')
```



12.5.2 100% for a baseline

Wow the model works really really well, it got every example perfectly. What we did here (a common mistake) was evaluate on the same data we 'trained' on which means the model just correctly recalled each example, if we try it on new images we can see the performance drop but still a reasonable result

```
fig, m_axs = plt.subplots(4, 6, figsize=(12, 12))
for i, c_ax in enumerate(m_axs.flatten(), 25):
    c_ax.imshow(img[i], cmap='gray')
    c_ax.set_title('{}\nPredicted: {}'.format(label[i],
                                              neigh_class.predict(img[i].reshape((1, -
↪1)))[0]))
    c_ax.axis('off')
```



12.6 How good is good?

We will cover more tools later in the class but now we will show the accuracy and the confusion matrix for our simple baseline model to evaluate how well it worked

12.6.1 Confusion Matrix

We show which cases were most frequently confused

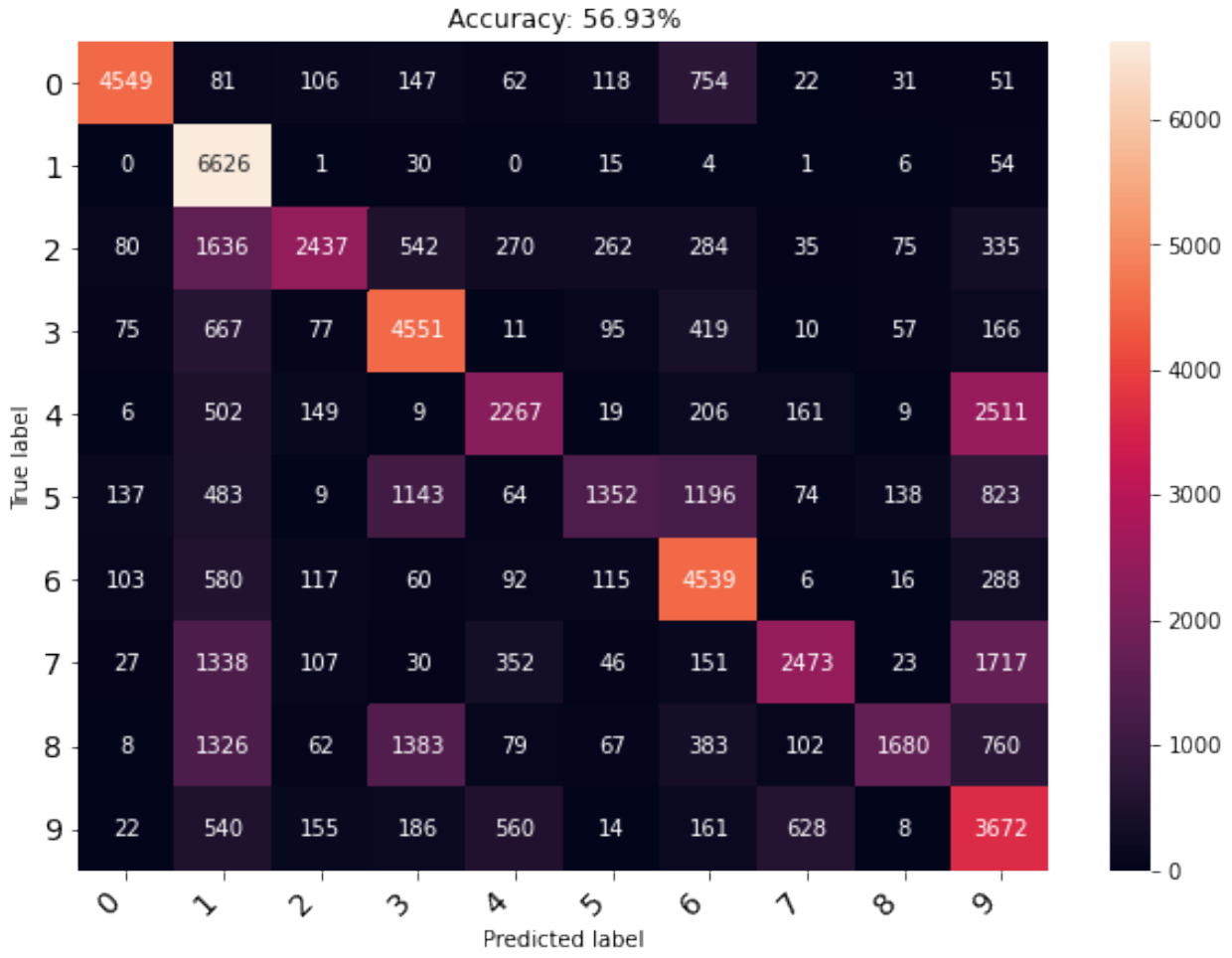
```
import seaborn as sns
import pandas as pd
def print_confusion_matrix(confusion_matrix, class_names, figsize = (10,7),␣
↪fontsize=14):
    """Prints a confusion matrix, as returned by sklearn.metrics.confusion_matrix, as␣
↪a heatmap.

    Stolen from: https://gist.github.com/shaypal5/94c53d765083101efc0240d776a23823

    Arguments
    ---------
    confusion_matrix: numpy.ndarray
        The numpy.ndarray object returned from a call to sklearn.metrics.confusion_
↪matrix.
        Similarly constructed ndarrays can also be used.
    class_names: list
        An ordered list of class names, in the order they index the given confusion␣
↪matrix.
    figsize: tuple
        A 2-long tuple, the first value determining the horizontal size of the␣
↪ouputted figure,
        the second determining the vertical size. Defaults to (10,7).
    fontsize: int
        Font size for axes labels. Defaults to 14.

    Returns
    -------
    matplotlib.figure.Figure
        The resulting confusion matrix figure
    """
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig, ax1 = plt.subplots(1, 1, figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right
↪', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha=
↪'right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    return ax1
```

```
from sklearn.metrics import accuracy_score, confusion_matrix
pred_values = neigh_class.predict(img[24:].reshape((-1, 28*28)))
ax1 = print_confusion_matrix(confusion_matrix(label[24:], pred_values), class_
↪names=range(10))
ax1.set_title('Accuracy: {:2.2%}'.format(accuracy_score(label[24:], pred_values)));
```


Accuracy: 56.93%
True label
Predicted label
0 4549 81 106 147 62 118 754 22 31 51
1 0 6626 1 30 0 15 4 1 6 54
2 80 1636 2437 542 270 262 284 35 75 335
3 75 667 77 4551 11 95 419 10 57 166
4 6 502 149 9 2267 19 206 161 9 2511
5 137 483 9 1143 64 1352 1196 74 138 823
6 103 580 117 60 92 115 4539 6 16 288
7 27 1338 107 30 352 46 151 2473 23 1717
8 8 1326 62 1383 79 67 383 102 1680 760
9 22 540 155 186 560 14 161 628 8 3672
0 1 2 3 4 5 6 7 8 9
6000
5000
4000
3000
2000
1000
0

CHAPTER

THIRTEEN

SUMMARY