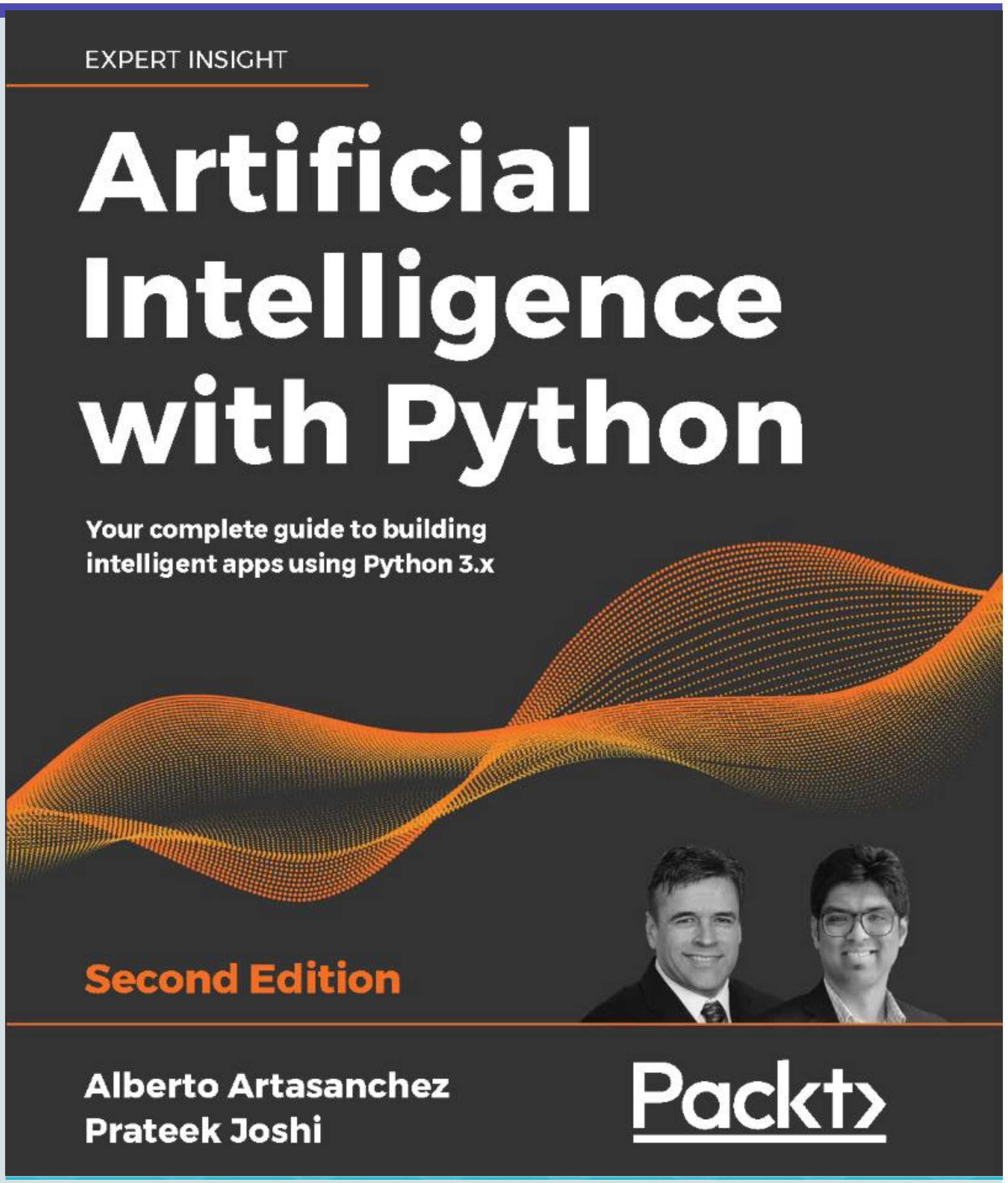
University of South-Eastern Norway
School of Business

Lecture 8: Artificial Neural Networks

Sinuo Wu

Course: AI for Business Applications (AI3000)



EXPERT INSIGHT

Artificial Intelligence with Python

Your complete guide to building intelligent apps using Python 3.x

Second Edition

Alberto Artasanchez
Prateek Joshi

Packt

University of South-Eastern Norway
School of Business

Do it before we start:

Download Data From Canvas – AI3000 - Files – Day8 Practice

Sinuo Wu

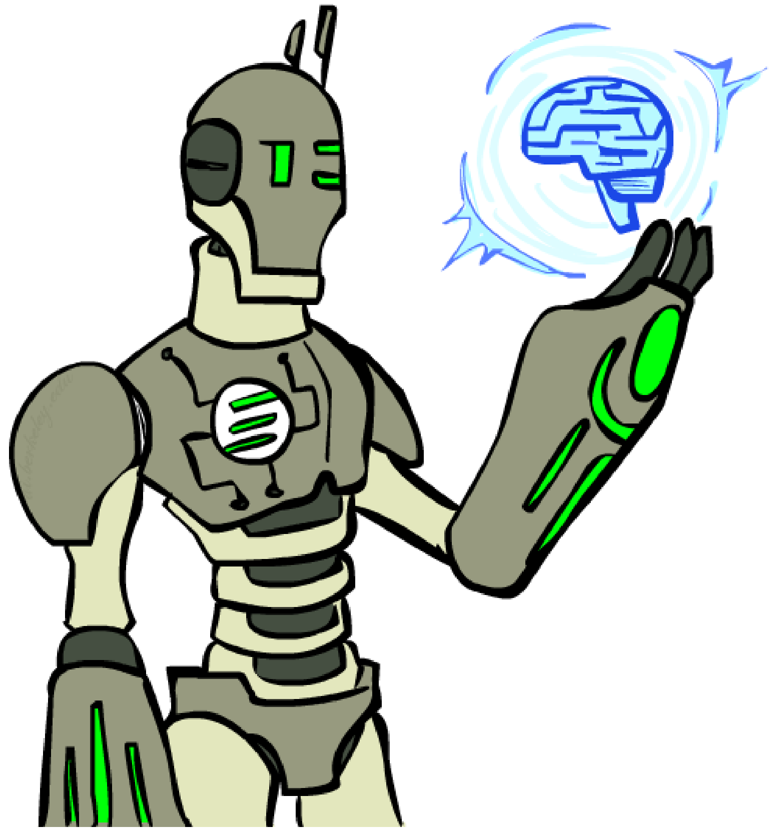
Course: AI for Business Applications (AI3000)

Quiz

- Enter room number or Scan the QR code

Today

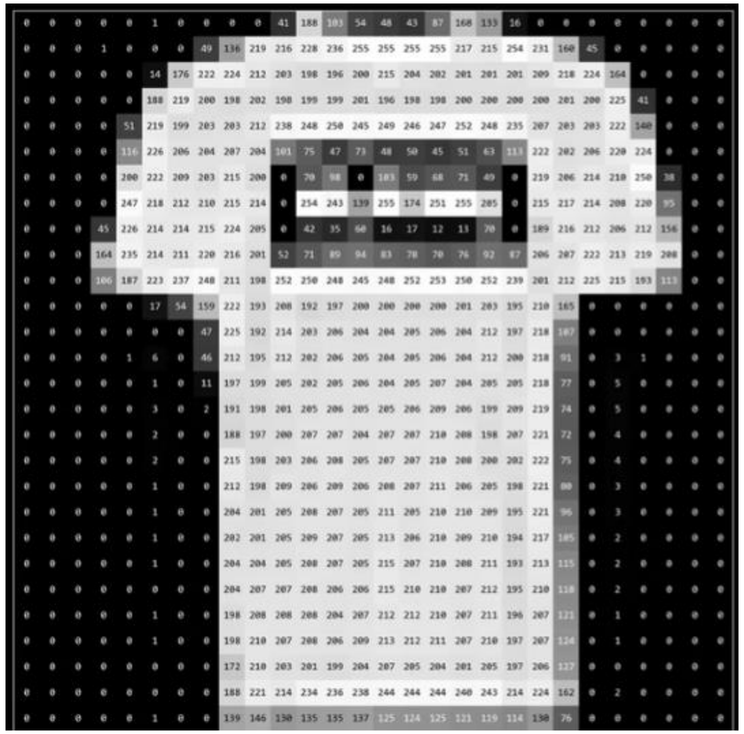
- Introduction to neural networks
- Applications of neural networks
- Case practice & Coding analysis
- Basic Libraries
- Case review (CV & NLP)



Introduction to Neural Networks

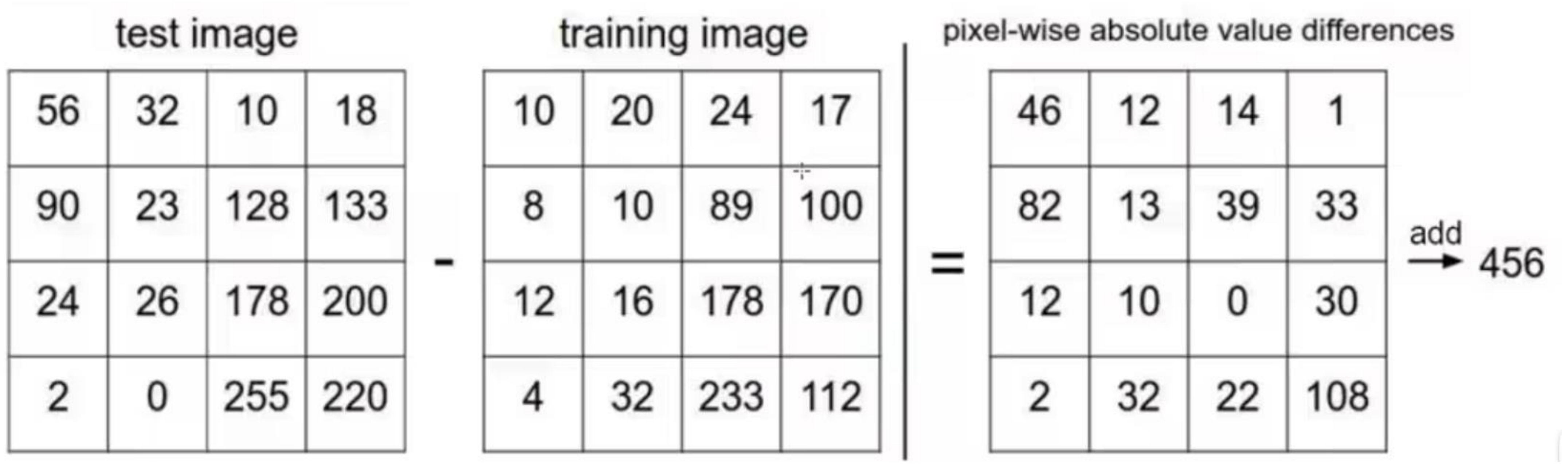
Example

Pixel Features



$$\begin{tikzcd} {} \arrow[r, "f(x)"] & 7 \arrow[r] & \text{T-Shirt} \end{tikzcd}$$

Example



test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

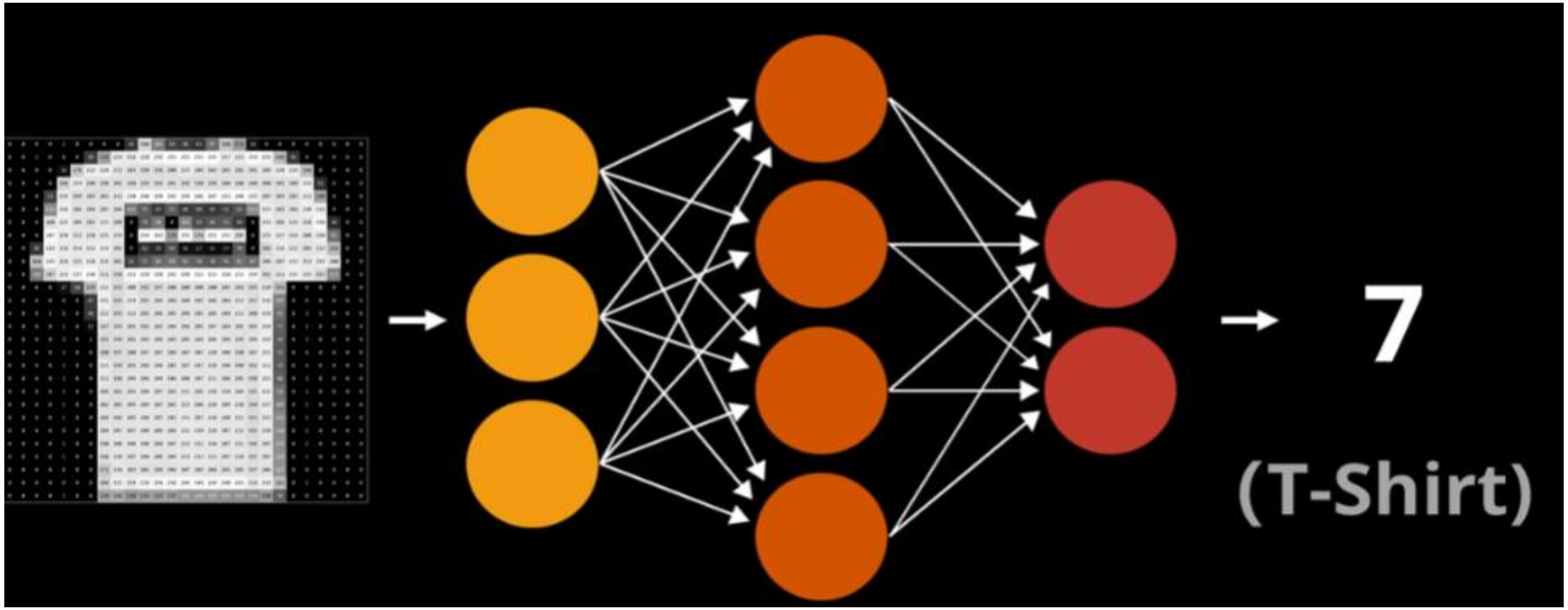
=

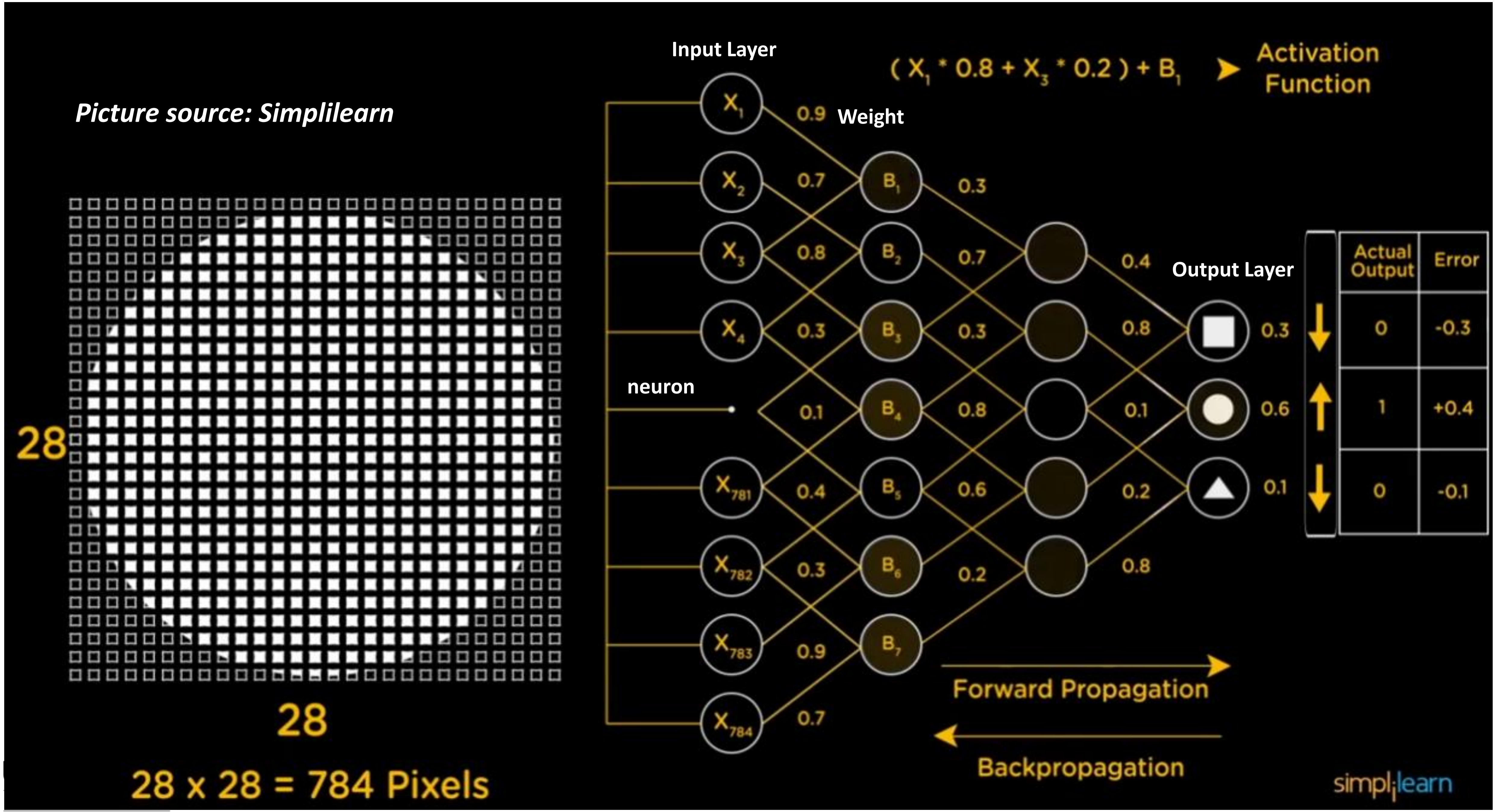
pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\xrightarrow{\text{add}}$ 456

Neural Network

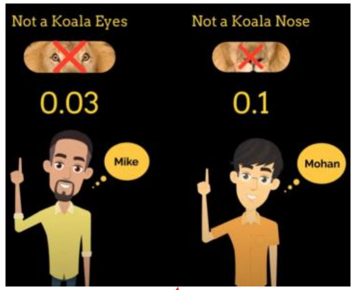


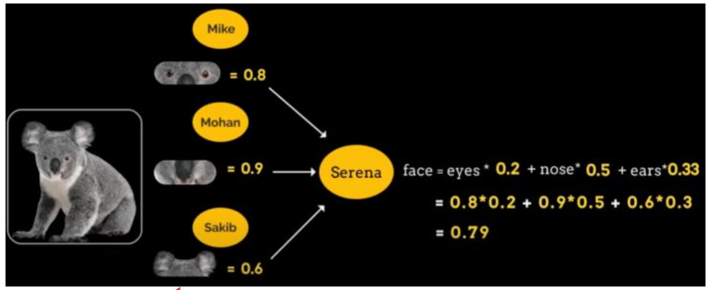


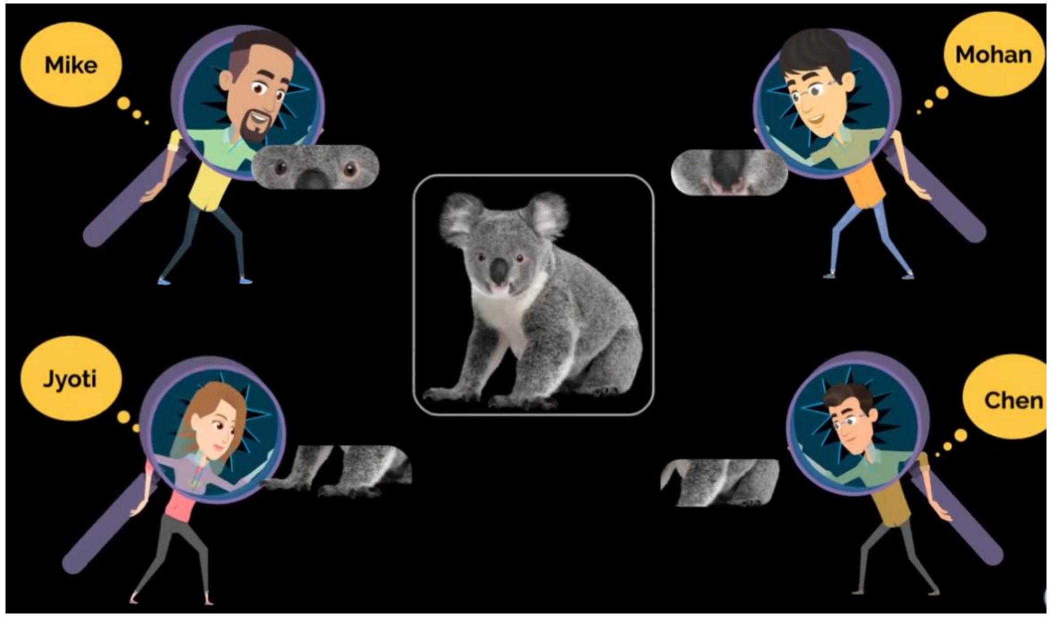
Actual Output	Error
0	-0.3
1	+0.4
0	-0.1

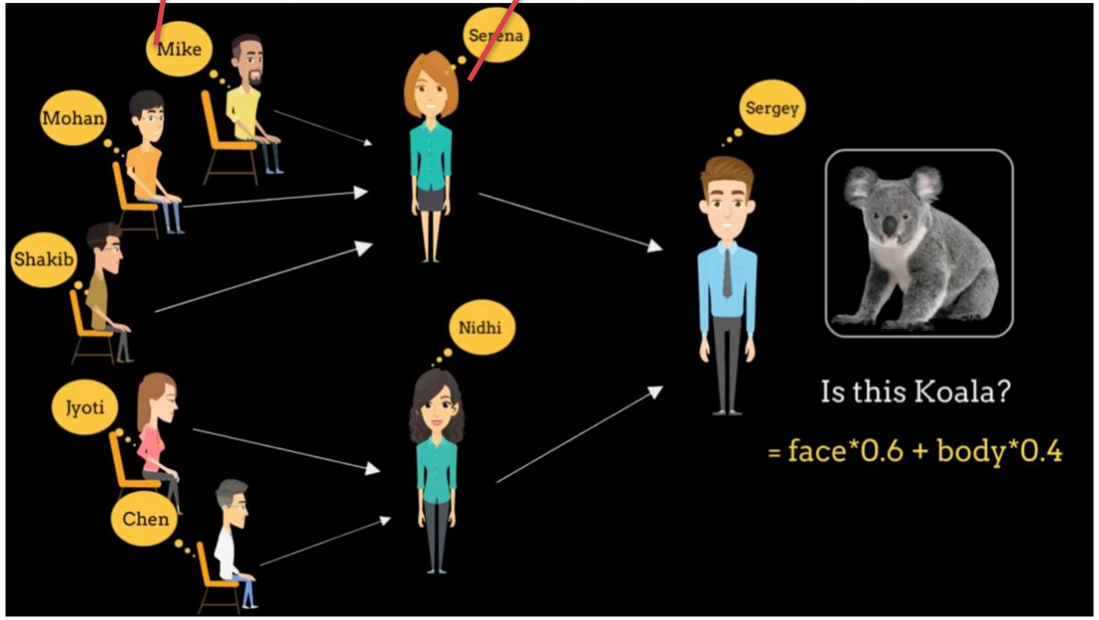
Picture source: Simplilearn

Neural Network









University of South-Eastern Norway
School of Business

Neural Network

- An Artificial Neural Network (ANN) is a computational model inspired by the structure and function of the **human brain**. It is a fundamental component of machine learning and deep learning, designed to process and learn from data to make predictions or decisions.

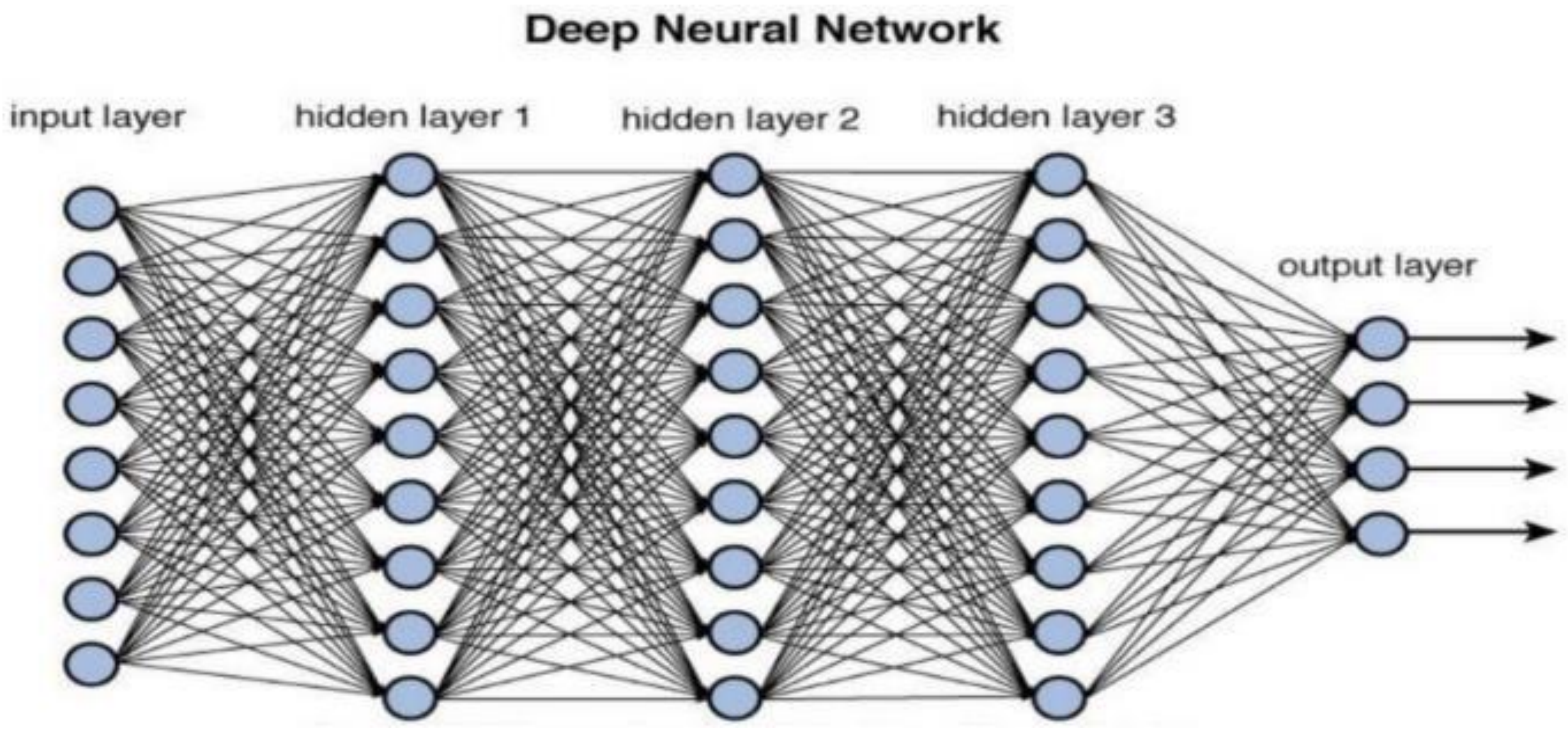
- ANNs consist of interconnected nodes, often referred to as neurons or units, organized into layers. These layers typically include an input layer, one or more hidden layers, and an output layer.

- ANNs are used in pattern recognition, regression, classification, computer vision, NLP, and deep learning, and have revolutionized machine learning.

Neural Network

- **Neurons:** Nodes that process and transmit information.
- **Layers:** Organization into input, hidden, and output layers.
- **Weights and Biases:** Adjusted during training to minimize errors.
- **Learning:** The process of adjusting weights to improve predictions.
- **Backpropagation:** Algorithm for updating weights during training.
- **Activation Functions:** Determine if neurons are activated.

Deep Neural Network



(w * x + b)

Training a Neural Network

- If we are dealing with *N*-dimensional input data, then the input layer will consist of *N* neurons.
- If we have *M* distinct classes in our training data, then the output layer will consist of *M* neurons.
- A simple neural network will consist of a couple of layers and a deep neural network will consist of many layers.

Training a Neural Network

- So how can a neural network be used to classify data?
  - The first step is to collect the appropriate training data and label it.
  - Each neuron acts as a simple function and the neural network trains itself until the error goes below a certain a threshold.
  - The error is the difference between the predicted output and the actual output.
  - Based on how big the error is, the neural network adjusts itself and retrains until it gets closer to the solution.

When to use it

- Neural networks are universal approximators, and they work best if the system you are using them to model has a high tolerance to error.
  - Capturing associations or discovering regularities within a set of patterns;
  - Where the volume, number of variables or diversity of the data is very great;
  - The relationships between variables are vaguely understood; or,
  - The relationships are difficult to describe adequately with conventional approaches.

http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

Have a Break!

University of
South-Eastern Norway
School of Business

Applications

Applications

Image and Video Analysis: Image classification, object detection, video analysis.

Natural Language Processing: Language translation, sentiment analysis, chatbots.

Speech Recognition and Synthesis: Speech-to-text, text-to-speech.

Autonomous Vehicles: Self-driving cars.

Healthcare and Medicine: Disease diagnosis, drug discovery, patient risk assessment.

Finance: Stock market prediction, credit scoring, fraud detection.

Robotics: Object manipulation, navigation.

Industrial and Manufacturing: Quality control, predictive maintenance.

Energy and Environmental Applications: Energy consumption prediction, environmental monitoring.

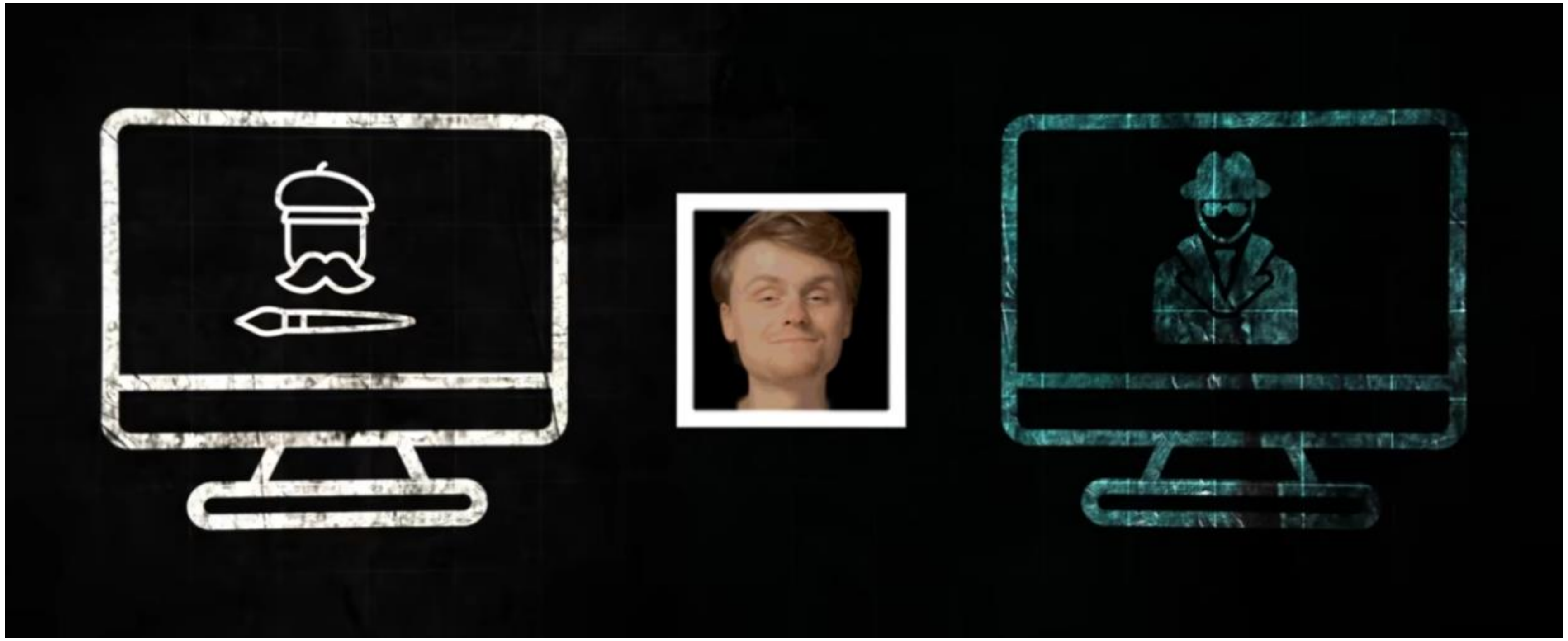
Anomaly Detection: Cybersecurity and fraud detection.

TensorFlow

Google's open-source machine learning Library.

- **Deep Learning:** Especially well-suited for deep neural networks.
- **Flexibility:** Uses a computational graph for customization.
- **High Performance:** Optimized for CPU and GPU, ideal for large datasets.
- **Community and Ecosystem:** Supported by a large community and rich tools.
- **Deployment**: Suitable for deploying models in various environments.
- **Research:** Used for cutting-edge machine learning research.
- **Open Source:** Freely available and actively maintained.

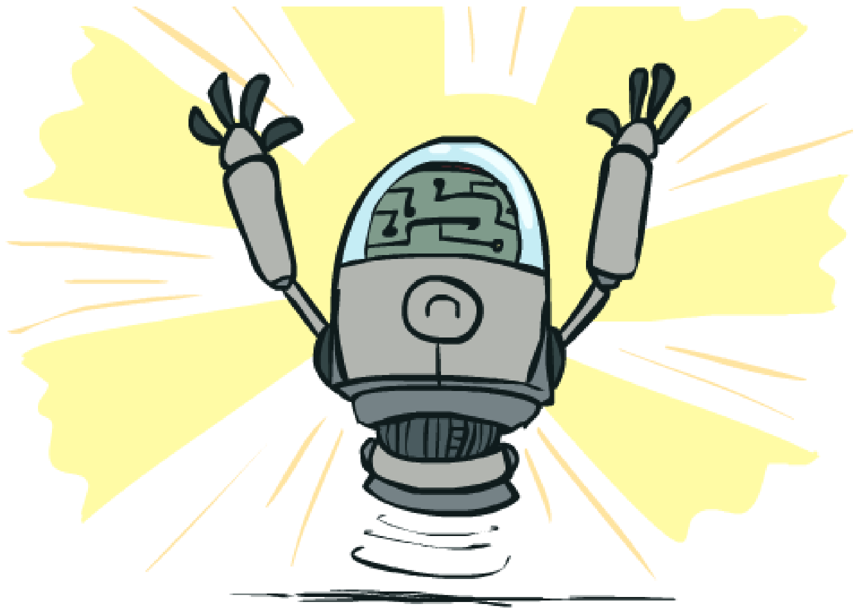
Inspiration



https://www.youtube.com/watch?v=S951cdansBI

Practice

Goal: 1. Successfully understand and run the coding
2. classify a random picture with established model



https://www.tensorflow.org/hub/tutorials/image_feature_vector

Libraries

Collections: Offers additional data structures like dictionaries, lists, and sets, along with specialized types.

io: Provides classes for input and output operations, commonly used for working with file-like objects and streams.

math: Includes mathematical functions and constants for tasks like trigonometric operations and using constants like pi and e.

Os: Facilitates interaction with the operating system, offering functions for file and directory manipulation, handling environment variables, and managing processes.

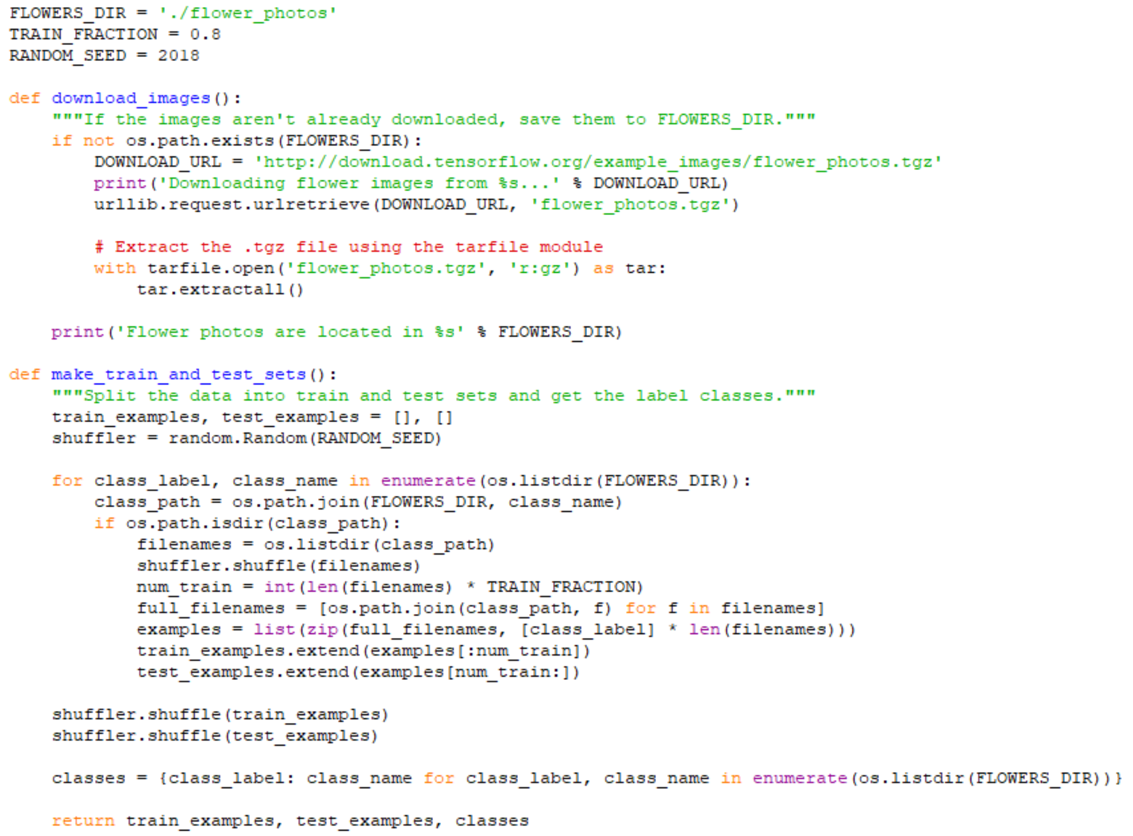
Random: Used for generating random numbers, often applied in simulations, games, and applications requiring randomness.

Six: Handles differences between different versions of Python.

Urllib: A module for working with URLs, including internet data fetching and downloading files from the web.

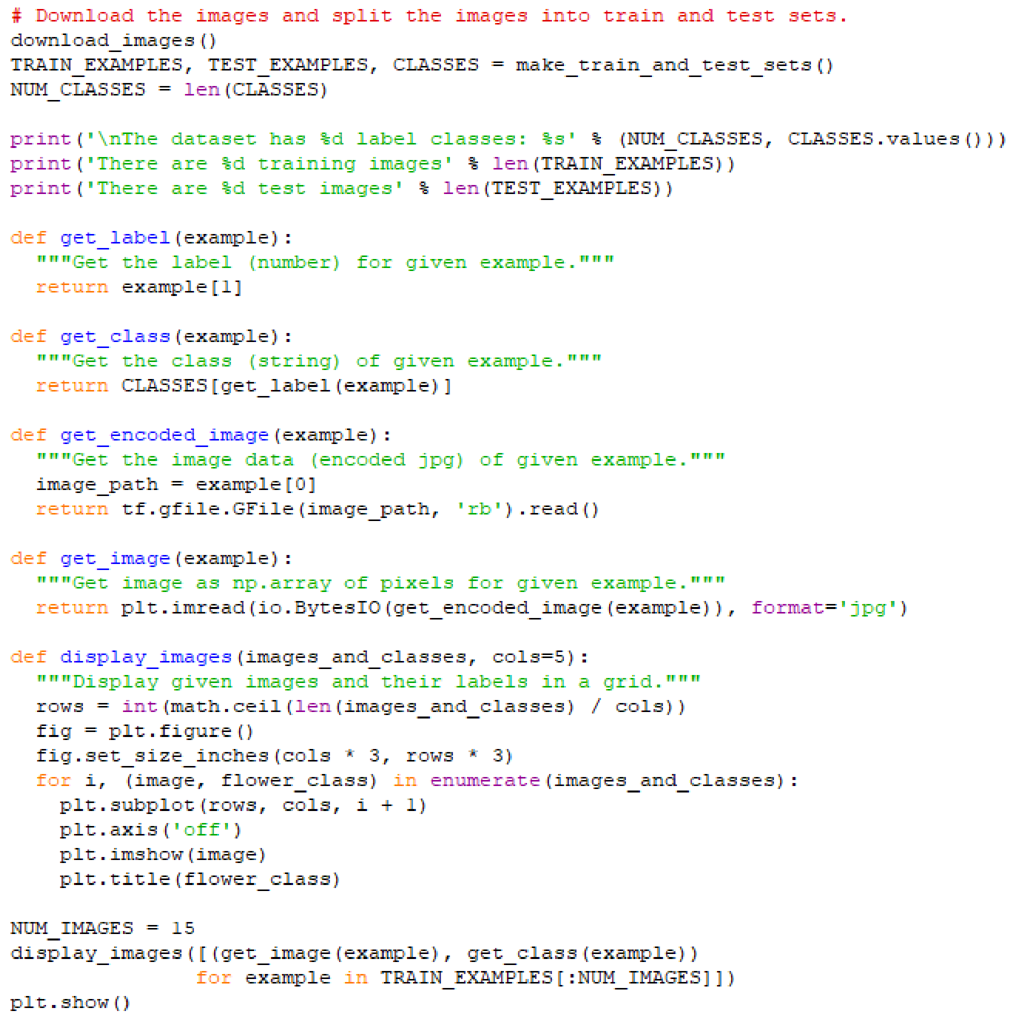
Tarfile: Enables working with tar archives, a common compressed file format in Unix and Linux, for extracting archive files.

Coding



```python
FLOWERS_DIR = './flower_photos'
TRAIN_FRACTION = 0.8
RANDOM_SEED = 2018

def download_images():
    """If the images aren't already downloaded, save them to FLOWERS_DIR."""
    if not os.path.exists(FLOWERS_DIR):
        DOWNLOAD_URL = 'http://download.tensorflow.org/example_images/flower_photos.tgz'
        print('Downloading flower images from %s...' % DOWNLOAD_URL)
        urllib.request.urlretrieve(DOWNLOAD_URL, 'flower_photos.tgz')

        # Extract the .tgz file using the tarfile module
        with tarfile.open('flower_photos.tgz', 'r:gz') as tar:
            tar.extractall()

    print('Flower photos are located in %s' % FLOWERS_DIR)

def make_train_and_test_sets():
    """Split the data into train and test sets and get the label classes."""
    train_examples, test_examples = [], []
    shuffler = random.Random(RANDOM_SEED)

    for class_label, class_name in enumerate(os.listdir(FLOWERS_DIR)):
        class_path = os.path.join(FLOWERS_DIR, class_name)
        if os.path.isdir(class_path):
            filenames = os.listdir(class_path)
            shuffler.shuffle(filenames)
            num_train = int(len(filenames) * TRAIN_FRACTION)
            full_filenames = [os.path.join(class_path, f) for f in filenames]
            examples = list(zip(full_filenames, [class_label] * len(filenames)))
            train_examples.extend(examples[:num_train])
            test_examples.extend(examples[num_train:])

    shuffler.shuffle(train_examples)
    shuffler.shuffle(test_examples)

    classes = {class_label: class_name for class_label, class_name in enumerate(os.listdir(FLOWERS_DIR))}

    return train_examples, test_examples, classes
```

Load and split the data



```python
# Download the images and split the images into train and test sets.
download_images()
TRAIN_EXAMPLES, TEST_EXAMPLES, CLASSES = make_train_and_test_sets()
NUM_CLASSES = len(CLASSES)

print('\nThe dataset has %d label classes: %s' % (NUM_CLASSES, CLASSES.values()))
print('There are %d training images' % len(TRAIN_EXAMPLES))
print('There are %d test images' % len(TEST_EXAMPLES))

def get_label(example):
  """Get the label (number) for given example."""
  return example[1]

def get_class(example):
  """Get the class (string) of given example."""
  return CLASSES[get_label(example)]

def get_encoded_image(example):
  """Get the image data (encoded jpg) of given example."""
  image_path = example[0]
  return tf.gfile.GFile(image_path, 'rb').read()

def get_image(example):
  """Get image as np.array of pixels for given example."""
  return plt.imread(io.BytesIO(get_encoded_image(example)), format='jpg')

def display_images(images_and_classes, cols=5):
  """Display given images and their labels in a grid."""
  rows = int(math.ceil(len(images_and_classes) / cols))
  fig = plt.figure()
  fig.set_size_inches(cols * 3, rows * 3)
  for i, (image, flower_class) in enumerate(images_and_classes):
    plt.subplot(rows, cols, i + 1)
    plt.axis('off')
    plt.imshow(image)
    plt.title(flower_class)

NUM_IMAGES = 15
display_images([(get_image(example), get_class(example))
               for example in TRAIN_EXAMPLES[:NUM_IMAGES]])
plt.show()
```

Define, test and check the images

University of South-Eastern Norway
School of Business

Coding

Set up layers and preprocessing images

```
LEARNING_RATE = 0.01

tf.reset_default_graph()

# Load a pre-trained TF-Hub module for extracting features from images. We've chosen this particular module for speed, but many other choices are available.
image_module = hub.Module('https://tfhub.dev/google/imagenet/mobilenet_v2_035_128/feature_vector/2')

# Preprocessing images into tensors with size expected by the image module.
encoded_images = tf.placeholder(tf.string, shape=[None])
image_size = hub.get_expected_image_size(image_module)


def decode_and_resize_image(encoded):
  decoded = tf.image.decode_jpeg(encoded, channels=3)
  decoded = tf.image.convert_image_dtype(decoded, tf.float32)
  return tf.image.resize_images(decoded, image_size)


batch_images = tf.map_fn(decode_and_resize_image, encoded_images, dtype=tf.float32)

# The image module can be applied as a function to extract feature vectors for a batch of images.
features = image_module(batch_images)


def create_model(features):
  """Build a model for classification from extracted features."""
  # Currently, the model is just a single linear layer. You can try to add another layer, but be careful... two linear layers (when activation=None) are equivalent to a single linear layer. You can create a nonlinear layer like this:
  # layer = tf.layers.dense(inputs=..., units=..., activation=tf.nn.relu)
  layer = tf.layers.dense(inputs=features, units=NUM_CLASSES, activation=None)
  return layer


# For each class (kind of flower), the model outputs some real number as a score how much the input resembles this class. This vector of numbers is often called the "logits".
logits = create_model(features)
labels = tf.placeholder(tf.float32, [None, NUM_CLASSES])

# Mathematically, a good way to measure how much the predicted probabilities diverge from the truth is the "cross-entropy" between the two probability distributions. For numerical stability, this is best done directly from the
# logits, not the probabilities extracted from them.
cross_entropy = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits, labels=labels)
cross_entropy_mean = tf.reduce_mean(cross_entropy)

# Let's add an optimizer so we can train the network.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=LEARNING_RATE)
train_op = optimizer.minimize(loss=cross_entropy_mean)

# The "softmax" function transforms the logits vector into a vector of probabilities: non-negative numbers that sum up to one, and the i-th number says how likely the input comes from class i.
probabilities = tf.nn.softmax(logits)

# We choose the highest one as the predicted class.
prediction = tf.argmax(probabilities, 1)
correct_prediction = tf.equal(prediction, tf.argmax(labels, 1))

# The accuracy will allow us to eval on our test set.
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
# How long will we train the network (number of batches).
NUM_TRAIN_STEPS = 100
# How many training examples we use in each step.
TRAIN_BATCH_SIZE = 10
# How often to evaluate the model performance.
EVAL_EVERY = 10
```

University of South-Eastern Norway
School of Business

Coding

```python
def get_batch(batch_size=None, test=False):
  """Get a random batch of examples."""
  examples = TEST_EXAMPLES if test else TRAIN_EXAMPLES
  batch_examples = random.sample(examples, batch_size) if batch_size else examples
  return batch_examples

def get_images_and_labels(batch_examples):
  images = [get_encoded_image(e) for e in batch_examples]
  one_hot_labels = [get_label_one_hot(e) for e in batch_examples]
  return images, one_hot_labels

def get_label_one_hot(example):
  """Get the one hot encoding vector for the example."""
  one_hot_vector = np.zeros(NUM_CLASSES)
  np.put(one_hot_vector, get_label(example), 1)
  return one_hot_vector
```

Model Training

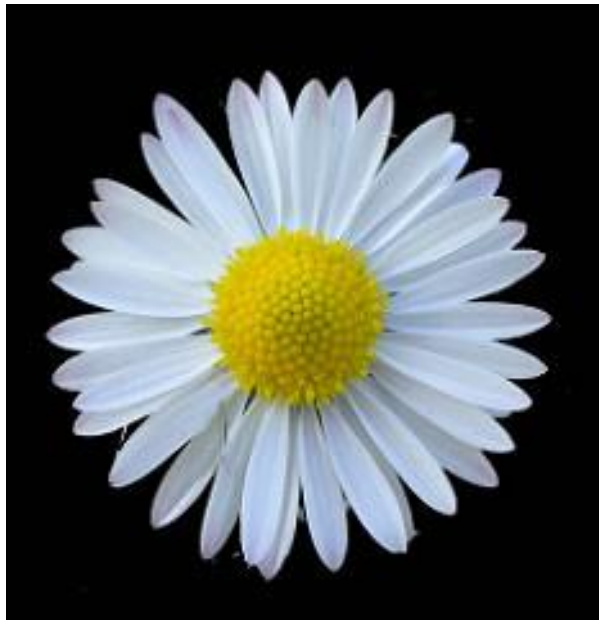
```python
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(NUM_TRAIN_STEPS):
    # Get a random batch of training examples.
    train_batch = get_batch(batch_size=TRAIN_BATCH_SIZE)
    batch_images, batch_labels = get_images_and_labels(train_batch)
    # Run the train_op to train the model.
    train_loss, _, train_accuracy = sess.run(
        [cross_entropy_mean, train_op, accuracy],
        feed_dict={encoded_images: batch_images, labels: batch_labels})
    is_final_step = (i == (NUM_TRAIN_STEPS - 1))
    if i % EVAL_EVERY == 0 or is_final_step:
      # Get a batch of test examples.
      test_batch = get_batch(batch_size=None, test=True)
      batch_images, batch_labels = get_images_and_labels(test_batch)
      # Evaluate how well our model performs on the test set.
      test_loss, test_accuracy, test_prediction, correct_predicate = sess.run(
        [cross_entropy_mean, accuracy, prediction, correct_prediction],
        feed_dict={encoded_images: batch_images, labels: batch_labels})
      print('Test accuracy at step %s: %.2f%%' % (i, (test_accuracy * 100)))
```

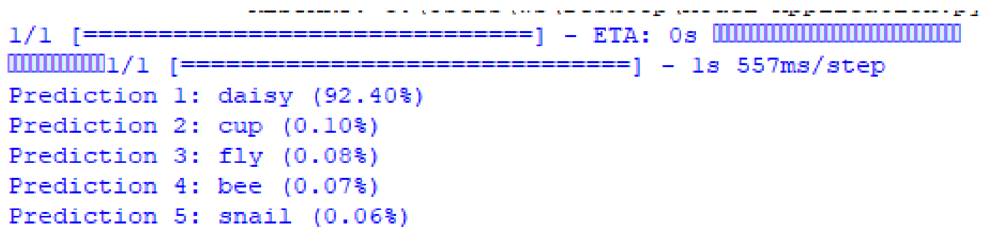
Evaluation

```python
def show_confusion_matrix(test_labels, predictions):
  """Compute confusion matrix and normalize."""
  confusion = sk_metrics.confusion_matrix(
    np.argmax(test_labels, axis=1), predictions)
  confusion_normalized = confusion.astype("float") / confusion.sum(axis=1)
  axis_labels = list(CLASSES.values())
  ax = sns.heatmap(
      confusion_normalized, xticklabels=axis_labels, yticklabels=axis_labels,
      cmap='Blues', annot=True, fmt='.2f', square=True)
  plt.title("Confusion matrix")
  plt.ylabel("True label")
  plt.xlabel("Predicted label")

show_confusion_matrix(batch_labels, test_prediction)
plt.show()
```

Apply model

```python
import tensorflow as tf
import numpy as np
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

# Use your model instead of the pre-defined model
model = MobileNetV2(weights='imagenet')

# Function to load and preprocess an image from a file path
def load_and_preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    return img_array

# Inference on new images
new_image_path = 'C:\\Users\\ws\\Desktop\\Test.jpg'  # Replace with the path to your new image

# Load and preprocess the new image
new_image = load_and_preprocess_image(new_image_path)

# Get predictions for the new image
predictions = model.predict(new_image)
decoded_predictions = decode_predictions(predictions, top=5)[0]

# Display the top 5 predicted classes
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print("Prediction {}: {} ({:.2f}%)".format(i + 1, label, score * 100))

# Display the image
img = image.load_img(new_image_path)
plt.imshow(img)
plt.axis('off')
plt.show()
```





```
[illegible]
1/1 [==============================] - ETA: 0s ▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯
▯▯▯▯▯▯▯▯▯▯▯▯▯1/1 [==============================] - 1s 557ms/step
Prediction 1: daisy (92.40%)
Prediction 2: cup (0.10%)
Prediction 3: fly (0.08%)
Prediction 4: bee (0.07%)
Prediction 5: snail (0.06%)
```

University of South-Eastern Norway
School of Business

How to save a TensorFlow model

- **Save Model During Training (Checkpointing)**: Save the model periodically during training to create checkpoints that allow you to resume training or later select the best model.

- **Save the Final Model**: Once training is complete, and you're satisfied with your model's performance. Use the same “saver” object for this purpose.

- **Export Model for Inference**: To use the model for inference (making predictions on new data), you can export it in a format that can be loaded by TensorFlow Serving, TensorFlow Lite, or other deployment options.

- **Load and Use the Model for Inference**: To load the model for inference, you can use TensorFlow Serving, TensorFlow Lite, or the TensorFlow Python API, depending on your deployment scenario.

Home Practice

Try to mimic the coding for:

1. Classify fire
2. Classify mushroom

Have a Break!

Types of Neural Networks

Basic Types

- **Feedforward Neural Networks (FNNs):**

Structure: FNNs consist of input, hidden, and output layers. Information flows in one direction, from input to output.

Use: Typically used for tasks like classification, regression, and pattern recognition. -- **Complex patterns**

- **Convolutional Neural Networks (CNNs):**

Structure: CNNs are specialized for processing grid-like data, such as images or 2D signals. They consist of convolutional layers, pooling layers, and fully connected layers.

Use: Excellent for image classification, object detection, and feature extraction.– **Computer vision**

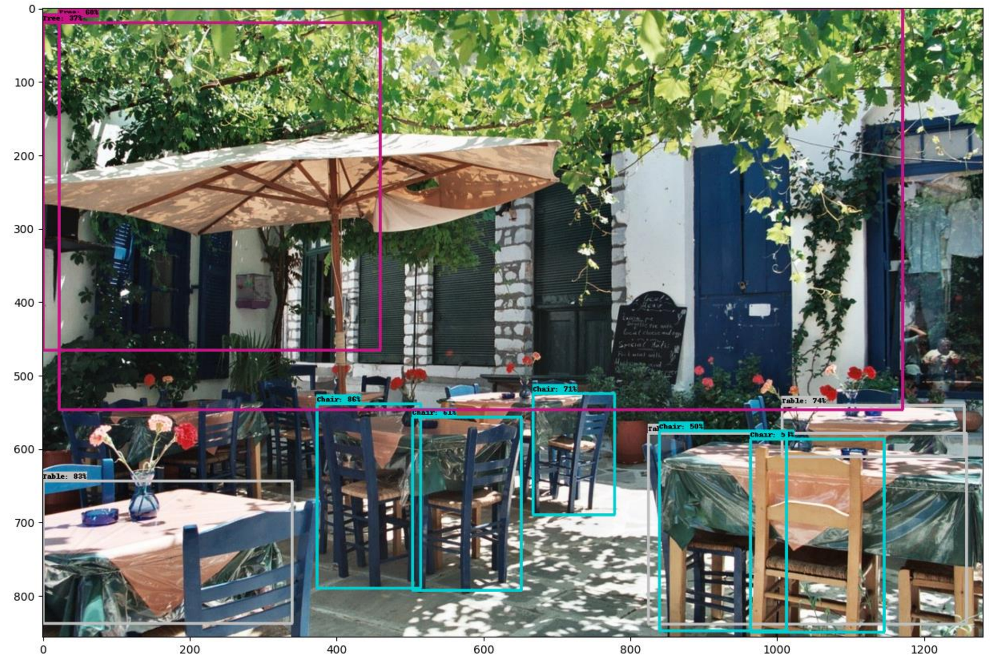
- **Recurrent Neural Networks (RNNs):**

Structure: RNNs process sequential data by maintaining hidden states that capture information from previous time steps. They can have one or more layers.

Use: Suited for tasks involving sequences. – **Natural language processing**

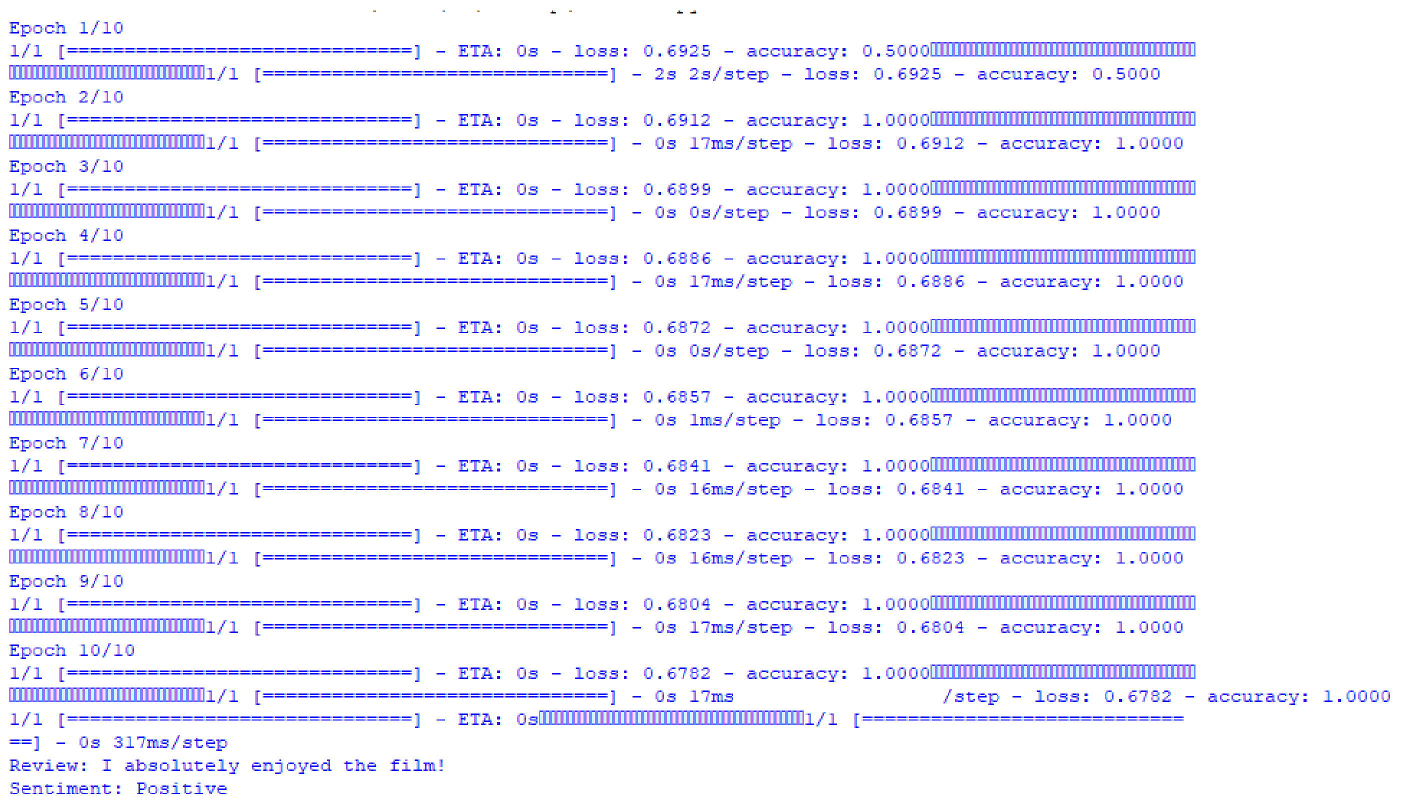
CNN Case – Object Detection

https://www.tensorflow.org/hub/tutorials/object_detection



TensorFlow - Tutorials - Object Detection

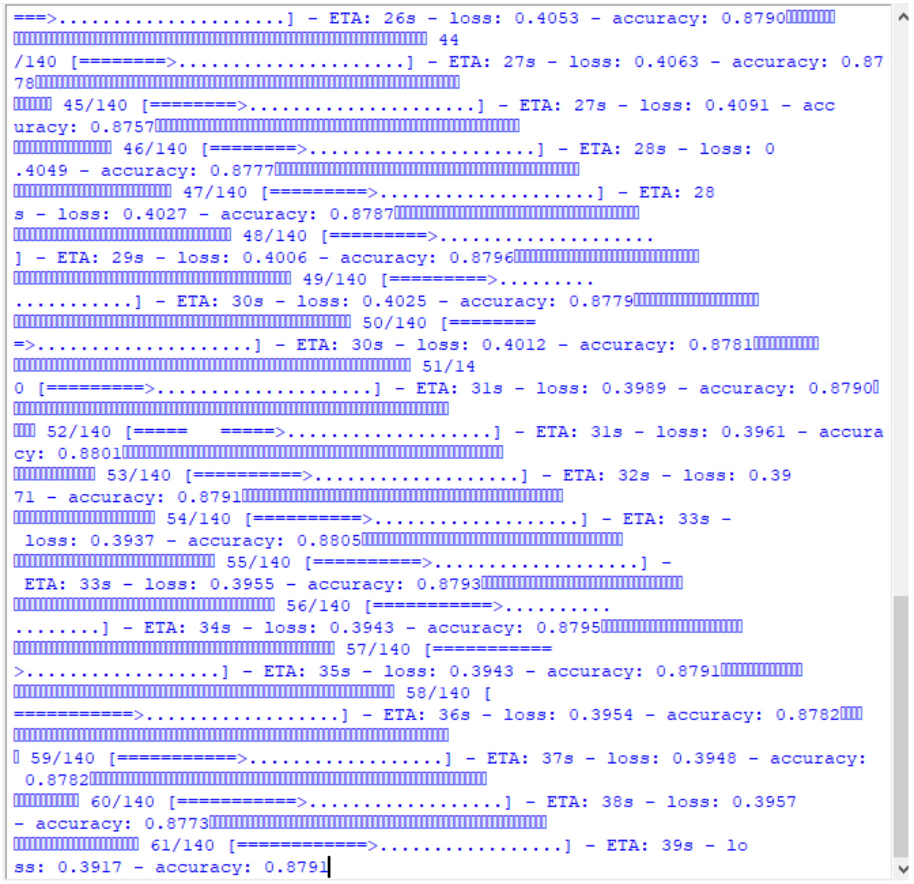
RNN Case – Natural Language Processing



```
Epoch 1/10
1/1 [==============================] - ETA: 0s - loss: 0.6925 - accuracy: 0.5000
1/1 [==============================] - 2s 2s/step - loss: 0.6925 - accuracy: 0.5000
Epoch 2/10
1/1 [==============================] - ETA: 0s - loss: 0.6912 - accuracy: 1.0000
1/1 [==============================] - 0s 17ms/step - loss: 0.6912 - accuracy: 1.0000
Epoch 3/10
1/1 [==============================] - ETA: 0s - loss: 0.6899 - accuracy: 1.0000
1/1 [==============================] - 0s 0s/step - loss: 0.6899 - accuracy: 1.0000
Epoch 4/10
1/1 [==============================] - ETA: 0s - loss: 0.6886 - accuracy: 1.0000
1/1 [==============================] - 0s 17ms/step - loss: 0.6886 - accuracy: 1.0000
Epoch 5/10
1/1 [==============================] - ETA: 0s - loss: 0.6872 - accuracy: 1.0000
1/1 [==============================] - 0s 0s/step - loss: 0.6872 - accuracy: 1.0000
Epoch 6/10
1/1 [==============================] - ETA: 0s - loss: 0.6857 - accuracy: 1.0000
1/1 [==============================] - 0s 1ms/step - loss: 0.6857 - accuracy: 1.0000
Epoch 7/10
1/1 [==============================] - ETA: 0s - loss: 0.6841 - accuracy: 1.0000
1/1 [==============================] - 0s 16ms/step - loss: 0.6841 - accuracy: 1.0000
Epoch 8/10
1/1 [==============================] - ETA: 0s - loss: 0.6823 - accuracy: 1.0000
1/1 [==============================] - 0s 16ms/step - loss: 0.6823 - accuracy: 1.0000
Epoch 9/10
1/1 [==============================] - ETA: 0s - loss: 0.6804 - accuracy: 1.0000
1/1 [==============================] - 0s 17ms/step - loss: 0.6804 - accuracy: 1.0000
Epoch 10/10
1/1 [==============================] - ETA: 0s - loss: 0.6782 - accuracy: 1.0000
1/1 [==============================] - 0s 17ms                  /step - loss: 0.6782 - accuracy: 1.0000
1/1 [==============================] - ETA: 0s1/1 [============================
==] - 0s 317ms/step
Review: I absolutely enjoyed the film!
Sentiment: Positive
```

University of South-Eastern Norway
School of Business

Review NLP Practice



```
===>.....................] - ETA: 26s - loss: 0.4053 - accuracy: 0.8790
 44
/140 [========>.....................] - ETA: 27s - loss: 0.4063 - accuracy: 0.87
78
 45/140 [========>.....................] - ETA: 27s - loss: 0.4091 - acc
uracy: 0.8757
 46/140 [========>.....................] - ETA: 28s - loss: 0
.4049 - accuracy: 0.8777
 47/140 [=========>....................] - ETA: 28
s - loss: 0.4027 - accuracy: 0.8787
 48/140 [=========>....................
] - ETA: 29s - loss: 0.4006 - accuracy: 0.8796
 49/140 [=========>.........
...........] - ETA: 30s - loss: 0.4025 - accuracy: 0.8779
 50/140 [========
=>....................] - ETA: 30s - loss: 0.4012 - accuracy: 0.8781
 51/14
0 [=========>....................] - ETA: 31s - loss: 0.3989 - accuracy: 0.8790

 52/140 [=====   =====>...................] - ETA: 31s - loss: 0.3961 - accura
cy: 0.8801
 53/140 [==========>...................] - ETA: 32s - loss: 0.39
71 - accuracy: 0.8791
 54/140 [==========>...................] - ETA: 33s -
 loss: 0.3937 - accuracy: 0.8805
 55/140 [==========>...................] -
 ETA: 33s - loss: 0.3955 - accuracy: 0.8793
 56/140 [===========>.........
........] - ETA: 34s - loss: 0.3943 - accuracy: 0.8795
 57/140 [===========
>..................] - ETA: 35s - loss: 0.3943 - accuracy: 0.8791
 58/140 [
===========>..................] - ETA: 36s - loss: 0.3954 - accuracy: 0.8782

 59/140 [===========>..................] - ETA: 37s - loss: 0.3948 - accuracy:
 0.8782
 60/140 [===========>..................] - ETA: 38s - loss: 0.3957
- accuracy: 0.8773
 61/140 [============>.................] - ETA: 39s - lo
ss: 0.3917 - accuracy: 0.8791
```

Introduction to Libraries

Libraries

NumPy: Python's numerical library for efficient array manipulation and mathematical operations.

Pandas: A data analysis library for structured data, offering DataFrames and Series. (eg. read numerical data)

Matplotlib: A data visualization library for creating charts, graphs, and plots. **(Visualization)**

NLTK (Natural Language Toolkit): A library for natural language processing and text analysis. **(NLP)**

CV2 (OpenCV): An open-source library for computer vision, used for image and video processing. **(CV)**

Sklearn(scikit-learn): A library for machine learning with tools or algorithms to build, train, and evaluate models.

Tensorflow: Google's machine learning framework for building and training models. **(NN/DL)**

Seaborn: A data visualization library that simplifies creating informative statistical graphics.

SkLearn

A Python machine learning library. Key points:

- **Machine Learning:** Used for building and training machine learning models.
- **Versatility:** Offers a wide range of machine learning algorithms and tools.
- **Data Preprocessing:** Provides features for data preprocessing and model evaluation.
- **Community:** Supported by an active community and widely used in data science.
- **Dataset:** Include dataset such as Iris, Breast Cancer, Boston housing, etc.
- **Open Source:** Open-source and freely available for machine learning projects.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

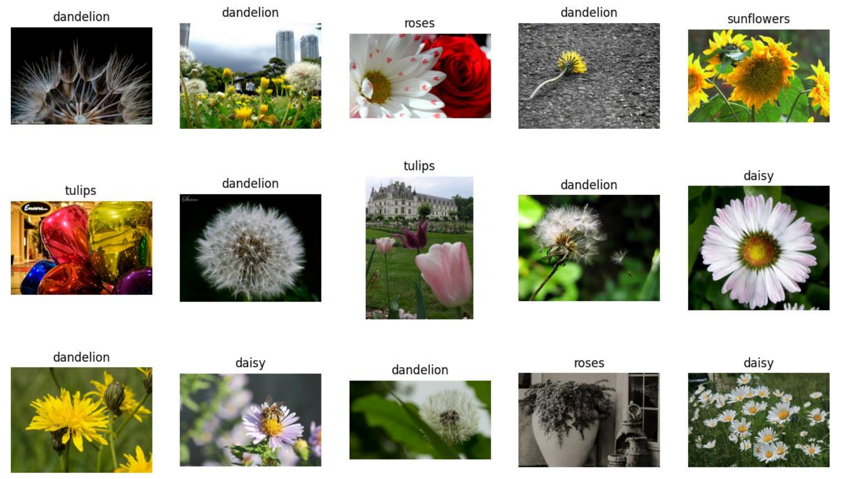
```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

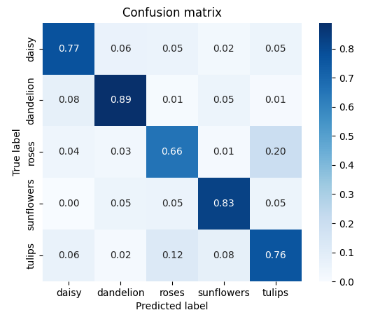
Matplotlib

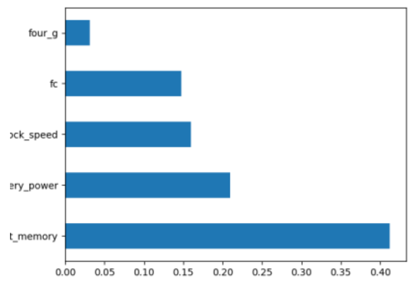
Matplotlib: A Python data visualization library.

- Data Visualization: Used to create charts, plots, and graphs.
- Customization: Highly customizable for creating various visualizations.
- Publication Quality: Suitable for producing publication-ready visualizations.
- Wide Adoption: Widely used in scientific and data analysis fields.
- Pythonic: Integrated well with other Python libraries like NumPy and Pandas.

Plt (show)







University of South-Eastern Norway
School of Business

NLTK

NLTK (Natural Language Toolkit): A Python library for natural language processing (NLP) and text analysis.

- **NLP Tools:** Provides a wide range of tools for working with human language data.
- **Text Analysis:** Used for text processing, classification, sentiment analysis, and more.
- **Research and Education:** Widely used in academia and industry for NLP research and education.
- **Community Support:** Has an active community and rich resources for NLP tasks.
- **Open Source:** Open-source and freely available for NLP projects.

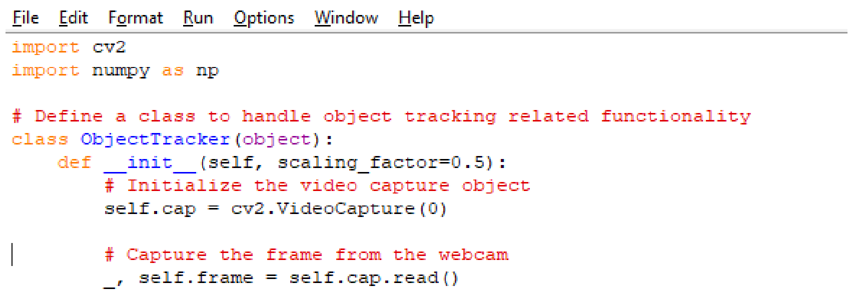
```python
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
```

```python
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy as nltk_accuracy
import nltk
```

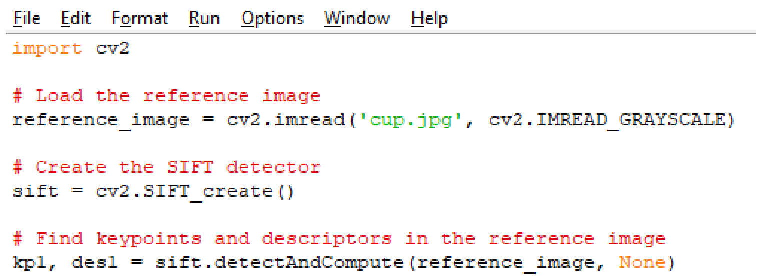
CV2

An open-source computer vision library.

- **Image and Video Processing:** Used for tasks like object detection, facial recognition, and image manipulation.
- **Computer Vision:** Ideal for computer vision applications and projects.
- **High Performance:** Optimized for efficient image and video processing.
- **Open Source:** Open-source and widely adopted in the computer vision community.
- **Integration:** Supports integration with Python for image analysis and computer vision tasks.



```
import cv2
import numpy as np

# Define a class to handle object tracking related functionality
class ObjectTracker(object):
    def __init__(self, scaling_factor=0.5):
        # Initialize the video capture object
        self.cap = cv2.VideoCapture(0)

        # Capture the frame from the webcam
        _, self.frame = self.cap.read()
```



```
import cv2

# Load the reference image
reference_image = cv2.imread('cup.jpg', cv2.IMREAD_GRAYSCALE)

# Create the SIFT detector
sift = cv2.SIFT_create()

# Find keypoints and descriptors in the reference image
kp1, des1 = sift.detectAndCompute(reference_image, None)
```

Joblib

A Python library that provides tools for saving and loading Python objects.

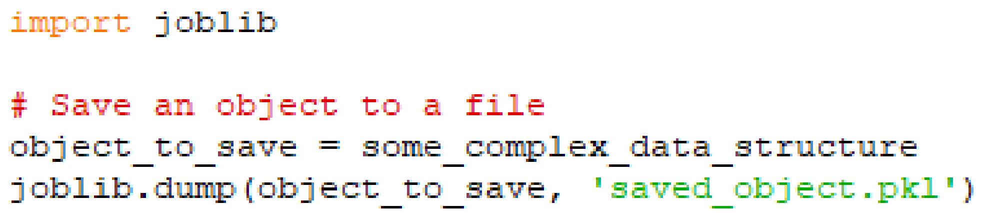
NumPy and SciPy Support: It works well with NumPy arrays and SciPy sparse matrices, beneficial for data scientists.

Machine Learning Models: Used for saving and loading machine learning models, avoiding retraining.

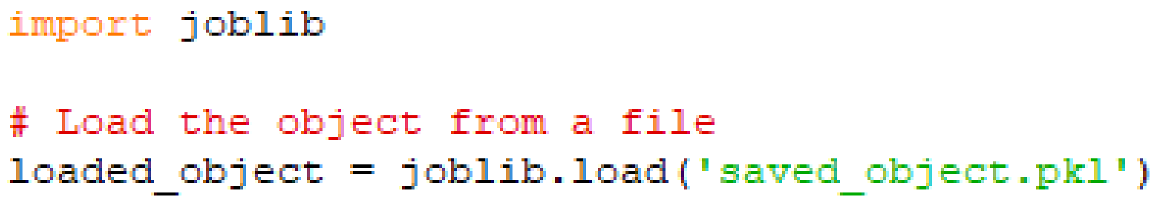
Parallel Computing: Offers parallel computing capabilities for certain tasks.

File Caching: Acts as a file-based cache for function results.

Simple Interface: Provides an easy-to-use interface for handling Python objects.



```python
import joblib

# Save an object to a file
object_to_save = some_complex_data_structure
joblib.dump(object_to_save, 'saved_object.pkl')
```



```python
import joblib

# Load the object from a file
loaded_object = joblib.load('saved_object.pkl')
```

TensorFlow

Google's open-source machine learning framework.

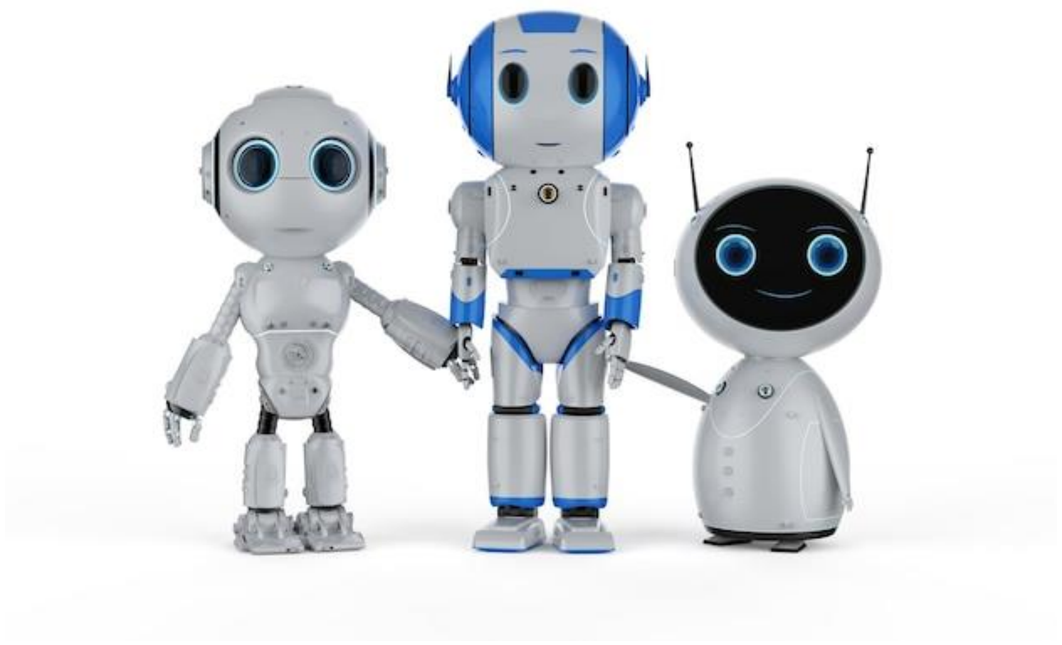
- **Deep Learning:** Especially well-suited for deep neural networks.
- **Flexibility:** Uses a computational graph for customization.
- **High Performance:** Optimized for CPU and GPU, ideal for large datasets.
- **Community and Ecosystem:** Supported by a large community and rich tools.
- **Deployment**: Suitable for deploying models in various environments.
- **Research:** Used for cutting-edge machine learning research.
- **Open Source:** Freely available and actively maintained.

Have a Break!

University of South-Eastern Norway

School of Business

Group Up!



Work on your assignment!

Guidance

https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

Cat or dog?

THANKS