Intro to Keras

Hello world!

Nachwa Aboubakr

Overview

- How to build your own Neural Network?
- Keras Library.
- First example.
- Exercise: Recognition of handwritten digits.

How to build your own NN?

Input X & Output Y
nature (fixed, sequential, ..), type, shape

The task
regression, classification, ..., and therefore, your loss function.

Architecture
Type of layers, # of layers, kernels size, ...

Tune hyperparameters
Learning rate, batch size, # of epochs, ...

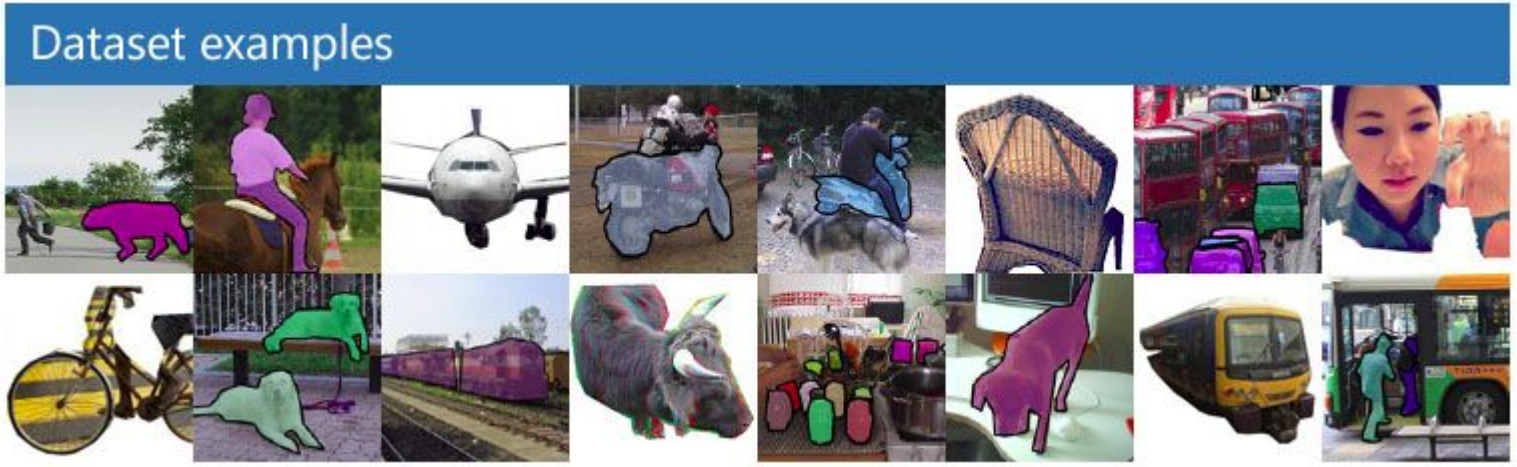
1. Specify Input (X) & Output (Y)

- Input:
  - Vector,
  - n-D matrix,
  - sequential data,
  - Multimodal input, ...
- Output:
  - discrete scalar,
  - vector,
  - n-D matrix,
  - sequential output, ...

2. Define the task

A. Classification predictive modeling:

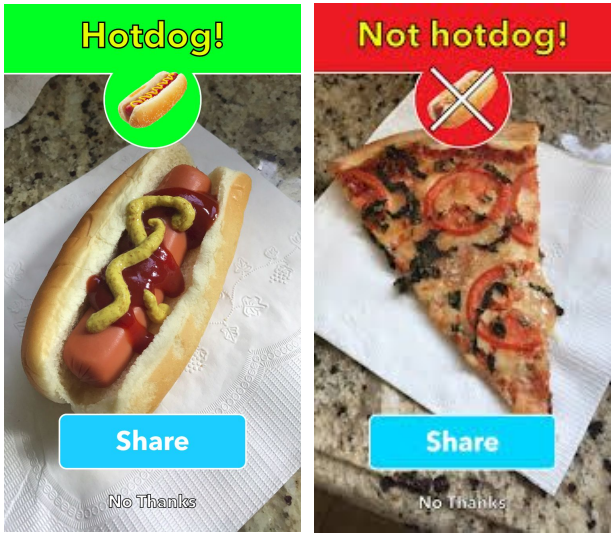
is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y). The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation.



2. Define the task

- A classification problem requires that examples be classified into one of two or more classes.
- A problem with two classes is often called a two-class or **binary classification problem**.
- A problem with more than two classes is often called a **multi-class classification problem**.
- A problem where an example is assigned to multiple classes is called a **multi-label classification problem**.

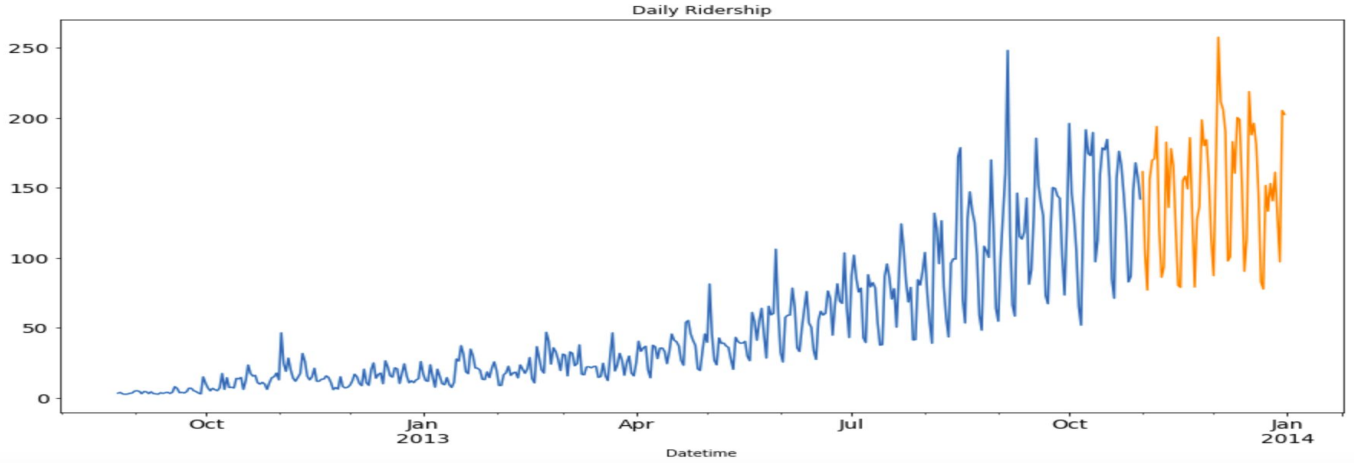
There are many ways to evaluate a classification predictive model, but perhaps the most common is to calculate the classification accuracy.



2. Define the task

B. Regression predictive modeling:

is the task of approximating a mapping function (f) from input variables (X) to a continuous output variable (y). A continuous output variable is a real-value, such as an integer or floating point value. These are often quantities, such as amounts and sizes. A regression predictive model predicts a quantity, therefore to evaluate the model we report an error in those predictions.



2. Define the task

Computer Vision Tasks

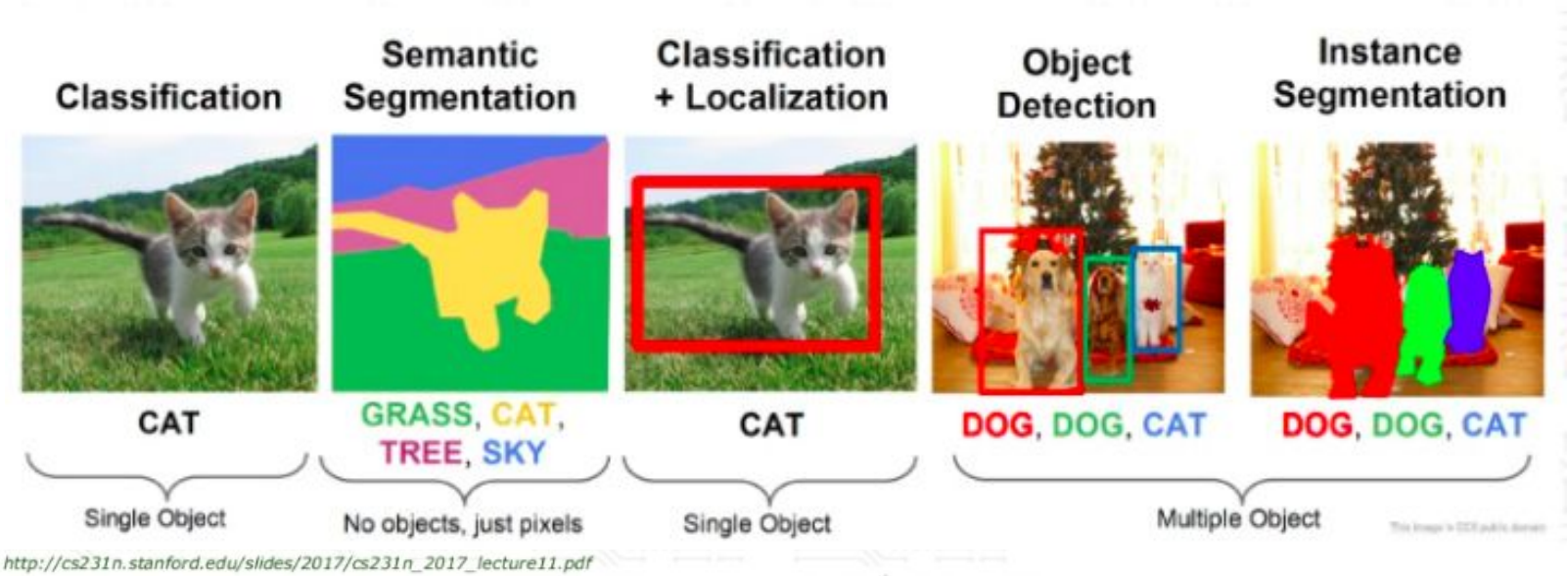


Figure 1: Different computer vision tasks. Source: Introduction to Artificial Intelligence and Computer Vision Revolution (https://www.slideshare.net/darian_f/introduction-to-the-artificial-intelligence-and-computer-vision-revolution).

3. Choose your Network Architecture

- Image related tasks:
  - Convolutional Neural Networks (CNN)
- Time-series tasks:
  - Recurrent Neural Networks (RNN)
- Video/Audio related tasks:
  - (?)

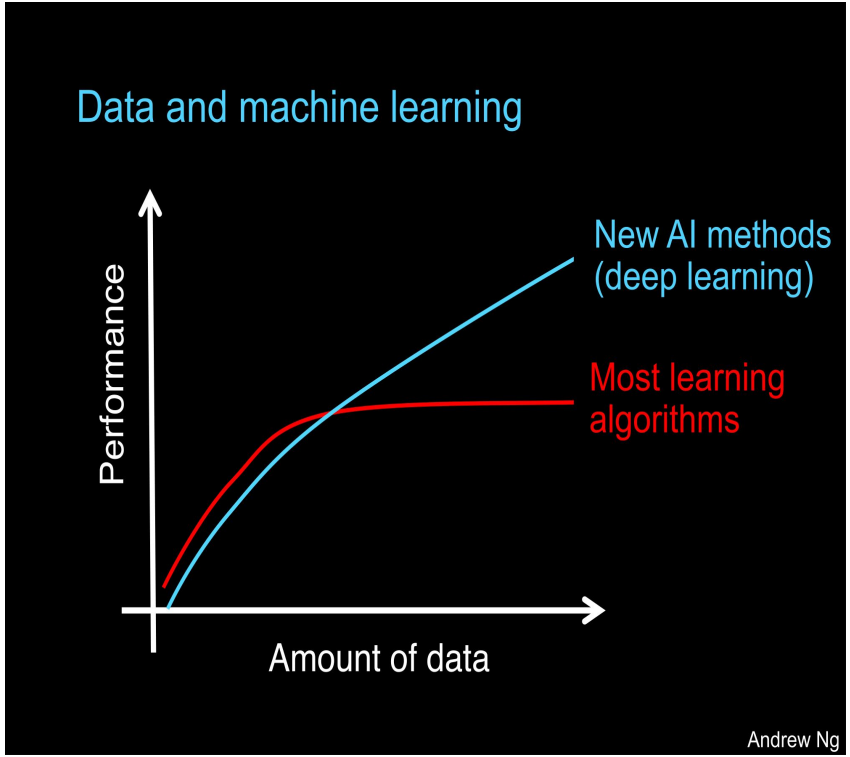
3. Choose your Network Architecture

- Image related tasks:
  - Convolutional Neural Networks (CNN)
- Time-series tasks:
  - Recurrent Neural Networks (RNN)
- Video/Audio related tasks:
  - (?)

Many architectures have been proposed in the literature.
Look for what suits your problem

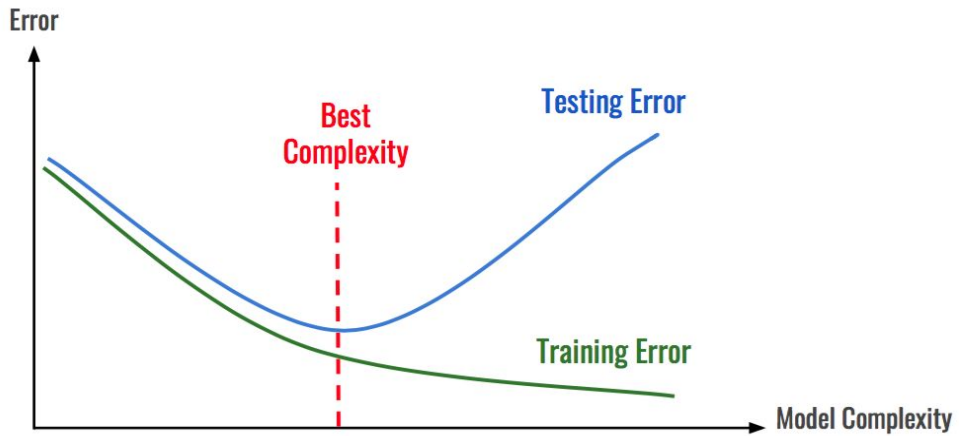
3. Choose your Network Architecture

- **Design choices:**
- **Transfer Learning** is a machine learning technique where a model trained on one task is re-purposed on a second related task.
- **Data Augmentation** is the technique of increasing the size of data used for training a model.



4. Start training ...

- Observe the progress of your training.
- Tune Hyperparameters.
  - Learning Rate
  - Number of epochs
- Test and evaluate your model.
- Don’t fall into an overfitting case!.



Existing Platforms

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both **convolutional networks** and **recurrent networks**, as well as combinations of the two.

[keras.io]

Sequential Model

The **Sequential** model is a linear stack of layers.

You can create a **Sequential** model by passing a list of layer instances to the constructor:

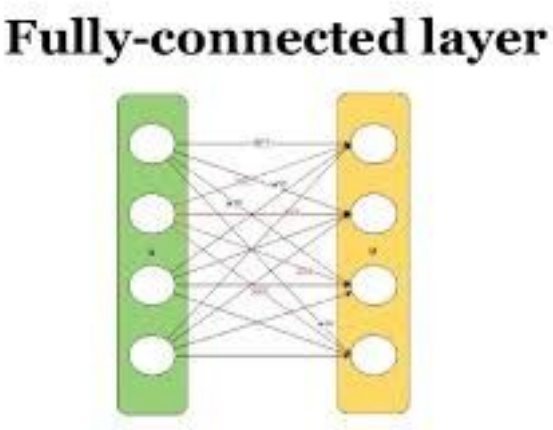
```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential([
    Dense(32, input_shape=(784,), activation='relu'),
    Dense(10,  activation='softmax')
])
```

```
from keras.models import Sequential
model = Sequential()
```

from keras import layers

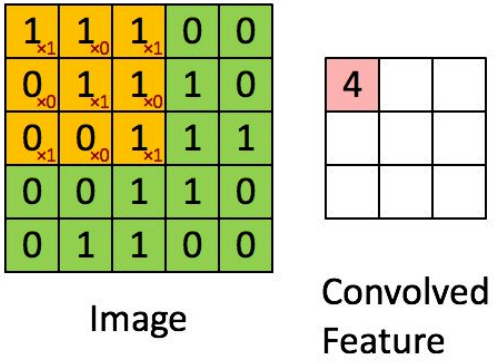
- Core layers:
  - Dense layer: fully connected layer

```
model.add(Dense(4,  activation='softmax'))
```



from keras import layers

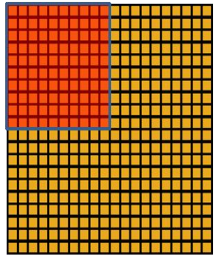
- Convolutional layers:
  - Conv1D, Conv2D, Conv3D
  - UpSampling1D, UpSampling2D...

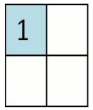



```
model.add(Conv2D(filters=5, kernel_size=(3,3), activation='sigmoid'))
```

from keras import layers

- Pooling Layers:
  - MaxPooling1D, MaxPooling2D, ...
  - AveragePooling1D, AveragePooling2D, ...





Convolved feature

Pooled feature

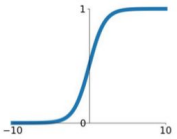
```
model.add(MaxPooling2D(pool_size=(8,8)))
```

from keras.layers import activations

Activation layers in neural networks, takes a value that is passed through a function which *squashes* the value into a range.

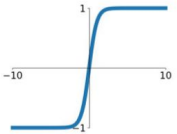
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



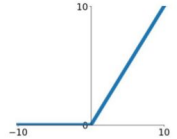
tanh

$\tanh(x)$



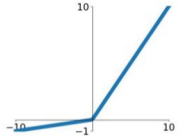
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

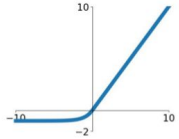


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



from keras.layers import activations

Softmax

- calculates the probabilities of each target class over all possible target classes.
- is often used in the final layer of a neural network-based classifier.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

```
model.add(Activation('softmax'))
```

Sigmoid

- returns a real-valued output.
- is often used as the activation function for artificial neurons.

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

```
model.add(Activation('sigmoid'))
```

from keras import losses

A **loss function** or **cost function** is a function that maps values of one or more variables onto a real number intuitively representing some associated "cost". An optimization problem seeks to minimize a loss function.

The loss function lets us quantify the quality of any particular set of parameters (weights **W** and biases **B**). Some available loss functions:

- **mean_squared_error**
- **mean_absolute_error**
- **...**
- **categorical_crossentropy**
- **binary_crossentropy**
- ...

from keras import optimizers

The goal of optimization is to find the set of parameters (**W** and **B**) that minimizes the loss function.

Some available optimizer:

- **SGD** #Stochastic gradient descent optimizer.
- **AdaGrad** #Adaptive gradient descent optimizer
- **RMSProp**
- **Adam** #adaptive moment estimation
- **...**

Algorithm Gradient Descent

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{g}_t$$

Source: Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).

from keras import optimizers

The goal of optimization is to find the set of parameters (**W** and **B**) that minimizes the loss function.

Some available optimizer:

- **SGD** #Stochastic gradient descent optimizer.
- **AdaGrad** #Adaptive gradient descent optimizer
- **RMSProp**
- **Adam** #adaptive moment estimation
- **...**

Algorithm Classical Momentum

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \mathbf{g}_t$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \mathbf{m}_t$$

Source: Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).

from keras import optimizers

The goal of optimization is to find the set of parameters (**W** and **B**) that minimizes the loss function.

Some available optimizer:

- **SGD** #Stochastic gradient descent optimizer.
- **AdaGrad** #Adaptive gradient descent optimizer
- **RMSProp**
- **Adam** #adaptive moment estimation
- **...**

Algorithm AdaGrad

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{n}_t \leftarrow \mathbf{n}_{t-1} + \mathbf{g}_t^2$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\mathbf{g}_t}{\sqrt{\mathbf{n}_t + \varepsilon}}$$

Source: Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).

from keras import optimizers

The goal of optimization is to find the set of parameters (**W** and **B**) that minimizes the loss function.

Some available optimizer:

- **SGD** #Stochastic gradient descent optimizer.
- **AdaGrad** #Adaptive gradient descent optimizer
- **RMSProp**
- **Adam** #adaptive moment estimation
- **...**

Algorithm RMSProp

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{n}_t \leftarrow \nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\mathbf{g}_t}{\sqrt{\mathbf{n}_t + \varepsilon}}$$

Source: Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).

from keras import optimizers

The goal of optimization is to find the set of parameters (**W** and **B**) that minimizes the loss function.

Some available optimizer:

- **SGD** #Stochastic gradient descent optimizer.
- **AdaGrad** #Adaptive gradient descent optimizer
- **RMSProp**
- **Adam** #adaptive moment estimation
- **...**

Algorithm Adam

$$\mathbf{g}_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$$
$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + (1-\mu)\mathbf{g}_t$$
$$\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1-\mu^t}$$
$$\mathbf{n}_t \leftarrow \nu \mathbf{n}_{t-1} + (1-\nu)\mathbf{g}_t^2$$
$$\hat{\mathbf{n}}_t \leftarrow \frac{\mathbf{n}_t}{1-\nu^t}$$
$$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{n}}_t + \varepsilon}}$$

Source: Dozat, Timothy. "Incorporating nesterov momentum into adam." (2016).

from keras import applications

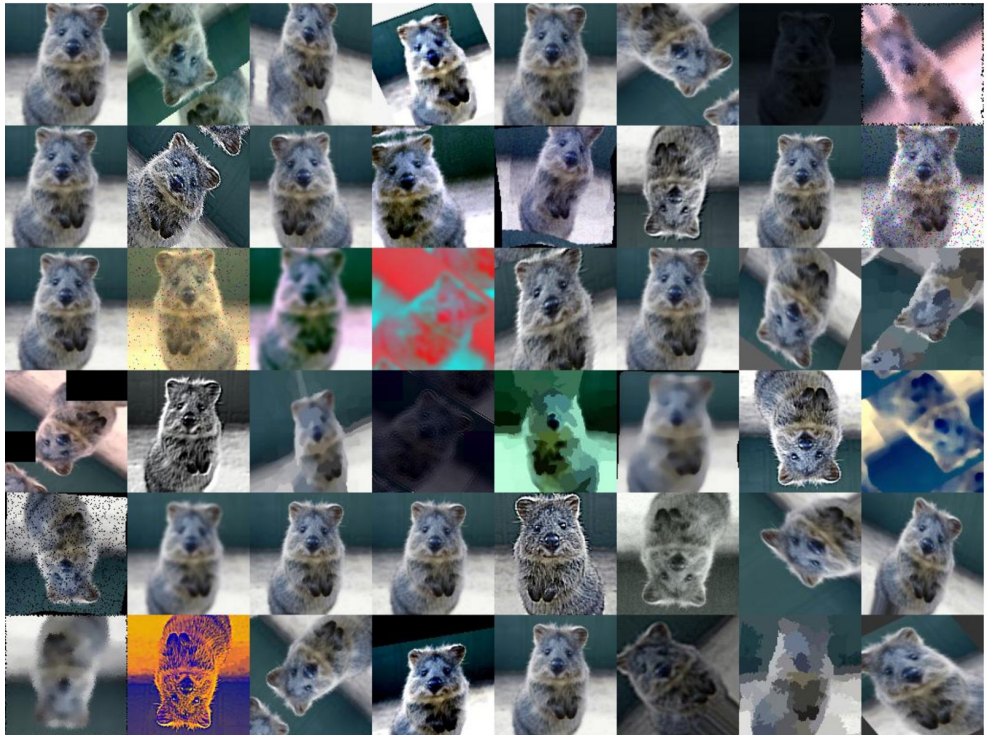
Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet
- DenseNet
- NASNet

```
from keras.applications.vgg16 import VGG16

model = VGG16(weights='imagenet', include_top=True)
```

from keras import data augmentation

Image Augmentation is the process of taking images that are already in a training dataset and manipulating them to create many altered versions of the same image.



from keras import data augmentation

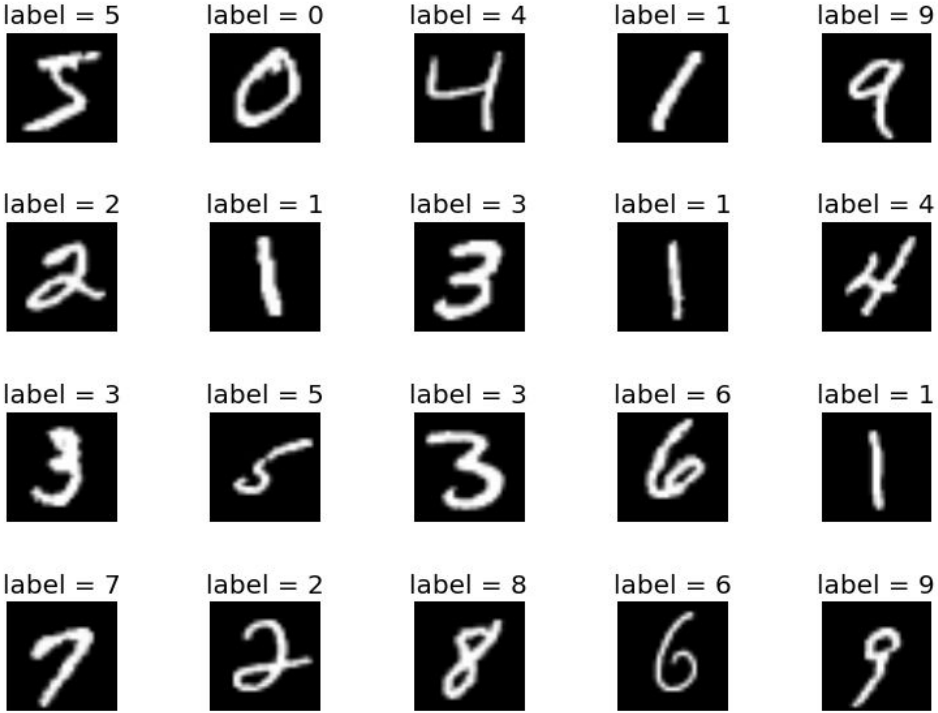
Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely.

```
datagen = ImageDataGenerator(
    featurewise_std_normalization=True,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)

datagen.fit(x_train)

for x, y in datagen.flow(x_train, y_train, batch_size=32):
    ...
```

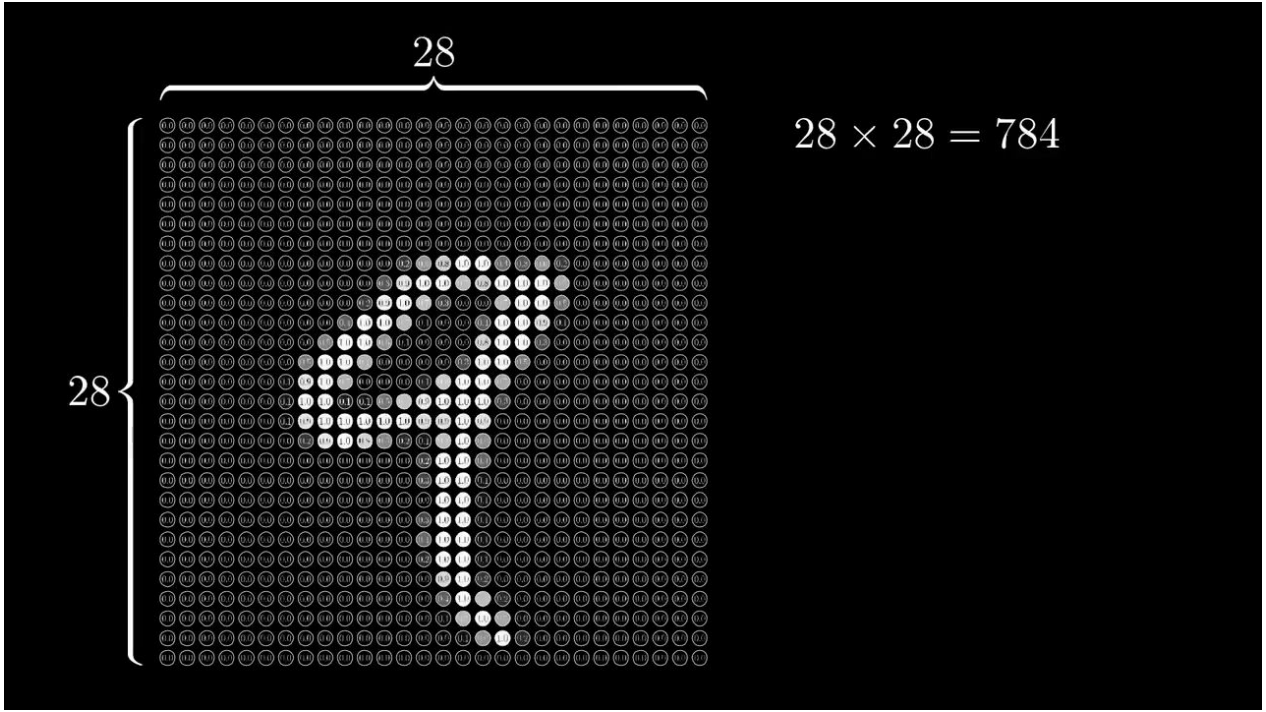
from keras import wrap-up!

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation(sigmoid))

opt = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=32)
score = model.evaluate(x_test, y_test, batch_size=32)
```

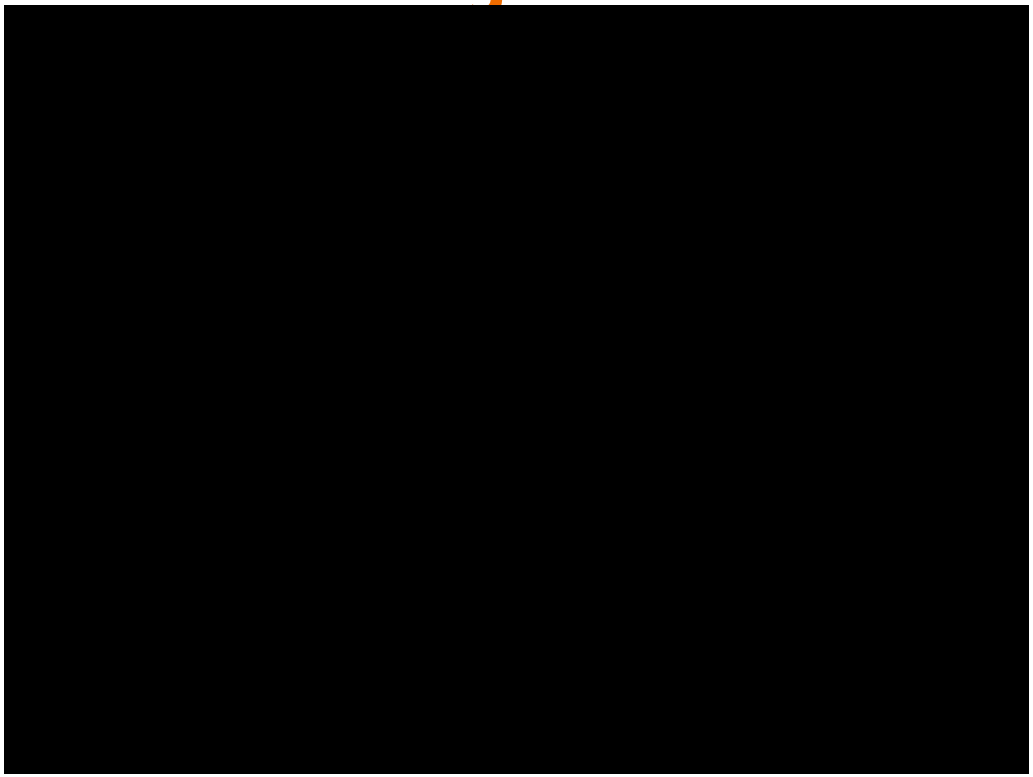
Exercise: handwritten digits



Exercise: handwritten digits



Exercise: handwritten digits



Useful links:

Conda: https://docs.conda.io/en/latest/miniconda.html

Install keras: https://anaconda.org/conda-forge/keras

Keras Docs: https://keras.io/

MNIST dataset: http://yann.lecun.com/exdb/mnist/

If you have any question, feel free to contact me

- nachwa.aboubakr@inria.fr