Deep Convolutional Neural Networks for Image Classification

Rodrigo Santa Cruz

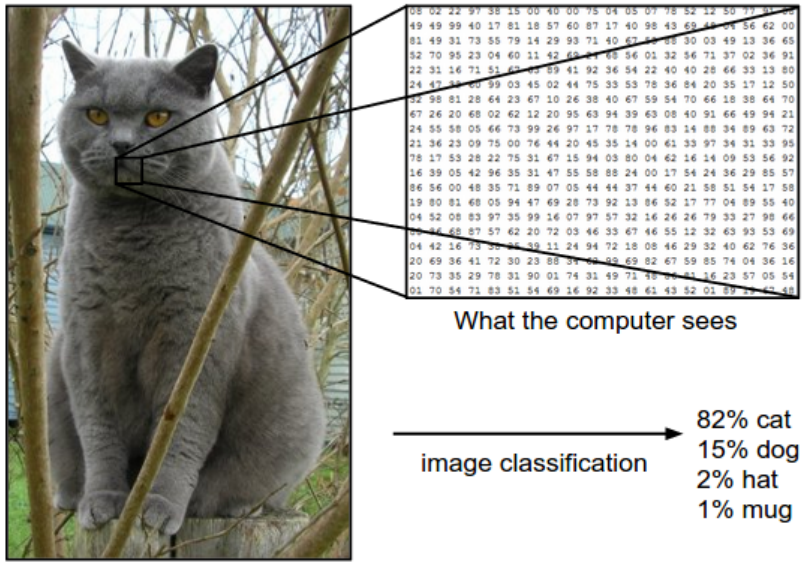
The Australian National University

March 15, 2017

Overview

- Image Classification
- Convolutional Neural Network
- CNN step-by-step with Keras

Image Classification

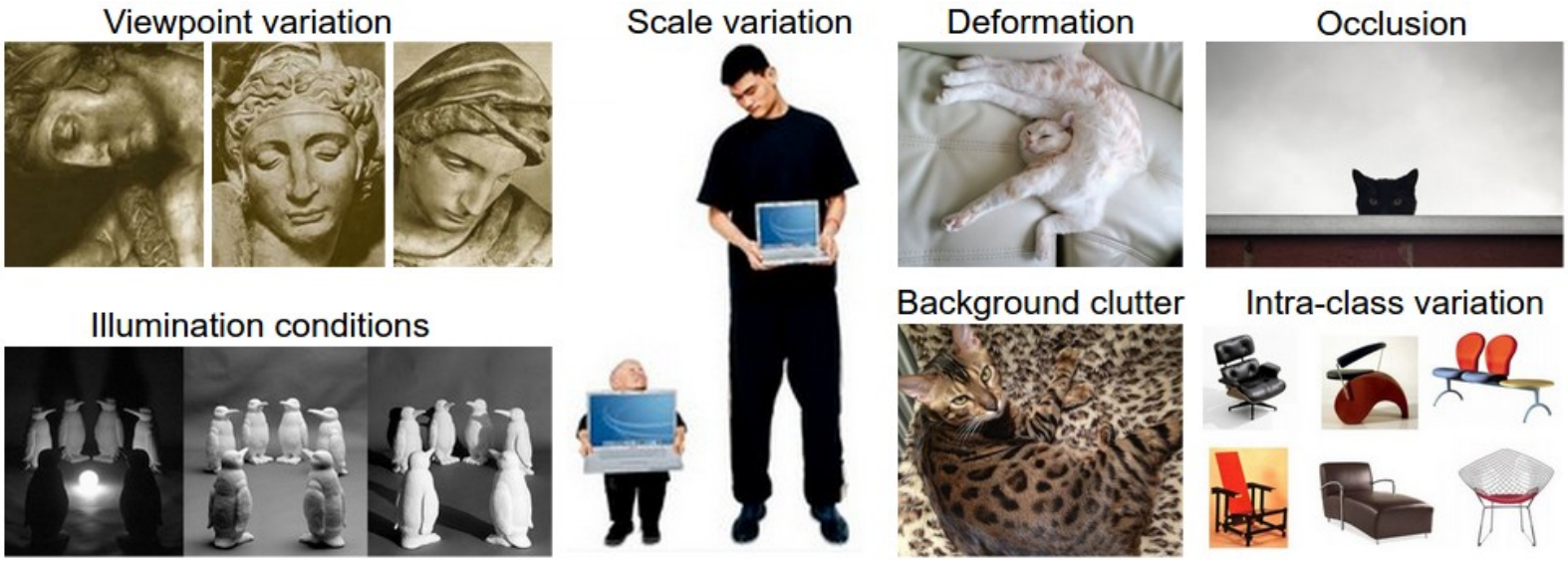


Motivation:

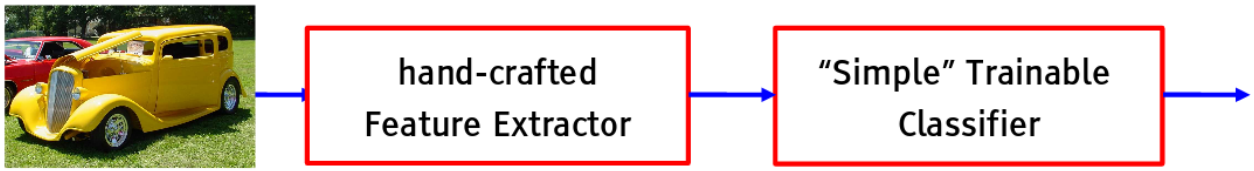
- Image Classification is the task of assigning an input image one label from a fixed set of categories.
- Computer Vision tasks, such as object detection and segmentation, can be reduced to image classification.

Image Classification

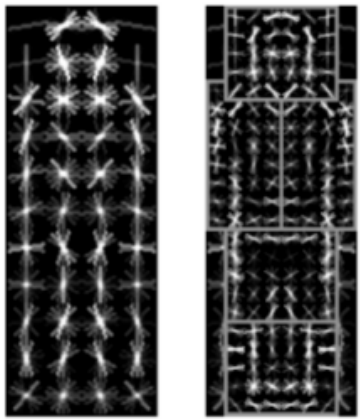
Challenges:



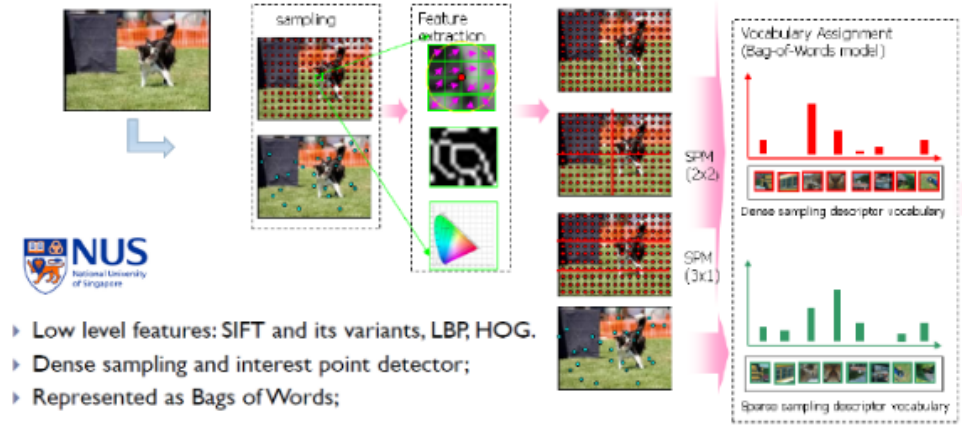
Traditional Recognition Approach



- Designing a feature extractor requires considerable efforts by experts.
- Ex: SIFT, HOG, LBP, ...
- The performance of machine learning algorithms depends on the quality of the extracted features.



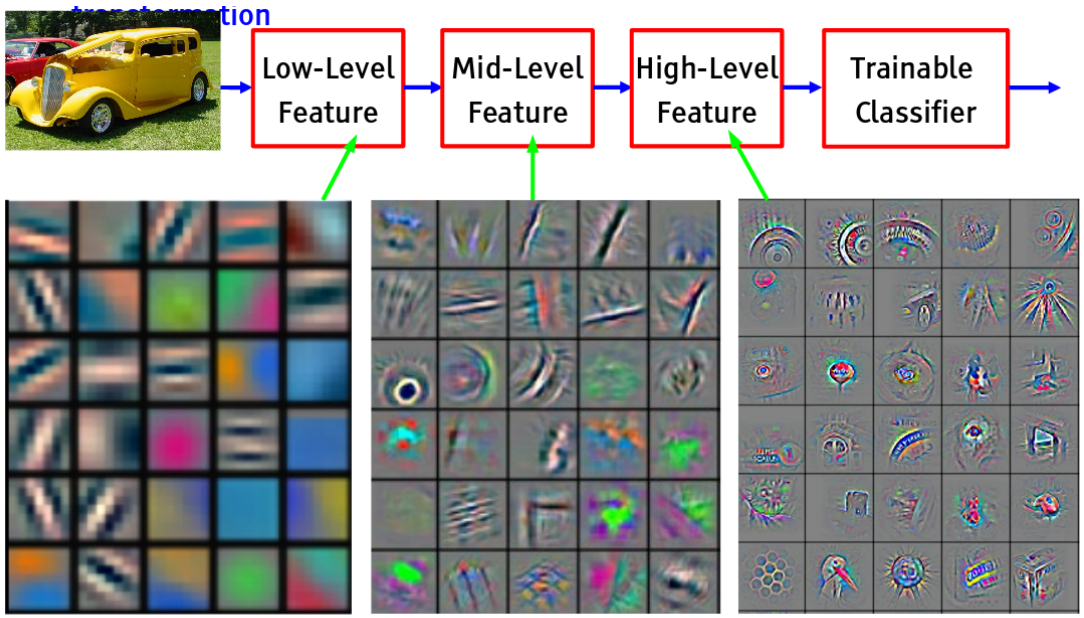
Felzenszwalb, Girshick,
McAllester and Ramanan, PAMI 2007



Yan & Huang
(Winner of PASCAL 2010 classification competition)

End-to-end Classifiers

Learn a feature hierarchy all the way from pixels to classifier.

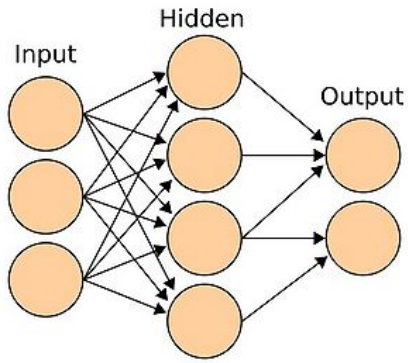


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Hierarchy of representations with increasing level of abstraction:

- Image recognition: Pixel $\rightarrow$ edge $\rightarrow$ texton $\rightarrow$ motif $\rightarrow$ part $\rightarrow$ object
- Text: Character $\rightarrow$ word $\rightarrow$ word group $\rightarrow$ clause $\rightarrow$ sentence $\rightarrow$ story

Multi-Layer Neural Network



- Model: Non-linear mapping from pixels to class labels.

$$f_w(x) = softmax(w_2^T \sigma(w_1^T x + b_1) + b_2)$$

- Training: Find network weights w which minimize the error between true training labels y_i, and predicted labels $f_w(x_i)$

$$E(w) = \frac{-1}{n} \sum_{i=1}^{n} \sum_{k=1}^{C} y_{i,k} log(f_w(x_i)_k)$$

- The training can be done by gradient descent provided f_w is differentiable which is commonly referenced as back-propagation.

Problems with MLPs

Regular Neural Nets don't scale well to full images.

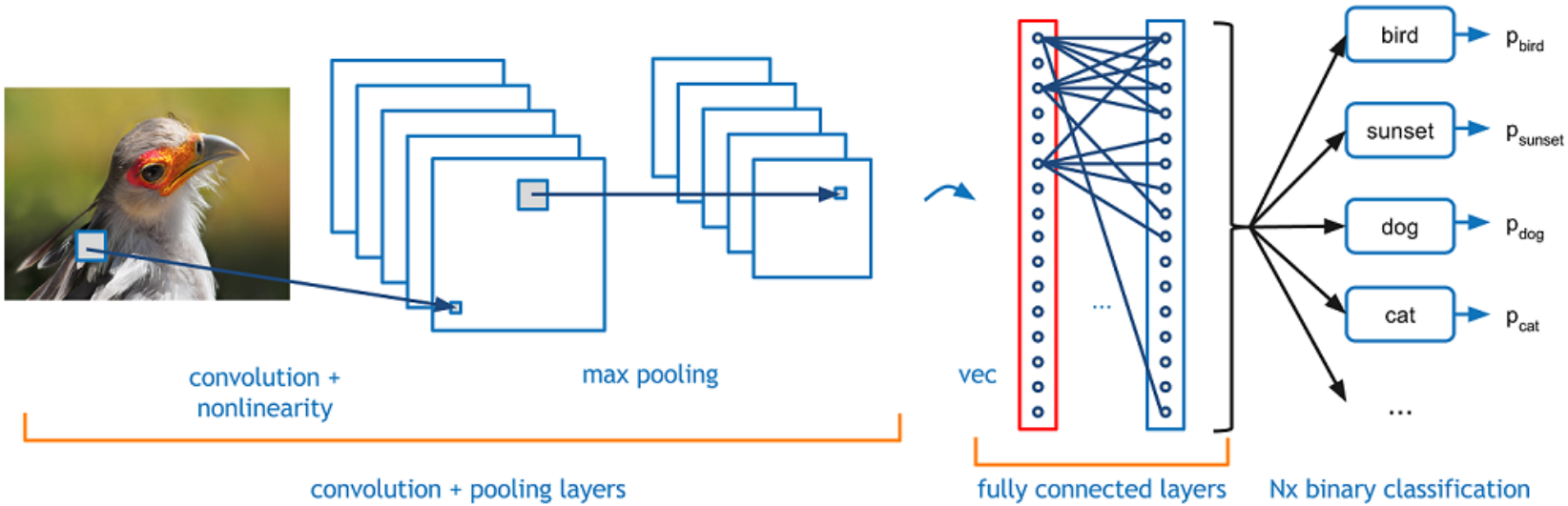
- A image of 200x200x3, would lead to neurons that have 120,000 weights.
- Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

Regular Neural Nets are spatial ignorant.

- learning to recognize a object in one location wouldn't transfer to the same object presented in a different part of the visual field.

We need an architecture that exploited the two dimensional spacial constraints imposed by images whilst reducing the amount of parameters involved in training. Convolutional neural networks are the architecture.

Convolutional Neural Networks



CNN Layers:

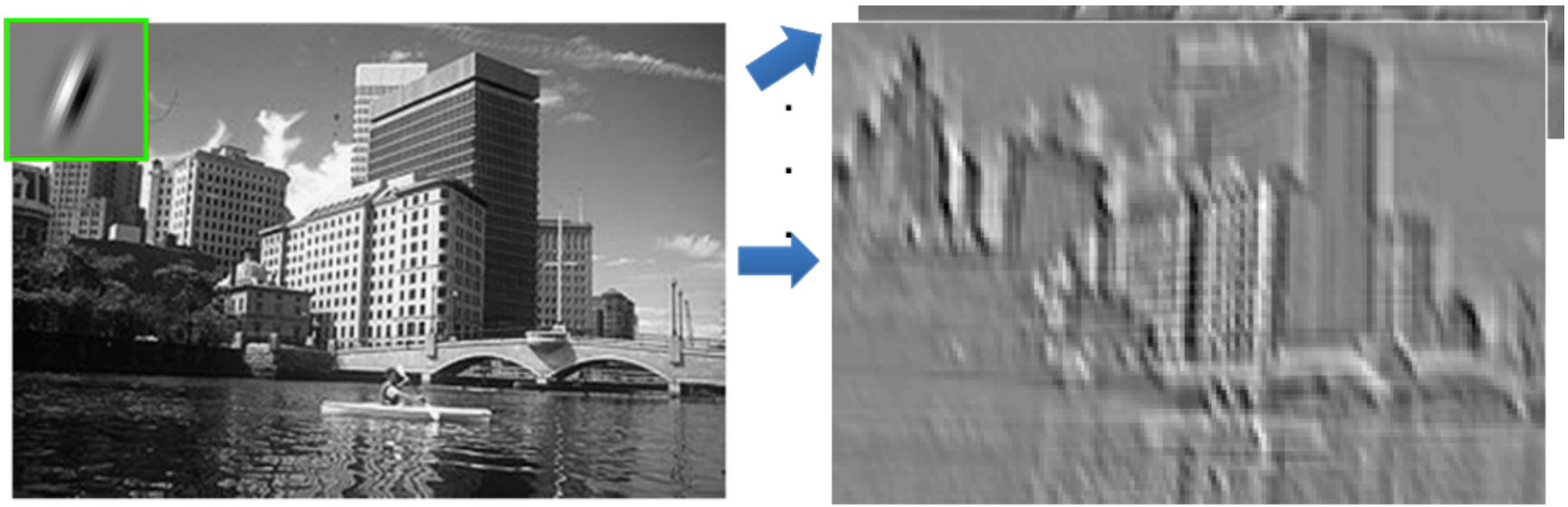
- Convolution
- Spatial Pooling
- Activations
- Fully-connected Layers (MLP's hidden layer)
- Dropout

Supervised training of convolutional filters by back-propagating classification error.

Convolution

Why convolution layers?

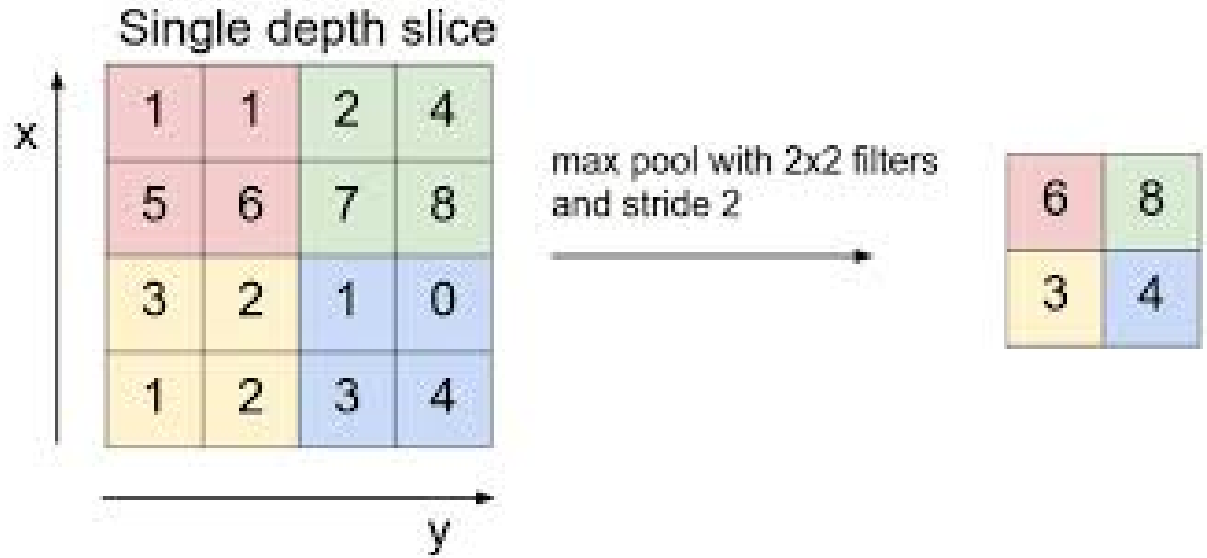
- Parameter sharing
- Translation Invariance
- Stride can be greater than 1 (faster, less memory).



Spatial Pooling

Why Spatial Pooling layers?

- Reducing the overall size of a signal (sub-sampling). Consequently, the amount of parameters and computation in the network are reduced.
- Increasing position invariance
- Operations: Sum, Max and Average.

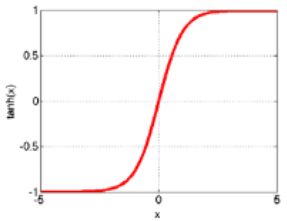


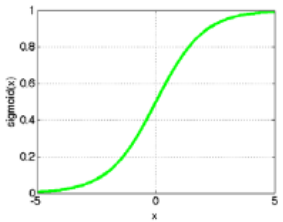
Activations

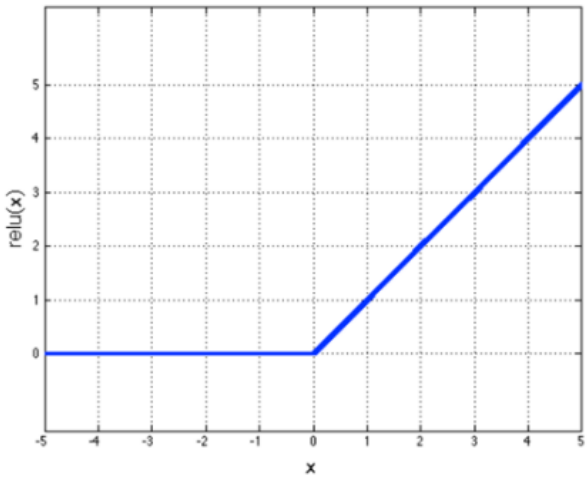
Rectified linear unit (ReLU)

- Simplifies backpropagation
- Makes learning faster
- Avoids saturation issues

other alternatives: sigmoid, tanh, ...

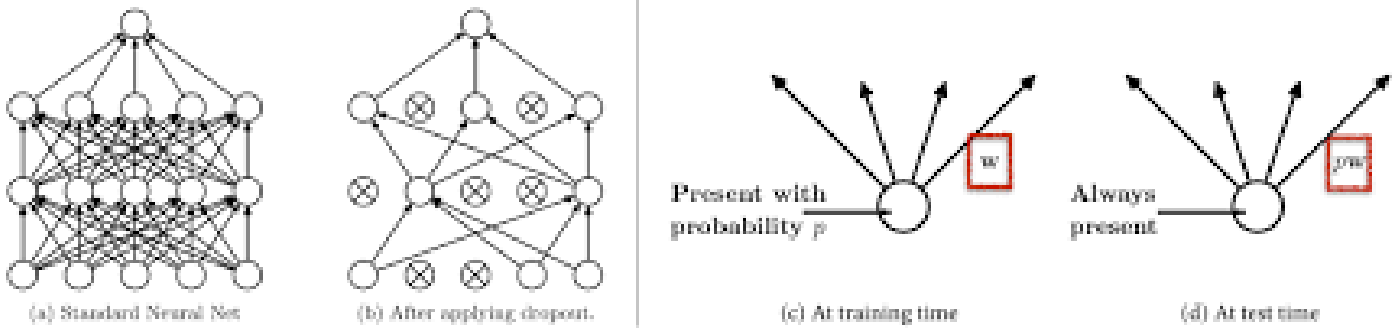






Dropout

- Dropout is a form of regularisation.
- It essentially forces an artificial neural network to learn multiple independent representations of the same data by alternately randomly disabling neurons in the learning phase (ensemble learning).
- The effect of this is that neurons are prevented from co-adapting too much which makes overfitting less likely.



(a) Standard Neural Net (b) After applying dropout. (c) At training time (d) At test time

Put it all together

There are still many free hyper-parameters.

- how many filters per convolutional layer?
- How big should the filters and subsamples be?
- how many overall layers should there be?

None of these questions can be answered definitively, as the effectiveness of each hyper-parameter setting depends on the task. Stick with AlexNet, VGG, ResNet, ...

Building a CNN-based image classifier step-by-step

1. Data Loading
2. Model Definition
3. Compile Model
4. Training
5. Validation
6. Prediction

Installation

- Installation using Anaconda and Pip

```
# optional
$ conda create -n keras_tf python=3
$ source activate keras_tf
$ conda install numpy scipy scikit-learn pillow h5py

$ export TF_BINARY_URL= ....<tensorflow version>...
$ pip install --upgrade $TF_BINARY_URL
$ pip install keras
```

- TensorFlow "Download and Setup" page

```
# Ubuntu/Linux 64-bit, GPU enabled, Python 3.4
# Requires CUDA toolkit 8.0 and CuDNN v5. For other versions, see "Install from sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.11.0-cp34-cp34m-li

# Ubuntu/Linux 64-bit, CPU only, Python 3.5
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-0.11.0-cp35-cp35m-li

# Ubuntu/Linux 64-bit, GPU enabled, Python 3.5
# Requires CUDA toolkit 8.0 and CuDNN v5. For other versions, see "Install from sources" below.
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.11.0-cp35-cp35m-li

# Mac OS X, CPU only, Python 3.4 or 3.5:
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-0.11.0-py3-none-any.wh
```

Verification

- Test installation

```
$ source activate keras_tf
$ python
>>> import keras
Using TensorFlow backend.
```

- verify `~/.keras/keras.json`

```
{
  "image_dim_ordering": "tf",
  "epsilon": 1e-07,
  "floatx": "float32",
  "backend": "tensorflow"
}
```

Data Loading

- Loading **small** dataset in memory using numpy arrays
- Building data points with python generators

```
def generate_arrays_from_file(path):
  while 1:
    f = open(path)
    for line in f:
      x1, y = process_line(line)
      yield ({'input_1': x1, 'output': y})
    f.close()
```

- Keras **ImageDataGenerator**: Generate batches of tensor image data from a given directory with real-time data augmentation.

Define Model

- Sequential API: The model is a stack of layers added sequentially via the .add() method.

```
model = Sequential()
model.add(Dense(64, input_dim=20, init='uniform', activation='sigmoid
model.add(Dense(64, init='uniform', activation='sigmoid'))
model.add(Dense(10, init='uniform', activation='softmax'))
```

- Functional API: The model is a sequence of functions having tensors as input and outputs.

```
inputs = Input(shape=(784,))
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
model = Model(input=inputs, output=predictions)
```

Compile Model

In order to compile our model we need to specify:

- Loss function: Binary Cross entropy, Mean Squared Error, ...
- Optimizer: SGD, ADAM, RMSProp, ...
- Metrics: accuracy, precision, ...

```
model.compile(loss='', optimizer='', metrics=[], callbacks=[])
```

Fit Model

We have defined and compiled our model. Now it is time to *fit* the model on some data. We can train our model by,

- calling the method fit on loaded data.

```
model.fit(x, y, batch_size, ...)
```

- calling fit_generator on data generated by a Python generator. The generator runs in parallel to the model training, for efficiency.

```
model.fit_generator(generator, ...)
```

- Callbacks: Functions to run at key stages of the training procedure. Ex.: ModelCheckpoint, EarlyStopping, TensorBoard.

Evaluate Model

We have trained our neural network on the training dataset. Now, we can evaluate the performance of the network on the validation dataset.

- evaluating on loaded data:

```
model.evaluate(x, y, batch_size=32, ...)
```

- evaluating on data generated by a Python generator:

```
model.evaluate_generator(generator, ...)
```

- The model can be evaluated at the end of each training epoch by passing the validation generator to the method model.fit_generator(...).

To make predictions for a new data point x, we do model.predict(x)

Deep Convolutional Neural Networks for Image Classification

Rodrigo Santa Cruz

The Australian National University

March 15, 2017