TA Session 10: Neural Networks

Keras Framework & Image Restoration

Image Processing Course (67829)

The Hebrew University of Jerusalem

December 27, 2017

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

The Goal of Learning

We wish to learn a mapping $f(\mathbf{x})$ from a *domain set* $\mathcal{X}$ and a *label set* $\mathcal{Y}$.

Two common settings:

- Classification: $\mathcal{Y}$ is discrete, where each $y \in \mathcal{Y}$ represents a "class".
  - $\mathcal{X}$ contains images of animals, and $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{deer}, \ldots\}$.
    Goal: $f(x)$ recognize the animal in the image.
  - $\mathcal{X}$ contains sounds of speech, and $\mathcal{Y}$ words in English.
    Goal: $f(x)$ convert speech to text.
- Regression: $\mathcal{Y}$ is continuous
  - $\mathcal{X}$ contains images of faces, and $\mathcal{Y} = \mathbb{R}^2$.
    Goal: $f(x)$ returns location of the mouth in the image.
  - $\mathcal{X}$ contains corrupted images, and $\mathcal{Y}$ natural images.
    Goal: $f(x)$ restore a corrupted image to its original version.

Learning Parametric Functions

Training set: a collection of "labeled examples" $S{=}\{(x_i{\in}\mathcal{X}, y_i{\in}\mathcal{Y})_{i=1}^N$ (the pair (x_i, y_i) is also called a "sample").
Example: we show people images and ask they write what is in them.

Hypothesis space: a set of parametric functions $\mathcal{H} = \{f_\theta(x) | \theta \in \mathbb{R}^s\}$.
Examples:

- Linear functions, $f(x){=}\sum_{i=1}^s x_i \cdot w_i$, parameterized by vector $w \in \mathbb{R}^s$.
- Linear transforms, $f(x){=}Ax$, parameterized by matrix $A \in \mathbb{R}^{d\times s}$.
- Neural networks...

Goal: Find the parameters θ s.t. $f_\theta \in \mathcal{H}$ "best fit" the training set S.

Not all parameters are learned!

- **Learned-Parameters:** the attributes of $\mathcal{H}$ we change to fit to S.
- **Hyper-parameters:** the fixed attributes of $\mathcal{H}$ (i.e. not learned!).

What does it mean "best fit"?

Loss function: $L(f, S)$ measure the "error" of f w.r.t. S – the lower the loss the better f fit to S.

Goal: Find f that minimizes the loss: $f^* = \text{argmin}_{f \in \mathcal{H}} L(f, S)$.

Examples:

- "0-1 Loss": the percentage of examples on which f is wrong.

$$L_{\text{0-1}}(f, S) = \frac{\text{number of times } f(x_i) \neq y_i}{\text{total number of examples}}$$

- "Mean Square Error": average distance of $f(x_i)$ from y_i:

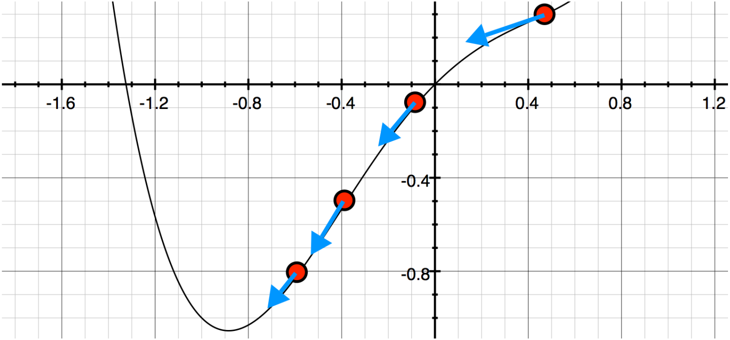
$$L_{\text{MSE}}(f, S) = \frac{1}{|S|} \sum_i ||f(x_i) - y_i||^2$$

- "Cross Entropy Loss"[1]: Assume $f(x)$ outputs the probabilities of x belonging to each possible class. Use probability of choosing the correct class on all examples.

[1] Sometimes known as "Softmax Loss"

Minimizing the Loss Function

If $L(f, S)$ is a differentiable function, then we can use **Gradient Descent**:



Key Principle: At each iteration of GD we take a step in the direction of steepest descent. The step size depends on the implementation.

Minimizing the Loss Function (Cont')

Stochastic GD: If $|S|$ is large then every GD iteration is expensive!
Instead, we approximate it using a small randomly chosen batch $B \subset S$.

SGD Loop:

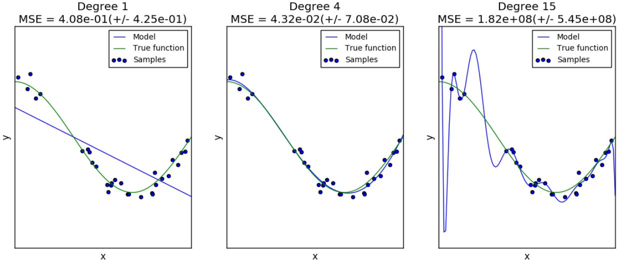
1. Sample a random batch $B \subset S$.
2. Compute the gradient of $L(f, B)$ w.r.t. the parameters of f.
3. Update parameters of f using the gradient.

Original SGD could be difficult to use for non-experts!
$\Rightarrow$ ADAM is an SGD variant which is easier to use in practice.

Generalization and Overfitting

Assume we have a model that fits the training set.

Q: Is it really good? Not necessarily, it could **overfit** the data!



$\Rightarrow$ We must evaluate the model on **unseen data**!

1. **Before training:** split data into training, validation and test sets.
2. **During training:** choose hyper-parameters using the validation set.
3. **After training:** evaluate final model on test set (generalization error).

Learning Summary

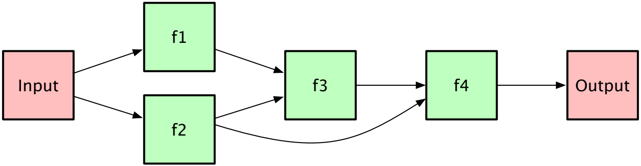
- We wish to fit a parametric function $f_\theta(x)$ to a training set S.
- We measure the fitness with a loss function $L(f_\theta, S)$:
  - For classification we typically use "Cross-Entropy Loss".
  - For regression we typically use "Mean Square Error".
- We find the best f_θ by minimizing the loss.
- For minimization we use SGD or one of its variants.
- Beware of overfit! Always evaluate your model on unseen data.

Outline

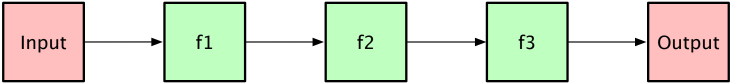
1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Neural Network Definition

In Broad Terms: A directed acyclic graph of differentiable operations



- Each node is called a "**layer**", defined by its type and parameters.
- All the parameters together are called "the **weights** of the network".
- The "**depth** of a network" is the longest path from input to output.
- When learning we fix the structure, and only learn the weights.
- A network is called "**feed-forward**" If every node has at most one input and one output connection:



Intermediate Results as "Tensors"

Basic Definitions:

- "Tensors"[1] are a fancy term for multi-dimensional arrays $A[d_1, \ldots, d_n]$
- For n as above, we say A is an n-D tensor.
- For each i, the index d_i ranges in $\{1, \ldots, M_i\}$.
- The shape of a tensor is the tuple $(M_1, \ldots, M_n)$.
- We can denote a tensor following the above as $A \in \mathbb{R}^{M_1 \times \ldots \times M_n}$.

In the context of Neural Networks for images:

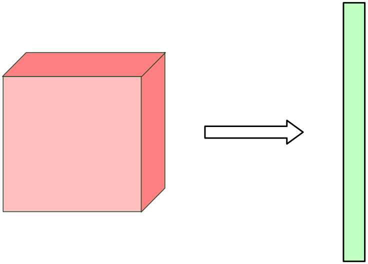
- Images are 3D tensors of the shape (channels, height, width)[2]. For RGB channels=3. For grayscale channels=1.
- The input/output of each layer are expressed as tensors.
- The shape and dimension of the output depends on the type of layer.

[1] Not strictly related to the mathematical object also called a "tensor".

[2] This is a common convention. Some use (width, height, channels) instead.

Common Layers: Dense / Fully-Connected

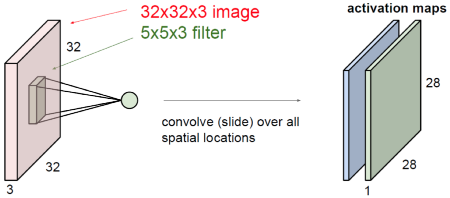
- **Operation**: $f(x) = Ax + b$.



- **Input:** assume $x \in \mathbb{R}^N$. General shapes are first flatten to 1D tensors.
- **Output:** $y \in \mathbb{R}^M$.
- **Hyper-Parameters:** dimension of the output, denoted by M.
- **Learned Parameters:** matrix $A \in \mathbb{R}^{M \times N}$ (known as "weights"), and vector $b \in \mathbb{R}^M$ (known as "bias").

Common Layers: Convolution

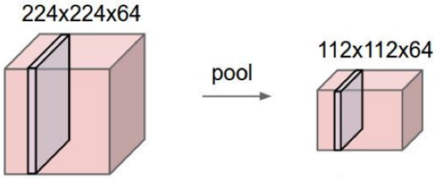
- **Operation:** Convolve the input with a set of kernels.



- **Input:** a 3D tensor of shape (in_channels, height, width).
- **output:** a 3D tensor of shape (out_channels, height, width).
- **Hyper-Parameters:** (i) spatial shape of kernels $B \times B$, and (ii) number of kernels M (known as "number of output channels").
- **Learned Parameters:** for $1 \leq j \leq M$, the kernel $w_j \in \mathbb{R}^{C \times B \times B}$, where C is the number of input channels.

Common Layers: Pooling

- **Pooling:** Sub-sample each input channel:



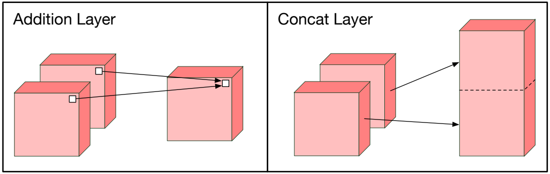
  - **Average Pooling:** Replace every window with its average value.
  - **Max Pooling:** Replace every window with its maximum value.
- **Input:** 3D tensor of shape $(\text{channels}, \text{height}, \text{width})$.
- **Output:** 3D tensor of shape $(\text{channels}, \frac{\text{height}}{\text{window's height}}, \frac{\text{width}}{\text{window's width}})$.
- **Hyper-Parameters:** size of pooling window (typically 2×2).
- **Learned Parameters:** None.

Common Layers: Activation

- **Operation:** Apply point-wise function $\sigma : \mathbb{R} \to \mathbb{R}$, called "activation function", on every entry of the input tensor. Common types:
  - **ReLU:** $\sigma(z) = \max(z, 0)$.
  - **Hyperbolic Tangent:** $\sigma(z) = \tanh(z)$.
  - **Sigmoid:** $\sigma(z) = \frac{1}{1+\exp(-z)}$.
- **Input:** any n-D tensor.
- **Output:** same shape as input.
- **Hyper-Parameters:** typically none (exception: Leaky ReLU).
- **Learned Parameters:** typically none (exception: PReLU).

Common Layers: Merge

Combines multiple input tensors $X^{(1)}, \ldots, X^{(k)}$ into a single output Y.



- **Addition:**
  - **Operation:** $Y[d_1, \ldots, d_n] = \sum_{i=1}^{k} X^{(i)}[d_1, \ldots, d_n]$ (pointwise addition)
  - **Input:** all input tensor must have the same shape!
  - **Output:** same shape as input tensor.
  - **Hyper-Parameters:** none. **Learned Parameters:** none.
- **Concat:**
  - **Operation:** Stack all tensors along a chosen axis (default to channels).
  - **Input:** same shape except the dimension of chosen axis.
  - **Output:** same as input, except the combined length of chosen axis.
  - **Hyper-Parameters:** chosen axis. **Learned Parameters:** none.

Outline

1 Recap on Neural Networks
- Learning
- Neural Networks
- ResNet Architecture

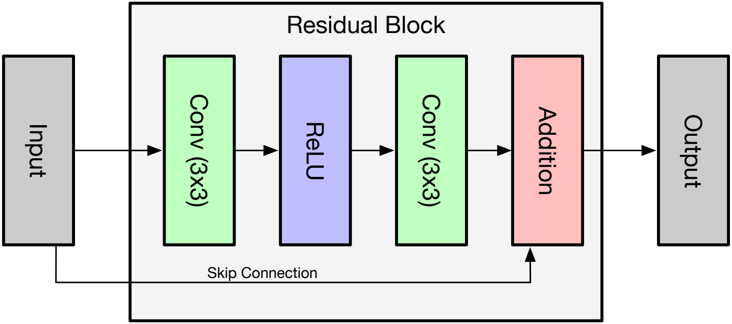
2 Keras Framework
- Model Definition
- Training a Model
- Prediction with a Trained Model

3 Ex5: Image Restoration
- General Workflow
- Applications to Denoising and Deblurring

Residual Block

Principle: represent functions as a difference from the identity ($f(x){=}x$).

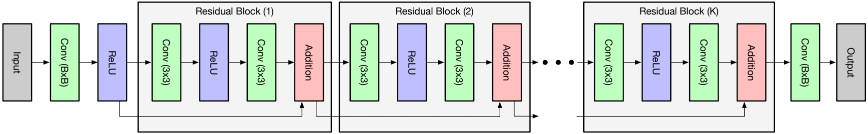
- $F(x)$ is a sequence of regular layers.
- We merge $F(x)$ with the input and return $H(x) = F(x) + x$.



Note: a residual block could contain any number of layers and any type.

ResNet Model

ResNet is compose of a sequence of residual blocks, in addition to standard layers:



Advantages:

- Allows training very deep networks.
- Deeper networks typically require overall less parameters.
- Faster convergence time.
- Improve generalization – better results on test set.

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Keras Framework for Neural Networks

Python packages for neural networks:

- **Tensorflow** is a general low-level machine learning library by Google.
- **Keras** is a neural network framework built on top of Tensorflow.

Keras has two methods for model definitions:

- **Sequential API:** Very simple for feed-forward networks (just stack layers). Limited for other cases (no ResNets!).
- **Model API:** More versatile but requires more work $\Leftarrow$ what we use!

See complete documentation in `https://keras.io`.

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Model API and Symbolic Tensors

Principles:

- Layers are connected by "symbolic" tensors – placeholder variables.
- Models are defined by input and output tensors.

Basic example:

```
from keras.layers import Input, Dense
from keras.models import Model

# this returns a symbolic tensor
a = Input(shape=(784,))

# a layer instance is callable on symbolic tensors
x = Dense(64)(a)
x = Dense(64)(x)
predictions = Dense(10)(x)

# Create a model from input and output tensors.
model = Model(input=a, output=predictions)
```

Common Layers in Keras

- The **Input** layer in Keras defines a symbolic tensor of a given shape:

```
from keras.layers import Input
a = Input(shape=(3,32,32))
```

- **Convolution2D**: for input tensors of shape (channels, height, width).
  Arguments: output channels, window's height, window's width.
  `border_mode='same'` adds zero padding to preserve input shape.

```
from keras.layers import Convolution2D
b = Convolution2D(16, 3, 3, border_mode='same')(a)
```

- **Activation**: a single input specifying activation type

```
from keras.layers import Activation
c = Activation('relu')(b)
```

Common Layers in Keras (cont')

- **MaxPooling2D**: for input tensors of shape (channels, height, width). **Arguments:** pooling window's height, pooling window's width.

```
from keras.layers import MaxPooling2D
d = MaxPooling2D(pool_size=(2, 2))(c)
```

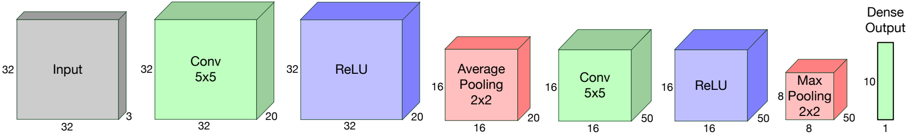
- **AveragePooling2D**: (similar to above)

```
from keras.layers import AveragePooling2D
d = AveragePooling2D(pool_size=(2, 2))(c)
```

- **Dense**: a single input specifying output dimension

```
from keras.layers import Dense
e = Dense(32)(d)
```

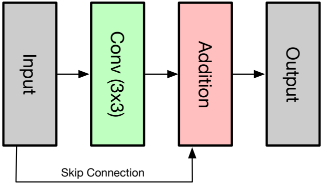
Feed-forward ConvNet Example




```
from keras.models import Model
from keras.layers import Input, Convolution2D #etc
a = Input(shape=(3, 32, 32))
b = Convolution2D(20, 5, 5, border_mode='same')(a)
b = Activation('relu')(b)
b = AveragePooling2D(pool_size=(2,2))(b)
b = Convolution2D(50, 5, 5, border_mode='same')(b)
b = Activation('relu')(b)
b = MaxPooling2D(pool_size=(2,2))(b)
b = Dense(10)(b)
model = Model(input=a, output=b)
```

Skip Connections via merge()

The merge() function takes a list of tensors, a merging method, and returns the output tensor of a Merge layer.




```
from keras.layers import Input, Convolution2D
from keras.layers import merge
a = Input(shape=(3, 32, 32))
b = Convolution2D(3, 5, 5, border_mode='same')(a)
b = merge([a, b], mode='sum')
model = Model(input=a, output=b)
```

Important: Do not use Merge() (with capital M).

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Compiling a Model for Training

After we build a model, we need to prepare it for training:

- Choose a proper loss function for the task.
- Choose an optimization algorithm.

In Keras you need to call the `compile()` function:

```
from keras.models import Model
a = Input(shape=(3, 32, 32))
# Setup model...
b = # output tensor
model = Model(input=a, output=b)
model.compile(loss='mean_square_loss',
              optimizer='adam')
```

Fitting Model to a Static Dataset

Assume our training set is $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$.

Preparing the dataset:

- Combine all training data into X: $X[i, d_1, \dots, d_k] = x_i[d_1, \dots, d_k]$.
- Combine all label data into Y: $Y[i, d_1, \dots, d_k] = y_i[d_1, \dots, d_k]$.

Example: assume each example is a grayscale image of shape $(1, 16, 16)$, and each label a scalar of shape $(1,)$.
$\Rightarrow$ the shapes of X is $(n, 1, 16, 16)$, and of Y is $(n, 1)$.

Training model: (could take a very long time)

```
model.fit(X, Y, batch_size=100,
          nb_epoch=5, validation_split=0.2)
```

`batch_size` – number of samples in each mini-batch.
`nb_epoch` – number of times to go over entire training set.
`validation_split` – fraction of (X, Y) to be used as validation set. Keras will test on the validation set after every epoch.

Fitting Model to a Dynamically Generated Dataset

Shortcomings of a static dataset:

- Has to fit in memory.
- Any data preprocessing must be done ahead of time.
- Less convenient for functionally generated datasets

Solution: prepare mini-batches of data on-the-fly.

In Keras this is done with Python's Generators. **Reminder:**

```
# A generator outputting random numbers
def build_generator():
    while True:
        yield np.random.random()
# Create generator
gen = build_generator()
# Call next() to generate a new value
for i in range(5):
    print('%.2f' % next(gen), end=' ')
>> 0.04 0.61 0.18 0.05 0.81
```

Fitting Model to a Dynamically Generated Dataset (cont')

Requirements of generator to serve as a dataset for Keras:

- Yield each time a pair (X, Y).
- X is a numpy array holding a mini-batch of examples.
- Y is a numpy array holding the respective labels of X.

Code example:

```
# Assumes batch size of generators is 100
train_set = # generator for training set
valid_set = # generator for validation set
model.fit_generator(train_set,
    samples_per_epoch=10000, nb_epoch=5,
    validation_data=valid_set, nb_val_samples=1000)
```

`samples_per_epoch` – defines an "epoch" as this many examples.
`nb_epoch` – number of times to iterate over "epochs" as above.
`validation_data` – generator for validation set
`nb_val_samples` – how many samples to generate for each validation test.

Example Log of Training

Training Log:

```
Using TensorFlow backend.
Epoch 1/5
10000/10000 [==========================] - 179s - loss: 0.2052 - val_loss: 0.0924
Epoch 2/5
10000/10000 [==========================] - 169s - loss: 0.0823 - val_loss: 0.0533
Epoch 3/5
10000/10000 [==========================] - 171s - loss: 0.0521 - val_loss: 0.0348
Epoch 4/5
10000/10000 [==========================] - 173s - loss: 0.0320 - val_loss: 0.0220
Epoch 5/5
10000/10000 [==========================] - 165s - loss: 0.0219 - val_loss: 0.0107
```

Things to notice:

- Keras shows the *averaged* training loss over all iterations in the epoch.
- Validation loss is reported after the last iteration in the epoch.
- A good model will quickly converge on the training set.
- A good model will have validation loss close to training loss.

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Prediction with a Trained Model

Assuming we have a trained model, just call `predict(X)`.

```
model = # trained model
X = # multiple images of shape (N, 1, 28, 28)
Y = model.predict(X)
x = # a single image of shape (1, 28, 28)
y = model.predict(x[np.newaxis,...])[0]
```

Notice: `predict()` expects a “batch” of inputs. If we wish to predict a single input, we treat it as a batch of size 1.

Saving, Loading, and Copying Weights

- **Saving:**

```
model = # trained model
model.save_weights(path_to_filename)
```

- **Loading:**

```
model = # untrained model before compiling it!
model.load_weights(path_to_filename)
```

- **Copying:**

```
source_model = # trained model
target_model = # untrained model before compiling it!
target_model.set_weights(source_model.get_weights())
```

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture

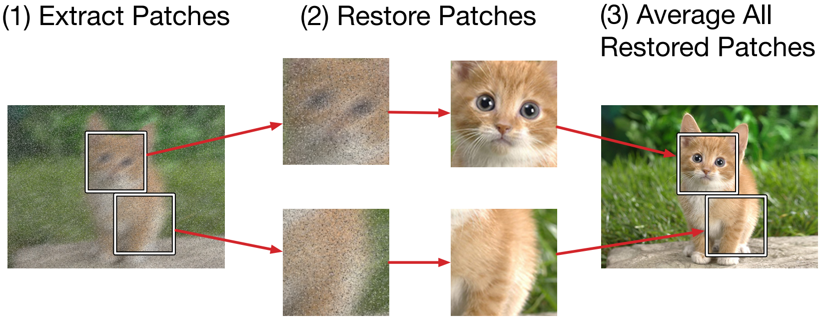
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model

3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Patch-Based Image Restoration



Notice: We should extract densely overlapping patches, so artifacts will be averaged out when we reassemble the image.

Learning from "Good" Image Patches

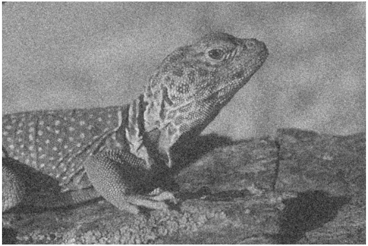
- Humans cannot correct corrupted images in the wild.
- Start from "good" images, and simulate a corruption: noise, blur, ...
  Important: we must use randomly sampled corruptions type!
- Build a dataset of small "good" and "bad" pair of patches.
- Train a network to map "bad" patches to "good patches".
- Use network for the patch-based's restoration step.

ConvNet "Trick": If we enlarge the input of a fully convolutional network, its output scale as well. It can be shown that the new output is an approximation of the patch averaging step.

Outline

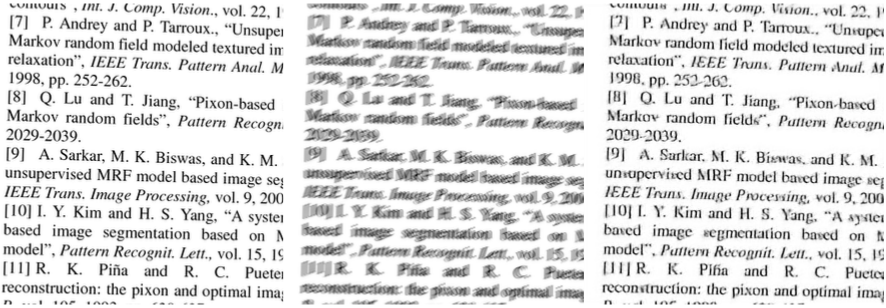
1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Image Denoising



Simulated corruption: randomly add gaussian noise.

Image Deblurring



Simulated Corruption: randomly apply motion blur.

Outline

1. Recap on Neural Networks
   - Learning
   - Neural Networks
   - ResNet Architecture
2. Keras Framework
   - Model Definition
   - Training a Model
   - Prediction with a Trained Model
3. Ex5: Image Restoration
   - General Workflow
   - Applications to Denoising and Deblurring

Good Luck!