# Computer Vision – TP10
# Deep Learning Resources and Examples

***Miguel Coimbra, Francesco Renna***

U. PORTO FC

# Outline

- Techniques to reduce overfitting
- Deep learning examples

# Outline

- Techniques to reduce overfitting
- Deep learning examples

# Generalization

- Deep neural network => high number of parameters (high complexity)
- They require large training datasets

- What can we do when we do not have a large annotated training dataset?

# Regularization

- “Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.”

# Weight regularization

- Reduce the generalization error by imposing constraints on the weights

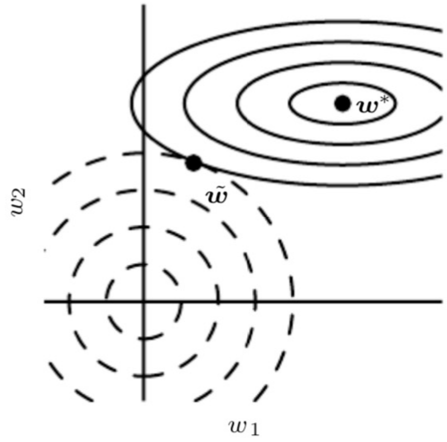
- Modifies the loss function in order to force some structure on the learned weights

$$L'(\theta, \{(x_i, y_i)_i\}) = L(\theta, \{(x_i, y_i)_i\}) + \gamma\Omega(\theta)$$

- Different $\Omega$, different effect on the weights

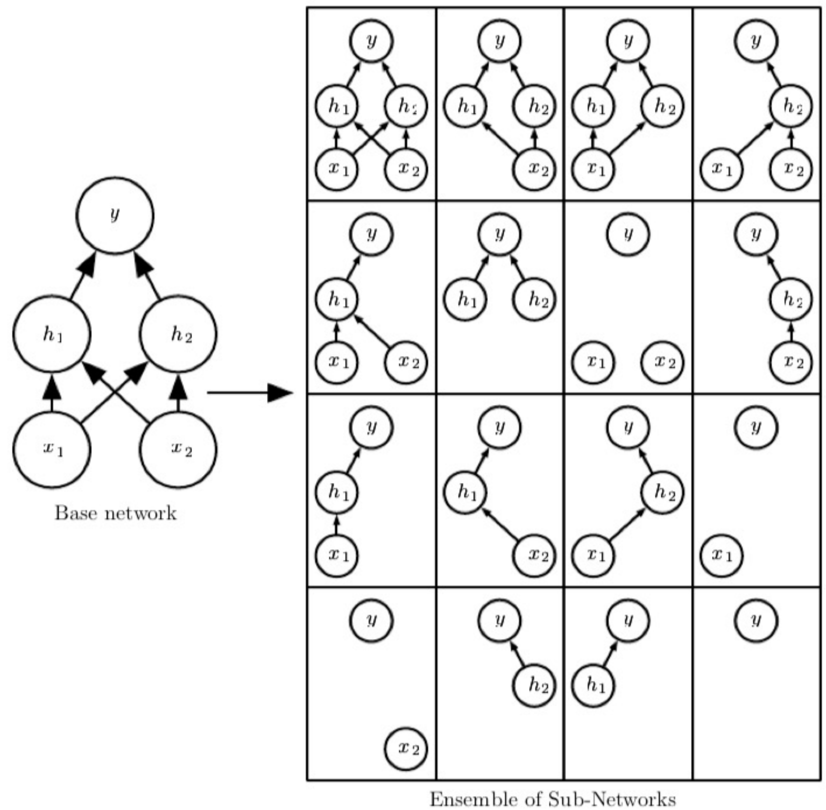
# Weight decay

- Weight decay: $\Omega(\theta) = \|\theta\|_2^2$
  - Drives the weights closer to the origin
  - Weight components that do not impact significantly the loss function are decayed



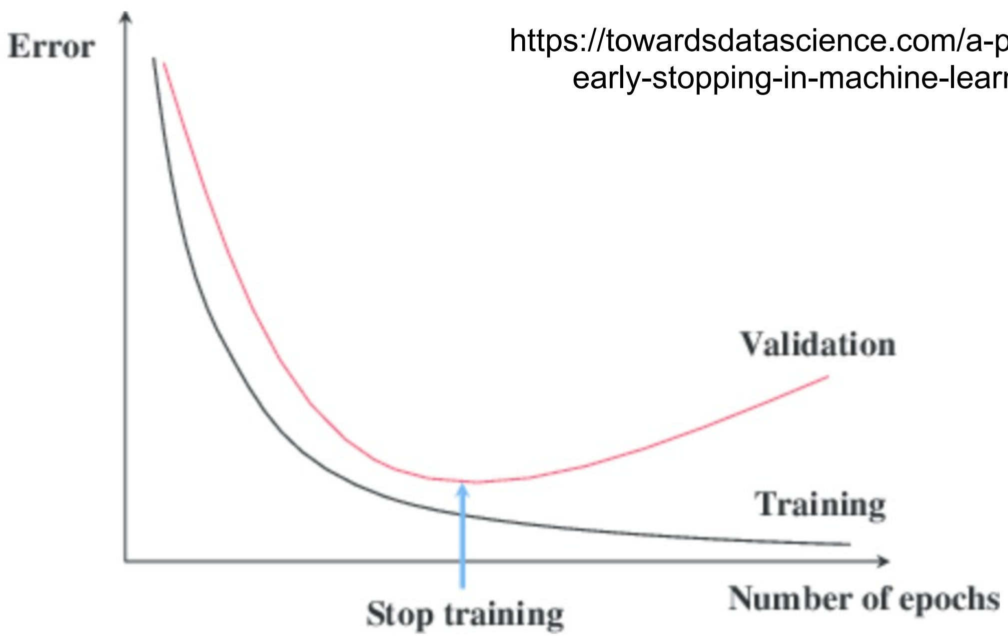
# Dropout

- During training, randomly switch off a fraction of the input or hidden units
- It avoids giving too much relevance to some training features
- It approximates bagging and ensemble learning over all sub-models (Monte-Carlo sampling)



# Early stopping

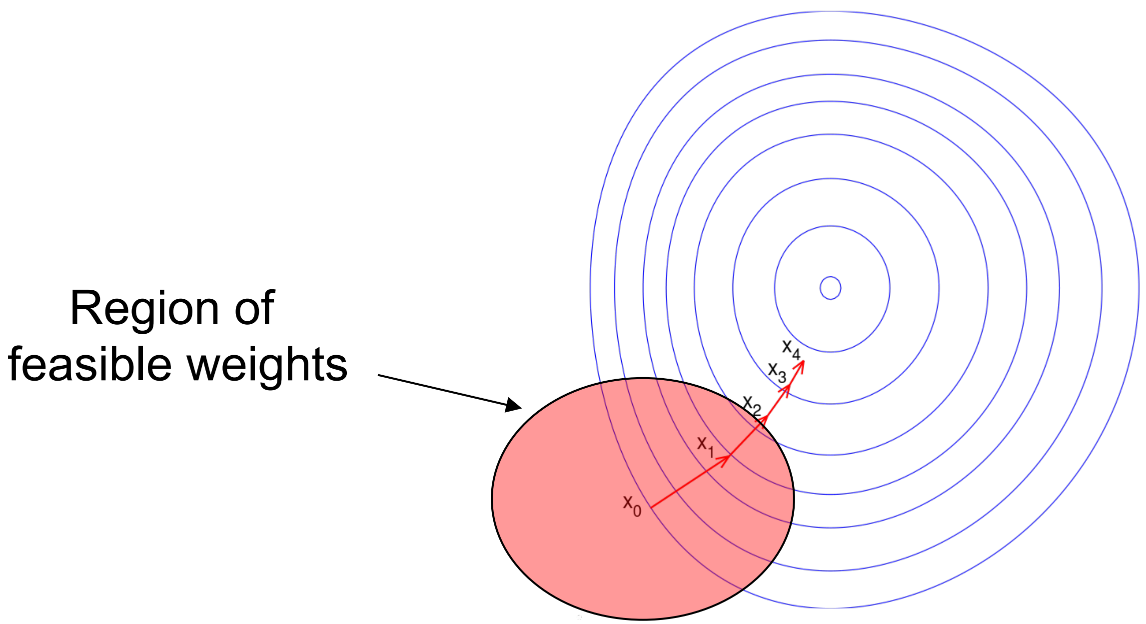
- Retain the model which performs best on the validation set (hopefully, test set too)




https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd

# Early stopping

- Regularization effect: constraint on the number of training steps

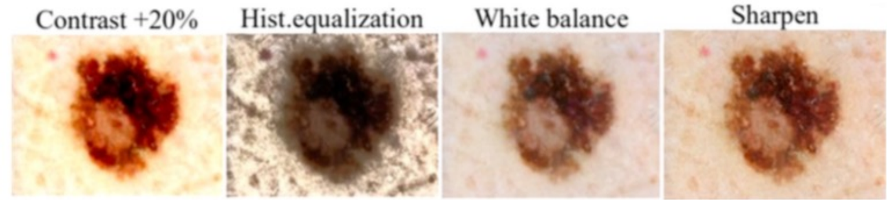


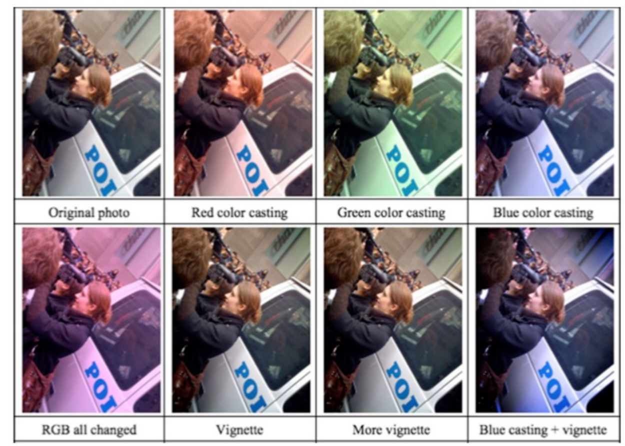
# Data augmentation

- Create fake data and add it to the **training dataset** (only training!)
- Especially useful for imaging data
- New data created from transformations of existing training data:
  - Different transformations may be more meaningful in different domains
  - A transformation should not change class meaning

# Data augmentation

- Transformations:
  - Translating
  - Rotating
  - Cropping
  - Flipping
  - Color space
  - Adding noise
  - Image mixing
  - Generative Adversarial Networks (GANs)
  - Etc.



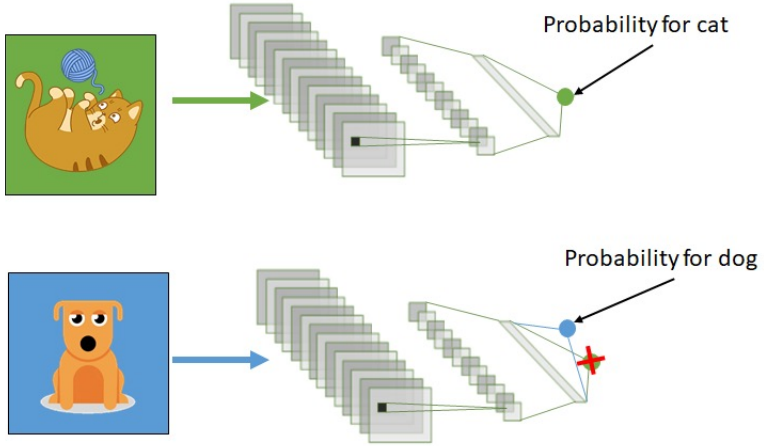





Shorten C, Khoshgoftaar TM. A survey on image data augmentation for deep learning. Journal of Big Data. 2019 Dec;6(1):1-48.

# Transfer learning

- Main idea:
  - Features to perform a task T1 may be relevant and useful for a different task T2




https://towardsdatascience.com/transfer-learning-3e9bb53549f6

# Transfer learning

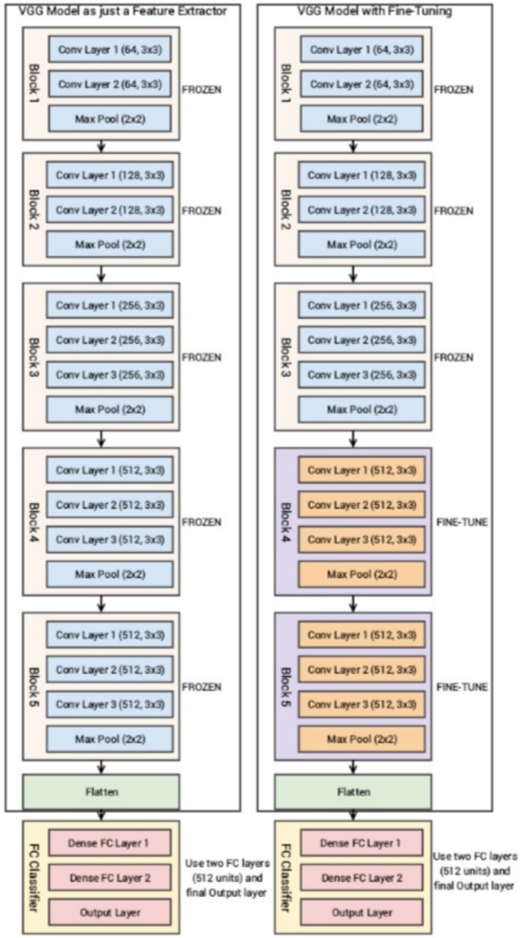
- When is it useful:
  - Reduced number of training samples for the considered task
  - Large number of training samples for a related task
  - Low-level features could be common to both tasks!

- Example:
  - Image classification
  - NNs pre-trained on the ImageNet dataset (~14 million images, ~20,000 categories)

# Transfer learning schemes

https://towardsdatascience.com/a-comprehensive-hands-on-guide-to- transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

- Feature extraction:
  - Keep convolutional layers frozen
  - Pre-trained networks works as feature extractor
  - Train fully connected/classification layers



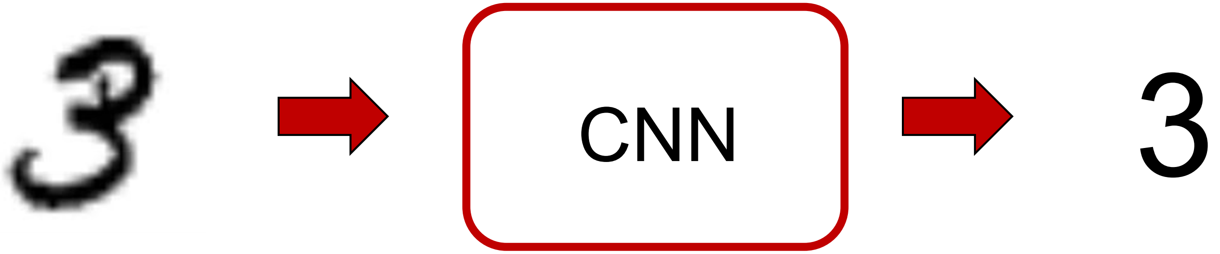

- Fine-tuning:
  - Use pre-trained weights as starting point for training
  - Can keep frozen first convolutional layers (mostly edge/geometry detectors)

# Outline

- Techniques to reduce overfitting
- Deep learning examples

# Example in Keras

- Task:
  - Classify hand-written digits
- Model:
  - Convolutional neural network
- Full code available at:
  - https://keras.io/examples/vision/mnist_convnet/



# Setup

- Import useful libraries

Useful for processing matrix-like data in Python, and much more…

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

# Prepare the data

- Model/data parameters

```
num_classes = 10
input_shape = (28, 28, 1)
```

Black and white images = 1 input channel

- Train and test data

```
(x_train, y_train), (x_test, y_test) =
keras.datasets.mnist.load_data()
```

Training labels

Training images: dimensions (60000, 28, 28, 1), “channel-last” ordering

# Prepare the data

- Scale images to [0,1] range

```
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
```

- One-hot encoding

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

E.g., from “3” to $[0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$

# Build the model - I

- Feature extraction

Number of feature maps

Stride = 1 if omitted

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
```

# Build the model - II

- Classification

Reshape feature maps into a vector

50% of nodes are used at during training

```
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

Provides a description of the model
with number of parameters

# Build the model - III

- Model summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

# Train the model

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
```

Gradient-based optimizer

10% of the training data is used for validation

# Evaluate the model

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

- Results

```
Test loss: 0.023472992703318596
Test accuracy: 0.9912999868392944
```

# Weight decay in Keras

- Added to each layer

```
l2_model = tf.keras.Sequential([
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001),
                 input_shape=(FEATURES,)),
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(512, activation='elu',
                 kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1)
])
```

# Early stopping in Keras

```
tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', min_delta=0, patience=0, verbose=0,
    mode='auto', baseline=None, restore_best_weights=False
)
```

- Then call “callbacks” into model.fit()

- Patience = number of epochs with no improvement after which training is stopped
- Min_delta = minimum change
- Restore_best_weights = keep best model

# Transfer learning Keras

- Load a pre-trained model

```
# Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

- Feature extractor: `base_model.trainable = False`
- Fine-tuning: `base_model.trainable = True`
- Then add a classification head

# Resources

- I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Cambridge: MIT press, 2016.
  - Chapter 7 – “Regularization for deep learning”
- https://www.tensorflow.org/tutorials/keras
- https://www.tensorflow.org/tutorials/images