Deep Learning par la Pratique

Convolutional Neural Networks et Keras

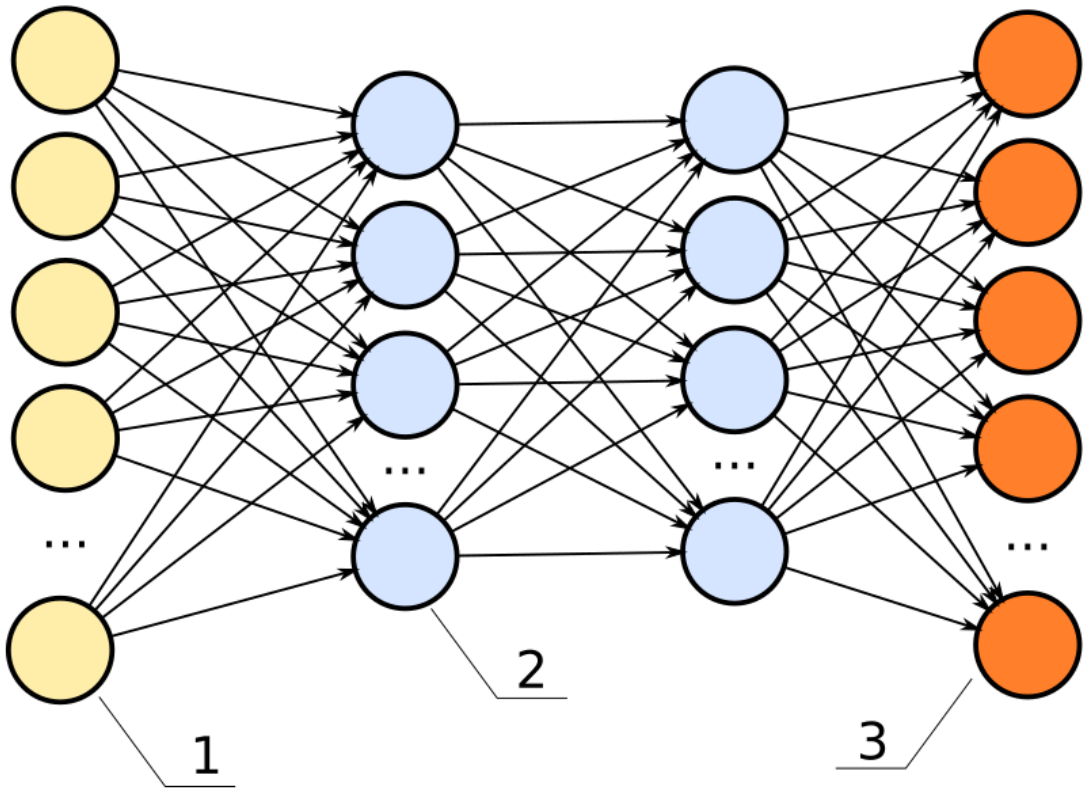
GIRAUD François-Marie

https ://www.orsys.fr/

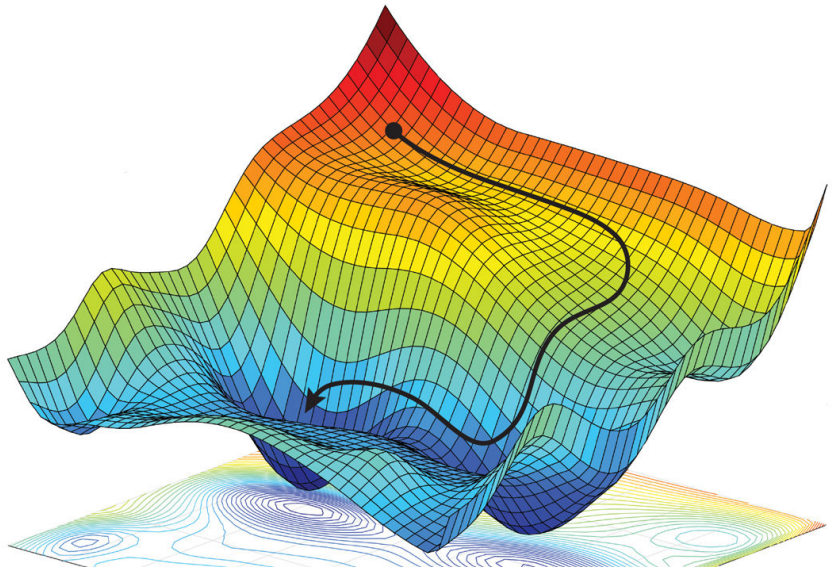
Deep Learning par la Pratique

Cours 2 : Keras et Convolutional Neural Networks

Rappels

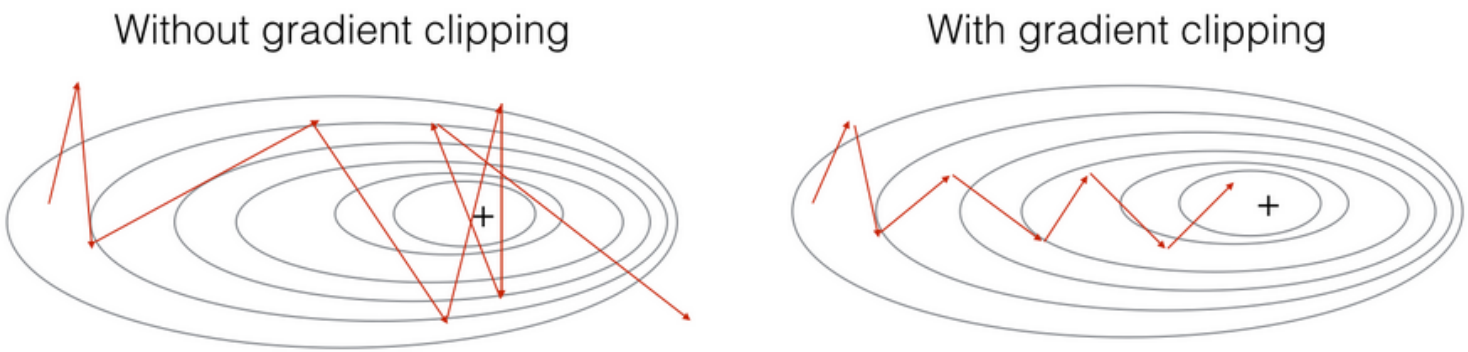


Rappels



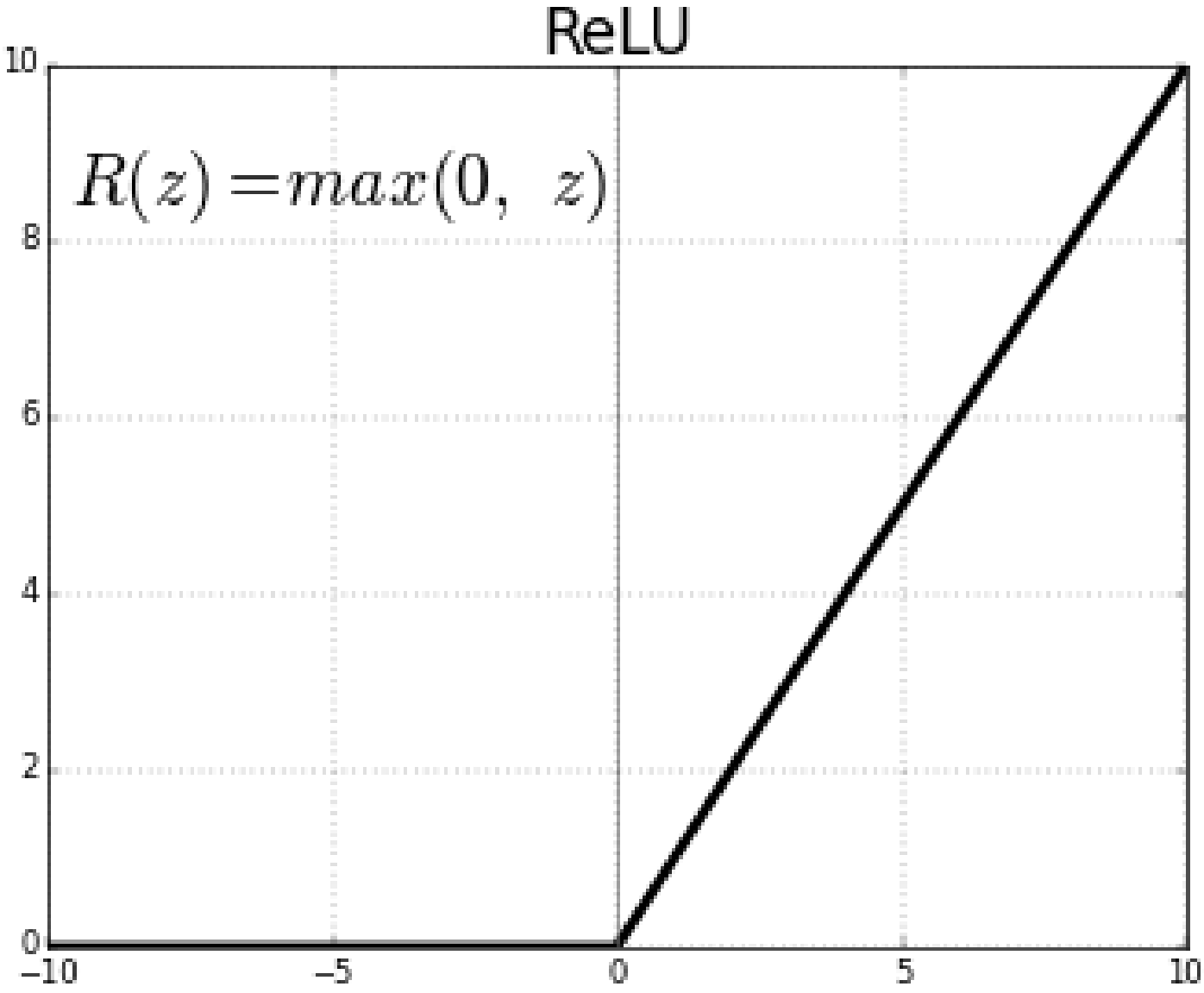
Rappels

Exploding Gradient $\Rightarrow$ Gradient clipping



Rappels

Vanishing Gradient ⇒ utilisation de ReLu plutôt que les sigmoïdes

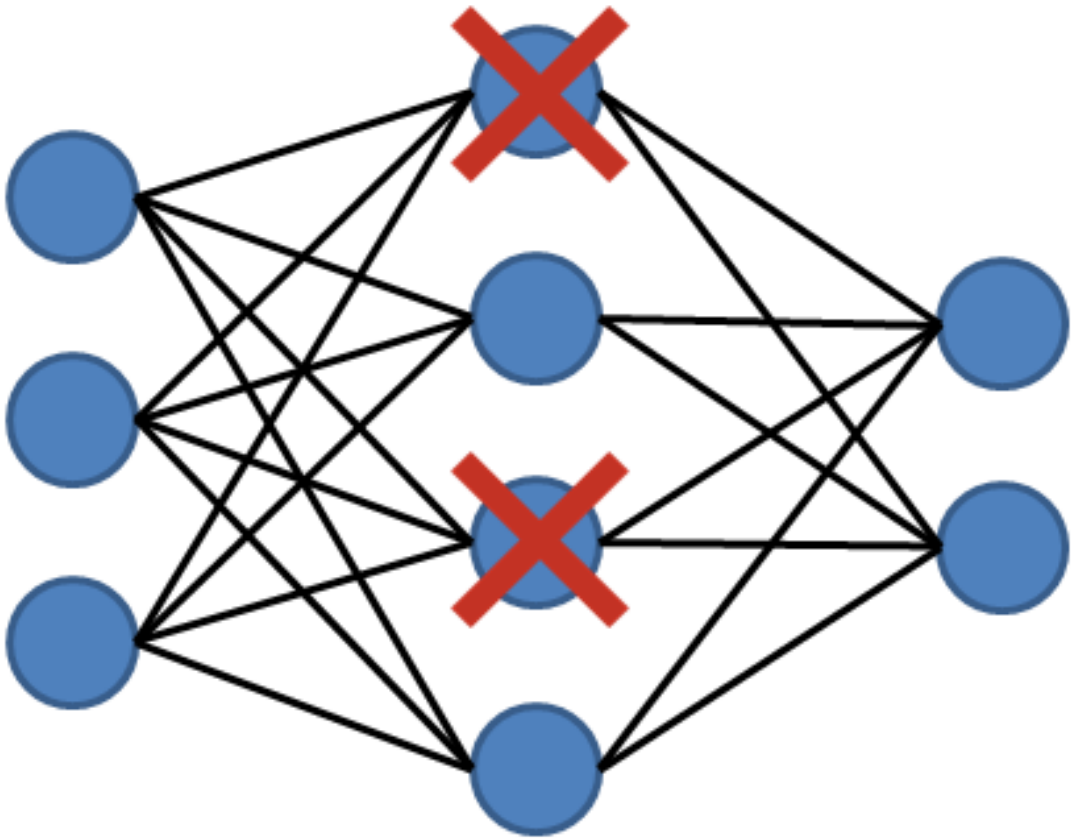


Rappels

Optimisateur plus rapide que SGD

- Root Mean Square Propagation (RMSProp)
- Adapative Gradient Algorithm (AdaGrad)
- Adaptative Moment Estimation (Adam) $\Leftarrow$
- ... AdaBound (2019) ?

Rappels

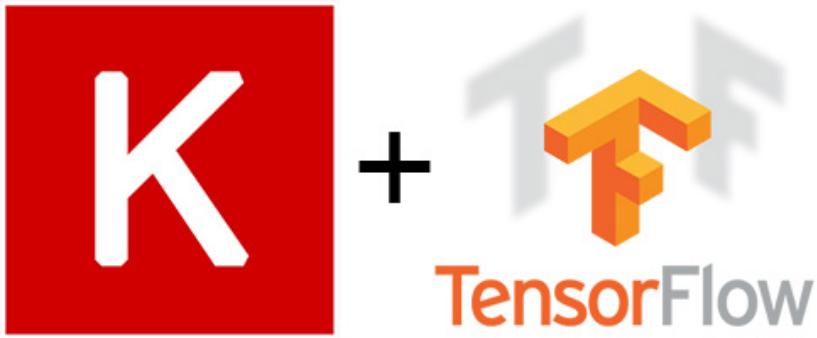


Deep Learning par la Pratique

Introduction à Keras

Introduction à Keras

Keras est une API de haut niveau qui permet de prototyper des réseaux de neurones de toute sorte.



Introduction à Keras

User-friendly :

- Interface simple
- Accès facilité aux métriques d'évaluation

Introduction à Keras

Modulaire :

- Les réseaux se 'branchent' facilement les uns avec les autres
- Tous les réseaux se configurent facilement

Introduction à Keras

Etat de l'art :

- Les modèles et optimiseurs pertinents sont rapidement ajouté à Keras
- Reproduction facile de résultats récents

Introduction à Keras

Facile de dévellopper de nouvelles :

- Couche de réseau (Layers)
- Fonction de perte (Loss)
- Métriques d'évaluation

Réseau de neurones en Keras

```
from tensorflow.keras import layers as kl

model = tf.keras.Sequential()
# Il est impératif de spécifier input_shape pour la première couche :
model.add(layers.Dense(64, activation='sigmoid', input_shape=(32,)))
# Ajouter une autre couche :
model.add(layers.Dense(64, activation='relu'))
# Une dernière couche de classification avec softmax ;
model.add(layers.Dense(10, activation='softmax'))
```

Options de configuration des Layers

```python
# Utiliser une fonction d'activation :
layers.Dense(64, activation='sigmoid')
# Ou:
layers.Dense(64, activation=tf.keras.activations.sigmoid)

# L1 regularization des poids de la matrice :
layers.Dense(64, kernel_regularizer=tf.keras.regularizers.l1(0.01))

# L2 regularization des biais:
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# Initialisation des poids avec une matrice orthogonale :
layers.Dense(64, kernel_initializer='orthogonal')

# Initialisation des biais avec une constante :
layers.Dense(64, bias_initializer=tf.keras.initializers.Constant(2.0))
```

Optimiseur, Loss et Métrique d'évaluation

```
#Compilation du modèle
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
                            loss='categorical_crossentropy',
                            metrics=['accuracy'])
# Configure a model for mean-squared error regression.
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='mse',       # mean squared error
              metrics=['mae'])  # mean absolute error
# Configure a model for categorical classification.
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.01),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

Affichage du modèle

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                2112
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 6,922
Trainable params: 6,922
Non-trainable params: 0
_________________________________________________________________
```

Apprentissage

```
import numpy as np

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.fit(data, labels, epochs=10, batch_size=32)
```

Apprentissage

```
1  import numpy as np
2
3  data = np.random.random((1000, 32))
4  labels = np.random.random((1000, 10))
5
6  val_data = np.random.random((100, 32))
7  val_labels = np.random.random((100, 10))
8
9  model.fit(data, labels, epochs=10, batch_size=32,
10           validation_data=(val_data, val_labels))
```

Apprentissage

```
import numpy as np

data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.fit(data, labels, epochs=10, batch_size=32, validation_split=0.3)
```

Apprentissage

```
# Instantiates a toy dataset instance:
dataset = tf.data.Dataset.from_tensor_slices((data, labels))
dataset = dataset.batch(32)

model.fit(dataset, epochs=10)
#Ou
model.fit(dataset, epochs=10, validation_split=0.3)
```

Evaluation

```
# With Numpy arrays
data = np.random.random((1000, 32))
labels = np.random.random((1000, 10))

model.evaluate(data, labels, batch_size=32)

# With a Dataset
dataset = tf.data.Dataset.from_tensor_slices((data, labels))
dataset = dataset.batch(32)

model.evaluate(dataset)
```

```
1000/1 [==============================] - 0s 70us/sample - loss: 199785.4507
                                          - categorical_accuracy: 0.0990
32/32 [==============================] - 0s 2ms/step - loss: 200361.9849
                                        - categorical_accuracy: 0.0990
```

Evaluation

```
result = model.predict(data, batch_size=32)
print(result.shape)
```

```
(1000, 10)
```

Sauvegarde de la configuration d'un modèle

```
1  import json
2  import pprint
3  # Serialize a model to JSON format
4  json_string = model.to_json()
5  pprint.pprint(json.loads(json_string))
6  # Load a model configuration
7  fresh_model = tf.keras.models.model_from_json(json_string)
```

Sauvegarde des poids d'un modèle

```
1  # Save weights to a HDF5 file
2  model.save_weights('my_model.h5', save_format='h5')
3
4  # Restore the model's state
5  model.load_weights('my_model.h5')
```

Sauvegarde d'un modèle complet

```
1  # Save entire model to a HDF5 file
2  model.save('my_model.h5')
3
4  # Recreate the exact same model, including weights and optimizer.
5  model = tf.keras.models.load_model('my_model.h5')
```

Callbacks

```
callbacks = [
    # Interrompt l'apprentissage si `val_loss`
    #ne s'améliore plus depuis 2 epochs
    tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss'),
    # Sauvegarde le meilleur model
    tf.keras.callbacks.ModelCheckpoint(filepath='models/bestmodel.hdf5'
                                       verbose=1, save_best_only=True)
]
model.fit(data, labels, batch_size=32, epochs=5,
          callbacks=callbacks,
          validation_split=0.2)
```

Création d'un Layer

```
class MyLayer(layers.Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(MyLayer, self).__init__(**kwargs)
    def build(self, input_shape):
        # Create a trainable weight variable for this layer.
        self.kernel=self.add_weight(name='kernel',
                                    shape=(input_shape[1], self.output_dim)
                                    initializer='uniform',
                                    trainable=True)
    def call(self, inputs):
        return tf.matmul(inputs, self.kernel)
    def get_config(self):
        base_config = super(MyLayer, self).get_config()
        base_config['output_dim'] = self.output_dim
        return base_config
    @classmethod
    def from_config(cls, config):
        return cls(**config)
```

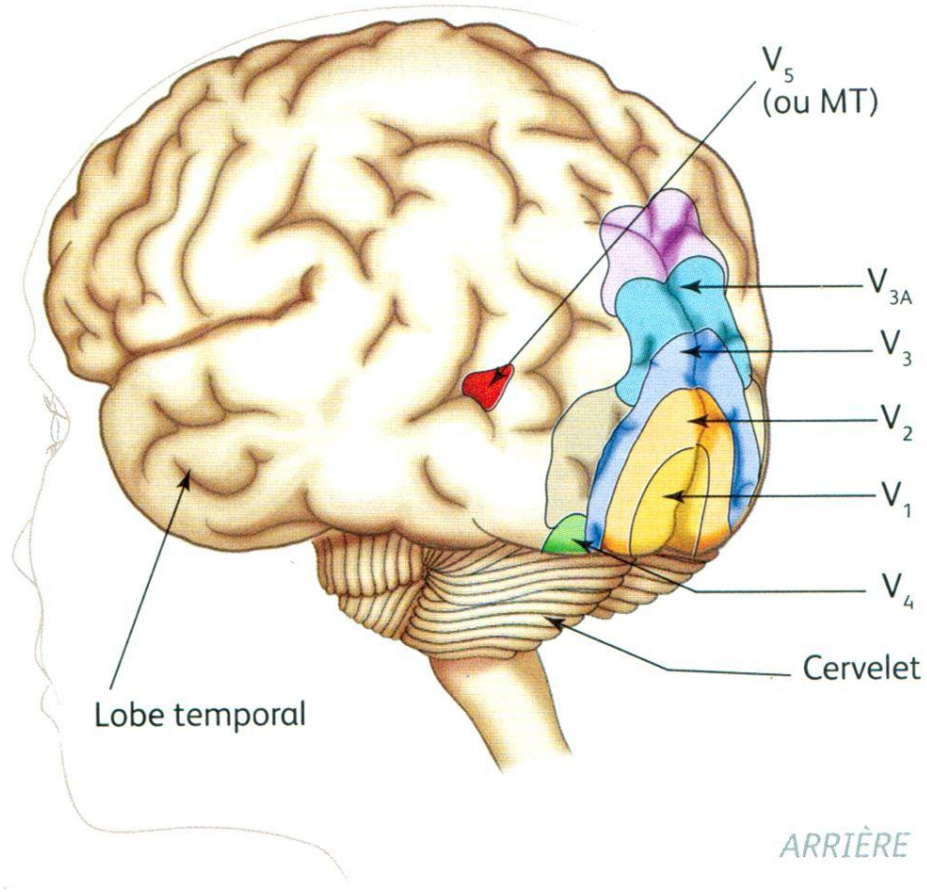
Utilisation de notre nouveau Layer

```
model = tf.keras.Sequential()
model.add(MyLayer(10))
model.add(tf.layers.Activation('softmax'))

# The compile step specifies the training configuration
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Trains for 5 epochs.
model.fit(data, labels, batch_size=32, epochs=5)
```

Deep Learning par la Pratique

Convolutional Neural Networks

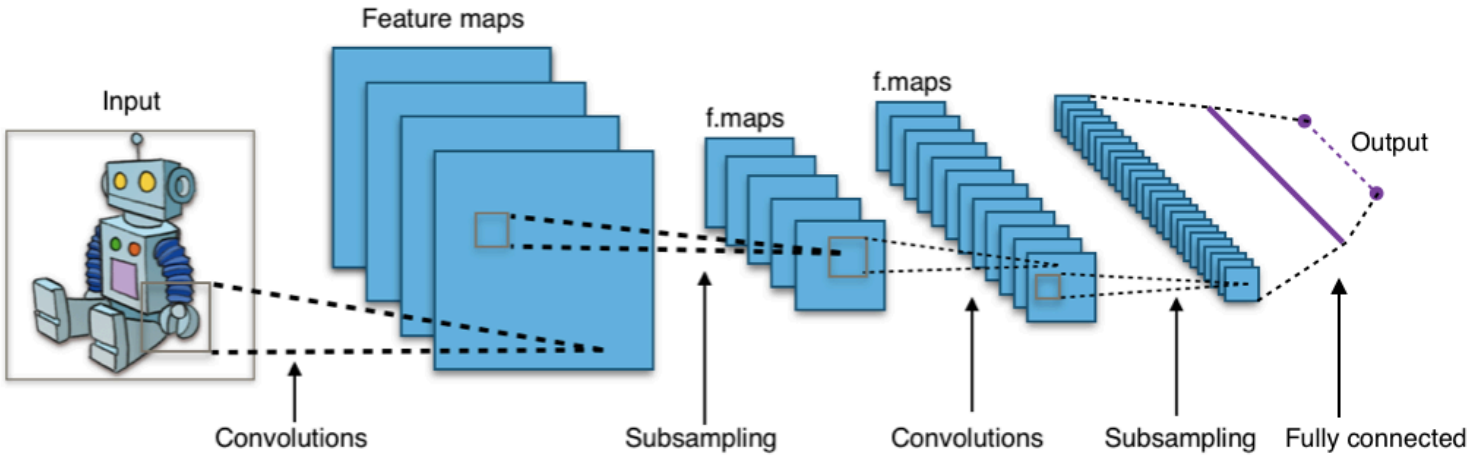
Architecture du cortex visuel



Connexion “en série” de couches

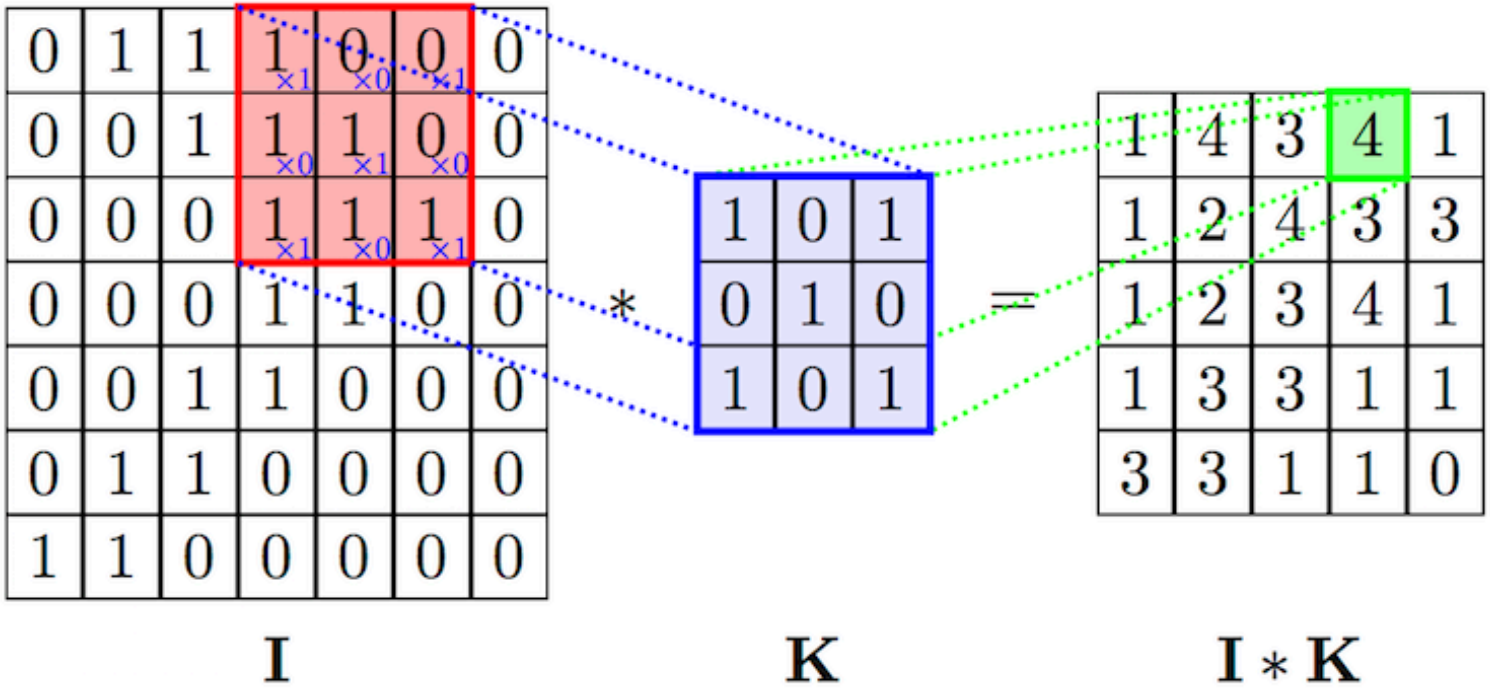
- V1 $\approx$ orientations de lignes
- V2 $\approx$ formes, tailles, couleurs
- V3 $\approx$ aide à la motricité
- V4 $\approx$ reconnaissance d’objets simples
- V5 $\approx$ vitesse des objets
- ...

Convolutional Neural Networks

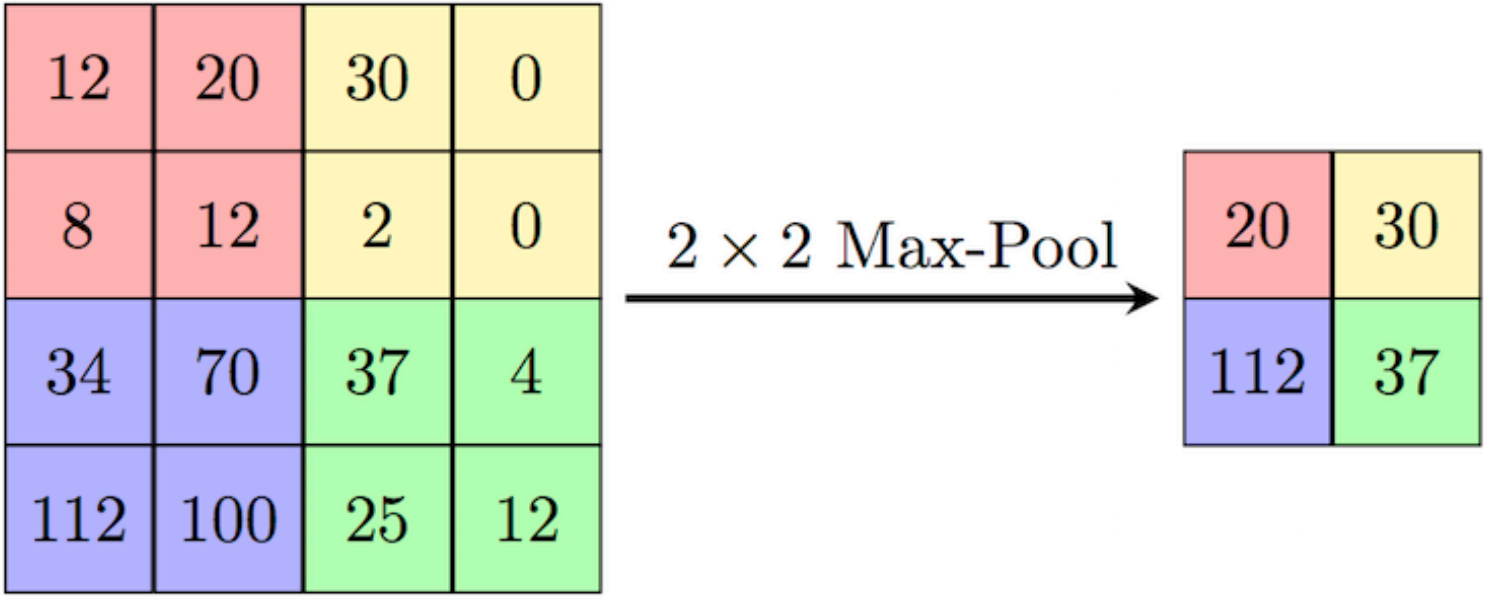


Convolutional Neural Networks

L'opérateur de **convolution**



L'opérateur de **pooling**



Deep Learning par la Pratique

Implémentation d'un CNN avec Keras

Définition d'un modèle CNN

```
model = Models.Sequential()
#Un premier Layer de 10 convolutions de 3x3 pixels
model.add(Layers.Conv2D(10,kernel_size=(3,3),activation='relu',
                        input_shape=(150,150,3)))
#Un Layer "max pooling"
model.add(Layers.MaxPool2D(3,3))
#Un Layer "Flatten"
model.add(Layers.Flatten())
#Un Layer "Dense" avec 6 sorties et un softmax
model.add(Layers.Dense(6,activation='softmax'))
#Compilation du modèle avec la définition de la loss
model.compile(optimizer=Optimizer.Adam(lr=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Affichage du modèle

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 10)      280
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 10)        0
_________________________________________________________________
flatten (Flatten)            (None, 24010)             0
_________________________________________________________________
dense (Dense)                (None, 6)                 144066
=================================================================
Total params: 144,346
Trainable params: 144,346
Non-trainable params: 0
_________________________________________________________________
```

Avez-vous des questions ?

Deep Learning par la Pratique

Travaux Pratiques : CNN et Keras

Travaux Pratiques

Keras Tutorial

Travaux Pratiques

Keras-CNN-landscape

Travaux Pratiques

Keras-CNN-emotions