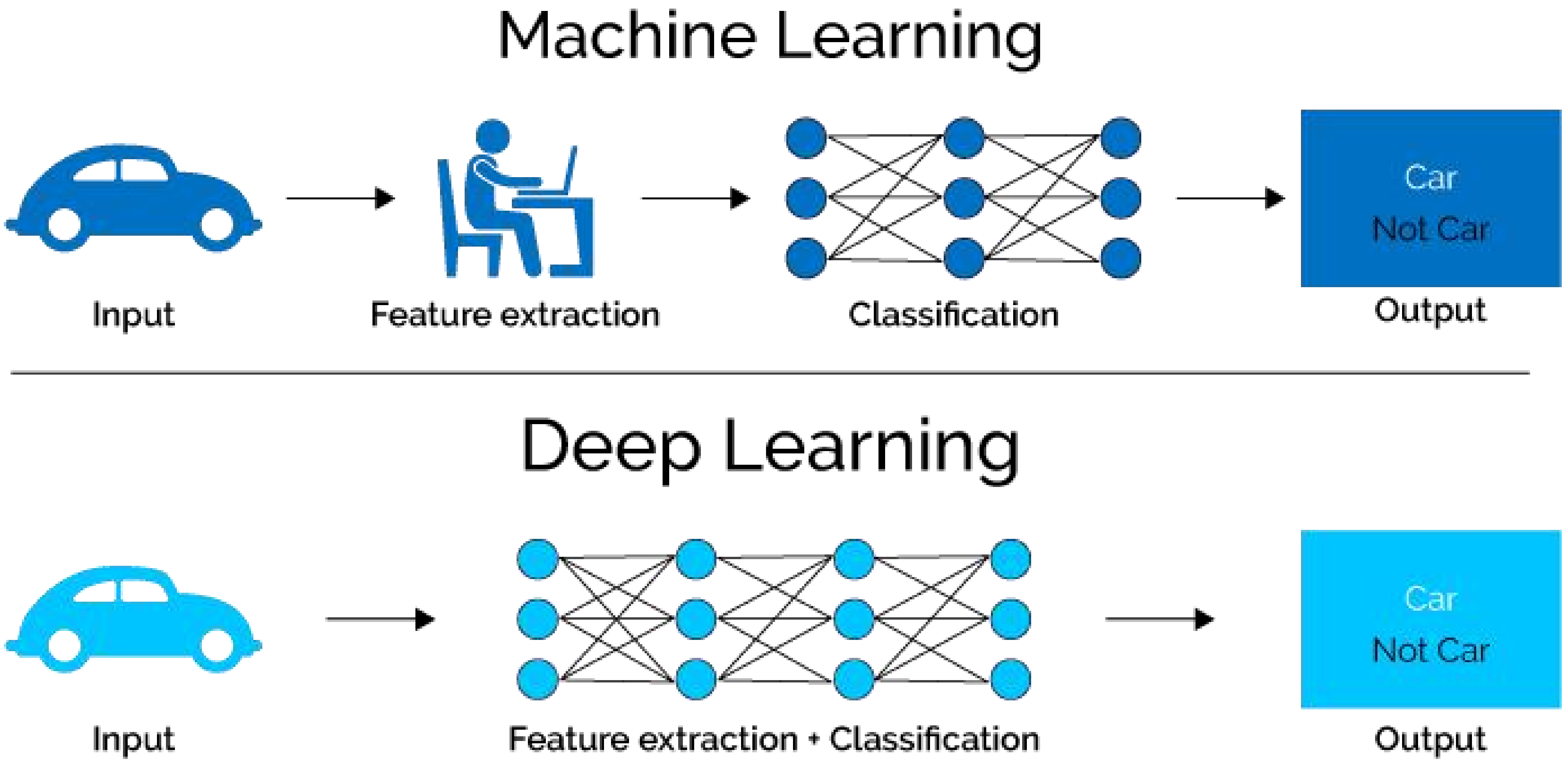
Build your own Convolutional Neural Network (CNN)

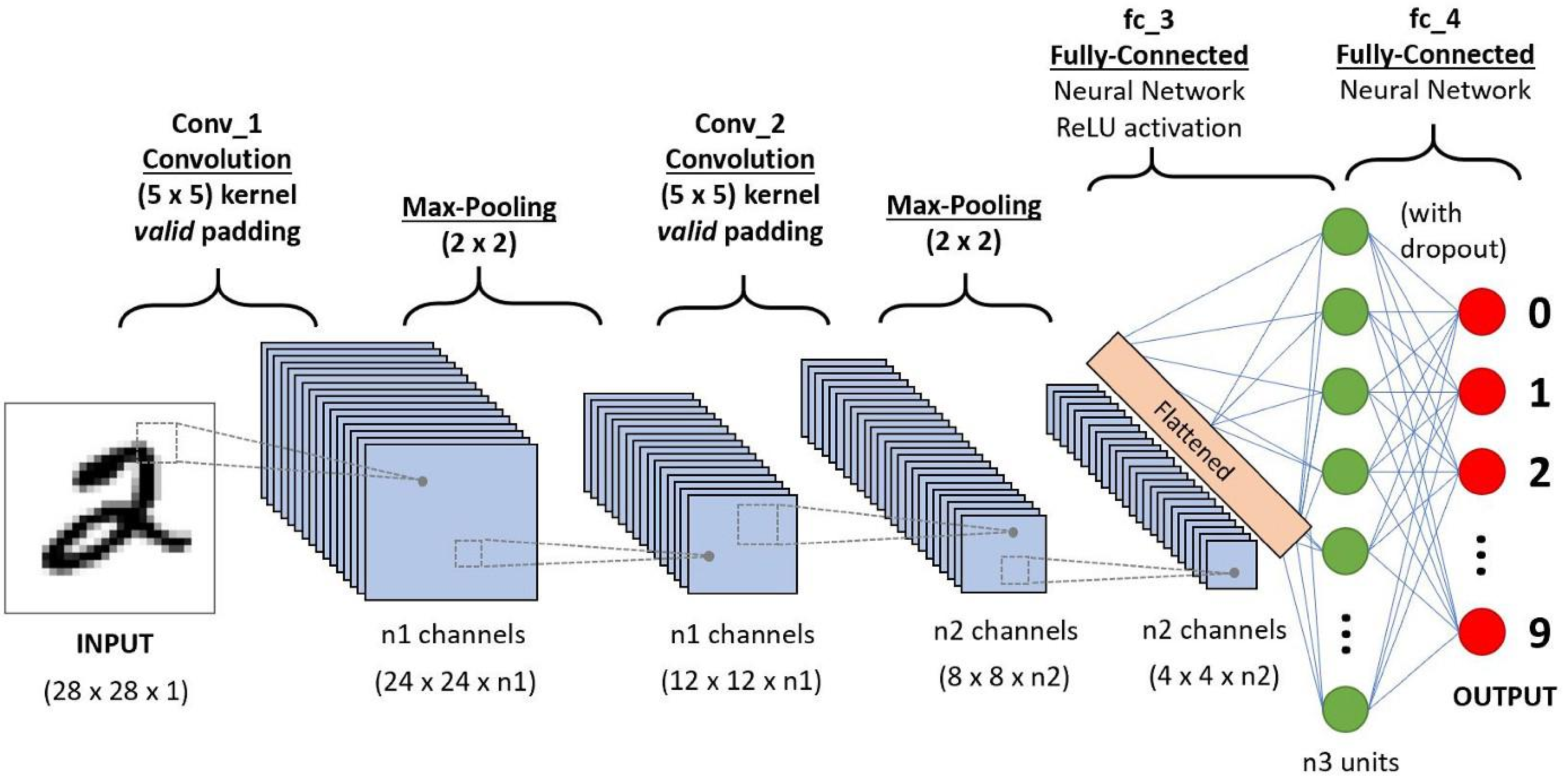
Keras & TensorFlow

Hichem Felouat
hichemfel@gmail.com

Machine Learning VS Deep Learning

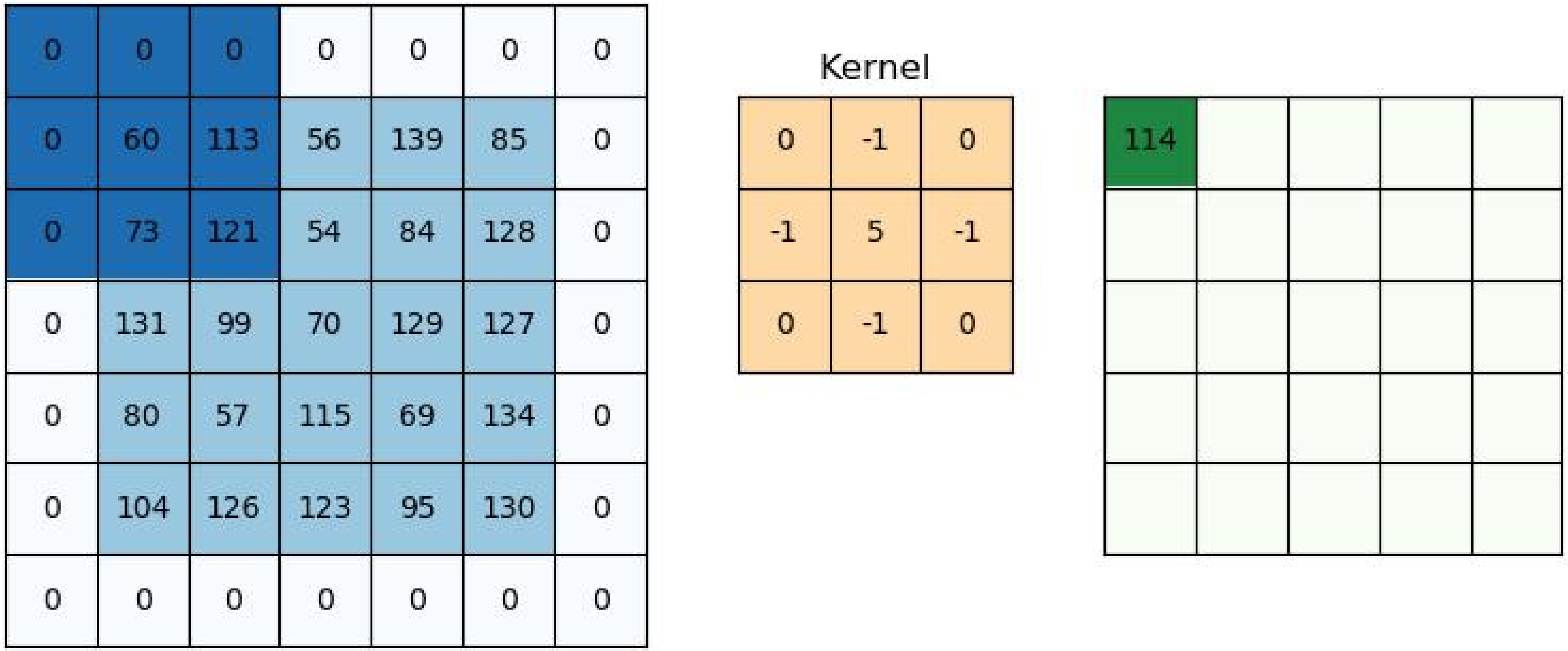


CNN Architecture



Convolutional Layers

- **Padding ?**
- **Stride ?**

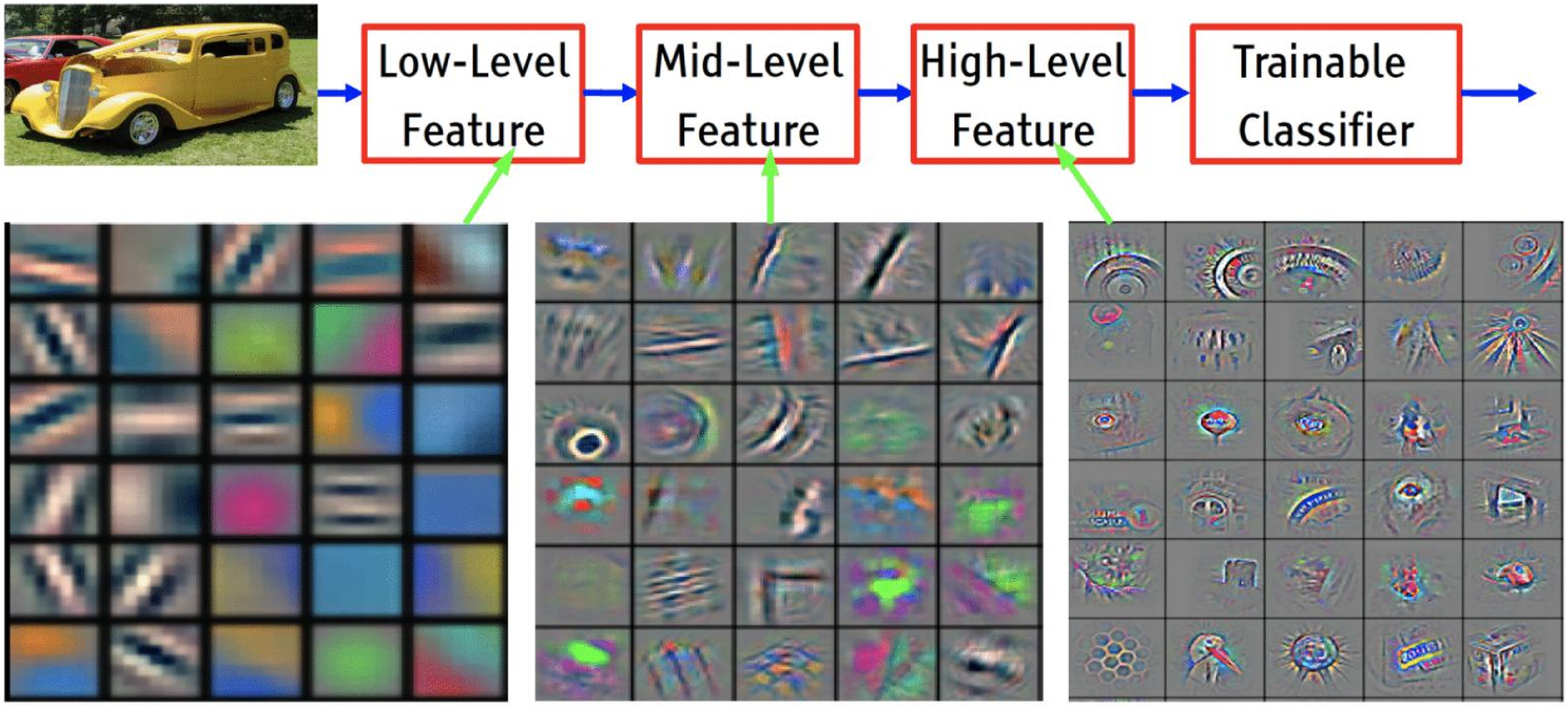


Convolutional Layers

This architecture allows the network to **concentrate on small low-level features** in the first hidden layer, then assemble them into **larger higher-level features** in the next hidden layer, and so on.

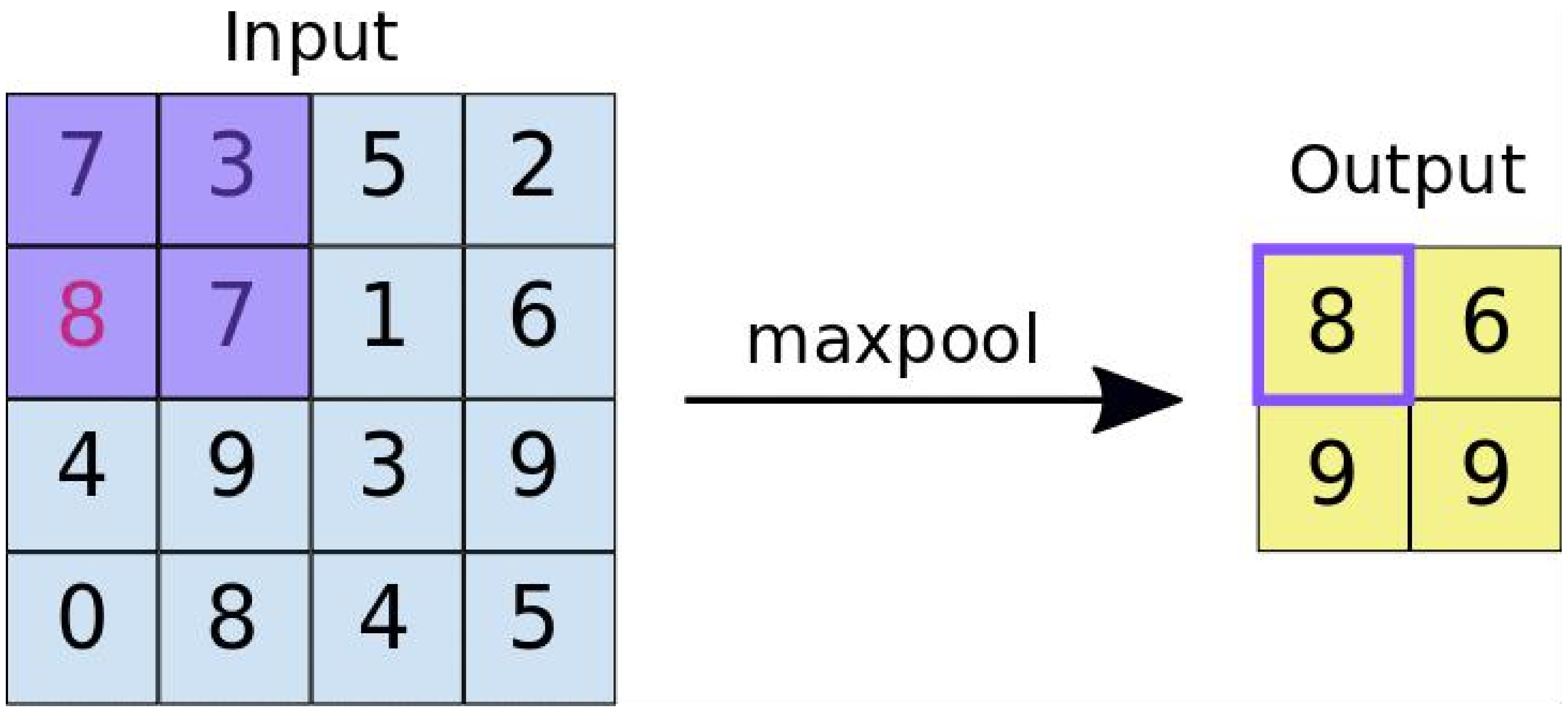
```
# padding : same or valid
conv = keras.layers.Conv2D(filters=32, kernel_size=3,
                           strides=1, padding="same", activation="relu")
```

Convolutional Layers



Pooling Layers

- **Average pooling ?**



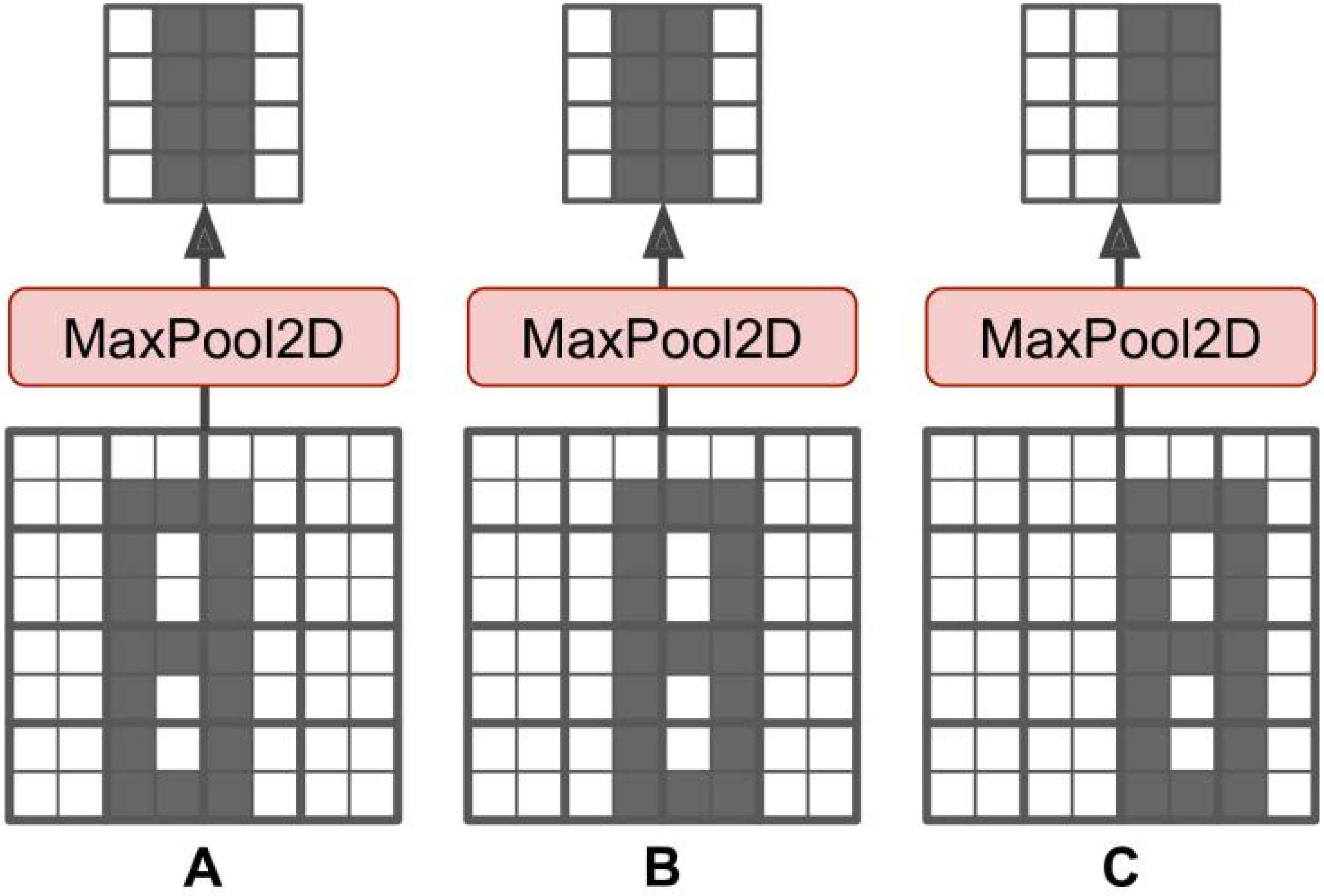
Pooling Layers

- Reduce the computational load, memory usage, and the number of parameters (limiting the risk of overfitting).

- Max pooling layer also introduces some level of invariance to small translations.

```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

Pooling Layers



Fully Connected Layers

Typical classification model architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
input neurons	One per input feature	One per input feature	One per input feature
hidden layers + neurons per hidden layer	Depends on the problem	Depends on the problem	Depends on the problem
output neurons	1	1 per label	1 per class
Hidden activation	ReLU (relu)	ReLU (relu)	ReLU (relu)
Output layer activation	Logistic (sigmoid)	Logistic (sigmoid)	Softmax (softmax)
Loss function	Cross entropy	Cross entropy	Cross entropy

Binary classification : binary_crossentropy - categorical_crossentropy
Multiclass classification : sparse_categorical_crossentropy

Avoiding Overfitting

1) Try tuning model hyperparameters such as (**the number of layers**, **the number of neurons per layer**, and the **types of activation functions** to use for each hidden layer.

2) Try tuning other hyperparameters, such as **the number of epochs** and **the batch size**.

3) **Reusing parts** of a pretrained network (possibly built on an auxiliary task or using unsupervised learning).

4) **class_weight**

Avoiding Overfitting

- Applying a good initialization strategy for the connection **weights**
- kernel_initializer="he_uniform" or "he_normal", "glorot_uniform"

```
keras.layers.Dense(10, activation="relu", kernel_initializer="he_normal")
```

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Abstract

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent

learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical success in vision (Ranzato et al., 2007; Larochelle et al., 2007; Vincent et al., 2008) and natural language processing (NLP) (Collobert & Weston, 2008; Mnih & Hinton,

Avoiding Overfitting

Batch Normalization: This operation simply zero-centers and normalizes each input, then scales and shifts the result (before or after the activation function of each hidden layer).

```
model = keras.models.Sequential([
   keras.layers.Flatten(input_shape=[28, 28]),
   keras.layers.BatchNormalization(),
   keras.layers.Dense(300, activation="elu", kernel_initializer="he_normal"),
   keras.layers.BatchNormalization(),
   keras.layers.Dense(100, activation="elu", kernel_initializer="he_normal"),
   keras.layers.BatchNormalization(),
  keras.layers.Dense(10, activation="softmax")
])
```

Avoiding Overfitting

Using faster optimizer (Gradient Descent, Momentum Optimization, Nesterov Accelerated Gradient, AdaGrad, RMSProp, Adam, Nadam).

Workshop track - ICLR 2016

INCORPORATING NESTEROV MOMENTUM INTO ADAM

Timothy Dozat
tdozat@stanford.edu

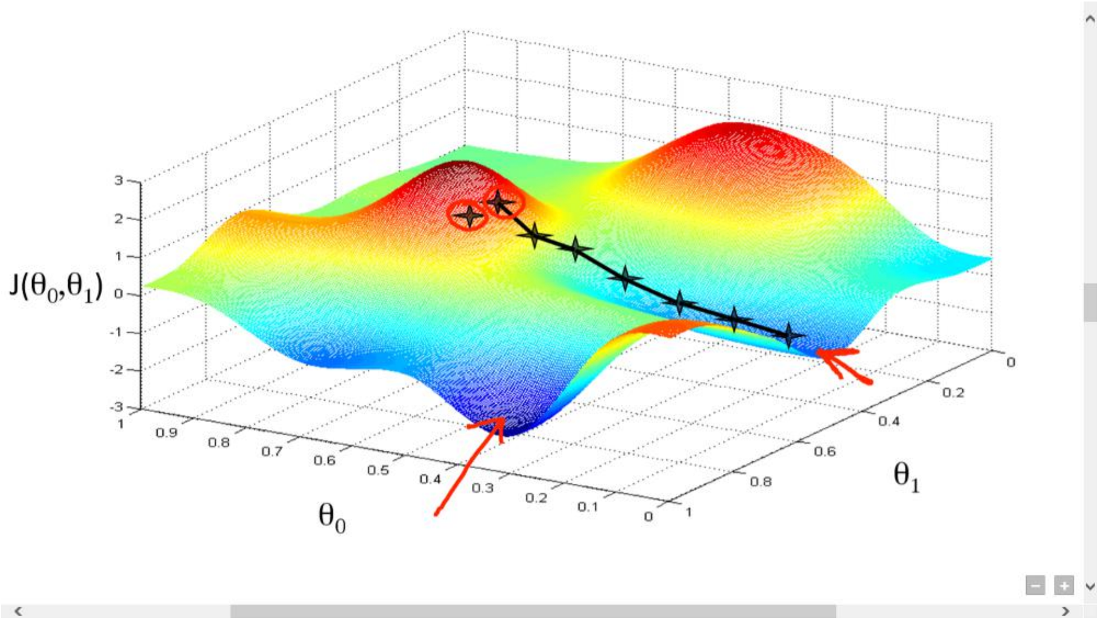
ABSTRACT

This work aims to improve upon the recently proposed and rapidly popularized optimization algorithm *Adam* (Kingma & Ba, 2014). Adam has two main components—a *momentum* component and an *adaptive learning rate* component. However, regular momentum can be shown conceptually and empirically to be inferior to a similar algorithm known as *Nesterov's accelerated gradient* (NAG). We show how to modify Adam's momentum component to take advantage of insights from NAG, and then we present preliminary evidence suggesting that making this substitution improves the speed of convergence and the quality of the learned mod-

Avoiding Overfitting

Update the optimizer's **learning_rate** attribute at the beginning of each epoch:

```
lr_scheduler = keras.callbacks.ReduceLROnPlateau(factor=0.5, patience=5)
history = model.fit(X_train, y_train, [...], callbacks=[lr_scheduler])
```



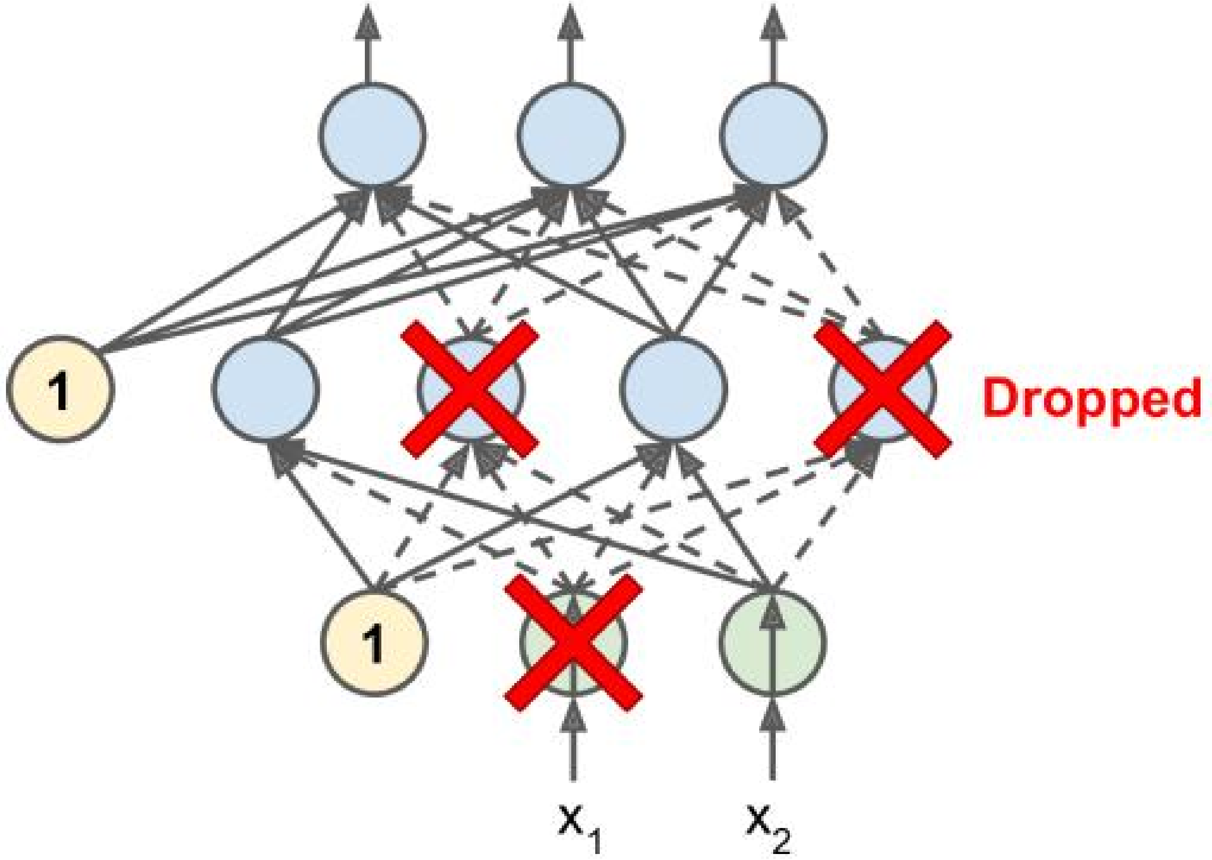
Avoiding Overfitting - ℓ1 and ℓ2 Regularization

```
layer_h_n = keras.layers.Dense(100, activation="elu",kernel_initializer="he_normal",
                                      kernel_regularizer=keras.regularizers.l2(0.01))
layer_output = keras.layers.Dense(10, activation="softmax",kernel_initializer="glorot_uniform")
model.add(layer_h_n)
model.add(layer_output)
```

- The l2() function returns a regularizer that will be called at each step during training to compute the regularization loss. This is then added to the final loss.
- l1 : keras.regularizers.l1(0.01)
- l1 + l2 : keras.regularizers.l1_l2()

Avoiding Overfitting - Dropout

```
model.add(keras.layers.Dropout(0.4))
```



How to increase your small image dataset

```
trainAug = ImageDataGenerator( rotation_range=40, width_shift_range=0.2,
                height_shift_range = 0.2, shear_range=0.2, zoom_range=0.2, horizontal_flip=True,
                fill_mode='nearest')

model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
H = model.fit_generator( trainAug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,validation_data=(testX, testY), validation_steps=len(testX)
// BS, epochs=EPOCHS)
```

Avoiding Overfitting - Using Callbacks

```
checkpoint_cb = keras.callbacks.ModelCheckpoint ("my_keras_model.h5",
                                                  save_best_only=True)
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])
# roll back to best model
model = keras.models.load_model("my_keras_model.h5")
```

- You can combine both callbacks to save checkpoints of your model (in case your computer crashes) and interrupt training early when there is no more progress (to avoid wasting time and resources):

```
early_stopping_cb = keras.callbacks.EarlyStopping(patience=10,  restore_best_weights=True)
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```

How to Save and Load Your Model

Keras use the **HDF5 format** to save both the **model's architecture (including every layer's hyperparameters)** and the values of all **the model parameters** for every layer **(e.g., connection weights and biases)**. It also saves **the optimizer** (including its hyperparameters and any state it may have).

```
model.save("my_keras_model.h5")
```

Loading the model:

```
model = keras.models.load_model("my_keras_model.h5")
```

Thank you for your attention

Code link Brain Tumor Detection CNN

kaggle
https://www.kaggle.com/hichemfelouat/brain-tumor-detection-cnn-01

Github
https://github.com/hichemfelouat/my-codes-of-machine-learning/blob/master/Brain%20Tumor%20Detection%20CNN.py