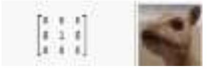
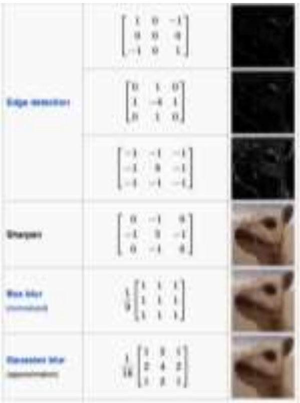
2.
Réseaux à convolution

CNN

Filtres



$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

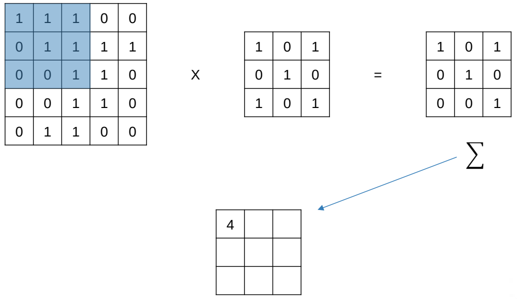


Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Usages :

- Reconnaissance d'image et vidéo,
- Systèmes de recommandation
- Traitement du langage naturel.

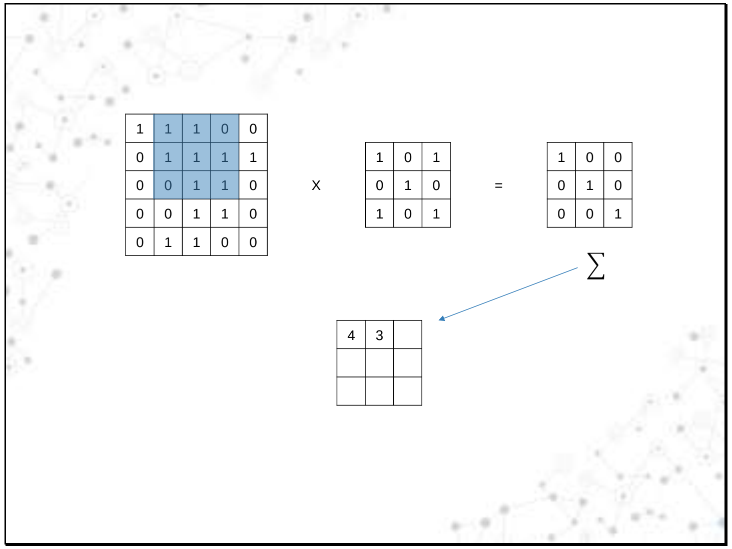
Convolutions : filtres



$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\sum$$

4		



1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

x

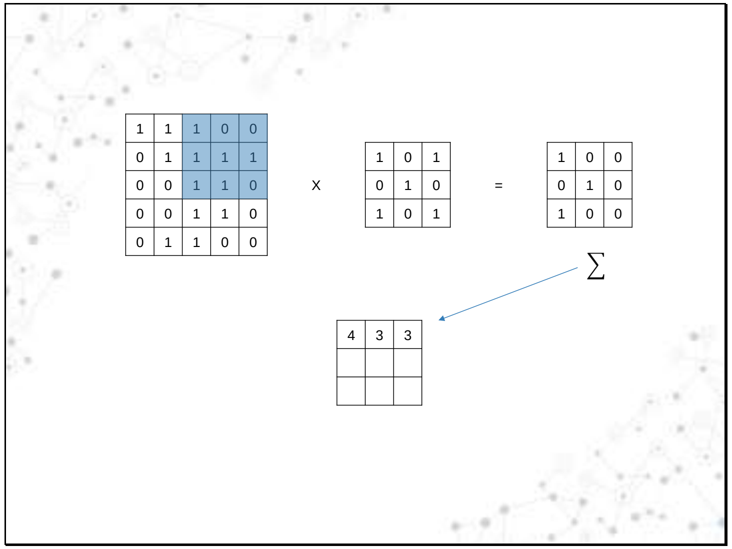
1	0	1
0	1	0
1	0	1

=

1	0	0
0	1	0
0	0	1

$\sum$

4	3	



1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

x

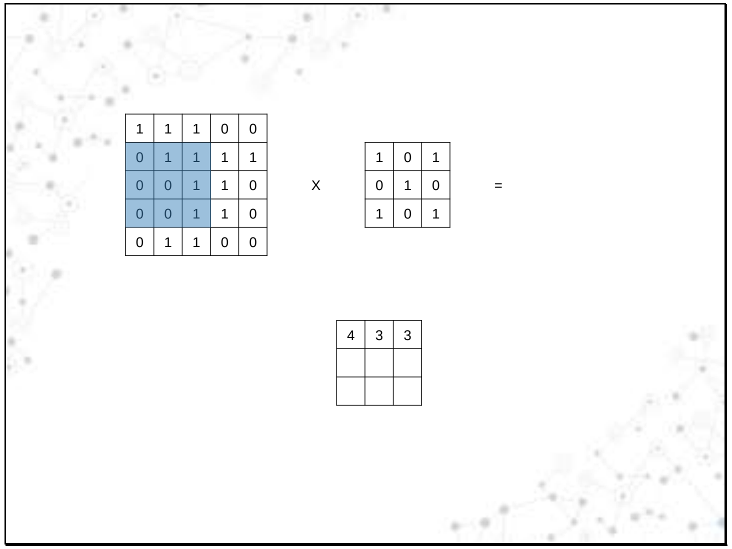
1	0	1
0	1	0
1	0	1

=

1	0	0
0	1	0
1	0	0

$\sum$

4	3	3



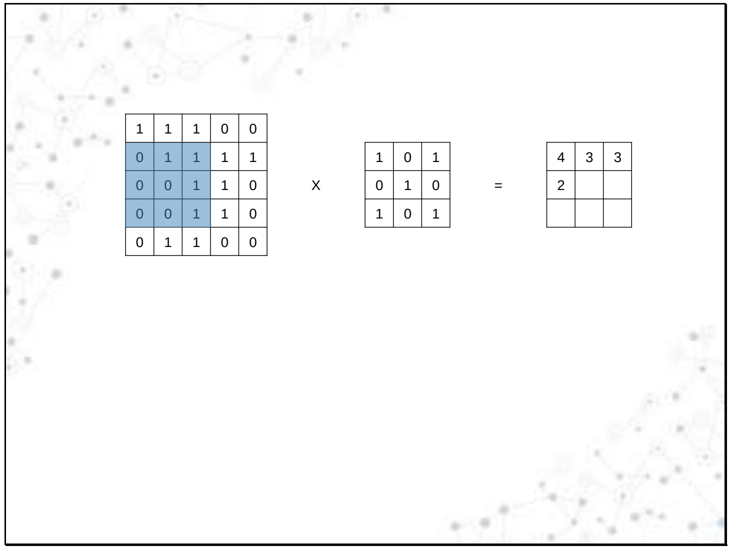
1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

x

1	0	1
0	1	0
1	0	1

=

4	3	3



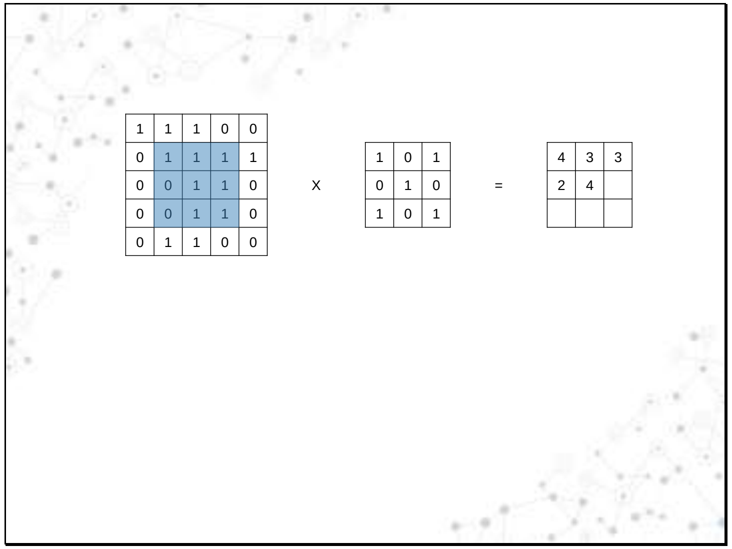
1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

x

1	0	1
0	1	0
1	0	1

=

4	3	3
2		



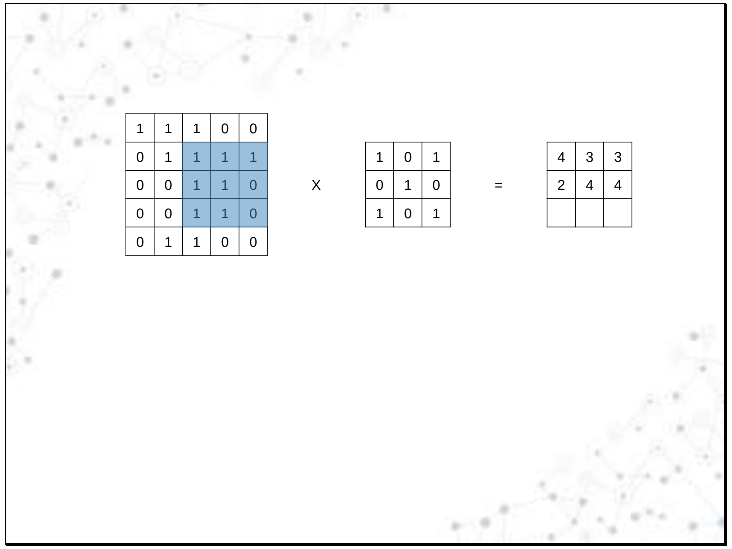
1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

X

1	0	1
0	1	0
1	0	1

=

4	3	3
2	4	



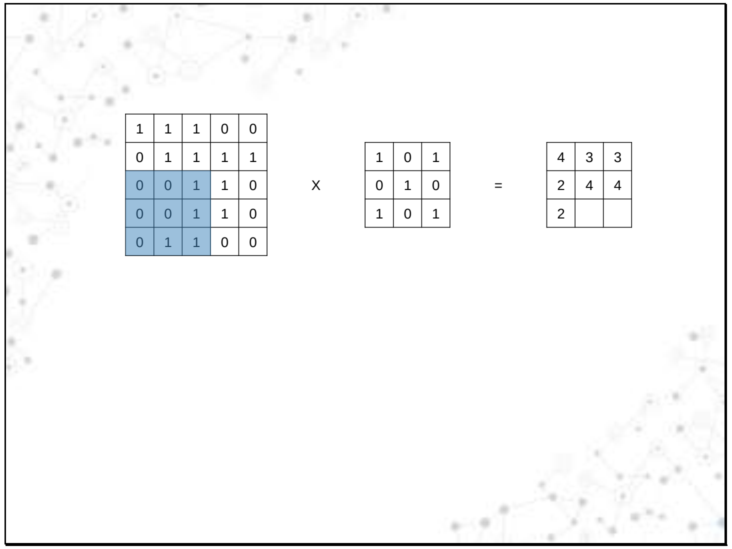
1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

X

1	0	1
0	1	0
1	0	1

=

4	3	3
2	4	4



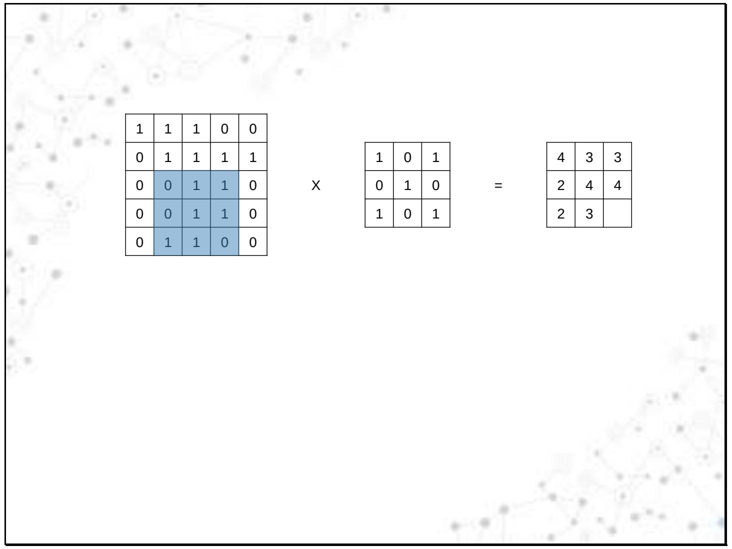
1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

X

1	0	1
0	1	0
1	0	1

=

4	3	3
2	4	4
2		



1	1	1	0	0
0	1	1	1	1
0	0	1	1	0
0	0	1	1	0
0	1	1	0	0

X

1	0	1
0	1	0
1	0	1

=

4	3	3
2	4	4
2	3	

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \text{ X } \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 4 & 3 & 3 \\ 2 & 4 & 4 \\ 2 & 3 & 3 \end{pmatrix}$$

Réduction de dimension : pooling

$$\begin{pmatrix} 4 & 3 & 3 \\ 2 & 4 & 4 \\ 2 & 3 & 3 \end{pmatrix}$$

Max pool →

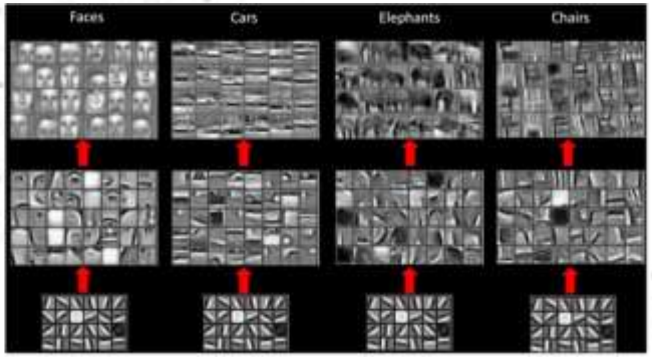
$$\begin{pmatrix} 4 & 4 \\ 4 & 4 \end{pmatrix}$$

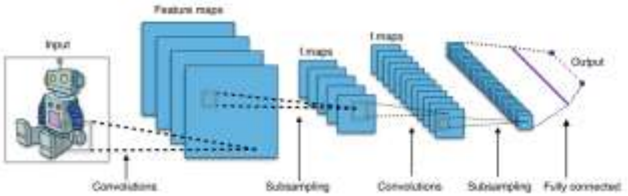
avg pool →

$$\begin{pmatrix} 3.25 & 3.5 \\ 2.75 & 3.5 \end{pmatrix}$$

Norm L2 pool →

$$\begin{pmatrix} 6.7 & 7.1 \\ 5.75 & 7.1 \end{pmatrix}$$





Par Aphex34 — Travail personnel, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45679374

Implémentation

CIFAR 10 :

60000 images

32 x 32 x RGB

10 classes : voitures, avions , oiseaux, chats, ….

Disponibles sur Kaggle

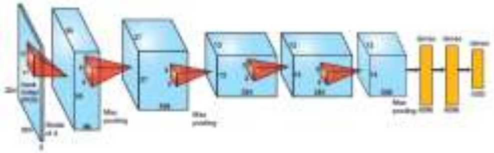
Prétraitement

```
# Random seed
seed = 7
numpy.random.seed(seed)

# Chargement des données en train/test
(X_train, y_train), (X_test, y_test) = cifar10.load_data()


# Normalisation de l'input
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0

#Traitement de l'output
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

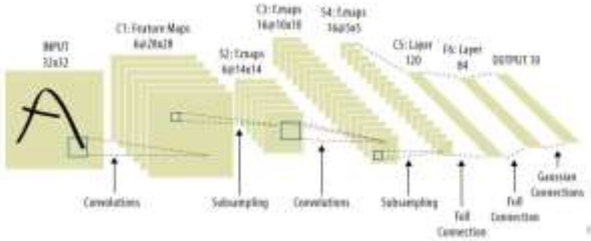
```
# Creation du modele
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 32, 32), padding='same', activation='relu'))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# Compile model
epochs = 10
lrate = 0.001
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
dropout_3 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
max_pooling2d_2              (MaxPooling2 (None, 32, 16, 16)   0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               4194816
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
```

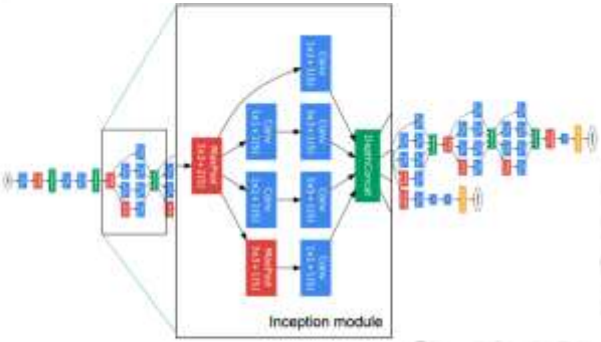
Alexnet - 2012



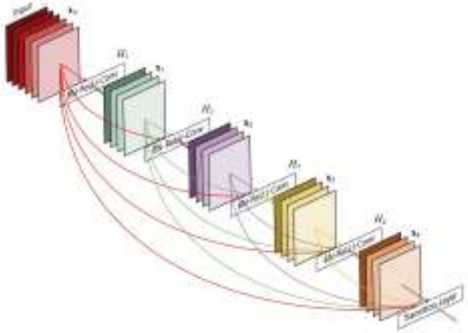
LeNet5 - 1988



GoogleNet/Inception - 2014



DenseNet 2016



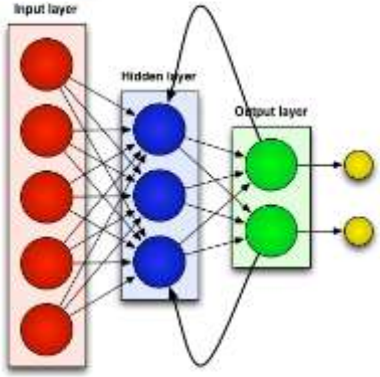
3.

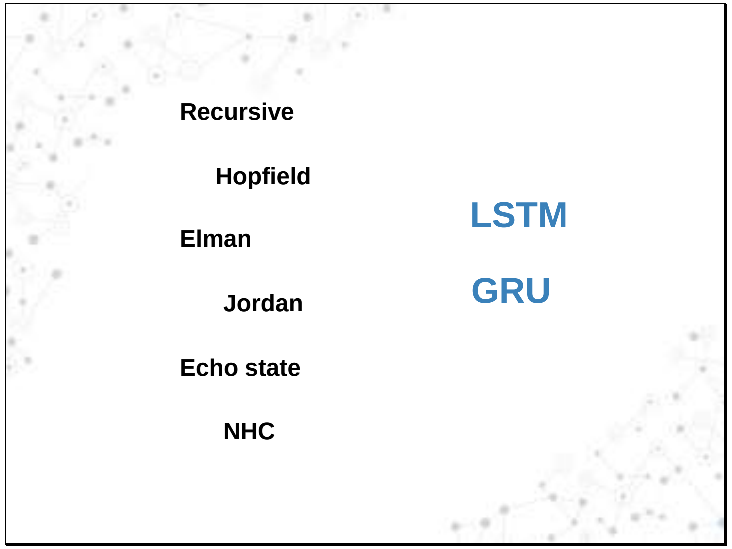
Réseaux récurrents

RNN

Usages :

- Séries temporelles
- Reconnaissance de la parole
- Reconnaissance des caractères manuscrits
- Reconnaissance de formes
- Traduction automatique





Recursive

Hopfield

Elman

Jordan

Echo state

NHC

LSTM

GRU