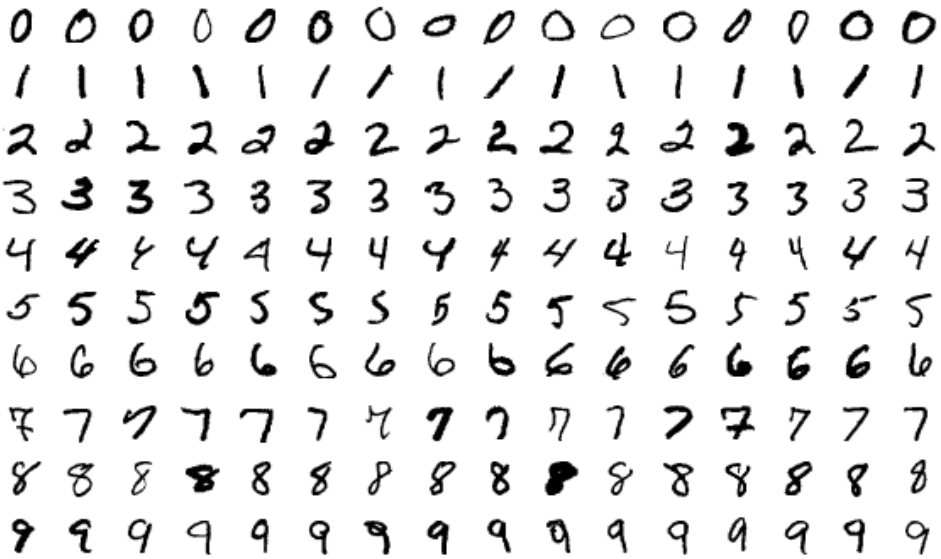
Rapport et présentation

MNIST Dataset est un ensemble de données de chiffres manuscrits et contient un ensemble de formation de 60 000 exemples et un ensemble de test de 10 000 exemples.

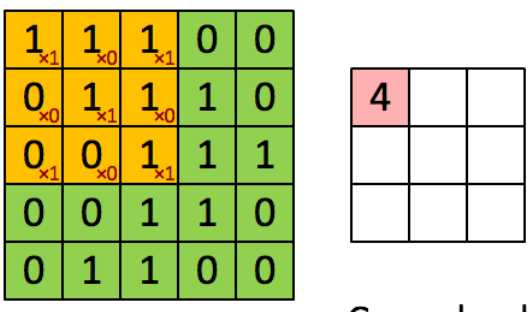
Il s'agit d'un échantillon de l'ensemble de données MNIST.



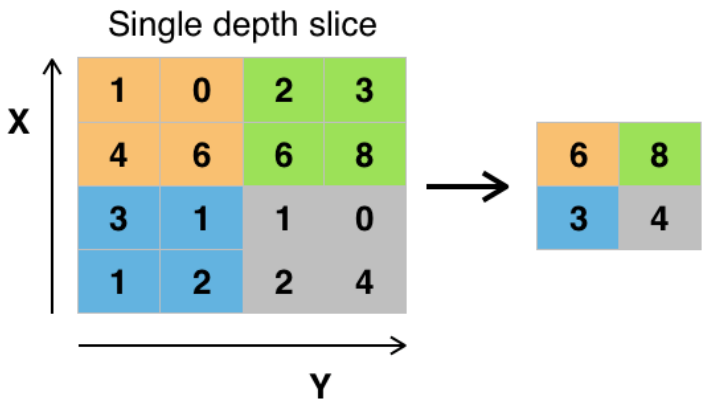
Définition du modèle CNN

Les réseaux de neurones convolutifs (CNN) ou ConvNet sont des architectures de réseaux de neurones populaires couramment utilisées dans les problèmes de vision par ordinateur comme la classification d'images et la détection d'objets. Considérez une image couleur de 1000x1000 pixels ou 3 millions d'entrées, l'utilisation d'un réseau neuronal normal avec 1000 unités cachées dans la première couche générera une matrice de poids de 3 milliards de paramètres! CNN utilise un ensemble d'opérations de convolution et de mise en commun pour faire face à cette complexité.

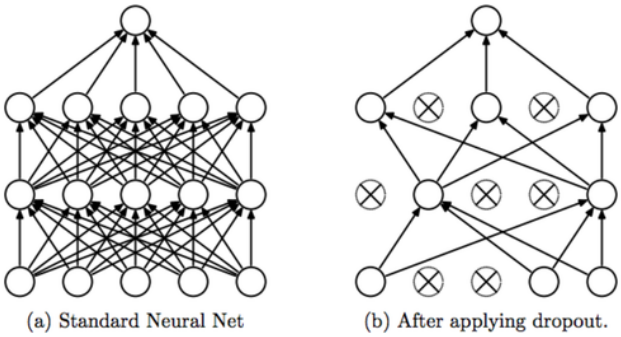
Opération de convolution: L'opération de convolution implique le chevauchement d'un filtre / noyau de taille fixe sur la matrice d'image d'entrée, puis un glissement pixel par pixel pour couvrir la totalité de l'image / matrice. Comme indiqué ci-dessous, la fenêtre du filtre 3x3 (jaune) glisse sur la matrice d'image 5x5, toutes les valeurs dans la matrice jaune sont ajoutées et stockées dans une nouvelle matrice convolutée (rose).



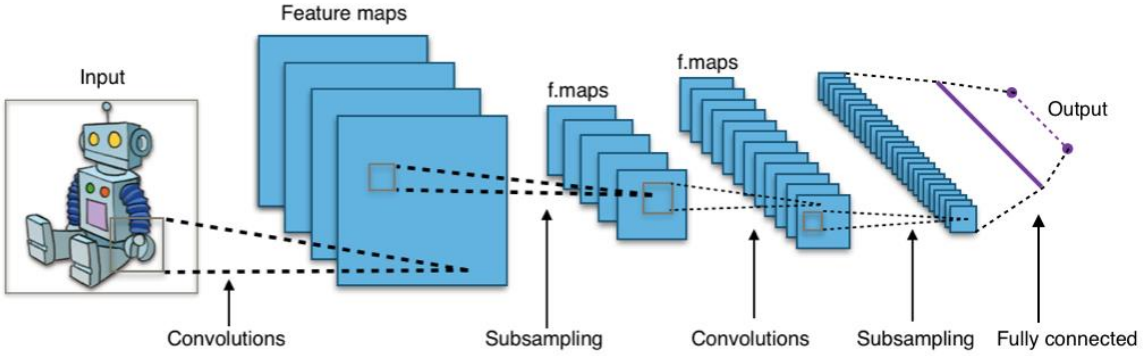
Opération de regroupement: avec les couches Convolution, CNN utilise également des couches de regroupement pour réduire la taille de la représentation, accélérer le calcul et rendre certaines des fonctionnalités détectées un peu plus robustes. Le regroupement est de 2 types: regroupement maximal et regroupement moyen. Nous utiliserons Max Pooling dans notre ConvNet. L'opération de mise en pool maximale trouve simplement le nombre maximum dans la fenêtre de filtre glissante sur la matrice d'image et lui renvoie la nouvelle matrice comme indiqué ci-dessous. Ainsi, les nombres maximums 6,8,3,4 sont sélectionnés dans chaque fenêtre 2x2 à partir d'une matrice d'image 4x4.



Dropout: Dropout est une technique de régularisation utilisée dans les réseaux de neurones pour éviter le sur-ajustement. Pour l'abandon, nous passons par chaque couche de réseau et définissons une certaine probabilité d'éliminer un nœud dans le réseau neuronal. L'élimination de ces unités au hasard entraîne une répartition et une réduction des poids.


(a) Standard Neural Net (b) After applying dropout.

Architecture CNN typique: La combinaison de toutes ces couches et de toutes les couches entièrement connectées donne lieu à diverses architectures ConvNet utilisées aujourd'hui pour diverses tâches de vision par ordinateur. Voici un exemple de cette architecture:



Architecture du modèle

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 28, 28, 32)	544
conv2d_10 (Conv2D)	(None, 28, 28, 32)	16416
max_pooling2d_5 (MaxPooling2	(None, 14, 14, 32)	0
dropout_5 (Dropout)	(None, 14, 14, 32)	0
conv2d_11 (Conv2D)	(None, 14, 14, 32)	9248
conv2d_12 (Conv2D)	(None, 14, 14, 32)	9248
max_pooling2d_6 (MaxPooling2	(None, 7, 7, 32)	0
flatten_3 (Flatten)	(None, 1568)	0
dense_5 (Dense)	(None, 256)	401664
dropout_6 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 10)	2570

Total params: 439,690
Trainable params: 439,690
Non-trainable params: 0

Max_pooling2d_6	INPUT	(None,14 ,14,32)
	OUTPUT	(None,7 ,7 ,32)
Flatten_3	INPUT	(None,7 ,7 ,32)
	OUTPUT	(None,1568)
Dense_5	INPUT	(None,1568)
	OUTPUT	(None,256)
Dropout_6	INPUT	(None,256)
	OUTPUT	(None,256)
Dense_6	INPUT	(None,256)
	OUTPUT	(None,10)

Entrainement :

J'utilise la carte graphique NVIDIA GPU GeForce GTX 750 Ti compatible CUDA pour m'entraîner au-dessus du modèle avec 15 époques, ce qui prend environ 20 minutes pour s'entraîner.

In [482]:
```
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
# With data augmentation to prevent overfitting (accuracy 0.99286)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=False,
        vertical_flip=False)
```

In [483]:
```
datagen.fit(X_train)
```

In [484]:
```
history=model.fit_generator(datagen.flow(X_train, Y_train, batch_size=86), epochs=100, validation_data=(X_
valid,Y_valid),verbose = 2, steps_per_epoch=X_train.shape[0] // 86,callbacks=[learning_rate_reduction])
```

```
Epoch 1/100
 - 36s - loss: 1.7688 - acc: 0.3815 - val_loss: 0.6188 - val_acc: 0.8657
Epoch 2/100
 - 33s - loss: 0.9391 - acc: 0.6941 - val_loss: 0.2616 - val_acc: 0.9347
Epoch 3/100
 - 33s - loss: 0.6539 - acc: 0.7936 - val_loss: 0.1714 - val_acc: 0.9530
Epoch 4/100
 - 33s - loss: 0.5323 - acc: 0.8339 - val loss: 0.1398 - val acc: 0.9588
```