PROJET DE RECHERCHE OPÉRATIONNELLE: TENSORFLOW APPLICATION

classification de l'image

Bouslama.S\Mhiri.K

10/12/2017

1 Introduction :

Depuis une dizaine d'années, les librairies et boîtes à outils informatiques destinées à l'intelligence artificielle se multiplient. Nous pouvons notamment citer Scikit-learn, sortie en 2010, qui est l'une des plus célèbres. Depuis, les acteurs majeurs de l'industrie ont décidé de créer et publier leurs propres librairies. TensorFlow est celle créée et rendue publique par Google en 2015. Utilisée entre autres par Ebay, Twitter, Airbus, AirBnb et Intel, cette librairie est performante et son avenir s'annonce prometteur. S'appuyer dessus semble donc être un choix tout à fait cohérent bien qu'il existe des alternatives sérieuses.

TensorFlow est un framework d'apprentissage en profondeur open source créé par Google qui donne aux développeurs un contrôle granulaire sur chaque neurone (connu sous le nom de «nœud» dans TensorFlow) afin que vous puissiez ajuster les poids et obtenir des performances optimales. TensorFlow a de nombreuses bibliothèques intégrées (dont peu seront utilisées pour la classification des images) et possède une communauté incroyable, vous pourrez donc trouver des implémentations open source pour pratiquement n'importe quel sujet d'apprentissage en profondeur.

Dans la suite, le type traité par TENSORFLOW est le CNN(Convolutional Neural Network)et plus précisément, l'exemple de la classification des images et leur recognition.

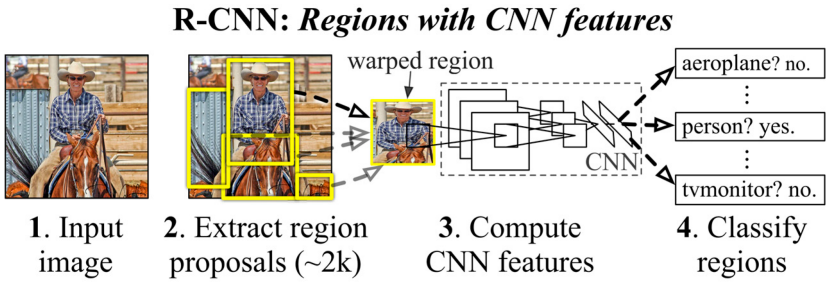
2 Principe de base du réseau de neurone convolutif :

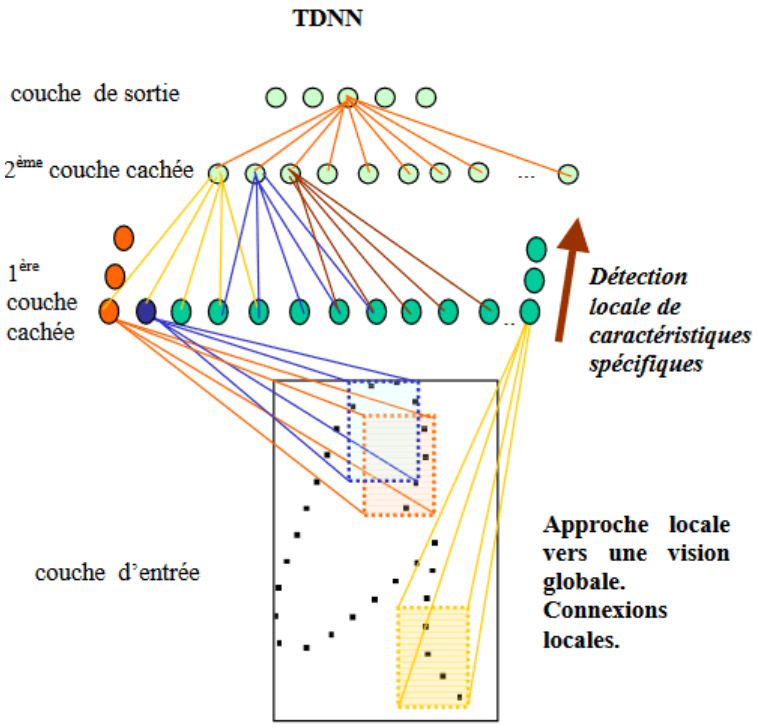
Les réseaux convolutifs sont une forme particulière de réseau neuronal multicouches dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères.

Leur conception suit la découverte de mécanismes visuels dans les organismes vivants. Ces réseaux de neurones artificiels (aussi baptisés réseau de neurones à convolution, ou CNN) sont capables de catégoriser les informations des plus simples aux plus complexes. Ils consistent en un empilage multicouche de neurones, des fonctions mathématiques à plusieurs paramètres ajustables, qui pré-traitent de petites quantités d'informations. Les réseaux convolutifs sont caractérisé par leurs premières couches convolutionelles (généralement une à trois). Une couche convolutive, est basée comme son nom l'indique sur le principe mathématique de convolution, et cherche à repérer la présence d'un motif (dans un signal ou dans une image par exemple).

Pour une image, la première couche convolutionelle peut détecter les contours des objets (par exemple un cercle), la seconde couche convolutionelle peut combiner les contours en objets (par exemple une roue), et les couches suivantes (non nécessairement convolutionelles) peuvent utiliser ces informations pour distinguer une voiture d'une moto. Une phase d'apprentissage sur des objets connus

permet de trouver les meilleurs paramètres en montrant par exemple à la machine des milliers d'images d'un chien, d'une voiture ou d'un sport... L'un des enjeux est de trouver des méthodes pour ajuster ces paramètres le plus rapidement et le plus efficacement possible. Les réseaux neuronaux convolutifs ont de nombreuses applications dans la reconnaissance d'images, de vidéos ou le traitement du langage naturel.Parmi eux on cite la classification des images.





3 la classification et la recognition de l'image :

3.1 definition :

Comme son nom l'indique,la classification s'agit d'une opération visant à classer les images en fonction des sujets, décors, actions ou toutes autres propriétés visibles sur les images. Cette opération correspond à un besoin réel de l'industrie. Nous pouvons citer notamment la modération automatisée d'images de profil, le contrôle parental ou encore la détection d'événements anormaux de toutes sortes.

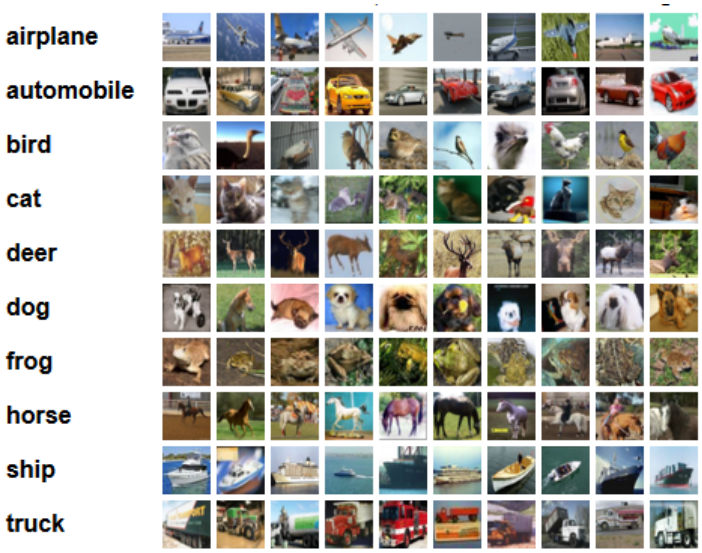
3.2 comment cela fonctionne :

Grâce à tensorflow, un ordinateur est désormais capable de detecter l'aspect d'une image et dire avec un pourcentage d'erreur minimal ce que représente cette derniere en contenu. Pour ce faire, nous devons d'abord apprendre à l'ordinateur à quoi ressemble un chat, un chien, un oiseau, etc. avant qu'il ne puisse reconnaître un nouvel objet. Plus l'ordinateur voit de chats, mieux il reconnaît les chats. Ceci est connu comme l'apprentissage supervisé. Nous pouvons effectuer cette tâche en étiquetant les images, l'ordinateur commencera à reconnaître les motifs présents dans les images de chat qui sont absents des autres et commencera à construire sa propre cognition

3.3 introduction de la dataset

L'ensemble de données CIFAR-10 se compose de 60000 images couleur 32x32 dans 10 classes, avec 6000 images par classe. Il y a 50000 images d'entraînement et 10000 images d'essai.

L'ensemble de données est divisé en cinq lots de formation et un lot de test, chacun avec 10000 images. Le lot de test contient exactement 1000 images sélectionnées au hasard dans chaque classe. Les lots d'entraînement contiennent les images restantes dans un ordre aléatoire, mais certains lots d'entraînement peuvent contenir plus d'images d'une classe que d'une autre. Entre eux, les lots d'entraînement contiennent exactement 5000 images de chaque classe.
Voici les classes dans l'ensemble de données, ainsi que 10 images aléatoires de chacun :



3.4 codage et démarche :

1-Obtenir les données :

In [1]:

```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
from urllib.request import urlretrieve
from os.path import isfile, isdir
from tqdm import tqdm
import problem_unittests as tests
import tarfile

cifar10_dataset_folder_path = 'cifar-10-batches-py'

class DLProgress(tqdm):
    last_block = 0

    def hook(self, block_num=1, block_size=1, total_size=None):
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num

if not isfile('cifar-10-python.tar.gz'):
    with DLProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10 Dataset') as pbar:
        urlretrieve(
            'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz',
            'cifar-10-python.tar.gz',
            pbar.hook)

if not isdir(cifar10_dataset_folder_path):
    with tarfile.open('cifar-10-python.tar.gz') as tar:
        tar.extractall()
        tar.close()
```

```
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num

if not isfile('cifar-10-python.tar.gz'):
    with DLProgress(unit='B', unit_scale=True, miniters=1, desc='CIFAR-10 Dataset') as pbar:
        urlretrieve(
            'https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz',
            'cifar-10-python.tar.gz',
            pbar.hook)

if not isdir(cifar10_dataset_folder_path):
    with tarfile.open('cifar-10-python.tar.gz') as tar:
        tar.extractall()
        tar.close()


tests.test_folder_path(cifar10_dataset_folder_path)
```

```
All files found!
```

2-Explorer les données :

In [2]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import helper
import numpy as np

# Explore the dataset
batch_id = 1
sample_id = 5
helper.display_stats(cifar10_dataset_folder_path, batch_id, sample_id)
```

3-Implémenter les fonctions de prépocès :

-Normaliser

In [3]:
```
def normalize(x):
    """
    Normalize a list of sample image data in the range of 0 to 1
    : x: List of image data.  The image shape is (32, 32, 3)
    : return: Numpy array of normalize data
    """
    # TODO: Implement Function
    return x / 255


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_normalize(normalize)
```

Tests Passed

In [8]:
```
def conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides):
    """
    Apply convolution then max pooling to x_tensor
    :param x_tensor: TensorFlow Tensor
    :param conv_num_outputs: Number of outputs for the convolutional layer
    :param conv_ksize: kernal size 2-D Tuple for the convolutional layer
    :param conv_strides: Stride 2-D Tuple for convolution
    :param pool_ksize: kernal size 2-D Tuple for pool
    :param pool_strides: Stride 2-D Tuple for pool
    : return: A tensor that represents convolution and max pooling of x_tensor
    """
    # TODO: Implement Function
    # Weight and bias
    channels = x_tensor.get_shape().as_list()[3]

    weights = tf.Variable(tf.random_normal([conv_ksize[0], conv_ksize[1], channels, conv_num_outputs]))
    bias = tf.Variable(tf.zeros(conv_num_outputs))

    # Apply convolution
    conv_layer = tf.nn.conv2d(x_tensor, weights, strides=[1, conv_strides[0], conv_strides[1], 1], padding="SAME")

    # Add bias
    conv_layer = tf.nn.bias_add(conv_layer, bias)

    # Apply non-linear activation function
    conv_layer = tf.nn.relu(conv_layer)
```

```
    conv_layer = tf.nn.bias_add(conv_layer, bias)

    # Apply non-linear activation function
    conv_layer = tf.nn.relu(conv_layer)

    # Apply max-pooling
    conv_layer = tf.nn.max_pool(conv_layer, \
                                ksize=[1, pool_ksize[0], pool_ksize[1], 1], \
                                strides=[1, pool_strides[0], pool_strides[1], 1], \
                                padding='SAME')

    return conv_layer


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_con_pool(conv2d_maxpool)
```

Tests Passed

-Applatir la couche

In [9]:
```
def flatten(x_tensor):
    """
    Flatten x_tensor to (Batch Size, Flattened Image Size)
    : x_tensor: A tensor of size (Batch Size, ...), where ... are the image dimensions.
    : return: A tensor of size (Batch Size, Flattened Image Size).
    """
    # TODO: Implement Function
    dim_list = x_tensor.get_shape().as_list()
    batch_size = dim_list[0]
    height = dim_list[1]
    width = dim_list[2]
    depth = dim_list[3]

    flattened = int(height * width * depth)

    return tf.reshape(x_tensor, [-1, flattened])

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_flatten(flatten)
```

Tests Passed

In [7]:
```
import tensorflow as tf

def neural_net_image_input(image_shape):
    """
    Return a Tensor for a bach of image input
    : image_shape: Shape of the images
    : return: Tensor for image input.
    """
    # TODO: Implement Function

    return tf.placeholder(tf.float32, shape=[None, image_shape[0], image_shape[1], image_shape[2]], name="x")


def neural_net_label_input(n_classes):
    """
    Return a Tensor for a batch of label input
    : n_classes: Number of classes
    : return: Tensor for label input.
    """
    # TODO: Implement Function

    return tf.placeholder(tf.float32, shape=[None, n_classes], name="y")


def neural_net_keep_prob_input():
```

```
def neural_net_keep_prob_input():
    """
    Return a Tensor for keep probability
    : return: Tensor for keep probability.
    """
    # TODO: Implement Function

    return tf.placeholder(tf.float32, name="keep_prob")


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tf.reset_default_graph()
tests.test_nn_image_inputs(neural_net_image_input)
tests.test_nn_label_inputs(neural_net_label_input)
tests.test_nn_keep_prob_inputs(neural_net_keep_prob_input)
```

```
Image Input Tests Passed.
Label Input Tests Passed.
Keep Prob Tests Passed.
```

4-Convolution et couche de regroupement maximum

In [8]:
```
def conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides):
    """
    Apply convolution then max pooling to x_tensor
    :param x_tensor: TensorFlow Tensor
    :param conv_num_outputs: Number of outputs for the convolutional layer
    :param conv_ksize: kernal size 2-D Tuple for the convolutional layer
    :param conv_strides: Stride 2-D Tuple for convolution
    :param pool_ksize: kernal size 2-D Tuple for pool
    :param pool_strides: Stride 2-D Tuple for pool
    : return: A tensor that represents convolution and max pooling of x_tensor
    """
    # TODO: Implement Function
    # Weight and bias
    channels = x_tensor.get_shape().as_list()[3]

    weights = tf.Variable(tf.random_normal([conv_ksize[0], conv_ksize[1], channels, conv_num_outputs]))
    bias = tf.Variable(tf.zeros(conv_num_outputs))

    # Apply convolution
    conv_layer = tf.nn.conv2d(x_tensor, weights, strides=[1, conv_strides[0], conv_strides[1], 1], padding="SAME")

    # Add bias
    conv_layer = tf.nn.bias_add(conv_layer, bias)

    # Apply non-linear activation function
    conv_layer = tf.nn.relu(conv_layer)
    conv_layer = tf.nn.bias_add(conv_layer, bias)

    # Apply non-linear activation function
    conv_layer = tf.nn.relu(conv_layer)

    # Apply max-pooling
    conv_layer = tf.nn.max_pool(conv_layer, \
                                ksize=[1, pool_ksize[0], pool_ksize[1], 1], \
                                strides=[1, pool_strides[0], pool_strides[1], 1], \
                                padding='SAME')

    return conv_layer


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_con_pool(conv2d_maxpool)
```

```
Tests Passed
```

-Applatir la couche

In [9]:
```
def flatten(x_tensor):
    """
    Flatten x_tensor to (Batch Size, Flattened Image Size)
    : x_tensor: A tensor of size (Batch Size, ...), where ... are the image dimensions.
    : return: A tensor of size (Batch Size, Flattened Image Size).
    """
    # TODO: Implement Function
    dim_list = x_tensor.get_shape().as_list()
    batch_size = dim_list[0]
    height = dim_list[1]
    width = dim_list[2]
    depth = dim_list[3]

    flattened = int(height * width * depth)

    return tf.reshape(x_tensor, [-1, flattened])

"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_flatten(flatten)
```

```
Tests Passed
```

-Couche entièrement connectée

Fully-Connected Layer

Implement the `fully_conn` function to apply a fully connected layer to `x_tensor` with the shape (*Batch Size*, *num_outputs*). Shortcut option: you can use classes from the TensorFlow Layers or TensorFlow Layers (contrib) packages for this layer. For more of a challenge, only use other TensorFlow packages.

In [10]:
```
def fully_conn(x_tensor, num_outputs):
    """
    Apply a fully connected layer to x_tensor using weight and bias
    : x_tensor: A 2-D tensor where the first dimension is batch size.
    : num_outputs: The number of output that the new tensor should be.
    : return: A 2-D tensor where the second dimension is num_outputs.
    """
    # TODO: Implement Function
    num_in = x_tensor.get_shape().as_list()[1]
    weights = tf.Variable(tf.random_normal([num_in, num_outputs]))
    bias = tf.Variable(tf.zeros(num_outputs))

    fully_conn = tf.add(tf.matmul(x_tensor, weights), bias)

    fully_conn = tf.nn.relu(fully_conn)

    return fully_conn


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_fully_conn(fully_conn)
```

```
Tests Passed
```

-Couche de sortie

Output Layer

Implement the `output` function to apply a fully connected layer to `x_tensor` with the shape (*Batch Size*, *num_outputs*). Shortcut option: you can use classes from the TensorFlow Layers or TensorFlow Layers (contrib) packages for this layer. For more of a challenge, only use other TensorFlow packages.

Note: Activation, softmax, or cross entropy should **not** be applied to this.

In [11]:
```
def output(x_tensor, num_outputs):
    """
    Apply a output layer to x_tensor using weight and bias
    : x_tensor: A 2-D tensor where the first dimension is batch size.
    : num_outputs: The number of output that the new tensor should be.
    : return: A 2-D tensor where the second dimension is num_outputs.
    """
    # TODO: Implement Function
    num_in = x_tensor.get_shape().as_list()[1]
    weights = tf.Variable(tf.random_normal([num_in, num_outputs]))
    bias = tf.Variable(tf.zeros(num_outputs))

    fully_conn = tf.add(tf.matmul(x_tensor, weights), bias)

    return fully_conn


"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
tests.test_output(output)
```

```
Tests Passed
```

5-Créer un modèle convolutif :

In [12]:
```
def conv_net(x, keep_prob):
    """
    Create a convolutional neural network model
    : x: Placeholder tensor that holds image data.
    : keep_prob: Placeholder tensor that hold dropout keep probability.
    : return: Tensor that represents logits
    """
    # TODO: Apply 1, 2, or 3 Convolution and Max Pool layers
    #    Play around with different number of outputs, kernel size and stride
    # Function Definition from Above:
    #    conv2d_maxpool(x_tensor, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides)

    # Variable initialization
    conv_ksize = (3, 3)
    conv_strides = (2, 2)
    pool_ksize = (2, 2)
    pool_strides = (1, 1)
    conv_num_outputs = 48

    # CNN Layer 1 -- 32x32x3 to 16x16x48
    conv_layer = conv2d_maxpool(x, conv_num_outputs, conv_ksize, conv_strides, pool_ksize, pool_strides)

    # CNN Layer 2 -- 16x16x48 to 8x8x192
    conv_layer = conv2d_maxpool(conv_layer, 192, (2, 2), conv_strides, pool_ksize, (1, 1))

    # CNN Layer 3 -- 8x8x192 to 4x4x384
    conv_layer = conv2d_maxpool(conv_layer, 384, (2, 2), conv_strides, (1, 1), pool_strides)

    # CNN Layer 4 - 4x4x384 to 2x2x512
    conv_layer = conv2d_maxpool(conv_layer, 512, (2, 2), (2, 2), (1, 1), (1, 1))

    # TODO: Apply a Flatten Layer
    # Function Definition from Above:
    #   flatten(x_tensor)

    # 2x2x512 to 2048
    conv_layer = flatten(conv_layer)

    # TODO: Apply 1, 2, or 3 Fully Connected Layers
    #    Play around with different number of outputs
    # Function Definition from Above:
    #   fully_conn(x_tensor, num_outputs)

    # Dropout -- 3072 to (keep_prob * 3072)
    fully_conn_layer = tf.nn.dropout(conv_layer, keep_prob)

    # Fully connected Layer 1 -- (keep_prob * 3072) to 512
    fully_conn_layer = fully_conn(conv_layer, 512)

    # Fully connected Layer 2 -- 512 to 128
    fully_conn_layer = fully_conn(fully_conn_layer, 128)

    # TODO: Apply an Output Layer
    #    Set this to the number of classes
    # Function Definition from Above:
    #   output(x_tensor, num_outputs)

    # Output Layer -- 128 to 10
    out = output(fully_conn_layer, 10)

    # TODO: return output
    return out
```

```
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""

##############################
## Build the Neural Network ##
##############################

# Remove previous weights, bias, inputs, etc..
tf.reset_default_graph()

# Inputs
x = neural_net_image_input((32, 32, 3))
y = neural_net_label_input(10)
keep_prob = neural_net_keep_prob_input()

# Model
logits = conv_net(x, keep_prob)

# Name logits Tensor, so that is can be loaded from disk after training
logits = tf.identity(logits, name='logits')

# Loss and Optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y))
optimizer = tf.train.AdamOptimizer().minimize(cost)

# Accuracy
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy')

tests.test_conv_net(conv_net)
```

6-Former le réseau neurone :

```
In [13]: def train_neural_network(session, optimizer, keep_probability, feature_batch, label_batch):
             """
             Optimize the session on a batch of images and labels
             : session: Current TensorFlow session
             : optimizer: TensorFlow optimizer function
             : keep_probability: keep probability
             : feature_batch: Batch of Numpy image data
             : label_batch: Batch of Numpy label data
             """
             # TODO: Implement Function
             session.run(optimizer, feed_dict={
                 x: feature_batch,
                 y: label_batch,
                 keep_prob: keep_probability
             })

             pass


         """
         DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
         """
         tests.test_train_nn(train_neural_network)
```

```
Tests Passed
```

7-Afficher la statistique :

```
In [14]: def print_stats(session, feature_batch, label_batch, cost, accuracy):
             """
             Print information about loss and validation accuracy
             : session: Current TensorFlow session
             : feature_batch: Batch of Numpy image data
             : label_batch: Batch of Numpy label data
             : cost: TensorFlow cost function
             : accuracy: TensorFlow accuracy function
             """
             # TODO: Implement Function
             loss = session.run(cost, feed_dict={
                 x: feature_batch,
                 y: label_batch,
                 keep_prob: 1.})
             valid_acc = session.run(accuracy, feed_dict={
                 x: valid_features,
                 y: valid_labels,
                 keep_prob: 1.})

             print('Loss: {:>10.4f} | Validation Accuracy: {:.4f}'.format(
                 loss,
                 valid_acc))

             pass
```

8-Former sur un seul lot CIFAR-10 :

In [ ]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
print('Checking the Training on a Single Batch...')
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(epochs):
        batch_i = 1
        for batch_features, batch_labels in helper.load_preprocess_training_batch(batch_i, batch_size):
            train_neural_network(sess, optimizer, keep_probability, batch_features, batch_labels)
        print('Epoch {:>2}, CIFAR-10 Batch {}:  '.format(epoch + 1, batch_i), end='')
        print_stats(sess, batch_features, batch_labels, cost, accuracy)
```

-Former comletement le modele

Fully Train the Model

Now that you got a good accuracy with a single CIFAR-10 batch, try it with all five batches.

In [ ]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
save_model_path = './image_classification'

print('Training...')
with tf.Session() as sess:
    # Initializing the variables
    sess.run(tf.global_variables_initializer())

    # Training cycle
    for epoch in range(epochs):
        # Loop over all batches
        n_batches = 5
        for batch_i in range(1, n_batches + 1):
            for batch_features, batch_labels in helper.load_preprocess_training_batch(batch_i, batch_size):
                train_neural_network(sess, optimizer, keep_probability, batch_features, batch_labels)
            print('Epoch {:>2}, CIFAR-10 Batch {}:  '.format(epoch + 1, batch_i), end='')
            print_stats(sess, batch_features, batch_labels, cost, accuracy)

    # Save Model
    saver = tf.train.Saver()
    save_path = saver.save(sess, save_model_path)
```

9-Tester le modele :

Test Model ¶

Test your model against the test dataset. This will be your final accuracy. You should have an accuracy greater than 50%. If you don't, keep tweaking the model architecture and parameters.

In [ ]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL
"""
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import tensorflow as tf
import pickle
import helper
import random

# Set batch size if not already set
try:
    if batch_size:
        pass
except NameError:
    batch_size = 64

save_model_path = './image_classification'
n_samples = 4
top_n_predictions = 3
```

```
def test_model():
    """
    Test the saved model against the test dataset
    """

    test_features, test_labels = pickle.load(open('preprocess_training.p', mode='rb'))
    loaded_graph = tf.Graph()

    with tf.Session(graph=loaded_graph) as sess:
        # Load model
        loader = tf.train.import_meta_graph(save_model_path + '.meta')
        loader.restore(sess, save_model_path)

        # Get Tensors from loaded model
        loaded_x = loaded_graph.get_tensor_by_name('x:0')
        loaded_y = loaded_graph.get_tensor_by_name('y:0')
        loaded_keep_prob = loaded_graph.get_tensor_by_name('keep_prob:0')
        loaded_logits = loaded_graph.get_tensor_by_name('logits:0')
        loaded_acc = loaded_graph.get_tensor_by_name('accuracy:0')

        # Get accuracy in batches for memory limitations
        test_batch_acc_total = 0
        test_batch_count = 0

        for train_feature_batch, train_label_batch in helper.batch_features_labels(test_features, test_labels, batch_size):
            test_batch_acc_total += sess.run(
                loaded_acc,
                feed_dict={loaded_x: train_feature_batch, loaded_y: train_label_batch, loaded_keep_prob: 1.0})
            test_batch_count += 1

        print('Testing Accuracy: {}\n'.format(test_batch_acc_total/test_batch_count))
```

```
        # Print Random Samples
        random_test_features, random_test_labels = tuple(zip(*random.sample(list(zip(test_features, test_labels)), n_samples)))
        random_test_predictions = sess.run(
            tf.nn.top_k(tf.nn.softmax(loaded_logits), top_n_predictions),
            feed_dict={loaded_x: random_test_features, loaded_y: random_test_labels, loaded_keep_prob: 1.0})
        helper.display_image_predictions(random_test_features, random_test_labels, random_test_predictions)


test_model()
```

4 Conclusion :

Nous avons été en mesure de construire un réseau de neurones convolutionnels artificiels qui peut reconnaître des images avec une précision entre 50% et 70% en utilisant TensorFlow. Nous l'avons fait en prétraitant les images pour rendre le modèle plus générique, diviser l'ensemble de données en un certain nombre de lots et finalement construire et former le modèle.