Starting with Neural Networks: Basic Classification

Make your first neural network for classifying images. It's okay if you don't understand all the details; each part will be explained in the future sections. In the set of this exercises we will be using `tf.keras` (a high-level API to build and train models in TensorFlow) and GoogleColab.
In Googlecolab environment, you have two options: adding code or text.
Exe. 1 Open a new notebook in GoogleColab for python3: `https://colab.research.google.com`. Name your file as "firstname_lasttime_week1.ipynb". Run the following code for activating tensorflow version 2.0:

```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
```

Exe. 2 Import tensorflow, keras, numpy and matplot using the following code:

```
from __future__ import absolute_import, division, print_function,
    unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
```

Exe. 3 Check your tensorflow version using:

```
print(tf.__version__)
```

The output should be `2.` or higher.
Exe. 4 Import the cifra10 data set. TheThe CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images: `https://www.tensorflow.org/datasets/catalog/cifar10`

```
data = keras.datasets.cifar10
cifar10_data = data.load_data()
```

Exe. 5 Before using a dataset, the datatype should be checked. Test `type(cifar10_data)` for verifying the variable type. `len(cifar10_data)` is another command for checking the data size.
Exe. 6 Load train and test images and labels with:

```
(train_images, train_labels),(test_images, test_labels) =
    cifar10_data
```

Loading the dataset returns four NumPy arrays:

- The train_images and train_labels arrays are the training set, the data the model uses to learn.
- The model is tested against the test set, the test_images, and test_labels arrays.

Exe 7 The images are 32×32 NumPy arrays, with pixel values ranging from 0 to 255. You can check an example with:

```
print(train_images[0])
print(train_images[0].shape)
```

The labels are an array of integers, ranging from 0 to 9. Check it with your own code. Each image is mapped to a single label. Since the class names are not included with the dataset, store them here to use later when plotting the images: (check the dataset link for more detailed information: `https://www.cs.toronto.edu/%7Ekriz/cifar.html`)

```
class_names = [ 'airplane' , 'automobile' , 'bird' , 'cat' , 'deer'
    , 'dog' , 'frog' , 'horse' , 'ship' , 'truck']
```

Exe 8 (Explore the data) Before training the model, explore the datasets. Number of train and test points, their array size and etc. You can use `len()` and `arrrayName.shape()` for verifying the train and test datasets. [1]

Exe 9 (Preprocess the data): An interesting fact about the image is that you can plot the image. It is possible using the following:

```
index = 8
plt.figure()
plt.imshow(train_images[index])
plt.colorbar()
plt.grid(False)
plt.show()

train_labels[index]
```

Exe 10 To verify that the data is in the correct format and that you're ready to build and train the network, let's display the first 25 images from the training set and display the class name below each image. To do so use the following commands:

```
plt.subplot()
plt.xticks([])
plt.yticks([])
plt.imshow()
plt.xlabel()
```

[1] Check the modules and functions in Python documentation (`https://docs.python.org/fr/3.6/`).

You can find the matplotlib documentation for Python here: `https://matplotlib.org/3.1.1/users/installing.html`.

Exe 11 You can check various images by changing the index value, or by calling `test_images`. You can see that the pixel values fall in the range of 0 to 255. normalise train and test sets using the following code:

```
train_images = train_images / 255.0
```

Building a neural network in general requires configuring the layers of the model, then compiling the model. The basic building block of a neural network is the **layer**. Layers extract representations from the data fed into them. Hopefully, these representations are meaningful for the problem at hand. Most of deep learning consists of chaining together simple layers. Most layers, such as `tf.keras.layers.Dense`, have parameters that are learned during training.

Exe 12 First neural network definition with three layers and two activation functions[2]

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(32, 32, 3)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')])
```

The first layer in this network, `tf.keras.layers.Flatten`, transforms the format of the images from a two-dimensional array (of 32 by 32 pixels) to a one-dimensional array (of $32 \times 32 = 1024$ pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data.

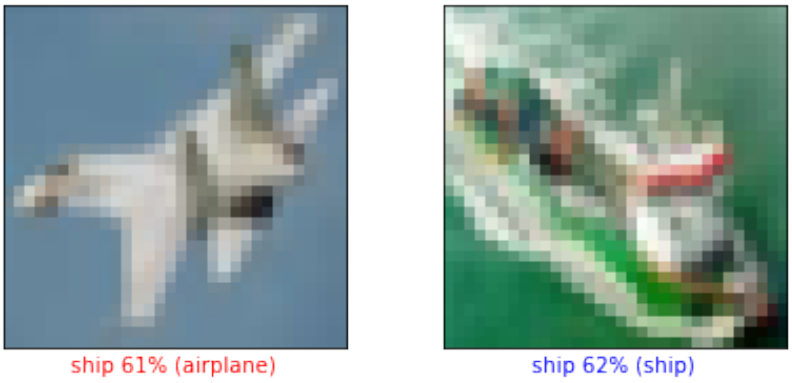
After the pixels are flattened, the network consists of a sequence of two `tf.keras.layers.Dense` layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 128 nodes (or neurons). The second (and last) layer is a 10-node softmax layer that returns an array of 10 probability scores that sum to 1. Each node contains a score that indicates the probability that the current image belongs to one of the 10 classes.

In this exercise, we don't explain the reasons of defining a neural network with this structure. For defining a network compatible with our data, we should define an input layer with the same size as the input data (images size) and an output corresponding the out put data (image labels).

Exe 13 Before the model is ready for training, it needs a few more settings. These are added during the model's compile step (We will explain the following parts in details during the future sections. Use them here as an example, because it is necessary for NN trainings):

- Loss function —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.
- Optimizer —This is how the model is updated based on the data it sees and its loss function.

[2]For more information on the keras code check: `https://keras.io/getting-started/sequential-model-guide/`



(a) (b)

- Metrics —Used to monitor the training and testing steps. The following example uses accuracy, the fraction of the images that are correctly classified.

Exe 14 Train the model: Training the neural network model requires the following steps:

- Feed the training data to the model. In this example, the training data is in the train_images and train_labels arrays.
- The model learns to associate images and labels.
- You ask the model to make predictions about a test set, in this example, the test_images array. Verify that the predictions match the labels from the test_labels array.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

To start training, call the model.fit method, so called because it "fits" the model to the training data:

```
model.fit(train_images, train_labels, epochs=10)
```

`epochs` will be explained in the future sections too.
Exe 15 Evaluate Accuracy: It is the moment for checking the model performance on the test dataset.

```
test_loss, test_acc = model.evaluate(test_images,  test_labels,
    verbose=2)
```

verbose is an Integer value containing 0, 1, or 2. Verbosity mode. 0 = silent, 1 =progress bar, 2 = one line per epoch. We remind you checking the Keras documentation for more details on any part of the code: `https://keras.io/`.

Check the test loss and accuracy in your code.
Exe 16 Make predictions : With the model trained, we can use it to make predictions about some images.

```
predictions = model.predict(test_images)
```

The model predicts a label for each image in the testing set. Print the first, second and third element of the predicted test sets. You can see that each element contains 10 values indicating a probability of each label. Choose the maximum one using `np.argmax()` function. Compare the predicted label of the first three elements with their predicted labels. How many are correct?
Exe 17 Write a function for checking the predicted labels. The result should be similar to the above Figure with a label indicating the probability of the predicted label with blue color if the prediction is correct otherwise in the red color?
Exe 18 Grab a single element from the test set such as test_images[5]. Send it to the `model.predict()` and check what will happen. Why? Correct it by your modification. (hint: you can use `expand_dims()`)
Exe 19 Respecting input and output sizes, try to change your model structure in exercises 12 and 13 and observe their affections on prediction precision.

Exe 20 Choose another image classification dataset from Tensorflow available datasets `https://www.tensorflow.org/datasets/catalog/overview` and predict a classification function for it.