Digit recognition with a CNN

Code to initiliaze Tensorflow 2.0 in Colab

```
from __future__ import absolute_import, division, print_function, unicode_literals
%tensorflow_version 2.x
import tensorflow as tf
%load_ext tensorboard
import datetime
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Import the MNIST dataset. The default loader will return tensors for the train/test partitions of the images and the labels.

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train[:,:,:,np.newaxis]/255.0   # our images are 4 dimensional (NumImages, Height, Width, Channels) - these image
x_test = x_test[:,:,:,np.newaxis]/255.0
```

[TODO] Check the size of the loaded tensors

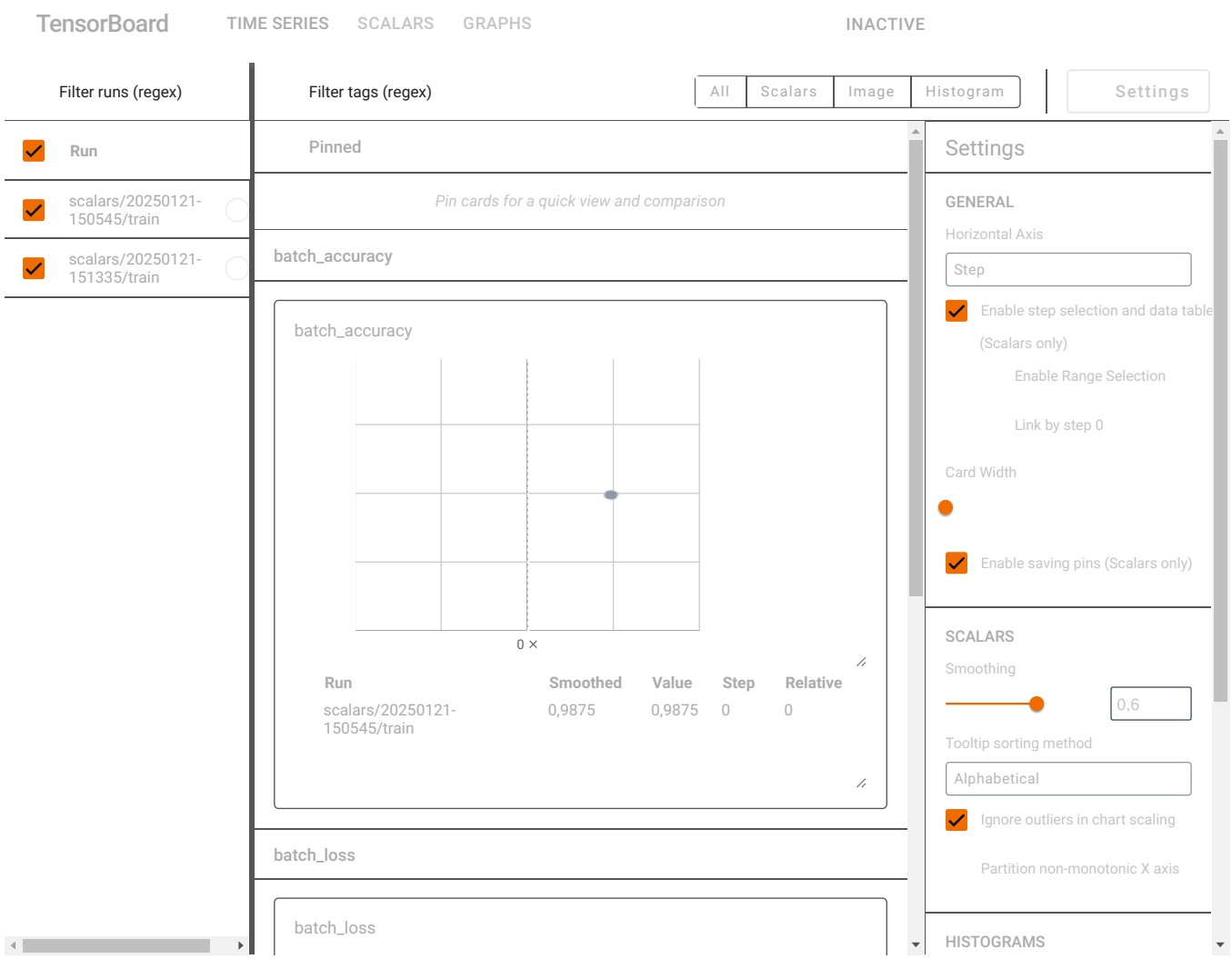
```
print(f'Dimension of X_train: {x_train.shape}') # 6000 images used as training set, each with size 28x28 and only one channel
print(f'Dimension of X_test: {x_test.shape}') # 1000 images used as test set
print(f'Dimension of Y_train: {y_train.shape}') # 6000 labels
print(f'Dimension of Y_test: {y_test.shape}') # 1000 labels
```

```
Dimension of X_train: (60000, 28, 28, 1)
Dimension of X_test: (10000, 28, 28, 1)
Dimension of Y_train: (60000,)
Dimension of Y_test: (10000,)
```

Prepare Keras callback for Tensorboard

```
logdir = "logs/scalars/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
%tensorboard --logdir logs
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir, update_freq='batch')
```

```
Reusing TensorBoard on port 6006 (pid 530), started 0:07:43 ago. (Use '!kill 530' to kill it.)
```



[TODO] Define a Keras Sequential model with the convolutional neural network

```
# we are performing images classification, so given a dataset of labeled images (supervised learning) we want that our model
# for the process of features extraction we use convolutional layers, that performs F different filters on the same image ir
# extracted from the images. These process of features extraction is called convolutional encoding, cause through the convol
# the size of the image. In our case we use 'same' padding, this means that the on the input image are added new pixels in t
# be added equals to 0 (zero padding), or with the same values of the real borders (symmetric padding), or with the same val
# so we give to the convolutional layer the original image, this method is called valid padding and the returned output will
# While we are going deeper in the NN, the convolutional layers should extract more features cause we want to extract more a
# is evaluated on a perceptive field of the size of the kernel used by the convolution, so while we are going deeper each pi
# Between the layers we are also performing Batch Normalization, this is important cause it normalize (by standardization) t
# of the data. This is important cause it reduce the effect of too low value that could bring to vanishing gradient, and toc
# The we use an activation function, that recives the input values and perform on them the ReLu activation function. The act
# non linear, cause if it was linear it will perform the same operations on each input, and so we loose the advantages of ha
# We can decide between sigmoid, ReLu and Leaky ReLu for applying the non linear activation function. Sigmoid function is th
# this cause saturation of the outputs and so vanishing gradient. ReLu is a linear function (y=x) on positive inputs, but it
# the negative input to 0 but it set them with a linear function (y=x/10).
# Between the convolutional layers we are also performing Pooling, these operations reduces the dimension of the image (not
# For applying the classification we use softmax function that returns a pdf between all the 10 possible classes. Before doi
# vector by using Flatten layer. From the pdf of the softmax we read the class with the higher probability. This is how a cc
# Before deciding the class i've added a new layer that perform dropout. This will remove randomly 50% of the neurons from t
# This method reduces the dimension of the neural network but it increases the values of the parameters. Parameters increase
# parameters could be not good, cause high weights made the model very sen sistive on input channges. So we could also perfc
# I have tested that the model with the regularization perform worst than the model without it, so i decided to remove regul
# Note that the NN is divided into two steps, before it applies features extraction and then it performs decision making on
model = tf.keras.models.Sequential([
    # tf.keras.layers.Conv2D(32, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.L2(0.01)),  # extracts basi
    tf.keras.layers.Conv2D(32, (3,3), padding='same'),
    tf.keras.layers.BatchNormalization(), # this layer normalize the output by applying standardization by using the statist
    tf.keras.layers.ReLU(), # non linear activation function. All the negative input are forced to 0, the positive ones are
    tf.keras.layers.MaxPooling2D((2,2)),    # halve the size of the image by taking the maximum value from a 2x2 square
    # tf.keras.layers.Conv2D(64, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.l2(0.01)),   # we increase
    tf.keras.layers.Conv2D(32, (3,3), padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.MaxPooling2D((2,2)),    # halve the size of the image by taking the maximum value from a 2x2 square
```

```
    # tf.keras.layers.Conv2D(128, (3,3), padding='same', kernel_regularizer=tf.keras.regularizers.l2(0.01)),   # we extract
    tf.keras.layers.Conv2D(32, (3,3), padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.Flatten(),  # we flatten the 3D tensor to a 1D tensor, because Dense layer only accepts 1D tensor
    tf.keras.layers.Dropout(0.5),   # we use dropout to avoid overfitting, this layer randomly set 0 some neurons, then the
    tf.keras.layers.Dense(10, activation='softmax') # softmax activation function is used to get the probabilities of each c
])
```

[TODO] Compile the Keras model: specify the optimization algorithm, the loss function and the test metric

```
# Train a NN means trying to find the best weights and biases (parameters of the NN) that can explain in the best way possib
# of error made by our model, so we can understand how far we are from the desired result. This evaluation is done through t
# evaluated between the desired output and the one that our NN returns. After having evaluated the amount error through the
# error. This could be done by evaluating the gradient, so we derive the cost function on each parameter and we obtain a vec
# for reducing the error. Our goal is to arrive to the minimum point of this function by exploiting the gradient, so we eval
# a convergence. This algorithm is called gradient descending, and it ensures that we could reach a local minimum point, but
# the algorithm brings us. Then we can try again with other random parameter and see if the gradient descend converge on bet
# minimum. Note that the gradient gives us only the direction where we have to move, but not the amount of step that we shou
# to evaluate (number of parameters)^2 derivative of the cost function, too complex in system with many parameters as the NN
# A too big step can cause jumping over minimum point and so guide on the wrong direction the algorithm, instead a too short
# We know that the cost function after having applied the step should be lower that the actual value: C(w + Δw) - C(w) = -n
# At each iteration gradient descending must read the entire dataset for evaluating the cost function and then update the pa
# performs the same actions but it doesn't load the entire dataset but it extract from it a batch and then it doesn't put it
# the size of the training set decreases of the batch size. The number of epochs rappresent the number of times that the alg
# by this algorithm is (n / b) * e, where n is the size of the training set, b is the size of the batch and e is the number
# Adam oprimizer is an optmization of the stochastic gradient descent, that changes the learnig rate through the iterations
lr = 0.01   # learning rate
model.compile(optimizer = tf.keras.optimizers.Adam(lr), loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

[TODO] Train the Keras model

```
model.fit(x_train, y_train, batch_size=128, epochs=5, callbacks=[tensorboard_callback])
```

```
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.8758 - loss: 0.4956
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9783 - loss: 0.0678
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9837 - loss: 0.0516
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.9851 - loss: 0.0489
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step - accuracy: 0.9862 - loss: 0.0433
<keras.src.callbacks.history.History at 0x7f13ad65fc10>
```

[TODO] Print model summary

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 28, 28, 32)	320
batch_normalization_4	(None, 14, 14, 32)	128
re_lu_4 (ReLU)	(None, 14, 14, 32)	0

[TODO] Test the Keras model by computing the accuracy the whole test set

```
model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9909 - loss: 0.0267
[0.020742597058415413, 0.9933000206947327]
```

[TODO] Visualize test image number 47 and the prediction from the neural network

```
plt.imshow(x_test[47].reshape(28, 28), cmap='gray')
plt.title(f'Label: {y_test[47]}')
plt.show()

y_pred = model.predict(x_test[47][np.newaxis, :, :, :])
print(f'Predicted label: {np.argmax(y_pred)}')
```