Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning

Week 1 - A New Programming Paradigm

In this week they talk about how machine learning and deep learning works and shows a simple example of training an ML and the results of the approximation to a certain function. Snippet of the code below.

The idea is to get explanatory variables and the target variable, establish a relationship on both of them (Y = 2X - 1) and see the results on how the NN identifies the pattern/function.

```
import tensorflow as tf
import numpy as np
from tensorflow import keras

model = tf.keras.Sequential([keras.layers.Dense(units=1,
input_shape=[1])])

model.compile(optimizer='sgd', loss='mean_squared_error')

xs = np.array([-1.0,  0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model.fit(xs, ys, epochs=500)

print(model.predict([10.0]))
```

Week 2 - Introduction to Computer Vision

This week they talk about a simple NN that learns how to classify images into 10 categories (from 0 to 9), the NN learns from the input (images with 28 per 28 pixels) how to identify those labels.

The input is the greyscale number for each pixel of the image, and this "matrix" is flatten into a single string that will be the input for training the model.

The snippet of the code can be shown below.

```
import tensorflow as tf

class myCallback(tf.keras.callbacks.Callback):
 def on_epoch_end(self, epoch, logs={}):
   if(logs.get('acc')>0.6):
     print("\nReached 60% accuracy so cancelling training!")
     self.model.stop_training = True

mnist = tf.keras.datasets.fashion_mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

callbacks = myCallback()

model = tf.keras.models.Sequential([
 tf.keras.layers.Flatten(input_shape=(28, 28)),
 tf.keras.layers.Dense(512, activation=tf.nn.relu),
 tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10, callbacks=[callbacks])
model.evaluate(x_test, y_test)

classifications = model.predict(test_images)
```

Week 3 - Enhancing Vision with Convolutional Neural Networks

In the "Try it for yourself" part, I made several changes in the settings of the layers in the NN and those were the results:

-> Conv2D (32)

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0034 - accuracy: 0.9988
313/313 [==============================] - 1s 3ms/step - loss: 0.0664 - accuracy: 0.9861

-> Conv2D (16)

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0043 - accuracy: 0.9986
313/313 [==============================] - 1s 3ms/step - loss: 0.0624 - accuracy: 0.9854

-> Conv2D (64)

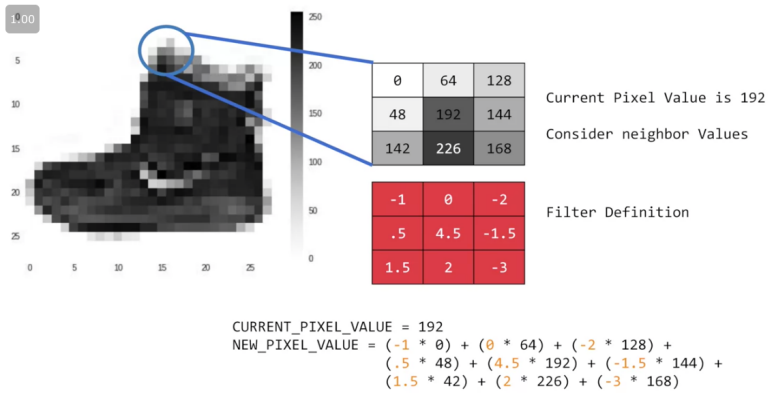
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0023 - accuracy: 0.9994
313/313 [==============================] - 1s 4ms/step - loss: 0.0498 - accuracy: 0.9879

-> Conv2D (64) + Conv2D (16)

1875/1875 [==============================] - 9s 5ms/step - loss: 0.0081 - accuracy: 0.9974
313/313 [==============================] - 1s 4ms/step - loss: 0.0310 - accuracy: 0.9922

It's relevant to mention how Convolution and Pooling work:

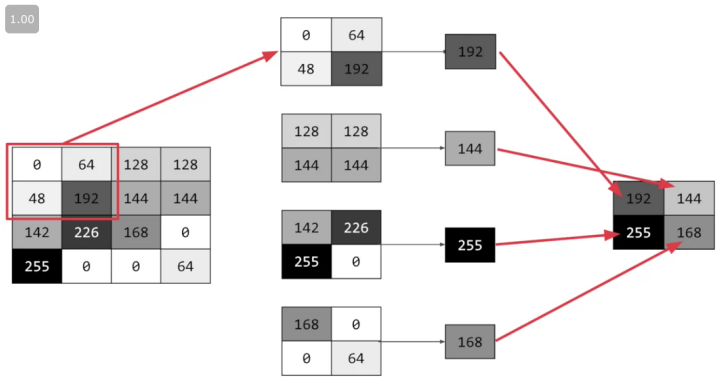
- Convolution is a way of creating filters and training the DNN on top of it to get better performance. Each filter is applied on top of the pixels (shown below)



Snippet:

```
tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
```

- Pooling is a way of compressing an image (shown below)



Snippet:

```
tf.keras.layers.MaxPooling2D(2, 2),
```

Week 4 - Using real-world images

This week the instructor introduces the concept of data generator, using the validation dataset during the fit of the model. The snippet is below.

Here the datasets (training and validation) are generated.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1/255)
validation_datagen = ImageDataGenerator(rescale=1/255)

# Flow training images in batches of 128 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        '/tmp/horse-or-human/',  # This is the source directory for training images
        target_size=(300, 300),  # All images will be resized to 300x300
        batch_size=128,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow training images in batches of 128 using train_datagen generator
validation_generator = validation_datagen.flow_from_directory(
        '/tmp/validation-horse-or-human/',  # This is the source directory for training images
        target_size=(300, 300),  # All images will be resized to 300x300
        batch_size=32,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')
```

Here the model is built with different layers ending with one to handle binary problems.

```
model = tf.keras.models.Sequential([
   # Note the input shape is the desired size of the image 300x300 with 3 bytes color
   tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(300, 300, 3)),
   tf.keras.layers.MaxPooling2D(2, 2),

   tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2,2),

   tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2,2),

   tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2,2),

   tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
   tf.keras.layers.MaxPooling2D(2,2),
   # Flatten the results to feed into a DNN
   tf.keras.layers.Flatten(),
```

```
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron. It will contain a value from 0-1 where 0 for 1 class ('horses') and
1 for the other ('humans')
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

Here the model is compiled and the loss function, optimizer and metrics to be used defined.

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

Here is the fit of the model, the training per se. Notice that the validation dataset is taking into account.

```
history = model.fit(
      train_generator,
      steps_per_epoch=8,
      epochs=15,
      verbose=1,
      validation_data = validation_generator,
      validation_steps=8)
```

Finally, here is a snippet for classifying the image.

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():
 # predicting images
 path = '/content/' + fn
 img = image.load_img(path, target_size=(300, 300))
 x = image.img_to_array(img)
 x = np.expand_dims(x, axis=0)

 images = np.vstack([x])
 classes = model.predict(images, batch_size=10)
 print(classes[0])
 if classes[0]>0.5:
   print(fn + " is a human")
 else:
   print(fn + " is a horse")
```