TensorFlow For Practice

By: Mohamed Aziz Tousli

Basics

How to “learn”?
1- Make a guess.
2- The LOSS function measures the guessed answers against the known correct answers
3- The OPTIMIZER function makes another guess. Based on how the loss function went, it will try to minimize the loss.
4- It will repeat this for the number of EPOCHS.

- #Necessary libraries

```
import tensorflow as tf
from tensorflow import keras
```

- #Import an already existing dataset

```
data_set_name = keras.datasets.data_set_name
(train_images, train_labes), (test_images, test_labels) = data_set_name.load_data()
train_images = train_images / 255.0 #Normalization
```

- #Basic code

```
model = /* model definition here */
model.compile(optimizer=‘optimizer_name’, loss=‘loss_name’, metrics=[‘metrics_name’])
model.fit(x, y, epochs=number_of_epochs, callbacks=[callbacks])
model.evaluate(x_test, y_test)
model.predict([10])
model.summary() #Summary of model architecture
```

Callback to stop training on condition

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('loss')<0.4):
      print("\nLoss is low so cancelling training!")
      self.model.stop_training = True
```

callbacks = myCallback()

→ model.fit(x, y, epochs=number_of_epochs, callbacks=[callbacks])

Model for Image Classification

model = keras.Sequential([#Define a sequence of layers

keras.layers.Flatten(input_shape=(x_pixel_size, y_pixel_size, 1|3)), #nD → 1D

keras.layers.Dense(number_of_neurons, activation=tf.nn.relu), #First layer of neurones

→ If multi classification: keras.layers.Dense(number_of_classes, activation='softmax')

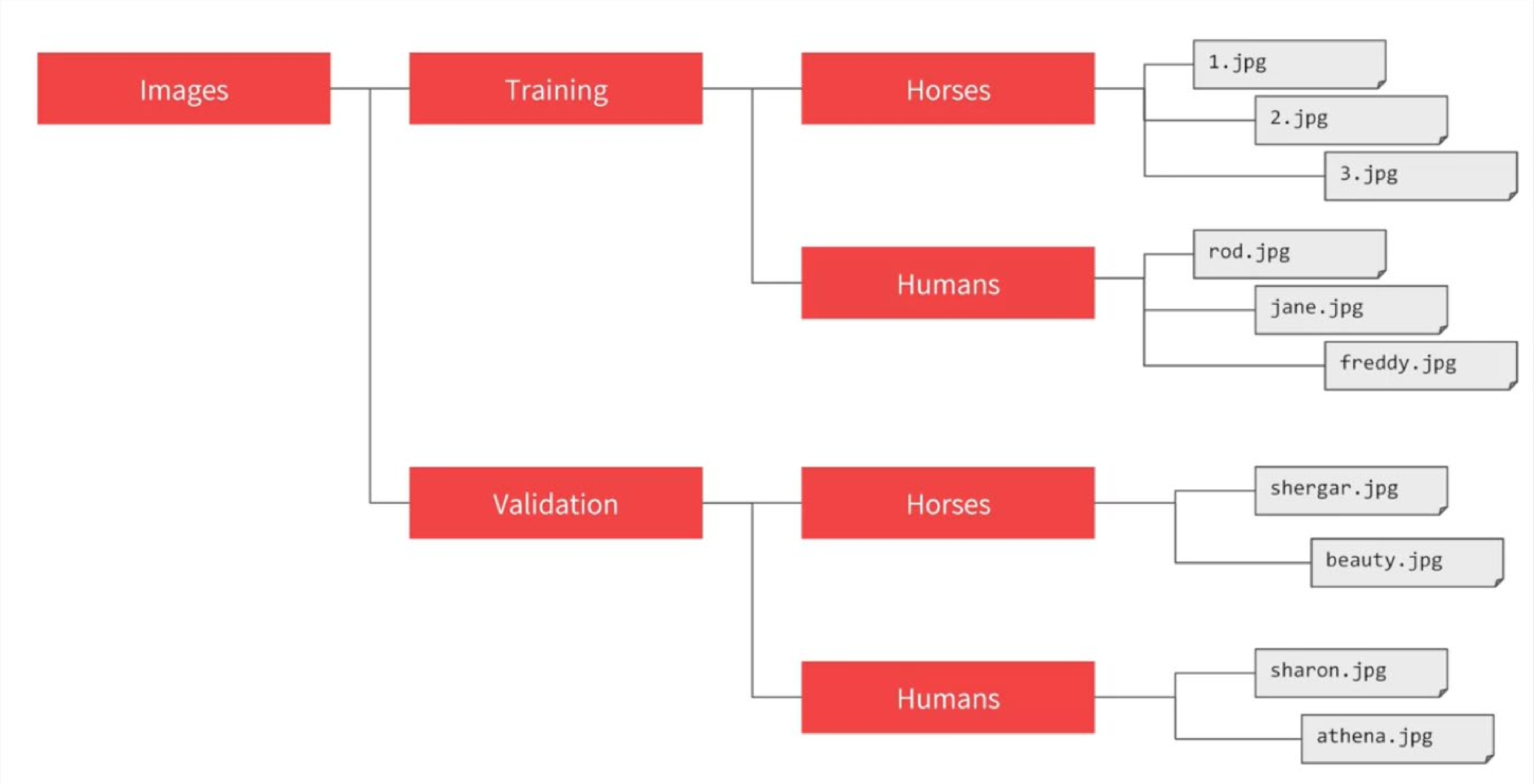
→ If binary classification: keras.layers.Dense(1, activation='sigmoid')

])

Model for Image Classification + Convolution

```
training_images = training_images.reshape(number_of_images, x_image_size, y_image_size, 1)
model = keras.Sequential([
    keras.layers.Conv2D(number_of_filters, (x_filter_size, y_filter_size), activation='relu',
                        input_shape=(x_image_size, y_image_size,1|3)),
    keras.layers.MaxPooling2D((x_pooling_size, y_pooling_size),
    keras.layers.Conv2D(number_of_filters, (x_filter_size, y_filter_size), activation='relu')),
    keras.layers.MaxPooling2D((x_pooling_size, y_pooling_size),
    keras.layers.Flatten(),
    keras.layers.Dense(number_of_neurons, activation='relu'),
    keras.layers.Dense(number_of_classes, activation='softmax')])
```

ImageDataGenerator



ImageDataGenerator

- **Goal**: Read images from subdirectories, and automatically label them

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(x_image_size, y_image_size),
    batch_size=size_of_batch,
    class_mode='binary'|'categorical')
```

- PS: Same thing has to be done for validation_generator
- fit → fit_generator | evaluate → evaluate_generator | predict → predict_generator

```
history = model.fit_generator(
    train_generator, steps_per_epoch=8, epochs=15, verbose=2,
    validation_data=validation_generator, validation_steps=8)
```

Prediction with upload button

```
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")
```

Data Augmentation with ImageDataGenerator

```
# Updated to do image augmentation
train_datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

These are just a few of the options available (for more, see the Keras documentation. Let's quickly go over what we just wrote:

- rotation_range is a value in degrees (0–180), a range within which to randomly rotate pictures.
- width_shift and height_shift are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- shear_range is for randomly applying shearing transformations.
- zoom_range is for randomly zooming inside pictures.
- horizontal_flip is for randomly flipping half of the images horizontally. This is relevant when there are no assumptions of horizontal assymmetry (e.g. real-world pictures).
- fill_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

Transfer Learning

- from tensorflow.keras import Model
- from tensorflow.keras.applications.inception_v3 import InceptionV3
- local_weights_file = ‘h_5_file’
- pre_trained_model = InceptionV3(input_shape=(x,y,1|3), include_top=False, weights=None)
- pre_trained_model.load_weights(local_weights_file)
- for layer in pre_trained_model.layers:

  layer.trainable = False
- last_layer = pre_trained_model.get_layer(‘last_layer_name’) #Get last layer of Inception
- last_output = get_layer.output
- x = layers.Flatten()(last_output) #Work from last layer of Inception (Transfer Learning)
- x = layers.Dense(1024, activation=‘relu’)(x)
- x = layers.Dense(1, activation=‘sigmoid’)(x)
- x = layers.Dropout(0.2)(x)
- model = Model(pre_trained_model.input, x)

NLP

- from tensorflow.keras.preprocessing.text import Tokenizer #Tokenizer
- from tensorflow.keras.preprocessing.sequence import pad_sequences #Padding to the longest sentence
- sentences = [‘sentence1’, ‘sentence2’]
- tokenizer = Tokenizer(num_words = vocab_size, oov_token=“<OOV>”)
- tokenizer.fit_on_texts(sentences)
- word_index = tokenizer.word_index # word_index = {‘a’: 1, ‘dog’: 2, …}
- sequences = tokenizer.texts_to_sequences(sentences) # sequences = [[4,2,3,1], …]
- padded = pad_sequences(sequences) # padded = [[0,0,0,4,2,3,1], …]
  - <u>Other arguments</u>: padding=‘post’, maxlen=max_length, truncating=trunc_type
- tokenized_string = tokenizer.encode(‘sample_string’)
- original_string = tokenizer.decode(tokenized_string)

NLP with a Dataset

- import tensorflow_datasets as tfds
  - db_name, info = tfds.load(“db_name”, with_info=True, as_supervised=True)
  - train_data, test_data = db_name[‘train’], db_name[‘test’]
  - tokenizer = info.features[‘text’].encoder
- training_sentences, training_labels, testing_sentences, testing_labels = [], [], [], []
- for s,l in train_data:

  training_sentences.append(str(s.tonumpy()))

  training_labels.append(l.numpy())
- for s,l in test_data:

  testing_sentences.append(str(s.tonumpy()))

  testing_labels.append(l.numpy())
- model = tf.keras.Sequential([

  tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),

  tf.keras.layers.GlobalAveragePooling1D(), #tf.keras.layers.Flatten() ← We can use this instead

  tf.keras.layers.Dense(6, activation=‘relu’),

  tf.keras.layers.Dense(1, activation=‘sigmoid’)])
- model.fit(training_padded, training_labels, epochs=num_epochs, validation_data=(testing_padded, testing_labels))

NLP with LSTM | GRU

- model = tf.keras.Sequential([

  tf.keras.layers.Embedding(tokenizer.vocab_size, 64),

  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM|GRU(64, return_sequences=True)),

  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)), #If we add this layer → return_sequences=True

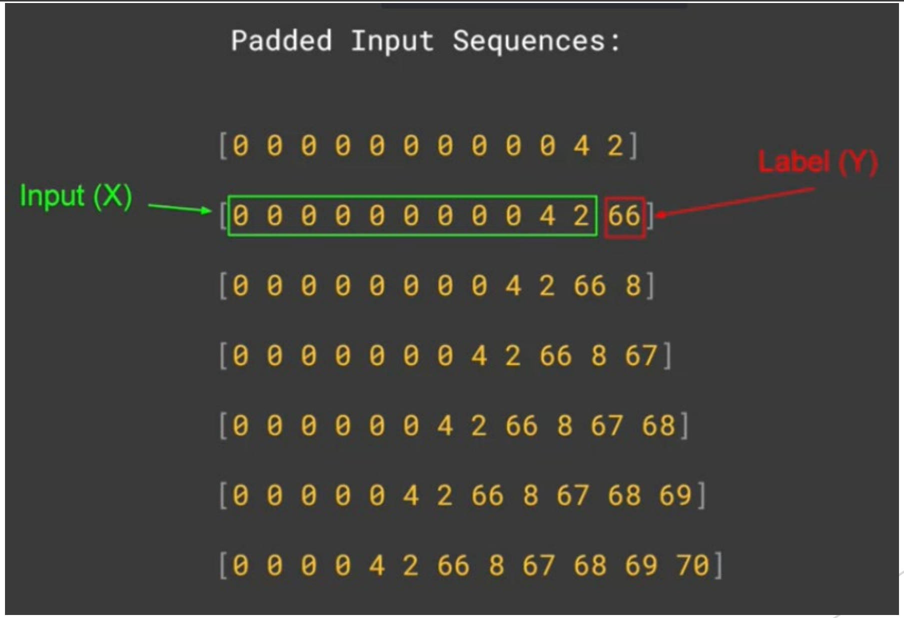
  tf.keras.layers.Dense(64, activation='relu'),

  tf.keras.layers.Dense(1, activation='sigmoid')])

- PS:
  - We can use Bidirectional() instead of Flatten() & GlobalAveragePooling1D()
  - We can use Conv1D() & GlobalMaxPooling1D() instead of Bidirectional()

Predict Next Word {1} - Preprocessing

```python
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
max_sequence_len = max([len(x) for x in input_sequences])
```



Predict Next Word {2} - Preprocessing

```
xs = input_sequences[:,:-1]
labels = input_sequences[:,-1]

ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```

```
Sentence: [0 0 0 0 4 2 66 8 67 68 69 70]

       X:[0 0 0 0 4 2 66 8 67 68 69]

    Label:[ 70 ]

       Y:[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
           0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
```

Predict Next Word {2} - Training

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=100, verbose=1)
```

Predict Next Word {2} - Predicting

```
seed_text = "Laurence went to dublin"
next_words = 10

for _ in range(next_words):
  token_list = tokenizer.texts_to_sequences([seed_text])[0]
  token_list = pad_sequences([token_list], maxlen=max_sequence_len - 1, padding='pre')
  predicted = model.predict_classes(token_list, verbose=0)
  output_word = ""
  for word, index in tokenizer.word_index.items():
    if index == predicted:
      output_word = word
      Break
  seed_text += " " + output_word
print(seed_text)
```

Time Series

- **Notebooks**:
  - Introduction to time series: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP_Week_1_Lesson_2.ipynb
  - Forecasting: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%201%20-%20Lesson%203%20-%20Notebook.ipynb
  - Preparing features and labels: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%202%20Lesson%201.ipynb
  - Single layer neural network: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%202%20Lesson%202.ipynb
  - Deep neural network: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%202%20Lesson%203.ipynb

Time Series

- **Notebooks**:
  - RNN: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%203%20Lesson%202%20-%20RNN.ipynb
  - LSTM1: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%203%20Lesson%204%20-%20LSTM.ipynb
  - LSTM2: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%204%20Lesson%201.ipynb
  - Sunspots1: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%204%20Lesson%205.ipynb
  - Sunspots2: https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20In%20Practice/Course%204%20-%20S%2BP/S%2BP%20Week%204%20Lesson%203.ipynb