AI Tutorial

IMP Links

* <https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb#scrollTo=he5u_okAYS4a>
* <https://www.tensorflow.org/lite/guide/hosted_models> (All Tensor Flow Models)
* <https://www.tensorflow.org/lite/examples> (Examples of Tensorflow Models)
* <https://www.tensorflow.org/lite/guide> (Starting guide for Tensorflow Model Development)
* https://www.tensorflow.org/lite/guide/android (Guide for Tensorflow Model Development on Android)
* <https://www.tensorflow.org/lite>
* [Intro to TensorFlow for Deep Learning - Udacity](https://classroom.udacity.com/courses/ud187/lessons/7b590cdb-0acf-4118-848c-8728ced19bc6/concepts/4e1b5f7e-5094-481e-ae29-284bc2fb847f) (Crash Course for beginners to learn Deep Learning) – added on 04-05-2022
* TensorFlow Lite is a set of tools that enables on-device machine learning by helping developers run their models on mobile, embedded, and edge devices.

**Development workflow**

### **1. Generate a TensorFlow Lite model**

A TensorFlow Lite model is represented in a special efficient portable format known as [FlatBuffers](https://google.github.io/flatbuffers/). This provides several advantages over TensorFlow's protocol buffer model format such as reduced size (small code footprint) and faster inference (data is directly accessed without an extra parsing/unpacking step) that enables TensorFlow Lite to execute efficiently on devices with limited compute and memory resources.

You can generate a TensorFlow Lite model in the following ways:

* **Use an existing TensorFlow Lite model:** Refer to [TensorFlow Lite Examples](https://www.tensorflow.org/lite/examples) to pick an existing model. Models may or may not contain metadata.
* **Create a TensorFlow Lite model:** Use the [TensorFlow Lite Model Maker](https://www.tensorflow.org/lite/guide/model_maker) to create a model with your own custom dataset. By default, all models contain metadata.
* **Convert a TensorFlow model into a TensorFlow Lite model:** Use the [TensorFlow Lite Converter](https://www.tensorflow.org/lite/convert/index) to convert a TensorFlow model into a TensorFlow Lite model. During conversion, you can apply [optimizations](https://www.tensorflow.org/lite/performance/model_optimization) such as [quantization](https://www.tensorflow.org/lite/performance/post_training_quantization) to reduce model size and latency with minimal or no loss in accuracy. By default, all models don't contain metadata.

**Simple ML Code to create a model and to predict Temperature from Celsius to Fahrenheit using ANN**

Link for the code : https://colab.research.google.com/drive/1MUg0qYcnmmGV6Iw3926WTSjL9HTDXVe6#scrollTo=mp7NnuK5jzAO

import tensorflow as tf

import numpy as np

#input data which is in Celsius format

x = np.array([0, 10, 20, 30, 38, 32], dtype=float)

y = np.array([32, 50, 68, 86, 100, 90], dtype=float)

#Creating Layers to process the input data and to predict the output

l1 = tf.keras.layers.Dense(units=4, name='l1', input\_shape=[1])

l2 = tf.keras.layers.Dense(units=4, name='l2')

l3 = tf.keras.layers.Dense(units=1, name='l3')

#Creating Model

model = tf.keras.Sequential([l1, l2, l3])

#Compile the model

model.compile(

    loss='mean\_squared\_error',

    optimizer=tf.keras.optimizers.Adam(0.1),

)

#Train the model

history = model.fit(x, y, epochs=500,verbose=False)

#Visualize the loss rate

import matplotlib.pyplot as plt

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.plot(history.history['loss'])

#predict the model

print(model.predict([100.0]))

# Summary

In this lesson we trained a neural network to classify images of articles of clothing. To do this we used the Fashion MNIST dataset, which contains 70,000 greyscale images of articles of clothing. We used 60,000 of them to train our network and 10,000 of them to test its performance. In order to feed these images into our neural network we had to flatten the 28 × 28 images into 1d vectors with 784 elements. Our network consisted of a fully connected layer with 128 units (neurons) and an output layer with 10 units, corresponding to the 10 output labels. These 10 outputs represent probabilities for each class. The softmax activation function calculated the probability distribution.

We also learned about the differences between regression and classification problems.

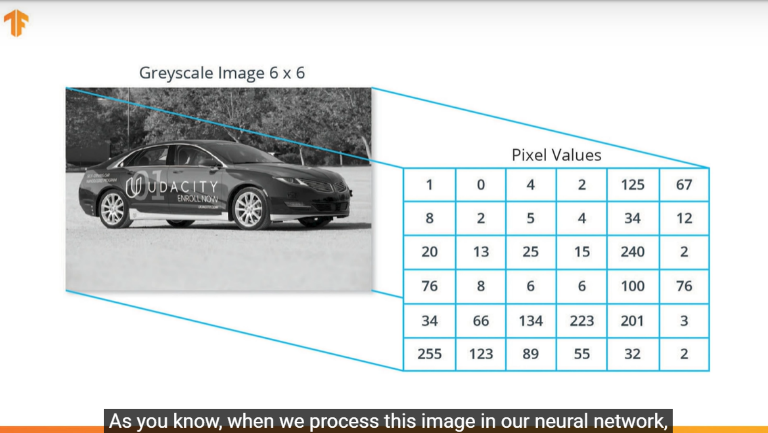
* **Regression:** A model that outputs a single value. For example, an estimate of a house’s value.
* **Classification:** A model that outputs a probability distribution across several categories. For example, in Fashion MNIST, the output was 10 probabilities, one for each of the different types of clothing. Remember, we use Softmax as the activation function in our last Dense layer to create this probability distribution.

Table

Description automatically generated

**Convolutional Neural Network**

* A convolution is the process of applying a filter (“kernel”) to an image. Max pooling is the process of reducing the size of the image through downsampling.
* convolutional layers can be added to the neural network model using the Conv2D layer type in Keras.
* This layer is similar to the Dense layer, and has weights and biases that need to be tuned to the right values.
* The Conv2D layer also has kernels (filters) whose values need to be tuned as well.
* So, in a Conv2D layer the **values** inside the **filter matrix** are the **variables** that get tuned in order to produce the right output.
* CNN is mostly used for high dimension image like 1920 x 1820

 Diagram

Description automatically generated

Here are some of terms that were introduced in this lesson:

* **CNNs:** Convolutional neural network. That is, a network which has at least one convolutional layer. A typical CNN also includes other types of layers, such as pooling layers and dense layers
* **Convolution:** The process of applying a kernel (filter) to an image
* **Kernel / filter:** A matrix, which is smaller than the input, used to transform the input into chunks
* **Padding:** Adding pixels of some value, usually 0, around the input image

A picture containing diagram

Description automatically generated

* **Pooling** The process of reducing the size of an image through downsampling. There are several types of pooling layers. For example, average pooling converts many values into a single value by taking the average. However, maxpooling is the most common.

Diagram

Description automatically generated

* **Maxpooling**: A pooling process in which many values are converted into a single value by taking the maximum value from among them.

Graphical user interface

Description automatically generated

* **Stride:** the number of pixels to slide the kernel (filter) across the image.

In above image, stride was 2. hence 2x2 maxpooling matrix will be moved to 2 pixels to the right

* **Downsampling:** The act of reducing the size of an image

**Note :** The main purpose of Convolution Neural Network is to reduce size of an image using some Maxpooling techniques and pass the reduced images to the remaining layers

A picture containing shape

Description automatically generated

**How The Convolved Future Calculated ?**

* For the First element,

1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 0x0 + 1x1 = 4

Same for remaining elements, where Image will be dot product by **Filter Matrix**

* Then ReLu Layer will identifies -ve values from Convolved Future and replace them with zero.
* This process will be repeated till all the features of the image were rectified called as **future map.**
* **Pooling** layer will do downsampling which means it will take nxn matrix of the **future map** and identifies max value of the matrix(ex: 2x2) and replace entire matrix with those values of an image which is used to reduce size of an image. Mostly **MaxPooling** will be used.
* **Finally,** Those future maps will be flattened into 1D array using **Flatten layer**
* Output of **Flatten layer** will be passed to Fully connected Layer like **Dense Layer**

Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 convolved feature

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, e.g., RGB)

* In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix .**
* The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.

Diagram

Description automatically generated

* The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

A picture containing diagram

Description automatically generated

Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

Shape

Description automatically generated

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

There are two types of results to the operation — one in which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same. This is done by applying **Valid Padding** in case of the former, or **Same Padding** in the case of the latter.

Shape

Description automatically generated

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **Same Padding**.

On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding**.

**How Convolution Neural Network Work on Image Classification?**

Table

Description automatically generated

In above Image, left side is the matrix representation of image pixels

We perform 3x3 matrix multiplication with kernel on the middle and output of new image will be calculated. This is called **Convolution operation.**

Here stride will be taken to move that pooling matrix in the input grid

The right hand side grid is called **future map.** Filters are nothing but future detectors.

Diagram

Description automatically generated

So we iteratively apply **convolution operation** on future maps as well till we find entire image.

Diagram

Description automatically generated

Chart

Description automatically generated with medium confidence