Regulations

We ask you to include your solutions (the code with every file needed) in a **zip folder** and upload it on Moodle. Include the names of all members (not more than 3) on top of your jupyter notebook as a markdown cell. **The submission deadline** is always the next week **before** the beginning of the class on **Wednesday at 09:14**. For clarification on the code, please try to include comments in order for it to be understandable.

1 Implementing Convolution Based on Look-up Table

Preparation:
In this first homework, you will build upon Exercise 1 from the first exercise group where you were introduced to the concept of convolution. Here, you are asked to implement the same program as done in the exercise group, this time however we want you to substitute the matrix multiplication of pythons intern numpy library with our own quantized look-up table.

> A look-up table is a predefined matrix that already has all possible (integer!) values multiplied that we need. Thus we can skip the task of computing the multiplication by simply looking it up. We are going to use that for the same convolution such that we can cut computational cost by not repeating the "cost-heavy" matrix multiplication with floating point values as it is done in every step of the convolution operation.

1.1 Subtask 1: Implement The Manual Convolution from the Exercise group

Exactly as done in the exercise group, we start off by implementing a convolution algorithm. This shall refresh your knowledge of what a convolution is and also hopefully help you compare the methods when implementing the look-up table.

Lets begin: We shortly outline each step and already give you the code for this task (you have already seen it in the exercise group)

1. **Import Modules**

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from scipy import signal
from tensorflow import keras
```

2. **Load Images**

```
img=np.load("sample.npy")
img=np.floor(img/2)
fltr=np.load("filter.npy")
```

3. **Implement the Convolution Function**

```
def Myconv2d(a,b):
    a=np.array(a)
    b=np.array(b)
    a_shape=np.shape(a)
    b_shape=np.shape(b)
    res_shape1=np.abs(a_shape[0]-b_shape[0])+1
    res_shape2=np.abs(a_shape[1]-b_shape[1])+1
```

```
    res=np.zeros([res_shape1,res_shape2])
    for i in range(res_shape1):
        for j in range(res_shape2):
            res[i,j]=np.sum(
                np.multiply(
                    np.flip(b),a[i:i+b_shape[0],j:j+
                        b_shape[1]]))
    return res
```

4. **Check with Pythons Convolution**

```
Result_Imp=Myconv2d(img,fltr)
Result_Python=signal.convolve2d(img,fltr,mode="valid")

checkup=np.sum(np.abs(Result_Imp-Result_Python))
print("The Error Valuse is:",checkup)
```

1.2 Subtask 2: The look-up table:

Having repeated the steps as discussed in the exercise group, we are now ready to implement the look-up table. For this, we want to substitute numpys element-wise matrix multiplication `np.multiply` with our own predefined function that we are going to call `My_Mult(a,b)`. [1]

> We remind you, that an element-wise multiplication between two matrices is **not** a matrix multiplication. As the name implies, multiplying two matrices element-wise (for this they must have the same shape) results in a matrix of the same shape with the i,j-th element being a multiplication of the i,j-th elements of both matrices. In physicist notation, if $c = a*b$ then $c_{ij} = a_{ij} \cdot b_{ij}$.

1. **First off:**
   The function `My_Mult` will take on the role of a matrix element-wise multiplication that however works with a predefined integer-valued look-up table that we will call `Multiplier`. **In this subtask you are only asked to implement the look-up table** `Multiplier`. In the next subtask you will implement `My_Mult`.

2. **How does the look-up table look like?**
   The integer-valued look-up table `Multiplier` should be a numpy matrix of the shape `(256,256)` where the element in the i-th row and j-th column has the value `(i-128)*(j-128)`. This will be the basis for the look-up table. Think of how the values within this matrix can be used to estimate an element-wise matrix multiplication. This will be important for the next subtask.

3. **Why quantized?**
   We will use the word quantized multiplication because the look-up table we are going to generate only contains integer multiplications. The process of looking up the results of a floating point value multiplication via an integer look-up table is in itself an estimation.

1.3 Subtask 3: Define Matrix Multiplication Function

Having defined the `Multiplier`-matrix (the look-up table!), use this matrix to define a function `My_Mult(a,b)` that takes as an input two matrices `a` and `b` of the same shape and returns a resulting matrix `res` of the same shape as well that is an element-wise multiplication of `a` and `b`.
Hint: You will have to convert the values of the matrices `a` and `b` to integers. To these integers add 128 and use the value to look-up the result of a quantized multiplication in `Multiplier`. Why did this help us save computational costs?

[1] The parameters a and b given to this function are the same matrices as `np.multiply` takes.

1.4 Subtask 4: Implement Look-up based Convolution

First, use the newly defined `My_Mult` in a convolution function that will take on the same role as pythons `My_conv2d`. This time, however, we will not use pythons element-wise matrix multiplication `np.multiply` but our newly defined `My_Mult` function instead.
Then compare the accuracy of the convolution by again checking the error compared to pythons `conv2d` just as it is done in subtask 1. Why does the accuracy go down?