# Project 2 - Computer Vision

**Introduction**

In this exercise, you're going to train an image classification algorithm using the CIFAR10 dataset. This dataset contains 60,000 small images, each 32x32 pixels, categorized into 10 different classes (such as airplane, automobile, etc.). The compact size of these images enables fast experimentation with various Computer Vision principles without the need for a GPU. We will use a subset of the dataset for faster experimentation.

**Description**

The code provided trains an image classifier using a subset of the CIFAR10 dataset and the MobileNet architecture as its foundation. It runs for 5 epochs, with every layer except the final one being set to non-trainable (frozen).

**Installation**

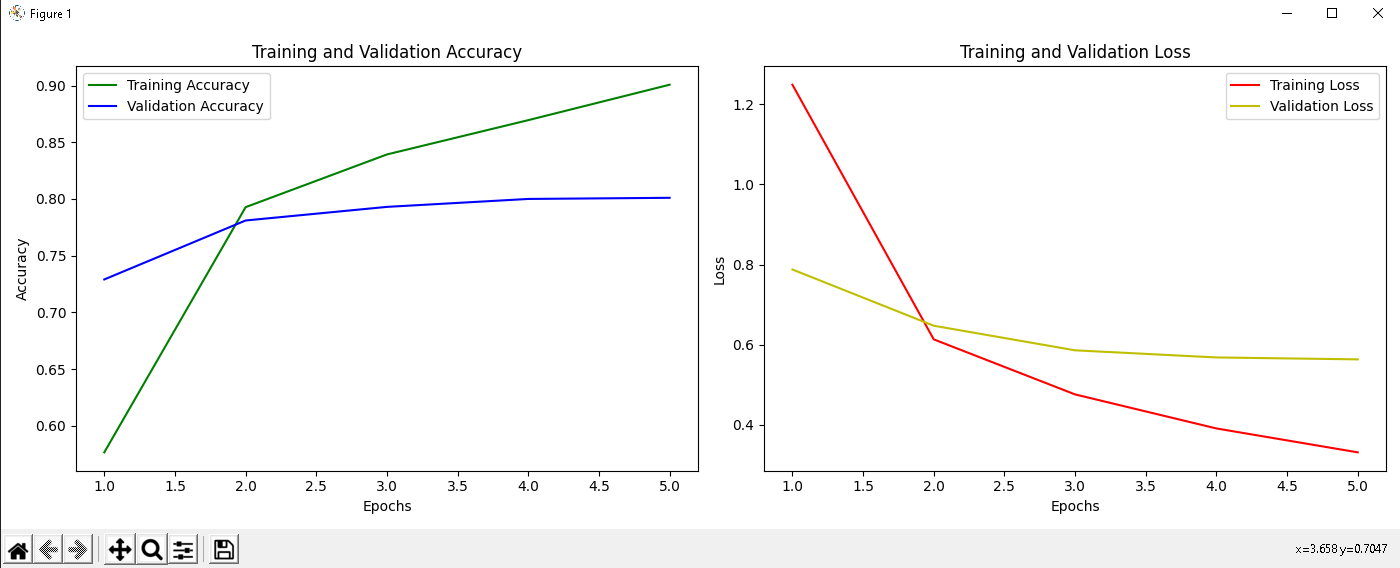
Create a new Anaconda environment with Python 3.8.19. Use pip to install all required libraries from requirements.txt

1. Create new anaconda environment -
2. Activate the environment - conda activate myenv
3. **pip install -r requirements.txt**

**Exercise**

1. Create a function that draws a plot using the variable *'history*'.

* Draw a plot of train/validation accuracy per epoch using matplotlib. Draw the train accuracy in **green** and validation in **blue**.
* Draw a plot of train/validation loss per epoch using matplotlib. Draw the train loss in **red** and validation loss in **yellow**.



2. Based on the plots,

1. Do you think that training for more epochs will improve the results? Explain.

No, and there are two reasons :

* + - 1. as we can see the validation and training are drifting away, both for loss and  
         for accuracy.   
         this points to us about an **overfitting** issue, when the **training** is getting better   
         but **validations** is not showing same progress and just drifting away.
      2. We are using *base\_model* that is aggregation of *MobileNet* , and within the code we set ***freeze*** on the weights (*base\_model.trainable = false*). This means that even if there was a productive learning from the training data – it wouldn’t had happened

1. Do you think that increasing the dataset will improve the results?

Since we have an overfitting is not only a case of lack of data, but also a lack of model complexity. I guess that model is not complex enough for this data and we might need also to add not only data – but also some layers to improve learning

3. For each of the following (A,B,C,D,E), write down the test accuracy, and training time. **Hint** : (<https://docs.python.org/3/library/timeit.html>).

For each item,

* Explain (in one sentence) why the accuracy improved. .
* Explain (in one sentence) why training is slower, faster or hasn’t changed.

**Note**: Changes are cumulative

First, write down the basic test accuracy and inference time in the first row.

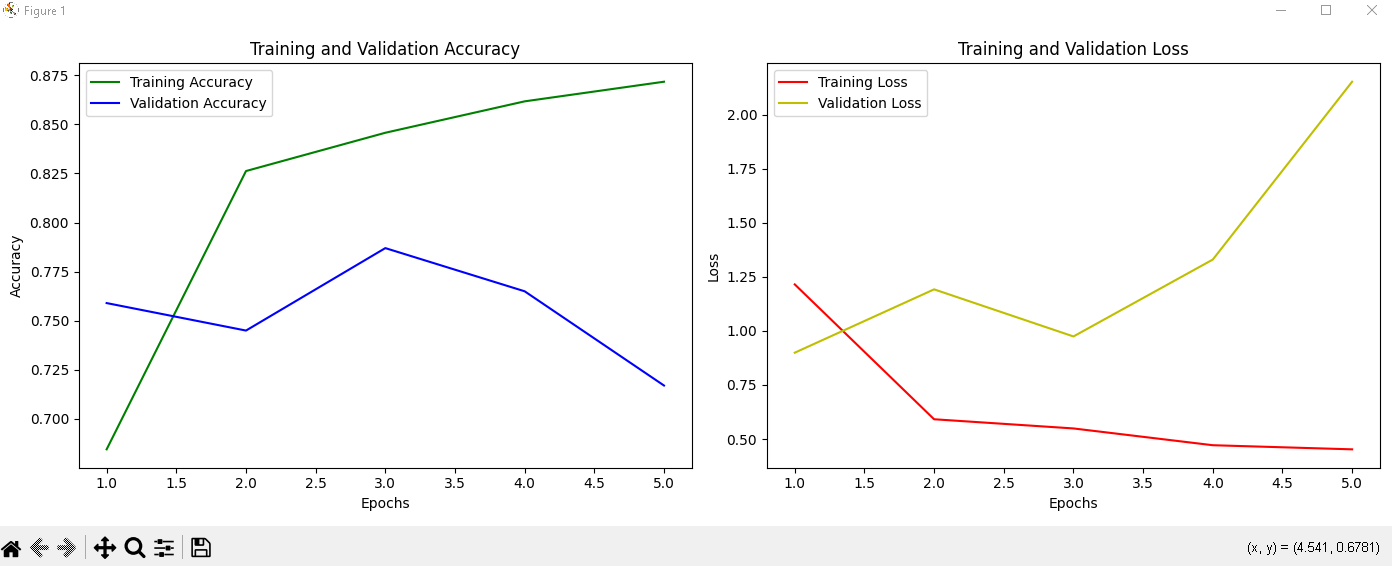
I did not use ***timeit***, since ***timeit.timeit*** is executing without the result of the execution   
so we should execute twice – once for timing and one for results   
so I wrote a module, named it timing util and over there execute the function while timing it   
  
function name : capture\_and\_time and this is exactly what is does:   
  
and call for the function is with pointers for the arguments and the function name :

model, model\_execution\_time = capture\_and\_time(

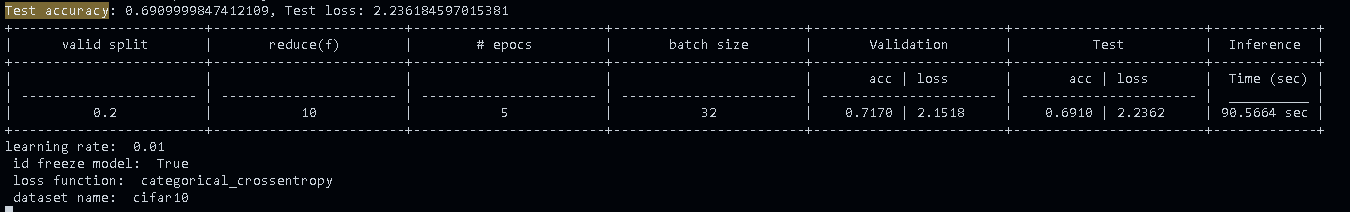
    create\_model, train\_images=train\_images, image\_size=const.IMAGE\_SIZE

)

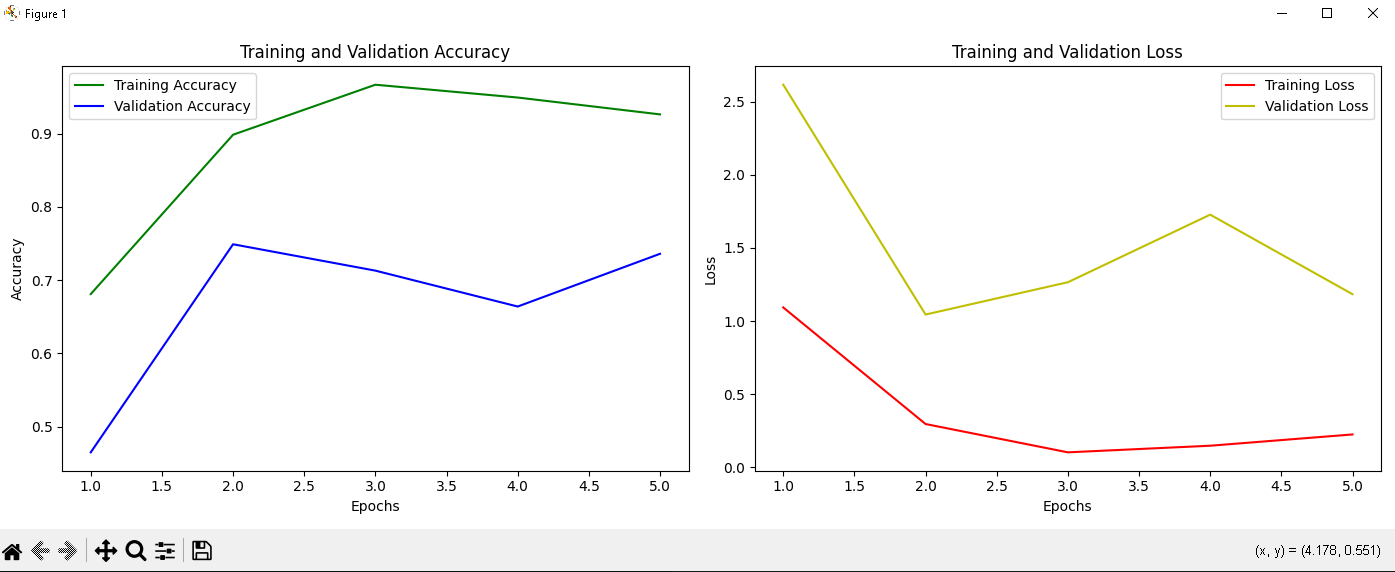
1rst execution :

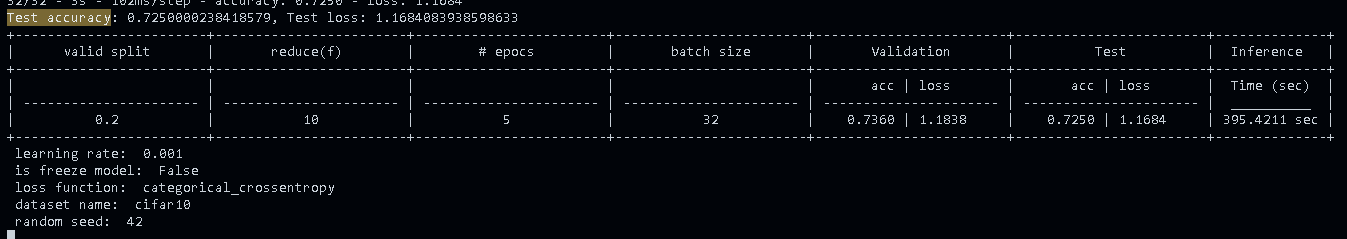


|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| validation  split | decrease factor | epocs | Batch size | Test | | Validation | | Inference time |
| accuracy | loss | accuracy | loss |
|  |  |  |  |  |  |  |  |  |

  
random seed = 42

A. Unfreeze the layers (*base\_model.trainable*) so that the whole network can learn. Hint : <https://keras.io/guides/transfer_learning/>

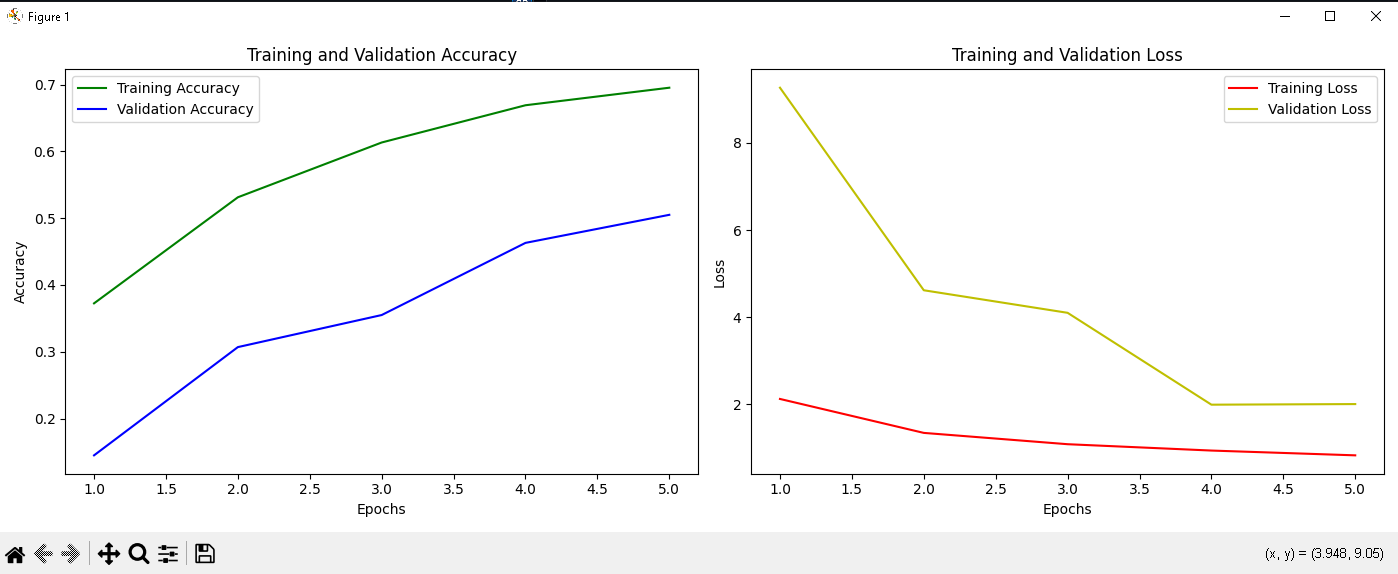




Training is slower since there is a proper learning.  
before the mode was freeze meaning no learning, weights are adjusted over gradiant descent calculation and he back propagation process is taking time.

Accuracy improved since we have learning with data.  
before the weights where not adjusted properly and accuracy was lousy.

B. Change the learning rate from the default *learning\_rate* to 0.01. Hint: <https://keras.io/api/optimizers/>



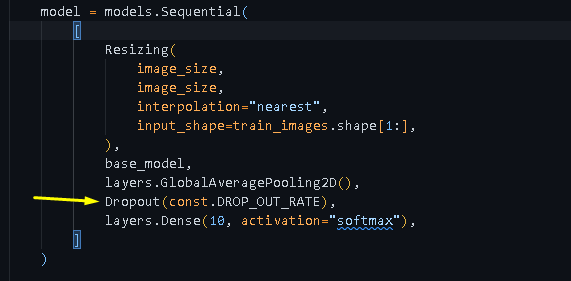


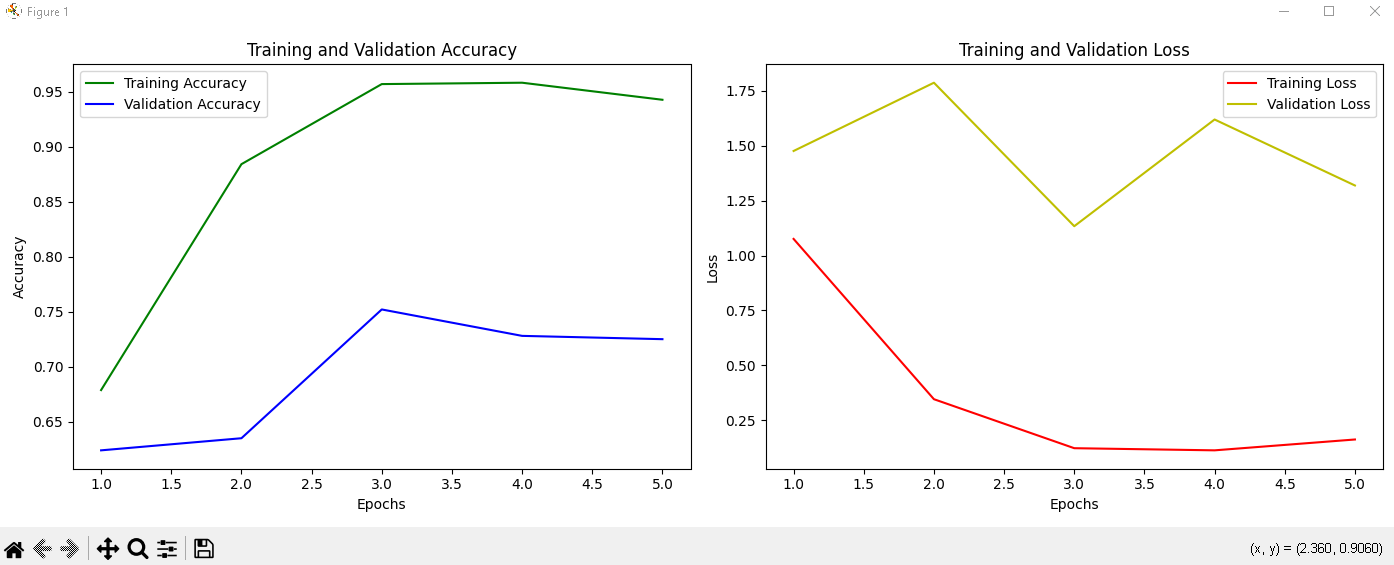
Training is faster, since when we increase the learning rate -the gradient is reducing faster and

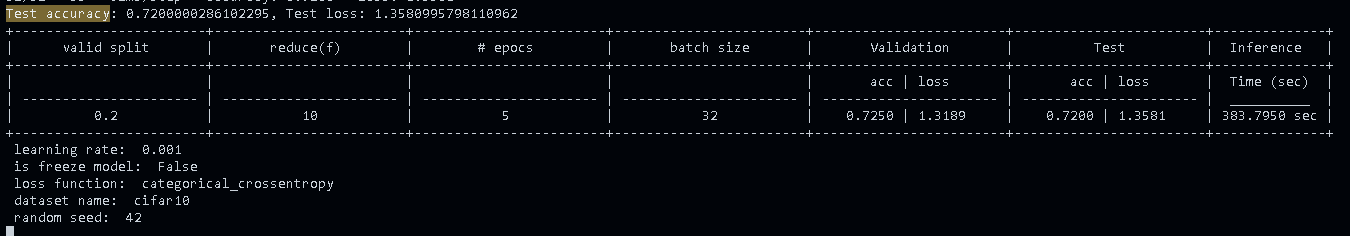
The steps are overcoming the gradient descent point of learning too soon.   
afterwards the learning is not happening , since we overcame the point of Convergence to soon.

since the learning rate is bigger the accuracy gets worse. The tuning is done with larger legs – steps missing the Accuracy reflected in the smallest point in gradient descent calculation.

C. Add a dropout layer to the DNN, right after global average pooling. Hint: <https://keras.io/api/layers/regularization_layers/dropout/>







Over here the

D. Add random flip augmentation to the DNN. Hint:

<https://keras.io/api/layers/preprocessing_layers/image_augmentation/random_flip/>

over here I managed to configure to execute with GPU , and got a bit different graph on same code , at courter of the time , and alos different stats :

python main.py

2024-07-14 01:42:50.175584: I tensorflow/c/logging.cc:34] Successfully opened dynamic library D:\projects\AI\deep-learning-class\project2\P3\_12\_4-1\_execution\.venv310\lib\site-packages\tensorflow-plugins/directml/directml.d6f03b303ac3c4f2eeb8ca631688c9757b361310.dll

2024-07-14 01:42:50.176778: I tensorflow/c/logging.cc:34] Successfully opened dynamic library dxgi.dll

2024-07-14 01:42:50.181700: I tensorflow/c/logging.cc:34] Successfully opened dynamic library d3d12.dll

2024-07-14 01:42:50.627043: I tensorflow/c/logging.cc:34] **DirectML device enumeration: found 2 compatible adapters.**

2024-07-14 01:42:52.459732: I tensorflow/core/platform/cpu\_feature\_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2

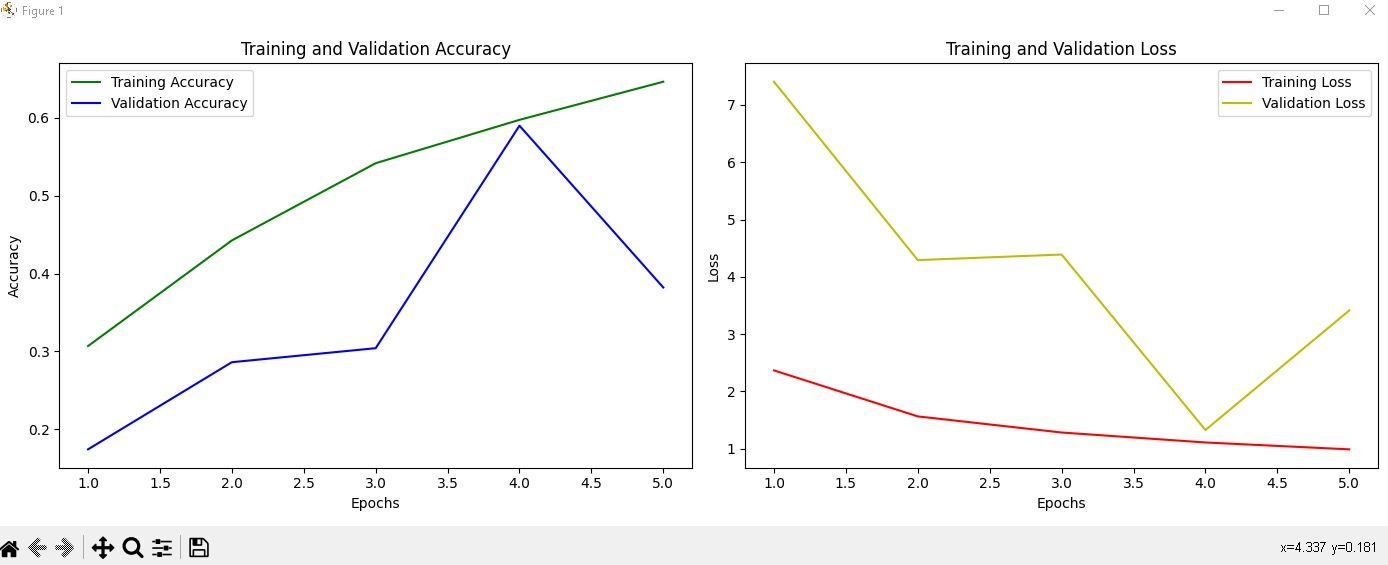
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2024-07-14 01:42:52.461261: I tensorflow/c/logging.cc:34] DirectML**: creating device on adapter 0 (NVIDIA Quadro T1000)**

2024-07-14 01:42:52.604027: I tensorflow/c/logging.cc:34] Successfully opened dynamic library Kernel32.dll

2024-07-14 01:42:52.605137: I tensorflow/c/logging.cc:34] DirectML: creating device on adapter 1 (Intel(R) UHD Graphics 630)

2024-07-14 01:42:52.671389: I tensorflow/core/common\_runtime/pluggable\_device/pluggable\_device\_factory.cc:306] Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not have been built with NUMA support.

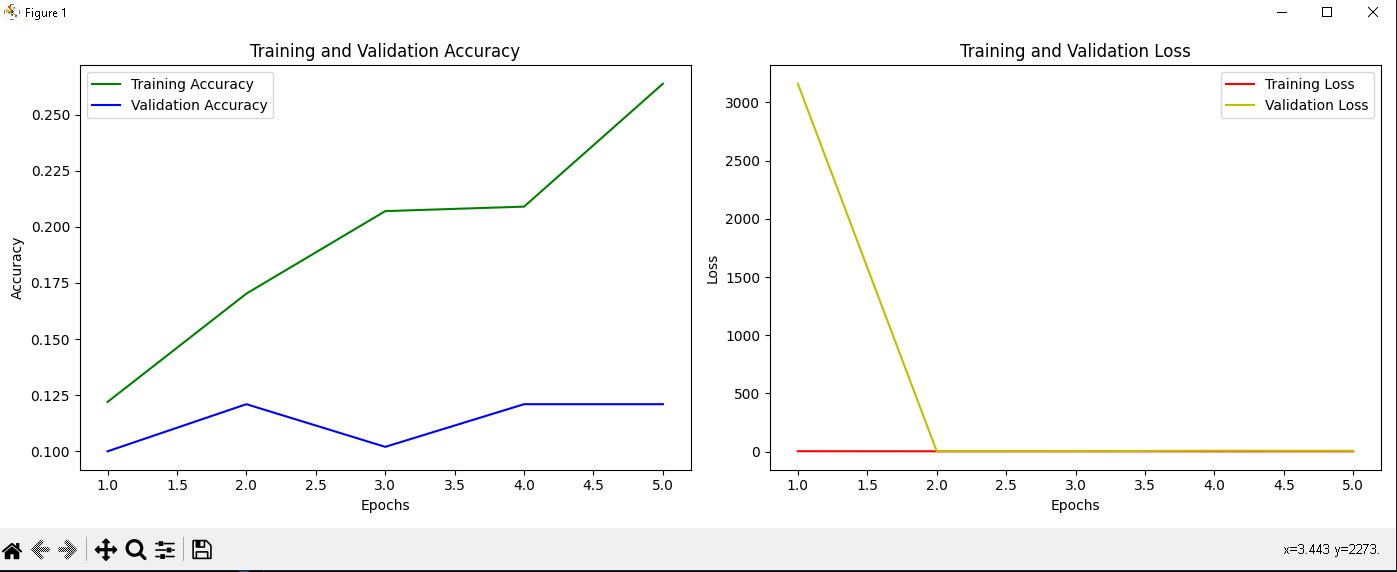




E. Change MobileNet to ResNet18 and retrain for 5 epochs. Calculate the average latency of MobileNet vs ResNet18 using the test set. Use (<https://github.com/qubvel/classification_models>) to get ResNet18. Use “pip install” and update “requirements.txt”. Add a parameter to *create\_model* function for easier switching between the models.

Hints:

* (<https://docs.python.org/3/library/timeit.html> - execution time)



Test accuracy: 0.0950000062584877, Test loss: 6.292205810546875

+------------------------+------------------------+--------------+

| Validation | Test | Inference |

+------------------------+------------------------+--------------+

| acc | loss | acc | loss | Time (sec) |

| ---------------------- | ---------------------- | \_\_\_\_\_\_\_\_\_\_ |

| 0.1210 | 6.3074 | 0.0950 | 6.2922 | 379.3545 sec |

+------------------------+------------------------+--------------+

validation split : 0.2

reduce factor : 10

number of epocs : 5

batch size : 32

learning rate : 0.01

is freeze model : False

loss function : categorical\_crossentropy

dataset name : cifar10

random seed : 42

**Advanced Section - Extra Points**

In this section, we will build from scratch a full computer vision solution, including dataset collection,tagging, training and inference. Let’s build a system that can count Israeli coins. The input to the system would be an image, which is a photo of several coins and the output is the total amount in NIS.

Example:

|  |  |
| --- | --- |
|  | Result : **17 NIS** |

For simplicity's sake, assume that there are no overlaps between the coins and the angle of view is reasonable.

We will focus on the following coins:

|  |  |
| --- | --- |
| **Coin** | **Class Name** |
| 1 Shekel | One |
| 2 Shekels (Shnekel) | Two |
| 5 Shekels | Five |
| 10 Shekels | Ten |

1. Collect using your smartphone a dataset of at least 50 images with different varieties of coins, backgrounds, angles, illuminations. Split the dataset into train and test.
2. Tag the coins bounding boxes and classes using the class names given above. You can use Vertex.ai [datasets tagging tool](https://console.cloud.google.com/vertex-ai/datasets), or any other object detection labeling tool .
   1. You can share the tagged dataset with your friends to increase the amount of images.
3. Train an object detector that returns one of the four classes, using your own dataset
   1. Try fine-tuning an existing object detection algorithm. You can use huggingface
4. Write a code that calculates the total amount of money in an image
5. Write a metric that calculates the accuracy by comparing the normalized absolute difference between the amount of money in an image and the predicted amount.
   1. For example:
      1. Predicted - 20 NIS
      2. Ground Truth - 25 NIS
      3. Accuracy per image = , which is 80% accuracy
   2. Write a metric that calculates the average accuracy over a dataset of N images.
6. What is your accuracy on the test set?

**Submission:**

Provide the results, including the plots and your explanations in a Google Doc. Use the attached template. Also provide your final Python code.

**Citations**:

* Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.
* Deep Residual Learning for Image Recognition, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015
* MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam