Transfer Learning

Part 2

Correct answer is within the top five highest-scoring categories predicted by the model.

Accuracy at 1

Accuracy at 5

Total number of trainable parameters in the model

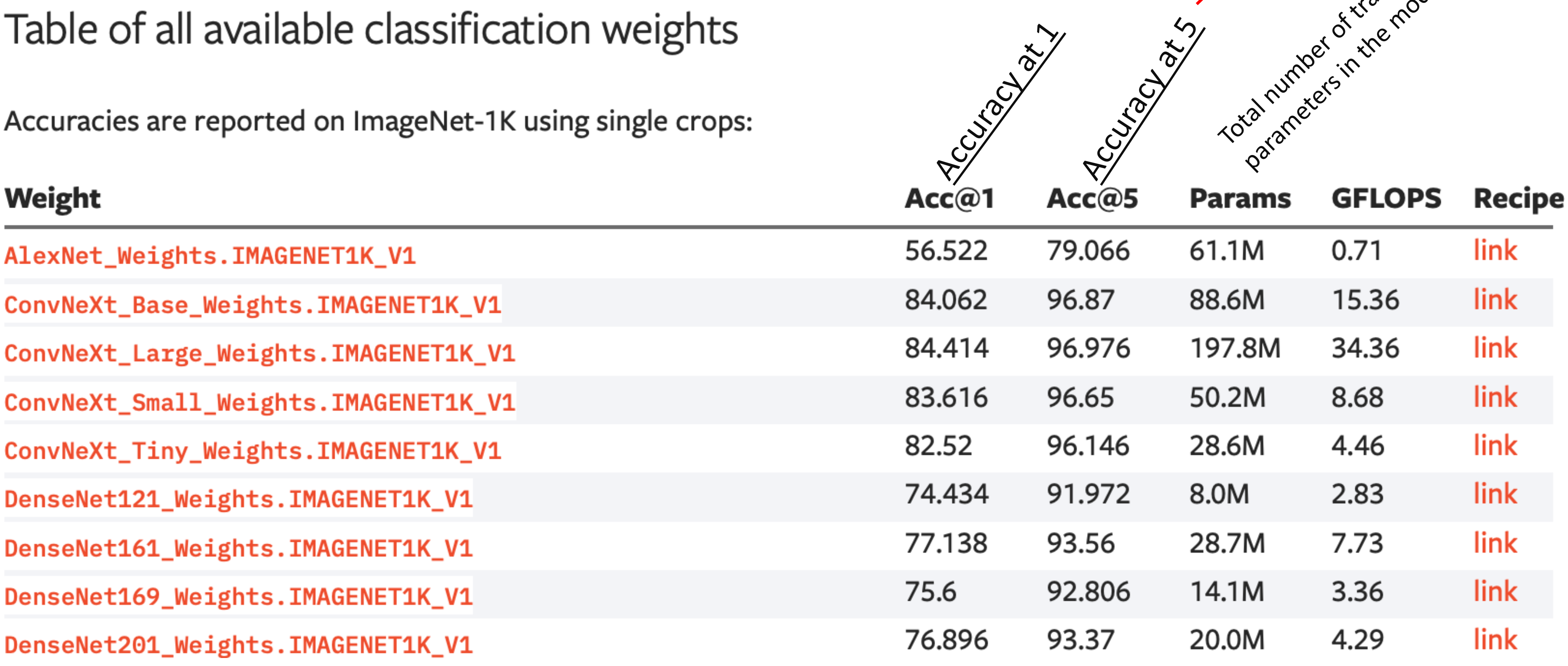


Table of all available classification weights

Accuracies are reported on ImageNet-1K using single crops:

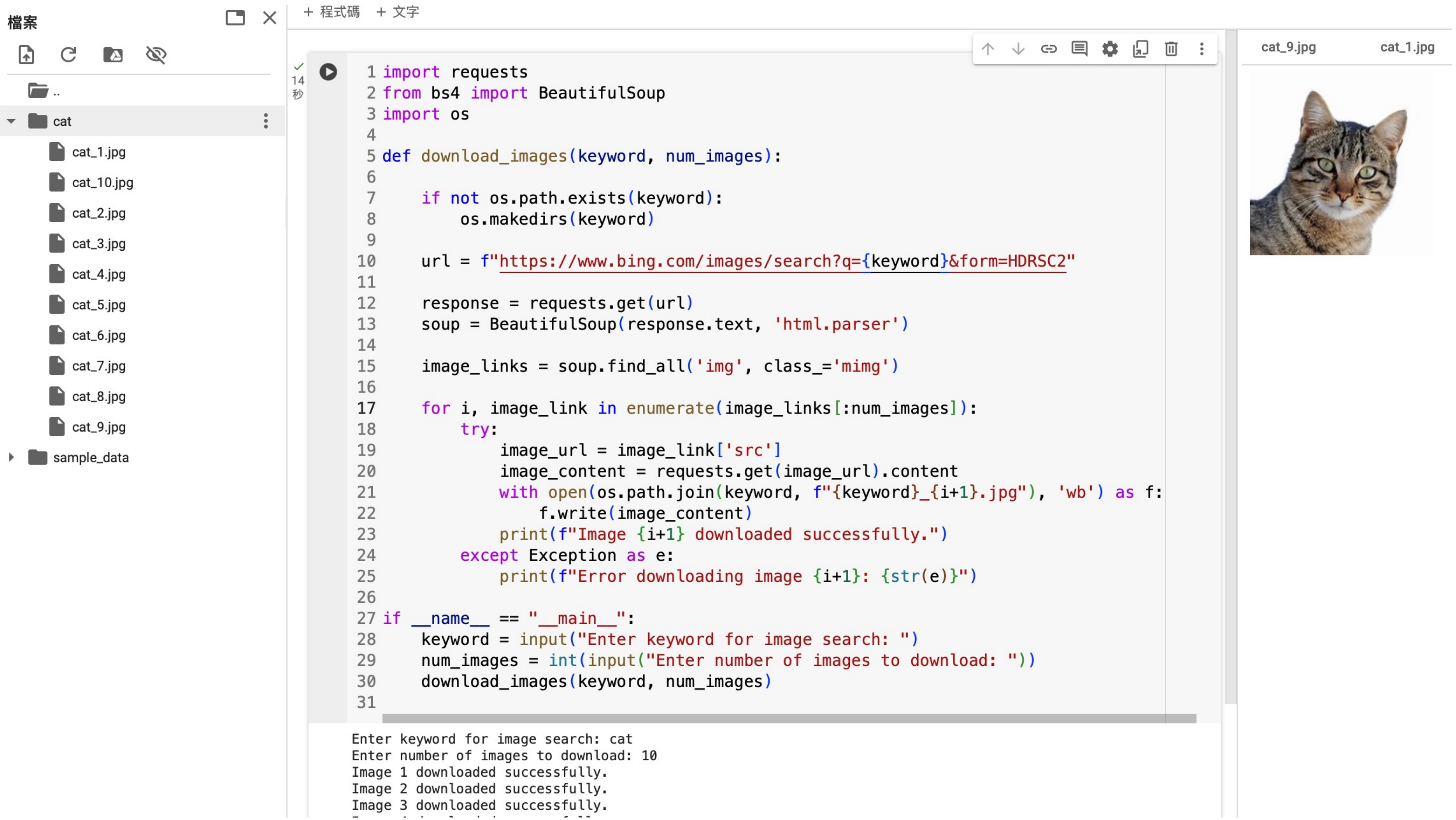
Weight	Acc@1	Acc@5	Params	GFLOPS	Recipe
AlexNet_Weights.IMAGENET1K_V1	56.522	79.066	61.1M	0.71	link
ConvNeXt_Base_Weights.IMAGENET1K_V1	84.062	96.87	88.6M	15.36	link
ConvNeXt_Large_Weights.IMAGENET1K_V1	84.414	96.976	197.8M	34.36	link
ConvNeXt_Small_Weights.IMAGENET1K_V1	83.616	96.65	50.2M	8.68	link
ConvNeXt_Tiny_Weights.IMAGENET1K_V1	82.52	96.146	28.6M	4.46	link
DenseNet121_Weights.IMAGENET1K_V1	74.434	91.972	8.0M	2.83	link
DenseNet161_Weights.IMAGENET1K_V1	77.138	93.56	28.7M	7.73	link
DenseNet169_Weights.IMAGENET1K_V1	75.6	92.806	14.1M	3.36	link
DenseNet201_Weights.IMAGENET1K_V1	76.896	93.37	20.0M	4.29	link

GFLOPS: Number of floating-point operations required for the model to perform one forward inference.

Recipe: The specific training process or settings used to achieve these performance metrics

https://pytorch.org/vision/stable/models.html

Image downloader



```python
import requests
from bs4 import BeautifulSoup
import os

def download_images(keyword, num_images):

    if not os.path.exists(keyword):
        os.makedirs(keyword)

    url = f"https://www.bing.com/images/search?q={keyword}&form=HDRSC2"

    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    image_links = soup.find_all('img', class_='mimg')

    for i, image_link in enumerate(image_links[:num_images]):
        try:
            image_url = image_link['src']
            image_content = requests.get(image_url).content
            with open(os.path.join(keyword, f"{keyword}_{i+1}.jpg"), 'wb') as f:
                f.write(image_content)
            print(f"Image {i+1} downloaded successfully.")
        except Exception as e:
            print(f"Error downloading image {i+1}: {str(e)}")

if __name__ == "__main__":
    keyword = input("Enter keyword for image search: ")
    num_images = int(input("Enter number of images to download: "))
    download_images(keyword, num_images)

```

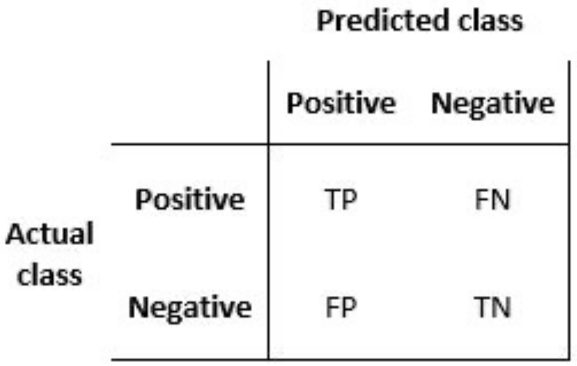
```
Enter keyword for image search: cat
Enter number of images to download: 10
Image 1 downloaded successfully.
Image 2 downloaded successfully.
Image 3 downloaded successfully.
```

Confusion Matrix (誤差矩陣、混淆矩陣)

- Confusion Matrix (error matrix), is a tool widely used in machine learning to **evaluate the performance of classification models**.

- It presents the relationship between the model's predictions and the actual labels in a matrix format.

- The confusion matrix is typically divided into four quadrants:
  - True Positive (TP)
  - True Negative (TN)
  - False Positive (FP)
  - False Negative (FN).



		Predicted class	
		Positive	Negative
Actual class	Positive	TP	FN
	Negative	FP	TN

https://en.m.wikipedia.org/wiki/File:Binary_confusion_matrix.jpg

Confusion Matrix (誤差矩陣、混淆矩陣)

Predicted class

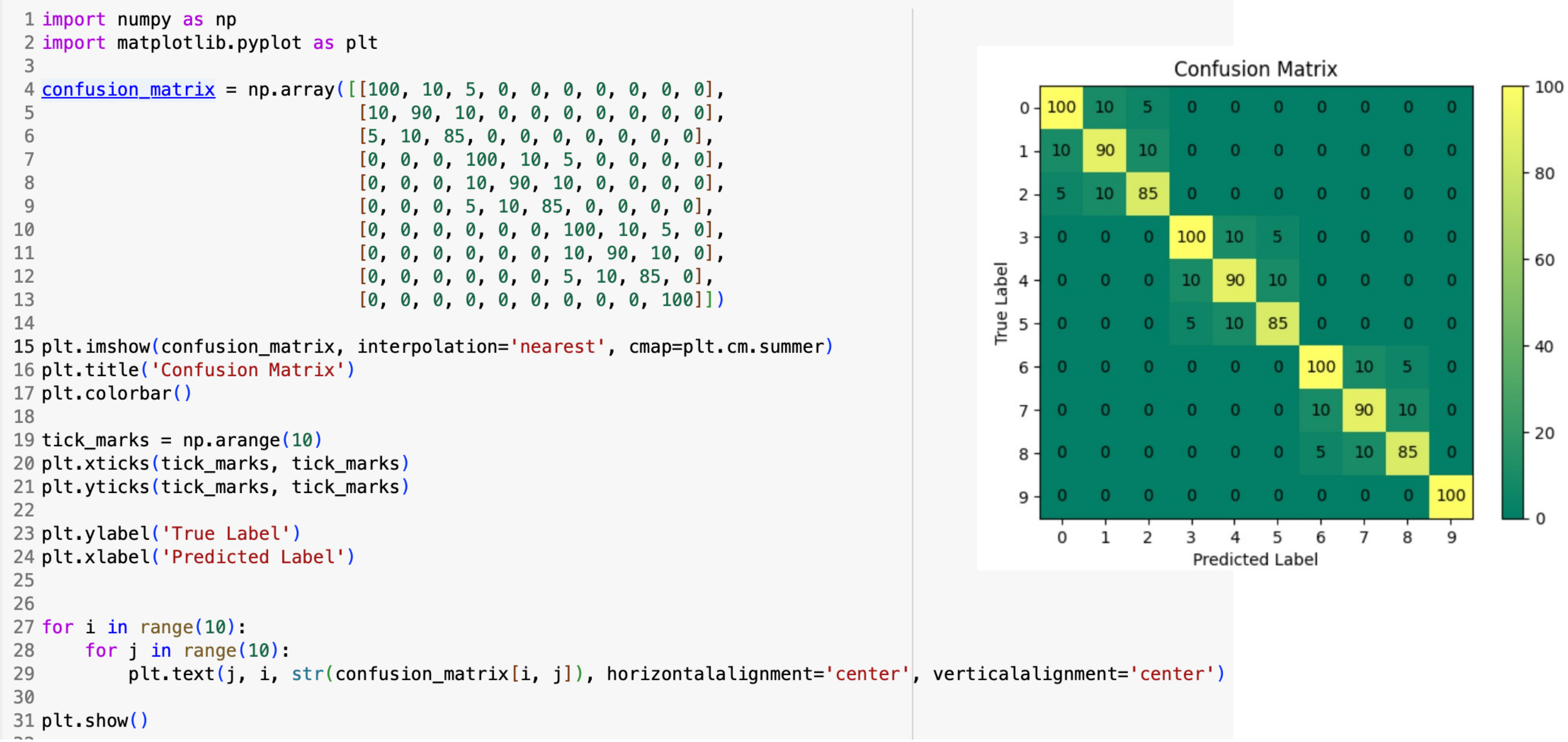
Actual class

	cat	dog	rat	tiger	
cat	98	1	32	10	→ 141
dog	3	82	1	4	→ 90
rat	5	5	60	2	→ 72
tiger	10	6	4	86	→ 106

Predicted class

	cat	dog	rat	tiger
cat	69.5%	0.7%	22.7%	7.1%
dog	3.3%	91.1%	1.1%	4.4%
rat	6.9%	6.9%	83.3%	2.8%
tiger	9.4%	5.7%	3.8%	81.1%

Confusion Matrix (誤差矩陣、混淆矩陣)



```python
import numpy as np
import matplotlib.pyplot as plt

confusion_matrix = np.array([[100, 10, 5, 0, 0, 0, 0, 0, 0, 0],
                             [10, 90, 10, 0, 0, 0, 0, 0, 0, 0],
                             [5, 10, 85, 0, 0, 0, 0, 0, 0, 0],
                             [0, 0, 0, 100, 10, 5, 0, 0, 0, 0],
                             [0, 0, 0, 10, 90, 10, 0, 0, 0, 0],
                             [0, 0, 0, 5, 10, 85, 0, 0, 0, 0],
                             [0, 0, 0, 0, 0, 0, 100, 10, 5, 0],
                             [0, 0, 0, 0, 0, 0, 10, 90, 10, 0],
                             [0, 0, 0, 0, 0, 0, 5, 10, 85, 0],
                             [0, 0, 0, 0, 0, 0, 0, 0, 0, 100]])

plt.imshow(confusion_matrix, interpolation='nearest', cmap=plt.cm.summer)
plt.title('Confusion Matrix')
plt.colorbar()

tick_marks = np.arange(10)
plt.xticks(tick_marks, tick_marks)
plt.yticks(tick_marks, tick_marks)

plt.ylabel('True Label')
plt.xlabel('Predicted Label')


for i in range(10):
    for j in range(10):
        plt.text(j, i, str(confusion_matrix[i, j]), horizontalalignment='center', verticalalignment='center')

plt.show()
```

One-vs-all matrix

		Predicted class	
		Positive	Negative
Actual class	Positive	TP	FN
	Negative	FP	TN

Predicted class

Actual class		cat	dog	rat	tiger
	cat	98	1	32	10
	dog	3	82	1	4
	rat	5	5	60	2
	tiger	10	6	4	86

Cat

Predicted class

Actual class		positive	negative
	positive	98	43
	negative	18	250

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$\textit{F1 score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Exercise:

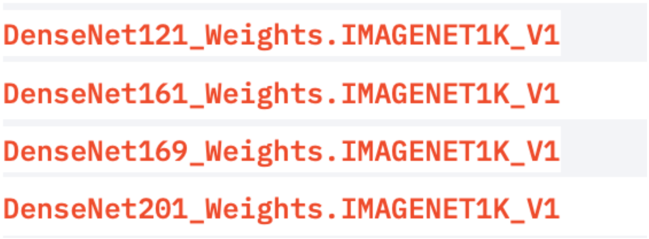
Continuing from the previous assignment, collect at least 200 images for each category:

1. Design a program to compare the prediction results.

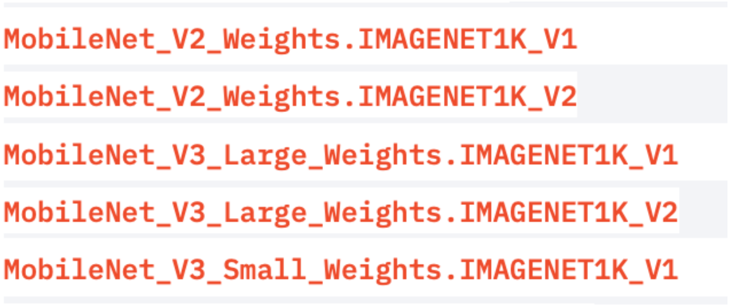
2. Select one model and use different weights (at least 4 weights) to compare the prediction results.

3. Choose 6 models (one weight for each model), and introduce your chosen models with graphics/tables and text, comparing the prediction results.

PS. At least 8 pages of A4 paper, font size 12, Arial font, line spacing 1.5.



```
DenseNet121_Weights.IMAGENET1K_V1
DenseNet161_Weights.IMAGENET1K_V1
DenseNet169_Weights.IMAGENET1K_V1
DenseNet201_Weights.IMAGENET1K_V1
```



```
MobileNet_V2_Weights.IMAGENET1K_V1
MobileNet_V2_Weights.IMAGENET1K_V2
MobileNet_V3_Large_Weights.IMAGENET1K_V1
MobileNet_V3_Large_Weights.IMAGENET1K_V2
MobileNet_V3_Small_Weights.IMAGENET1K_V1
```

Exercise:

Submission requirements:

1. source code(s)

2. PDF document

3. Upload to e-learning before 5/3 14:10