

УДК 004.85

Bukhanovskiy Volodymyr, Ryabova Nataliya²

¹ PhD student, Kharkiv National University of Radio Electronics

² professor of AI Department, Kharkiv National University of Radio Electronics

TRANSFER LEARNING METHODS IN COMPUTER VISION

Transfer learning is a machine learning technique that uses a previously trained model to solve a new, but related task. This approach significantly reduces the amount of resources required to solve the problem compared to training the model from scratch.

Transfer learning is effective because convolutional neural networks learn the object features gradually, increasing complexity with each layer. In the first few layers, the network operates on low-level features (lines, curves, edges, etc.) that are common to any dataset or task. In the middle layers, the network operates on mid-level features (shapes, corners, circles), that usually differ if the datasets or tasks are very different (e.g. face recognition and machine recognition). In the highest layers, the network operates on high-level features that are extremely specific to the chosen task. Depending on the similarity of the task, we can use different amounts of knowledge that is already in the pre-trained model.

There are three transfer learning methods: using pretrained model as a classifier, using pretrained model as a feature extractor, fine-tuning [1]. Each of these methods is applicable under different conditions.

To choose a suitable method, we need to pay attention to the following parameters of the task: the size of the available data source for training (absent, small, large), the similarity of the domains of the pre-trained model and the task at hand (small, large). Based on these parameters, you can consider five main scenarios:

- if data sources are absent and the similarity is high, you can use pre-trained model as a classifier;
- if the size of the data source is small and the similarity is high, it is worth using pre-trained model as a feature extractor;
- if the size of the data source is large and the similarity is high, it is worth using pre-trained model as a feature extractor, or fine-tuning with freezing 60-80% of the pre-trained model;
- if the size of the data source is small and the similarity is low, it is worth using fine-tuning and freezing 30-50% of the pre-trained model;
- if the size of the data source is large and the similarity is small, it is worth using fine-tuning and training the entire pre-trained model.

The applied task was object detection, namely solid waste. It is relevant due to the global increase in the amount of waste and its negative impact on the environment [2].

Our previous work focused on researching data augmentation methods and training the entire model [3]. The proposed model can be used in various scenarios: monitoring pollution on city streets or roadsides, integration into waste recycling lines, and detection of unauthorized landfills.

The task can be solved in more optimal way using transfer learning technique. For this, we can use the same YOLOv8 model and the Trash annotation in context (TACO) data source [4]. Instead of using a pre-trained model on the COCO dataset, it is worth paying attention to the model trained on the Open Images V7 dataset due to the significantly larger amount of data (330k images vs. 9m images, respectively) and classes (80 classes vs. 600 classes, respectively) [5]. A larger number of classes allows us to match classes from the source data set for the pre-trained model and the source data set for transfer learning. As a result, we get the following list of classes: bottle, can (matches “can”, “tin can” in source dataset), cup (matches “cup”, “coffee cup”), plastic bag and wrapper (matches “plastic bag”), container (matches “plastic container”, “container”).

Considering the availability of the data source and the similarity of the domains, we can choose the second of the scenarios listed above.

In the previous work, YOLOv8s was chosen out of 5 available model sizes. It is the second smallest option among the proposed ones. The model architecture consists of 23 elements, has a total of 225 layers, 11140244 parameters, 11140228 gradients (Figure 1).

Instead of training the entire model, we can use the pre-trained model as a feature extractor and train only the classifier. To do this, we freeze all other layers and leave only the last one, which consists of 2120692 parameters. The number of parameters to train has dropped by about 5 times, which will significantly speed up training. It means a significant reduction in training epochs and training time.

If additional resources are present, we can consider a third scenario with fine-tuning and freezing 80% of the pre-trained model. With this approach, we rely on the information already present in the model about the low-level (lines, edges, curves, and small parts of objects) and medium-level features. The number of parameters will increase by about half. Potentially, the model will start to work better with the chosen classes.

	from	n	params	module	arguments
0	-1	1	928	ultralytics.nn.modules.conv.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.conv.Conv	[32, 64, 3, 2]
2	-1	1	29056	ultralytics.nn.modules.block.C2f	[64, 64, 1, True]
3	-1	1	73984	ultralytics.nn.modules.conv.Conv	[64, 128, 3, 2]
4	-1	2	197632	ultralytics.nn.modules.block.C2f	[128, 128, 2, True]
5	-1	1	295424	ultralytics.nn.modules.conv.Conv	[128, 256, 3, 2]
6	-1	2	788480	ultralytics.nn.modules.block.C2f	[256, 256, 2, True]
7	-1	1	1180672	ultralytics.nn.modules.conv.Conv	[256, 512, 3, 2]
8	-1	1	1838080	ultralytics.nn.modules.block.C2f	[512, 512, 1, True]
9	-1	1	856896	ultralytics.nn.modules.block.SPPF	[512, 512, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.conv.Concat	[1]
12	-1	1	591360	ultralytics.nn.modules.block.C2f	[768, 256, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
14	[-1, 4]	1	0	ultralytics.nn.modules.conv.Concat	[1]
15	-1	1	148224	ultralytics.nn.modules.block.C2f	[384, 128, 1]
16	-1	1	147712	ultralytics.nn.modules.conv.Conv	[128, 128, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.conv.Concat	[1]
18	-1	1	493056	ultralytics.nn.modules.block.C2f	[384, 256, 1]
19	-1	1	590336	ultralytics.nn.modules.conv.Conv	[256, 256, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.conv.Concat	[1]
21	-1	1	1969152	ultralytics.nn.modules.block.C2f	[768, 512, 1]
22	[15, 18, 21]	1	2120692	ultralytics.nn.modules.head.Detect	[12, [128, 256, 512]]

Model summary: 225 layers, 11140244 parameters, 11140228 gradients, 28.7 GFLOPs

Figure 1 – YOLOv8s architecture

As a result of applying transfer learning, an effectively working model can be obtained using significantly fewer resources.

REFERENCES

1. Elgendy M. Deep learning for vision systems. Manning Publications Co. LLC, 2020.
2. Solid waste management. UNEP - UN Environment Programme. URL: <https://www.unep.org/explore-topics/resource-efficiency/what-we-do/cities/solid-waste-management> (date of access: 08.11.2024).
3. Бухановський В. О. Дослідження та застосування методів комп'ютерного зору для вирішення задач ідентифікації та класифікації твердих відходів : кваліфікаційна робота здобувача вищої освіти, пояснювальна записка. Харків, 2024. 60 р. URL: <https://catalogue.nure.ua/document=266177> (date of access: 08.11.2024).
4. Бухановський В. О. Дослідження та застосування методів computer vision для розпізнавання та класифікації забруднень. Радіоелектроніка та молодь у XXI столітті. Т. 6 : Конференція "Інформаційні інтелектуальні системи" : матеріали 28-го Міжнар. молодіж. форуму, Харків, 18 April 2024. Харків, 2024. Р. 958. URL: <https://doi.org/10.30837/IYF.IIS.2024.061> (date of access: 08.11.2024).
5. Ultralytics. YOLOv8. Ultralytics YOLO Docs. URL: <https://docs.ultralytics.com/models/yolov8/#performance-metrics> (date of access: 08.11.2024).