Garbage Classification

- Sara Dellacasa - 5409694
- Giulia Franceschina - 5407529

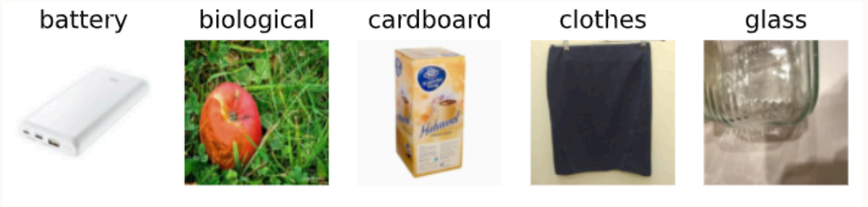
Dataset Overview: Goals & key Characteristics

Our goals

- Train AI models to classify garbage in order to promote automated waste management
- facilitate the spread of awareness regarding sustainability and recycling.

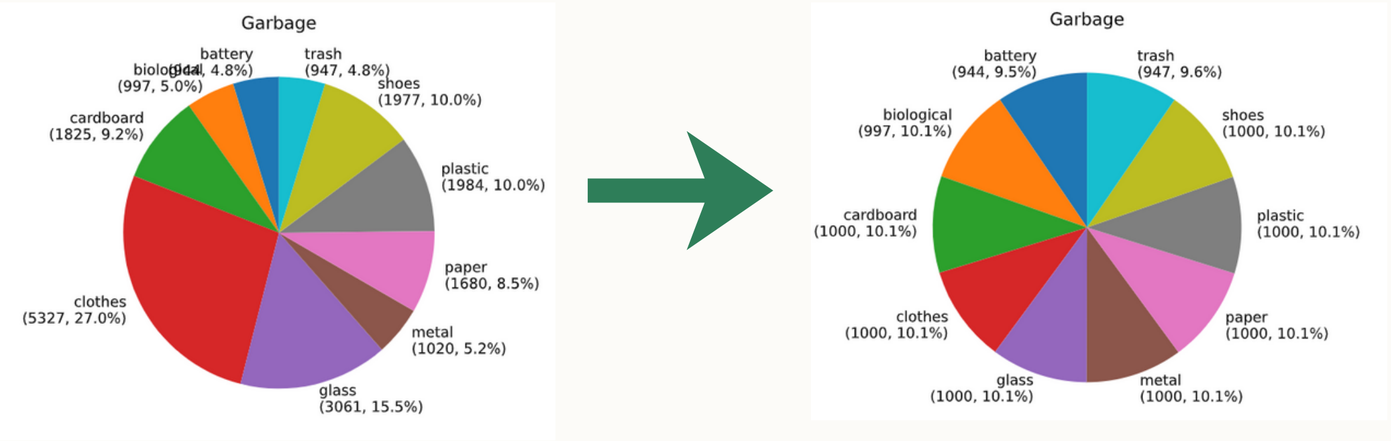
Key Characteristics of our dataset

- Content: 19.762 high-quality images;
- Classes: 10 common household waste classes;
- Images: color model RGB, different size and different number per class.



Data preparation

1. Class-wise image subsampling for uniform class distributions (maximum 1,000 Images per class)



2. Basic image preprocessing: resizing and normalization to ensure that all input images have a consistent size and intensity distribution

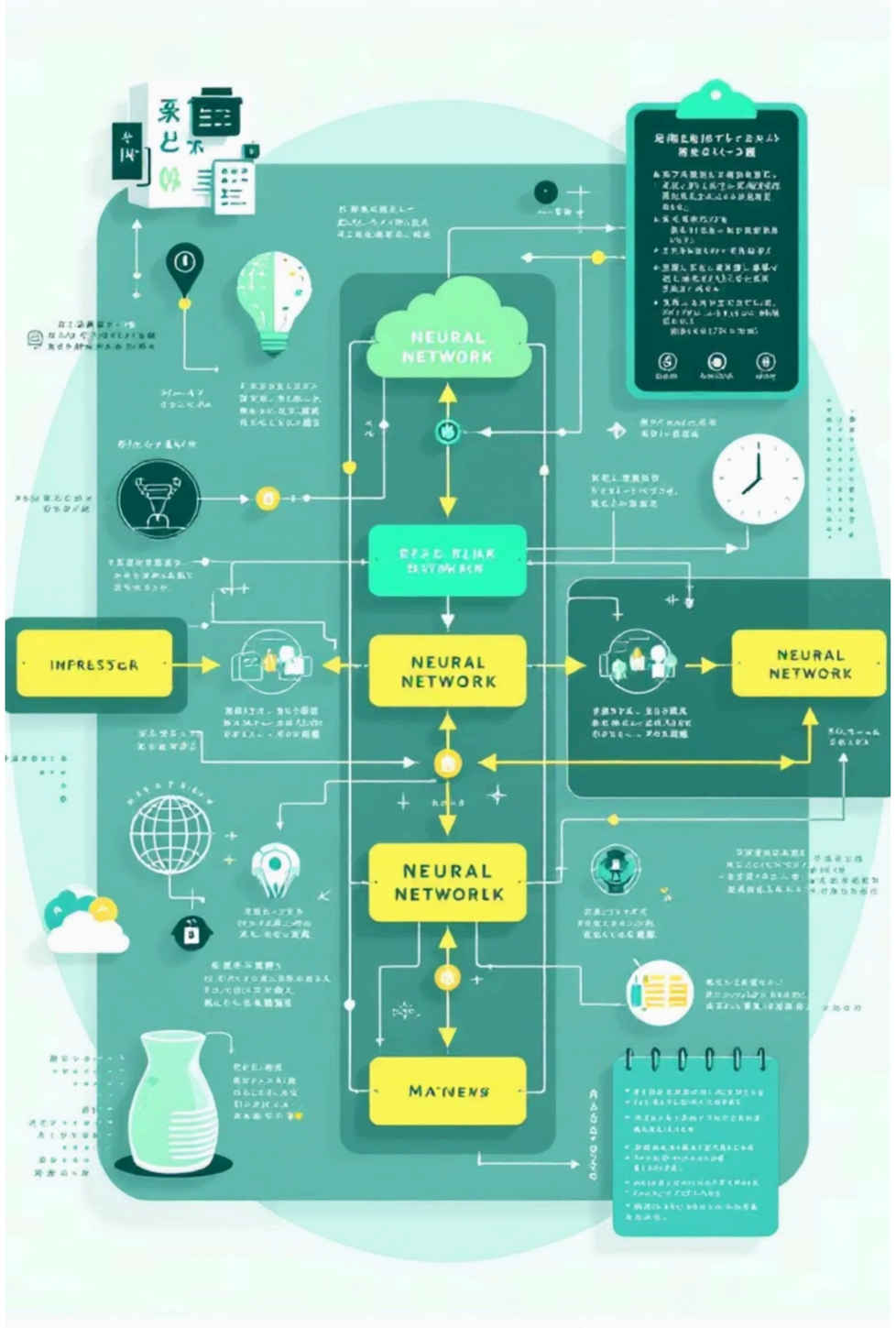
```
mean = [0.5, 0.5, 0.5]
std = [0.5, 0.5, 0.5]
data_augmentation = transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),
        transforms.Normalize(mean, std),])
```

3. Veryfing shape after preprocessing and splitting dataset into training and testing sets (80%–20%)

```
torch.Size([3, 224, 224])
```

```
Train: 7910
Test: 1978
```

→ Total: 9,888 images



Model 1: AlexNet Journey

1

1A: Baseline

Plain SGD, 10 epochs. Test accuracy: 0.372. Slow convergence, basic optimization.

2

1B: Adam Optimizer

Usage of SGD + Momentum (=0.9), then switched to Adam (lr=0.001). Test accuracy: 0.605. Loss reduced 55%, major improvement.

3

1C: Deeper Architecture

Added FC layer, 15 epochs. Test accuracy: 0.602. Overfitting emerged, no generalization gain.

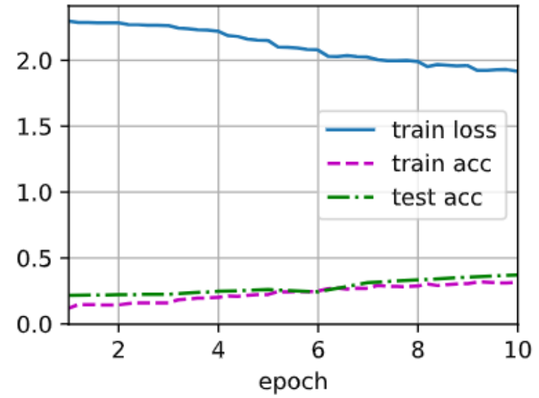
4

1D: Final Model

Controlled augmentation, 30 epochs. Test accuracy: 0.748. Best performance achieved.

AlexNet: 1A baseline model

```
loss 1.920, train acc 0.312, test acc 0.372
793.0 examples/sec on cuda:0
```



The baseline model utilizes a sequential AlexNet-style CNN:

- 5 convolutional layers (ReLu activation and Max Pooling)
- 3-layer fully connected block with dropout (p=0.5) between hidden FC layers.

Initial training using Stochastic Gradient Descent (SGD) with fixed lr=0.01 for 10 epochs to establish a controlled performance reference.

Baseline for subsequent experimentation with advanced optimization strategies aimed at improving convergence and overall performance.

AlexNet: 1B finding best optimizer

We first trained the model for 10 epochs using **SGD with momentum (0.9)**, obtaining an improvement over the baseline. As a second trial, we trained the model with same number of epochs and **Adam** (Adaptive Moment Estimation) optimizer using lr = 0.001, achieving better performance than both the previous setups.

	Baseline SGD	SGD + Momentum	Adam
Final loss	1,920	1,270	0,866
Train acc	0,312	0,565	0,709
Test acc	0,372	0,596	0,605

AlexNet 1C - Architecture Refinement

```
from d2l import torch as d2l
import torch
from torch import nn
net = nn.Sequential(
    nn.Conv2d(3, 96, kernel_size=11, stride=4, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(96, 256, kernel_size=5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Conv2d(256, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Flatten(),
    nn.Linear(6400, 4096),
    nn.ReLU(),
    nn.Dropout(p=0.3),
    nn.Linear(4096, 4096),
    nn.ReLU(),
    nn.Linear(4096,1024),
    nn.ReLU(),
    nn.Linear(1024, 10))
```

Selected the best optimizer, we try modifying the AlexNet architecture:

- add an intermediate FC layer 4096 → 1024;
- lower dropout to avoid overly constraining the optimization;
- number of epochs 15.

Results: Training improved, reduction in training loss, test accuracy stagnant.

Growing disparity between train and test accuracy → overfitting

In the next step we will add noise to the train set. Given that the current architecture did not negatively impact generalization , it will be retained.

AlexNet 1D - Data Augmentation

We apply data augmentation to improve generalization.

1st model aggressive augmentation setup to reduce overfitting

Results:

- loss = 1.575;
- train acc = 0.458;
- test acc = 0.488 (test slightly higher than train).

The train test gap became smaller because training got worse, not because generalization improved. The aggressive augmentation made many training images look too altered and harder to interpret. As a consequence, the model struggled to learn properly and performance dropped.

```
train_transform_v2 = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomResizedCrop(224, scale=(0.5, 1.0)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(20),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.1),
    transforms.ToTensor(),
    transforms.Lambda(lambda t: t + 0.05 * torch.randn_like(t)),
    transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
    transforms.Normalize(mean, std),])

test_transform_v2 = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean, std),])
```

AlexNet 1D - Final Model and predictions

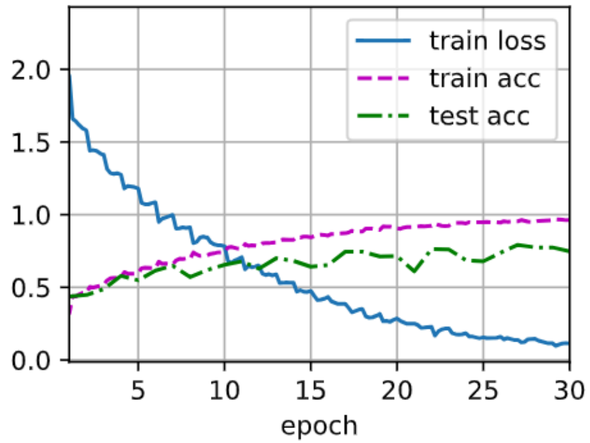
2nd model lighter data augmentation

- random cropping restricted to a narrower scale range (0.75–1.0);
- geometric and color changes are reducted;
- noise and Gaussian blur removed;
- the test set is unchanged for a fair comparison.

Results: lighter data augmentation + 30 training epochs → **best performance**

- Test accuracy improves w.r.t. non-augmented setup (≈ 0.60 → 0.748) and first data augmentation;
- Training curves also show smooth convergence;
- Train–test gap remains, but it stabilizes in later epochs and does not prevent the test result.

```
loss 0.114, train acc 0.963, test acc 0.748
1491.8 examples/sec on cuda:0
```

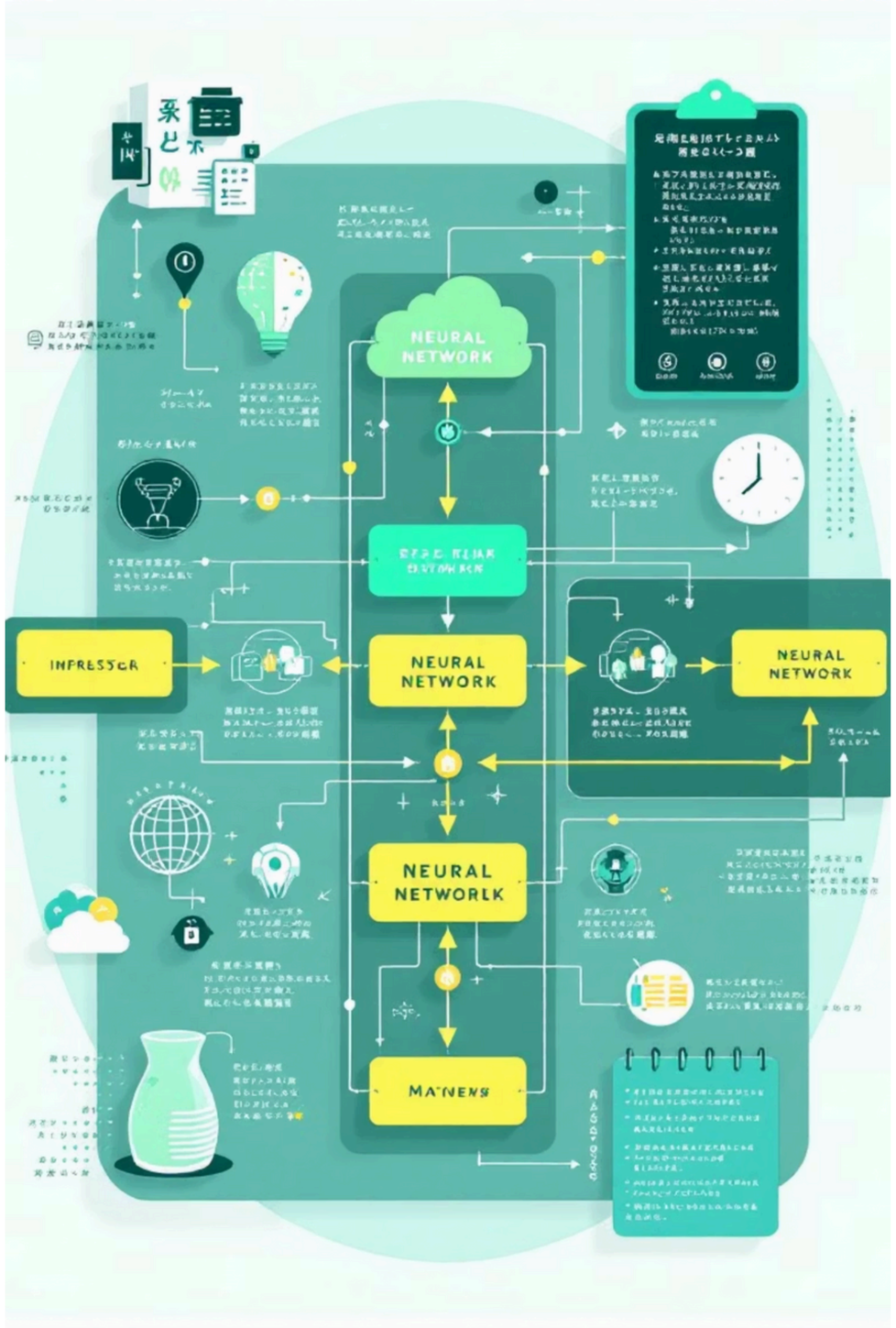


Predictions: The model correctly classifies several samples so it has learned meaningful, class-specific cues.

Errors represented are semantically plausible:

- clothes instead of shoes
- metal for images of batteries

Model relies on material and texture information rather than random guesses, coherent decision-making



Model 2: ResNet18 Journey

1

2A: Baseline

Plain SGD, test accuracy 0.565. Residual connections showed early promise.

2

2B: Optimized

SGD + momentum (0.9), weight decay, StepLR scheduler. Test accuracy: 0.666.

3

2C: Wider Stem

128 channels vs 64. Test accuracy: 0.675. Marginal architectural gain.

4

2D: Final Model

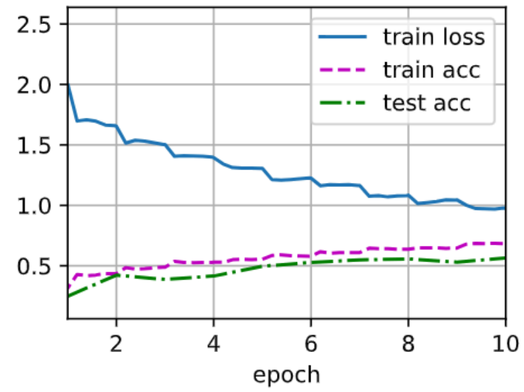
Data augmentation, 30 epochs. Test accuracy: 0.720. Strong generalization.

ResNet18 2A - Baseline model

In the ResNet18 baseline model:

- **7×7 kernel** (stride 2) followed by BatchNorm, ReLU, and max pooling;
- residual blocks;
  → **increasing** the number of channels to learn more features;
  → **downsampling** using stride = 2;
- **features maps are aggregated** with adaptive global average pooling, flattened, and passed to a fully connected layer.

```
loss 0.977, train acc 0.683, test acc 0.565
354.1 examples/sec on cuda:0
```



Results:
Good early performance, with mild overfitting, but using plain SGD is not sufficient for learning fastly.

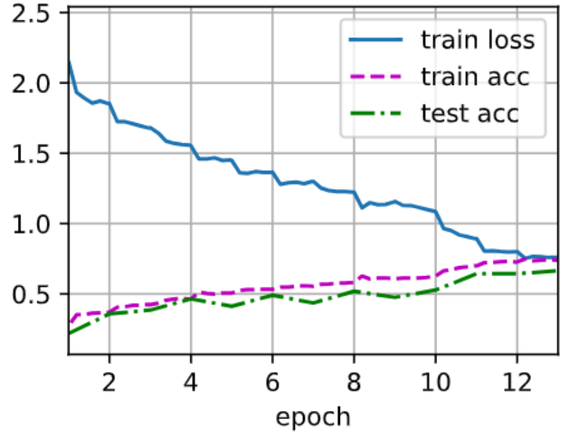
ResNet18 2B - Choosing a different optimizer

This phase focused on exploring various training configurations to establish an optimal optimization strategy for the fixed ResNet18 architecture.

- Stochastic Gradient Descent (**SGD**) with **momentum** 0.9 and a **weight decay** of 0,0001;
- a **StepLR learning rate** scheduler was implemented;
- the **total number of training epochs** was extended, learning rate decay occurred after epoch 10.

Results: significantly improving of the test accuracy (from 0.565 to 0.666), confirming that optimization strategy is critical for realizing the model's capacity.

```
loss 0.758, train acc 0.742, test acc 0.666
347.5 examples/sec on cuda:0
```



ResNet18 2C - Architecture refinements

```python
class ResNet18(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 128,kernel_size=7, stride=2,
                               padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(128)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2,
                                    padding=1)
        self.in_channels = 128
        self.layer1 = self._make_layer(128,  2, stride=1)
        self.layer2 = self._make_layer(128, 2, stride=2)
        self.layer3 = self._make_layer(256, 2, stride=2)
        self.layer4 = self._make_layer(512, 2, stride=2)
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)
```

We keep the ResNet18 residual-block structure unchanged but increase capacity:

The **first convolution is expanded from 3→64 to 3→128**, and all following stages are consequntly adjusted.

The goal is to test whether a wider early backbone learns richer features and improves performance, at the cost of higher complexity and compute.

Results: marginal improvement in test accuracy, with well-aligned training/test curves suggesting stability but still a generalization gap.

This minimal benefit indicates architectural widening alone is insufficient.

ResNet18 2D - Data augmentation and final model

We want to improve generalization using data augmentation on train data and keeping the modified architecture.

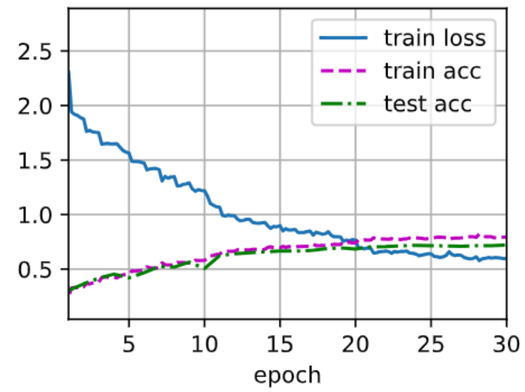
We reuse the same augmentation pipeline that worked best for the AlexNet-based model and trained for 30 epochs.

Results: clear improvement in the generalization performance

- test accuracy increases substantially
- moderate gap between training and test accuracy
- training loss decrease smoothly
- train and test accuracy converge toward a plateau

This suggests that the model has reached a overall good regime under the current state.

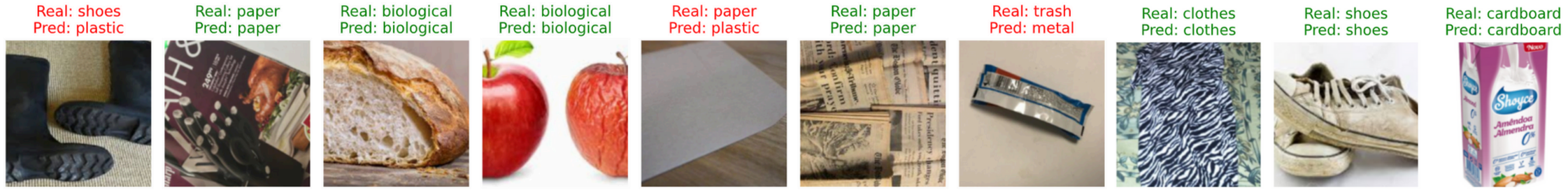
```
loss 0.596, train acc 0.794, test acc 0.720
1498.0 examples/sec on cuda:0
```



Predictions: Error inspection indicates the model has successfully learned meaningful high-level features.
Specific confusion examples include:

- shoes classified as plastic
- paper predicted as plastic
- trash confused with metal

Thus, this behavior validates the ResNet as a reasonably well-trained model.



Conclusions

After finalizing our two final models, we can highlight conclusions from both training journeys: biggest improvements came from **training strategy**, not from simply “adding layers”, in particular, switching the optimizer and introducing data augmentation produced the largest gains.

Changing the optimizer moved the models from clear underfitting to solid performance. Then, **data augmentation** helped reduce the gap between training and test accuracy. This was especially evident in the AlexNet case, where, after the architectural modification, the model developed a large and persistent generalization gap: training accuracy kept improving while test accuracy stagnated.

Overall, this suggests that some architectural changes may have been suboptimal, and that keeping the standard network configuration (while improving the training setup) could likely have achieved comparable results.

Finally, by further **analyzing the predictions** of both final models, we observed that the remaining errors are mostly class confusions driven by visual similarity between categories and some level of dataset noise.
However, the models tend to make mistakes more frequently on particular classes, so maybe a further step could be to investigate the reason why the model has more error in one class than an other and the most common class confusion error.

The next round of **improvements** should therefore focus on those **problematic classes**, investigating whether the issue is:

- option 1: insufficient diversity/representation (the model learns too few features for those classes);
- option 2: overly aggressive augmentation that may have distorted already ambiguous examples and obstacled learning.

Thanks for attention!