Increasing Network Size for Class Incremental Learning

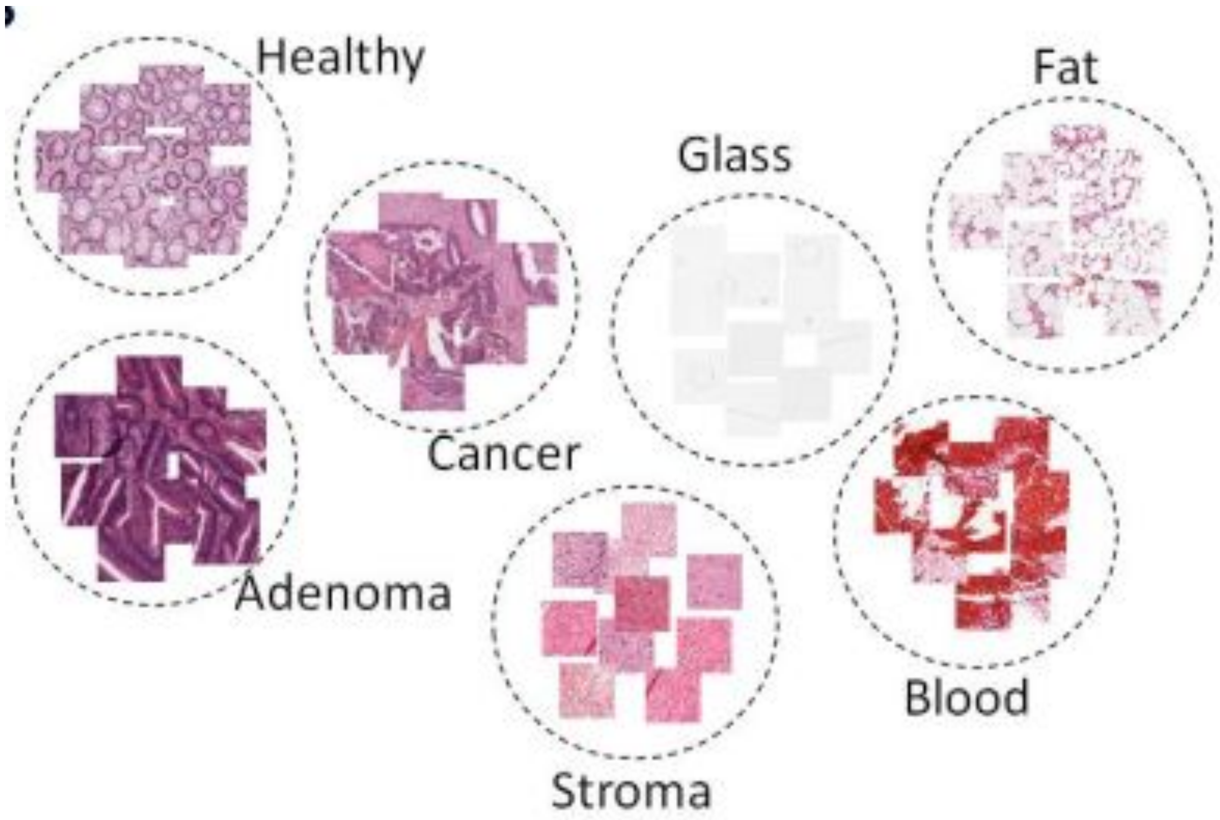
Bioinformatics Project #11

Jana Magdeska S261427
Francesco Dibitonto S265421

Introduction

Task: Learning new classes by increasing the network size

Data: 7 classes of Colorectal Cancer classification



Data introduction

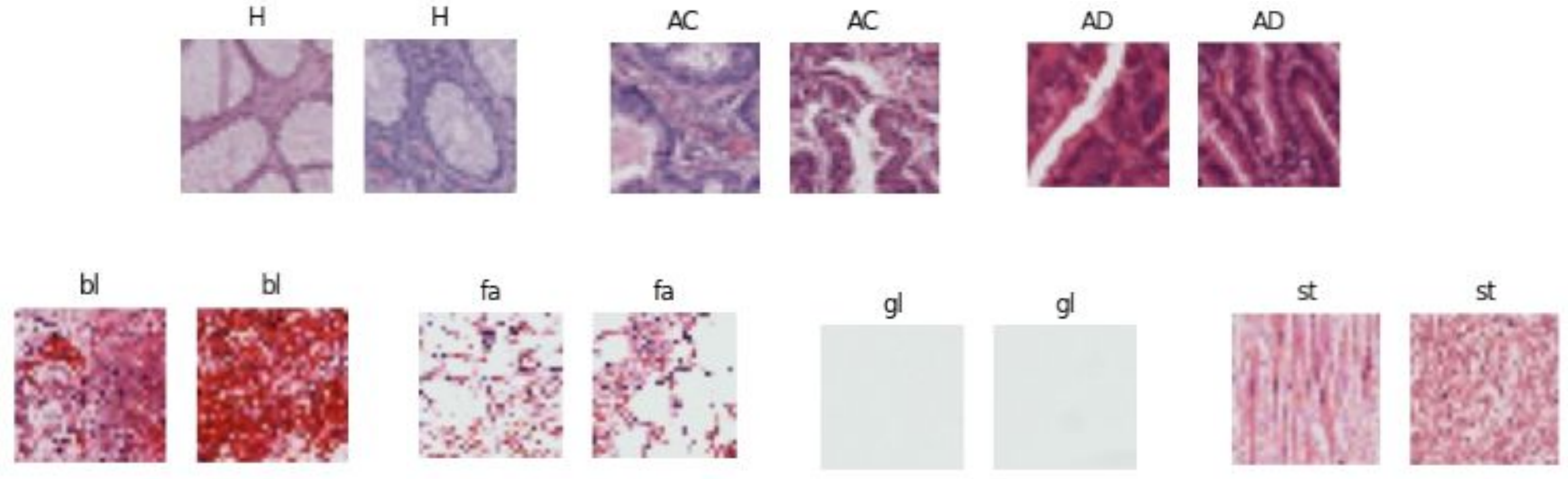
Dataset of images from 7 classes, 3 of which classes of interest:

- Healthy tissue (H)
- Cancer (AC)
- Adenoma (AD)

and 4 associated with a fake class from AC, AD, H:

- Blood (BLOOD)
- Fat (FAT)
- Glass (GLASS)
- Stroma (STROMA)

- The images (32x32x3) are in form of numpy arrays
- Divided in training set (12336 samples) and testing set (7308 samples)

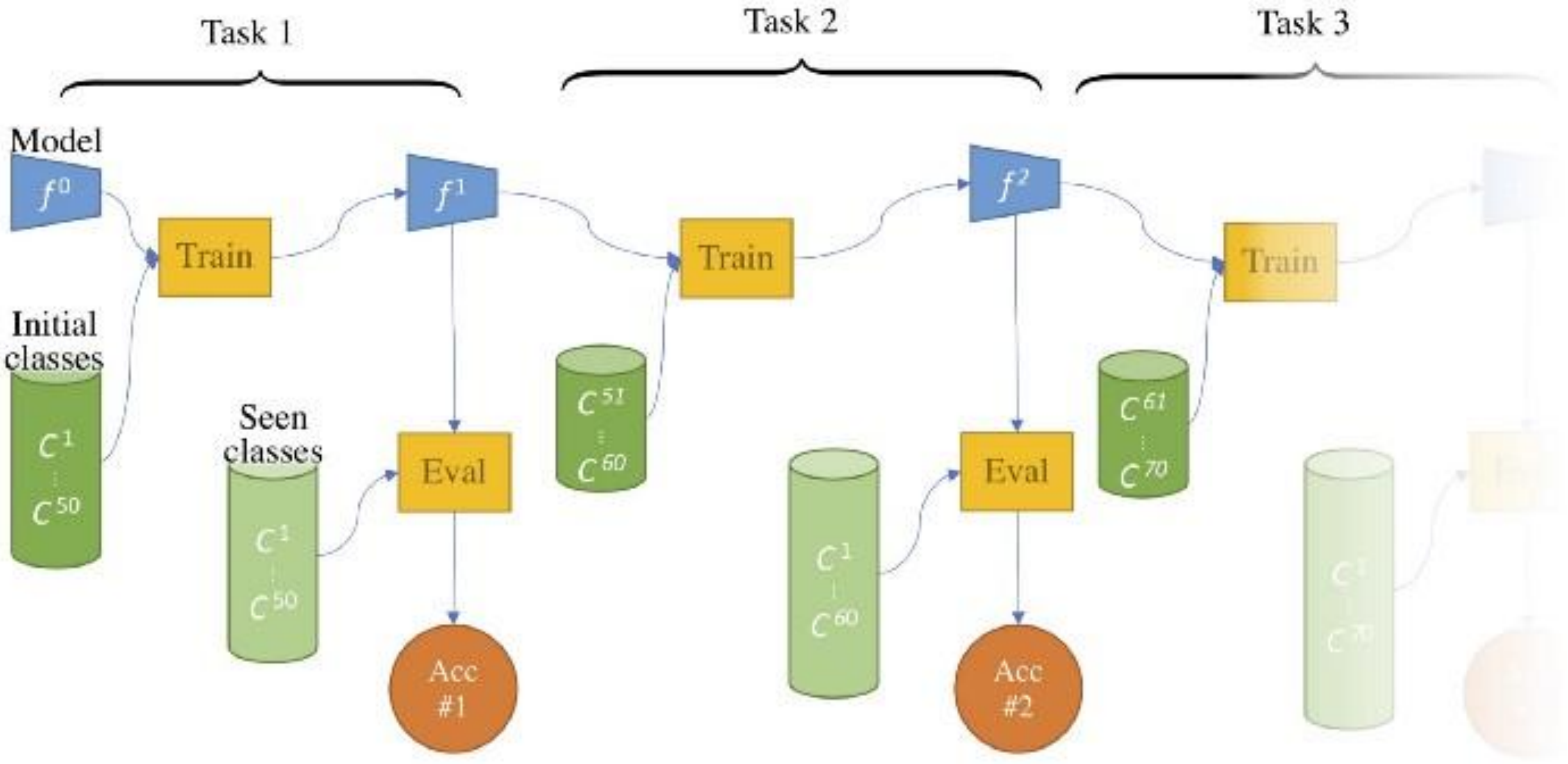


Class incremental learning

- Incrementally learn new classes as training data for them becomes available
- Remember the previous knowledge and expand it to the new data as well (new class)

Process:

- Train model on initial classes & evaluate performance
- Feed network with samples from new classes
- Perform training on the new part & keep previous knowledge


Task 1
Task 2
Task 3
Model
f^0
Train
f^1
Initial classes
C^1
C^{50}
Seen classes
C^1
C^{50}
Eval
Acc #1
Train
f^2
C^{51}
C^{60}
C^1
C^{60}
Eval
Acc #2
Train
C^{61}
C^{70}

Exemplar sets

- Dynamically created from dataset
- Maximum number of images in exemplar set shouldn't exceed parameter *K*
- Split *K* into *num_classes* parts, with each class having an equal number of samples present

Incremental learning: Training process

Initial step (number of classes = 1)

1. Record network’s output predictions
2. Compute loss and back-propagate the error
3. Repeat steps 1-2 for all batches of the samples

Next steps (number of classes > 1)

1. Save a copy of the old network
2. Update the network by adding one more output neuron
3. Record the network’s output predictions
4. Compute total loss and back-propagate the error
5. Repeat steps 3-4 for all batches of the samples

- Choose new exemplars after each training step
- Test current model performance
- Process of updating the network

Incremental learning: Loss function

- *Classification* loss (BCEWithLogitsLoss):

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

- *Distillation* loss (MSELoss):

$$\ell(x, y) = L = l_1, \ldots, l_N{}^{\top}, l_n = (x_n - y_n)^2$$

Number of classes = 1:

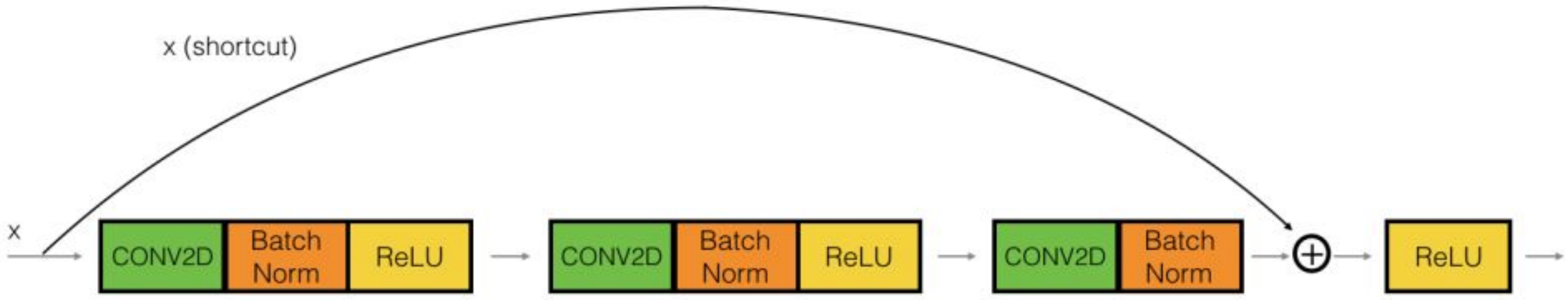
$$loss = loss_{classification}$$

Number of classes > 1:

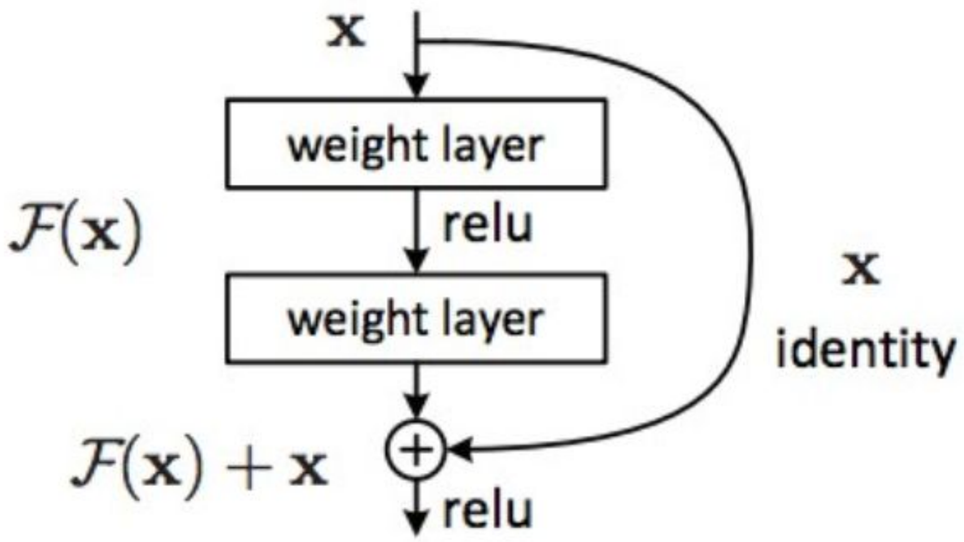
$$loss = loss_{classification} + loss_{distill} * lambda$$

Architecture: ResNet

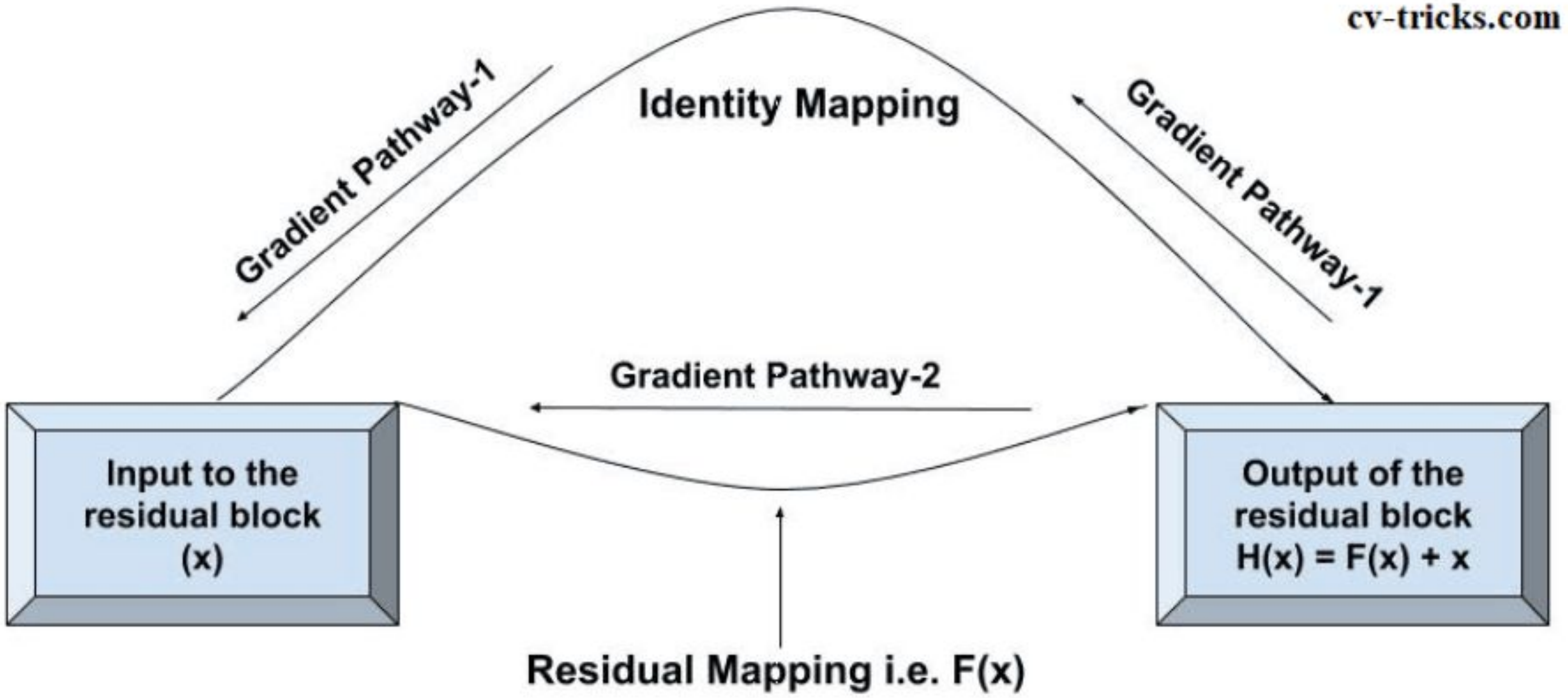
- Characterized by the addition of skip connections, or shortcuts to jump over some layer



- Introduces the skip connections (identity shortcut connections) for preserving the gradient
- They allow the flow of information from earlier layers in the network to later layers



- Skip connections allow an alternate shortcut path for the gradient to flow through during back-propagation



Network choice: ResNet32

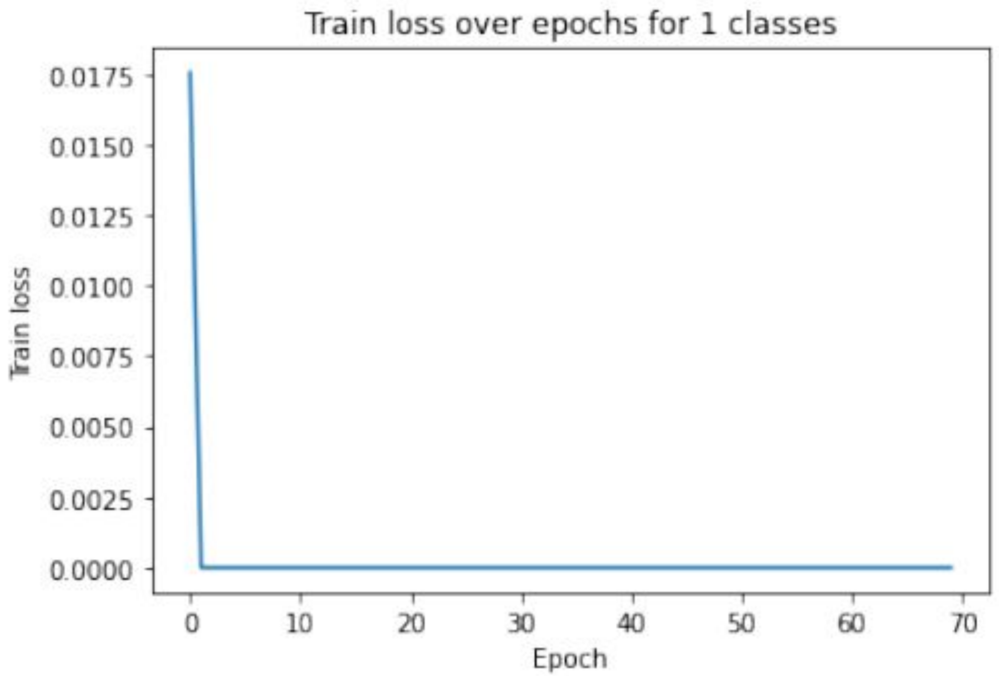
- Proved to be good when used in the incremental learning context (in particular in the CIL challenges on the CIFAR10 and CIFAR100)
- Use pre-trained ResNet32 model on CIFAR100

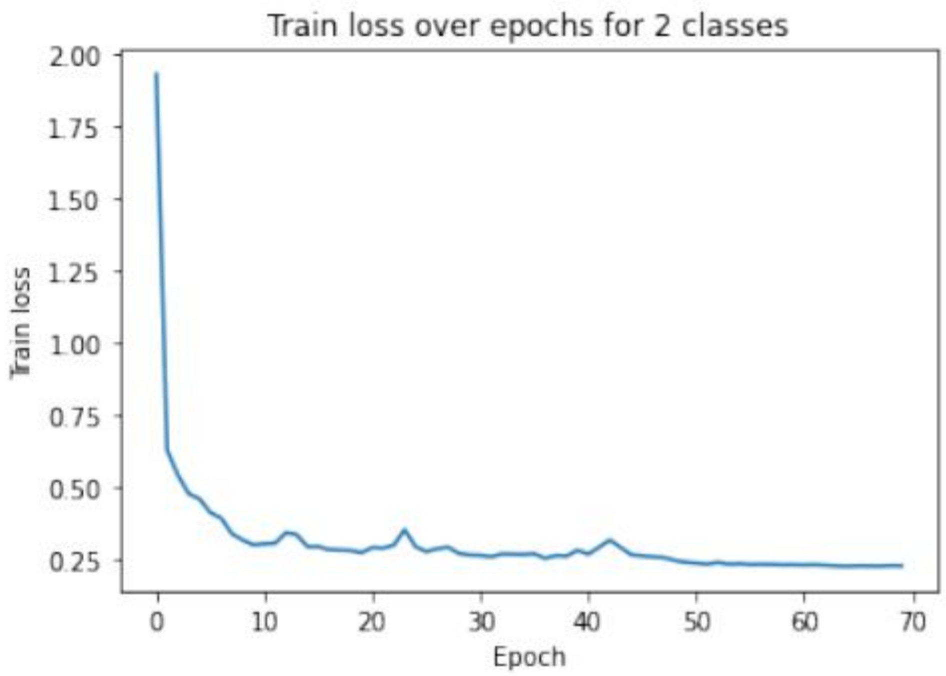
Fine tuning

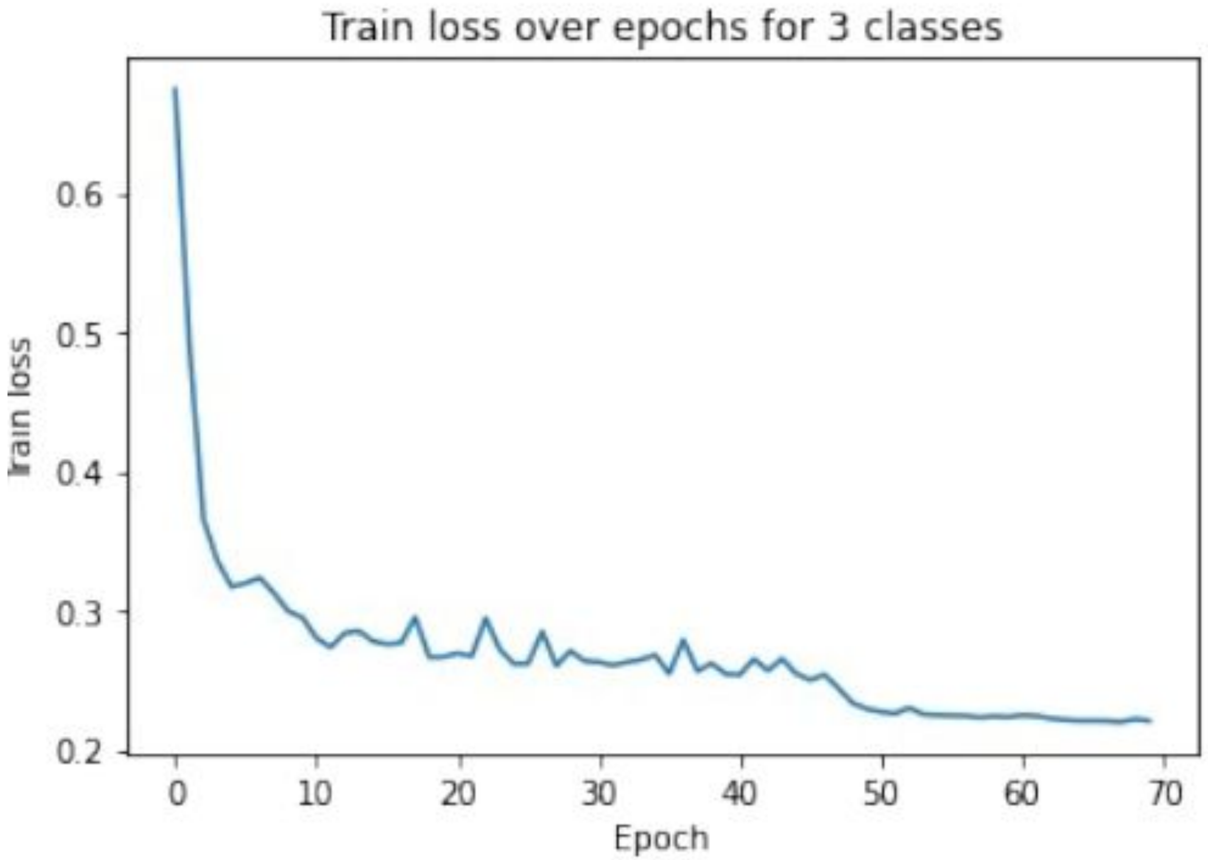
- Use full dataset with all classes in order to compare performance to incremental learning approach
- Use all class samples instead of exemplars
- Loss function is simple classification term (BCEWithLogitsLoss)

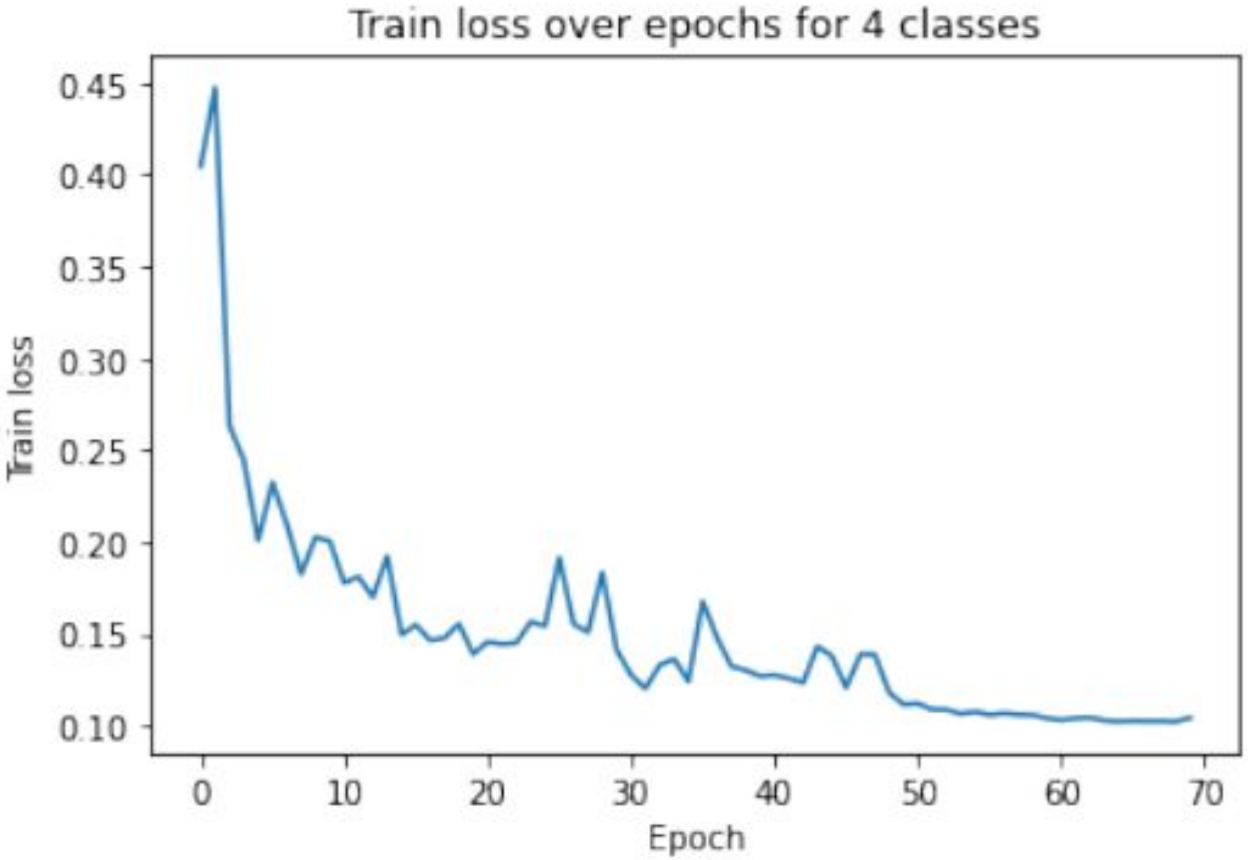
Results: ResNet32 pretrained on CIFAR100

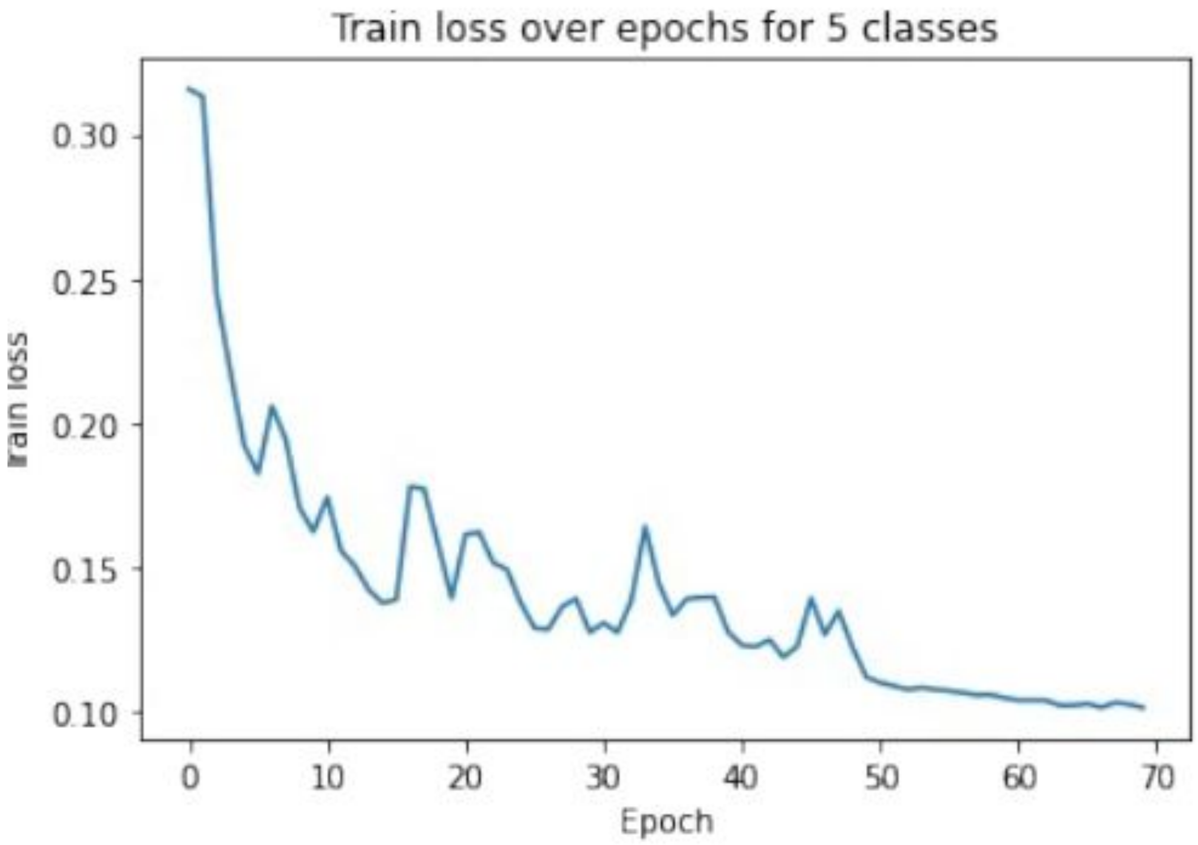
- Load pretrained ResNet32 model
- Replace FC layer with new Linear layer with output dimension = num. classes

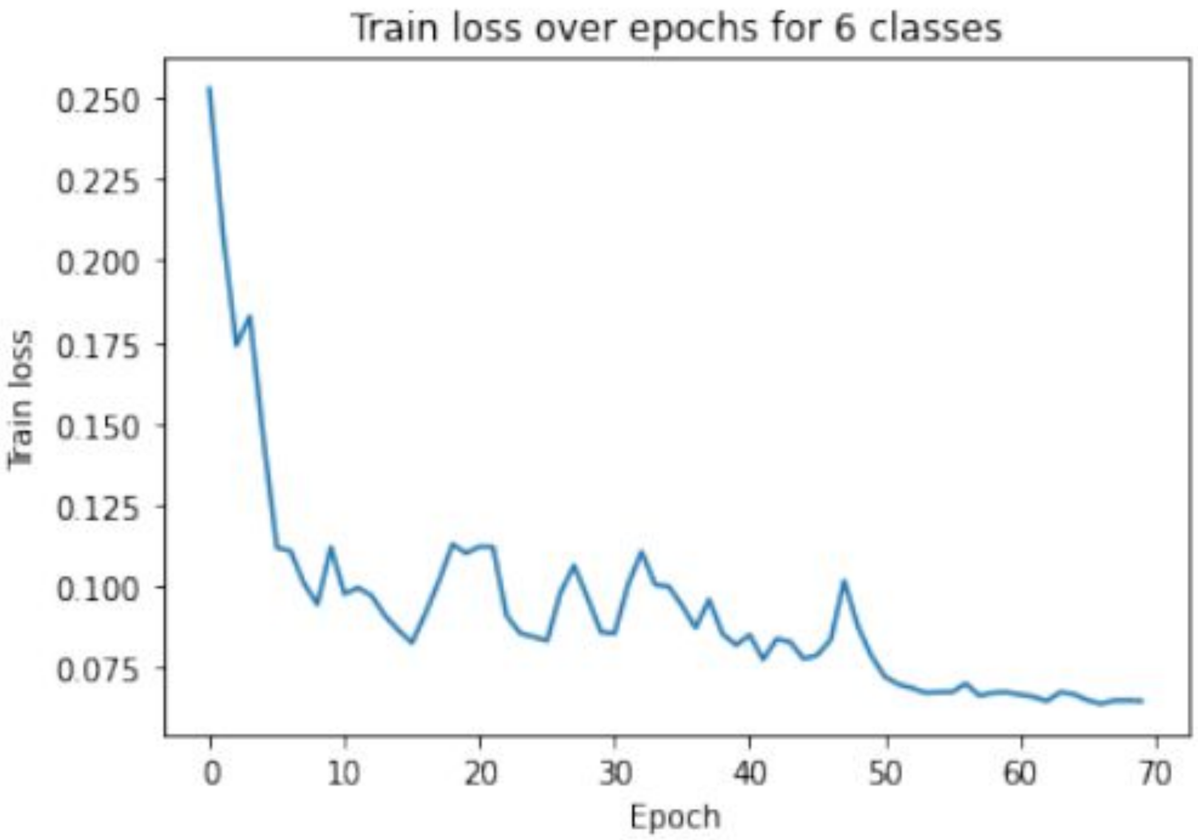


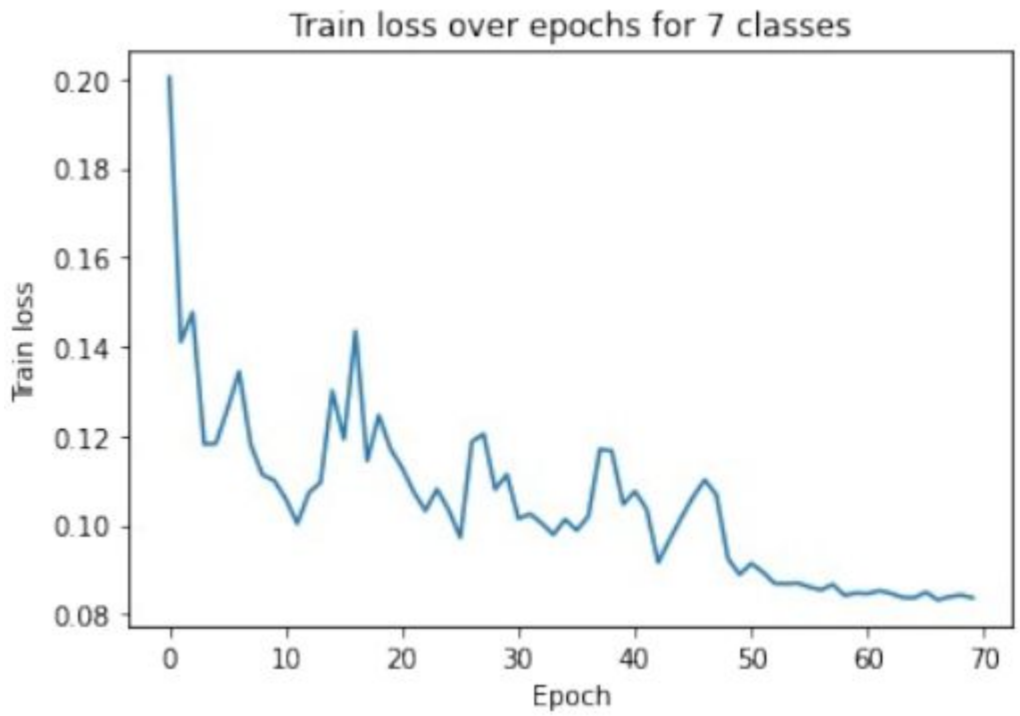


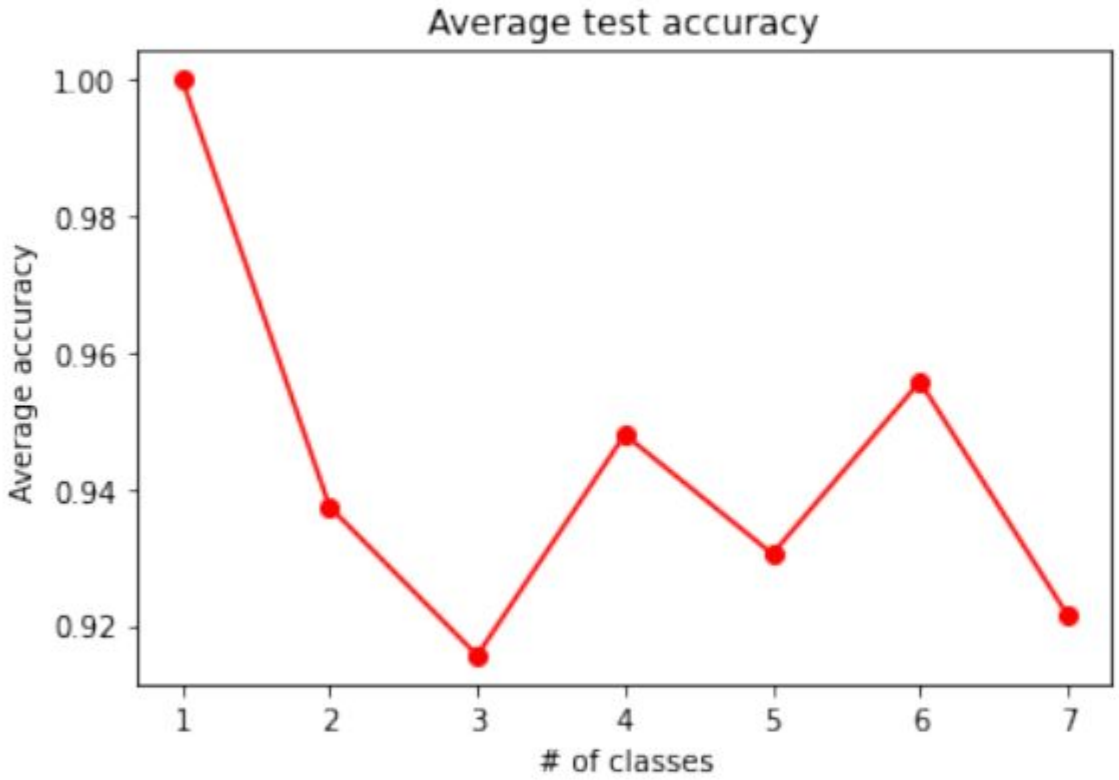

Train loss over epochs for 3 classes
Train loss
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50
60
70
Epoch


Train loss over epochs for 4 classes
Train loss
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0
10
20
30
40
50
60
70
Epoch


Train loss over epochs for 5 classes
Train loss
Epoch
0.30
0.25
0.20
0.15
0.10
0
10
20
30
40
50
60
70

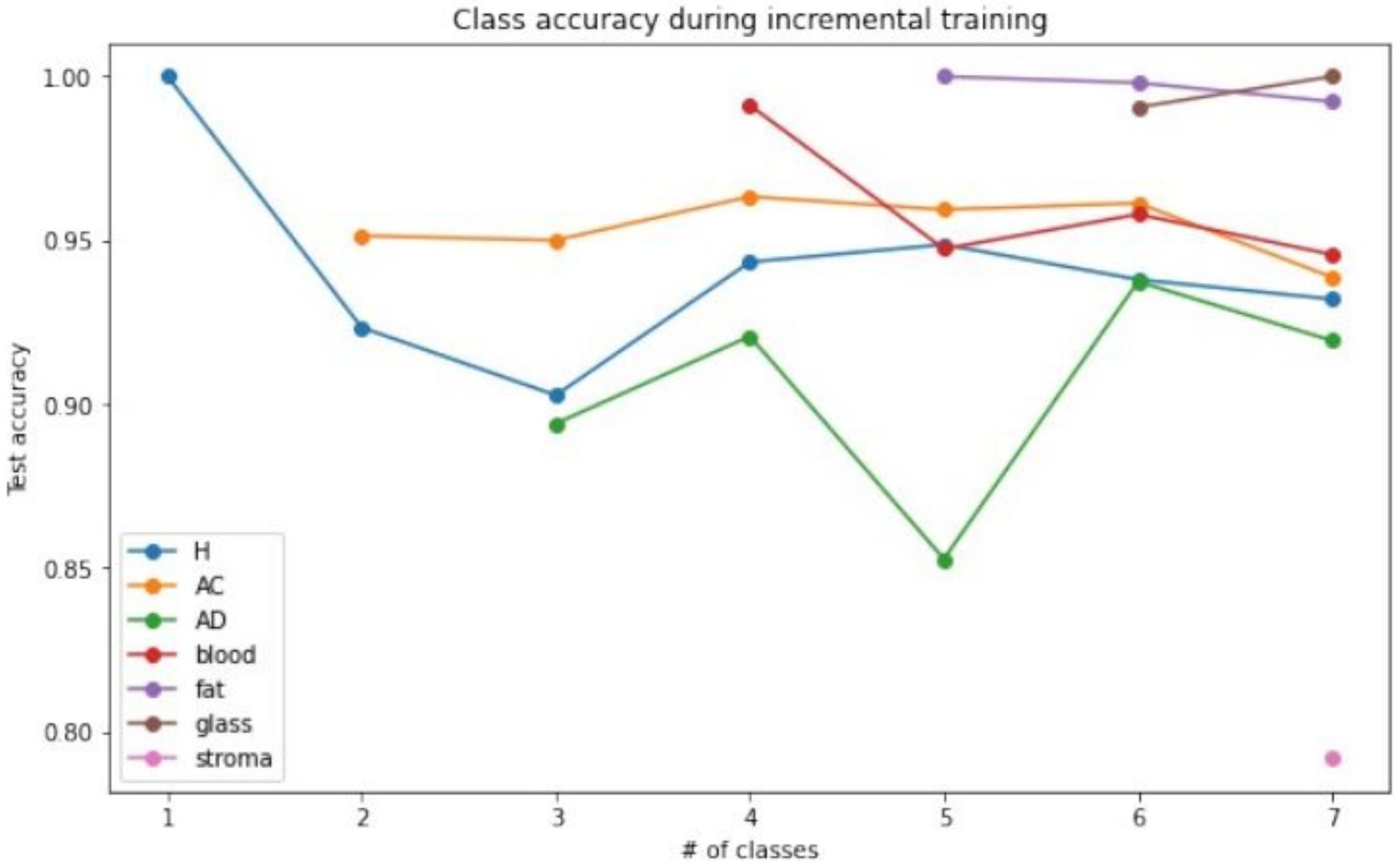

Train loss over epochs for 6 classes
Train loss
Epoch
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0
10
20
30
40
50
60
70





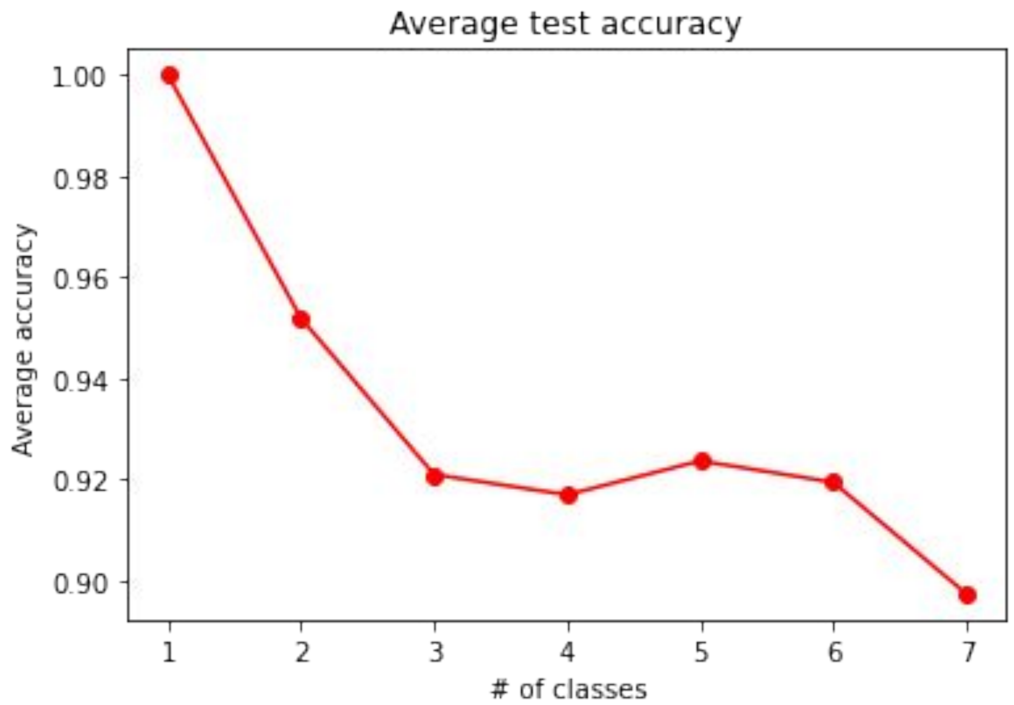
- After each train batch, testing is performed on all the classes up to that point

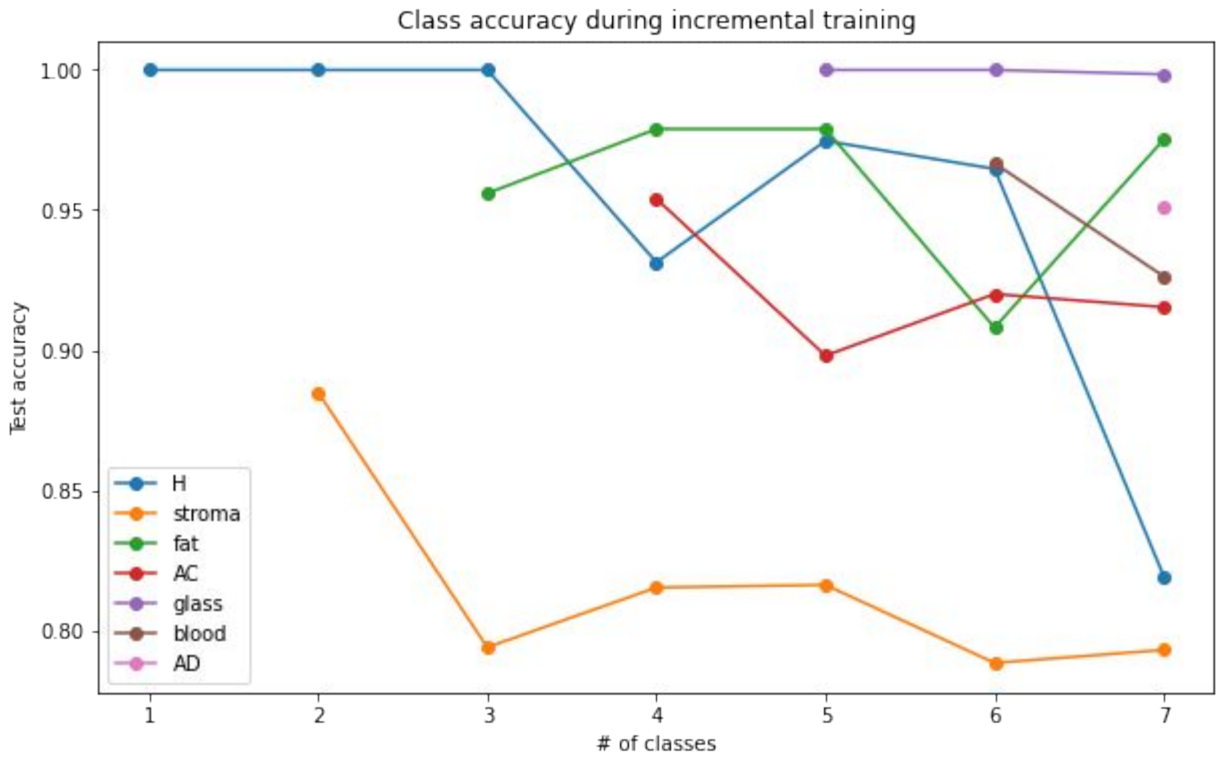
- Variation of each class accuracy during incremental learning:



Alternative class order

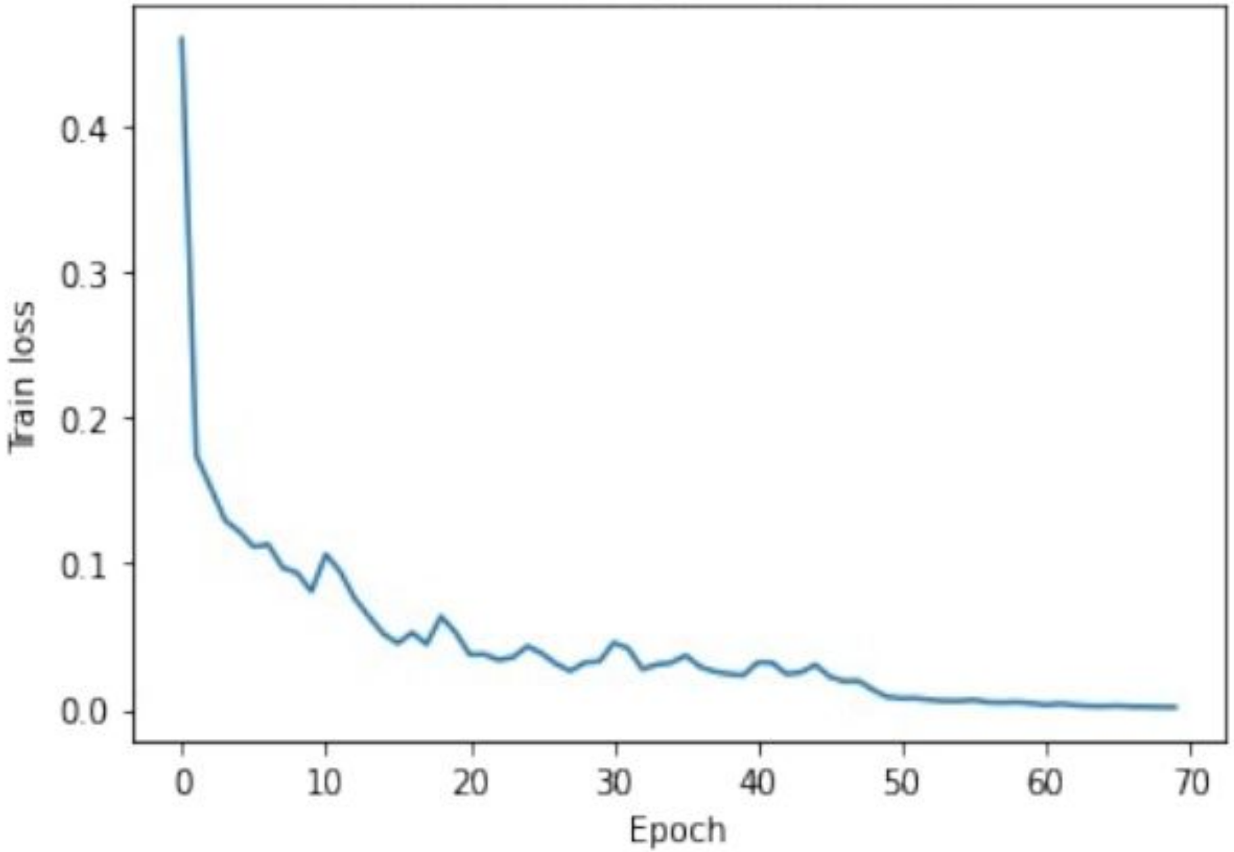
- Some class orders showed smoother accuracy decrease
- Example class order: H, stroma, fat, AC, glass, blood, AD





Results: ResNet32 model with finetuning

- Train ResNet32 model on the ful dataset
- Average test accuracy after evaluation of mode: ~87%



Comparison

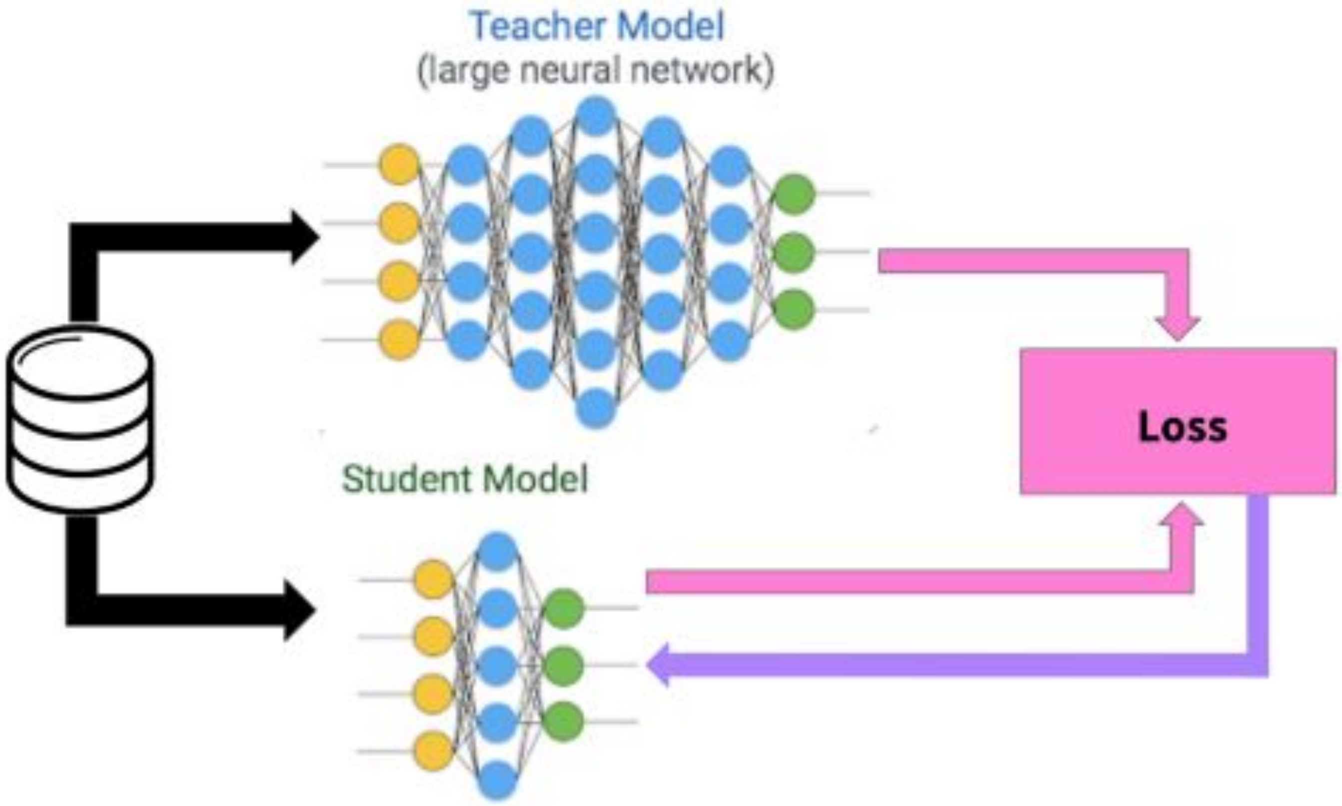
```
### Test accuracy for H: 0.94
    Test accuracy for AC: 0.88
    Test accuracy for AD: 0.76
    Test accuracy for blood: 0.92
    Test accuracy for fat: 0.98
    Test accuracy for glass: 1.0
    Test accuracy for stroma: 0.79
# Total Accuracy: 0.87
```

```
### Test accuracy for H: 0.93
    Test accuracy for AC: 0.94
    Test accuracy for AD: 0.92
    Test accuracy for blood: 0.95
    Test accuracy for fat: 0.99
    Test accuracy for glass: 1.0
    Test accuracy for stroma: 0.79
# Total Accuracy: 0.92
```

Fine tuning (on the left) compared to Incremental learning default order (on the right)

Knowledge distillation

- Train the simple student network to replicate the behavior of the more complex (already trained) teacher network



Knowledge distillation: Training steps

- Load the trained teacher model & set to evaluation mode
- Initialize a simple student model

Training process:

1. Compute output of teacher and student networks for current samples
2. Compute loss and back-propage it
3. Repeat 1-2 for all batches of samples

Knowledge distillation: Loss function

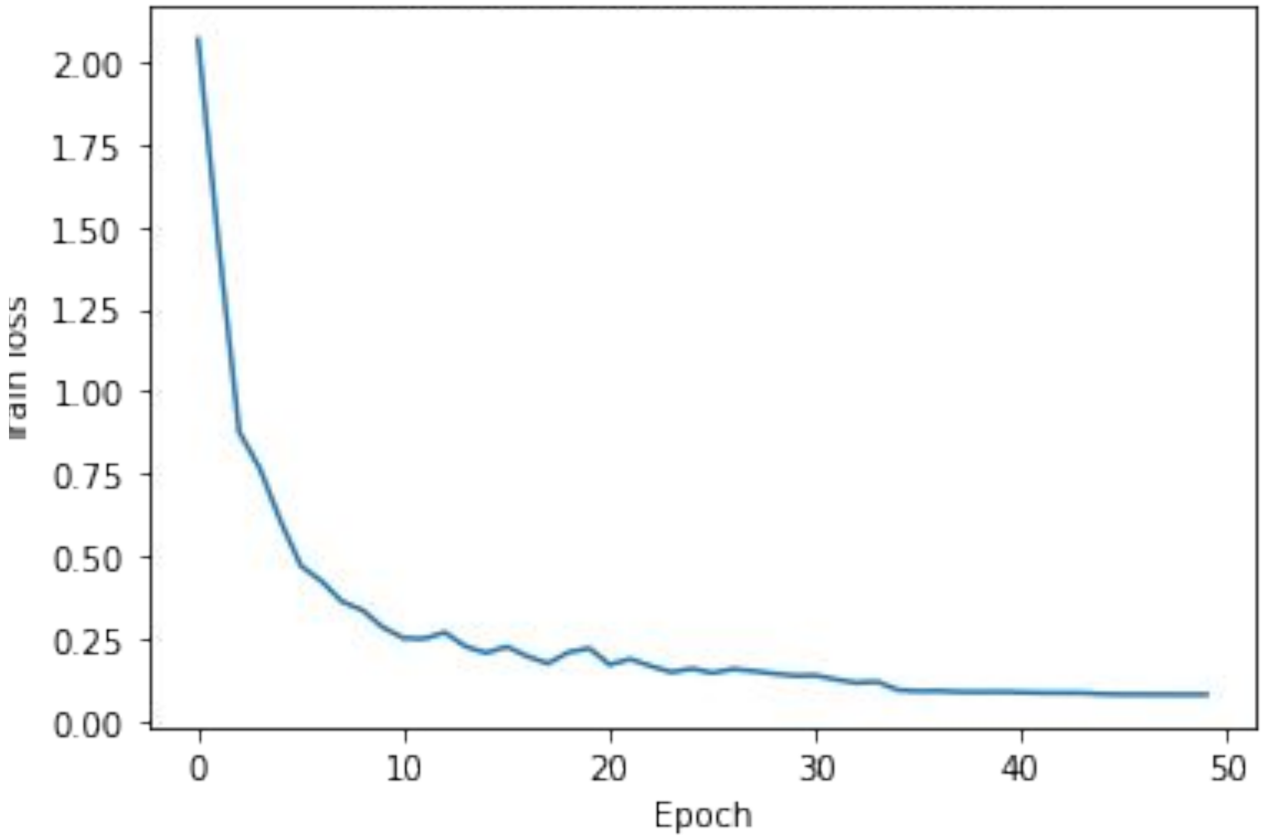
$$loss = loss_{classification} + loss_{kd\ distill} * lambda_{kd}$$

- *Classification* loss: difference between student network's output and target (BCEWithLogitsLoss used)
- *Knowledge distillation* loss: difference between student network's output and recorded teacher's output (MSELoss used)

Knowledge distillation: Results

Starting from pre-trained ResNet32 model

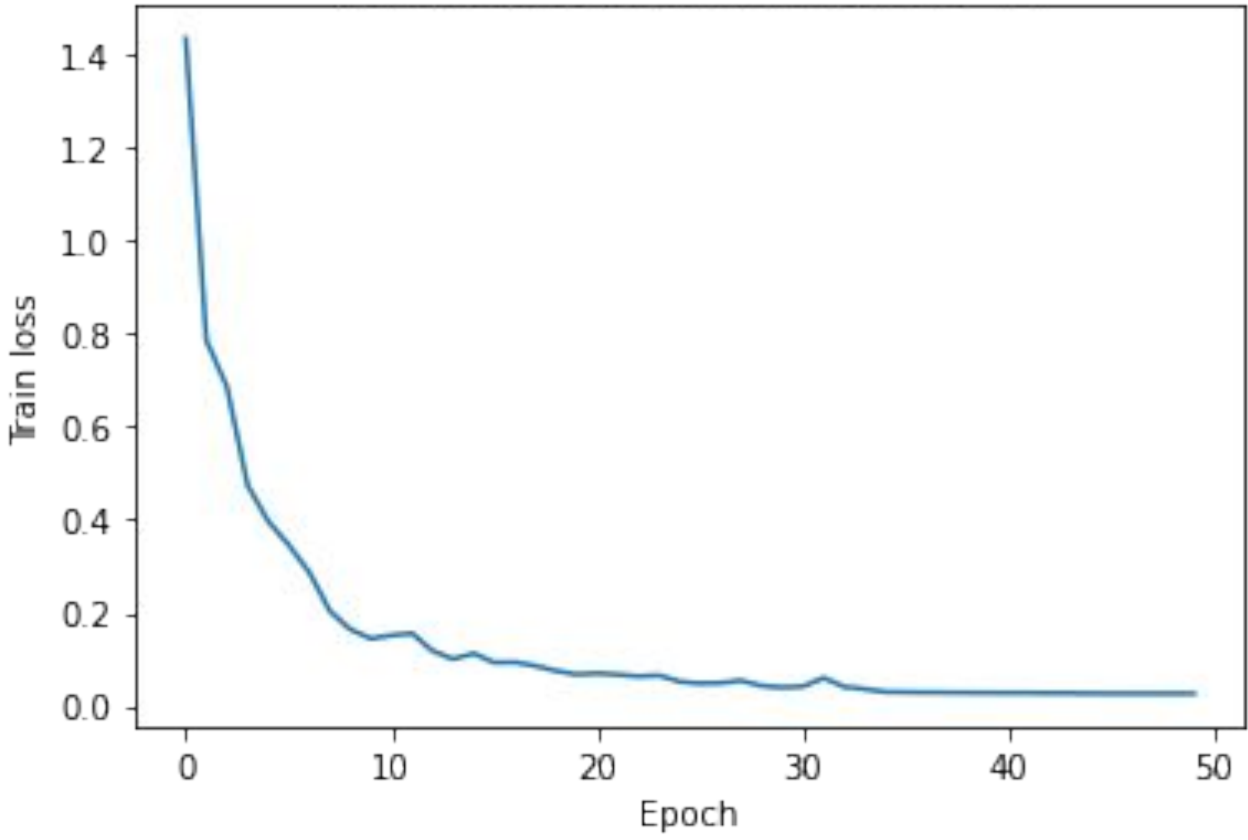
- After training of student network the average test accuracy is about 86%
- Without distillation loss term average test accuracy: 77%



Knowledge distillation: Results

Starting from finetuning

- After training of student network the average test accuracy is about 85%
- Without distillation loss average test accuracy: 77%



Conclusions

- Incremental learning performed slightly better than fine tuning
- Combination of replay and distillation gives the best results
- Distillation works also for encouraging a small network to behave similar to the bigger (trained) network

Thank you for your attention!