Project HLB
Hyper Lightspeed Bench
on Imagenette Dataset

2A – Louis Caubet, Firas Ben Jedidia, Long Vân Tran Ha, Inès Vignal

https://github.com/LouisCaubet/hlb-imagenette

Overview of HLB -Hyper Lightspeed Bench repo

What is it?

Motivation

- Speed and cost
- State-of-the-art techniques
- DAWNBench

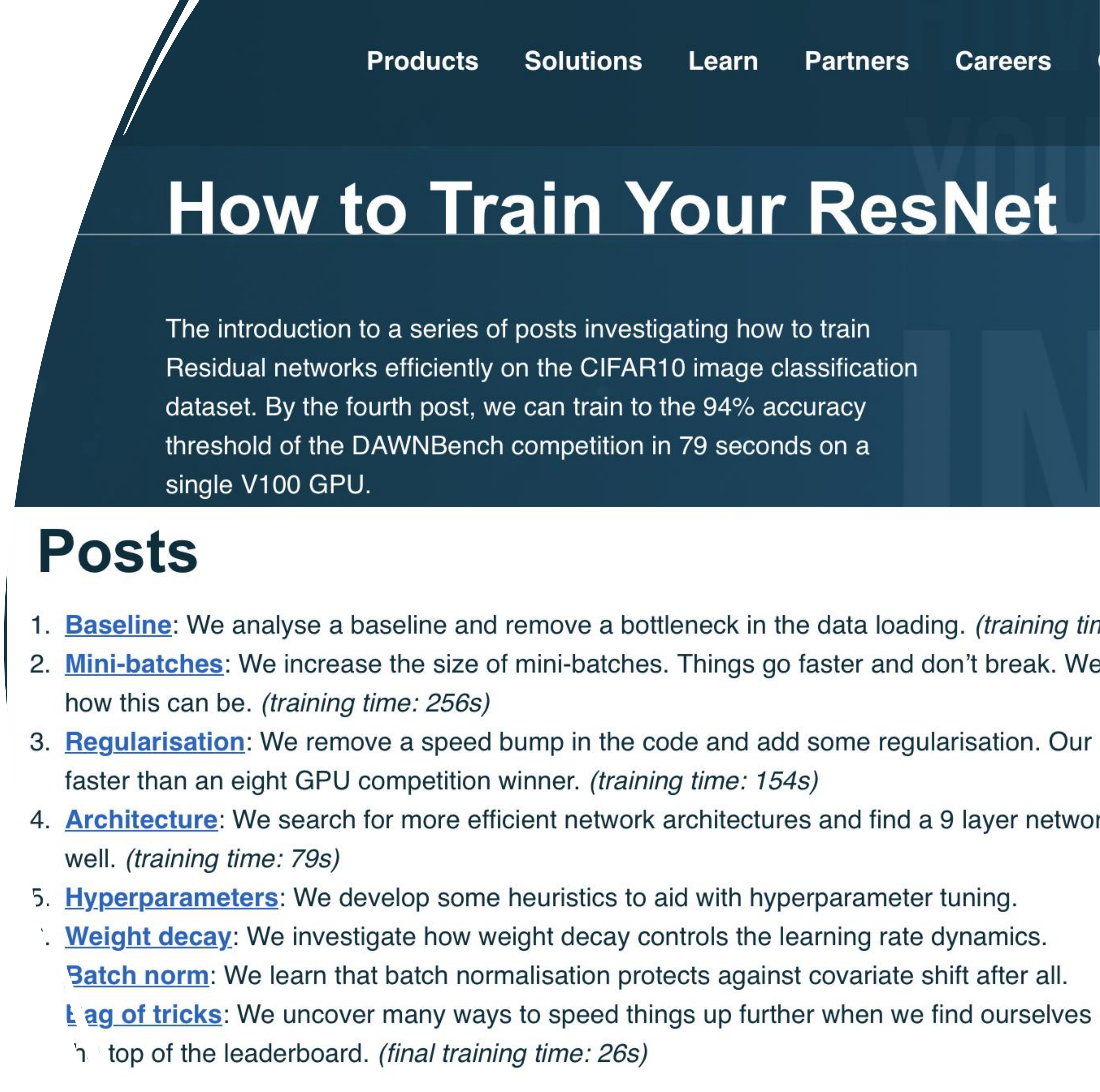
About Submit

) Training

ubmission Date	Model	Time to 94% Accuracy	Cost (USD)	Max Accuracy	Hardwa
Dec 2019	Custom Resnet 9 *Santiago Akle Serrano, Hadi Pour Ansari, Vipul Gupta, Dennis DeCoste* source	0:00:10	N/A	94.23%	Tesla V100 * 8 GP 40 CPU
Jan 2020	Custom ResNet 9 *Ajay Uppili Arasanipalai* source	0:00:11	N/A	94.05%	IBM AC922 + 4 * V100 (NCSA
Oct 2019	Kakao Brain Custom ResNet9 *clint@KakaoBrain* source	0:00:28	N/A	94.04%	Tesla V100 * 4 GP 56 CPU (Kaka BrainClou
May 2019	BaiduNet9P *Baidu USA GAIT LEOPARD team: Baopu Li, Zhiyu Cheng, Yingze Bao* source	0:00:45	$0.11	94.18%	Baidu Cloud Tesl 16GB/448 GB/
Oct 2019	Kakao Brain Custom ResNet9 *clint@KakaoBrain* source	0:00:58	N/A	94.20%	Tesla V100 * 1 GP 56 CPU (Kaka BrainClou
May 2019	BaiduNet9 *Baidu USA GAIT LEOPARD team: Baopu Li, Zhiyu Cheng, Yingze Bao* source	0:01:12	$0.02	94.10%	Baidu Cloud Tesl 16GB/56 GB/
Apr 2019	Custom ResNet 9 *Ajay Uppili Arasanipalai* source	0:01:14	N/A	94.06%	IBM AC922 + Nv V100 (Nimbix
Nov 2018	Custom ResNet 9 *David Page, myrtle.ai* source	0:01:15	$0.06	94.08%	V100 (AWS p3.

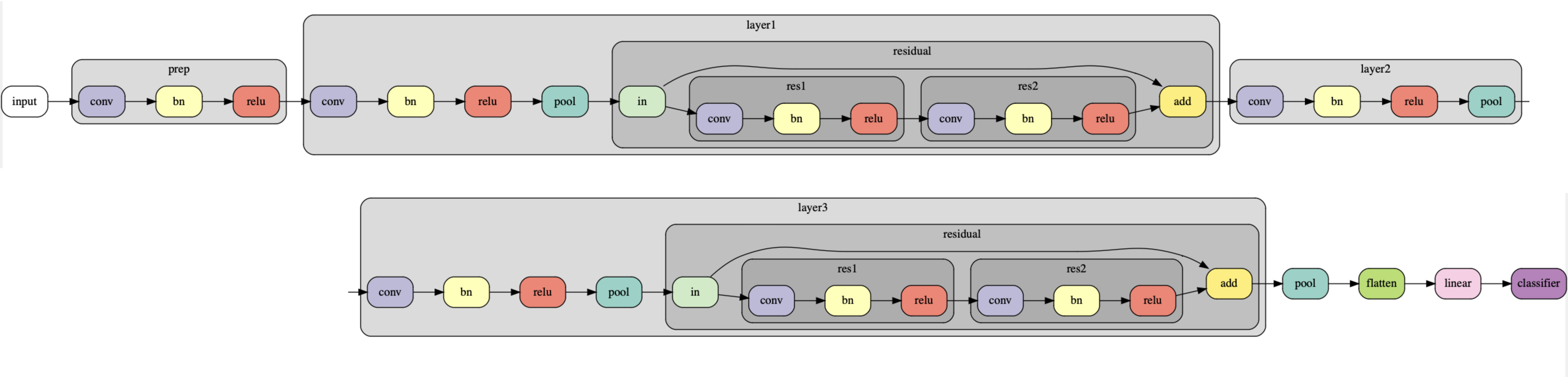
HLB - Hyper Lightspeed Bench

- Simple repo GitHub
- Blog How to Train Your ResNet
- Hackable
- Cifar-10



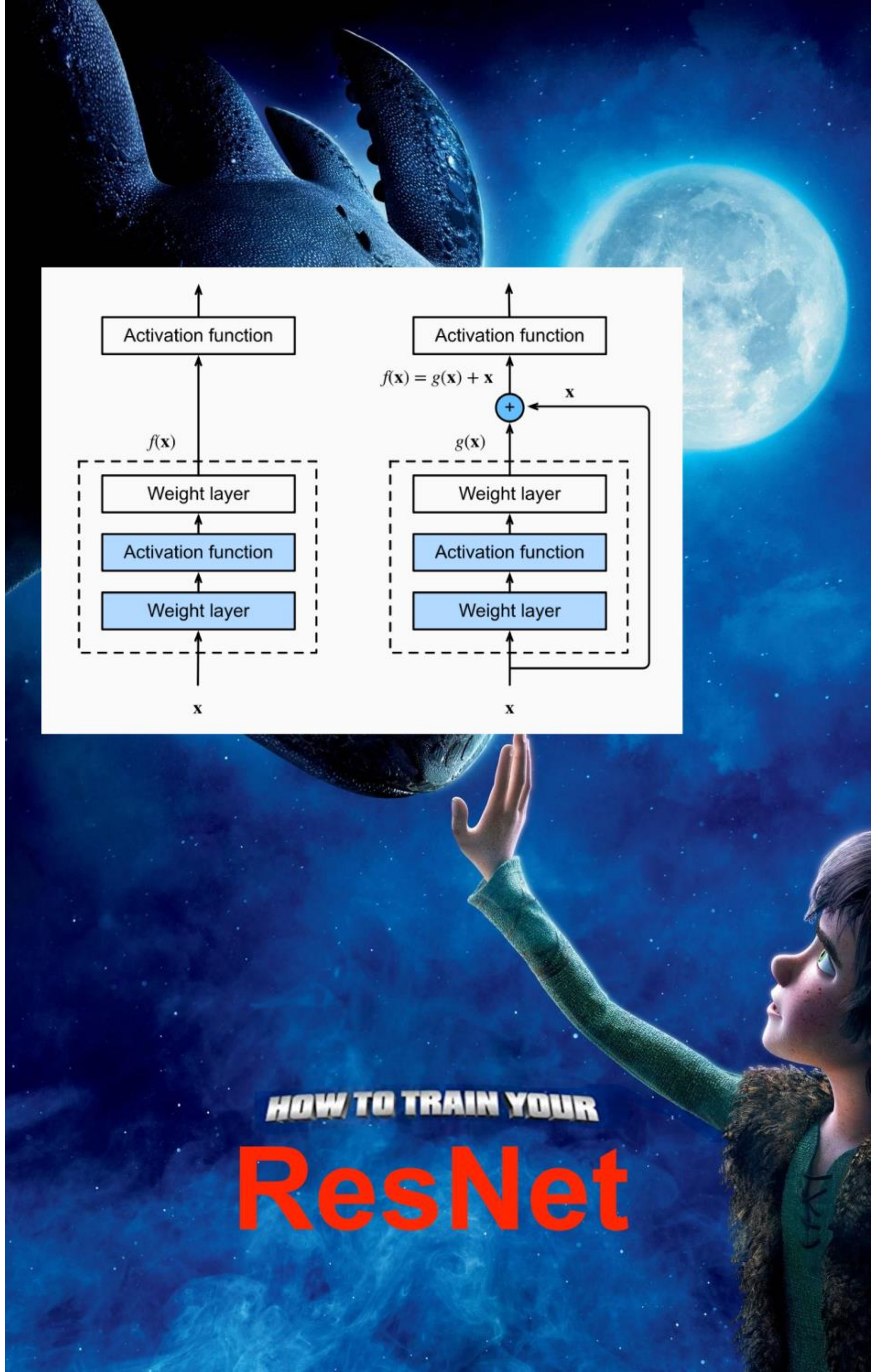
Architecture inspiration

24 epochs, 79 seconds, 94% on CIFAR-10 !



Goal of our MAP583 project

- Use another dataset: Imagenette
- Hack the code
- Optimize
- Compare the results to state-of-the-art models



Outline

Imagenette dataset

Scheduling the learning rate

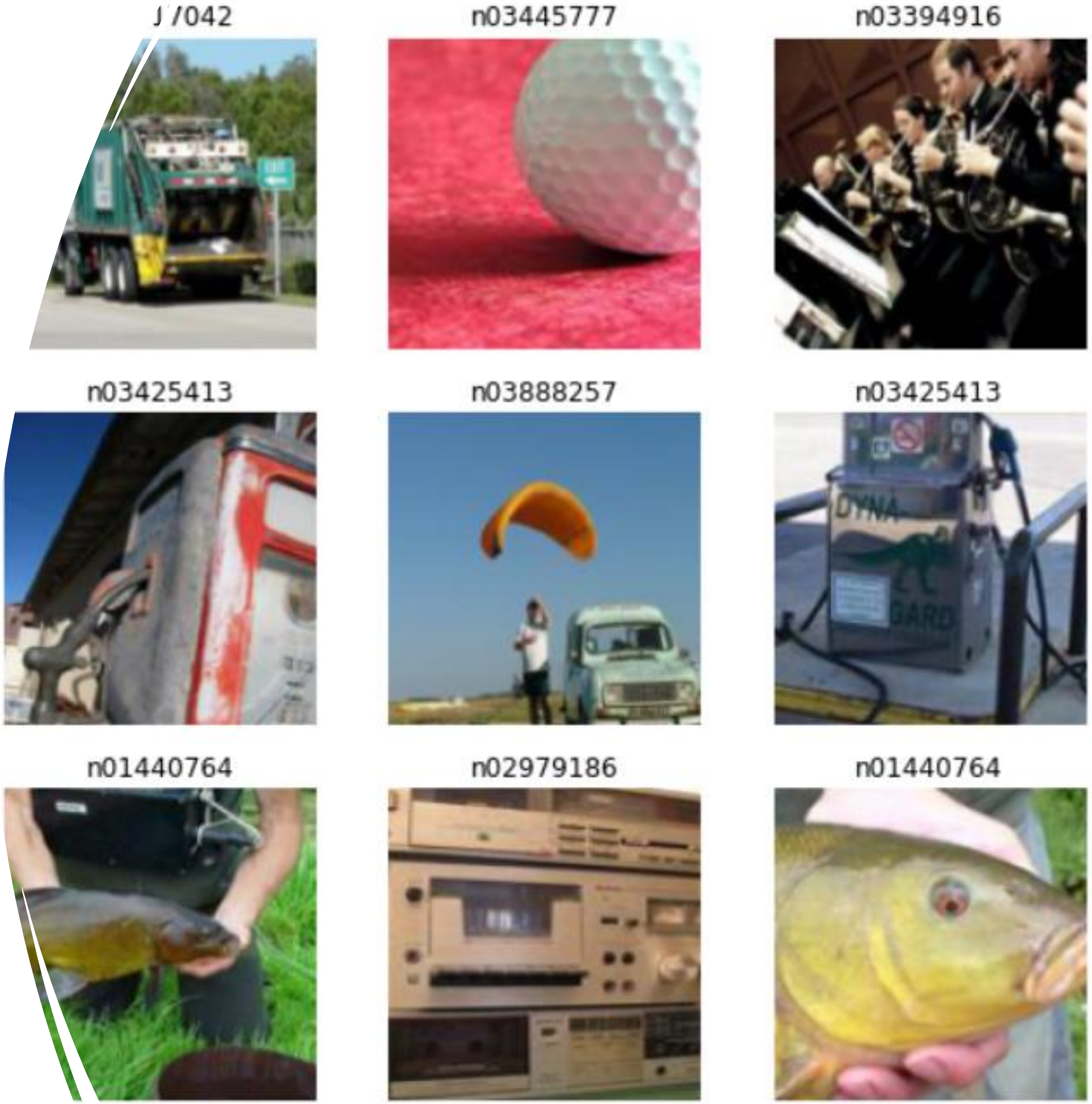
Ablation study

Adapting to another dataset

How well does this technique generalize?

ImageNette

- Smaller version of ImageNet with only 10 classes



Loading ImageNette in PyTorch

```python
from torch.utils.data import Dataset
import os
from PIL import Image
import numpy as np
from torchvision.transforms import transforms

IMAGENETTE_CLASSES = {
    'tench': 'n01440764',
    'English springer': 'n02102040',
    'cassette player': 'n02979186',
    'chain saw': 'n03000684',
    'church': 'n03028079',
    'French horn': 'n03394916',
    'garbage truck': 'n03417042',
    'gas pump': 'n03425413',
    'golf ball': 'n03445777',
    'parachute': 'n03888257'
}

CLASSES_TO_IDX = {
    'tench': 0,
    'English springer': 1,
    'cassette player': 2,
    'chain saw': 3,
    'church': 4,
    'French horn': 5,
    'garbage truck': 6,
    'gas pump': 7,
    'golf ball': 8,
    'parachute': 9
}
```

```python
class Imagenette(Dataset):

    def __init__(self, path: str, train: bool):
        self.path = path
        self.train = train
        self.transform = transforms.Compose([
            transforms.Resize((64, 64)),
            transforms.ToTensor(),
        ])

        if train:
            self.path += '/train'
        else:
            self.path += '/val'

        self.images = []
        self.labels = []

        for label, folder in IMAGENETTE_CLASSES.items():
            label_idx = CLASSES_TO_IDX[label]
            for image in os.listdir(f'{self.path}/{folder}'):
                self.images.append(f'{self.path}/{folder}/{image}')
                self.labels.append(label_idx)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        # Load image with PIL
        image = Image.open(self.images[index]).convert('RGB')
        tensor = self.transform(image)
        image.close()
        return tensor, self.labels[index]
```

ImageNette images are larger than in CIFAR10, require resizing to apply HLB

A few hyperparameters to tweak

The ImageNette dataset contains way fewer elements than CIFAR10, so we need more epochs (e.g. 50 takes the same time as 10 epochs on CIFAR10)

The eval batchsize must evenly divide the eval dataset. For ImageNette, we can set it to 785.

Results

37	1.1963	1.1926	0.8506	0.8652		38.3794
38	1.1885	1.2202	0.8457	0.8456		39.3834
39	1.1758	1.1834	0.8545	0.8711		40.3895
40	1.1523	1.1943	0.8730	0.8596		41.3931
41	1.1553	1.2247	0.8682	0.8380	0.8104	42.3999
42	1.1396	1.1855	0.8896	0.8708	0.8471	43.4090
43	1.1260	1.1464	0.8994	0.8859	0.8683	44.4256
44	1.1426	1.1592	0.8711	0.8815	0.8785	45.4402
45	1.0742	1.1387	0.9209	0.8940	0.8833	46.4573
46	1.0811	1.1332	0.9180	0.8971	0.8876	47.4783
47	1.0879	1.1386	0.9189	0.8955	0.8940	48.4997
48	1.0781	1.1295	0.9180	0.8991	0.8973	49.5192
49	1.0479	1.1290	0.9316	0.9004	0.8994	50.5423

End of training: we reach 90% accuracy after 50s.

(original HLB-CIFAR10 reaches 94% on CIFAR10 after 50s)

What’s the state-of-the-art?

On CIFAR10 [1]:

- Record 99.5% accuracy using Transformers
- >99% with a Resnet

1	ViT-H/14	99.5	632M	×	Worth 16x16 Words: Transformers for Image Recognition at

6	BiT-L (ResNet)	99.37	99.37	✓	(BiT): General Visual Representatio

On ImageNette [2]:

- Best around 95%

Imagenette Leaderboard

Size (px)	Epochs	URL	Accuracy	# Runs
128	5	fastai2 train_imagenette.py 2020-10 + MaxBlurPool + tuned hyperparams	87.43%	5, mean
128	20	fastai2 train_imagenette.py 2020-01 + MaxBlurPool	91.57%	5, mean
128	80	fastai2 train_imagenette.py 2020-01	93.55%	1
128	200	fastai2 train_imagenette.py 2020-01	94.24%	1
192	5	fastai2 train_imagenette.py 2020-01 + MaxBlurPool	86.76%	5, mean
192	20	fastai2 train_imagenette.py 2020-01 + MaxBlurPool	92.50%	5, mean
192	80	fastai2 train_imagenette.py 2020-01	94.50%	1
192	200	fastai2 train_imagenette.py 2020-01	95.03%	1
256	5	fastai2 train_imagenette.py 2020-01 + MaxBlurPool	86.85%	5, mean
256	20	fastai2 train_imagenette.py 2020-01 + MaxBlurPool	93.53%	5, mean
256	80	fastai2 train_imagenette.py 2020-01	94.90%	1
256	200	fastai2 train_imagenette.py 2020-01	95.11%	1

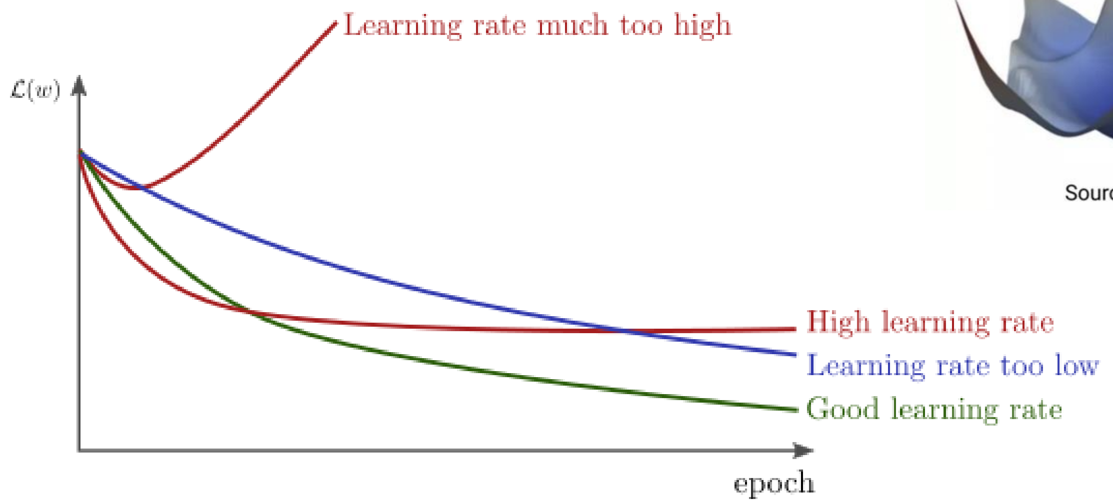
[1]: Source https://paperswithcode.com/sota/image-classification-on-cifar-10
[2]: Source https://github.com/fastai/imagenette

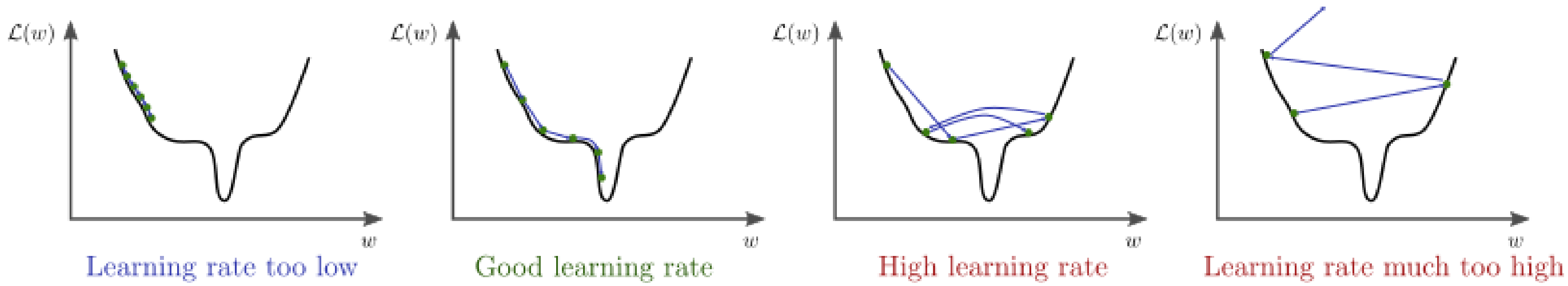
Adaptive learning rate

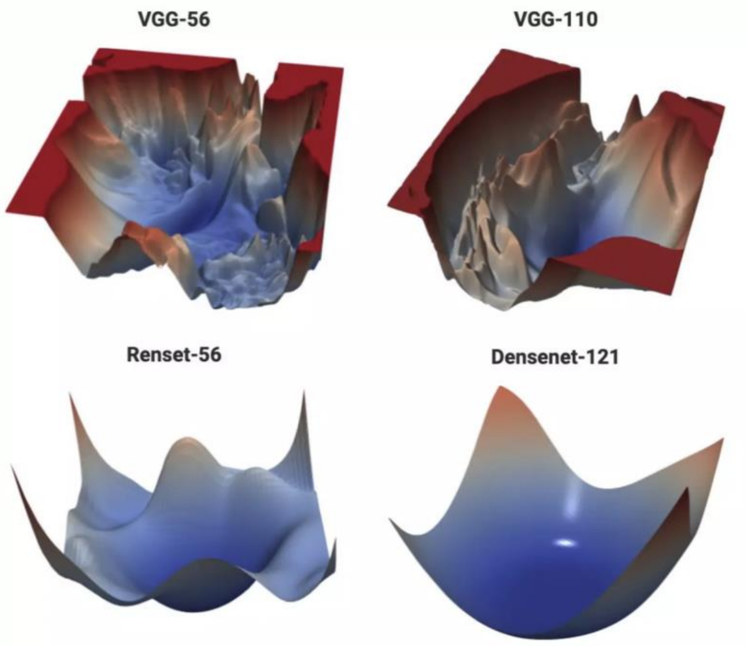
Improving performance with learning rate scheduling

Why use adaptive learning rate?

- Faster convergence
- More stable training
- Better generalization



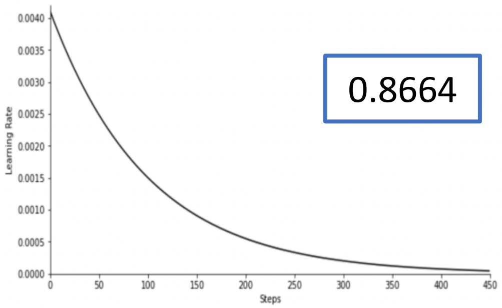




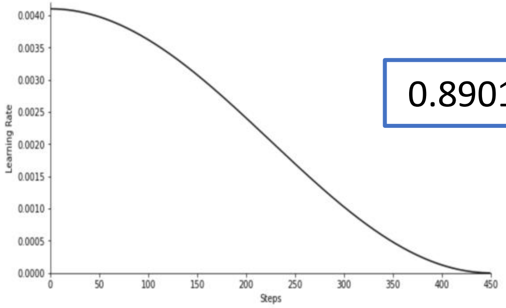
Source: VISUALIZING THE LOSS LANDSCAPE OF NEURAL NETS

Popular Learning Rate Schedules

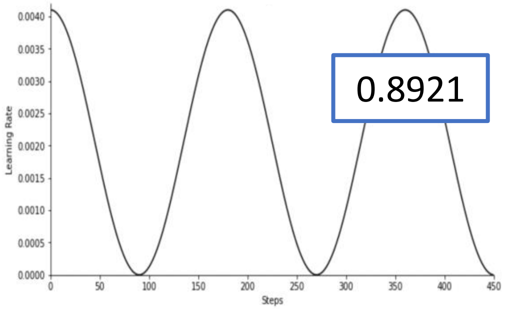
ExponentialLR



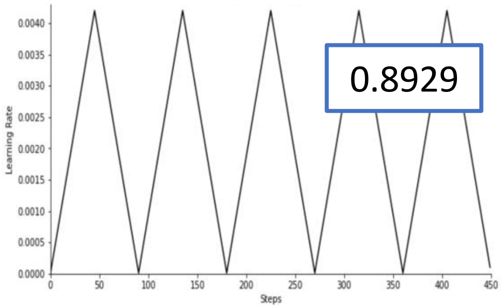
CosineAnnealing



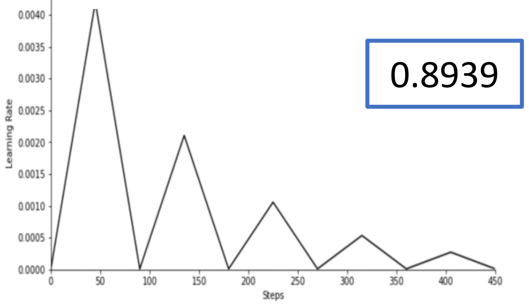
CosineAnnealingLR



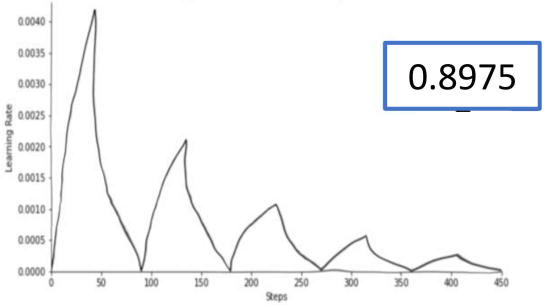
CyclicLR(triangular)



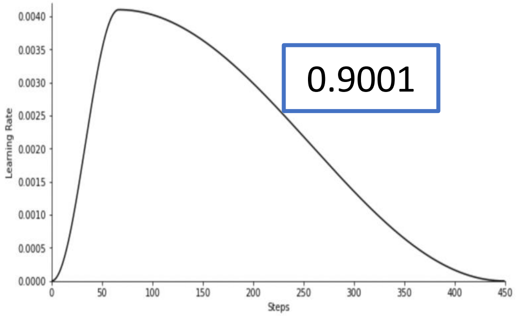
CyclicLR(triangular2)



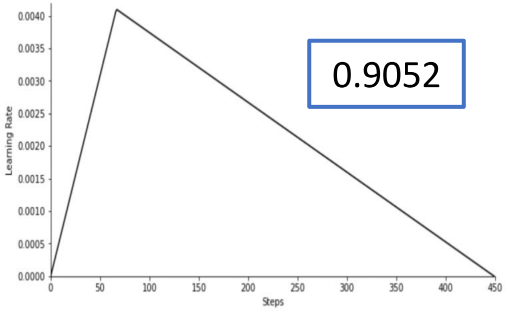
CyclicLR(exp-range)



OneCycleLR(cos)

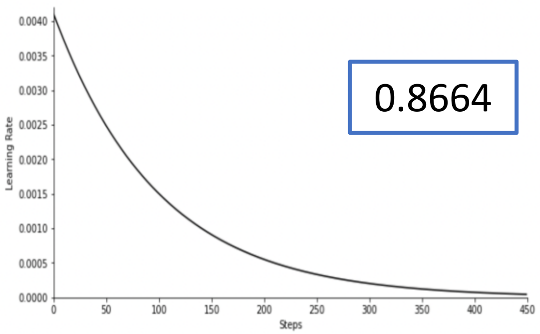


OneCycleLR(linear)

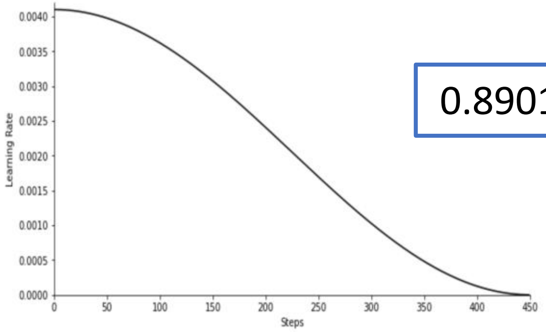


Performance Analysis – 1/3

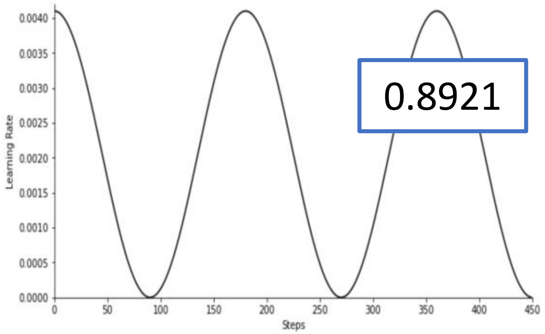
ExponentialLR



CosineAnnealing



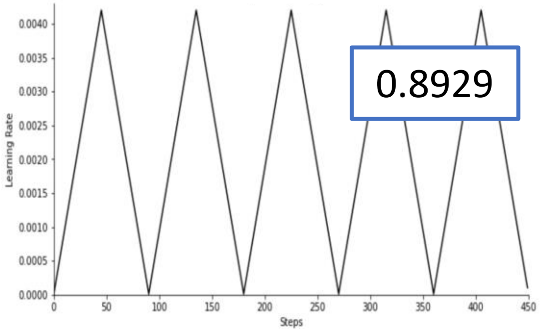
CosineAnnealingLR



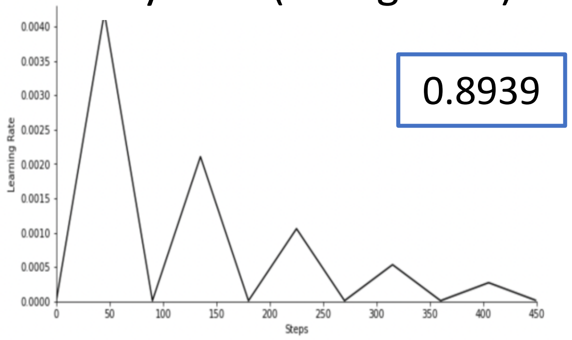
- Exponential decay
- Reduce LR slowly
  → less overfitting
- Introduces cyclic scheduling

Performance Analysis – 2/3

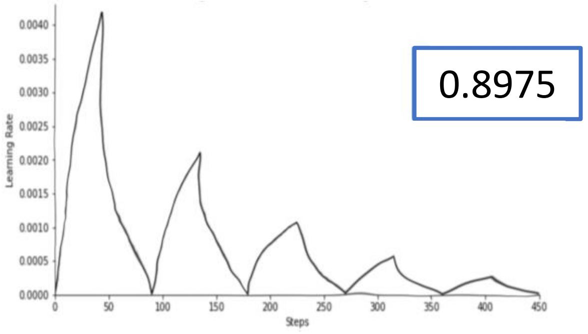
CyclicLR(triangular)



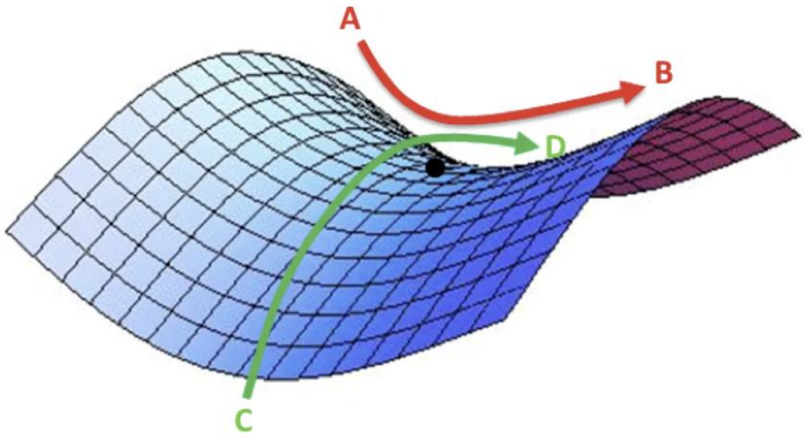
CyclicLR(triangular2)



CyclicLR(exp-range)

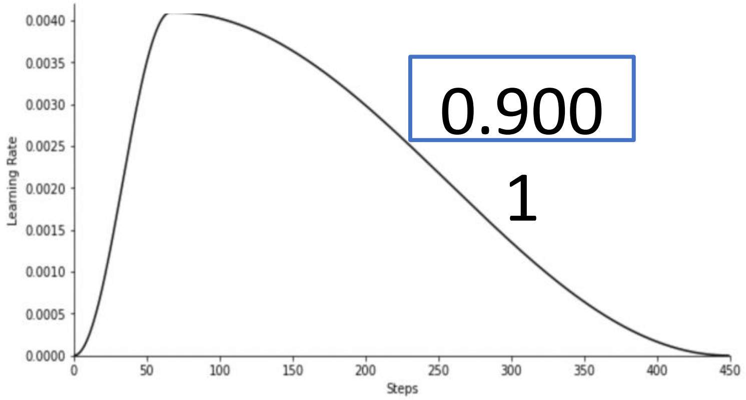


- Cyclic schedules allows the model to escape local minima with « warm restarts »
- With exp-range, more exploration is allowed

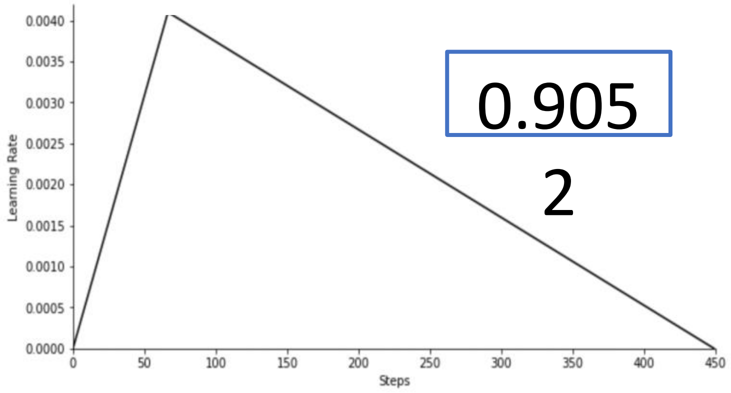


Performance Analysis – 3/3

OneCycleLR(cos)

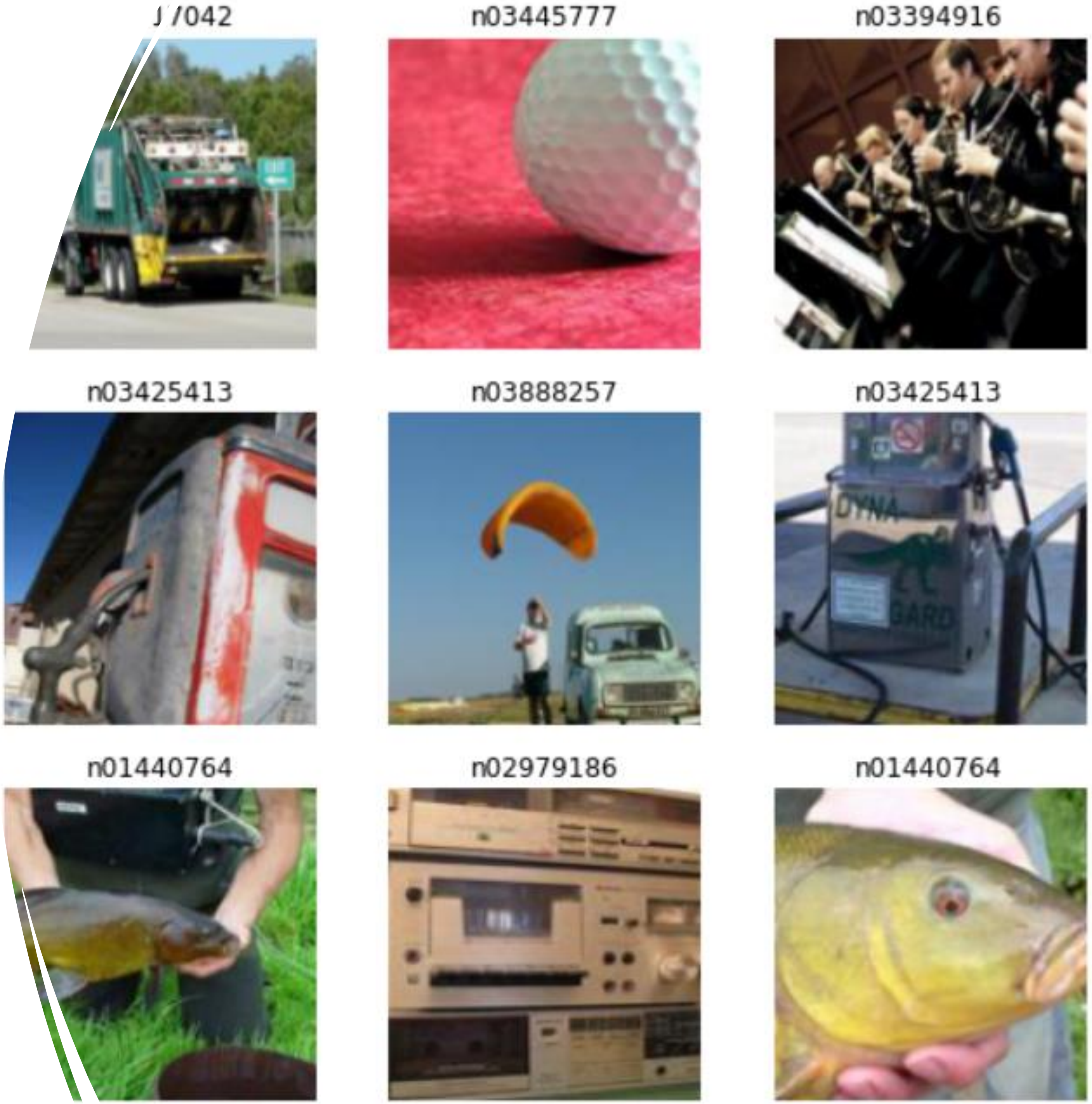


OneCycleLR(linear)

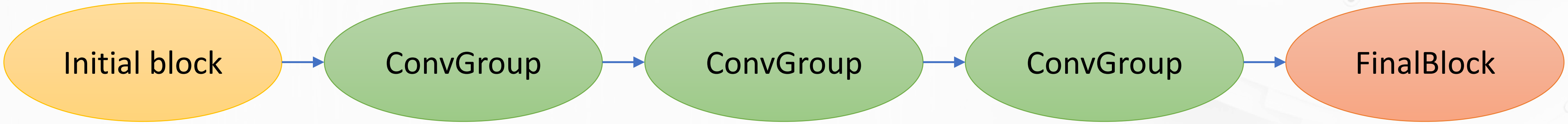


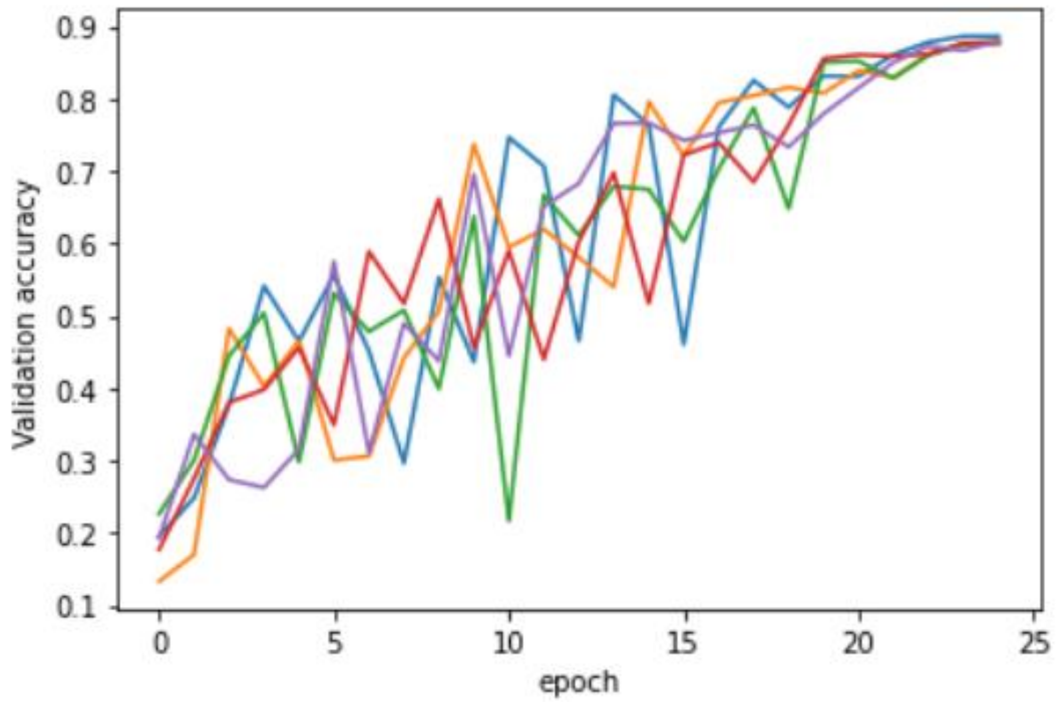
- High LR in the middle regularises the model and avoids overfitting
- Increase the LR faster → better for small datasets (Cifar10, ImageNette)

Investigating the validity of different techniques



Base network architecture



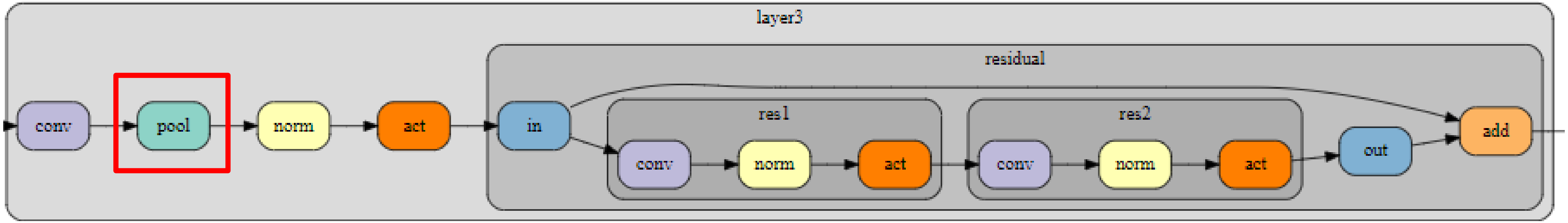


Base architecture

- Runtime: 34.9s
- Validation accuracy: 0.879
- (averaged over 5 runs)

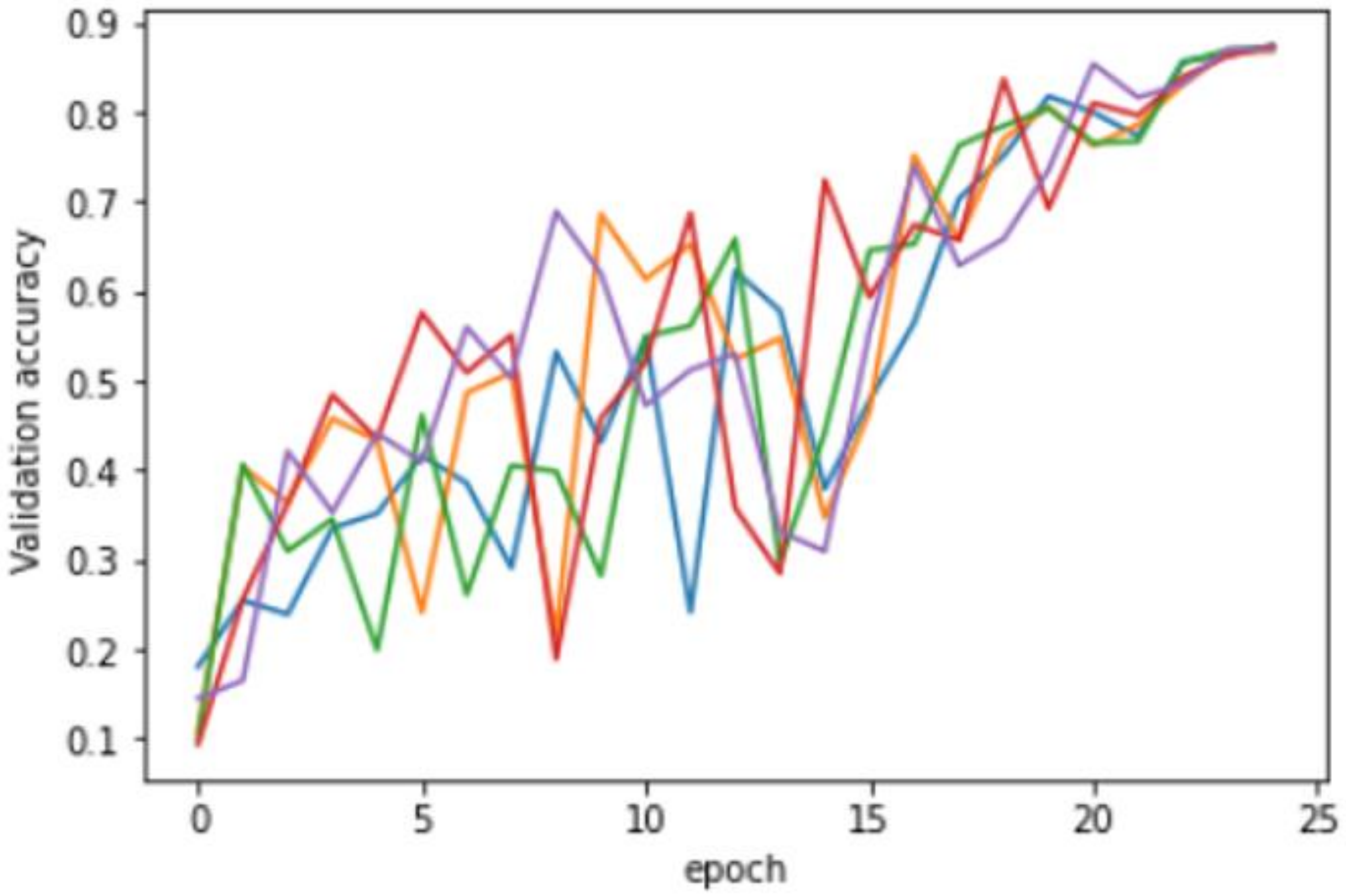
Changing ConvGroup

- ***Preactivation architecture:*** ***S***wapping activation function and max pooling
- **Mitigating variance shift:** Swapping max pooling and batch-normalization

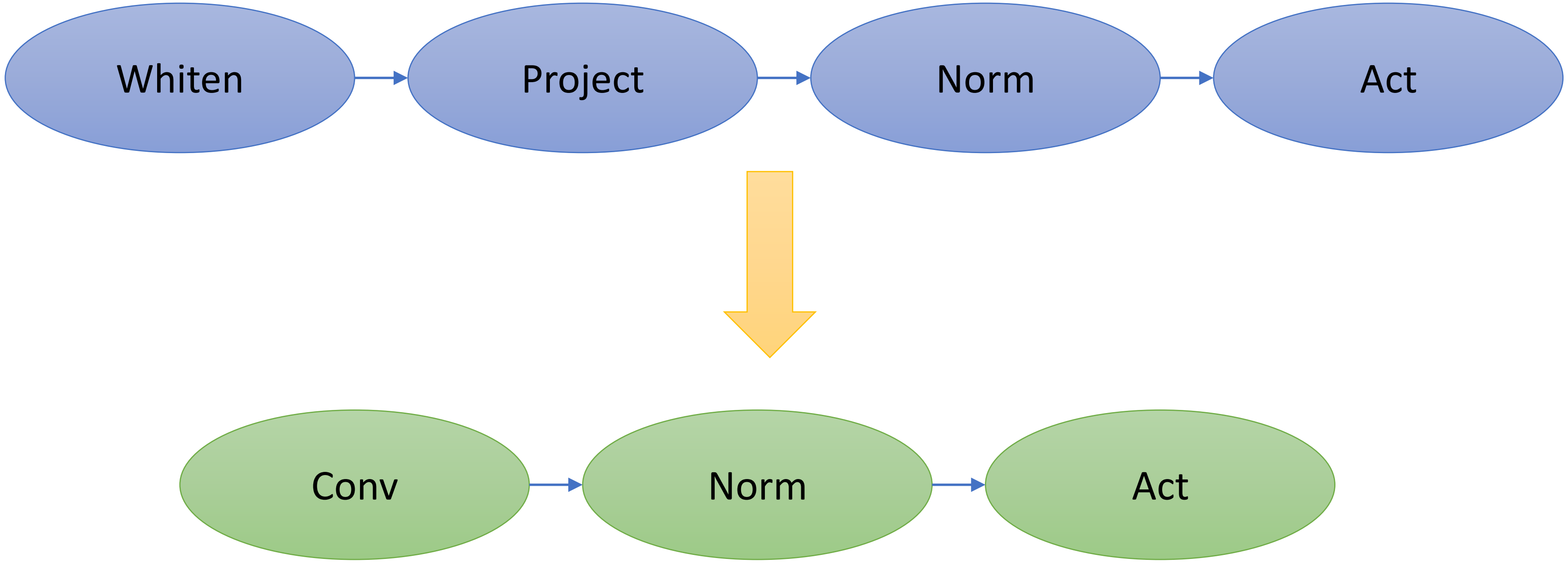


Changing ConvGroup

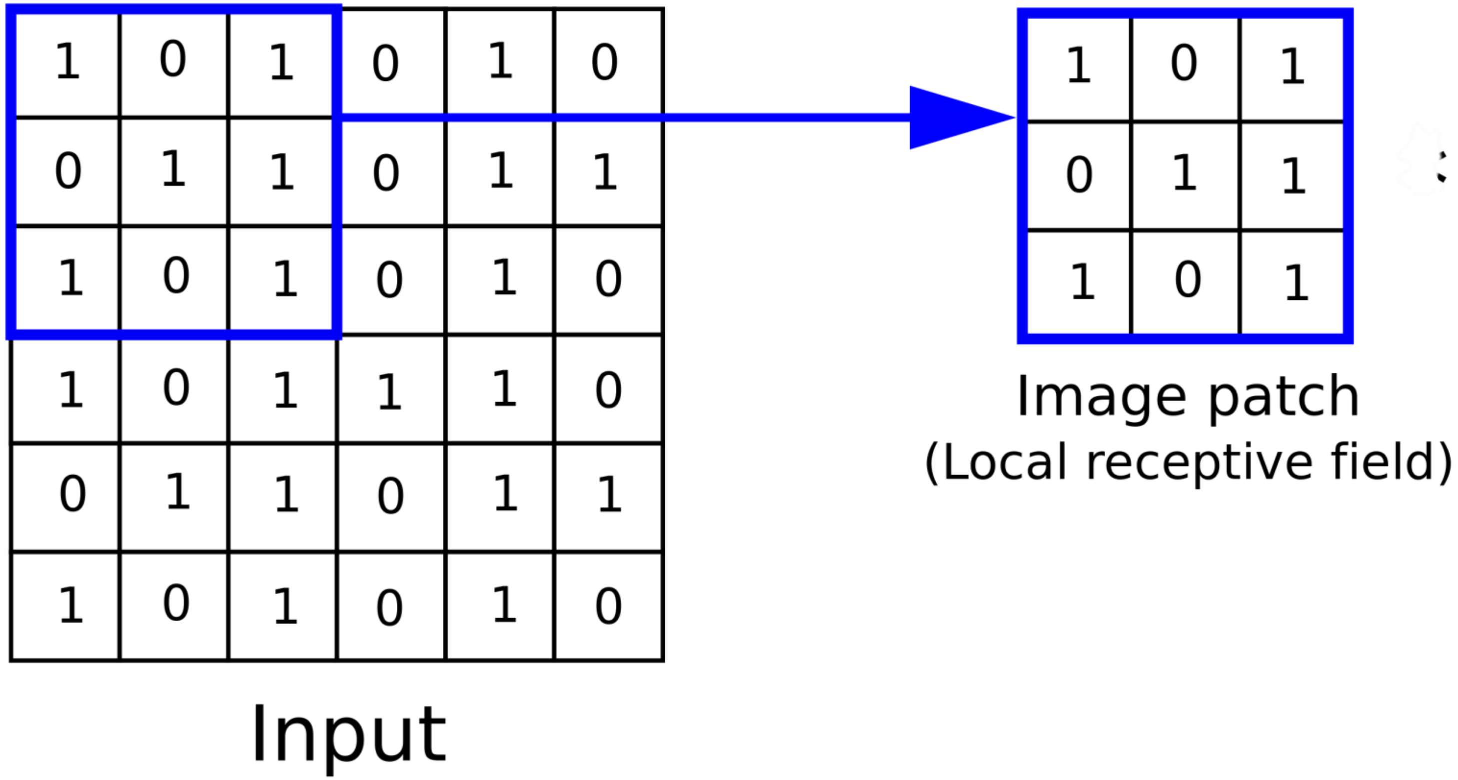
- Runtime: 41.8s
- Validation accuracy: 0.871
- (averaged over 5 runs)

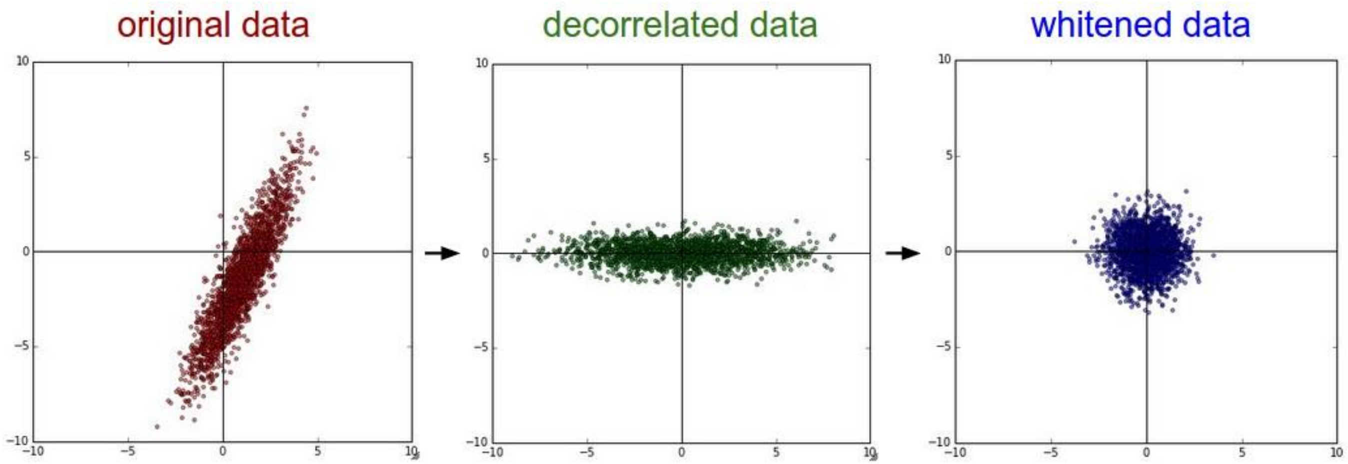


Removing Whitening



Input whitening in convolutional networks

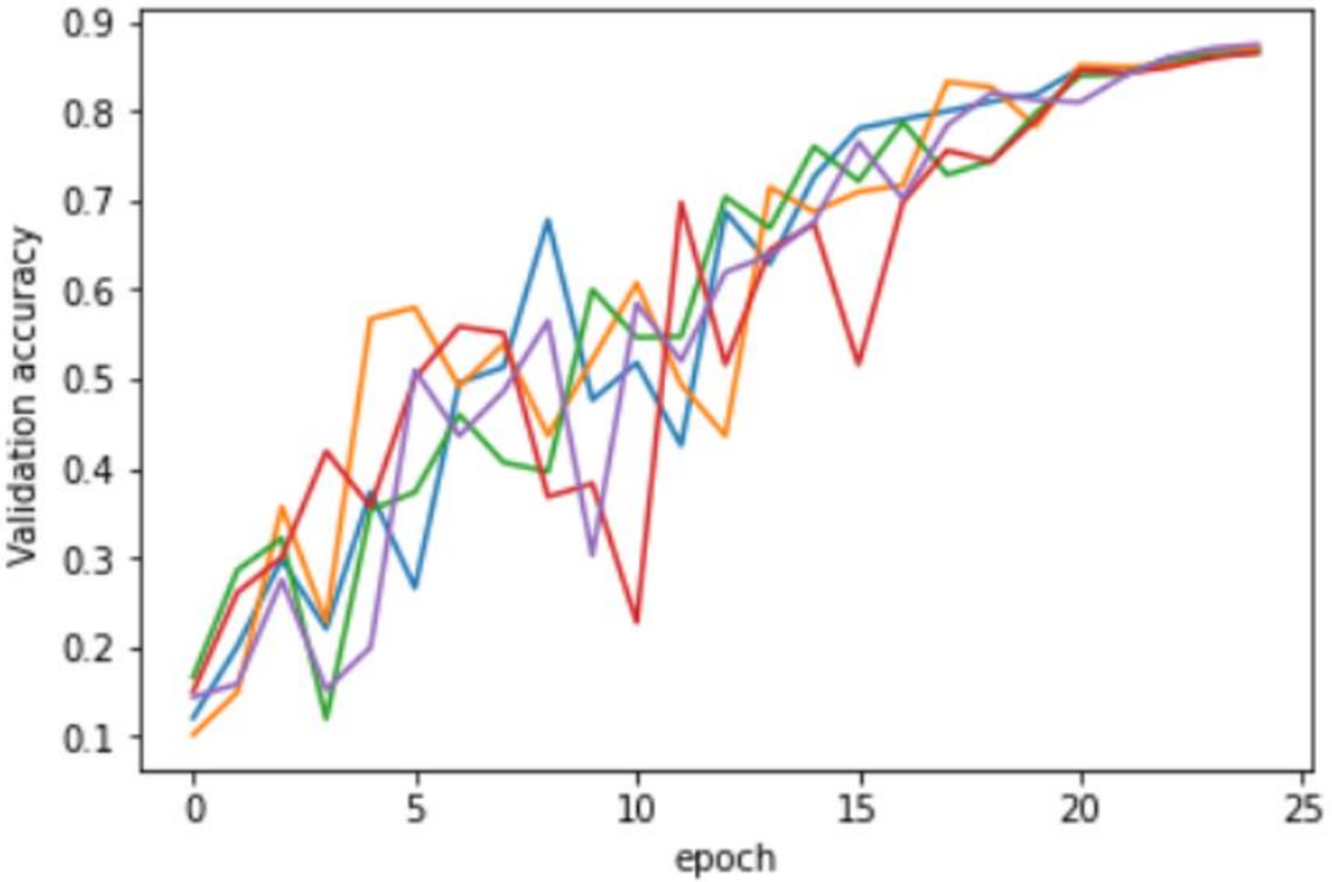




Input whitening in convolutional networks

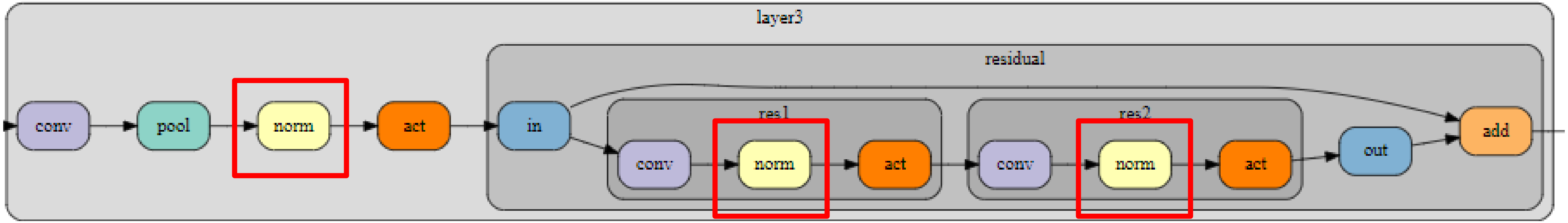
Removing whitening

- Runtime: 40.3s
- Validation accuracy: 0.869
- (averaged over 5 runs)



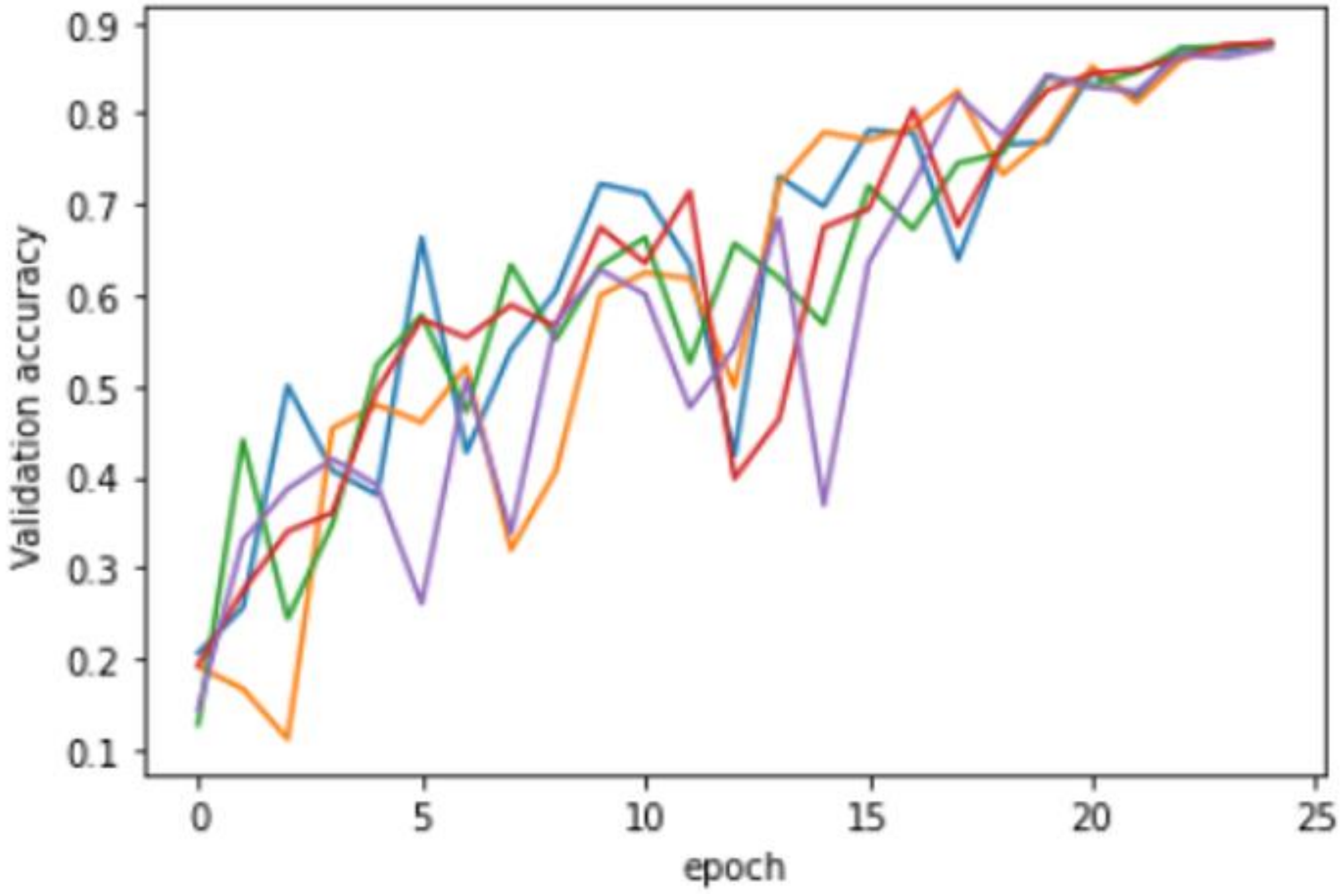
Ghost Batch Normalization

- ***Adding randomness in the calculation of batch statistics***
- ***Computationally faster in a distributed setting***



Ghost Batch Normalization

- Runtime: 38.1s
- Validation accuracy: 0.875
- (averaged over 5 runs)



Conclusion

https://github.com/LouisCaubet/hlb-imagenette