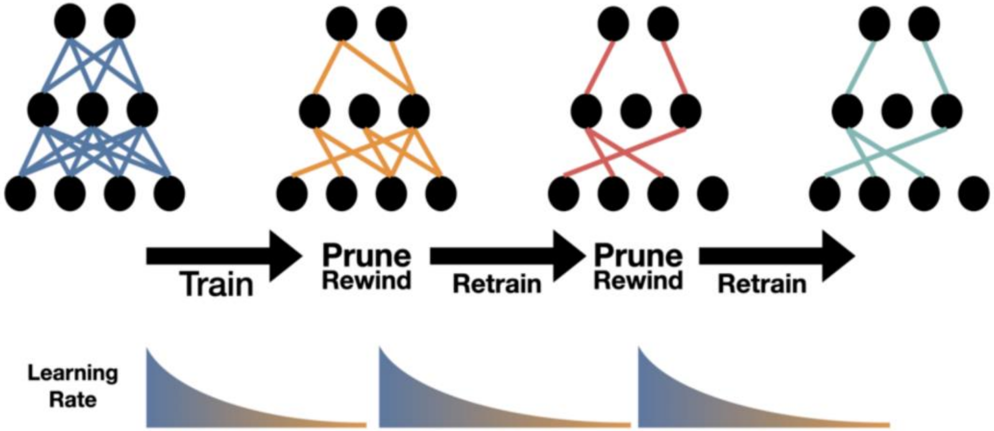
Patricia Gschoßmann

Model Pruning

Standard Pruning

How to?

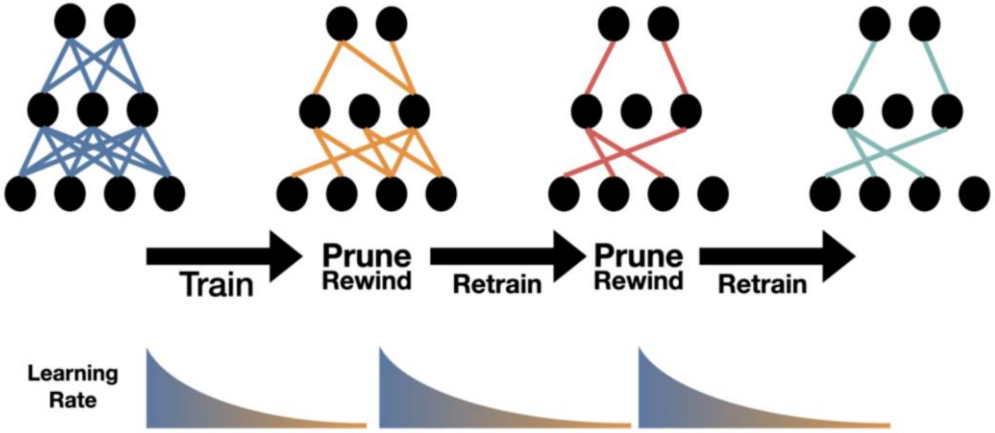


https://news.mit.edu/2020/foolproof-way-shrink-deep-learning-models-0430

Standard Pruning

How to?

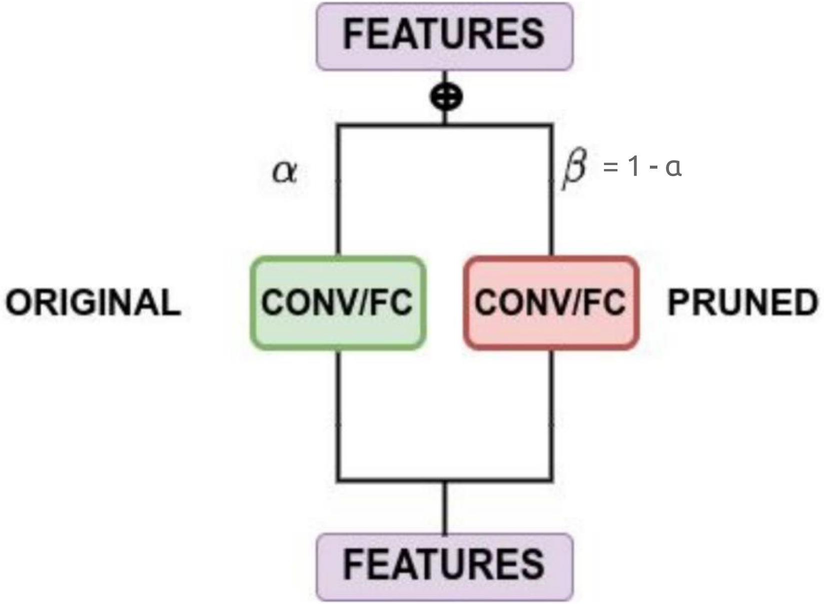
How can you speed up the process?



https://news.mit.edu/2020/foolproof-way-shrink-deep-learning-models-0430

New Approach

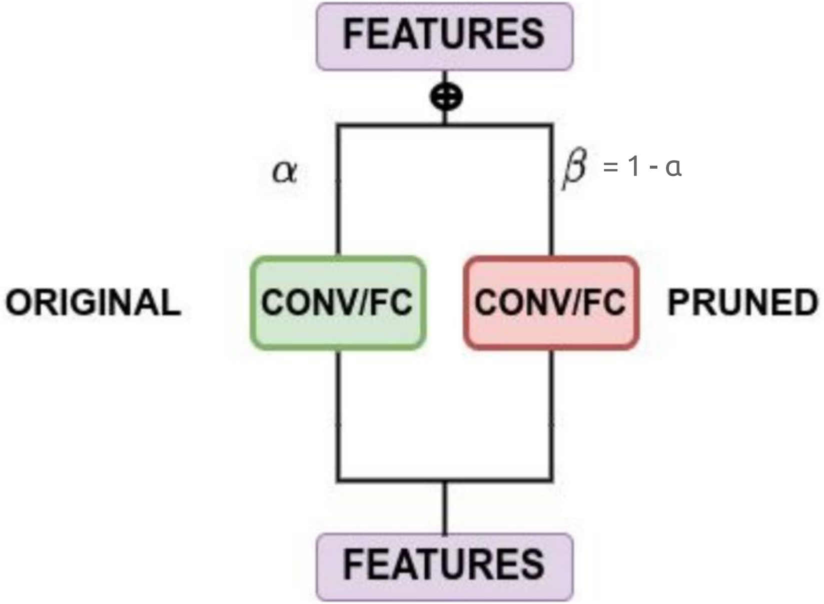
How to?



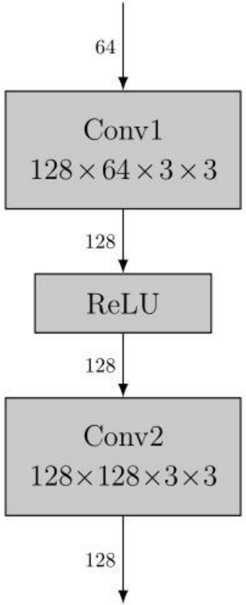
New Approach

How to?

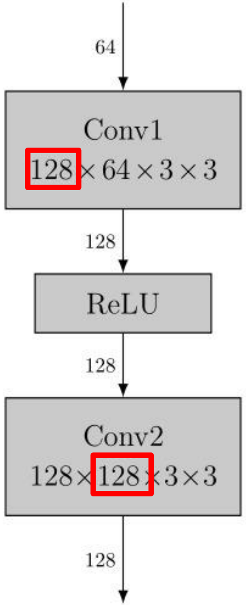
Advantage: Start at e.g. 60%



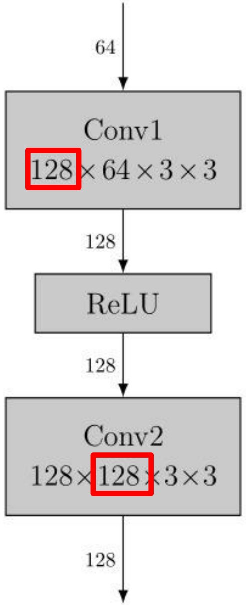
Original model



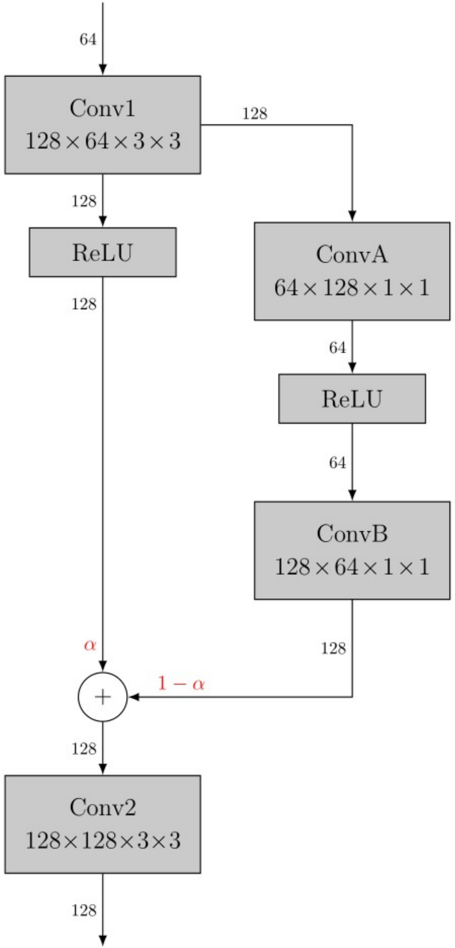
Original model



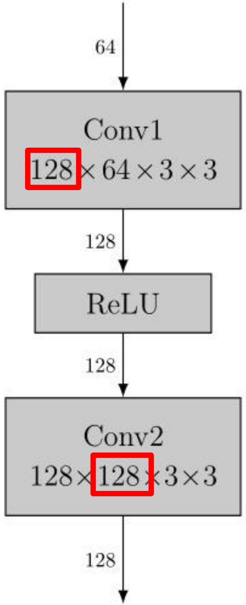
Original model



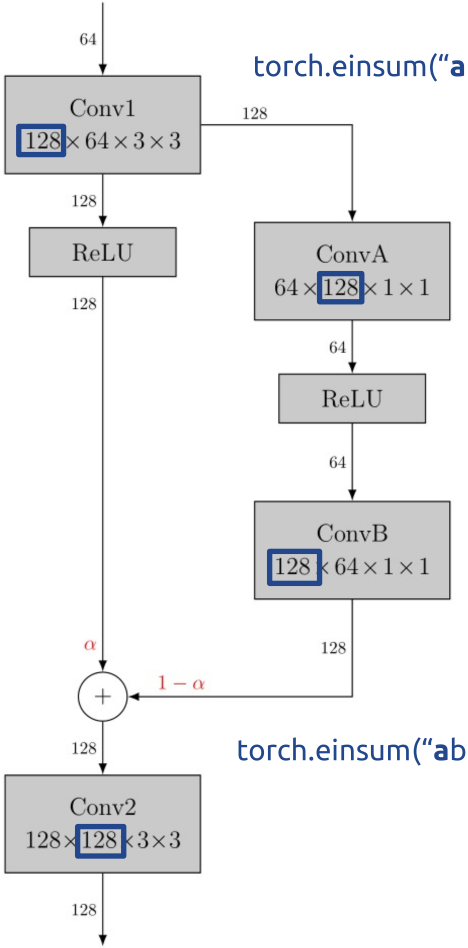
With parallel branches



Original model



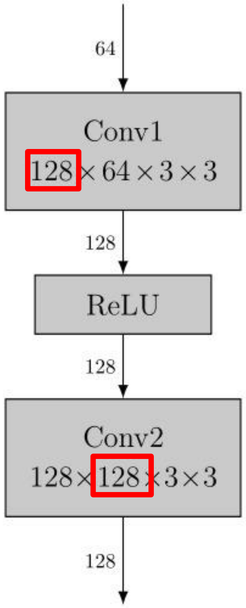
With parallel branches



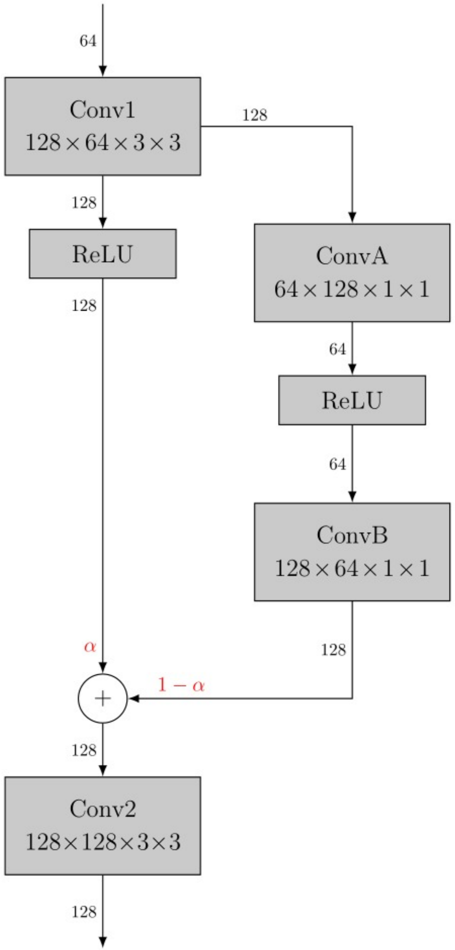
torch.einsum(“**a**bcd, e**a**fg -> ebcd”, conv1_weights, convA_weights)

torch.einsum(“**a**bcd, e**a**fg -> ebfg”, convB_weights, conv2_weights)

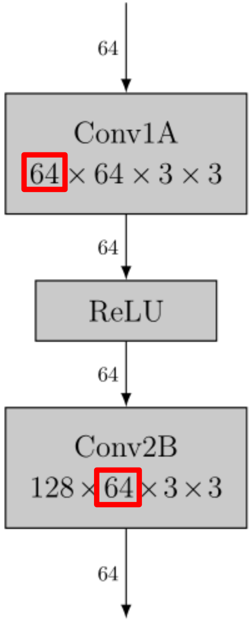
Original model



With parallel branches



Pruned model



Challenges

α-schedule

- Type of decay (step, exponential, multiplicative, ...)
- Decay rate

Pretrained model

Should the weights of the pretrained model be freezed or updated?

Learning rate

- How big/small?
- Static or dynamic?
- Schedulers?

VGG16

Original performance:

- **Training accuracy: 100%**
- **Validation accuracy: 92%, Test accuracy: 91.3%**

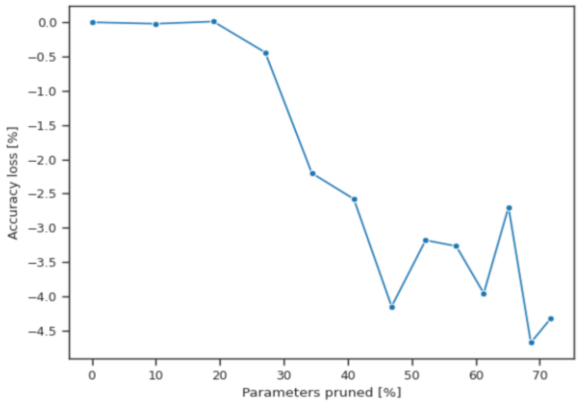
VGG16

Original performance:

- **Training accuracy: 100%**
- **Validation accuracy: 92%, Test accuracy: 91.3%**

Standard pruning results:

- **Validation accuracy: 90%**
- **Pruned model size: ≈ 35% of original**



VGG16

Original performance:

○ **Training accuracy: 100%**

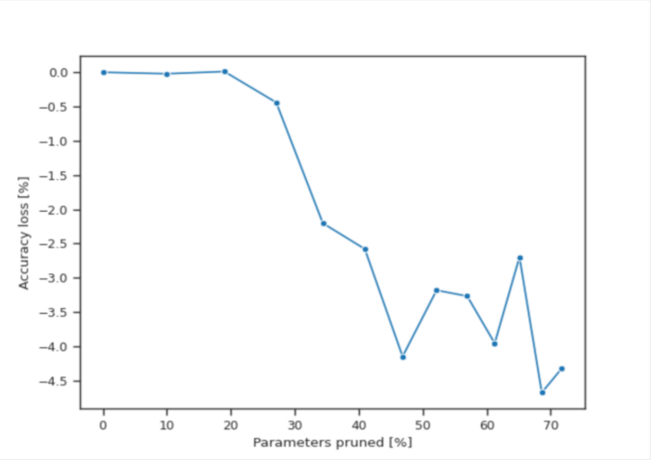
○ **Validation accuracy: 92%, Test accuracy: 91.3%**

Standard pruning results:

○ **Validation accuracy: 90%**

○ **Pruned model size: ≈ 35% of original**

⇨ Goal: **Prune 65% at once**



VGG16: Results

VGG16: Results

Failed attempts/mistakes:

- **Freeze pretrained + reduceLrOnPlateau-scheduler + constant initial LR (0.001)**
- **Step decay/Exp. decay**

VGG16: Results

Failed attempts/mistakes:

- **Freeze pretrained + reduceLrOnPlateau-scheduler + constant initial LR (0.001)**
- **Step decay/Exp. decay**

Final setup:

- **Update pretrained + step decay (-0.1) + reduceLrOnPlateau-scheduler**
- **At start: Initial LR of 0.001, at end: Repeated increase of initial LR x10**

VGG16: Results

Failed attempts/mistakes:

- **Freeze pretrained + reduceLrOnPlateau-scheduler + constant initial LR (0.001)**
- **Step decay/Exp. decay**

Final setup:

- **Update pretrained + step decay (-0.1) + reduceLrOnPlateau-scheduler**
- **At start: Initial LR of 0.001, at end: Repeated increase of initial LR x10**

Results:

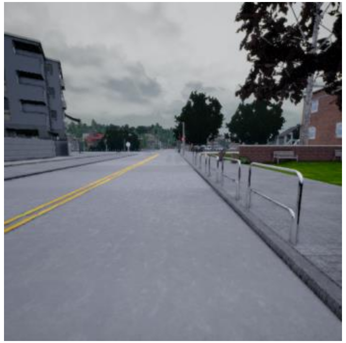
- **Validation accuracy: 88.5%**
- **Parameters remaining: ≈ 35%**

	Original	Pruned
Time/img (GPU)	≈0.0531s	≈0.0529s
GPU VRAM	1.53GB	0.53GB
Time/img (CPU)	≈0.0409s	≈0.0257s
RAM consumption	3.64GB	2.32GB

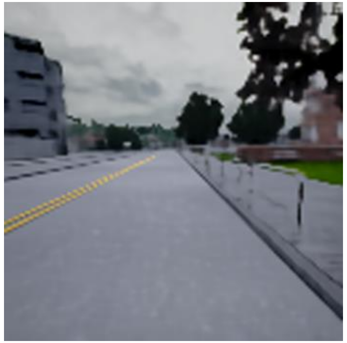
RGB Autoencoder

Original performance (input of size 3x128x128, bottleneck of size 128x16x16):

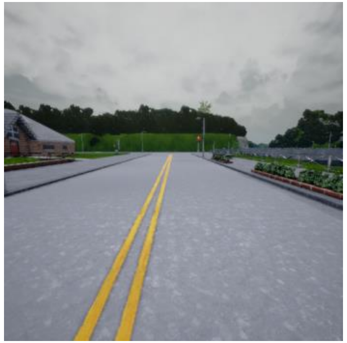
- **Training loss: 1.14**
- **Validation loss: 1.40, Test loss: 1.41**



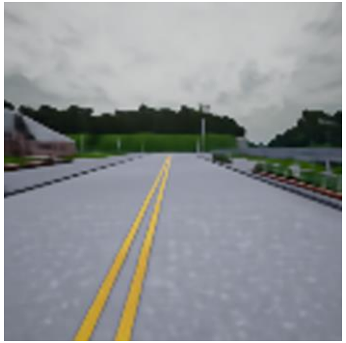
Original



Reconstructed



Original

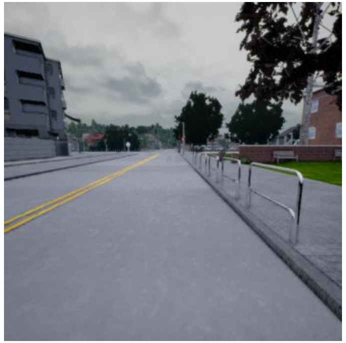


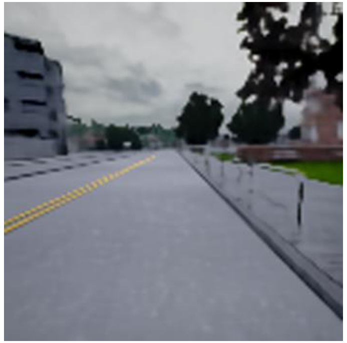
Reconstructed

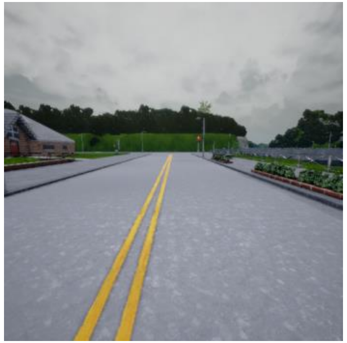
RGB Autoencoder

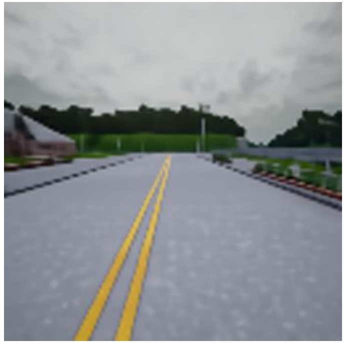
Original performance (input of size 3x128x128, bottleneck of size 128x16x16):

- ○ **Training loss: 1.14**
- ○ **Validation loss: 1.40, Test loss: 1.41**


Original


Reconstructed


Original


Reconstructed

⇨ Goal: **Prune 65% at once – bottleneck of size 44x16x16**

RGB Autoencoder: Approach and Results

RGB Autoencoder: Approach and Results

Possible approaches:

- **Prune encoder – then decoder**
- **Prune en- and decoder simultaneously**

RGB Autoencoder: Approach and Results

Possible approaches:

- **Prune encoder – then decoder**
- **Prune en- and decoder simultaneously**

Setup: **Same setup as with VGG**

RGB Autoencoder: Approach and Results

Possible approaches:

- ○ **Prune encoder – then decoder**
- ○ **Prune en- and decoder simultaneously**

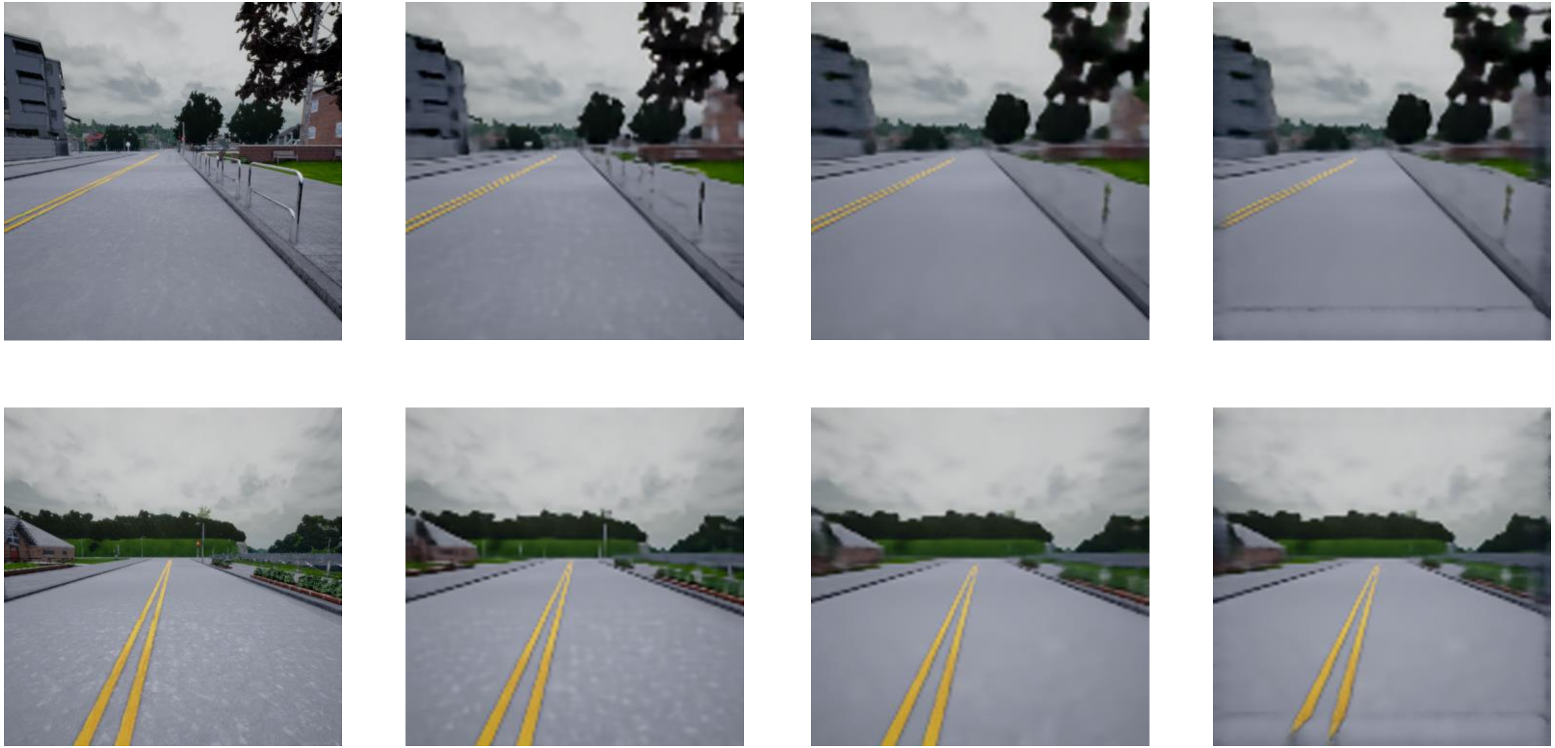
Setup: **Same setup as with VGG**

Results – Prune encoder:

- **Training loss: 2.31**
- **Validation loss: 2.59**
- **Test loss: 2.65**

	Original	**Pruned**
Time/img (GPU)	≈0.0573s	≈0.0582s
GPU VRAM	6.83GB	5.23GB
Time/img (CPU)	≈0.0891s	≈0.0836s
RAM consumption	7.97GB	7.89GB

RGB Autoencoder: Results – Prune Encoder



Img. 1

Img. 2

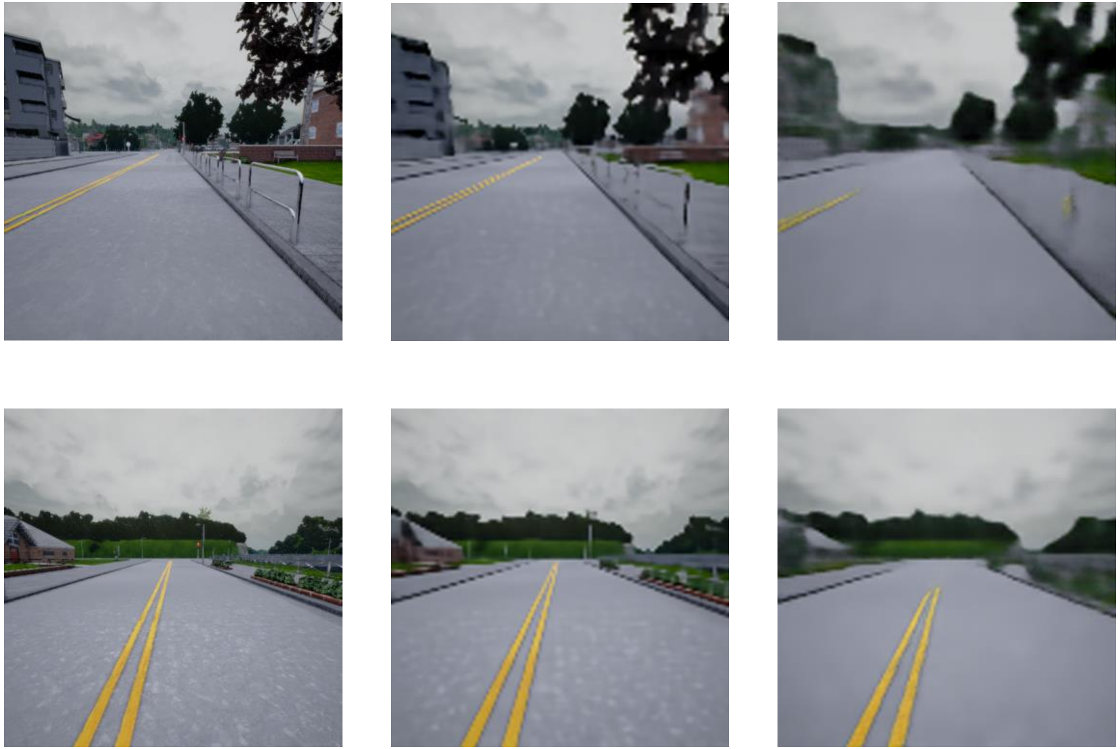
Original

Reconstructed
(before pruning)

Reconstructed
(after pruning,
before multiplication)

Reconstructed
(after pruning,
after multiplication)

RGB Autoencoder: Intermediate Results – Prune Both



Img. 1

Img. 2

Original

Reconstructed (before pruning)

Reconstructed (after pruning, before multiplication)

Conclusion

α-schedule

- Type of decay (step, exponential, multiplicative, ...)
- Decay rate

Pretrained model

Should the weights of the pretrained model be freezed or updated?

Learning rate

- How big/small?
- Static or dynamic?
- Schedulers?

Conclusion

α-schedule

- Type of decay (step, exponential, multiplicative, ...)
- Decay rate

Open question

Pretrained model

Should the weights of the pretrained model be freezed or updated?

Learning rate

- How big/small?
- Static or dynamic?
- Schedulers?

Conclusion

α-schedule

- Type of decay (step, exponential, multiplicative, ...)
- Decay rate

Open question

Pretrained model

Should the weights of the pretrained model be freezed or updated?

Update weights

Learning rate

- How big/small?
- Static or dynamic?
- Schedulers?

Conclusion

α-schedule

- Type of decay (step, exponential, multiplicative, ...)
- Decay rate

Open question

Pretrained model

Should the weights of the pretrained model be freezed or updated?

Update weights

Learning rate

- How big/small?
- Static or dynamic?
- Schedulers?

Initial LR of 0.001
Increase at local minima

Future Work

- Prune autoencoder decoder
- Compare results of both autoencoder approaches
- …
- Prune autoencoder with skip connections
- In general: Test new pruning approach with a larger α-decay
- …
- Goal: Prune DCGAN

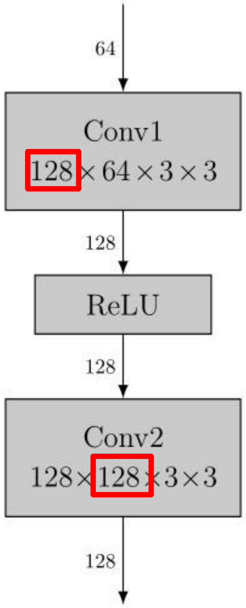
Thanks!

Does anyone have any questions?

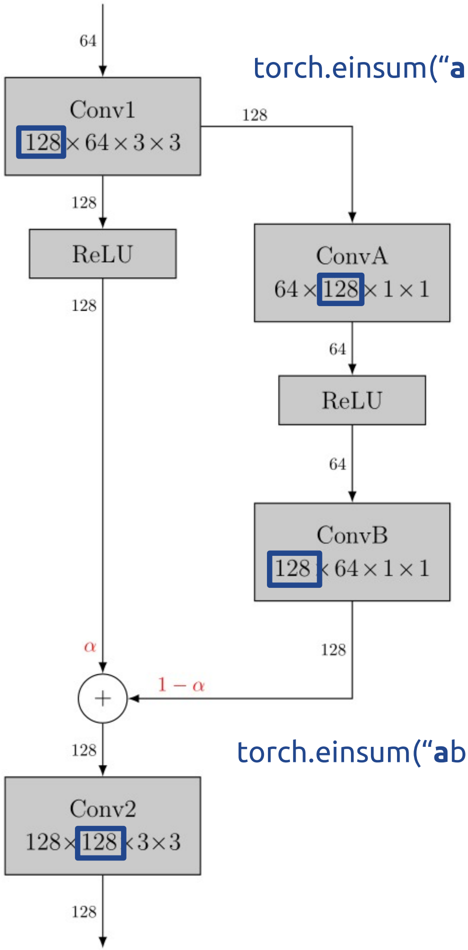
Sources

Presentation template by Slidesgo

Original model



With parallel branches



torch.einsum(“**a**bcd, e**a**fg -> ebcd”, conv1_weights, convA_weights)

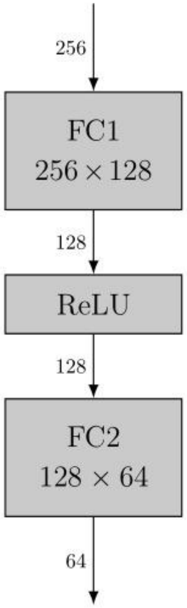
If bias:

W2(W1x + b1) + b2 = (W2W1)x + (W2b1 + b2)

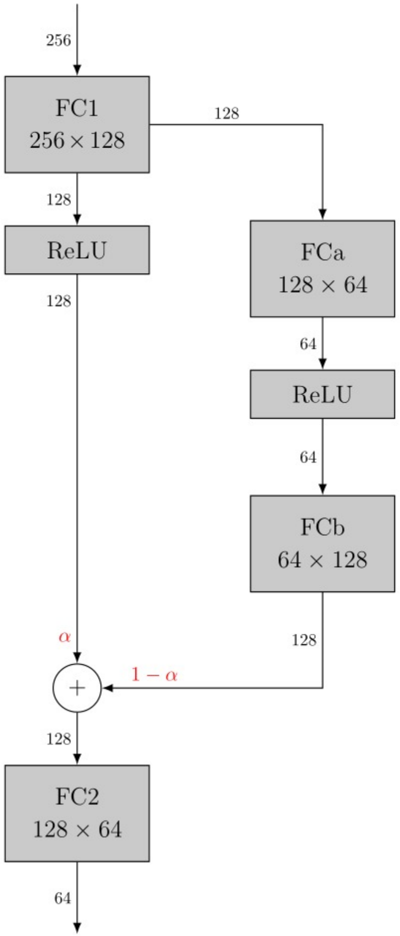
⬆ New weights ⬆ New bias

torch.einsum(“**a**bcd, e**a**fg -> ebfg”, convB_weights, conv2_weights)

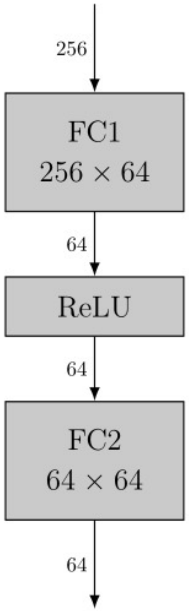
Original model



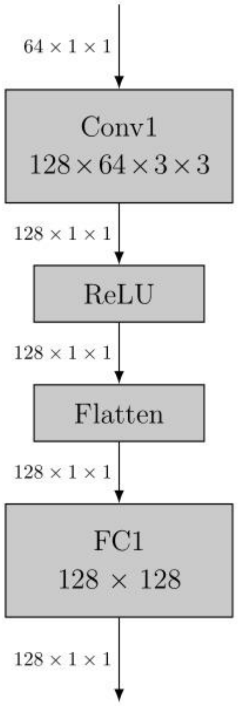
With parallel branches



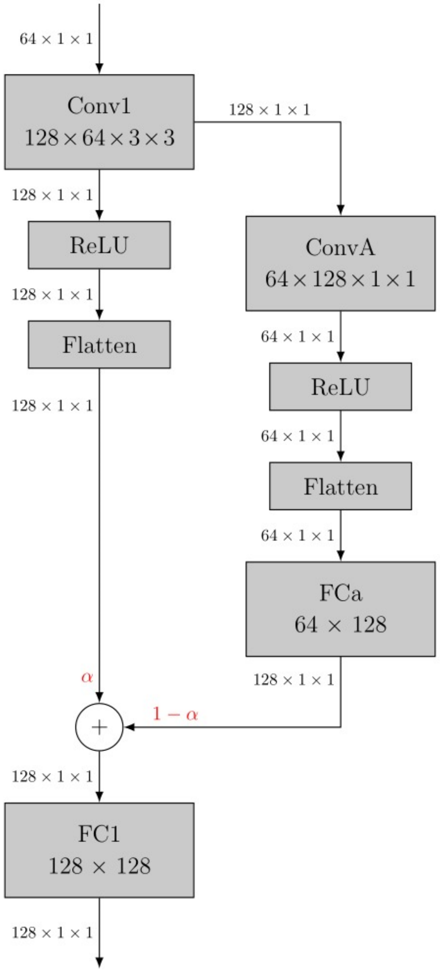
Pruned model



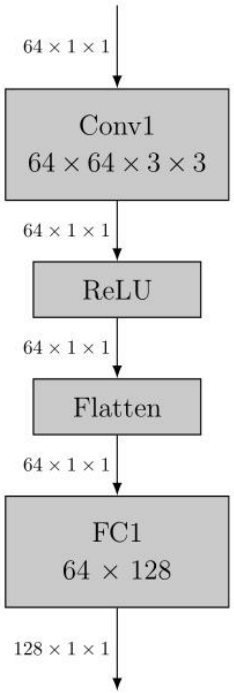
Original model



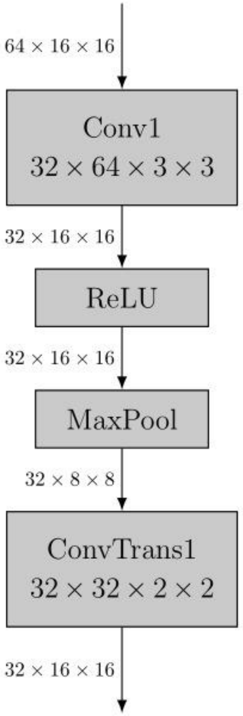
With parallel branches



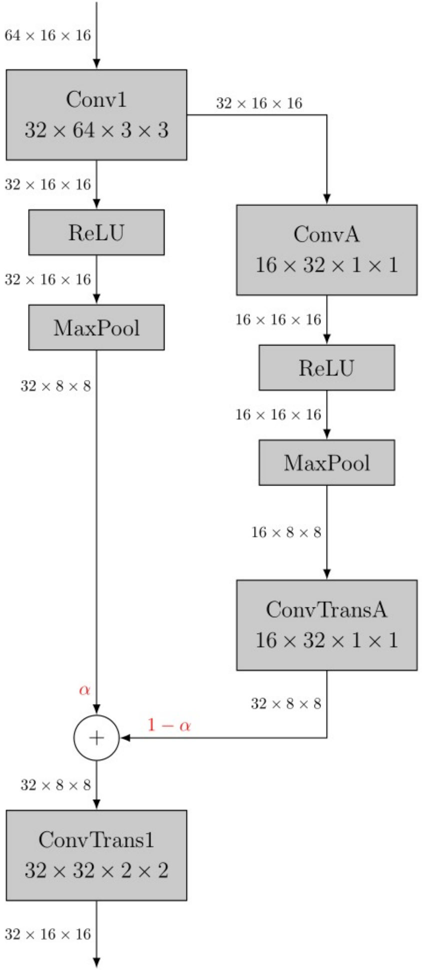
Pruned model



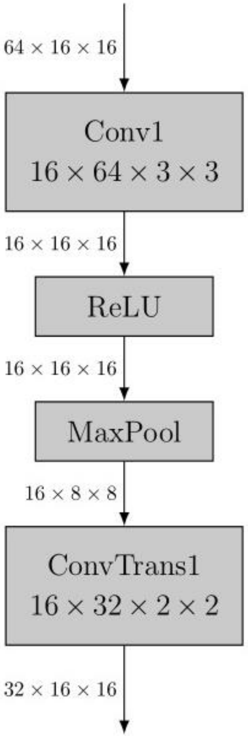
Original model



With parallel branches



Pruned model


64 × 16 × 16
Conv1
16 × 64 × 3 × 3
16 × 16 × 16
ReLU
16 × 16 × 16
MaxPool
16 × 8 × 8
ConvTrans1
16 × 32 × 2 × 2
32 × 16 × 16