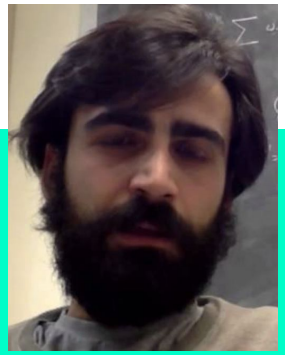
Finding The Needle In the Haystack

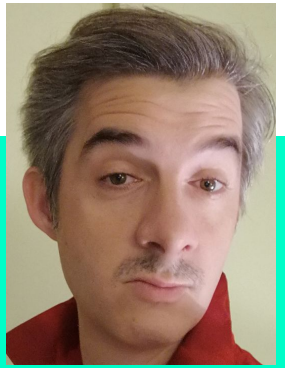
On The Benefits Of Architectural Constraints

Stéphane d'Ascoli



Levent Sagun

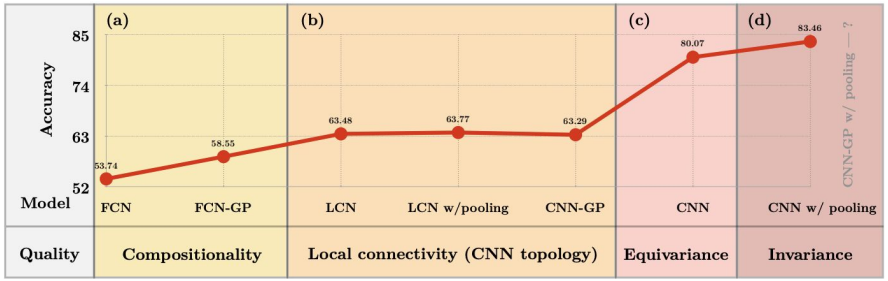
Joan Bruna

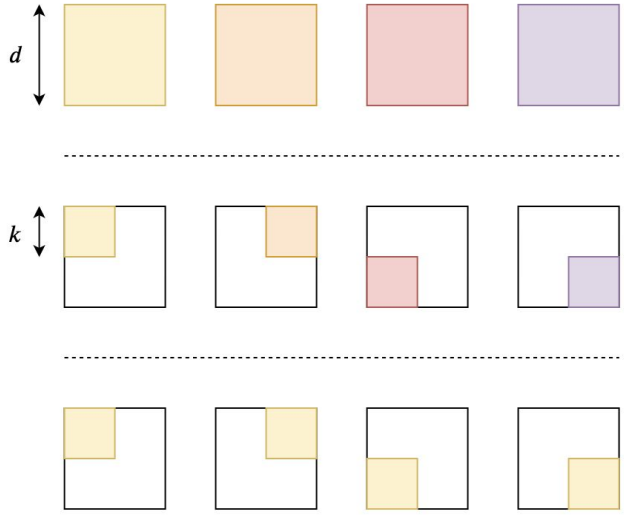


Giulio Biroli

FCN vs CNN

[Novak et al. 2019]





Fully connected :

$$N_{params} \sim N_{channels} d^4$$

Locally connected :

$$N_{params} \sim N_{channels} d^2 k^2$$

Convolutional :

$$N_{params} \sim N_{channels} k^2$$

CNNs are just FCNs which generalize better thanks to two constraints :

- Locality : smaller receptive field
- Weight sharing : filters use same weights

How do these constraints help navigate the loss landscape to find rare basins of high generalisation ?

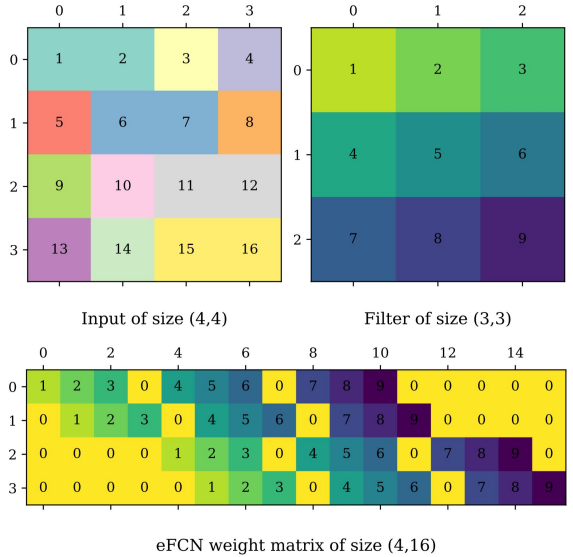
Can we use of their benefits (less parameters) without their potential disadvantages (loss of generality) ?

FCN vs CNN

One can always express a convolution using a matrix

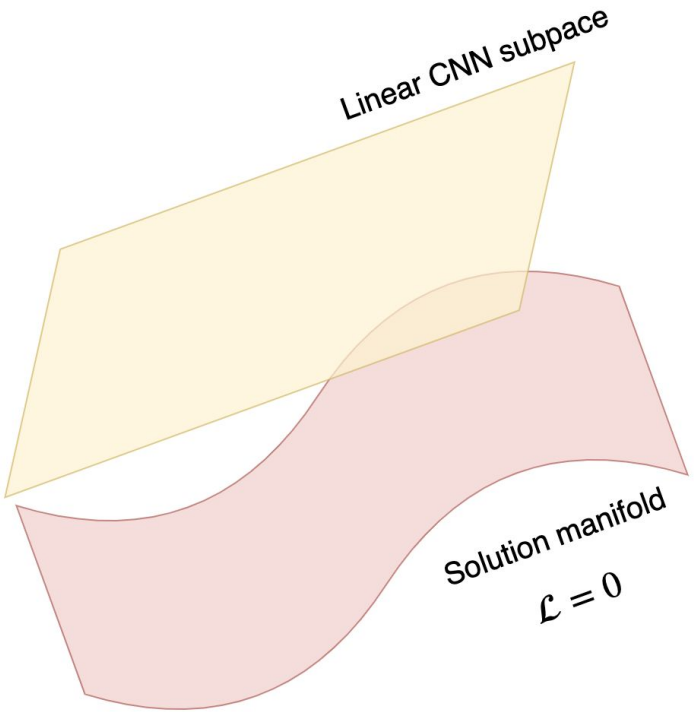
Hence, one can always map a CNN to its equivalent FCN (eFCN) or LCN (eLCN)

The eFCN / eLCN obtained are highly sparse and redundant



$$d_{in} = 4$$
$$(k, s, p) = (3, 1, 0)$$
$$d_{out} = \frac{d_{in} - 2p - k}{s} + 1 = 2$$

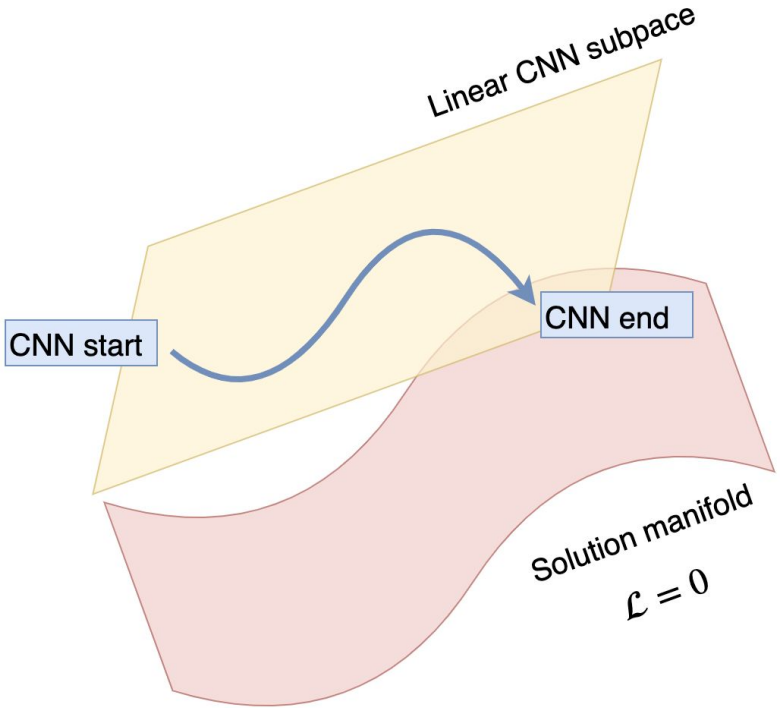
Embedding To Fully Connected Space



Build an embedding ϕ from a CNN to its eFCN that satisfies

$$f(\theta^{CNN}) = f(\Phi(\theta^{CNN})) = f(\theta^{eFCN})$$

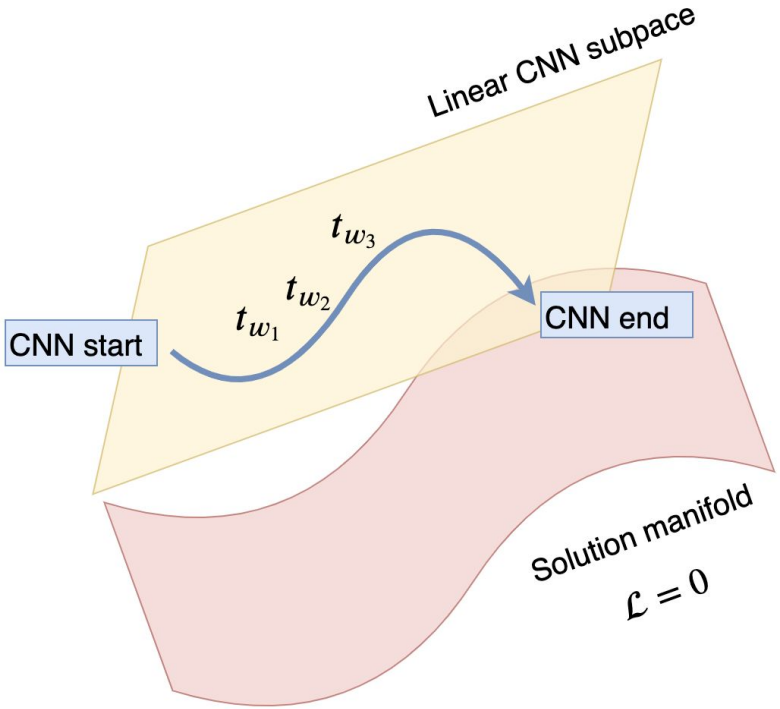
Our Experiment



A toy experiment :

1. Train a CNN until it reaches zero loss

Our Experiment

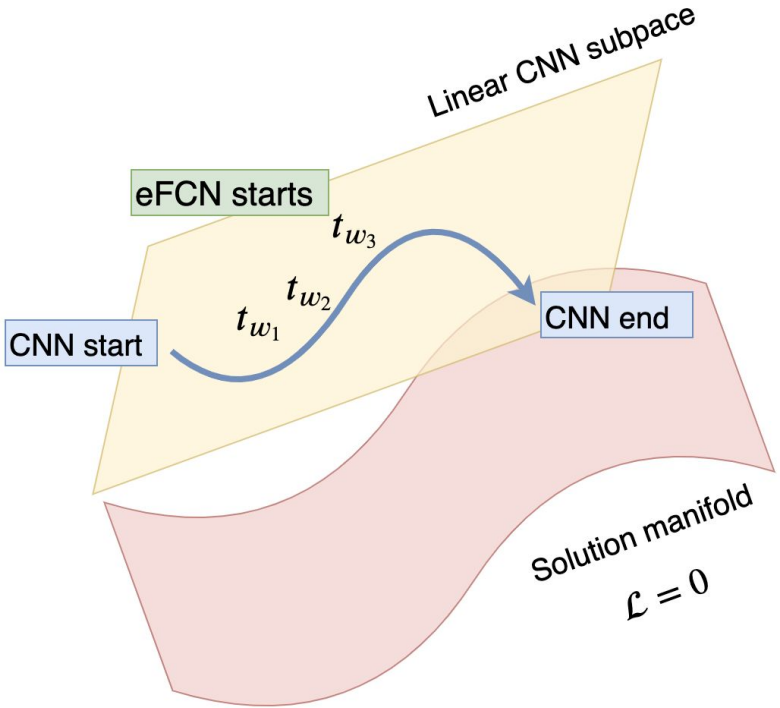


A toy experiment :

1. Train a CNN until it reaches zero loss

2. Along the training, save snapshots of the weights at various “switch” times

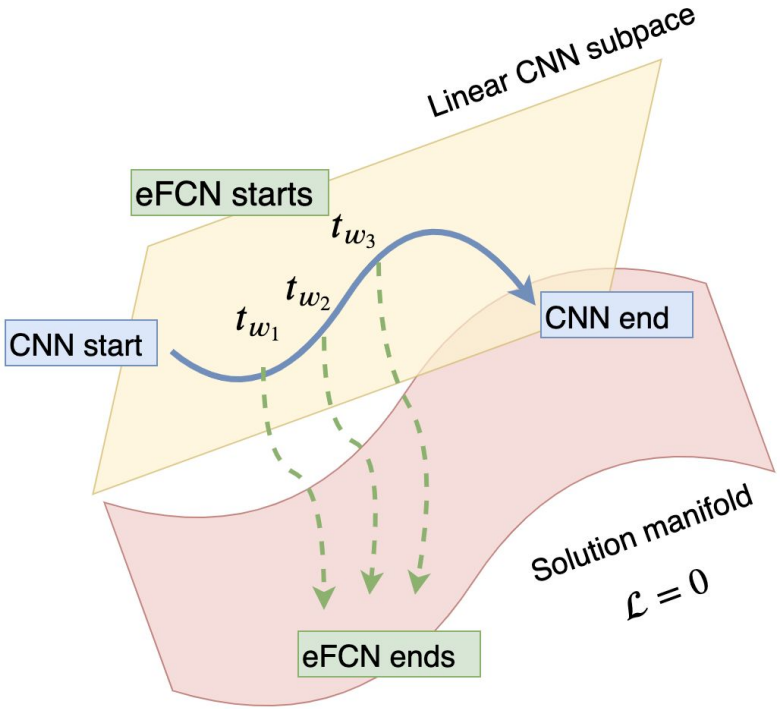
Our Experiment



A toy experiment :

1. Train a CNN until it reaches zero loss
2. Along the training, save snapshots of the weights at various “switch” times
3. Map these to eFCN space using ϕ

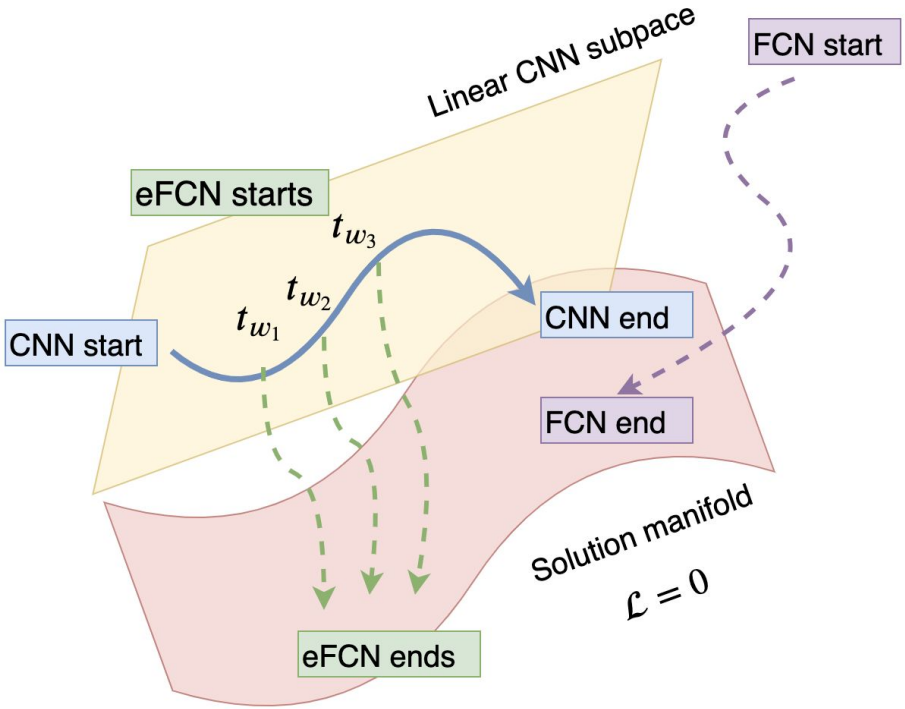
Our Experiment



A toy experiment :

1. Train a CNN until it reaches zero loss
2. Along the training, save snapshots of the weights at various “switch” times
3. Map these to eFCN space using ϕ
4. Train the eFCNs until they reach zero loss

Our Experiment



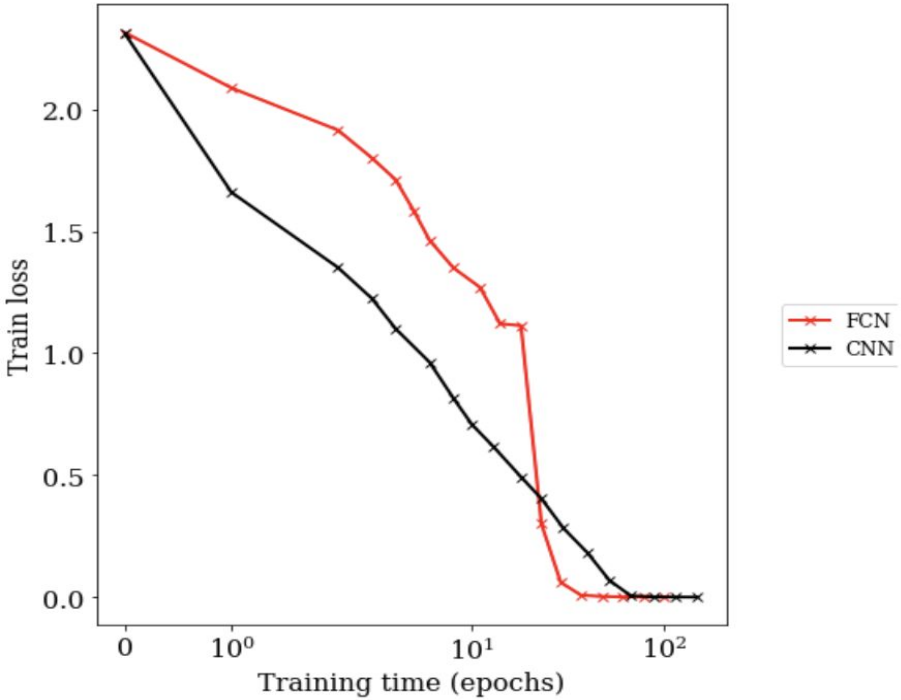
A toy experiment :

1. Train a CNN until it reaches zero loss
2. Along the training, save snapshots of the weights at various “switch” times
3. Map these to eFCN space using ϕ
4. Train the eFCNs until they reach zero loss
5. For comparison, train a vanilla FCN of same size

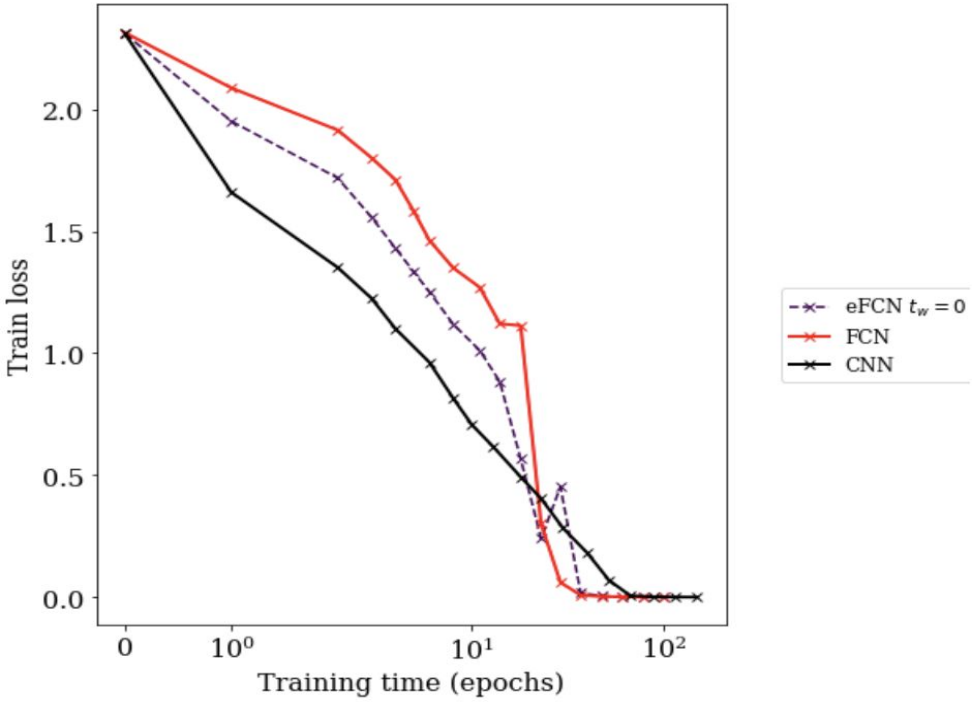
Experiment Details

- Models: VanillaCNN (3 convolutional layers, 64 channels) + AlexNet
- Memory overhead for AlexNet: 10MB → 1GB !
- Datasets: CIFAR-10 and CIFAR-100
- All layers apart from convolutional (dense, Dropout, Pooling, ReLU) are unchanged apart from reshaping
- PyTorch implementation open-sourced at github.com/sdascoli/anarchitectural-search

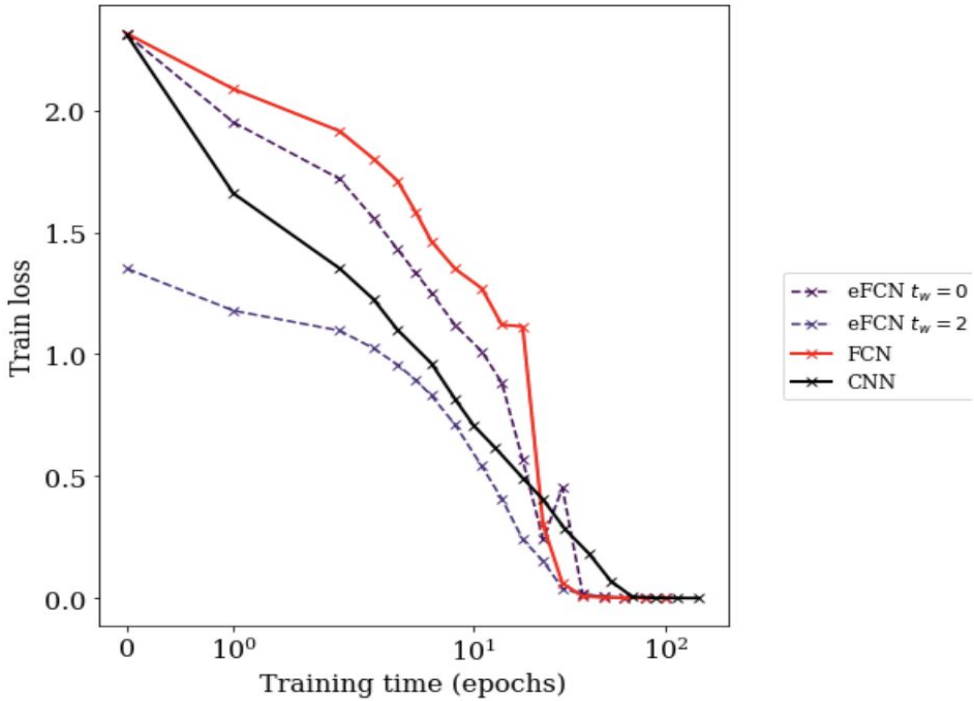
Training Dynamics



Training Dynamics

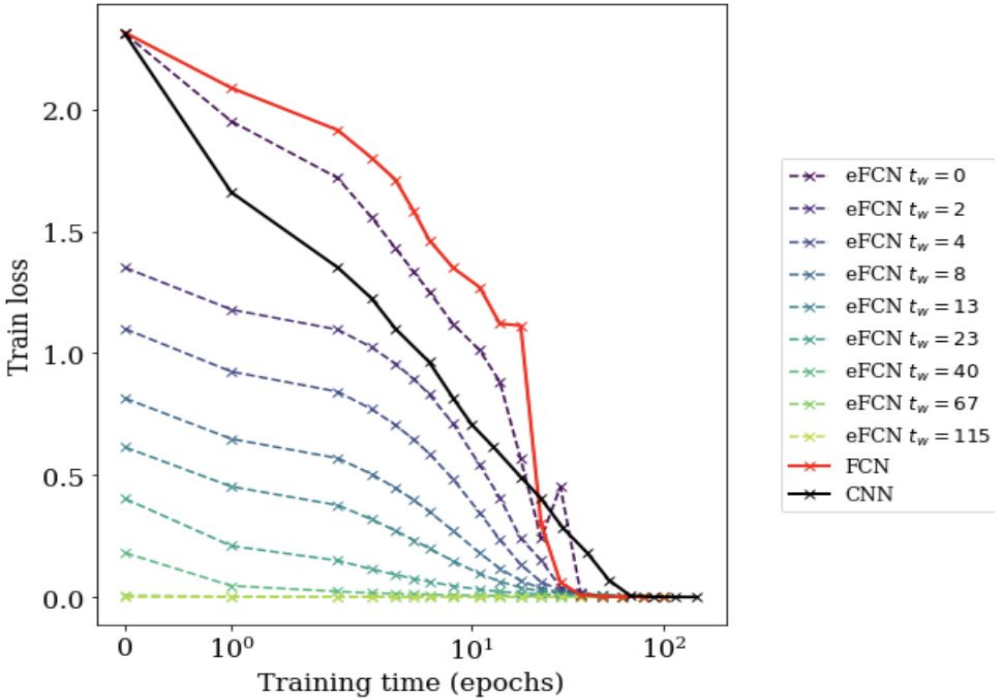


Training Dynamics



- Train loss keeps going down after relaxing constraints

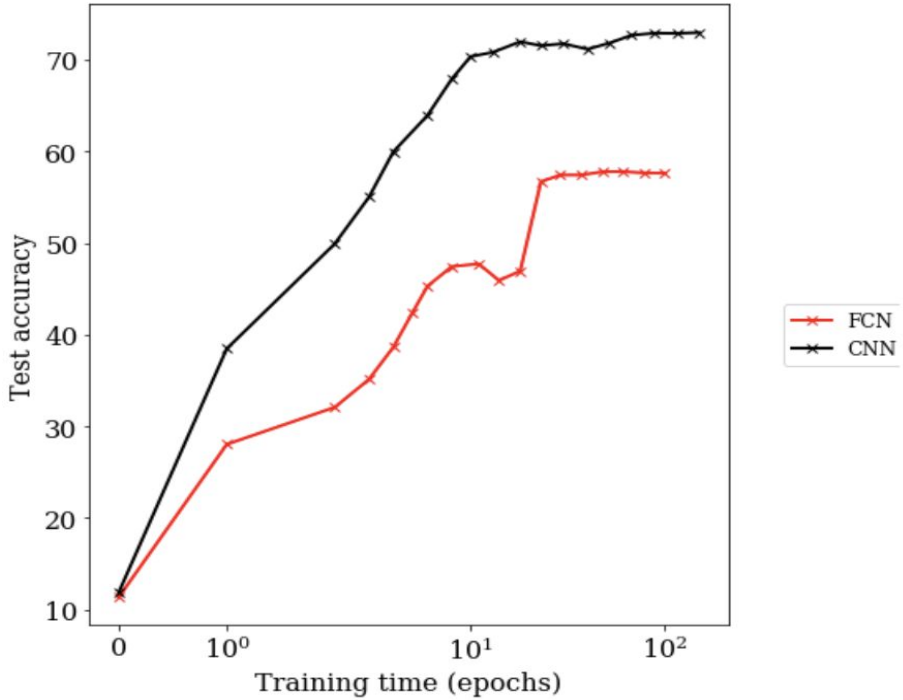
Training Dynamics



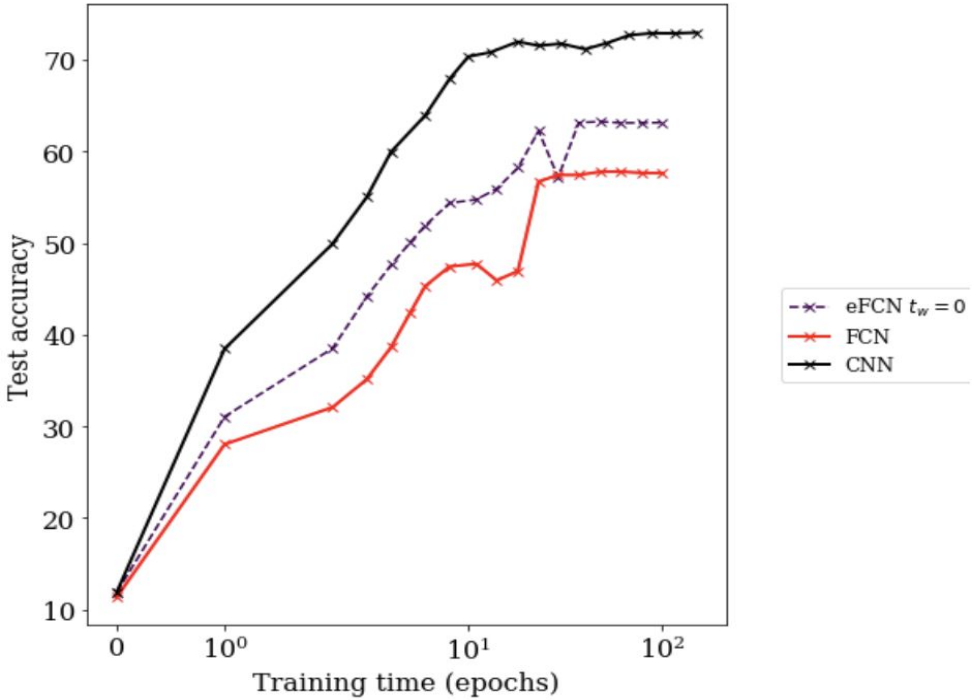
- Train loss keeps going down after relaxing constraints
- Even when we release close to convergence !

The eFCN stays in the basin reached by the CNN

Generalization Dynamics

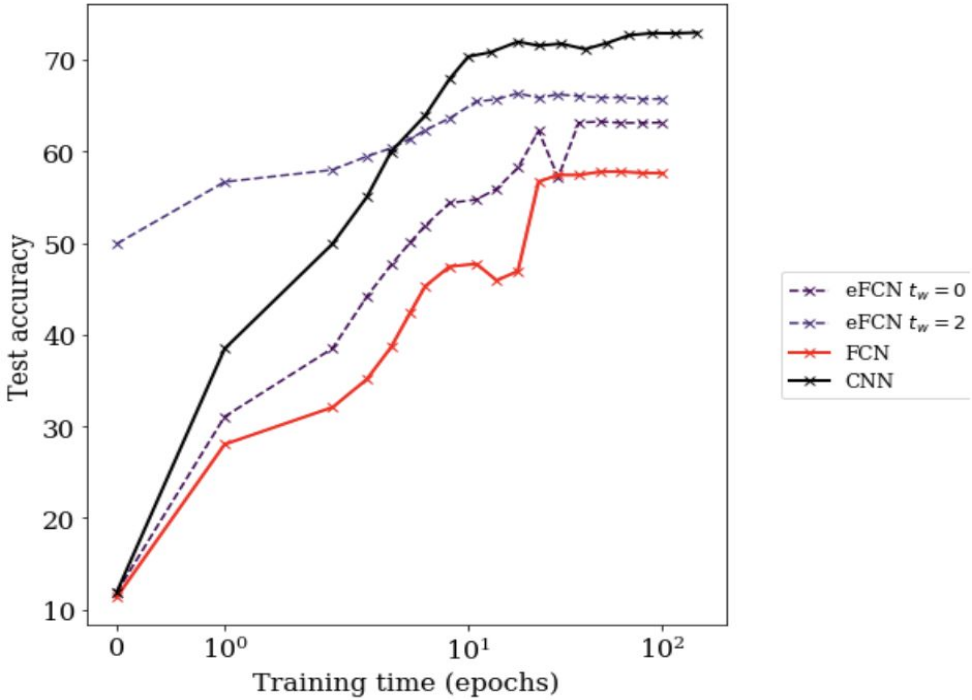


Generalization Dynamics



- FCN with CNN initialization beats vanilla FCN
  ⇨ CNN constraints start us off in a good region

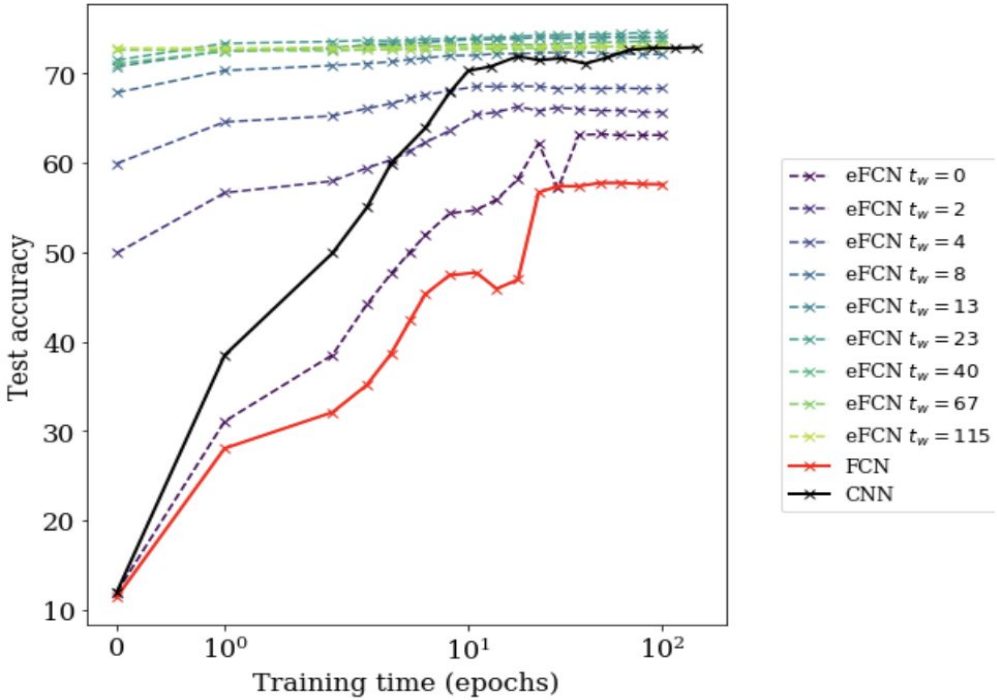
Generalization Dynamics



- FCN with CNN initialization beats vanilla FCN
  ⇨ CNN constraints start us off in a good region

- Test accuracy keeps going up after relaxing constraints
  ⇨ No overfitting

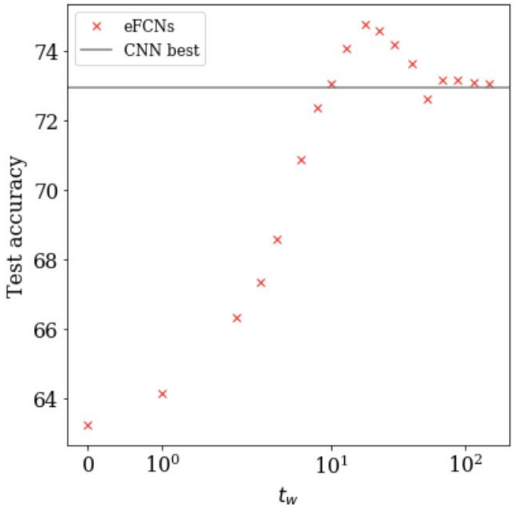
Generalization Dynamics



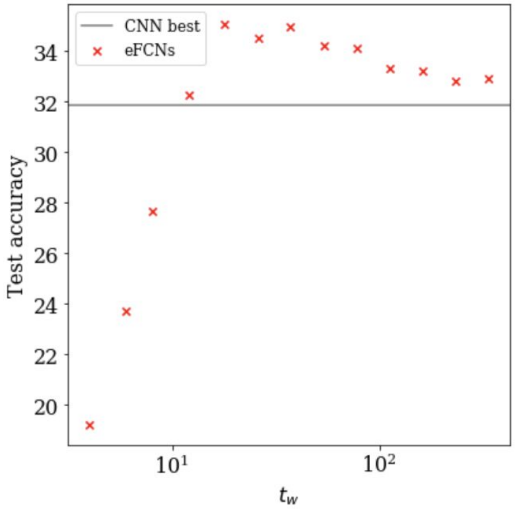
- FCN with CNN initialization beats vanilla FCN
  ⇨ CNN constraints start us off in a good region
- Test accuracy keeps going up after relaxing constraints
  ⇨ No overfitting
- When released late, the eFCNs even beat the CNN !
  ⇨ CNN constraints are mostly helpful at the beginning of optimization

Releasing architectural constraints once they have lead us to the right region of the landscape can be helpful !

What Is the Best Time To Relax The Constraints ?



VanillaCNN, CIFAR-10

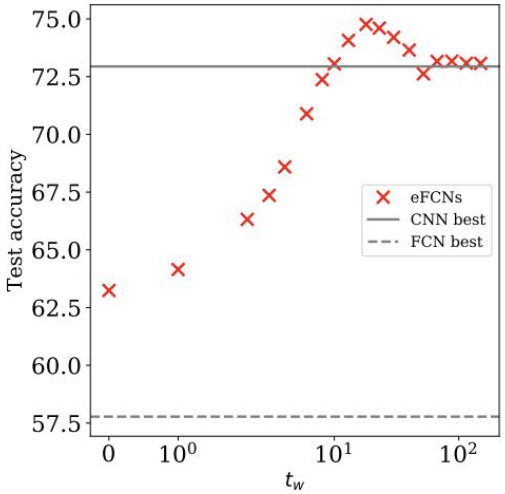


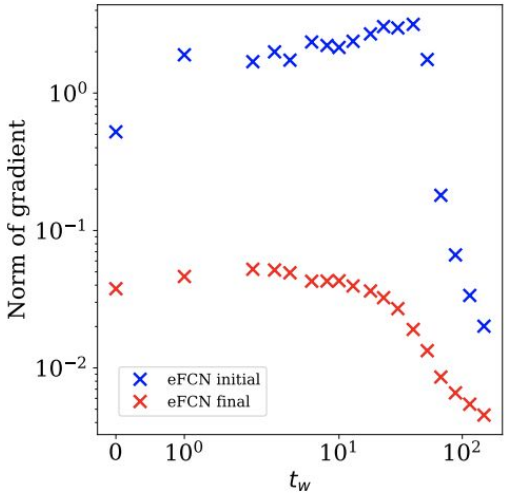
AlexNet, CIFAR-100, Adam

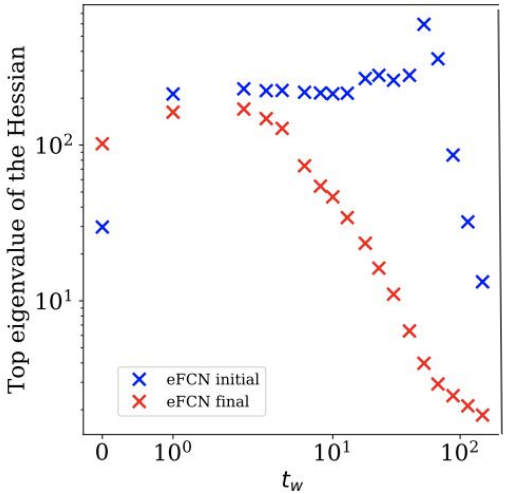
The improvement is best when constraints are released at intermediate times.

So what exactly are the extra degrees of freedom doing ?

A closer Look at the Landscape

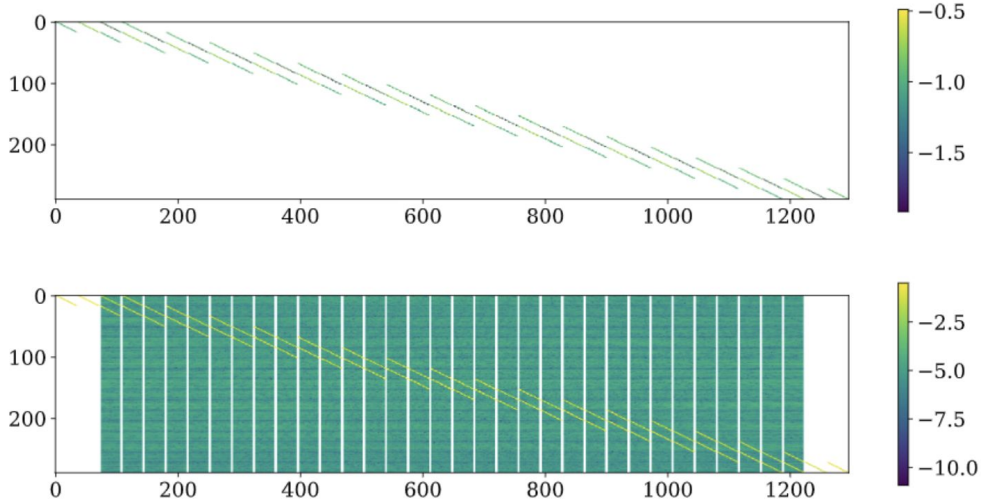






- The improvement is highest when constraints are released in the roughest/sharpest part of the landscape
- eFCN reaches wider basins

Local VS Off-Local Blocks



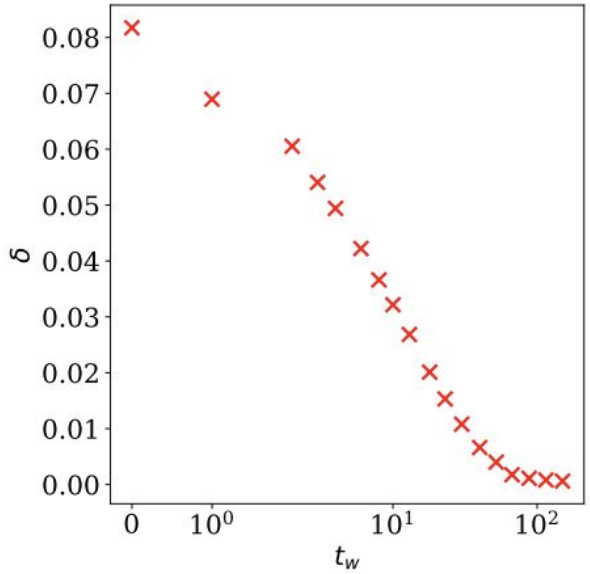
eFCN at initialization

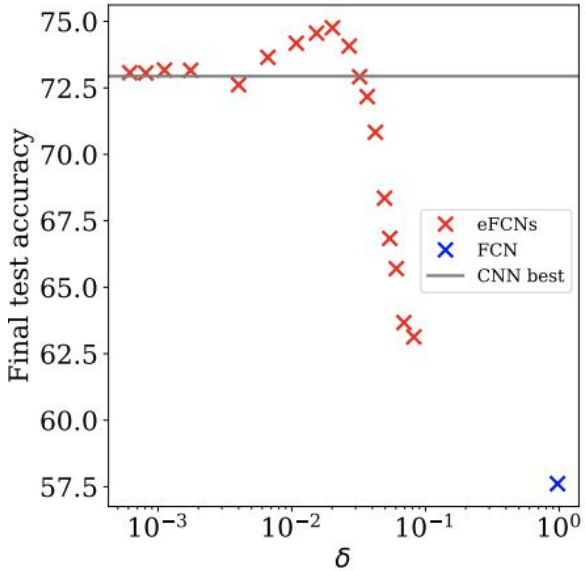
eFCN after 100 epochs

Measure of how far the model is from CNN subspace:

$$\delta = \frac{||\theta_{\text{off-local}}||_2}{||\theta||_2}$$

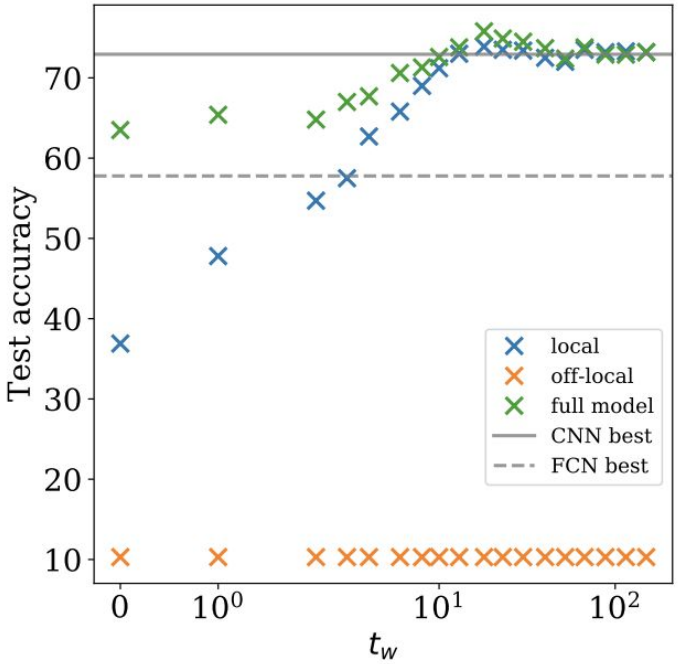
How Far Do We Escape From The CNN Subspace ?





- The earlier we relax constraints, the further we escape
- There is an optimal and a cutoff distance from CNN subspace

Role Of Off-Local Blocks



- All by itself the off-local part is useless
- However combined with local part it provides improvement

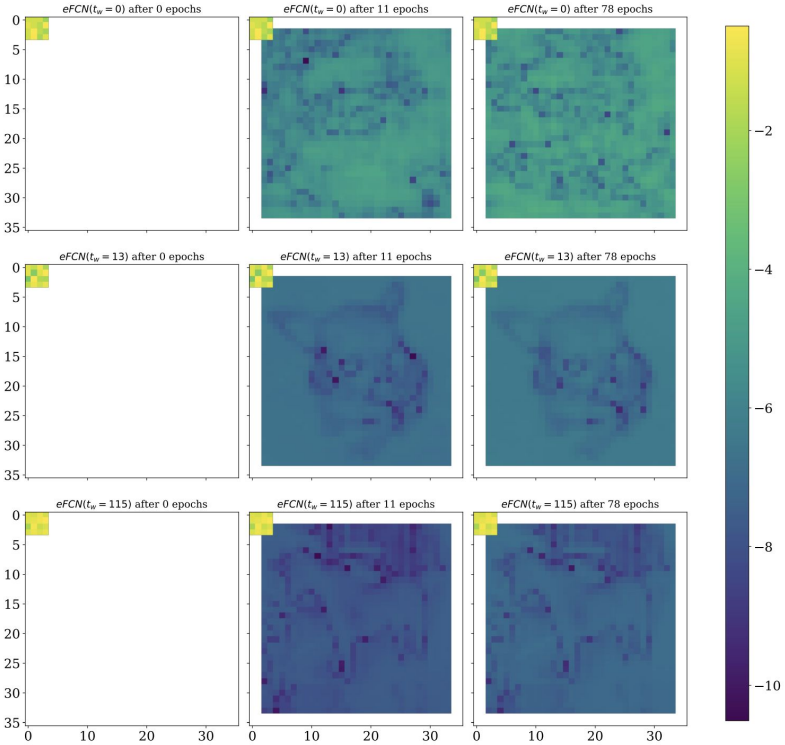
How can local and off-local blocks play such complementary roles ?

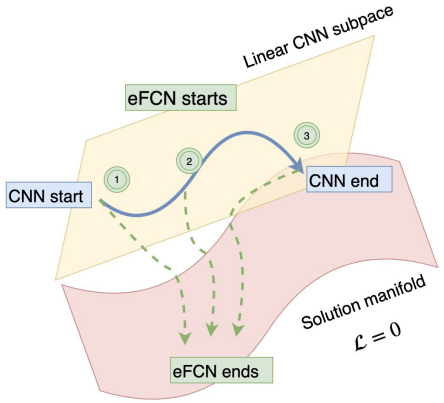
What Representation is Learnt ?

1

2

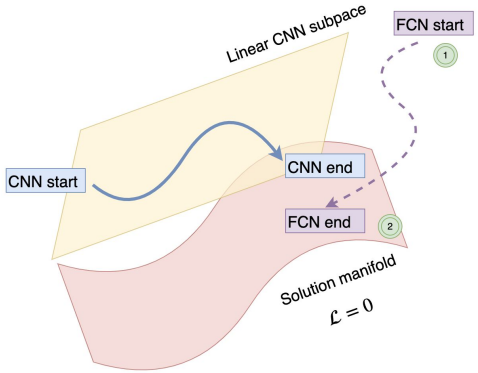
3

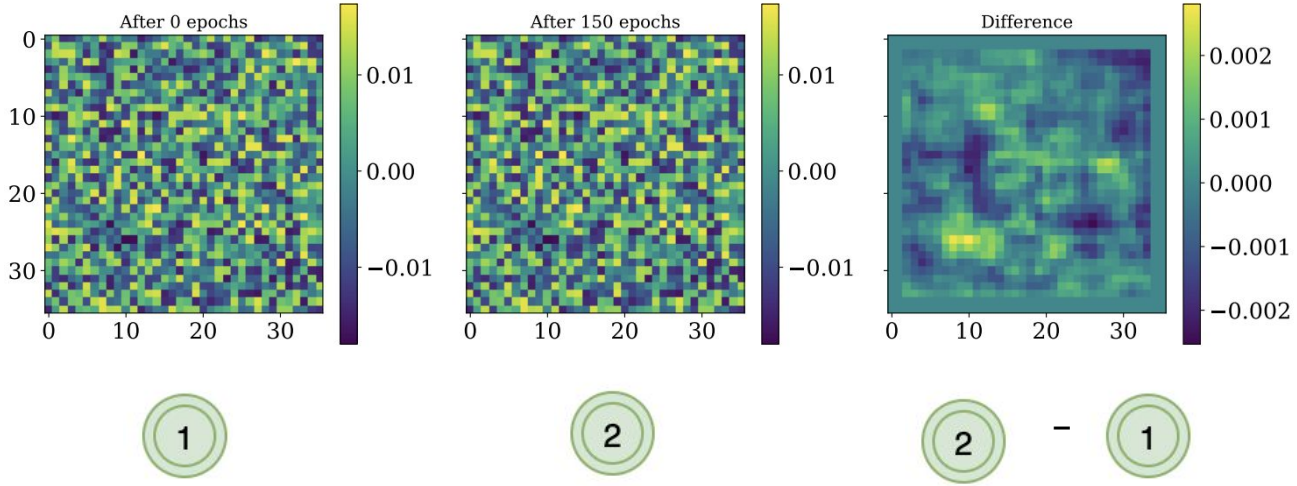




- The local blocks hardly change at all
- The off-local part learns silhouettes : performs some kind of template-matching
- Plays a complementary role to the local part's feature extraction

What Representation Is Learnt ?

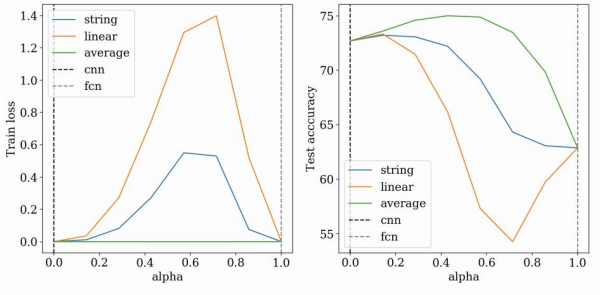




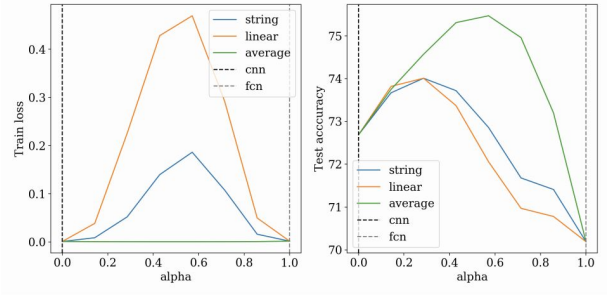
In the regular FCN case, template-matching is not a good enough strategy to make the loss diminish.

The weights capture a loose texture but nothing recognizable.

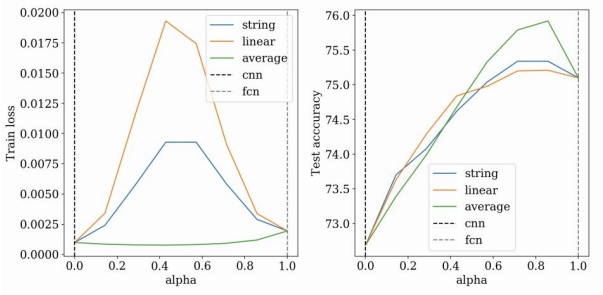
Interpolating in Weight Space



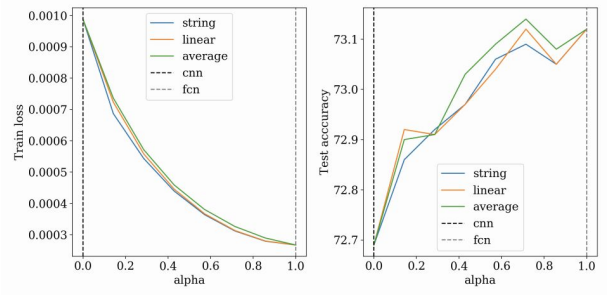
(a) $t_w = 0$



(b) $t_w = 5$



(c) $t_w = 18$



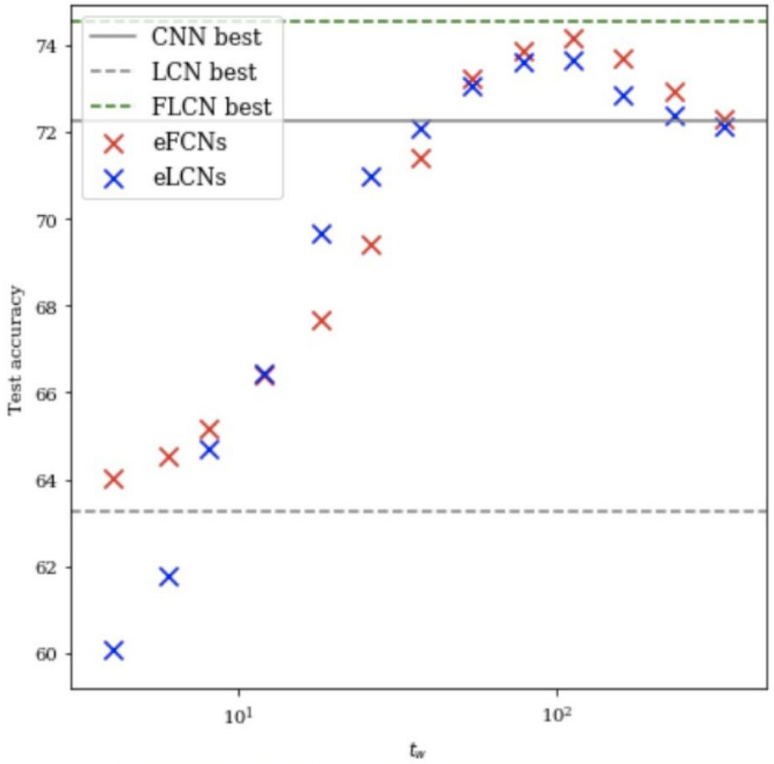
(d) $t_w = 61$

- Interpolating in weight space between the CNN and the eFCN actually increases generalization

- Maybe the best solution is somewhere in between the CNN and the eFCN ?

→ eLCNs !

What Happens For Locally Connected Networks ?



- The gain in accuracy is very similar, but this time, memory increase is much more reasonable (factor 10)
- We can get further increase in accuracy by relaxing the constraints **hierarchically** :
  FLCN(tw1, tw2) = CNN -> eLCN(tw1) -> eFCN(tw2)
- Next step : plot the performance increase in the (tw1, tw2) plane (WIP)

Conclusions

1. CNN constraints are mostly helpful at the **initialization** and **early stages** of training, bringing us to good regions of the landscape

2. Releasing the constraints once they have done their job, we can improve performance

3. The extra degrees degrees of freedom are useful in two ways:

    ⇨ Landscape point-of-view : they give access to wider basins

    ⇨ Representation learning point-of-view : they perform complementary tasks (*feature-extracting* local filters + *discriminative* template matching)

Thank You