HW0

david assaraf

January 2021

1 Model Description

The model we are using is a replication of Conv-4 from the literature, with 4 different layers, each comprising BN and MaxPooling. The size of the kernel for the CNN layer is a parameter of our model and will be changed across our different experiments.

2 Robustness

Link to code: https://gist.github.com/DavidAssaraf106/340dc0eea5c37f8abe13b6cbe17f23f1

2.1 Upside-down flip on the test set

Questions we would like to have answered:

- What happens if you flip all of the images in the test set vertically?
- Does an upside-down cat still look like a cat to your CNN?

In other words, we are trying to explore if the features map that we construct for our images are robust to a vertical flip (upside-down). We already know that CNN are robust to some kind of symmetries: small perturbations and translations. What happens when using rotations ? Are there some ways in order for us so that our CNN learns some robust features that are robust to vertical flips ? **Is our CNN robust to vertical flips ? Are these transformations invariant for our CNN ?** First of all, visually, meaning to human representation, a flipped cat is no longer a cat (we could recognize it, but the only thing our brain does is that it does the flip unconsciously). Let us explore this by flipping a cat on the CIFAR-10 dataset (cf Figure (1)).

We realize that, while some features are similar to the eye, is is not very likely that our Convolutional Neural Network will be robust to such upside-down flips.

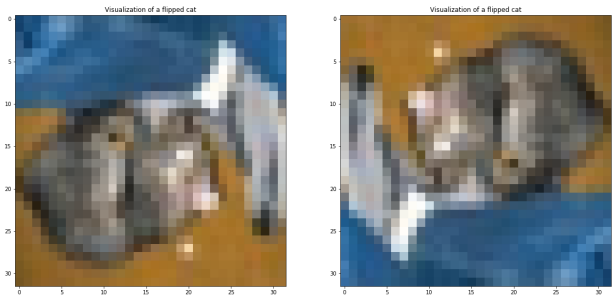

Figure 1: Visualization of flipped and un-flipped cats from the CIFAR-10 dataset. (Left): original image of a cat from CIFAR-10 dataset, (Right): upside-down flip of an image from a cat in the CIFAR-10 dataset

2.1.1 Global Performances on the Test Set without changing the training procedure

The first experiment we conducted was to test the robustness of our CNN **without** changing the training procedure. Therefore, we had:

- One Training Set, CIFAR-10: 50000 training examples, with 10 different classes
- One Testing set, the one from CIFAR-10: 10000 testing examples, with every one of the 10 classes
- One **flipped** Testing set, created from CIFAR-10 testing set, where we vertically flipped all of the images, which we will call the *Flipped Test Set*

The settings of the experience we conducted were:

- Batch-size : 64
- Loss Function: Cross Entropy Loss
- Optimizer with constant learning rate schedule set to 0.1 (this problem is fairly easy to solve so I set up an 'easy optimization')
- Trained the model for 80 epochs (the training loss was 0 after 40 epochs and the monitoring metrics were constant)

The results of the experiments are displayed in Figure (2).

We can see that the test accuracy drops from 0.858 to 0.392. Therefore, the new model is not completely random, but the drop in performance is important.

In terms of training dynamics, we can see that initially, on the flipped test set, the performances are random-like, with an accuracy of 0.1. Then, the accuracy

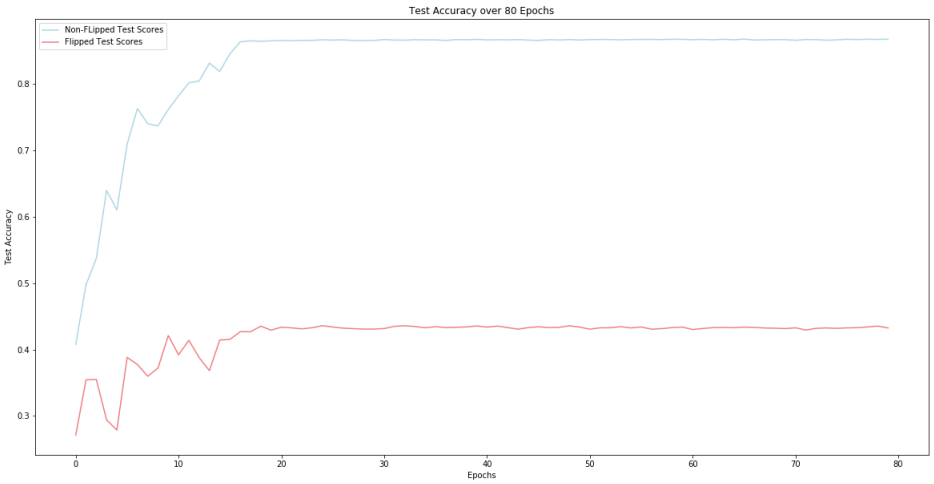

Figure 2: Accuracy Performances of CNN on initial test set (blue curve) and on the flipped test set (red curve)

improves to almost 0.4, in the same epochs as the accuracy on the original test set increase to 0.8. Therefore, the features we are learning initially to classify our images are helpful for the flipped test set, but once these initial features are learnt, the rest of the training is not useful for the performances on the flipped test set. This could be interpreted as what is being done in [Fra+20a], where the emphasis is being set on the initial phases of training, emphasis being set in a follow-up work [Fra+20b], where the optimization yields to a subspace in the loss landscape where minima are linearly connected. Therefore, our optimization procedure learns a feature map in the early phases of training where *some* features are robust to vertical-flip, but subsequent training does not help since it is only a fine-tuning in the already determined local optima.

2.1.2 Per class Performances on the Test Set without changing the training procedure

The subsequent question we are asking ourselves from the previous results is: why is our network not performing well when flipping the images ? Aren't we supposed to extract relevant features for classification, that maybe should not be dependent on the orientation of the image ? Well, it appears that the **current architecture** we are leveraging does not seem to work. Maybe it is because some classes are highly non symmetric to the 'flipping operation'. Moreover, we can see that the classification accuracy on the flipped test-set is **not** purely random: why is it so ? Is it because some flipped images are very much alike

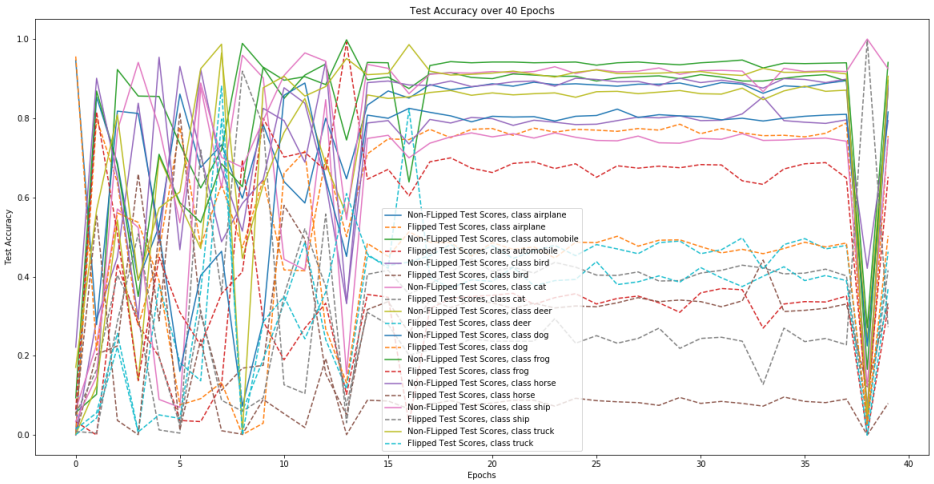

Figure 3: Per Class Accuracy Performances of CNN on initial test set (Plain lines) and on the flipped test set (Dashed lines)

then the initial images ? Or is it because our network learns some features that are partially robust to vertical flips ? Let us verify this behaviour by checking what are the performances drop per class when entirely flipping the test set (cf figure (3)).

We can see that our network does not learn to classify flipped images, but rather that some classes are more easily recognizable when flipped than the others (the visual difference between a flipped image of this class and the original image is not significantly different). This is the case for:

- airplanes
- frogs

Now that we have identified this non-robustness, we must do something in order to alleviate it. Amongst the several solutions, I identified some:

- Change the architecture into an architecture being vertical-flip-invariant
- Change the training procedure

2.2 Same Architecture, new Training procedure

In order to alleviate this issue, I have created a new training set: the original images from CIFAR-10 + the vertical flipped images from CIFAR-10, resulting in a training base of 100000 images. After this data augmentation step, we have

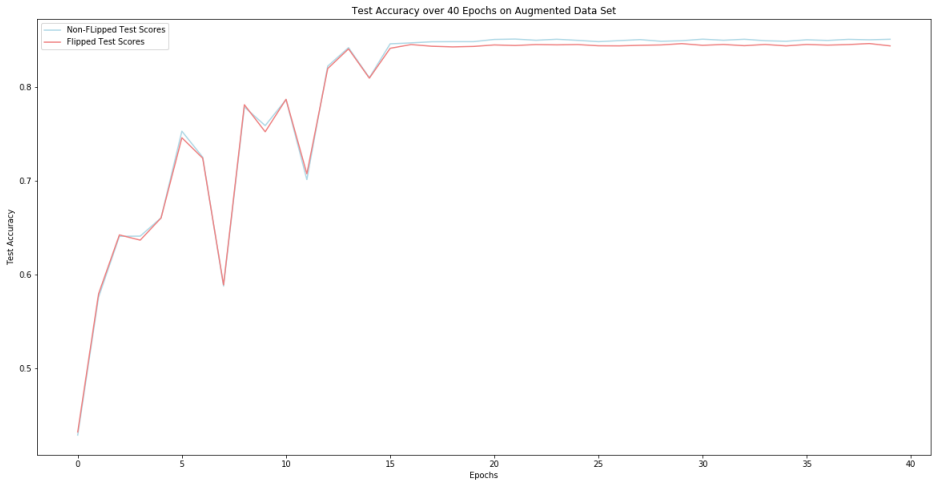

Figure 4: Accuracy Performances of CNN on initial test set (blue curve) and on the flipped test set (red curve), after having performed a data augmentation step

trained our CNN with the exactly same training procedure as we have done with the original training set. The results we got are that after 40 epochs, the global training loss function (over the entire flipped and non-flipped training set) is 0. The monitoring of the metrics are:

2.3 New architecture, Same Training Procedure

However, lots of studies are directing towards the fact that data augmentation is not enough to build robust CNN. One more reliable way to do so would be to devise an architecture that would be invariant to random rotations. This is something that has been done with very promising results in [Coh+18]. Some implementations are already made within the pytorch framework, I hope to find the time to test them.

3 Random Labels

Link to code: https://gist.github.com/DavidAssaraf106/438460aa52bea048eac519ac573eaf65

This exercise is entirely based on the experiment conducted by Zhang et al. in the paper Understanding Deep Learning Requires Rethinking Generalization, [Zha+16]. In this paper, the problem of generalization is being tackled:

how is it possible that such models with an incredible number of parameters (being a proxy for model complexity) generalize well ?

Indeed, from the classical statistical learning theory, the common generalization bound heavily depend on the sample size and the model complexity. However, when training Deep Neural Networks, it is very usual to train model with 10s millions of parameters with fairly lower amount of data: therefore, the usual bounds cannot help in explaining the generalization ability.

In the following, we are going to show that traditional approaches fail to explain will large neural networks generalize well in practice: we are going to show that the traditional complexity theory for statistical learning is incapable of distinguishing between different neural networks that have radically different generalization performances.

Namely, we are going to show that state of the art large CNN trained with SGD easily fit a random labeling of the training data (**Over Parametrized models**, $p >> n$). More precisely, when trained on a completely random labeling of the true data, neural networks achieve 0 training error. The test error, of course, is no better than random chance as there is no correlation between the training labels and the test labels. In other words, by randomizing labels alone we can force the generalization error of a model to jump up considerably without changing the model, its size, hyperparameters, or the optimizer.

On the other hand, we are going to show that for smaller models, this fit is less obvious and less easy to obtain (**Under Parametrized models**, $p << n$). This is coherent with the experiments of Zhang et al. showing that simple depth 2 neural Networks already have perfect finite sample expressivity **as soon as the number of parameters exceeds the number of data points.**

3.1 Experiment

The experiment we conducted in order to show that large models can easily fit a random labeling of the training data is the following: considering CIFAR-10 dataset, we are going to randomize the labels of p fraction of the dataset:

- $y_i = y_i$ with probability $1 - p$
- $y_i = \mathcal{U}[0; 9]$ with probability p

We want to analyze the **Generalization Behavior** of the model loss between training and testing set, and see how it depends on: **model complexity** (number of parameters of the neural network) and **aleatoric behavior of the labelling** (ie the value of p).

3.2 Under parametrized model

The first model we have trained is the CNN with width c = 4. The number of trainable parameters for this model is $p = 6666 << n = 50000$. For the training of this model, we have used SGD as an optimizer, with a learning rate constant

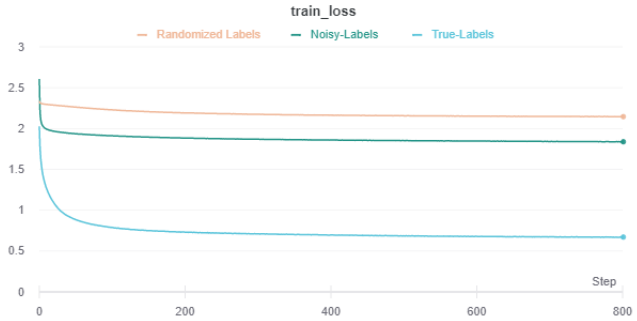

Figure 5: Training Loss for the Under parametrized model : CNN with c = 4. The training loss are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

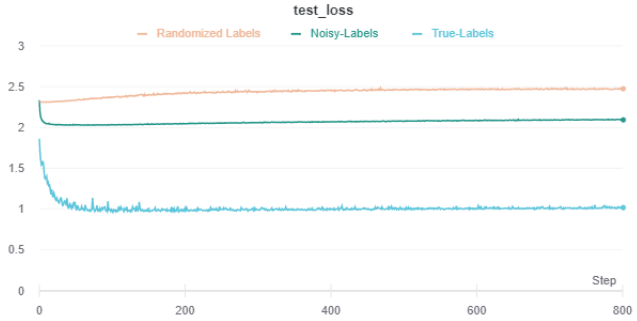

Figure 6: Testing Loss for the Under parametrized model : CNN with c = 4. The testing losses are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

schedule of 0.003 (as suggested per [FC18] where they have been training Conv-8 on CIFAR-10 with this learning rate). We have trained these models for 800 epochs (we have been waiting for the train loss to become completely flat), for each of the different (increasingly random) datasets. The results we had from training and testing loss are:

Some observations:

- We can see that the under-parametrized model does not seem able to learn the noisy or randomized labels
- Randomizing the dataset seem problematic for the under parametrized model: the optimization seems more difficult and slower
- The under parametrized model does not seem complex enough to memorize the labels in the dataset

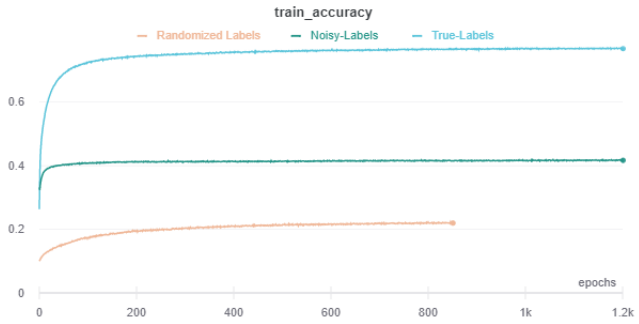

Figure 7: Training Accuracy for the Under parametrized model : CNN with c = 4. The training accuracy are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

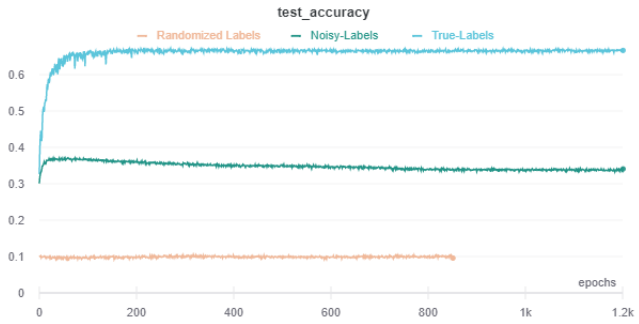

Figure 8: Testing Accuracy for the Under parametrized model : CNN with c = 4. The testing accuracy are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

Figure 9: Training Loss for the Over parametrized model : CNN with c = 64. The training loss are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

Figure 10: Testing Loss for the Over parametrized model : CNN with c = 64. The testing losses are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

- **Generalization:** the generalization gap between training and testing seems reasonable, meaning that one can assume a control over the test loss based on the train loss

3.3 Over parametrized model

The second model we have trained is the CNN with width c = 64. The number of trainable parameters for this model is $p = 1558026 >> n = 50000$. For the training of this model, we have used Adam as an optimizer, with a learning rate constant schedule of 0.0003 (as suggested per [FC18] where they have been training Conv-8 on CIFAR-10 with optimizer and this learning rate). We have trained these models for 500, 1000 and 1000 epochs (we have been waiting for the train loss to become completely flat), for each of the different (increasingly random) datasets. The results we had from training and testing loss are:

Some observations:

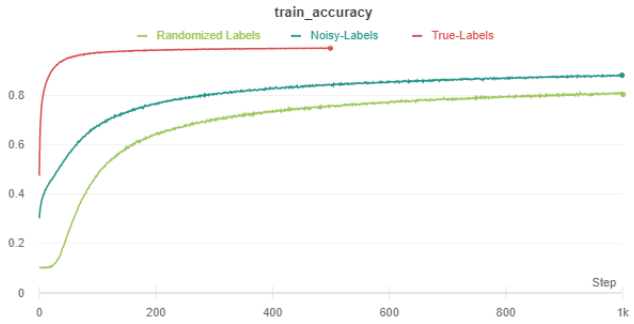

Figure 11: Training Accuracy for the Over parametrized model : CNN with c = 64. The training accuracy are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

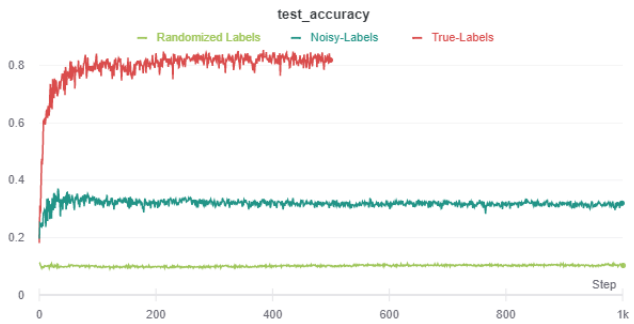

Figure 12: Testing accuracy for the Over parametrized model : CNN with c = 64. The testing accuracies are displayed for True labels (p=0), Noisy labels (p=0.5) and completely random labels (p=1).

- We can see that the over-parametrized model seems able to learn the noisy or randomized labels. However, there is nothing to learn, ie no relevant relationship between $\mathcal{X}$and$\mathcal{Y}$: therefore, the over-parametrized model memorizes the different training labels
- Randomizing the dataset seem problematic for the over parametrized model: the optimization seems more difficult and slower: the more noisy the dataset is, the slower the optimization process is
- The over parametrized model seems complex enough to memorize the labels in the dataset
- **Generalization:** the generalization gap between training and testing is arbitrary. We can see that, based on this experiment, **training error is not a proxy for test error for Deep Neural Networks** . We can see here and can infer that, for an arbitrary large number of epochs, the training error will go down to 0. However, we can see that the testing error for both noisy and randomized dataset does not seem to lower, whereas it is lowering for the true dataset. Therefore, the same model (with the same architecture and optimizer) generalizes very differently based on the population it has been trained on: we cannot infer some generalization bound from this experiment

3.4 Discussion and Questions

- **Complexity Theory:** We can see from these experiments (and Zhang paper, [Zha+16]) that for over-parametrized deep neural networks, the usual generalization bounds does not seem to hold. However, they do seem to hold for smaller architectures (like the one we have been training on the first model, CNN with c=4). It seems that this differentiation between memorizing and generalizing neural networks is based on the complexity of the model: is there an estimation for the number of parameters necessary in order to be in the memory regime ? This should be a function of n + the network architecture (repartition of parameter accross the width and/or depth).
- From the LTH, [FC18], and other papers related to pruning, we have discovered that from Deep Neural Networks, we could create very sparse networks with a dramatically reduced number of parameters, that would be able to learn and generalize in the same way as the original network did. Isn't this a complexity measure ? Something that would be the *Effective Number of Parameters*? Those would be responsible for learning and generalizing, and the other parameters would be responsible for memorizing the data. Another paper has been made on the same context, [ZBS19], pointing out to the fact that some layers of a network are ambient and other are critical. Those ambient layers might also reduce the effective complexity of a Network, maybe allowing to provide some more insights on the Generalization gap.

- From the paper [MRB18], some insights have been provided on the representational similarity between memorizing network and between generalizing networks, and that wider networks converge to more similar solutions. One important insight from this paper is also the fact that they demonstrated that **canonical correlation convergence was a proxy for generalization**. If we could prove that generalizing networks converge all towards a set of equivalent networks (with an equivalence relationship to be determined), and if one could extract the efficient set of weights from the representative of this class, maybe this would enable to have a proxy for model complexity and differentiate memorizing models from generalizing models.

4 Linear Regression

Link to code:https://gist.github.com/DavidAssaraf106/71bd6c5e538923e6c14c692fb6538d15

4.1 Normal equation

1. The rank of our design matrix is 500. The rank of the matrix is ultimately constrained by $min(n, p)$ where n is the number of rows and p is the number of columns of the specific matrix. Therefore, here our matrix is not full rank, and furthermore we have more features than data points:$p > n$: the matrix is going to be singular and it means that there are several solutions to the normal equations.

2. We are looking for :
$$\hat{\beta} = \min_{\beta} ||X\beta - Y||^2 \tag{1}$$
This is a free-optimization problem on $\mathcal{L}(\beta) = ||X\beta - Y||^2$. We can solve it finding the point where the gradient of $\mathcal{L} = 0$
$$\nabla(\mathcal{L}(\beta)) = 2X^T(X\beta - Y) = 0 \iff X^T X\hat{\beta} = X^T Y$$
This is the normal equation we were looking for. Therefore, several conclusions might be assumed based on this equation:
   - If X is non singular ($X^T X$ is invertible), then there is only one unique solution to the OLS problem : $\hat{\beta} = (X^T X)^{-1} X^T Y$
   - If X is singular, there is an infinity of solutions to this normal equation. (One solution to dealing with this problem is to perform Ridge Regularization to the OLS problem: the empirical covariance matrix becomes $X^T X + \lambda * I$, which is invertible because $X^T X$ is SDP).

   In our case, our matrix is singular, but we do not want to perform Regularization, we want to find **one** weight parameter working. We can do so finding the pseudo-inverse of the matrix $X^T X$: The Moore–Penrose

inverse. This is what scikit-learn or statsmodel do when trying to inverse a general matrix. Having found our parameter, we can therefore estimate

$$\mathbb{E}_{x,y\sim D}[(f_\beta(x) - y)^2] \tag{2}$$

on the test set distribution. The result we get is 9.48.

3. Like computed beforehand, we have:

$$\nabla(\mathcal{L}(\beta)) = 2X^T(X\beta - Y) \tag{3}$$

Let us try to perform SGD on the objective function, with the gradient of the objective function above. The experiment we conducted was using sklearn SGD Regressor in order to perform SGD on the objective function. Our algorithm converges towards a solution, and it is expected to be the solution (amongst all the solutions) that has minimal norm. The norm of our weight parameter is ≈ 1, which seems reasonable. The learning rate I set was a constant learning rate schedule, $\eta = 0.001$, with automatic initialization of the weights at 0.

4. I thought that my SGD did work. However, after printing the training error, I realized that it was not explicitly zero and no not understand why. Regarding the **spectrum** distribution was also helpful and made me realize that the eigenvalues of the matrix X^TX indeed followed the **Marchenko-Pastur** distribution. The condition number is effectively zero but the conditioning number in the non-null space is decent, which allows to perform gradient descent.

References

[Zha+16] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).

[Coh+18] Taco S Cohen et al. "Spherical cnns". In: *arXiv preprint arXiv:1801.10130* (2018).

[FC18] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[MRB18] Ari S Morcos, Maithra Raghu, and Samy Bengio. "Insights on representational similarity in neural networks with canonical correlation". In: *arXiv preprint arXiv:1806.05759* (2018).

[ZBS19] Chiyuan Zhang, Samy Bengio, and Yoram Singer. "Are all layers created equal?" In: *arXiv preprint arXiv:1902.01996* (2019).

[Fra+20a] Jonathan Frankle et al. "Linear mode connectivity and the lottery ticket hypothesis". In: (2020), pp. 3259–3269.

[Fra+20b] Jonathan Frankle et al. "The Early Phase of Neural Network Training". In: (2020), pp. 3259–3269.