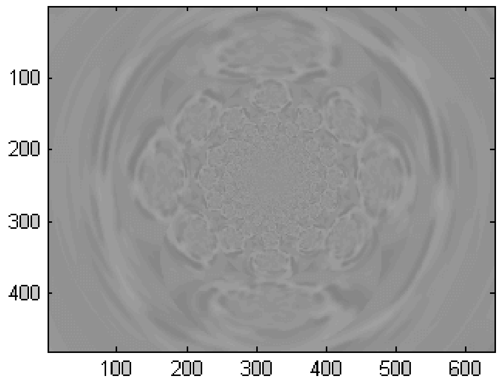
S6

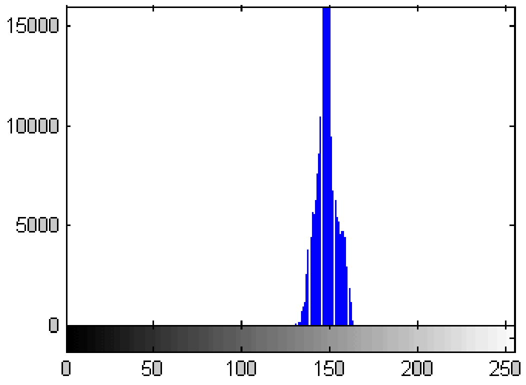
- Due No Due Date
- Points None
- Available after Feb 23, 2020 at 9:30am

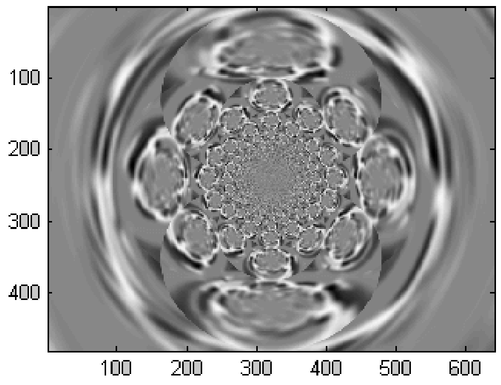
Batch Normalization & Regularization

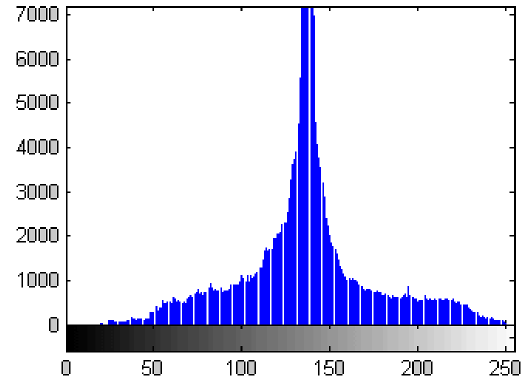
Normalizing input (LeCun et al 1998 “Efficient Backprop”)

Let's understand through some examples:









Why the redistribution of data is important for us?

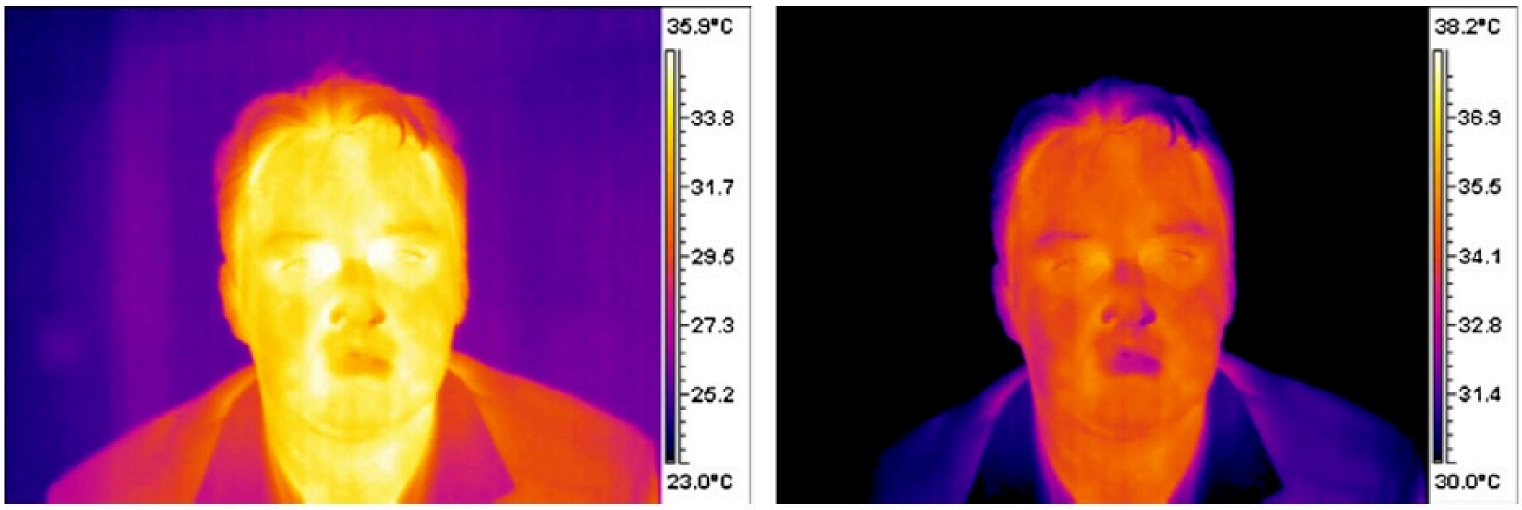


Fig. 10. Examples of thermal face images. Raw thermal image (left). Normalized thermal image (right).

Normalization is not Equalization

The normalize is quite simple, it looks for the maximum intensity pixel (we will use a grayscale example here) and a minimum intensity and then will determine
a factor that scales the min intensity to black and the max intensity to white.
This is applied to every pixel in the image which produces the final result.

The equalize will attempt to produce a histogram with equal amounts of pixels in each intensity level. This can produce unrealistic images since the intensities
can be radically distorted but can also produce images very similar
to normalization which preserves relative levels in which the equalization process does not.

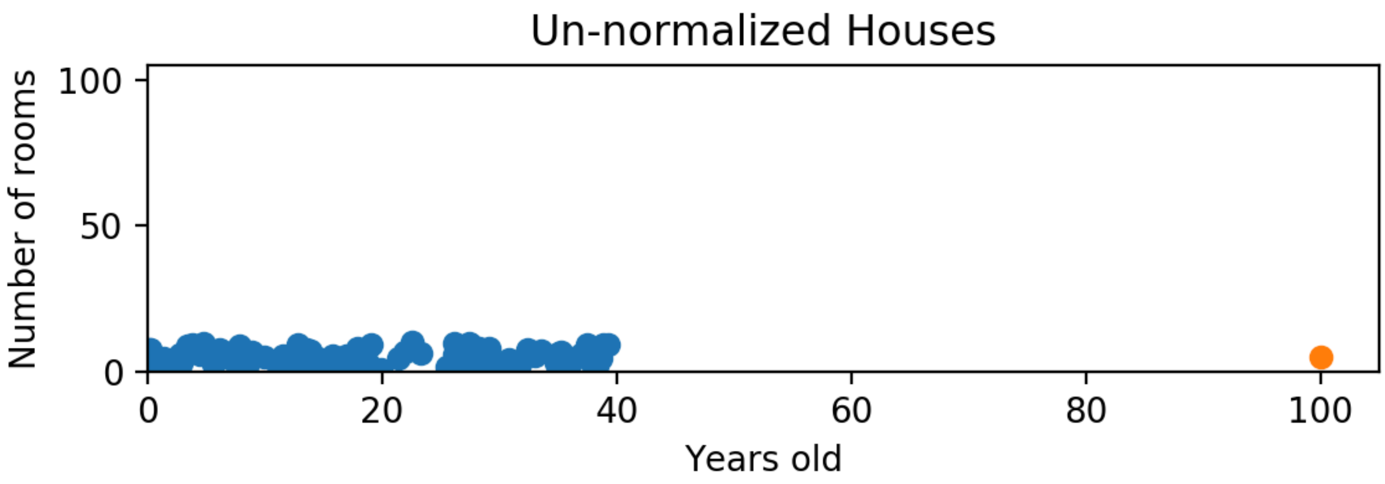
So if you are concerned about keeping an image realistic then use normalization,
but if you want a more even distribution of intensity levels then equalize can help with that. [SOURCE](http://www.roborealm.com/forum/index.php?thread_id=4350) (http://www.roborealm.com/forum/index.php?thread_id=4350)

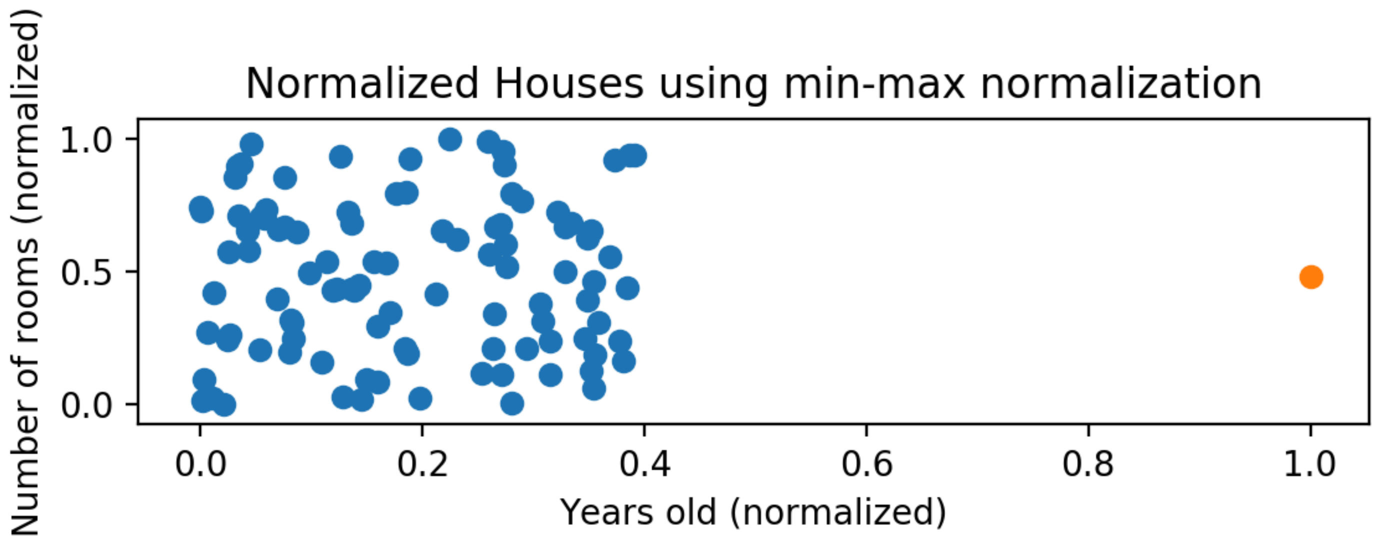
original image

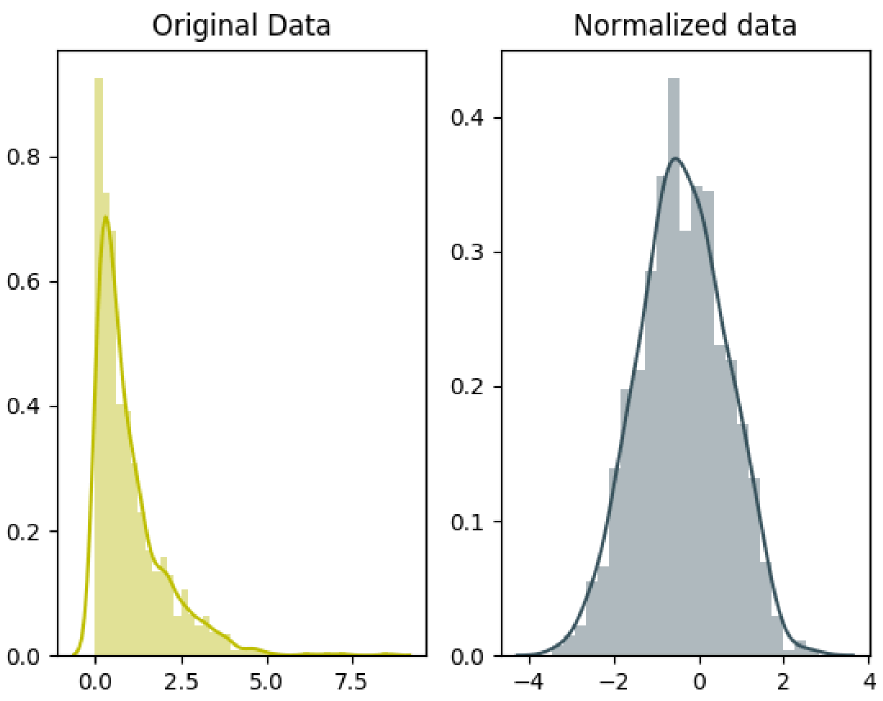
normalized image

equalized image

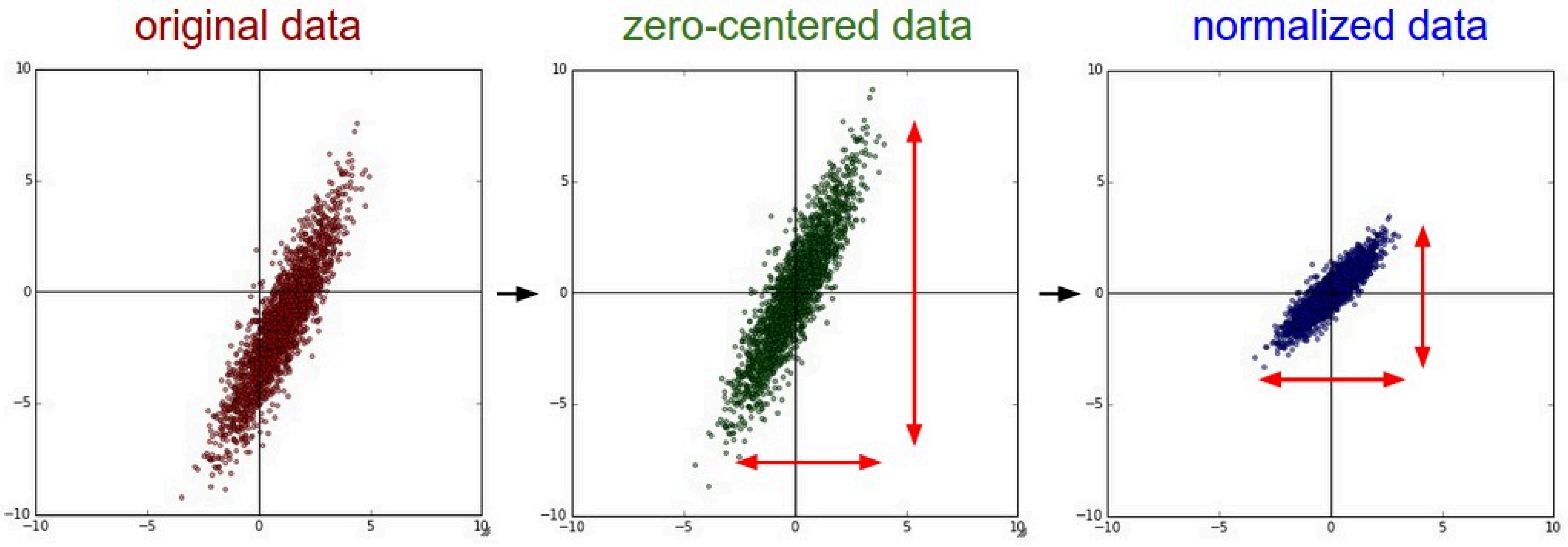
Let's look at normalization working on other kinds of data.



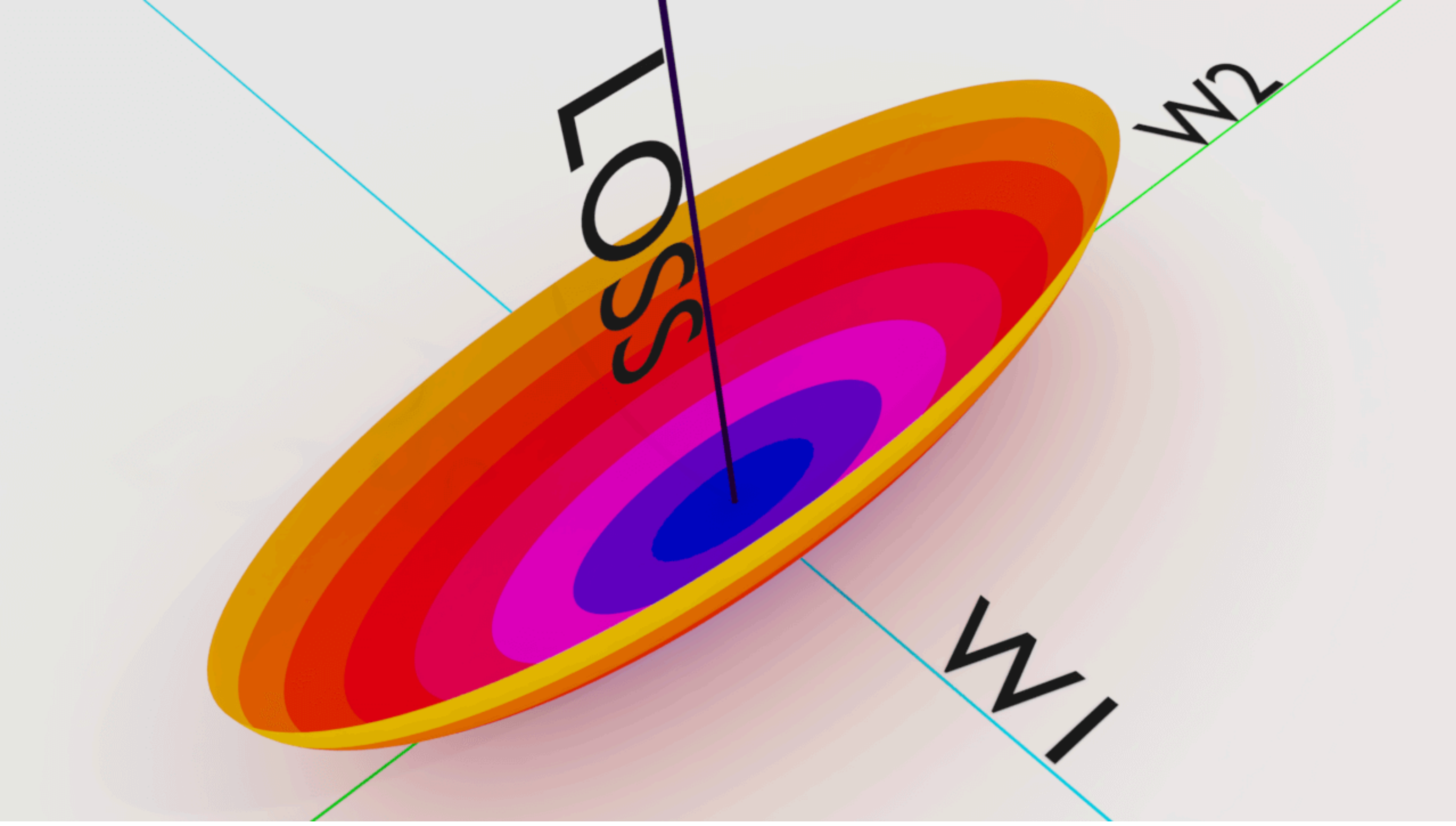




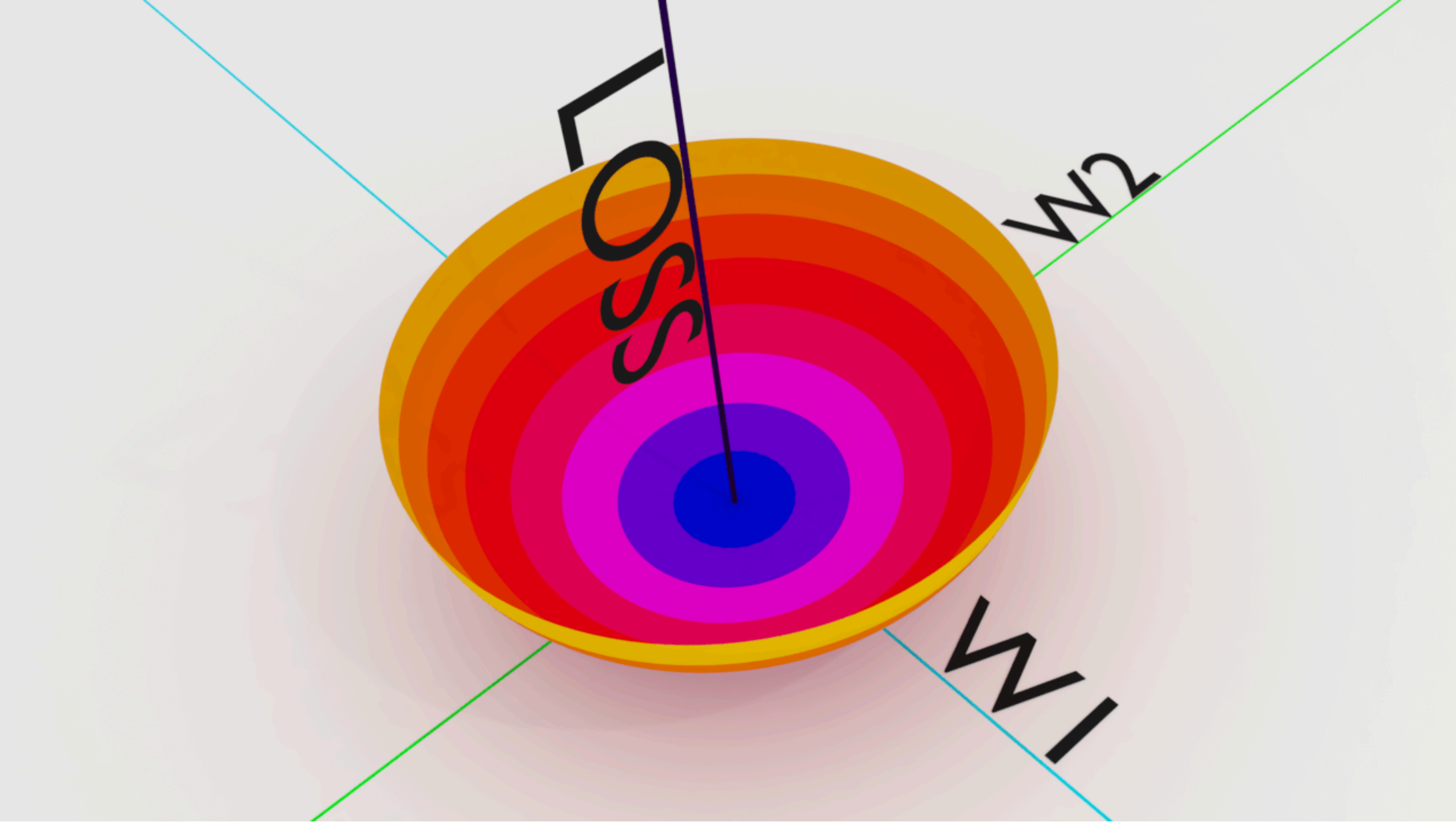
How do we achieve normalization?



Let's think about our un-normalized kernels and how loss function would look like:

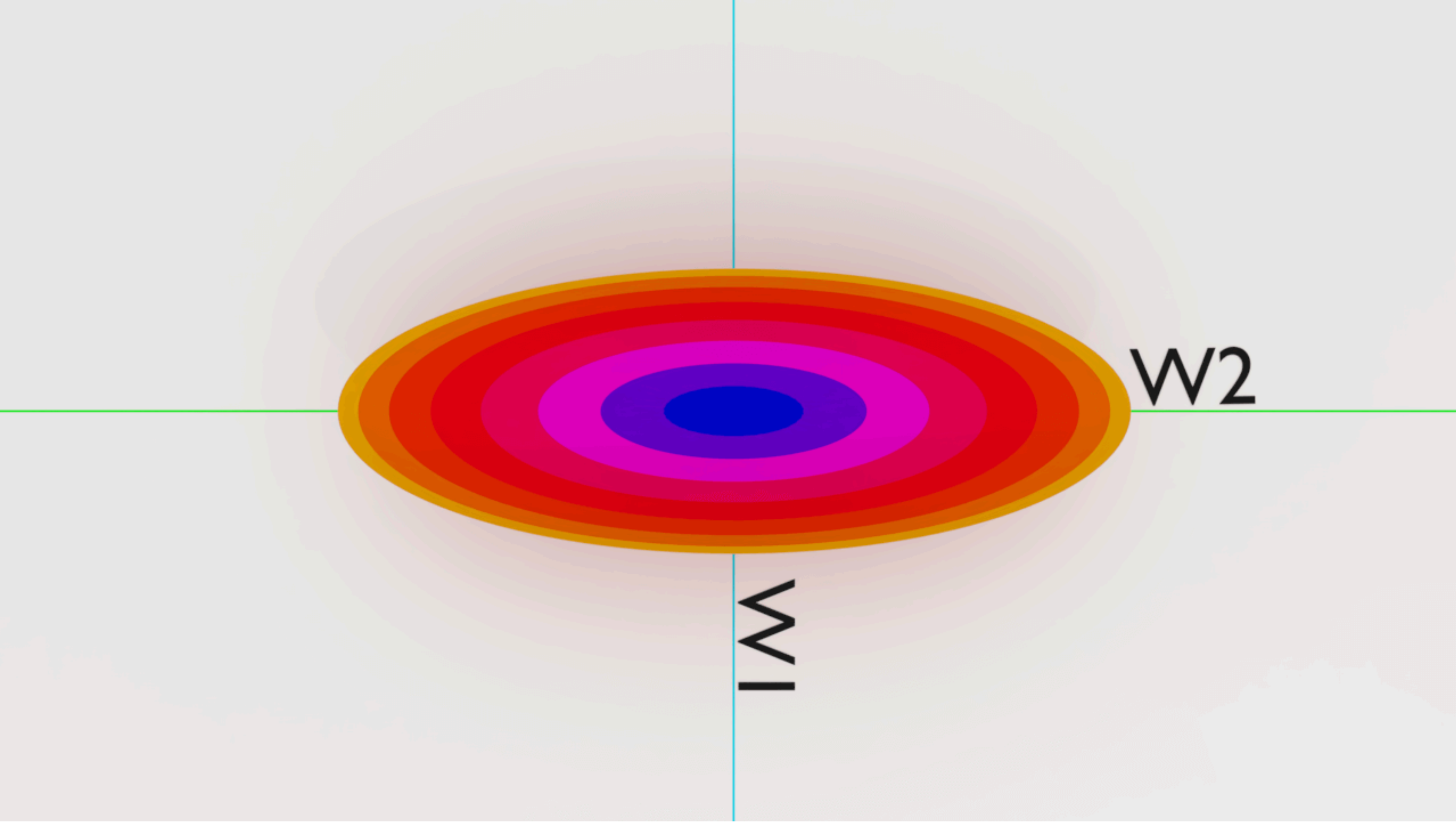


If we had normalized our kernels (indirectly channels), this is how it would look like:

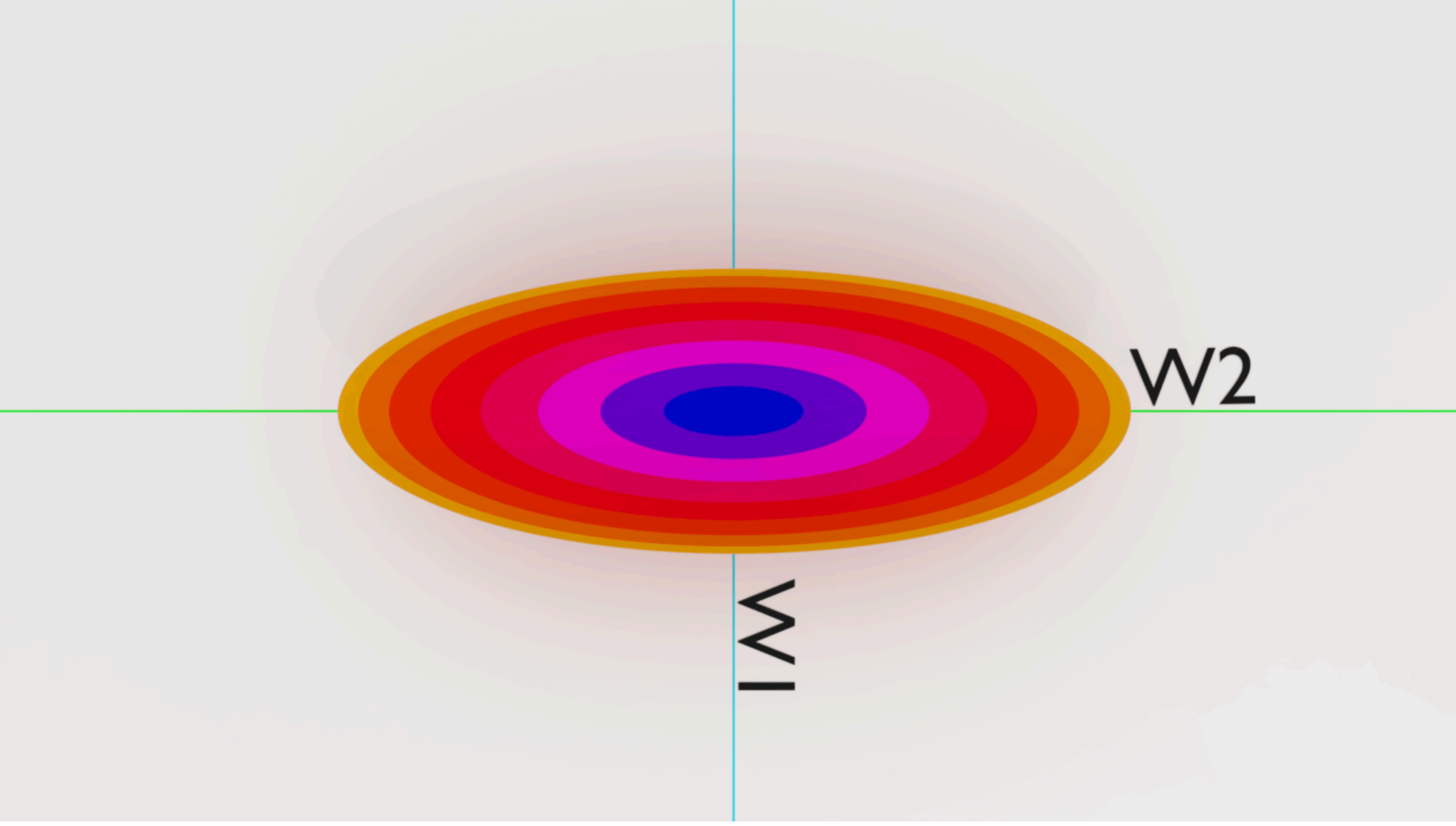


Let's look at the top view to appreciate the trouble here:

Un-normalized



Normalized



If features are found at a similar scale, then weights would be in a similar scale, and then backprop would actually make sense, ponder!

Why limit normalization to images only then?

BATCH NORMALIZATION

Batch Normalization (http://jmlr.org/proceedings/papers/v37/ioffe15.pdf) (BN), introduced in 2015
and is now the defacto standard for all CNNs and RNNs.

***Batch Normalization** solves a problem called the **Internal Covariate shift**.*

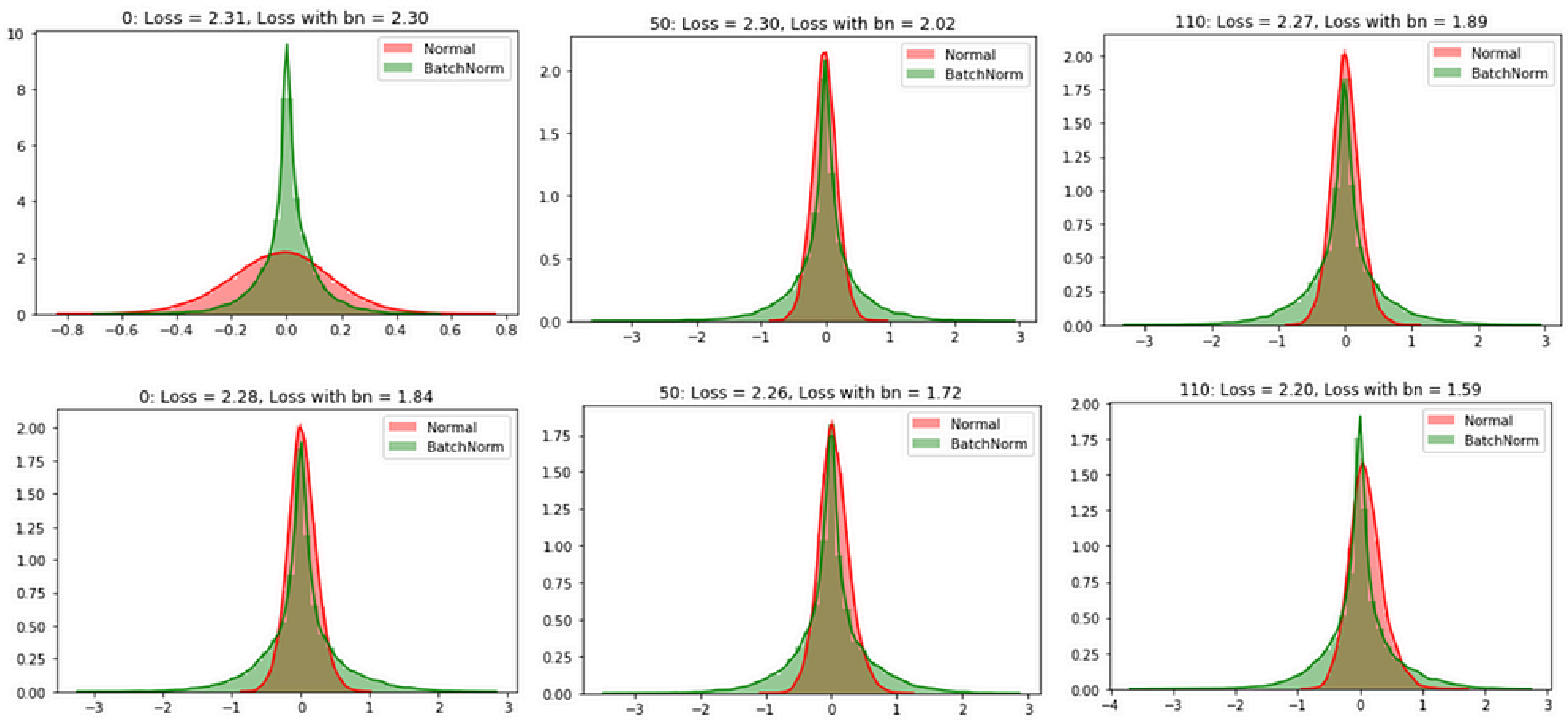
To understand BN we need to understand what is CS.

Covariate *means input features.*

Covariate shift means that the distribution of the features is different in different parts of the training/test data.

Internal Covariate shift refers to changes within the neural network, between layers. *A kernel always giving out higher activation makes next layer*
kernels always expect this higher activation and so on.

Imagine what would happen if One channel ranges between -1 to 1 and another between -1000 to 1000



Very Deep nets can be trained faster and generalize better when the distribution of activations is kept normalized during BackProp.

We regularly see Ultra-Deep ConvNets like Inception, Highway Networks, and ResNet.

And giant RNNs for speech recognition, [machine translation (https://research.googleblog.com/2016/09/a-neural-network-for-machine.html)](https://research.googleblog.com/2016/09/a-neural-network-for-machine.html), etc.

Let's look at some math:

Batch Normalization (BN)

$$\rightarrow \boxed{\text{layer}} \rightarrow x \rightarrow \quad \hat{x} = \frac{x-\mu}{\sigma} \quad \rightarrow \quad y = \gamma\hat{x} + \beta$$

- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

This is how we implement "batch" normalization:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Some crucial points:

How many gammas and betas? And what do they depend on?

BIAS GET'S SUBTRACTED OUT IN BATCH NORMALIZATION

ReLU before BN or BN before ReLU?

Batch Normalization calculates its normalization statistics over each minibatch
of data separately while training but during inference a moving average of training
statistics are used, simulating the expected value of the normalization statistics.

Read this research paper:
http://proceedings.mlr.press/v37/ioffe15.pdf (http://proceedings.mlr.press/v37/ioffe15.pdf)

GHOST BATCH NORMALIZATION

Ghost Batch Normalization, a technique originally developed for training with very large batch sizes across many accelerators.

It consists of calculating normalization statistics on disjoint subsets of each training batch. Concurrently, with an overall batch size of B and a "ghost" batch size of B' such that B' evenly divides B, the normalization statistics for example i are calculated as:

$$\mu_i = \frac{1}{B'} \sum_{j=1}^{B} x_j [\left\lfloor \frac{jB'}{B} \right\rfloor = \left\lfloor \frac{iB'}{B} \right\rfloor]$$

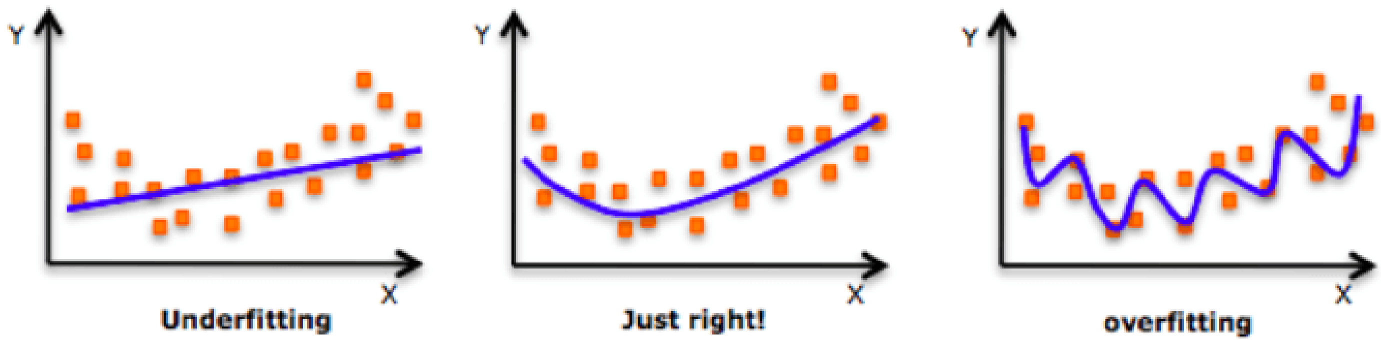
$$\sigma_i^2 = \frac{1}{B'} \sum_{j=1}^{B} x_j^2 [\left\lfloor \frac{jB'}{B} \right\rfloor = \left\lfloor \frac{iB'}{B} \right\rfloor] - \mu_i^2$$

Why might Ghost Batch Normalization be useful? One reason is its power as a regularizer: due to
the stochasticity in normalization statistics caused by the random selection of minibatches during
training, Batch Normalization causes the representation of a training example to randomly change
every time it appears in a different batch of data. Ghost Batch Normalization, by decreasing the
number of examples that the normalization statistics are calculated over, increases the strength of this
stochasticity, thereby increasing the amount of regularization.

Surprisingly, just using this one simple technique was capable
of improving performance by 5.8% on Caltech-256 and **0.84% on CIFAR-100**, which is remarkable
given it has n**o additional cost during training.**

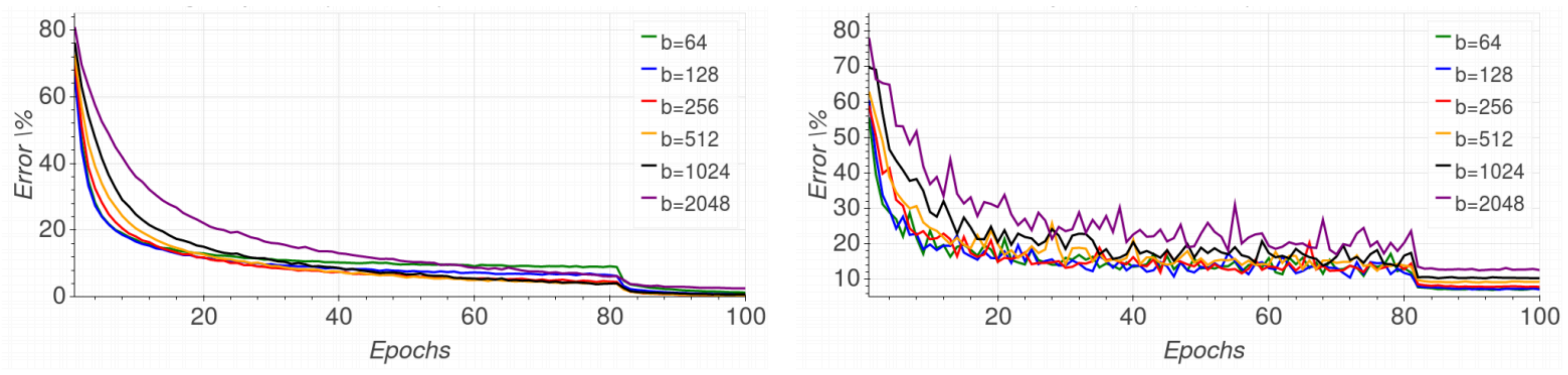
REFERENCE CODE: **SOURCE (https://github.com/apple/ml-cifar-10-faster/blob/master/utils.py)**

Regularization



Regularization is a key component in preventing overfitting.
Also, some techniques of regularization can be used to reduce model parameters while maintaining accuracy, for example, to drive some of the parameters to zero. This might be desirable for reducing the model size or driving
down the cost of evaluation in a mobile environment where processor power is constrained.

Regularization effect of batch size



(a) Training error

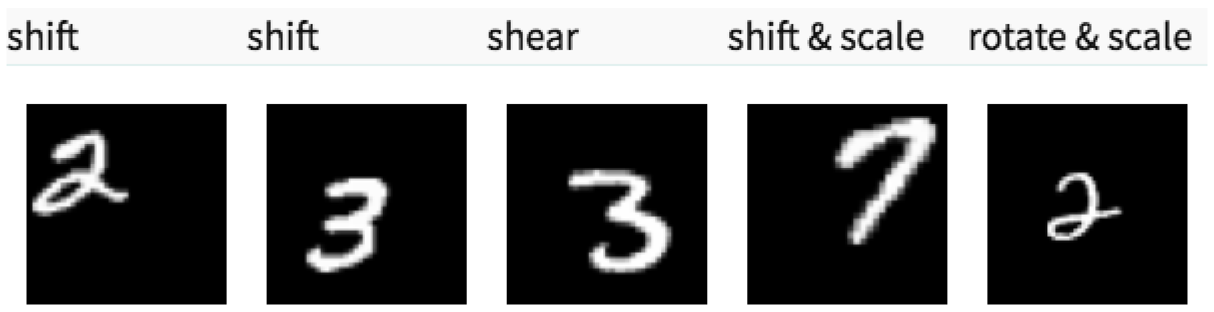
(b) Validation error

Figure 1: Impact of batch size on classification error

Most common techniques of regularization used nowadays in the industry:

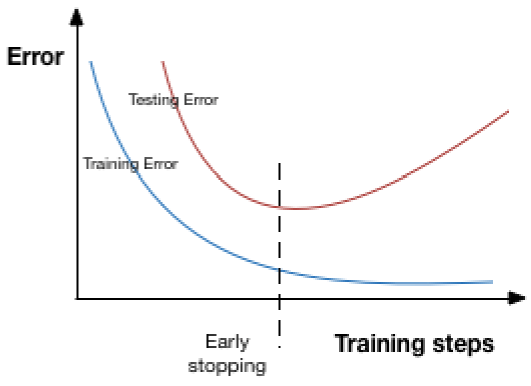
Dataset augmentation

An overfitting model (neural network or any other type of model) can perform better if the learning algorithm processes more training data.



Early stopping

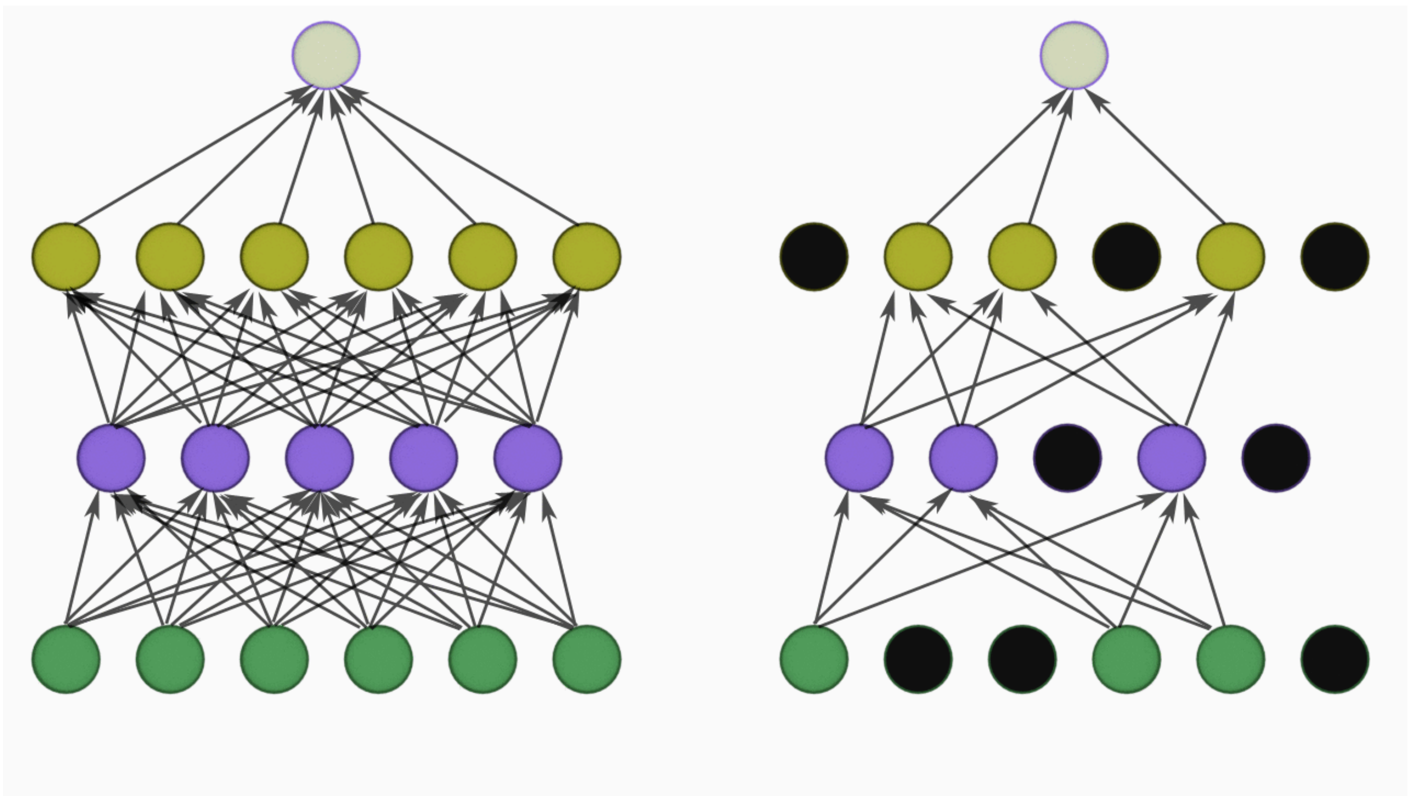
Early-stopping combats overfitting interrupting the training procedure
once the model's performance on a *validation* set gets worse.



Dropout layer

DropOut

(What gets dropped? How does it help? What happens during inference?)



Weight penalty L1 and L2

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

L1 Regularization(Lasso Regression)

L1 regularization adds an L1 penalty equal to the absolute value of the ***magnitude of coefficients***.
When our input features have weights closer to zero this leads to sparse

L1 norm.
In the Sparse solution, the majority of the input features have zero weights and
very few features have non zero weights.

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}|\theta_j|\right]$$

Features:
L1 penalizes the sum of the absolute value of weights.
L1 has a sparse solution
L1 generates a model that is simple and interpretable but cannot learn complex patterns
L1 is robust to outliers

L2 Regularization(Ridge regularization)

L2 regularization is similar to L1 regularization.
But it adds a ***squared magnitude of coefficient*** as penalty term to the loss function.
L2 will *not* yield sparse models and all coefficients are shrunk by the same factor
(none are eliminated like L1 regression)

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

Features:

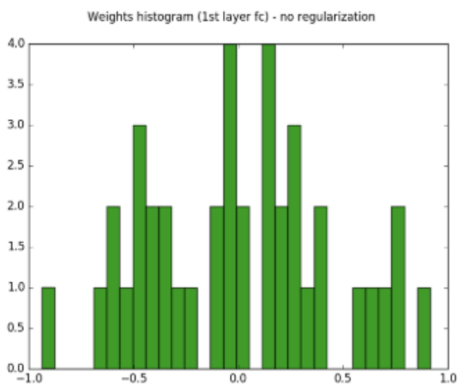
L2 regularization penalizes the sum of square weights.

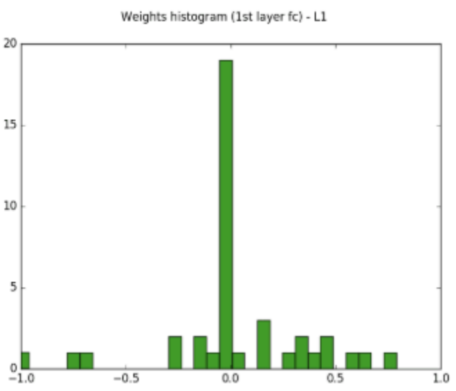
L2 has a non-sparse solution

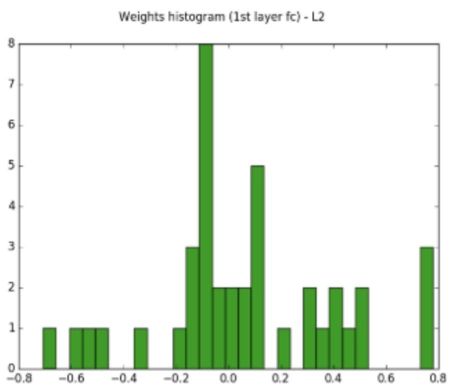
L2 regularization is able to learn complex data patterns

L2 has no feature selection

L2 is not robust to outliers







L2 In PyTorch

`torch.optim.` `SGD` **(*params*, *lr=<required parameter>*, *momentum=0*, *dampening=0*, *weight_decay=0*, *nesterov=False*)**

weight_decay (*python:float, optional*) – weight decay (L2 penalty) (default: 0)

L1 in PyTorch

```
l1_crit = nn.L1Loss(size_average=False)
reg_loss = 0
for param in model.parameters():
    reg_loss += l1_crit(param)

factor = 0.0005
loss += factor * reg_loss
```

Assignment:

Your **assignment 7** will demand these things:

1. Change your code in such a way that all of these are in their respective files:
    1. model
    2. training code
    3. testing code
    4. regularization techniques (dropout, L1, L2, etc)
    5. dataloader/transformations/image-augmentations
    6. misc items like finding misclassified images
2. So, while doing assignment 6, please think how would you be able to do this in the next assignment

GBN Note: read the paper. Please make sure you are not using same batch size for BN and GBN jobs below.

1. **Your assignment 6** is:
    1. take your 5th code
    2. run your model for 25 epochs for each:
        1. without L1/L2 with BN
        2. without L1/L2 with GBN
        3. with L1 with BN
        4. with L1 with GBN
        5. with L2 with BN
        6. with L2 with GBN

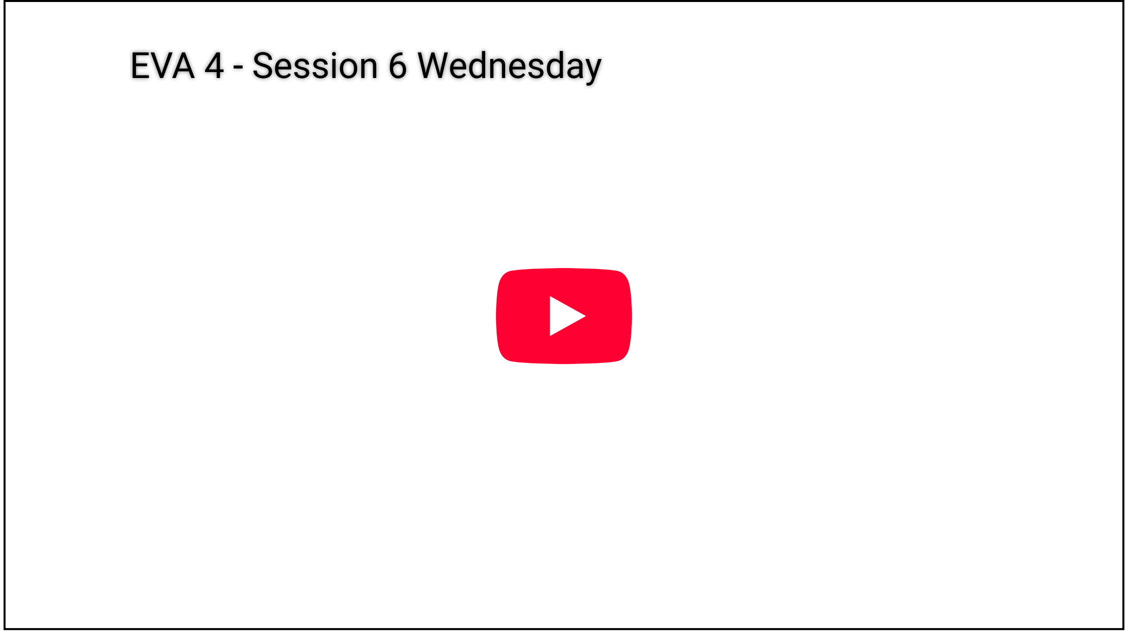
    7. with L1 and L2 with BN
    8. with L1 and L2 with GBN
3. **You cannot be running your code 8 times manually (-500 points for that). You need to be smarter and write a single loop or iterator to iterate through these conditions.**
4. draw **ONE** graph to show the validation accuracy curves for all 8 jobs above. This graph must have proper legends and it should be clear what we are looking at.
5. draw **ONE** graph to show the loss change curves for all 8 jobs above. This graph must have proper legends and it should be clear what we are looking at.
6. find any 25 misclassified images for "without L1/L2 with BN" AND "without L1/L2 with GBN" model. You should be using the saved model from the above jobs.
7. and L2 models. You MUST show the actual and predicted class names.
8. make all the images available on Github Readme page, so you can upload the images for your assignment (you can upload them somewhere else as well for add image url).
9. submit the Github link for your notebook with logs, and also upload the images in the S6-Assignment Solution.

Here are some of the questions for S6-Assignment-Solution:

1. Upload the Validation Accuracy Change Graph
2. Upload the Loss Change Graph
3. Upload the 25 misclassified images plot for without L1/L2 with BN
4. Upload the 25 misclassified images plot for without L1/L2 with GBN
5. Explain your observation w.r.t. L1 and L2's performance in the regularization of your model.

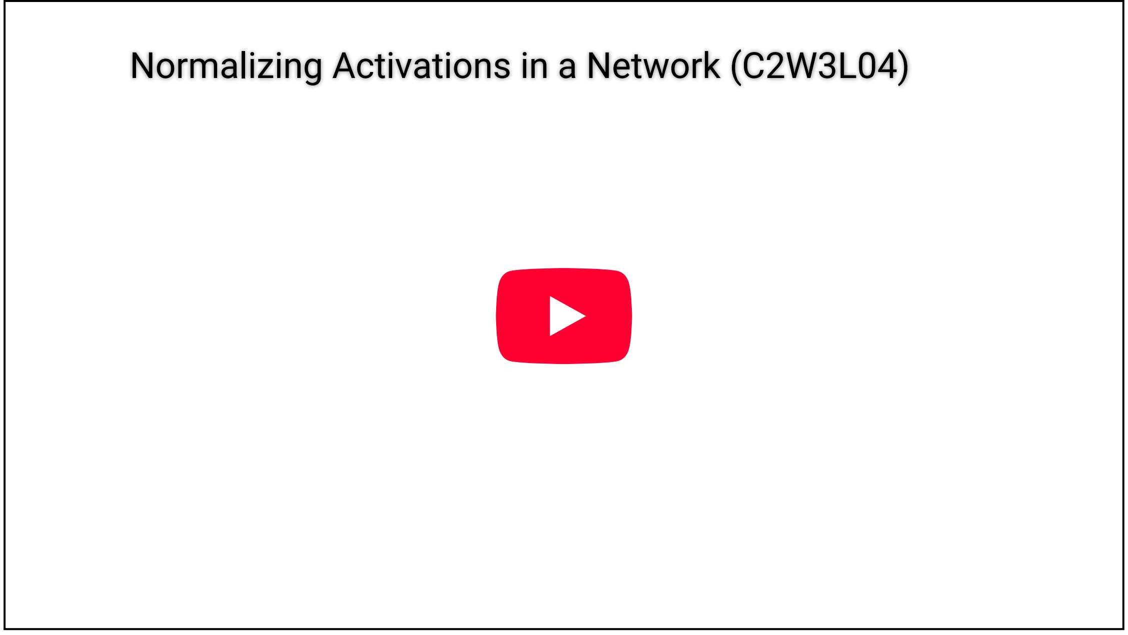
BATCH 2

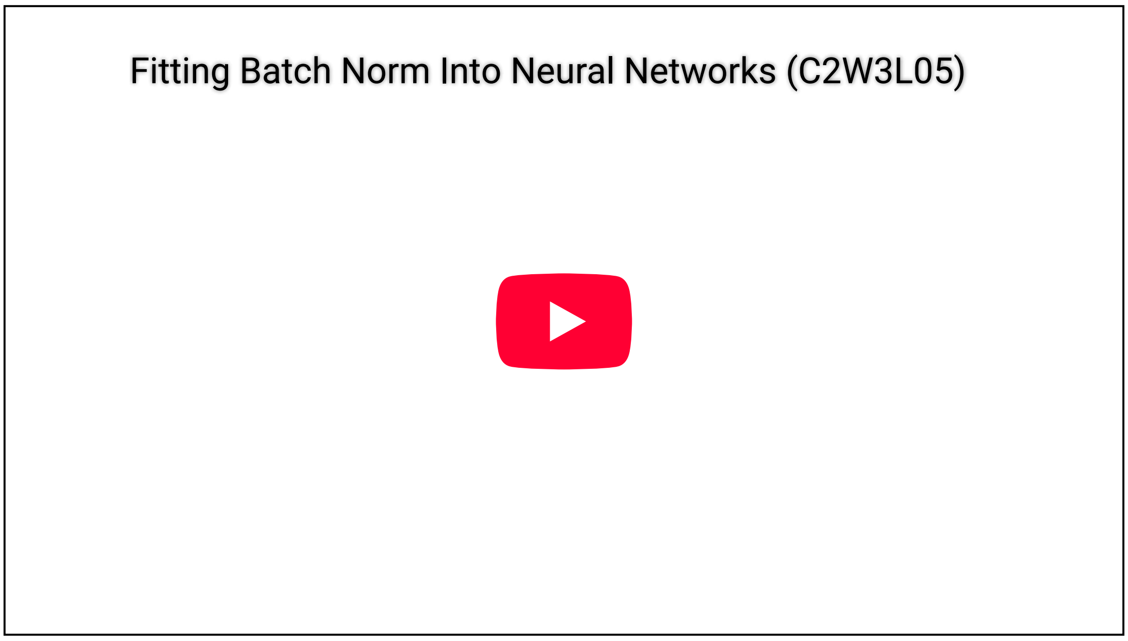
Session Video - Wednesday

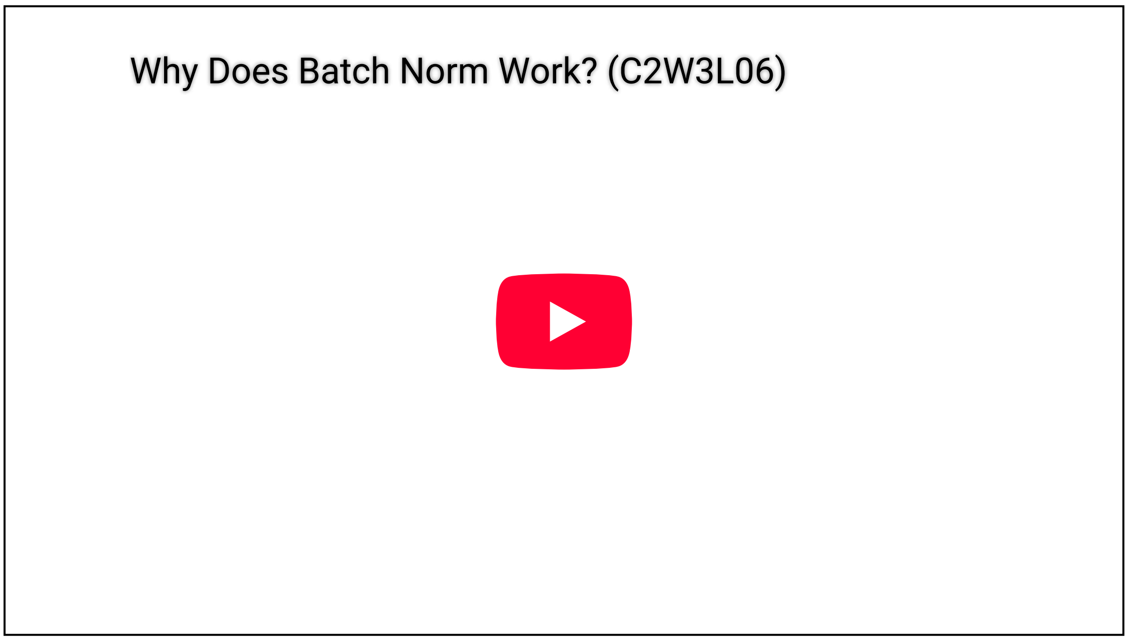


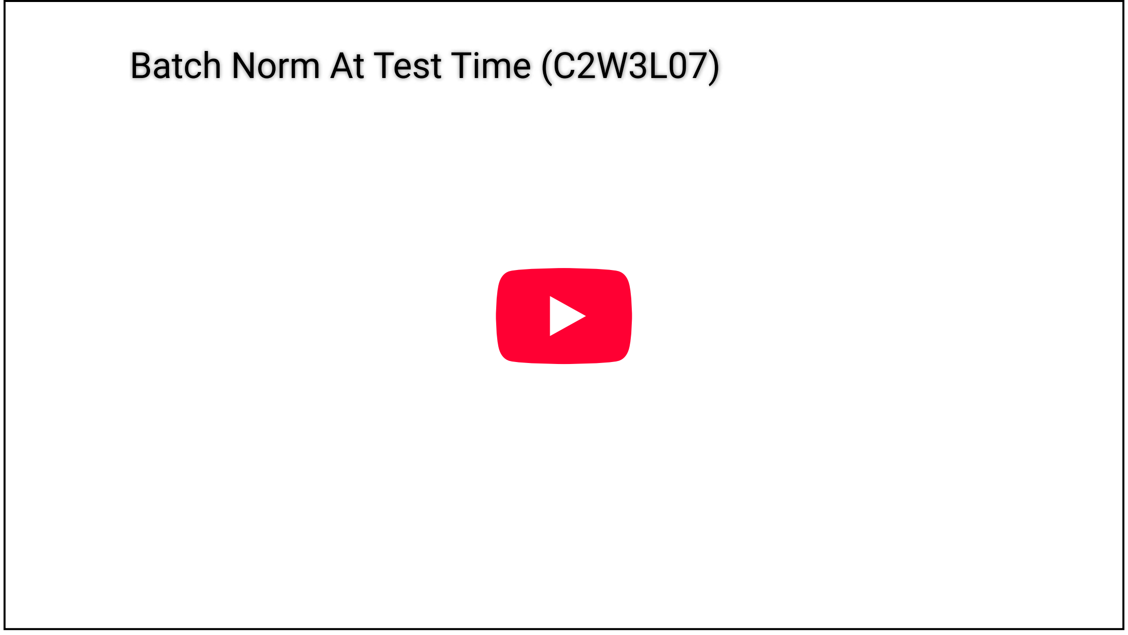
Sunday Video:

Additional References:


Normalizing Activations in a Network (C2W3L04)


Fitting Batch Norm Into Neural Networks (C2W3L05)


Why Does Batch Norm Work? (C2W3L06)


Batch Norm At Test Time (C2W3L07)

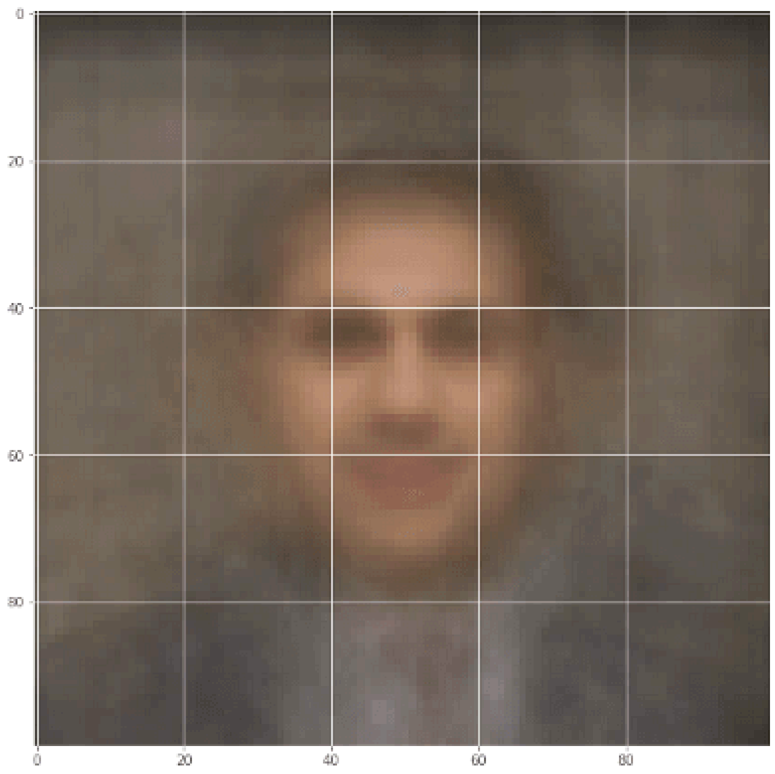
[ARCHIEVE/OBSOLETE - NOT TO BE COVERED]

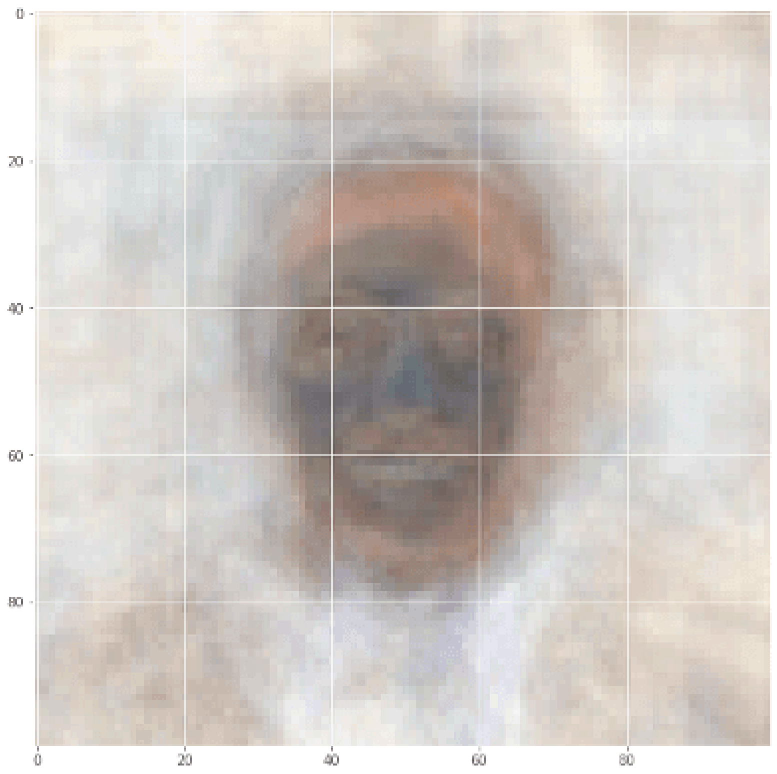
DO NOT PERFORM THIS AT HOME/WORK

THIS IS ANOTHER METHOD OF NORMALIZATION

"WHICH WE DO NOT USE"

Taking the mean (left) and standard deviation (right) of the batch, we get the following:





Our centered data resembles a face, but more importantly, our standard deviation image shows great variance along the borders and everywhere except the face. From our z-score formula, we can predict that the standardized values near the border and irrelevant areas will be relatively squashed more than values around the face. The resultant normalization appears as such:

[Excellent Source (https://datascience.stackexchange.com/questions/26881/data-preprocessing-should-we-normalise-images-pixel-wise)](https://datascience.stackexchange.com/questions/26881/data-preprocessing-should-we-normalise-images-pixel-wise)