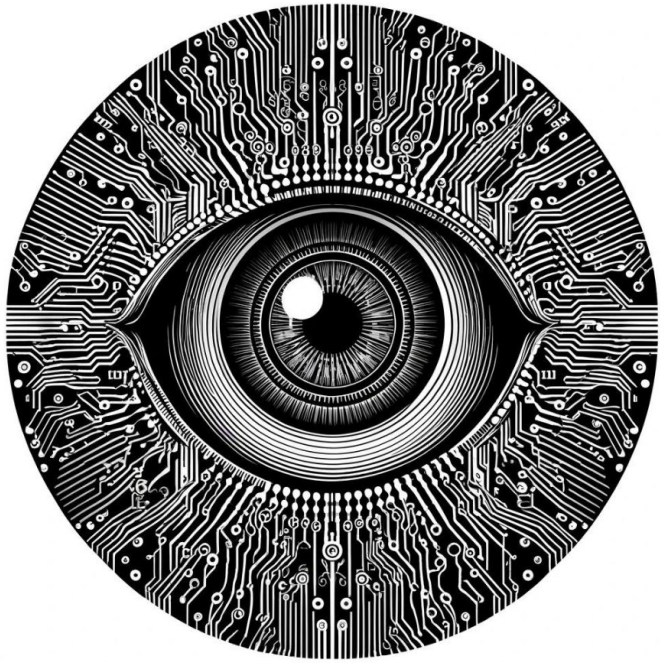
KI Service Zentrum by Hasso-Plattner-Institut HPI

Regularization with Dropout and Batch Normalization



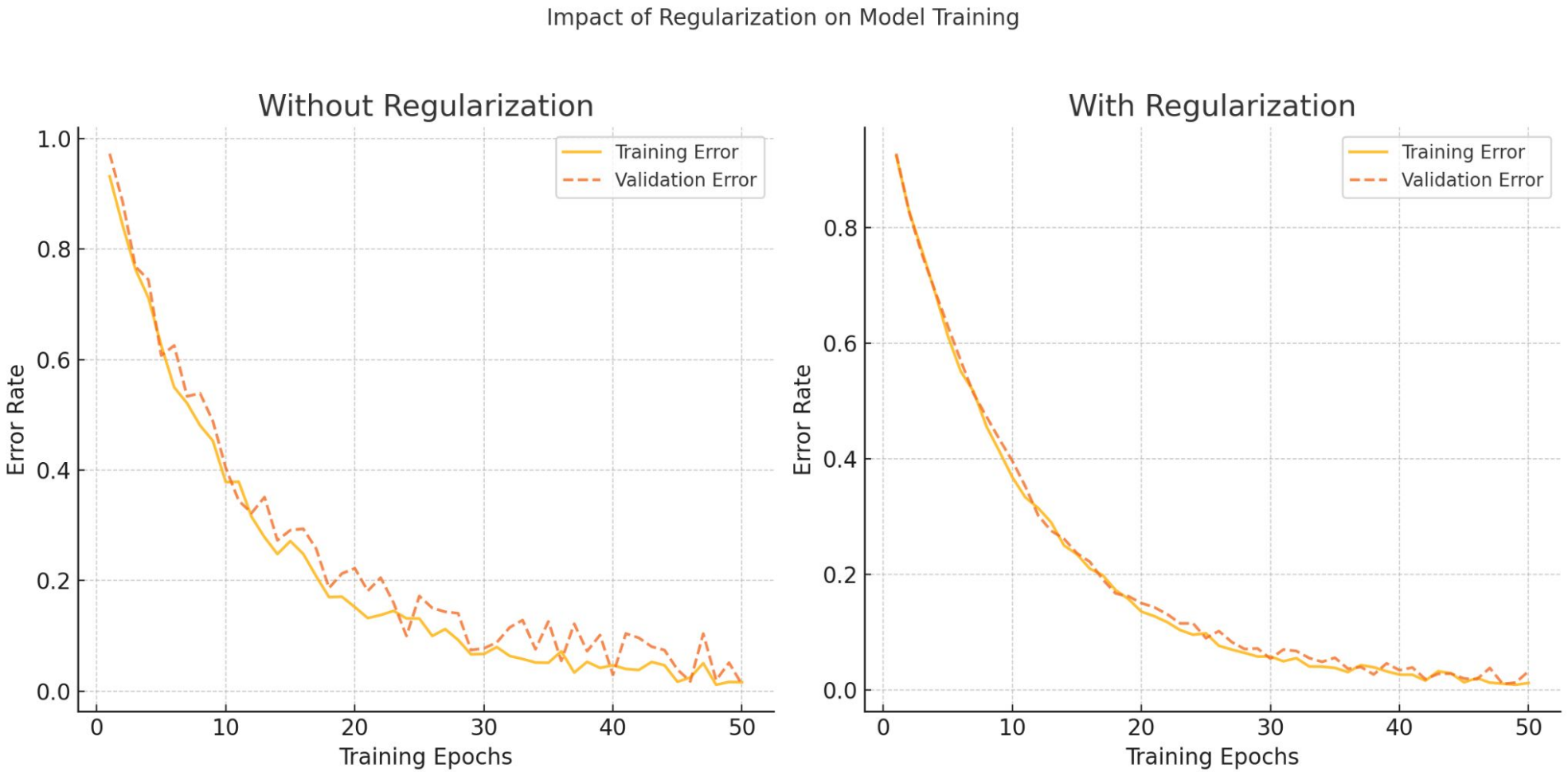
Antonio Rueda-Toicen

SPONSORED BY THE
Federal Ministry of Education and Research

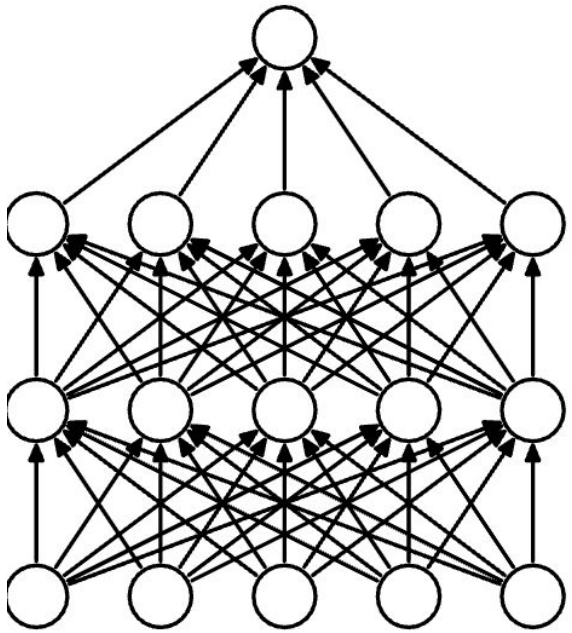
Learning goals

- Use dropout and batch normalization during the training process as regularization techniques
- Understand the behavior of batch normalization and dropout during training and inference with model.train() and model.eval()

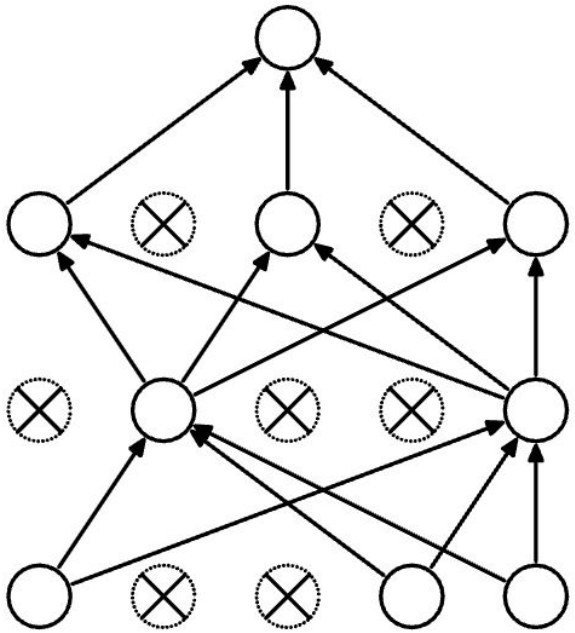
Overfitting vs underfitting - training vs validation



Dropout



(a) Standard Neural Net



(b) After applying dropout.

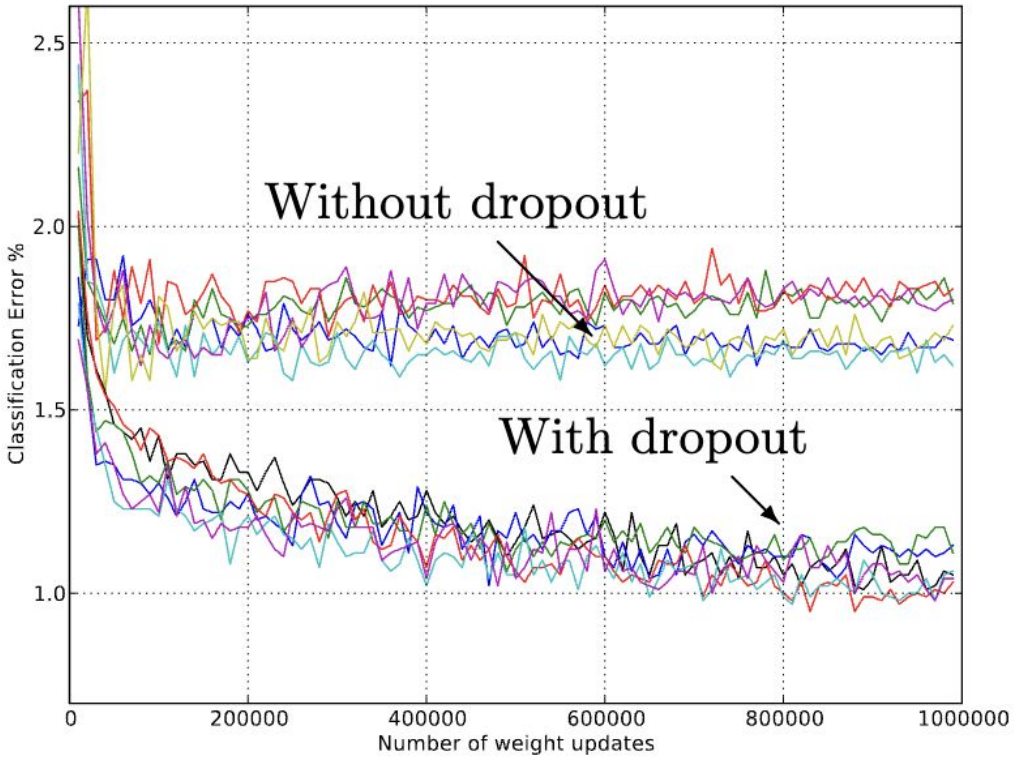
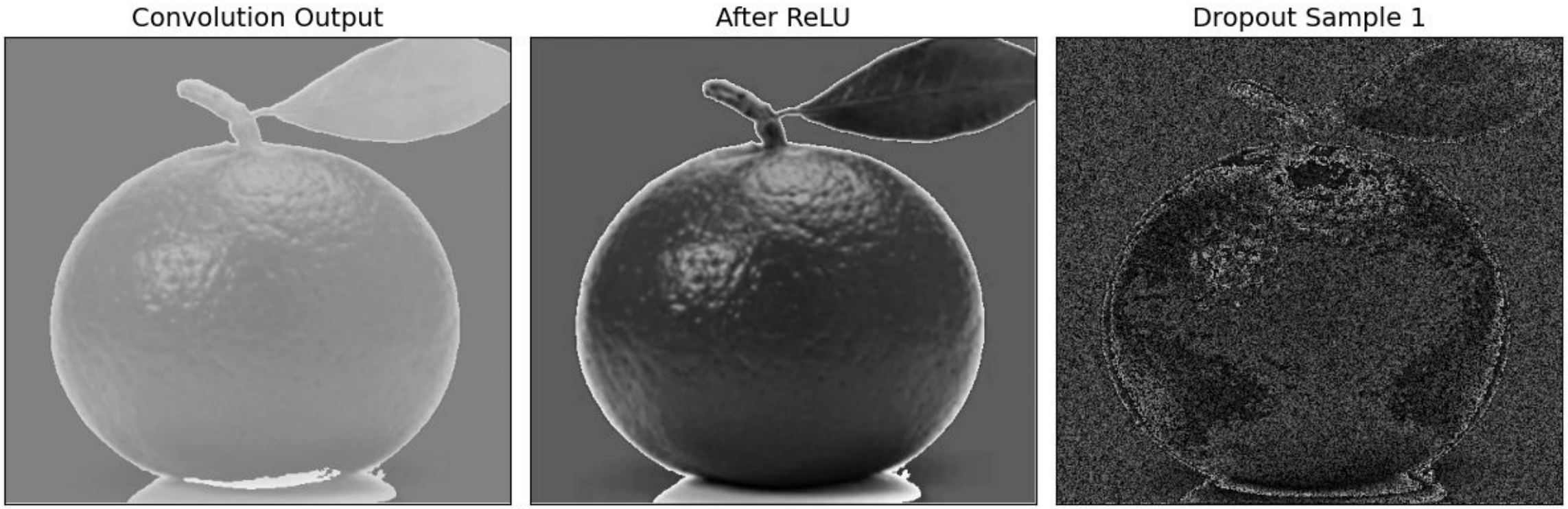


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Visualizing dropout




```
nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
    nn.ReLU(),
    nn.Dropout(p=0.5)  # Dropout after ReLU
)
```

Batch Normalization

```
import torch.nn as nn
model = nn.Sequential(
        nn.Conv2d(in_channels=3,
                  out_channels=16,
                   kernel_size=3,
                   padding=1),
        nn.BatchNorm2d(num_features=16),
        nn.ReLU(),
        )
```

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

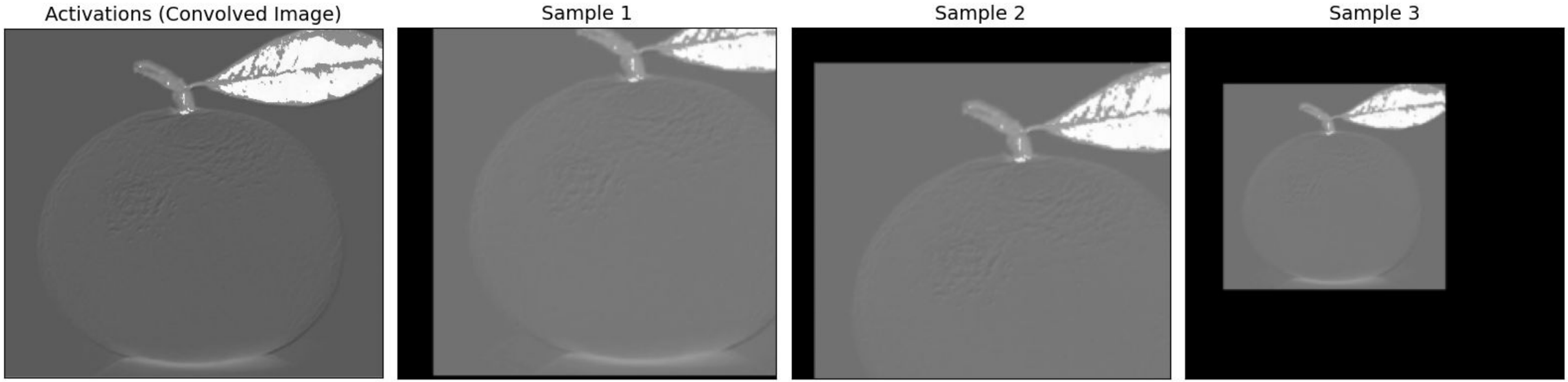
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

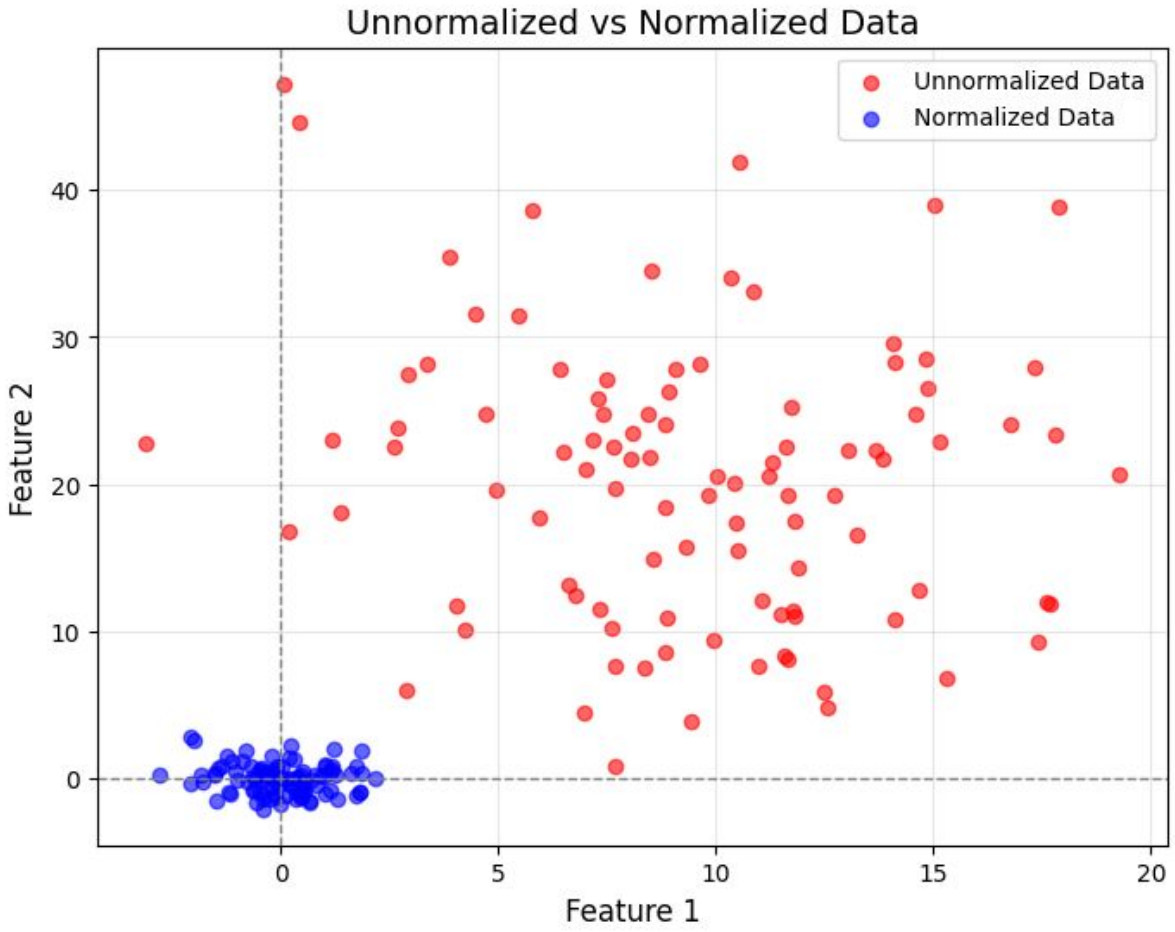
m corresponds to the batch size that we have defined in our DataLoader

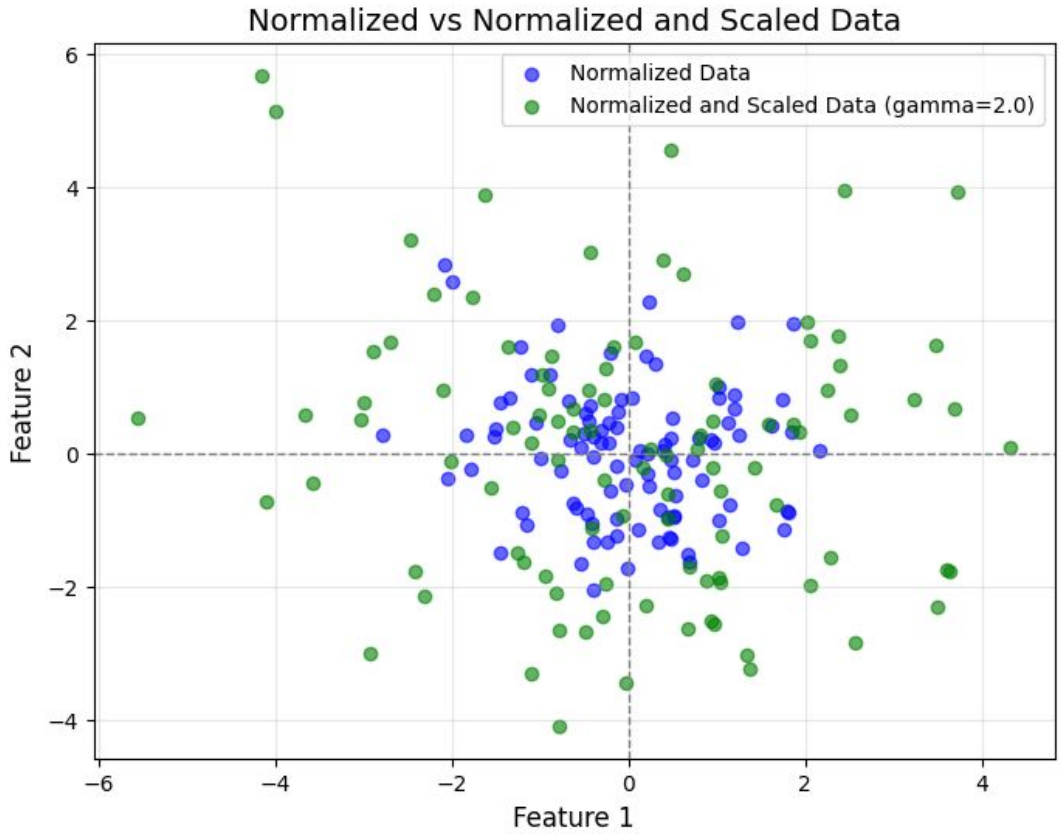
Regularization in batch normalization

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{\gamma_x}{\sqrt{\sigma_x^2+\varepsilon}} & 0 \\ 0 & \frac{\gamma_y}{\sqrt{\sigma_y^2+\varepsilon}} \end{pmatrix}}_{\text{the “scale” matrix}} \begin{pmatrix} x - \mu_x \\ y - \mu_y \end{pmatrix} + \underbrace{\begin{pmatrix} \beta_x - \frac{\gamma_x \mu_x}{\sqrt{\sigma_x^2+\varepsilon}} \\ \beta_y - \frac{\gamma_y \mu_y}{\sqrt{\sigma_y^2+\varepsilon}} \end{pmatrix}}_{\text{the “shift” vector}}.$$

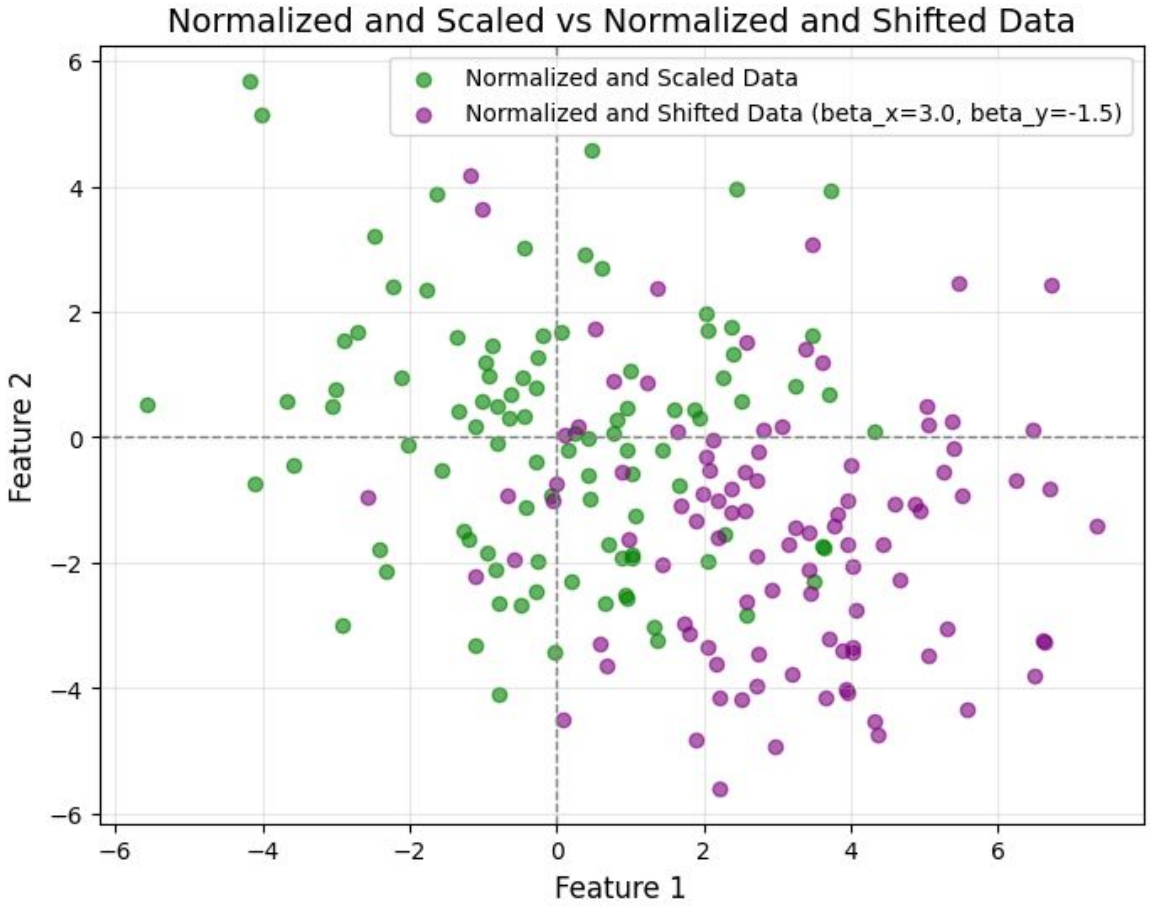


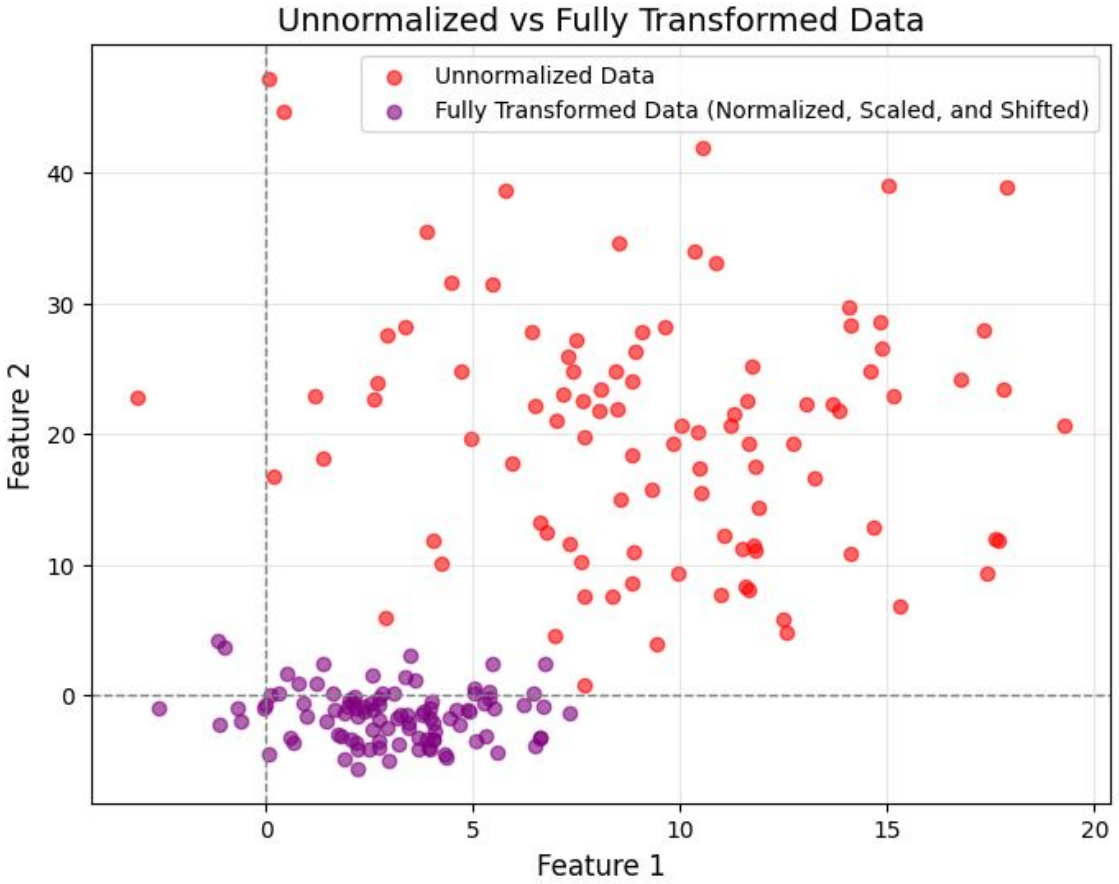
Visualizing batch normalization





Visualizing batch normalization

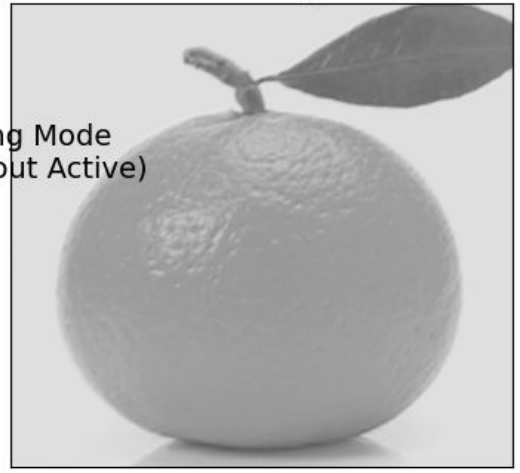




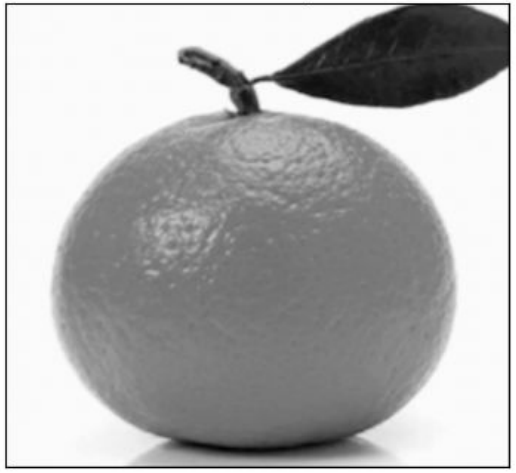
Dropout during model.train() and model.eval()

Training Mode
(Dropout Active)

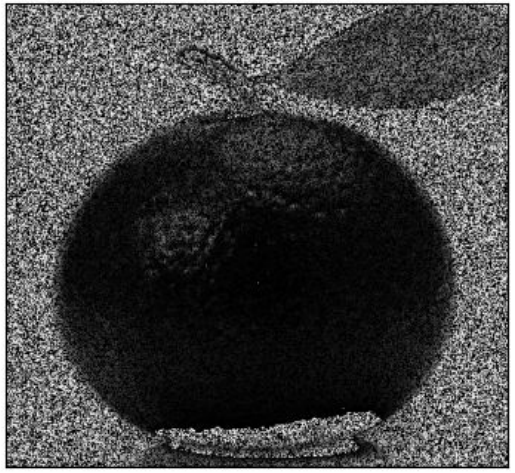
Conv Output



ReLU Output



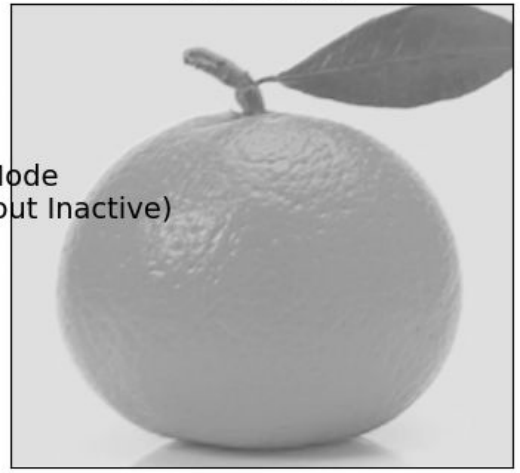
Train Mode
Dropout



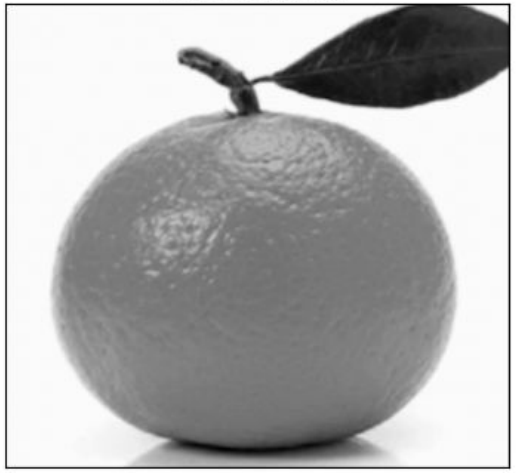
`model.train()`

Eval Mode
(Dropout Inactive)

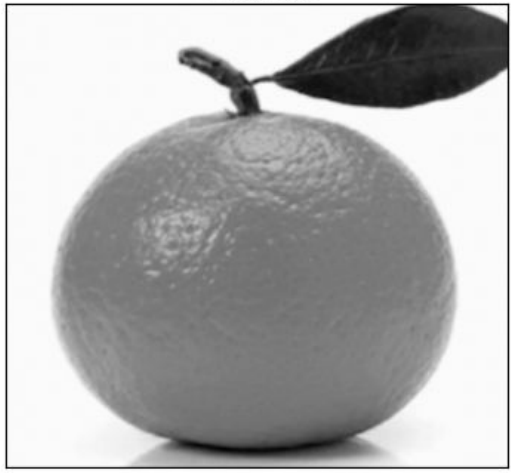
Conv Output



ReLU Output



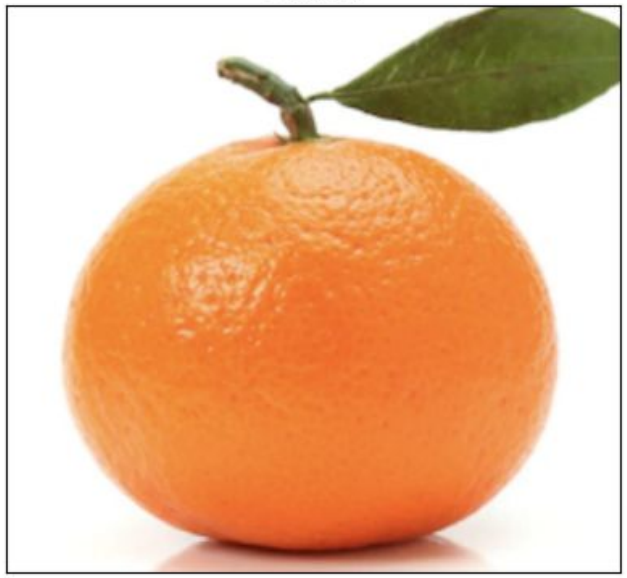
Eval Mode
Dropout



`model.eval()`

Dropout makes inference output non-deterministic

input

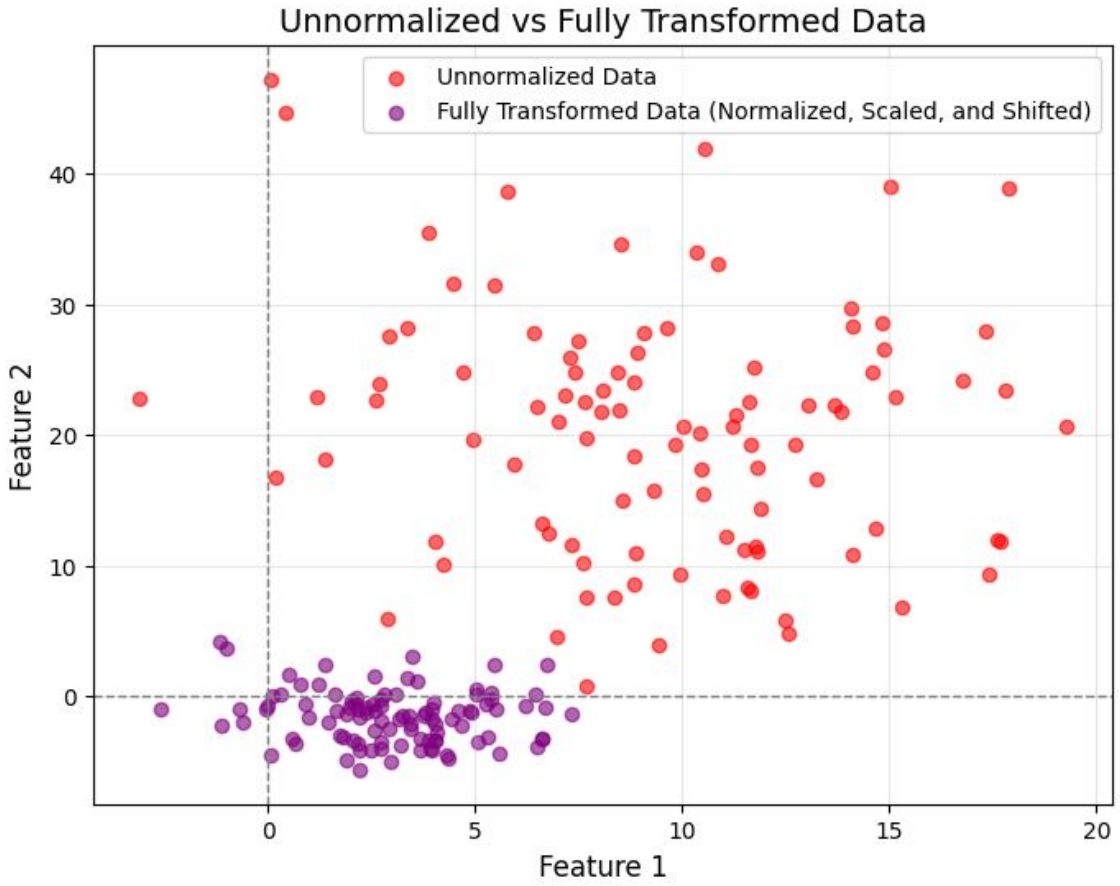


```
model.train()
```

```
orange: 0.9979          orange: 0.9703
lemon: 0.0014           lemon: 0.0151
croquet ball: 0.0002    Granny Smith: 0.0104
Granny Smith: 0.0001    pomegranate: 0.0019
banana: 0.0001          spaghetti squash: 0.0007
```

Evaluation on a pretrained VGG-16 network

BatchNorm during model.eval()



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

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

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

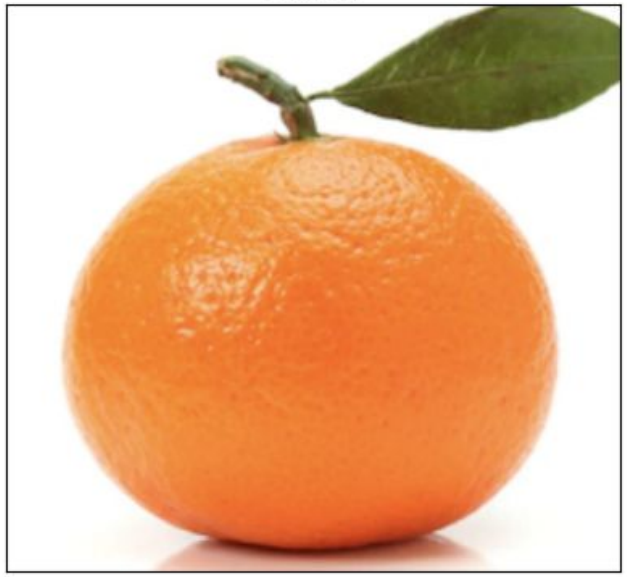
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

Effect of BatchNorm in a pretrained resnet18

input



batch size = 1

`model.eval()`

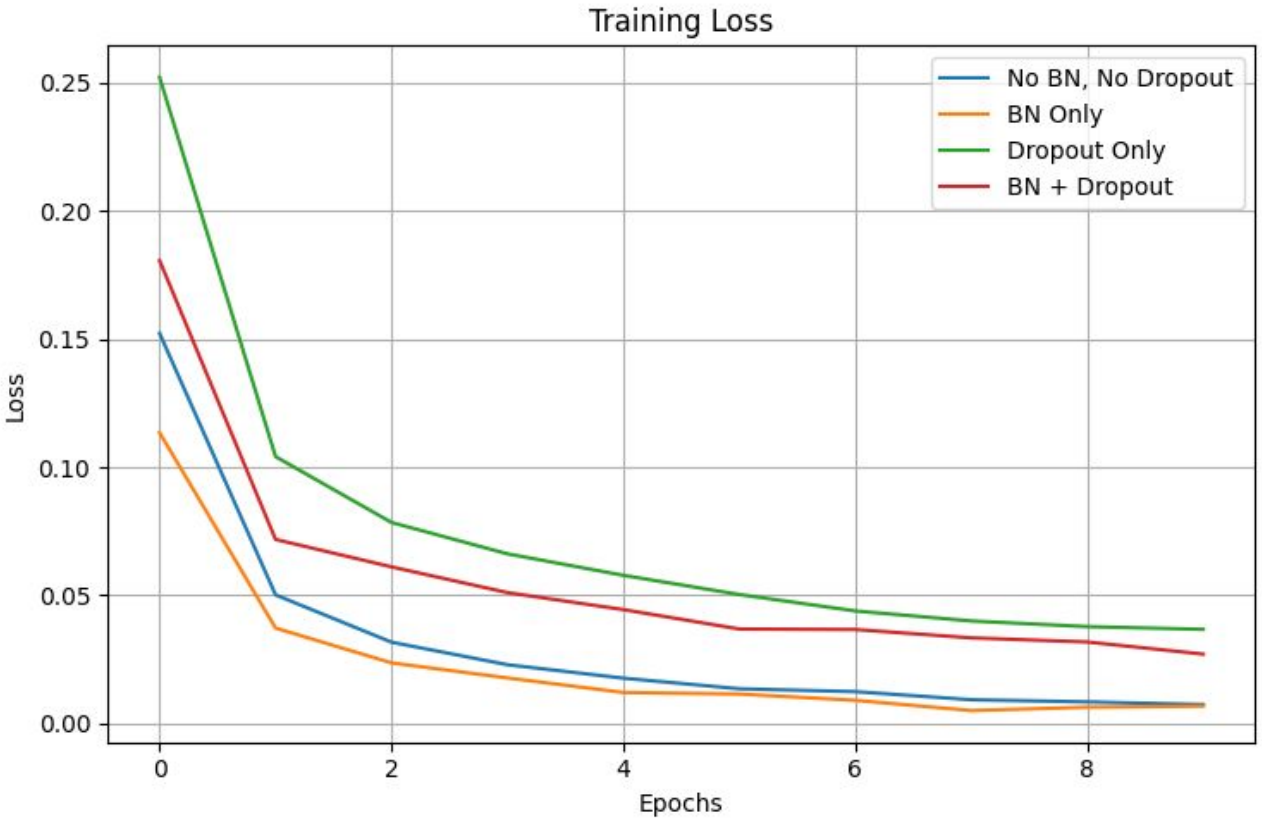
```
orange: 0.9701
lemon: 0.0286
Granny Smith: 0.0004
banana: 0.0003
pomegranate: 0.0001
```

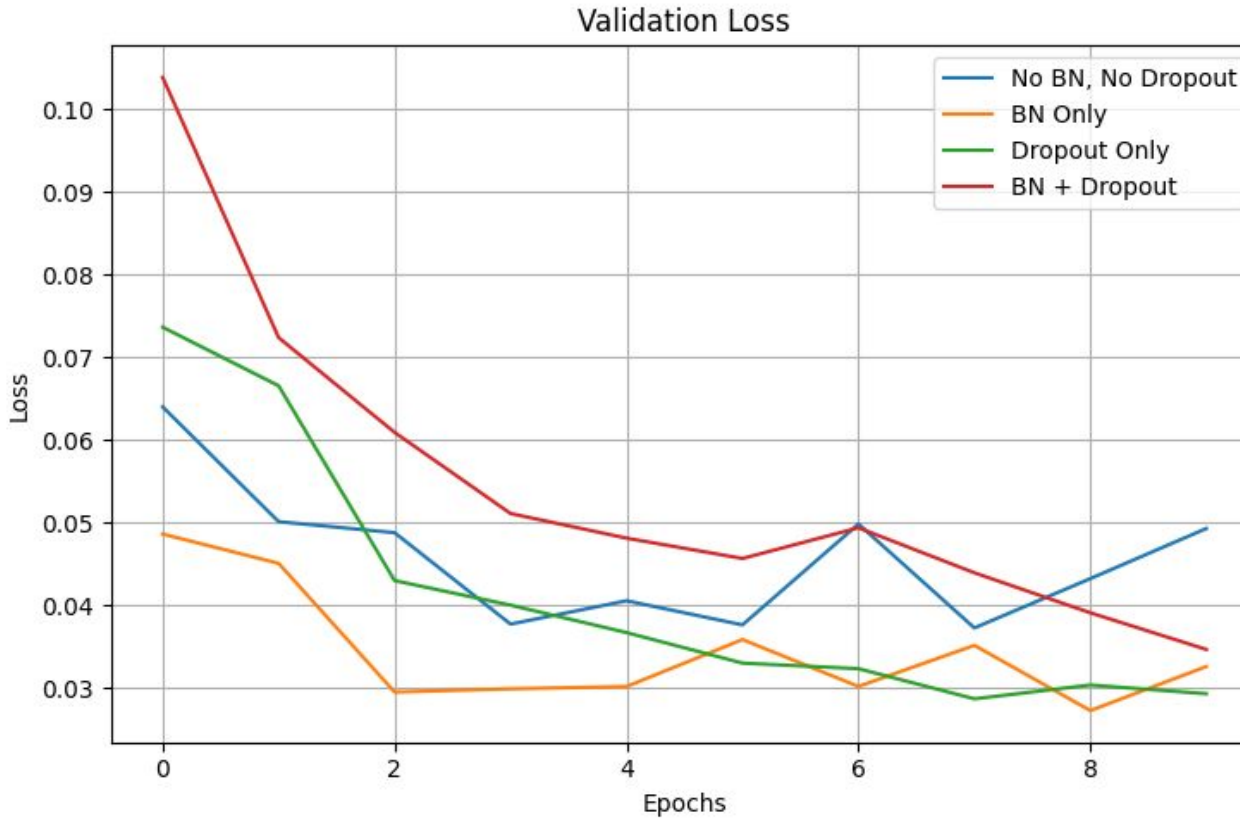
`model.train()`

```
bucket: 0.0086
plunger: 0.0065
hook: 0.0061
waste container: 0.005
ladle: 0.0051
```

Batch statistics affect the probability outputs

Do your own experiments





Colab notebook

Summary

Both dropout and batch normalization function as data augmentation

- They work on activations (e.g. convolution outputs) instead of input images

Dropout and batch normalization need to be controlled during inference

- `model.eval()` disables dropout producing deterministic output
- Batch normalization is still applied after `model.eval()`, however it uses then the mean and the standard deviation of the whole training set, instead of just the batch

Dropout and batch normalization may act against each other

- Both provide regularization effects, combining them might require careful tuning of their settings

SPONSORED BY THE
Federal Ministry of Education and Research

KI Service Zentrum by Hasso-Plattner-Institut HPI

Further reading and references

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

- https://arxiv.org/pdf/1502.03167

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

- https://jmlr.org/papers/v15/srivastava14a.html

Where to use model.eval()? (PyTorch forum discussion)

- https://discuss.pytorch.org/t/where-to-use-model-eval/89200/2

SPONSORED BY THE
Federal Ministry of Education and Research