Ministério da Ciência, Tecnologia e Inovações

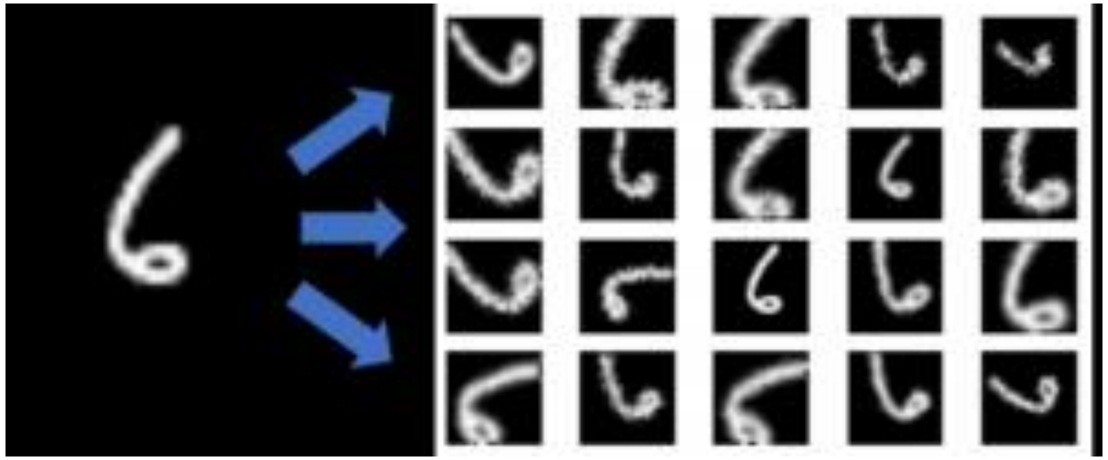
Centro Brasileiro de Pesquisas Físicas

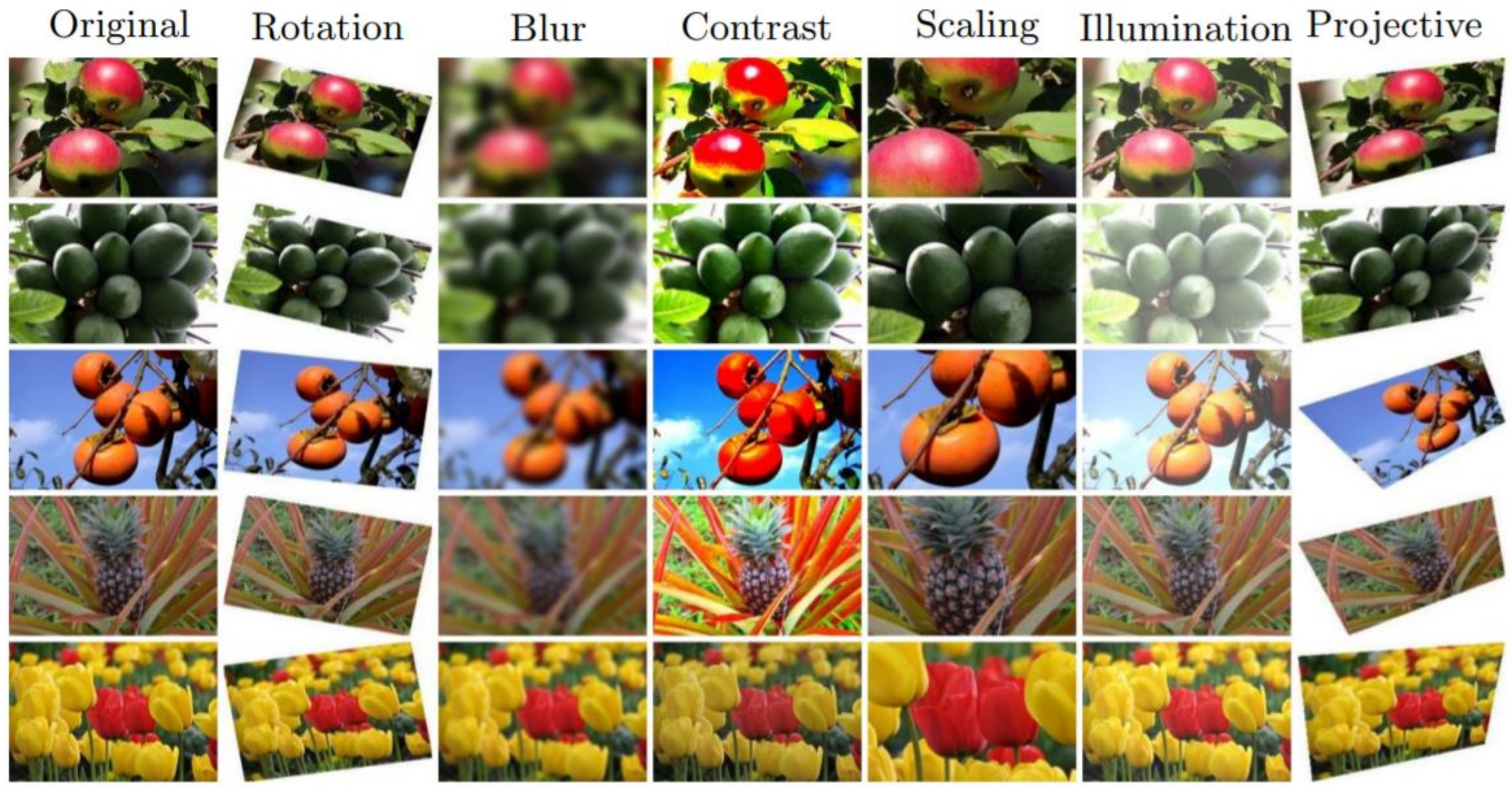
Redes Neurais profundas e aplicações Deep Learning

Clécio Roque De Bom – debom@cbpf.br

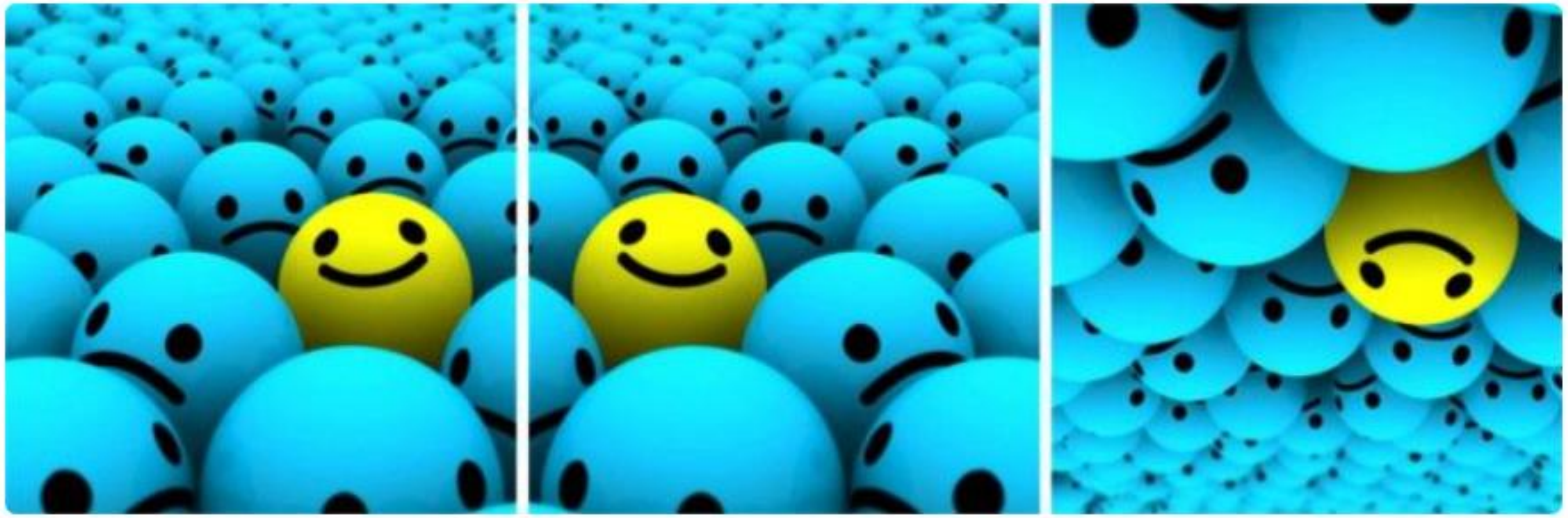
clearnightsrthebest.com

Agumentation Fun!

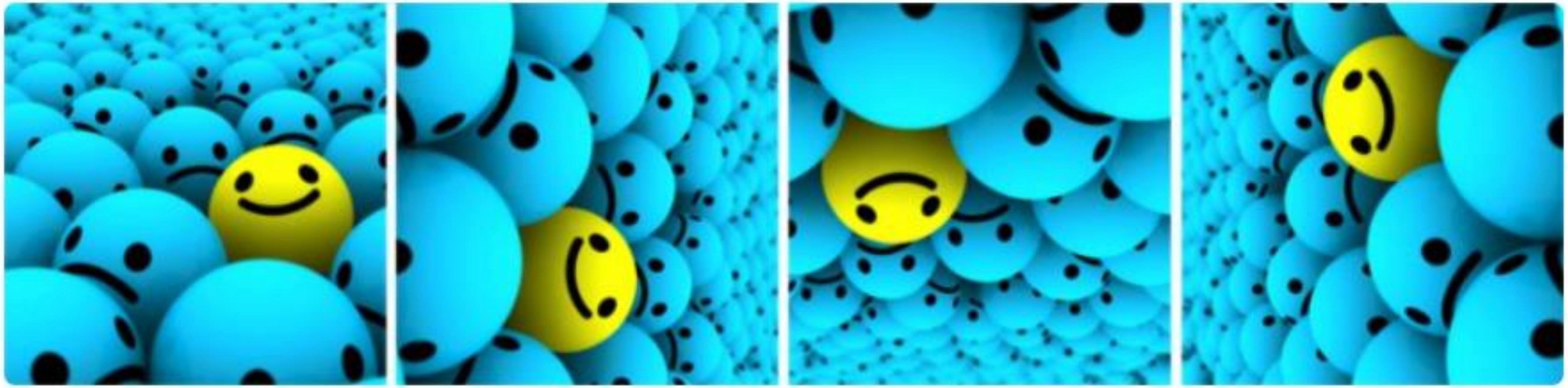




Agumentation Fun!

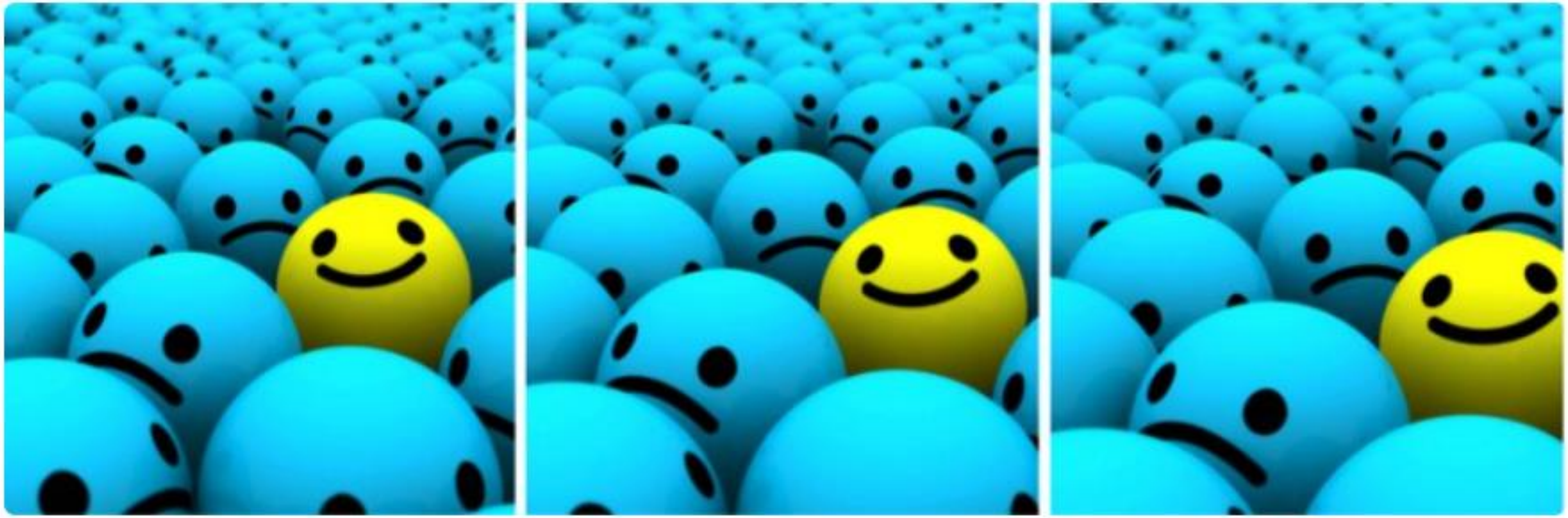

From the left, we have the original image, followed by the image flipped horizontally, and then the image flipped vertically.

Agumentation Fun!



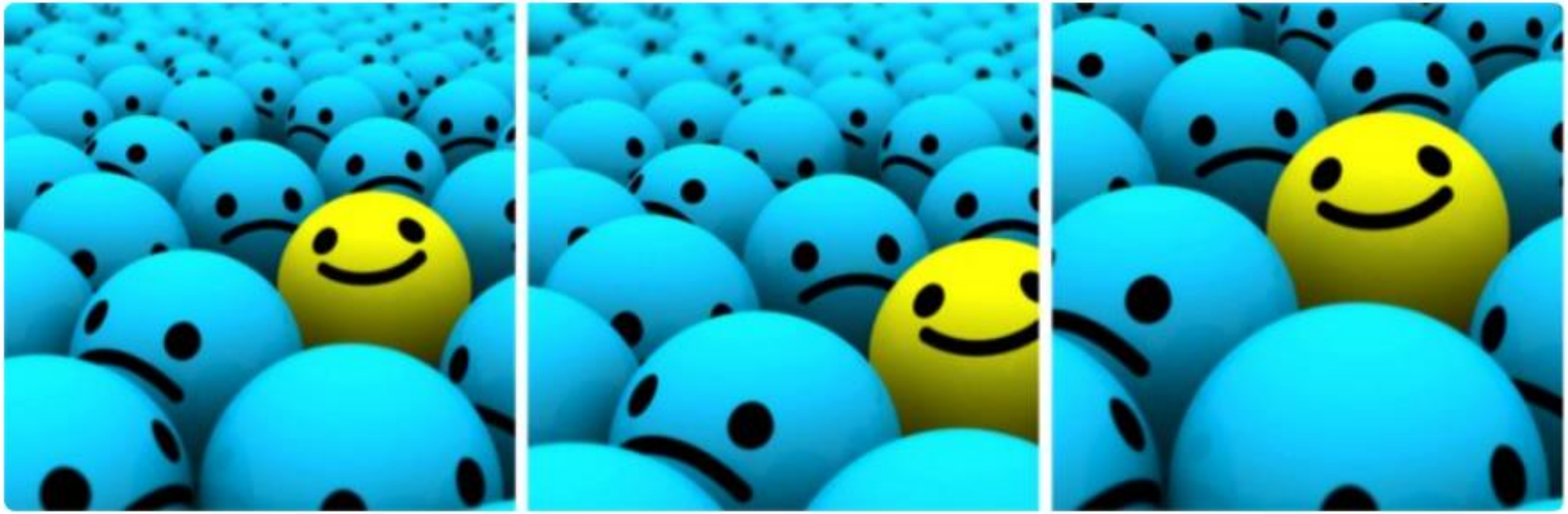
The images are rotated by 90 degrees clockwise with respect to the previous one, as we move from left to right.

Agumentation Fun!

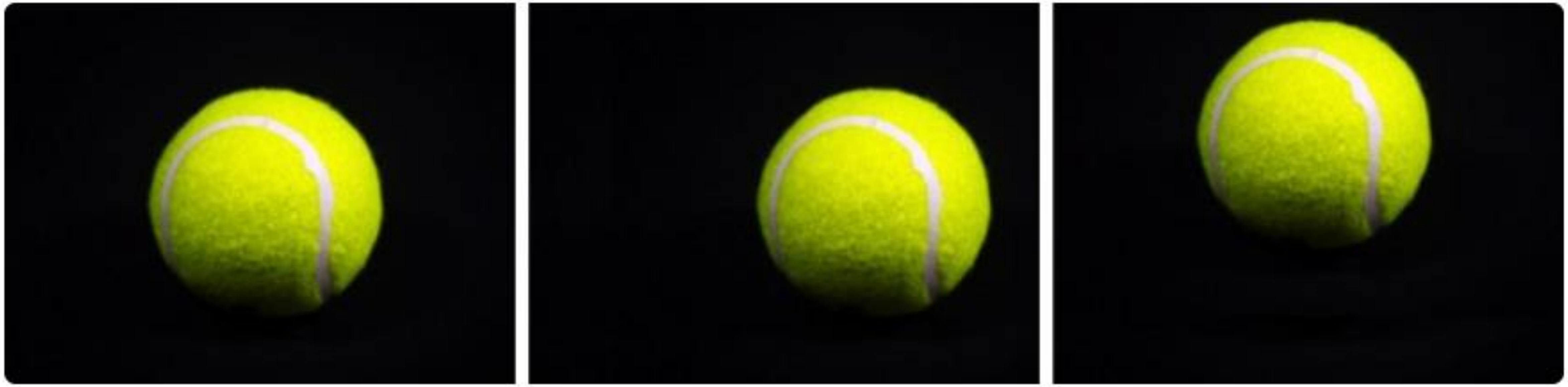


From the left, we have the original image, the image scaled outward by 10%, and the image scaled outward by 20%

Agumentation Fun!


From the left, we have the original image, a square section cropped from the top-left, and then a square section cropped from the bottom-right. The cropped sections were resized to the original image size.

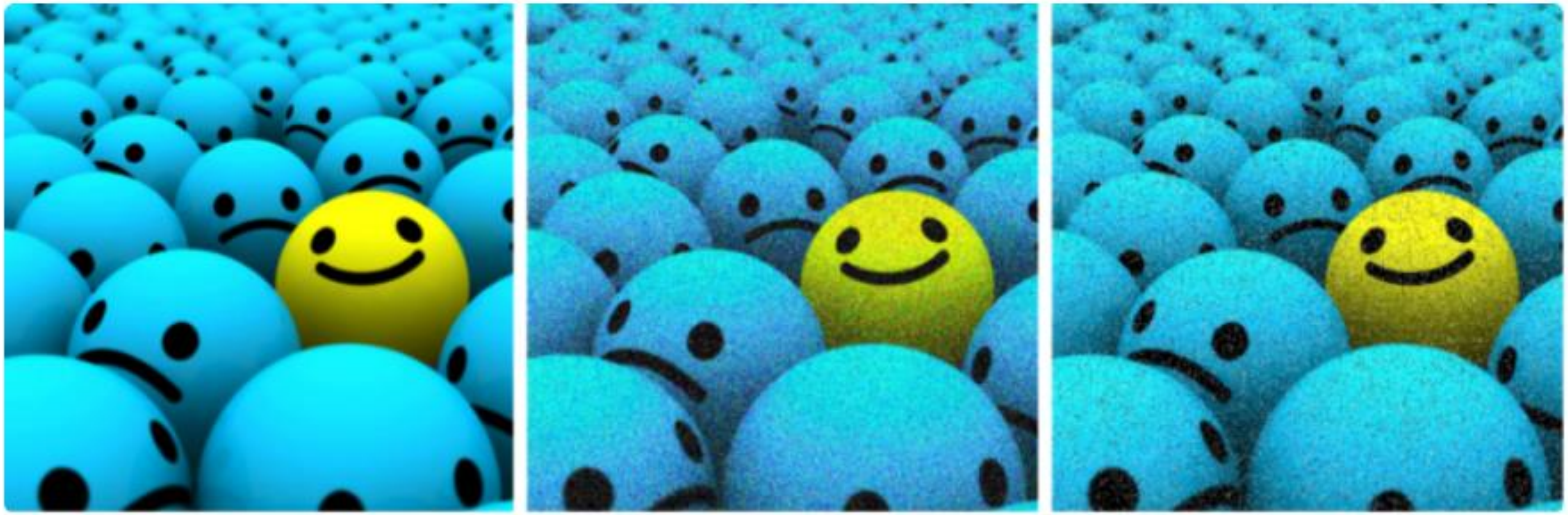
Agumentation Fun!



From the left, we have the original image, the image translated to the right, and the image translated upwards.

```
# pad_left, pad_right, pad_top, pad_bottom denote the pixel
# displacement. Set one of them to the desired value and rest to 0
shape = [batch, height, width, channels]
x = tf.placeholder(dtype = tf.float32, shape = shape)
# We use two functions to get our desired augmentation
x = tf.image.pad_to_bounding_box(x, pad_top, pad_left, height +
pad_bottom + pad_top, width + pad_right + pad_left)
output = tf.image.crop_to_bounding_box(x, pad_bottom, pad_right,
height, width)
```

Agumentation Fun!

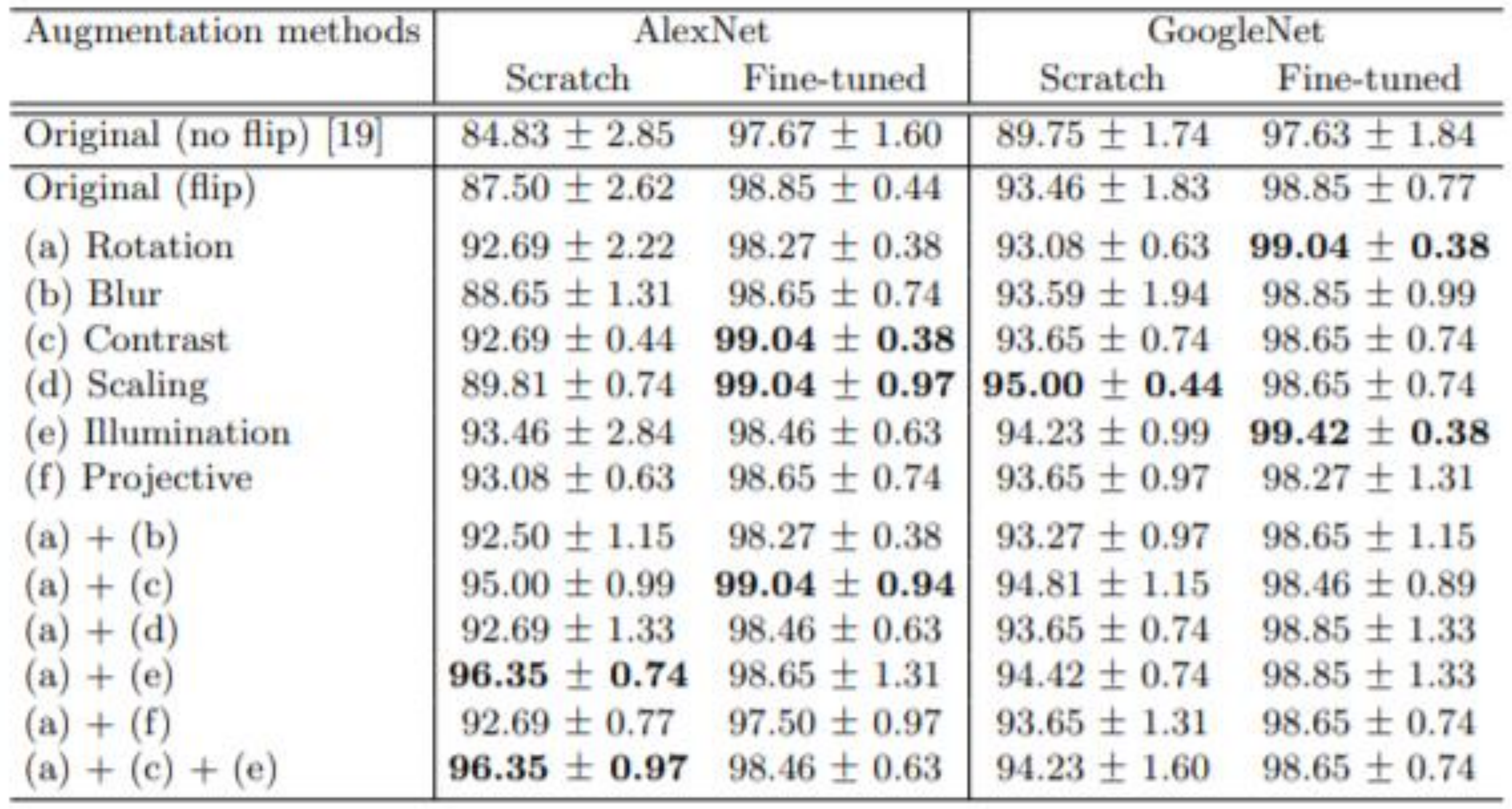


From the left, we have the original image, image with added Gaussian noise, image with added salt and pepper noise

https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

Agumentation Fun!

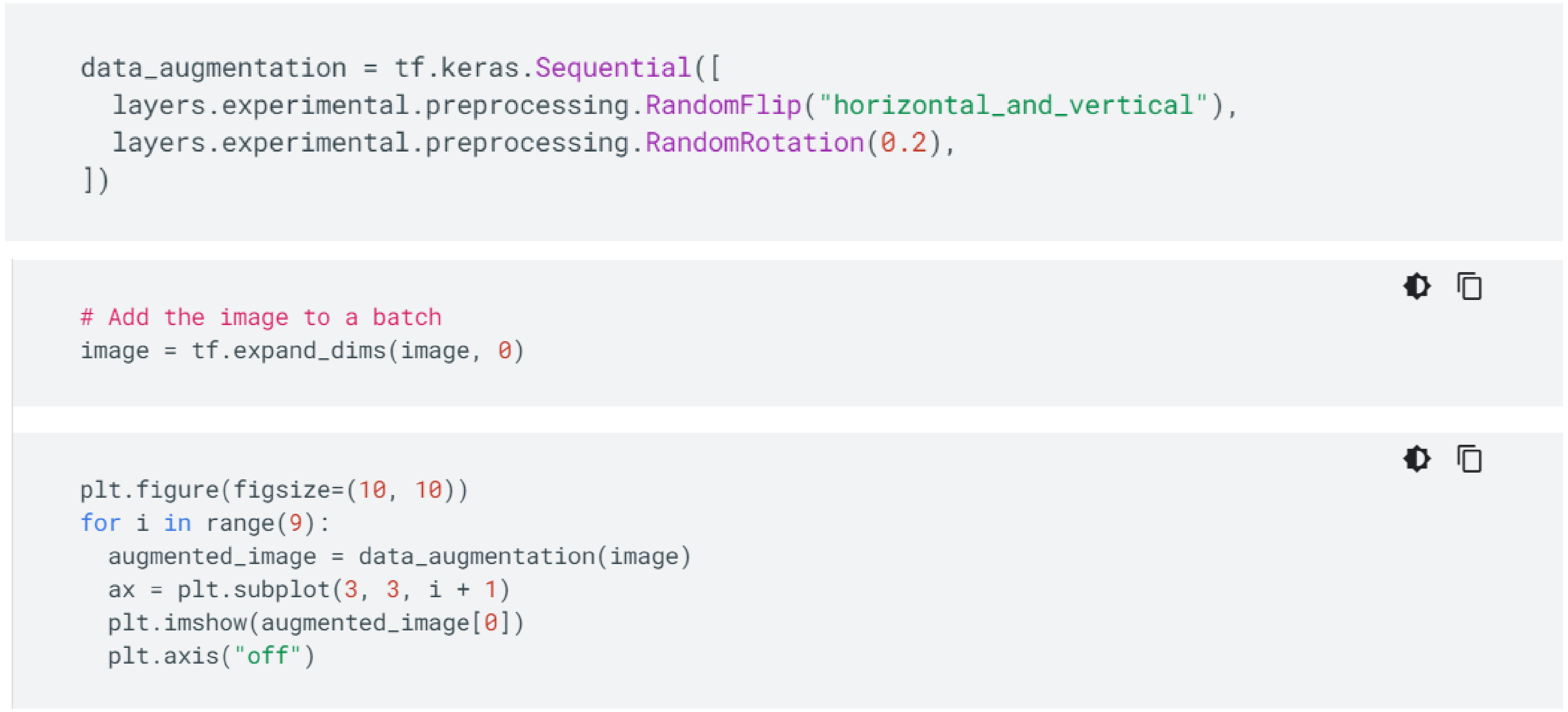
Table 2. Recognition results (accuracy and standard deviation) using different DA schemes for the Folio dataset.



Augmentation methods	AlexNet		GoogleNet	
	Scratch	Fine-tuned	Scratch	Fine-tuned
Original (no flip) [19]	84.83 ± 2.85	97.67 ± 1.60	89.75 ± 1.74	97.63 ± 1.84
Original (flip)	87.50 ± 2.62	98.85 ± 0.44	93.46 ± 1.83	98.85 ± 0.77
(a) Rotation	92.69 ± 2.22	98.27 ± 0.38	93.08 ± 0.63	**99.04 ± 0.38**
(b) Blur	88.65 ± 1.31	98.65 ± 0.74	93.59 ± 1.94	98.85 ± 0.99
(c) Contrast	92.69 ± 0.44	**99.04 ± 0.38**	93.65 ± 0.74	98.65 ± 0.74
(d) Scaling	89.81 ± 0.74	**99.04 ± 0.97**	**95.00 ± 0.44**	98.65 ± 0.74
(e) Illumination	93.46 ± 2.84	98.46 ± 0.63	94.23 ± 0.99	**99.42 ± 0.38**
(f) Projective	93.08 ± 0.63	98.65 ± 0.74	93.65 ± 0.97	98.27 ± 1.31
(a) + (b)	92.50 ± 1.15	98.27 ± 0.38	93.27 ± 0.97	98.65 ± 1.15
(a) + (c)	95.00 ± 0.99	**99.04 ± 0.94**	94.81 ± 1.15	98.46 ± 0.89
(a) + (d)	92.69 ± 1.33	98.46 ± 0.63	93.65 ± 0.74	98.85 ± 1.33
(a) + (e)	**96.35 ± 0.74**	98.65 ± 1.31	94.42 ± 0.74	98.85 ± 1.33
(a) + (f)	92.69 ± 0.77	97.50 ± 0.97	93.65 ± 1.31	98.65 ± 0.74
(a) + (c) + (e)	**96.35 ± 0.97**	98.46 ± 0.63	94.23 ± 1.60	98.65 ± 0.74

Data Augmentation for Plant Classification
Pornntiwa Pawara, Emmanuel Okafor, Lambert Schomaker, and Marco Wiering

Agumentation Fun!



```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

```
# Add the image to a batch
image = tf.expand_dims(image, 0)
```

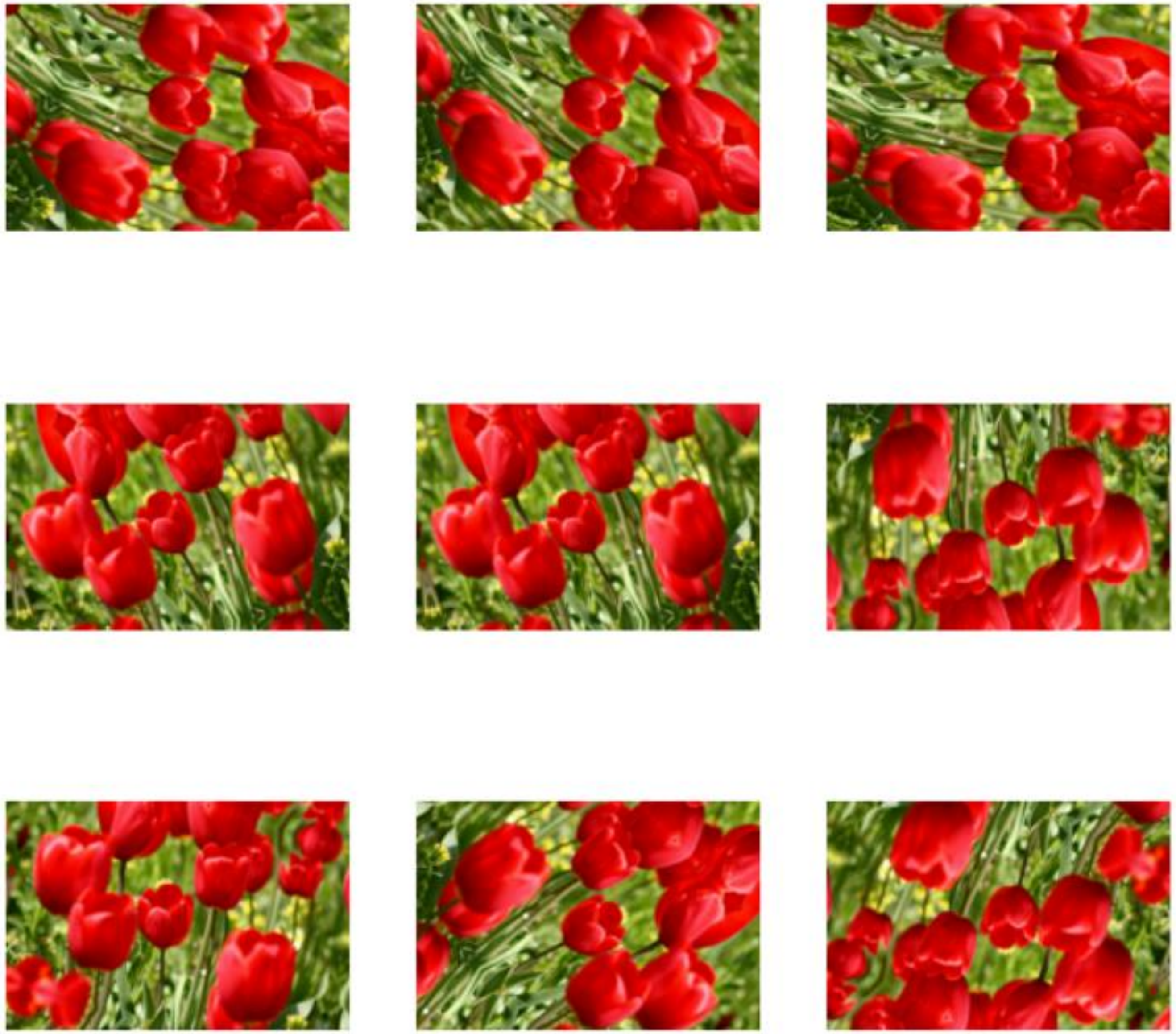
```
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
```

Agumentation Fun!

```
data_
  lay
  lay
])
```

```
# Add
image
```

```
plt.fi
for i
  augm
  ax =
  plt.
  plt.
```

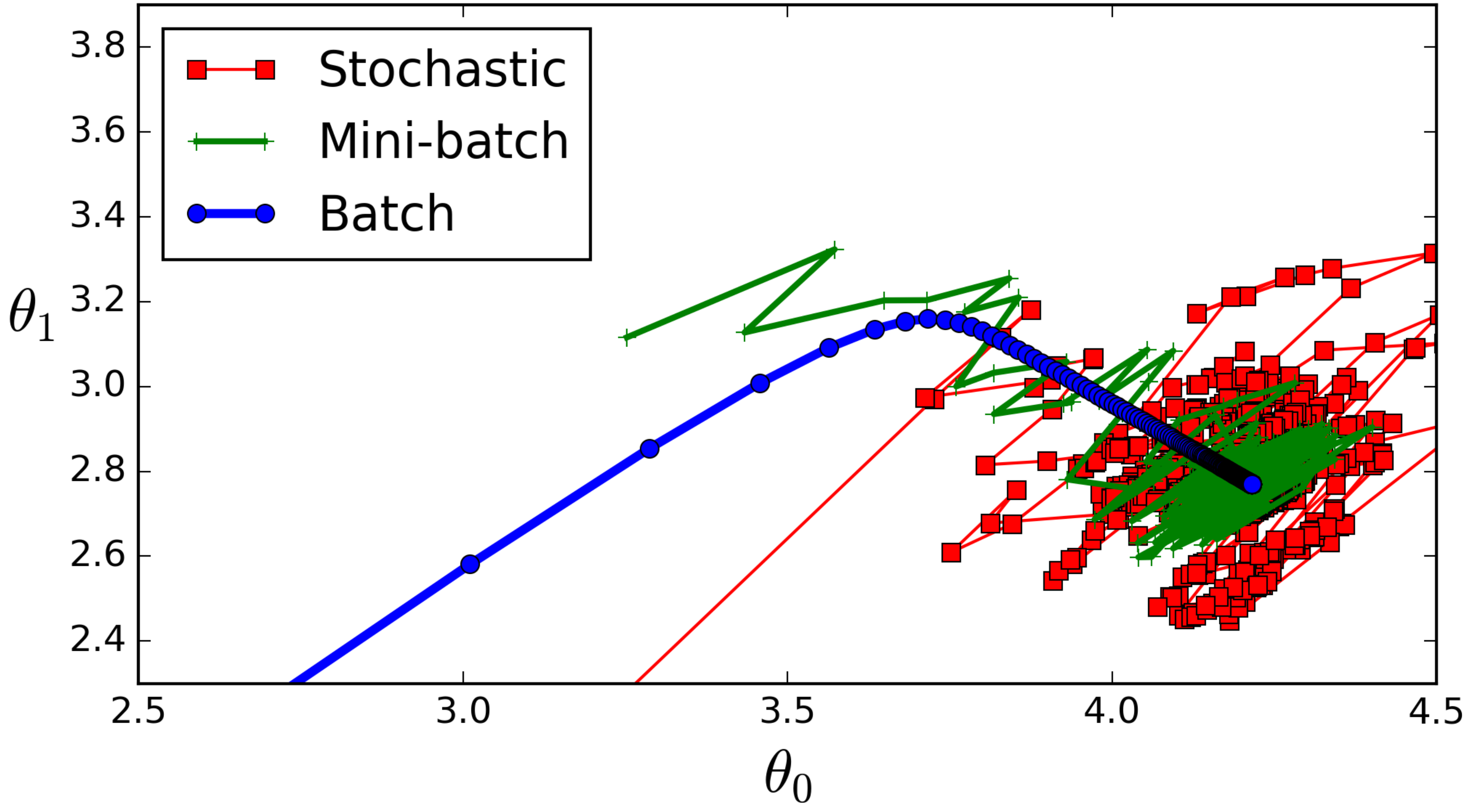


Agumentation Fun!

```
# carregar a imagem original e converter em array
image = load_img(IMAGE_PATH)
image = img_to_array(image)

# adicionar uma dimensão extra no array
image = np.expand_dims(image, axis=0)

# criar um gerador (generator) com as imagens do
# data augmentation
imgAug = ImageDataGenerator(rotation_range=45, width_shift_range=0.1,
                            height_shift_range=0.1, zoom_range=0.25,
                            fill_mode='nearest', horizontal_flip=True)
imgGen = imgAug.flow(image, save_to_dir=OUTPUT_PATH,
                     save_format='jpg', save_prefix='t27_')
```

Batch Size

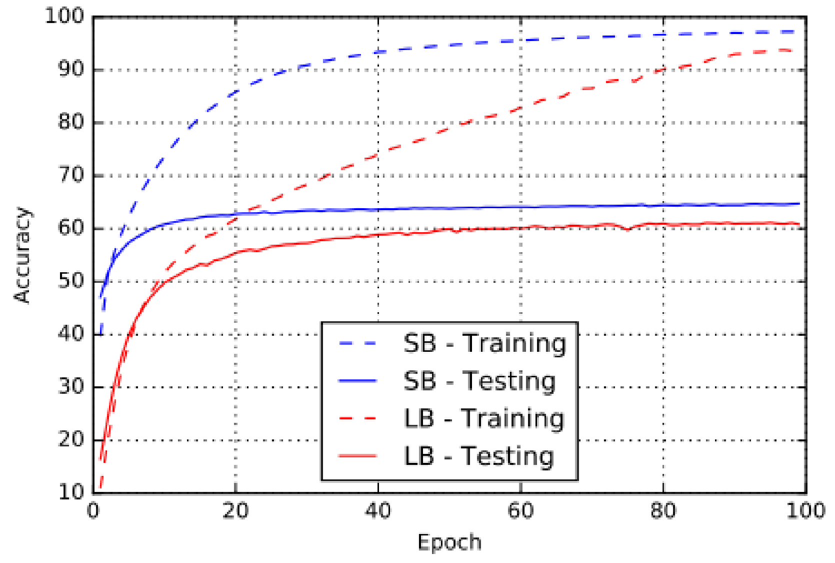
Computational speed X Speed of convergence. Consider two parameters



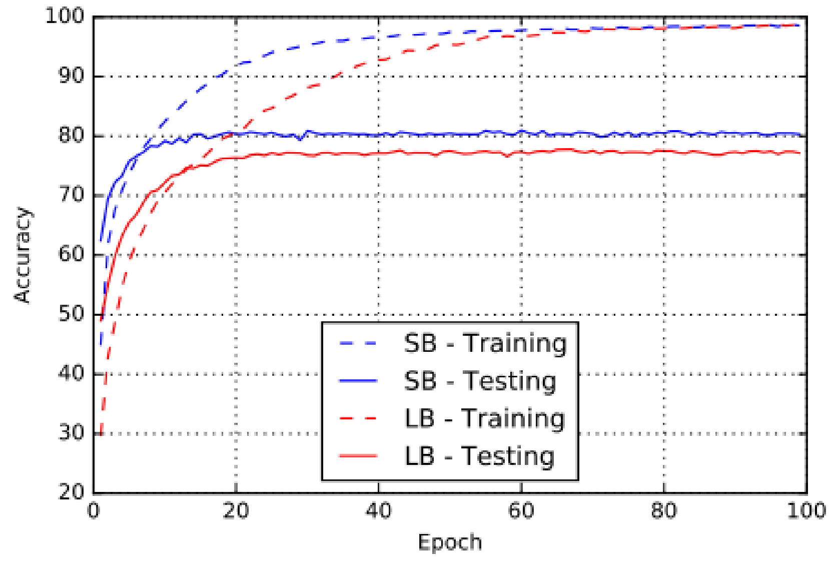
https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

Batch Size

Shall we go for the biggest possible Batch? Consider Two architectures F2 and C1, in a LARGE dataset



(a) F_2



(b) C_1

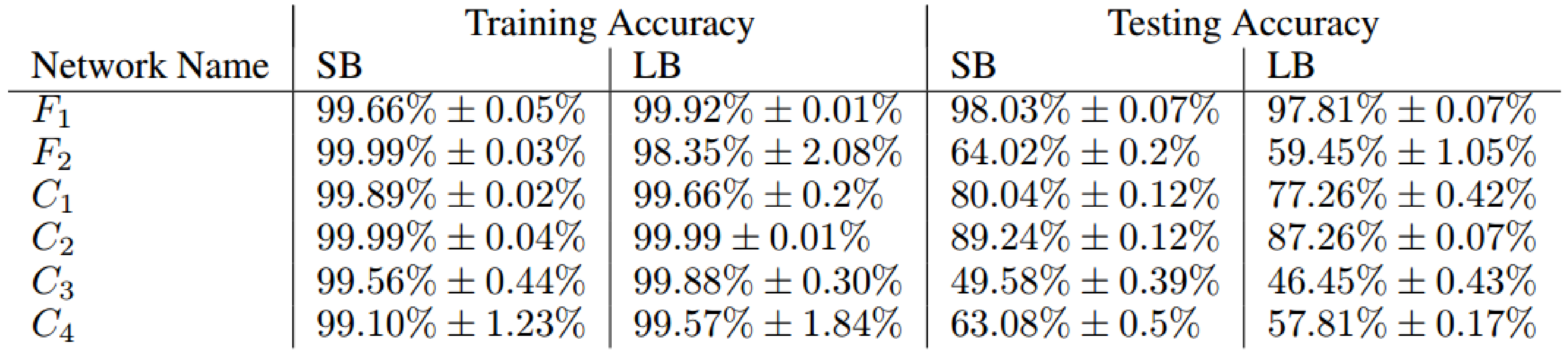
Bigger Batches, more memory it can converge in few epochs, smaller Batches more updates in the Net.

Computational speed X Speed of convergence

https://arxiv.org/abs/1609.04836

Batch Size

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1



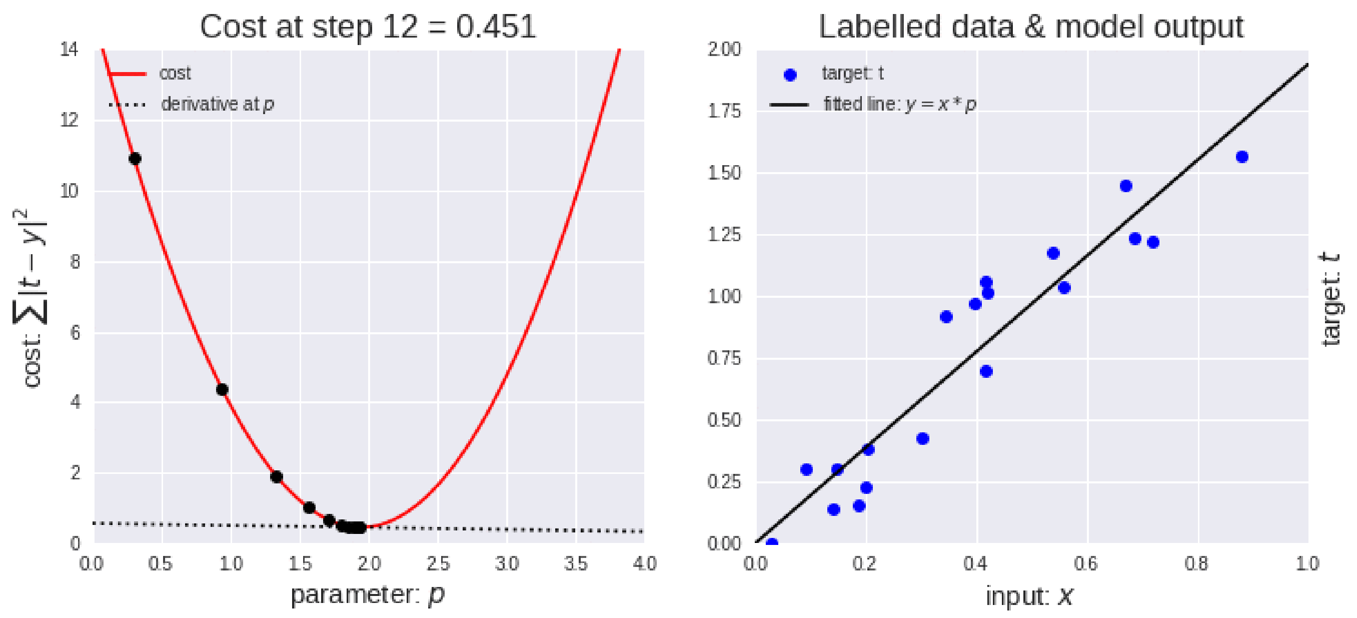
	Training Accuracy		Testing Accuracy	
Network Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99 \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$

The stochastic gradient descent method and its variants are algorithms of choice for many Deep Learning tasks. These methods operate in a small-batch regime wherein a fraction of the training data, usually 32--512 data points, is sampled to compute an approximation to the gradient. **It has been observed in practice that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize.** (https://arxiv.org/abs/1609.04836).

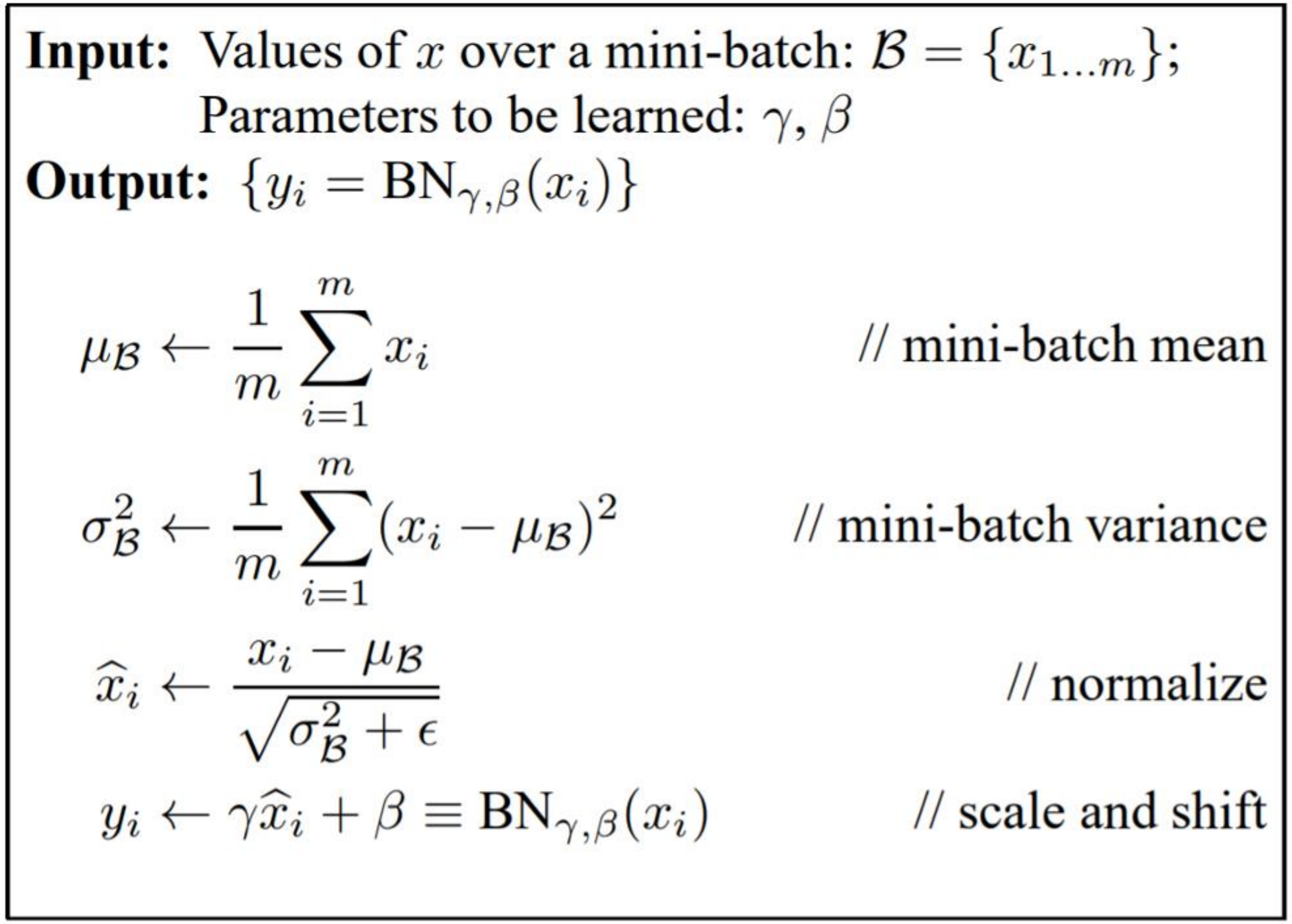
https://arxiv.org/abs/1609.04836

Batch Size

A reasonable choice in the size of the learning rate depends on how curved the cost function is. You can think of gradient descent as making a linear approximation to the cost function. If you move downhill along that approximate cost and If the cost function is highly non-linear then the approximation will not work well.

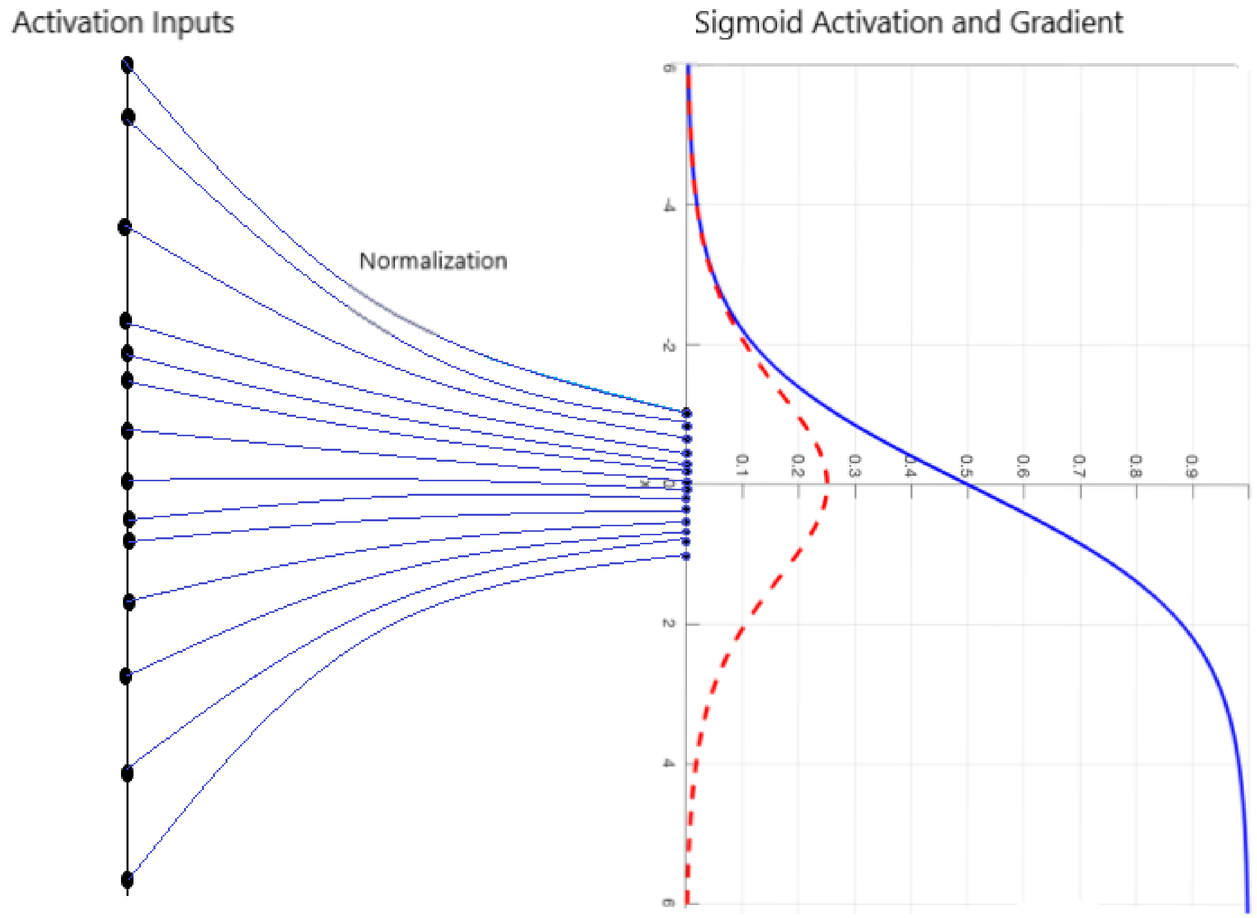


Batch Normalization



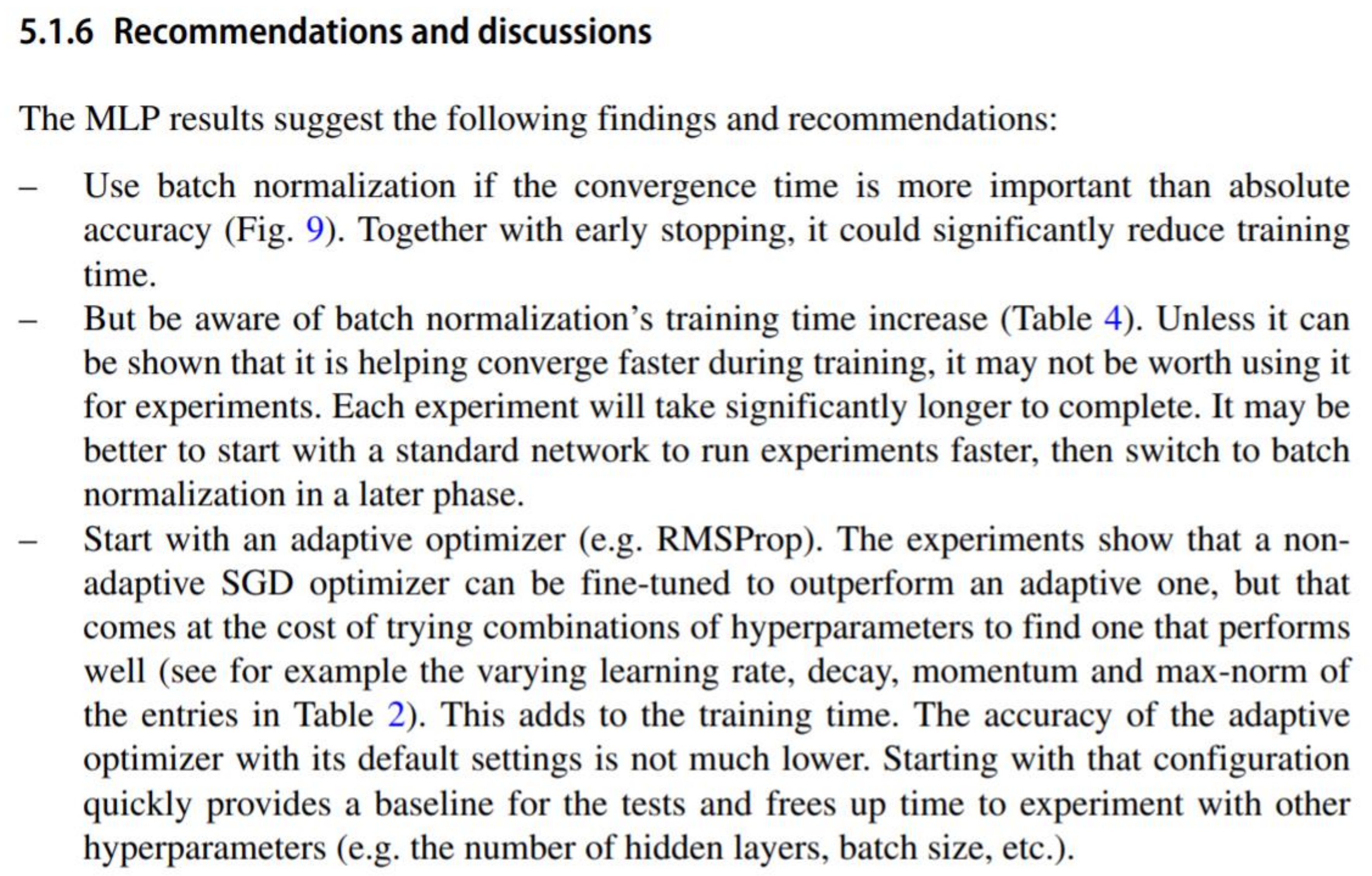
```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$    // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$    // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$    // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$    // scale and shift
```

Batch Normalization



“We presented an algorithm for constructing, training, and performing inference with batch-normalized networks. The resulting networks can be trained with saturating nonlinearities, are more tolerant to increased training rates, and often do not require Dropout for regularization.”

Batch Normalization



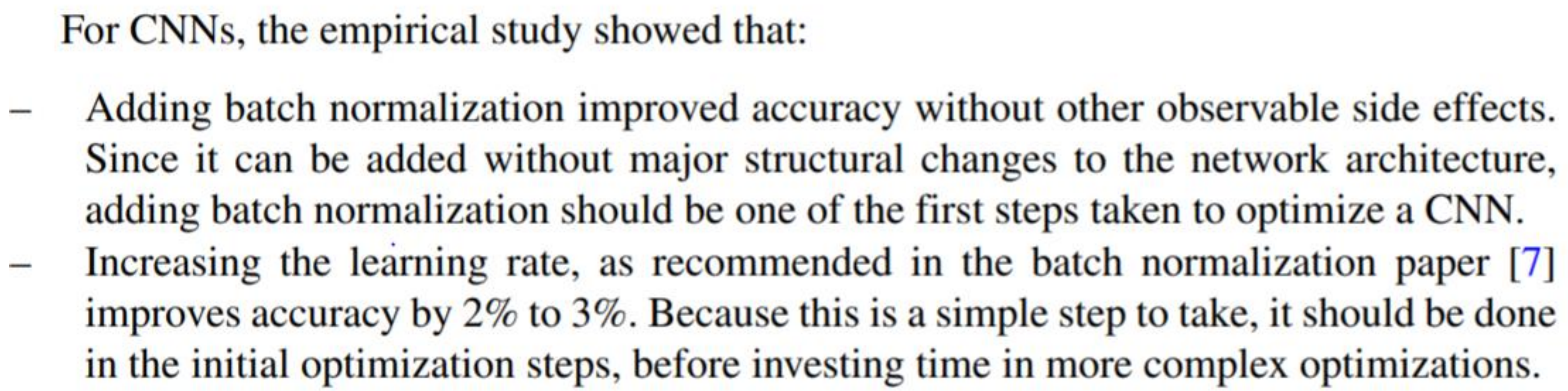
5.1.6 Recommendations and discussions

The MLP results suggest the following findings and recommendations:

- Use batch normalization if the convergence time is more important than absolute accuracy (Fig. 9). Together with early stopping, it could significantly reduce training time.
- But be aware of batch normalization's training time increase (Table 4). Unless it can be shown that it is helping converge faster during training, it may not be worth using it for experiments. Each experiment will take significantly longer to complete. It may be better to start with a standard network to run experiments faster, then switch to batch normalization in a later phase.
- Start with an adaptive optimizer (e.g. RMSProp). The experiments show that a non-adaptive SGD optimizer can be fine-tuned to outperform an adaptive one, but that comes at the cost of trying combinations of hyperparameters to find one that performs well (see for example the varying learning rate, decay, momentum and max-norm of the entries in Table 2). This adds to the training time. The accuracy of the adaptive optimizer with its default settings is not much lower. Starting with that configuration quickly provides a baseline for the tests and frees up time to experiment with other hyperparameters (e.g. the number of hidden layers, batch size, etc.).

Garbin, C., Zhu, X., & Marques, O. (2020). *Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimedia Tools and Applications.* doi:10.1007/s11042-019-

Batch Normalization



For CNNs, the empirical study showed that:

- Adding batch normalization improved accuracy without other observable side effects. Since it can be added without major structural changes to the network architecture, adding batch normalization should be one of the first steps taken to optimize a CNN.
- Increasing the learning rate, as recommended in the batch normalization paper [7] improves accuracy by 2% to 3%. Because this is a simple step to take, it should be done in the initial optimization steps, before investing time in more complex optimizations.

Garbin, C., Zhu, X., & Marques, O. (2020). *Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimedia Tools and Applications.* doi:10.1007/s11042-019-

Ministério da Ciência, Tecnologia e Inovações

Centro Brasileiro de Pesquisas Físicas

Redes Neurais profundas e aplicações Deep Learning

Clécio Roque De Bom – debom@cbpf.br

clearnightsrthebest.com

CBPF