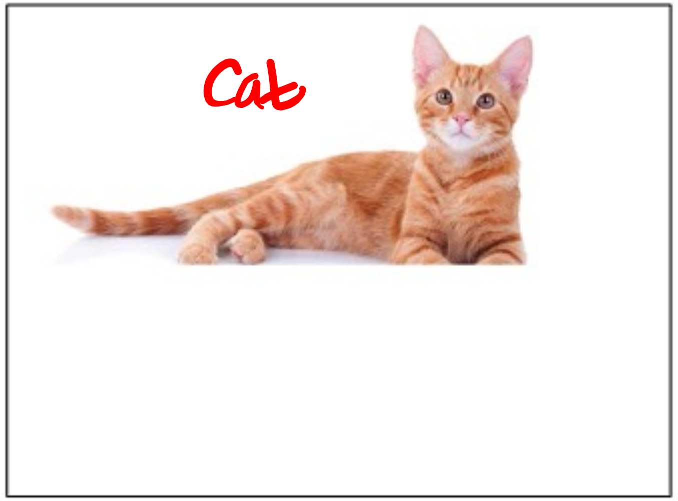
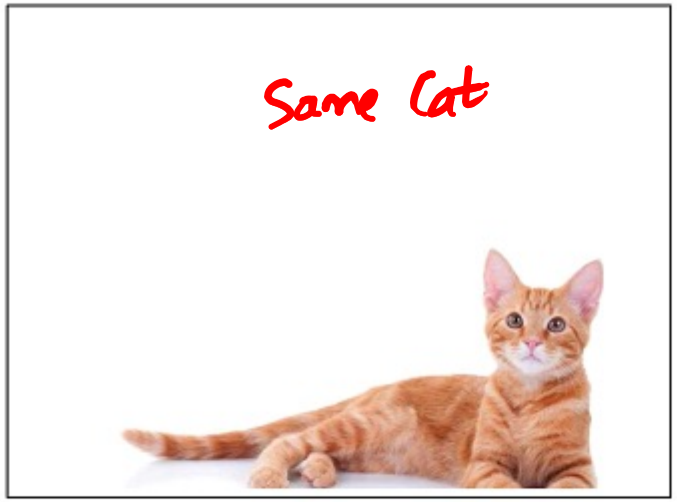
DEEP LEARNING – Convolutional Neural Network

Trainer: Dr. Darshan Ingle.

Data Augmentation



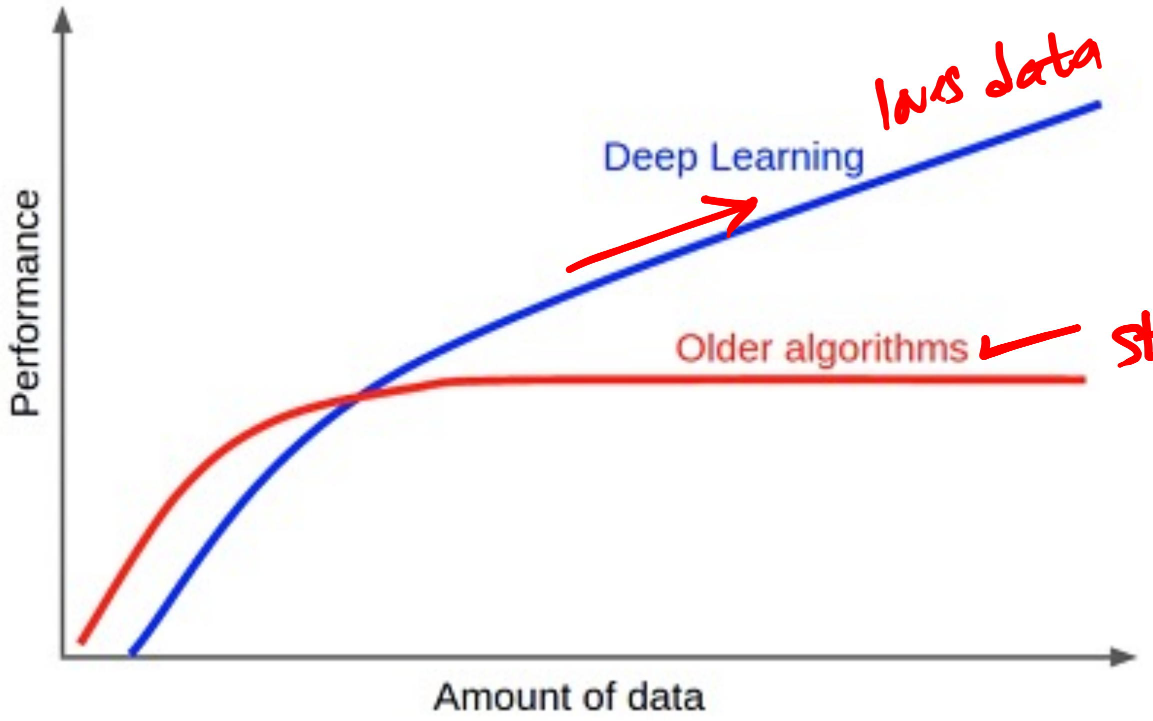


Why data is important?

891 rows × 12 cols.

X ~y 1|0

891
+100 (bias)



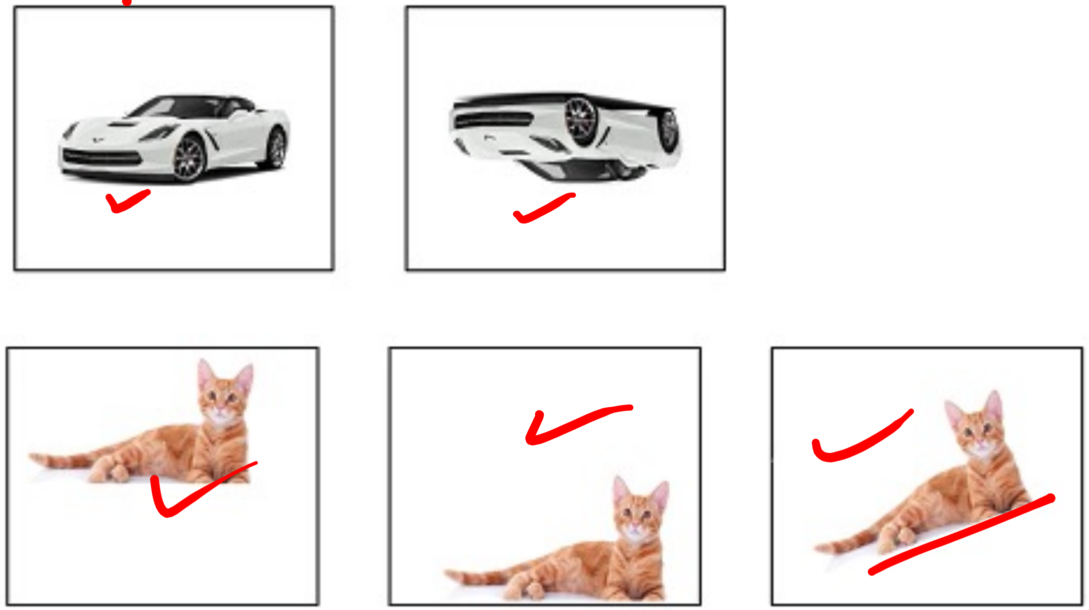
loves data

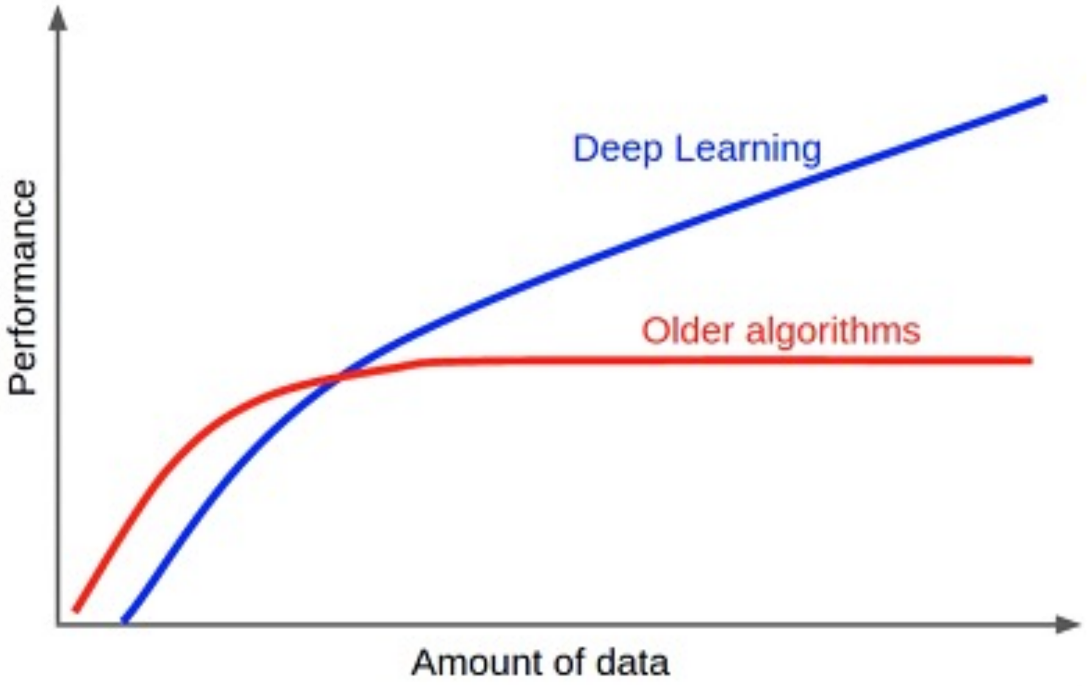
stalls

Why data is important?

With images, we can invent new data & still its meaning will remain the same.

Also, no biases will be included.





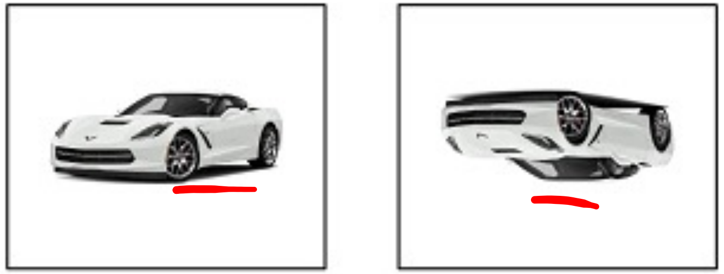
Problem with this approach

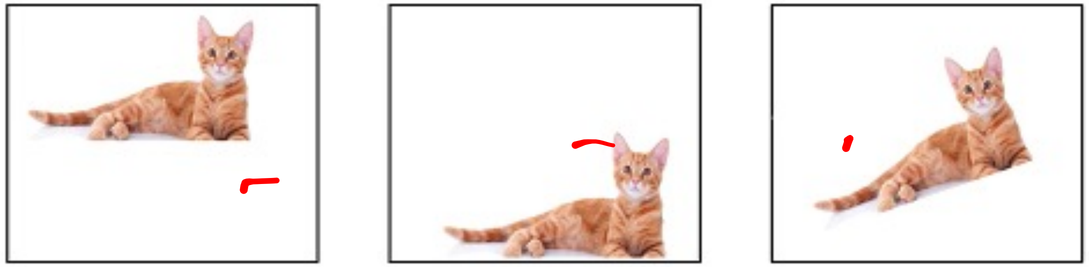
Data takes up space

The more data I invent, the more space it takes up!

There are an endless number of ways I can invent new data

- Shift to the left by 1, 2, 3, 4, 5, ... pixels
- Right / up / down also
- Rotation



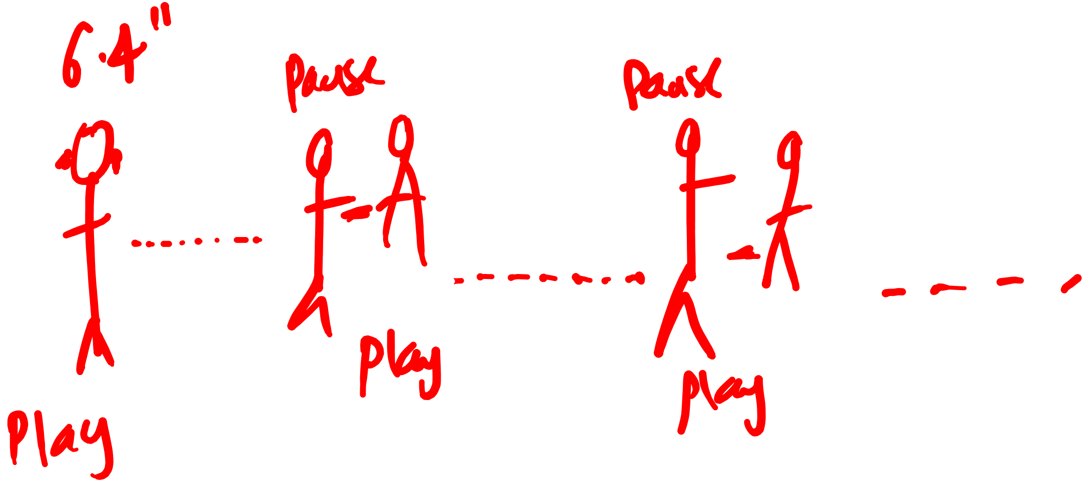


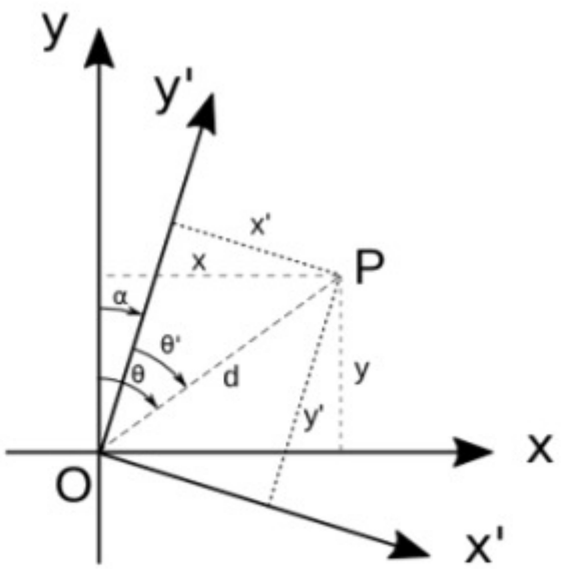
This can be done all automatically!

Have you ever considered how to rotate an image?

Use Tensorflow's Keras API instead

We need to know about generators / iterators





Generators / Iterators

Loop from 0...10: `for i in range(10)`

0, 1, - - - - 9

In Python 2, `range(10) = [0,1,2,3,4,5,6,7,8,9] # a list!`

In Python 2, use `xrange(10) # does NOT create a list`

In Python 3, `print(range(10))` yields `range(0,10)`

- Not a list!

0

1

2

3

4

5

⋮

Create your own generator

- Can you write your own function to do something like this?

```
for x in my_random_generator():
  print(x)
```

```
# 0.06654313
# -0.68315371
# 1.46795401
# -0.9017639
# 0.77572637
```

Yield

```
def my_random_generator():
    for _ in range(10):
        x = np.random.randn()
        yield x
```

Notice: no list is ever created

All values do not need to be stored in memory simultaneously

Apply this to Data Augmentation

You could similarly generate augmented data on the fly

Conceptually, think of it like this

```
def my_image_augmentation_generator():
  for x_batch, y_batch in zip(x_train, y_train):
    x_batch = augment(x_batch)
    yield x_batch, y_batch
```

~~return~~

1000 images

Batch Size = 100

No. of Batches = $\frac{1000}{100}$ = 10

✓ B1 (x, y)

B2 (x, y)

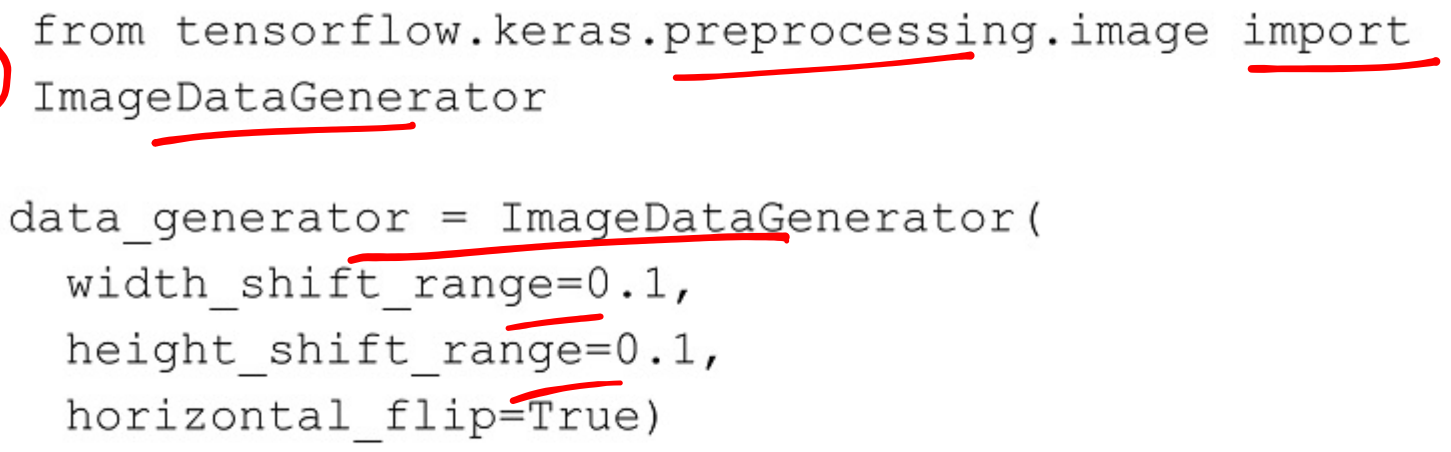
B3 (x, y)

⋮ ⋮

B10 (x, y)

How does it work in tf.keras?

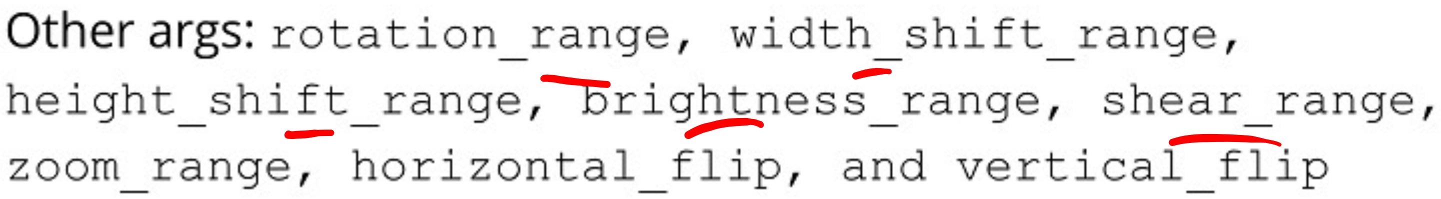
1



```
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
```

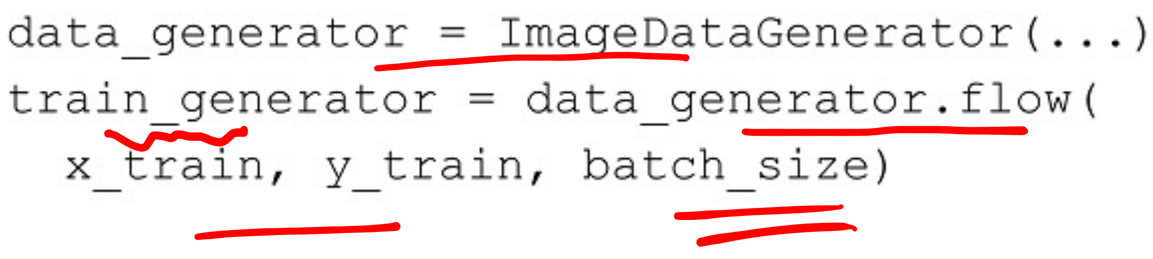
2

```
data_generator = ImageDataGenerator(
  width_shift_range=0.1,
  height_shift_range=0.1,
  horizontal_flip=True)
```

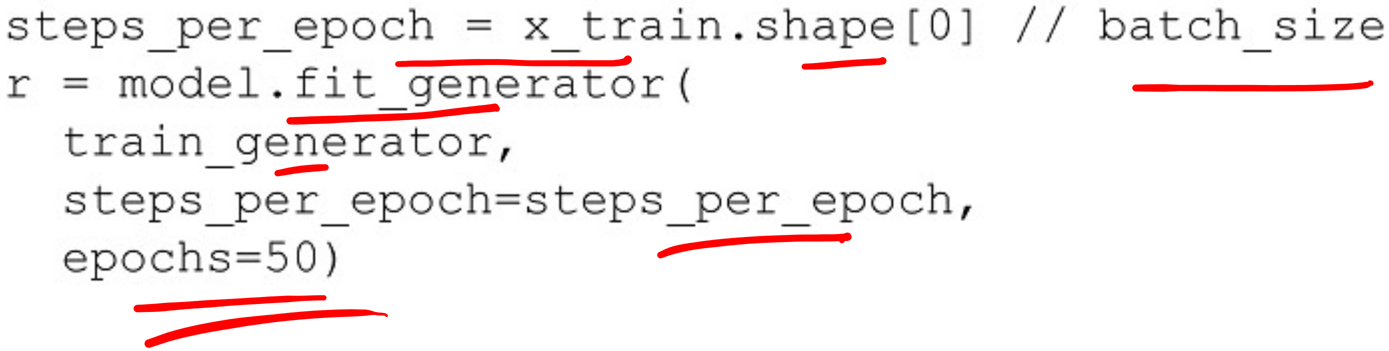


Other args: `rotation_range`, `width_shift_range`, `height_shift_range`, `brightness_range`, `shear_range`, `zoom_range`, `horizontal_flip`, and `vertical_flip`

How does it work in tf.keras?



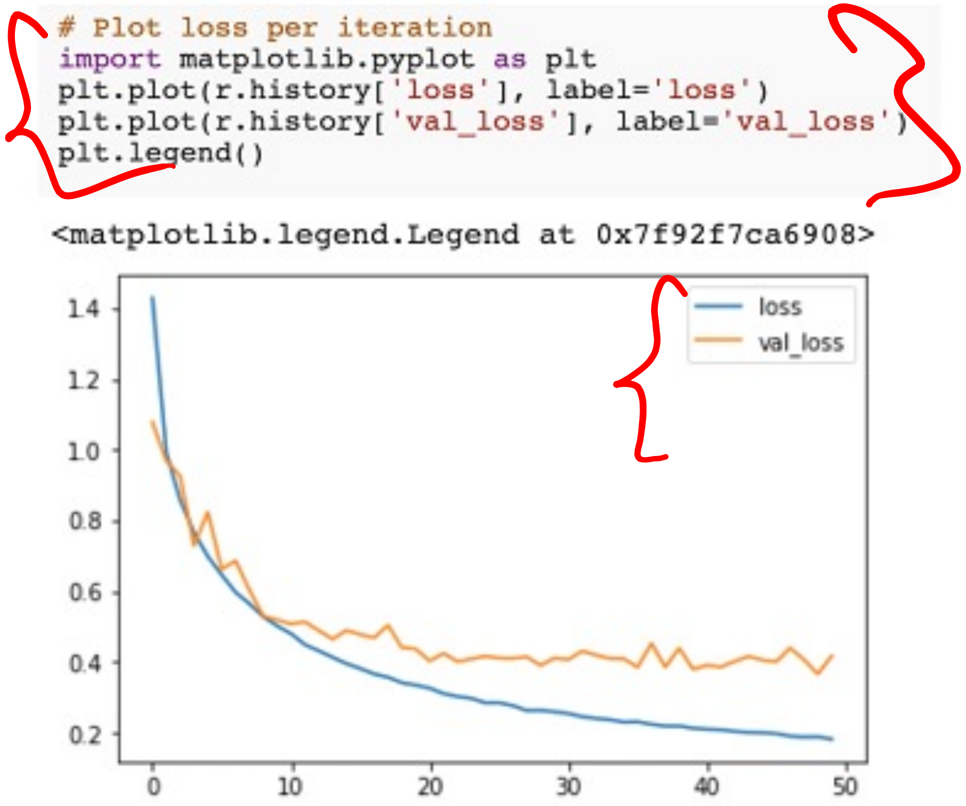
```
data_generator = ImageDataGenerator(...)
train_generator = data_generator.flow(
  x_train, y_train, batch_size)
```



```
steps_per_epoch = x_train.shape[0] // batch_size
r = model.fit_generator(
  train_generator,
  steps_per_epoch=steps_per_epoch,
  epochs=50)
```

Business as usual

`fit_generator` returns history, so plot loss per iteration, etc.

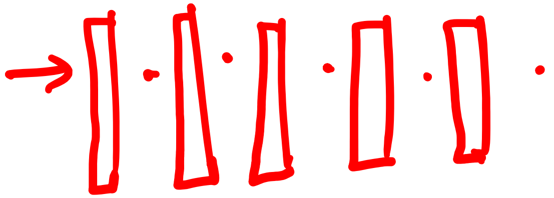


```
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
plt.legend()
```

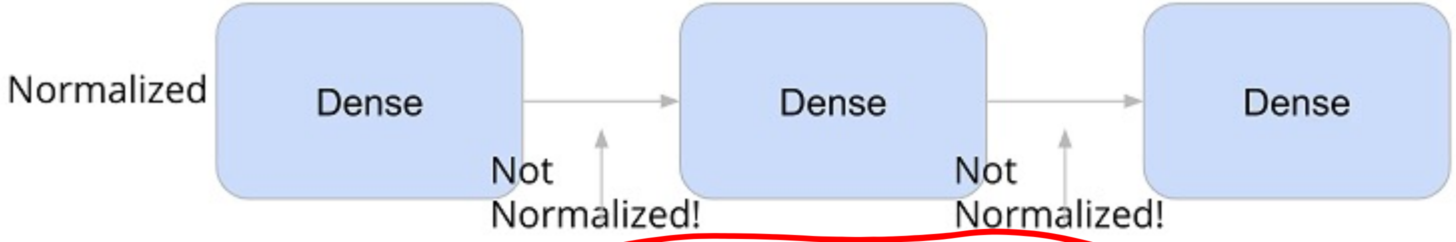
```
<matplotlib.legend.Legend at 0x7f92f7ca6908>
```

Batch Normalization

Img → Norm.



- Early on, we noted that it's important to normalize/standardize data before passing it into algorithms like linear/logistic regression
- Problem: because this operation is done only on *input data*, only the first layer sees normalized data (after being transformed by Dense layer it's no longer normalized)

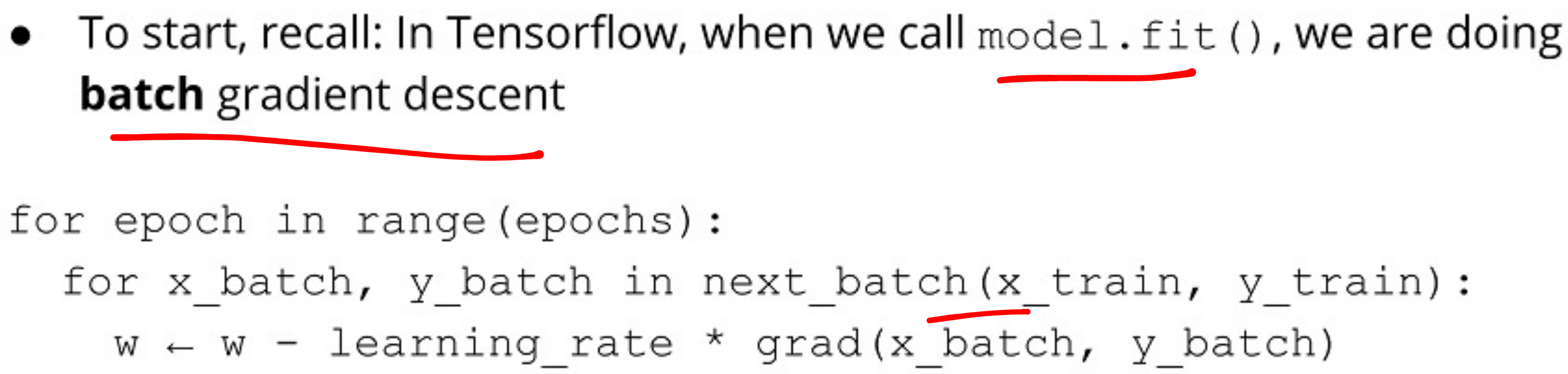


$$\boxed{z = (x - \mu)/\sigma}$$

How can we mk the data at every layer normalized!

Soln: Batch Normalization

Batch Normalization



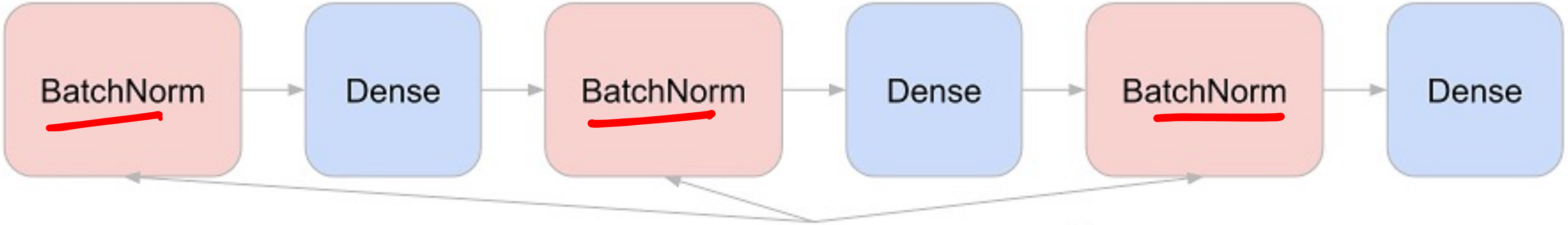
- To start, recall: In Tensorflow, when we call `model.fit()`, we are doing **batch** gradient descent

```
for epoch in range(epochs):
  for x_batch, y_batch in next_batch(x_train, y_train):
    w ← w - learning_rate * grad(x_batch, y_batch)
```

Batch Normalization

- What if we had a *layer* that would look at each batch, calculate the mean and standard deviation on the fly, and standardize based on that?

Y



$$z = (x - \mu)/\sigma$$

Batch Norm as Regularization

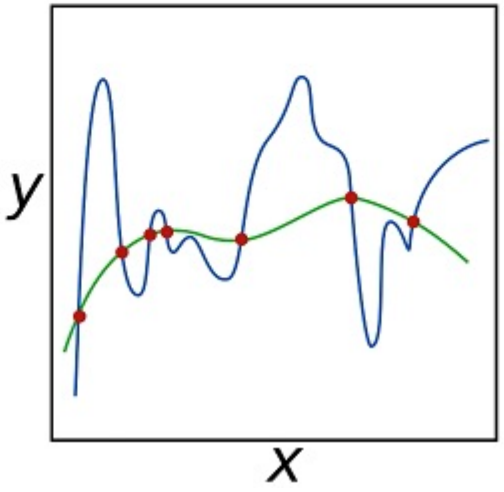
Can help with overfitting

Since every batch is slightly different, you'll get a slightly different μ_B, σ_B

They are not the true mean / std of the whole dataset

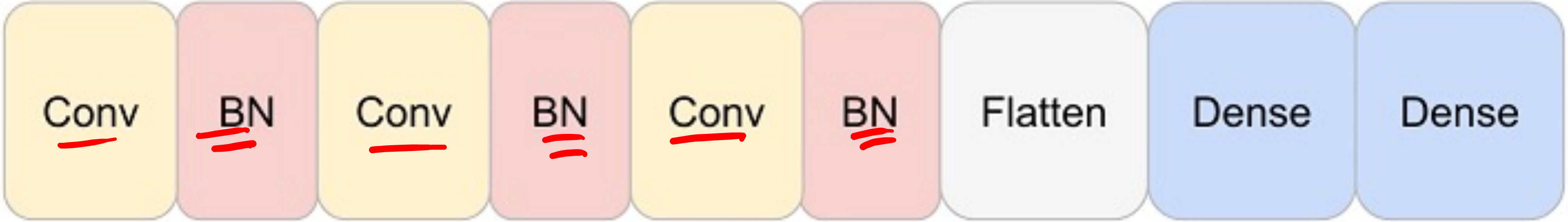
This is essentially noise, and using noise during training makes the neural network impervious to noise

- Rather than fitting to the noise! (a.k.a. "overfitting")



Where is Batch Norm used?

Imp



Conv

BN

Conv

BN

Conv

BN

Flatten

Dense

Dense

Check out architecture of popular CNN : VGG, ResNet, Inception etc.

Refer NB “7 CNN_CIFAR_Improved.ipynb”

CNN Completed!!!