```
pip install fastai
```

```
pip install fastbook
```

```
from fastai import *
from fastbook import *
```

Not to use a separate image regression application; all we've had to do is label the data, and tell fastai what kinds of data the independent and dependent variables represent.

#### Fine Tuning imagenet - Why not fitting ?

- We just fine tuning a **IMAGE CLASSIFICATION MODEL** and it turned it into a **REGRESSION MODEL**
- This worked so well because imagenet CLASSIFICATION MODEL - why because imagenet clasification model learned alot about images

#### Conclusion

- completely different (single-label classification, multi-label classification, and regression), we end up using the same model with just different numbers of outputs. The loss function is the one thing that changes, which is why it's important to double-check that you are using the right loss function for your problem.

## Image Regression Model

- Regression Model : When Dependent vairiable is a continuous number
- Classification Model : Dependent Variable a or a set of discrete categories

## A key Point Model

- independent variable -> images
- dependent -> one or more floats

## Assemble The Data

```
path=untar_data(URLs.BIWI_HEAD_POSE)
Path.BASE_PATH=path
path.ls().sorted()
```

```
(#50)
[Path('01'),Path('01.obj'),Path('02'),Path('02.obj'),Path('03'),Path('03.obj'),Path('04'),Path('04.c
```

There are 24 directories numbered from 01 to 24 (they correspond to the different people photographed), and a corresponding .obj file for each (we won't need them here). Let's take a look inside one of these directories:

```
(path/'01').ls().sorted()
```

```
(#1000)
[Path('01/depth.cal'),Path('01/frame_00003_pose.txt'),Path('01/frame_00003_rgb.jpg'),Path('01/frame_
```

Inside Sub Directories

- different frame names
- image
- pose file

We can easily get all the image files recursively with get_image_files

a function that converts an image filename to its associated pose file:

```
img_files=get_image_files(path)
def img_to_pose(x):
  return Path(f'{str(x)[:-7]}pose.txt')
```

```
img_to_pose(img_files[0])
```

```
    Path('22/frame_00562_pose.txt')
```

```
im=PILImage.create(img_files[0])
im.shape
```

```
    (480, 640)
```

```
im.to_thumb(260)
```

## Extracting the head center point:

**The Biwi dataset website used to explain the format of the pose text file associated with each image, which shows the location of the center of the head. The details of this aren't important for our purposes, so we'll just show the function we use to extract the head center point:**

```
cal = np.genfromtxt(path/'01'/'rgb.cal', skip_footer=6)
def get_ctr(f):
    ctr = np.genfromtxt(img_to_pose(f), skip_header=3)
    c1 = ctr[0] * cal[0][0]/ctr[2] + cal[0][2]
    c2 = ctr[1] * cal[1][1]/ctr[2] + cal[1][2]
    return tensor([c1,c2])
```

return coordinates

```
get_ctr(img_files[0])
```

```
    tensor([315.5824, 278.5382])
```

We can pass this function to `DataBlock` as `get_y`, since it is responsible for labeling each item. We'll resize the images to half their input size, just to speed up training a bit.

One important point to note is that we should not just use a random splitter. The reason for this is that the same people appear in multiple images in this dataset, but we want to ensure that our model can generalize to people that it hasn't seen yet. Each folder in the dataset contains the images for one person. Therefore, we can create a splitter function that returns true for just one person, resulting in a validation set containing just that person's images.

The only other difference from the previous data block examples is that the second block is a `PointBlock`. This is necessary so that fastai knows that the labels represent coordinates; that way, it knows that when doing data augmentation, it should do the same augmentation to these coordinates as it does to the images:

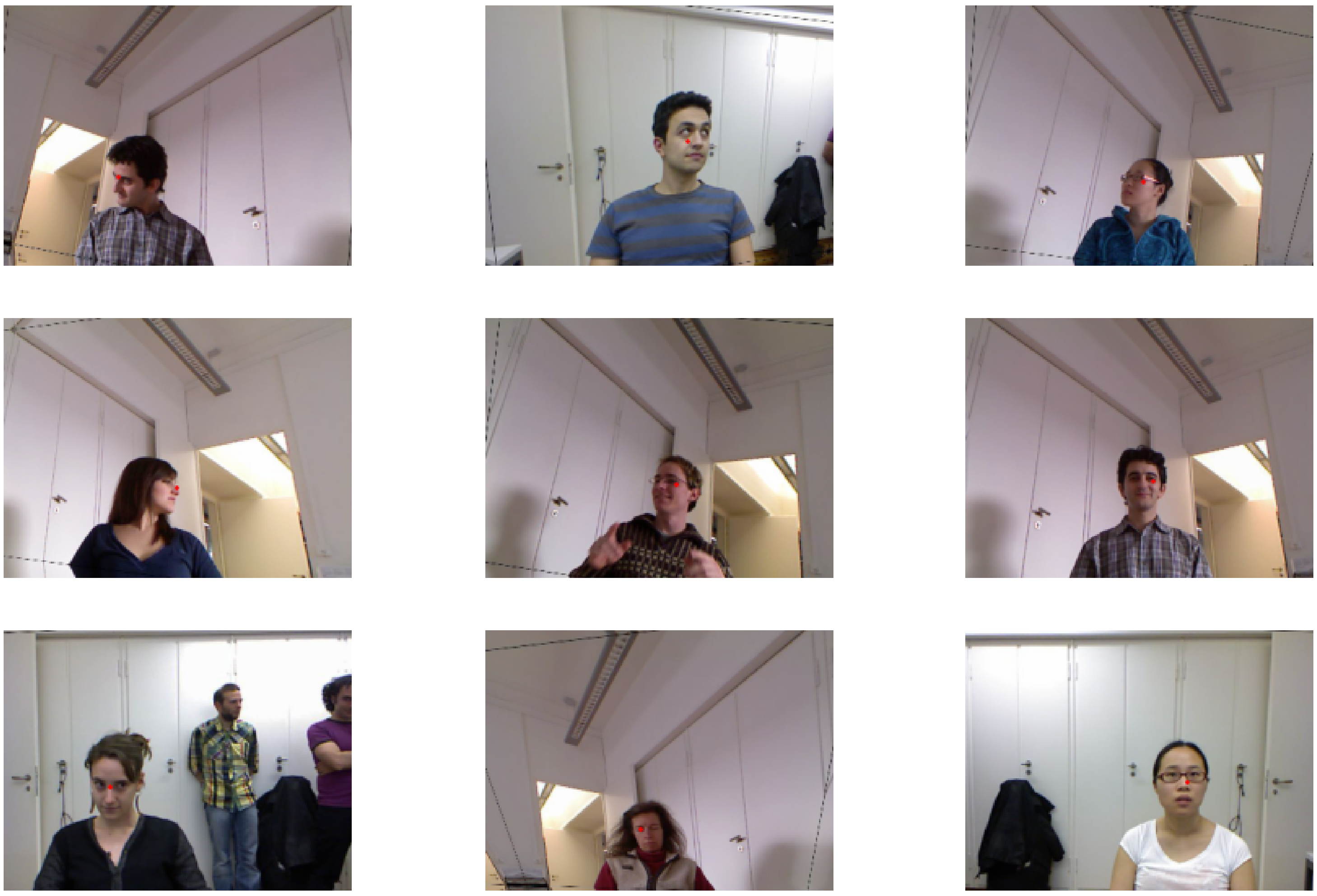
```
biwi=DataBlock(
                blocks=(ImageBlock,PointBlock),
               get_items=get_image_files,
               get_y=get_ctr,
               # make sure validation set contains one or more
               # people that dont appear in the training set
               # so grabbed randomly person # 13
               # reason -> not to overfit as alot of images looksame
               # so to introduce some some unseen data is good for this kind of data
               splitter=FuncSplitter(lambda o: o.parent.name=='13'),
               batch_tfms=[*aug_transforms(size=(240,320)),
               Normalize.from_stats(*imagenet_stats)]
)
```

**important: Points and Data Augmentation: We're not aware of other libraries (except for fastai) that automatically and correctly apply data augmentation to coordinates. So, if you're working with another library, you may need to disable data augmentation for these kinds of problems.**

```
dls=biwi.dataloaders(path)
```

## How A Batch Looks

```
dls.show_batch(max_n=9,figsize=(15,10))
```



```
xb,yb=dls.one_batch()
xb.shape,yb.shape
```

```
    (torch.Size([64, 3, 240, 320]), torch.Size([64, 1, 2]))
```

[64,3,240,320]

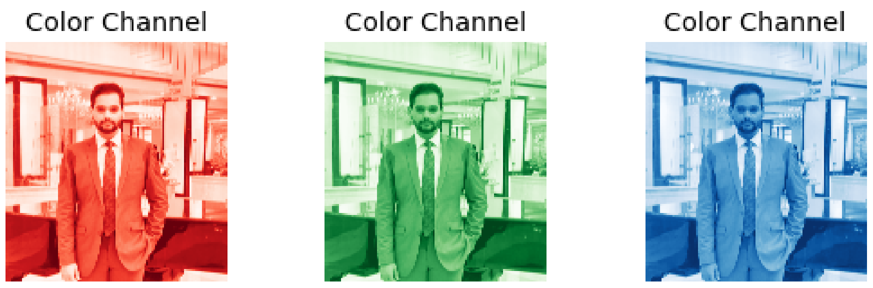
- **How a Three Channel RGB is stored**
- 64-> batch size
- 3 -> channels RGB
- 240*320 pixels

[64,1,2]

- 64 -> batch size
- Coordinates -> 1,2 (Single Point wiht 2 points represented byt (x,y) -> face center)

## Color Channels

```
from matplotlib.pyplot import subplot
im=image2tensor(Image.open('ali.jpg'))
_,axs=subplots(1,3)
for ali,ax,color in zip(im,axs,('Reds','Greens','Blues')):
  show_image(255-ali,ax=ax,cmap=color,title='Color Channel')
```



## Independent Variable

```
show_image(xb[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for i
<matplotlib.axes._subplots.AxesSubplot at 0x7ff38933d250>
```

## Dependent Variable / Labels

```
yb[0] # center of face
```

```
TensorPoint([[-0.0295,  0.2345]], device='cuda:0')
```

## Training Model

using y_range: to tell fastai the range of our targets

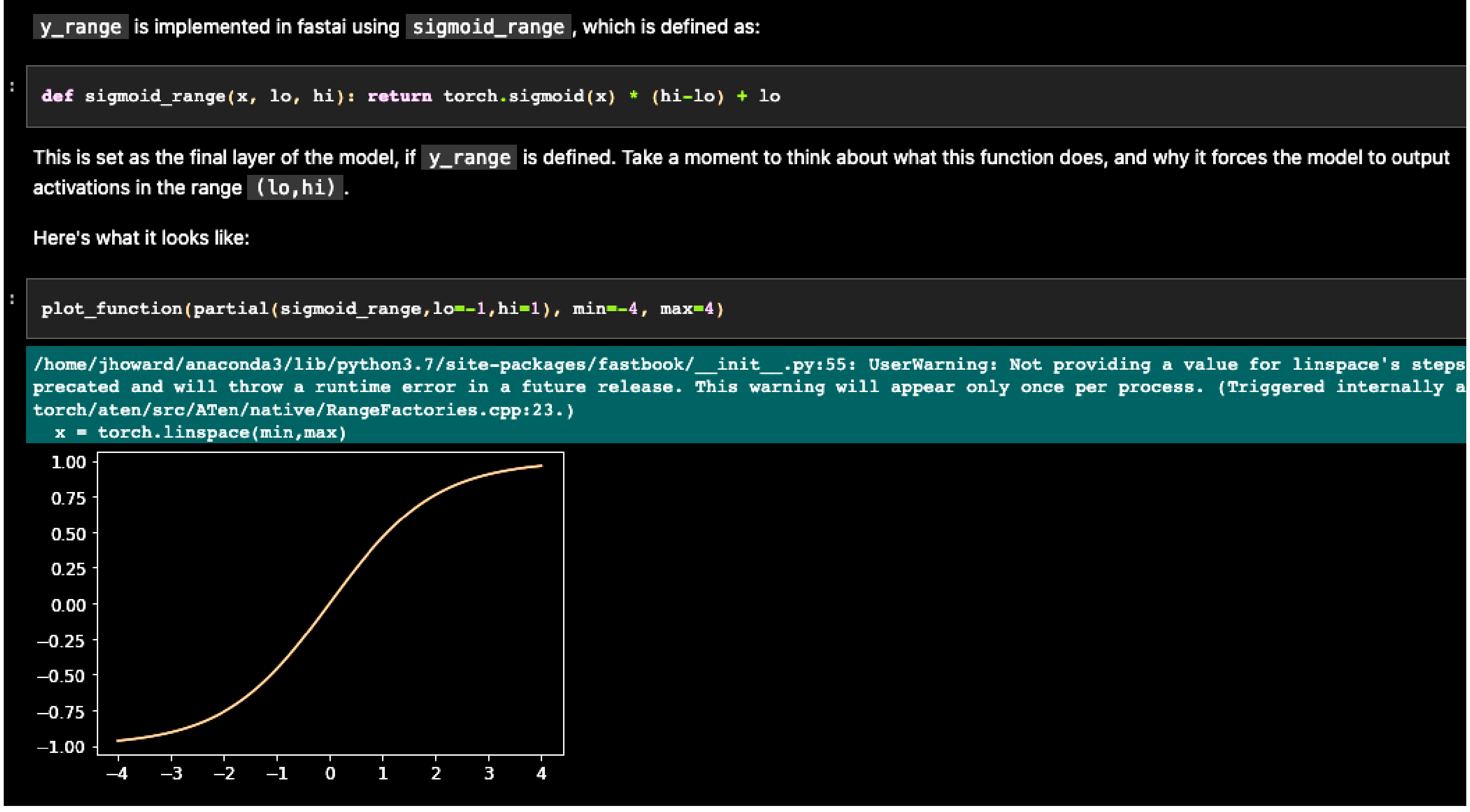
- what range of data we expect to see in our dependent variable/ probabilities

```
learn =vision_learner(dls,resnet18,y_range=(-1,1))
```

```
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:136: UserWarning: Using 'weights
  f"Using {sequence_to_str(tuple(keyword_only_kwargs.keys()), separate_last='and ')} as positional "
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments othe
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/c
100%                                        44.7M/44.7M [00:00<00:00, 71.4MB/s]
```

**y_range in fasti is implemented using sigmoid_range**

- defined as :
- def sigmoid_range(x, lo, hi): return torch.sigmoid(x) * (hi-lo) + lo


`y_range` is implemented in fastai using `sigmoid_range`, which is defined as:

```
def sigmoid_range(x, lo, hi): return torch.sigmoid(x) * (hi-lo) + lo
```

This is set as the final layer of the model, if `y_range` is defined. Take a moment to think about what this function does, and why it forces the model to output activations in the range `(lo,hi)`.

Here's what it looks like:

```
plot_function(partial(sigmoid_range,lo=-1,hi=1), min=-4, max=4)
```

```
/home/jhoward/anaconda3/lib/python3.7/site-packages/fastbook/__init__.py:55: UserWarning: Not providing a value for linspace's steps
precated and will throw a runtime error in a future release. This warning will appear only once per process. (Triggered internally a
torch/aten/src/ATen/native/RangeFactories.cpp:23.)
  x = torch.linspace(min,max)
```

## Loss Funtion

selected by default - by fastai

```
dls.loss_func
```

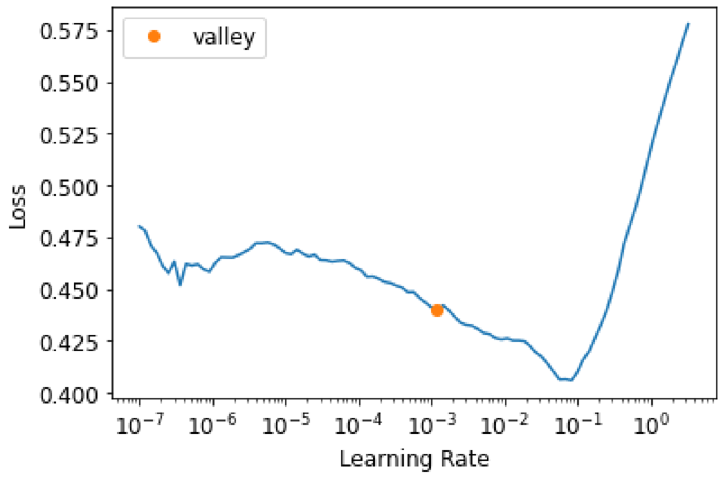
```
    FlattenedLoss of MSELoss()
```

- **This makes sense, since when coordinates are used as the dependent variable, most of the time we're likely to be trying to predict something as close as possible; that's basically what MSELoss (mean squared error loss) does. If you want to use a different loss function, you can pass it to vision_learner using the loss_func parameter.**
- **Note also that we didn't specify any metrics. That's because the MSE is already a useful metric for this task (although it's probably more interpretable after we take the square root).**

## Picking Learning Rate

```
learn.lr_find()
```

```
    SuggestedLRs(valley=0.0012022644514217973)
```



0.025 seems good

```
lr=1e-2
```

## Fine Tuning imagenet - Why not fitting ?

- **We just fine tuning a IMAGE CLASSIFICATION MODEL and it turned it into a REGRESSION MODEL**
- **This worked so well because imagenet CLASSIFICATION MODEL - why because imagenet clasification model learned alot about images**

```
learn.fine_tune(3,lr)
```

```
epoch  train_loss  valid_loss  time
    0    0.049500    0.004348  02:00
epoch  train_loss  valid_loss  time
    0    0.007719    0.002206  02:07
    1    0.002889    0.000202  02:07
    2    0.001382    0.000050  02:07
```

a loss of around 0.0001, which corresponds to an average coordinate prediction error of:

```
math.sqrt(0.0001)
```

```
0.01
```

This sounds very accurate!

## Models Predictions

```
learn.show_results(ds_idx=1, nrows=3, figsize=(6,8))
```

## Conclusion

## Conclusion

In problems that are at first glance completely different (single-label classification, multi-label classification, and regression), we end up using the same model with just different numbers of outputs. The loss function is the one thing that changes, which is why it's important to double-check that you are using the right loss function for your problem.

fastai will automatically try to pick the right one from the data you built, but if you are using pure PyTorch to build your `DataLoader`s, make sure you think hard when you have to decide on your choice of loss function, and remember that you most probably want:

- `nn.CrossEntropyLoss` for single-label classification
- `nn.BCEWithLogitsLoss` for multi-label classification
- `nn.MSELoss` for regression

✓ 1s completed at 20:31