

Multicat Notebook

Bu kısımda ilk olarak multi-label classification problemine bakacağız. Bu problemde tek bir data point birden fazla class'a ait olabilir, örneğin bir fotoğraf içerisinde hem bicycle hem car hem de person olabilir, o halde modelimiz yalnızca tek bir output'u seçmek zorunda değil.

Bu örnek için Pascal dataset'i kullanılacak, specifically pascal_2007 kullanılacak, bu dataset içinde image başına birden fazla classified object yer alabilir.

Öncekilerden farklı olarak bu dataset için labeling filename veya foldername ile yapılmamış, onun yerine bir csv kullanılmış, bu csv içerisinde her image'a karşılık labelları yazılmış, aşağıdaki gibi train.csv'i bir pandas dataframe içerisinde alarak bir bakalım:

```
path = untar_data(URLs.PASCAL_2007)
```

```
In [4]: df = pd.read_csv(path/'train.csv')
df.head()
```

```
Out[4]:
```

	fname	labels	is_valid
0	000005.jpg	chair	True
1	000007.jpg	car	True
2	000009.jpg	horse person	True
3	000012.jpg	car	False
4	000016.jpg	bicycle	True

Gördüğümüz üzere, filename'e karşılık, labels space delimited olarak verilmiş, ayrıca ilgili datapoint'in training veya validation setine ait olup olmadığı da bu csv içerisinde belirtilmiş.

Constructing a Data Block

Şimdi yukarıdaki csv dosyasından ve datablock api'dan yararlanarak bir datablock yaratalım. Daha sonra bu datablock da dataloaders yaratmak için kullanılacak.

- `Dataset`:: A collection that returns a tuple of your independent and dependent variable for a single item
- `DataLoader`:: An iterator that provides a stream of mini-batches, where each mini-batch is a tuple of a batch of independent variables and a batch of dependent variables

On top of these, fastai provides two classes for bringing your training and validation sets together:

- `Datasets`:: An object that contains a training `Dataset` and a validation `Dataset`
- `Dataloaders`:: An object that contains a training `DataLoader` and a validation `DataLoader`

Since a `DataLoader` builds on top of a `Dataset` and adds additional functionality to it (collating multiple items into a mini-batch), it's often easiest to start by creating and testing `Datasets`, and then look at `DataLoaders` after that's working.

When we create a `DataBlock`, we build up gradually, step by step, and use the notebook to check our data along the way.

Şimdi başlangıçta hiçbir parametresi olmadan bir datablock yaratalım ve bunu adım adım geliştirelim:

```
dblock = DataBlock()
```

Sırada bu datablock objesi içerisinde bir datasets oluşturmak var, bu datasets'ı oluşturmak için bize source gerekli bunun için de csv'den elde edilen dataframe'ı kullanacağız:

```
dsets = dblock.datasets(df)
```

Datablock default olarak source df'in random %20'sini validation olarak alır, bu yüzden aşağıdaki gibi dsets üzerinden train ve valid dataset'lerine ulaşabiliriz:

```
len(dsets.train), len(dsets.valid)

(4009, 1002)
```

Aşağıdaki gibi train dataset'inin ilk datapoint'ini x'e ve label'ı da y'e grab edebilirim. Aslında bu değer ilk değer olmaz, çünkü datablock source datasını otomatik olarak shuffle ediyor. Ayrıca burada grab edilen değerler tam olarak beklediğim değerler olmaz, hem x hem de y içerisine alınan değerler csv'nin tüm column'larını içerir. Yani aşağıda yapılan şey, rastgele oluşturulan train dataset'ine gidip, rastgele bir satır çekmek ve bu satırı hem x hem de y içerisine olduğu gibi atmak:

```
x, y = dsets.train[0]

x, y

(fname      008663.jpg
 labels     car person
 is_valid    False
 Name: 4346, dtype: object,
 fname      008663.jpg
 labels     car person
 is_valid    False
 Name: 4346, dtype: object)
```

As you can see, this simply returns a row of the DataFrame, twice. This is because by default, the data block assumes we have two things: input and target. We are going to need to grab the appropriate fields from the DataFrame, which we can do by passing `get_x` and `get_y` functions:

Burada x ve y içerisine istenilen column'ları almak için `get_x` ve `get_y` fonksiyonlarından yararlanacağız.

Gerçekte istediğimiz, x olarak yani independent variable olarak fname'in yani dosya isminin alınması, çünkü x'imiz bu image olacak. Buna karşılık, dependent variable'ımız ise, labels column'u olmalı, ilgili x'in içinde hangi objelerin bulunduğu yazacak örneğin car bicycle person gibi.

Şimdi istediğimiz columnları x ve y olarak seçecek yeni bir datablock oluşturalım:

```
: def get_x(r): return r['fname']
def get_y(r): return r['labels']
dblock = DataBlock(get_x = get_x, get_y = get_y)
dsets = dblock.datasets(df)
dsets.train[0]

: ('002549.jpg', 'tvmonitor')
```

Gördüğümüz gibi burada get_x ve get_y function'ı tanımlandı ve yani dedik ki x olarak alınacak satır için ilgili örneğin 'fname' column'u return edilsin buna karşın y olarak alınacak satırlar ise ilgili örneğin 'labels' column'u return edilsin.

Aşağıda oluşturulan datablock objesi üzerinden train dataset'inin 1. elemanına ulaşılmış. Görüyoruz ki sonuç beklendiği gibi.

```
dsets.train[1]
('008801.jpg', 'person motorbike car')
```

Yukarıdaki yapılardan anladık ki get_x ve get_y kullanılarak, verilen dataframe'den datasets oluşturulabiliyor, default olarak dset.train[0] dediğimizde bir tuple elde ediyoruz, bu tuple'ın ilk elemanı x ikinci elemanı y oluyor.

Yukarıdaki işlemlerle x olarak sadece 'fname' column'u alınsın, y olarak ise 'labels' column'u alınsın dedik. Ancak yukarıdaki tuple yapısı model eğitmek için henüz yeterli değil.

İsteriz ki x sadece dosya ismini içermesin full path'i de içersin böylece x kullanılarak image direkt olarak açılabilsin, bunun yanında y ise space delimited labels içeriyor, ben her bir label'ı ayrı ayrı görmek isterim:

```
def get_x(r): return path/'train'/r['fname']
def get_y(r): return r['labels'].split(' ')
dblock = DataBlock(get_x = get_x, get_y = get_y)
dsets = dblock.datasets(df)
dsets.train[1]

(Path('/storage/data/pascal_2007/train/009596.jpg'),
 ['motorbike', 'person', 'car'])
```

Görüyoruz ki artık x dediğimizde bir path objesi olarak bir ilgili image'ın tüm path'i söz konusu, buna karşılık bu x'in y'si ise bir bir array ve array içerisinde string olarak labels tutuluyor.

Datablock oluştururken işimiz hala bitmedi, x olarak full image path'i aldık ama image'i açabilmemiz için bazı transformlara ihtiyacımız olacak, benzer şekilde y datası string array olarak tutulmak yerine bir one hot encoded tensor olarak tutulacak, bunu elde etmek için bazı transformlar gerekecek. İşte bu transformları da block types'ı belirterek elde edeceğiz.

Önceden ImageBlock ve CategoryBlock kullanırdık, imageblock yine çalışacaktır çünkü x içerisinde valid bir path tutuyoruz, ancak CategoryBlock burada çalışmayacaktır, çünkü bu block tek bir integer return ediyordu, yani diyelim ki 10 sınıflı bir classification yapıyoruz, bu bloğu kullanırsak output 0-9 arasında bir tensor oluyordu. Ancak bu örnekte multiple label söz konusu o halde one hot encoded bir y gerekecek, onun için MultiCategoryBlock kullanılacak:

```
dblock = DataBlock(blocks=(ImageBlock, MultiCategoryBlock),
                    get_x = get_x, get_y = get_y)
dsets = dblock.datasets(df)
dsets.train[0]

(PILImage mode=RGB size=500x375,
 TensorMultiCategory([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]))
```

Aşağıdaki yapıyı kullanarak örneğin train dataset'inin 8. datapoint'inin label'ının 1 olduğu index'ler bulunuyor sonrasında da vocab kullanılarak bu index'lerin hangi objelere karşılık geldiği bulunuyor:

```
idxs = torch.where(dsets.train[8][1]==1.)[0]
dsets.train.vocab[idxs]

(#2) ['chair', 'person']
```

We have ignored the column `is_valid` up until now, which means that `DataBlock` has been using a random split by default. To explicitly choose the elements of our validation set, we need to write a function and pass it to `splitter` (or use one of fastai's predefined functions or classes).

Kendimiz bir splitter belirtmezsek default olarak %20 oranla random validation splitting yapılır, ancak bu örnekte df içerisinde hangi örneğin validation'a ait olduğu verilmiş, bundan yararlanmak için splitter fonksiyonu tanımlıyoruz ve datablock içerisinde bu fonksiyonu kullanıyoruz, burada dedik ki `is_valid` alanı true olanlar validation olarak ayrılın.

```
def splitter(df):
    train = df.index[~df['is_valid']].tolist()
    valid = df.index[df['is_valid']].tolist()
    return train,valid

dblock = DataBlock(blocks=(ImageBlock, MultiCategoryBlock),
                    splitter=splitter,
                    get_x=get_x,
                    get_y=get_y)

dsets = dblock.datasets(df)
dsets.train[0]

(PILImage mode=RGB size=500x333,
 TensorMultiCategory([0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]))
```

Artık görüyoruz ki dataset'imizin bir elemanına bakınca, x bana bir image veriyor, y ise buna karşılık gelen one-hot encoded tensor'u ve datasetlerim istediğim gibi bölünüyor.

Şimdi son adım olarak, datasets yerine dataloaders tanımlarız ve `item_tfms`'u da ekleriz:

```
dblock = DataBlock(blocks=(ImageBlock, MultiCategoryBlock),
                    splitter=splitter,
                    get_x=get_x,
                    get_y=get_y,
                    item_tfms = RandomResizedCrop(128, min_scale=0.35))
dls = dblock.dataloaders(df)
```

And now we can display a sample of our data:

```
dls.show_batch(nrows=1, ncols=3)
```



Unutma summary'i çağırarak datablock oluşturma sürecinde ters giden herhangi bir durumu debug edebiliriz.

Binary Cross-Entropy Loss

Şimdi biz eskisi gibi learner'ımızı oluşturabiliriz, hiçbir loss function belirtmesek dahi binar cross entropy loss default olarak seçilecek, çünkü multi-label classification için kullanılacak loss function bu olmalı.

Temelde yapılan şey, activation'ın sonuna bir sigmoid eklemek, daha sonra da her bir output için binary cross entropy loss'u hesaplayıp toplamak ve mean almak. Yani sigmoid sonrasında çıktıya göre logaritmik hatalar hesaplanacak ve toplanıp mean alınacak.

Multi class classification için categorical cross entropy loss kullanılıyordu, burada fark şuydu activations'ın sonuna sigmoids eklenmiyordu onun yerine bir softmax layer ekleniyordu, daha sonra log alınıp hatalar toplanıp mean alınıyordu. Mantık olarak çok benzer ama pratikte softmax kullanıldığı için farklı oluyor.

İlk olarak learner'ımızı aşağıdaki gibi tanımlayalım:

```
learn = cnn_learner(dls, resnet18)
```

Şimdi train dataloader'ımızı kullanarak rastgele bir batch'ı modele besleyelim ve çıktıya bakalım:

```
x,y = to_cpu(dls.train.one_batch())
activs = learn.model(x)
activs.shape

torch.Size([64, 20])
```

Learn.model'e beslenen x bir minibatch yani 64 farklı fotoğraf. Her bir fotoğraf için modelin 20 tane output'u olmalı çünkü 20 farklı class söz konusu ohalde activs.shape tam da beklediğimiz gibi.

Örneğin ilk fotoğraf için model activations'ına bakalım, 20 tane unit'in activation'ları aşağıda görülüyor, bu activationlara henüz sigmoid uygulanmamış.

```
activs[0]

TensorImage([ 0.7476, -1.1988,  4.5421, -1.5915, -0.6749,  0.0343, -2.4930, -0.8330, -0.3817, -1.4876, -0.1683,  2.1547, -3.415
1, -1.1743,  0.1530, -1.6801, -2.3067,  0.7063, -1.3358, -0.3715],
grad_fn=<AliasBackward>)
```

Loss'u aşağıdaki gibi tanımlayabiliriz, bunun yapacağı şey, yukarıdaki gibi bir activations'ı input olarak alacak bunun yanına aynı boyutlarda labels'ı targets olarak alacak ve aktivasyonların sonuna sigmoid ekledikten sonra sonda görüldüğü gibi logaritmik hataların mean'ini return edecek:

```
def binary_cross_entropy(inputs, targets):
    inputs = inputs.sigmoid()
    return -torch.where(targets==1, inputs, 1-inputs).log().mean()
```

PyTorch already provides this function for us. In fact, it provides a number of versions, with rather confusing names!

`F.binary_cross_entropy` and its module equivalent `nn.BCELoss` calculate cross-entropy on a one-hot-encoded target, but do not include the initial `sigmoid`. Normally for one-hot-encoded targets you'll want `F.binary_cross_entropy_with_logits` (or `nn.BCEWithLogitsLoss`), which do both `sigmoid` and binary cross-entropy in a single function, as in the preceding example.

The equivalent for single-label datasets (like MNIST or the Pet dataset), where the target is encoded as a single integer, is `F.nll_loss` or `nn.NLLLoss` for the version without the initial `softmax`, and `F.cross_entropy` or `nn.CrossEntropyLoss` for the version with the initial `softmax`.

Özetle bu fonksiyonlar bizim için zaten tanımlanmış, eğer `sigmoid loss function` içinde dahil edilecekse `BCEWithLogitLoss`'u kullanırsınız, aynen `softmax loss` içinde dahil edilecekse `NLLLoss` yerine `CrossEntropyLoss` kullandığımız gibi.

```
loss_func = nn.BCEWithLogitsLoss()
loss = loss_func(activs, y)
loss
TensorImage(1.0342, grad_fn=<AliasBackward>)
```

Tek bir datapoint için `loss` yukarıdaki gibi hesaplanacak.

Ama daha önce de dediğimiz gibi, `fastai` zaten biz belirtmesek de otomatik olarak doğru `loss function`'ı seçecektir.

Metrics

Bir başka konu ise `metric` seçimi, `multiclass classification` için `accuracy` metriğini kullanıyorduk, bunu nasıl yapıyorduk, `softmax`'in sonucunda `max` sonucu veren `unit`'i 1 kabul ediyorduk buna göre `label` ile karşılaştırma yapıp doğru mu yanlış mı olduğuna karar veriyorduk.

Ancak `multi-label classification` için, birden fazla `output`'un 1 olabilmesini istiyoruz, yani `softmax` kullanılmayacak dolayısıyla yukarıdaki yaklaşım işe yaramaz, onun yerine bir `threshold` belirlenir ve `sigmoid`'lerin çıkışı bu `threshold`'dan büyükse ilgili `unit` 1 `output` verdi gibi kabul edilir `accuracy` ona göre hesaplanır.

Aşağıda eski `accuracy` ve yeni `accuracy_multi` fonksiyonları yer alıyor:

```
def accuracy(inp, targ, axis=-1):
    "Compute accuracy with `targ` when `pred` is bs * n_classes"
    pred = inp.argmax(dim=axis)
    return (pred == targ).float().mean()
```

The class predicted was the one with the highest activation (this is what `argmax` does). Here it doesn't work because we could have more than one prediction on a single image. After applying the `sigmoid` to our activations (to make them between 0 and 1), we need to decide which ones are 0s and which ones are 1s by picking a *threshold*. Each value above the threshold will be considered as a 1, and each value lower than the threshold will be considered a 0:

```
def accuracy_multi(inp, targ, thresh=0.5, sigmoid=True):
    "Compute accuracy when `inp` and `targ` are the same size."
    if sigmoid: inp = inp.sigmoid()
    return ((inp>thresh)==targ.bool()).float().mean()
```

Görüldüğü gibi `accuracy_multi` içerisine bir `thresh` alıyor, `accuracy_multi` input olarak aynı `loss function` gibi içerisine `activations` alır, daha sonra bu `activations`'ın `sigmoid`'ini alıyor, ve her bir output eğer `thresh`'dan büyükse sonucu 1 gibi kabul edip label ile karşılaştırıp mean döndürüyor.

Ancak bu `thresh` her zaman 0.5 seçilmek zorunda değil bu bir `hyperparameter`'dır, aslında tam olarak `hyperparameter` da değil modelin eğitimini etkileyen bir durum değil bu tamamen bizim gözlemimizi etkileyen bir durum.

Öncelikle aşağıdaki örneklerde görüyoruz ki `learner` oluşturulurken farklı `thresh` değerleri verilirse, sonuçta farklı `accuracy`'ler göreceğiz, doğal olarak böyle olacak. Peki biz geliştirici veya kullanıcı olarak hangi `thresh`'u seçmeliyiz?

```
learn = cnn_learner(dls, resnet50, metrics=partial(accuracy_multi, thresh=0.2))
learn.fine_tune(3, base_lr=3e-3, freeze_epochs=4)
```

epoch	train_loss	valid_loss	accuracy_multi	time
0	0.942663	0.703737	0.233307	00:08
1	0.821548	0.550827	0.295319	00:08
2	0.604189	0.202585	0.816474	00:08
3	0.359258	0.123299	0.944283	00:08

epoch	train_loss	valid_loss	accuracy_multi	time
0	0.135746	0.123404	0.944442	00:09
1	0.118443	0.107534	0.951255	00:09
2	0.098525	0.104778	0.951554	00:10

Picking a threshold is important. If you pick a threshold that's too low, you'll often be failing to select correctly labeled objects. We can see this by changing our metric, and then calling `validate`, which returns the validation loss and metrics:

```
learn.metrics = partial(accuracy_multi, thresh=0.1)
learn.validate()
```

(#2) [0.10477833449840546, 0.9314740300178528]

If you pick a threshold that's too high, you'll only be selecting the objects for which your model is very confident:

```
learn.metrics = partial(accuracy_multi, thresh=0.99)
learn.validate()
```

(#2) [0.10477833449840546, 0.9429482221603394]

Bu soruya cevap vermeden önce yukarıdaki `partial` yapısını anlayalım, `partial` yapısı bir fonksiyondan bir istediğimiz parametre ile yeni bir fonksiyon türetmemizi sağlar. Yani yukarıda yapılan şey, `learn.metrics`'e bir fonksiyon vermek, fakat bu fonksiyon `accuracy_multi` olsaydı default `thresh=0.5` olacaktı, bunun yerine biz `partial` keyword'ü ile default `thresh`'u girilen değer olan yeni fonksiyonlar ürettik.

Şimdi şu sorya geri dönelim, hangi treshold değerini seçmeliyiz?

We can find the best threshold by trying a few levels and seeing what works best. This is much faster if we just grab the predictions once. Yani sanıyorum validation set için tüm predictions'ı `learn.get_preds` ile alınız. Artık bu alınan değerleri preds olarak `accuracy_multi`'ye besleyebiliriz, hangi treshold bize minimum sonuç verecek ona bakacağız, bunun için basitçe yapılacak iş, 0.05 ile 0.95 arasında farklı treshol değerleri için `accuracy_multi`'yi tüm preds için çalıştırmak böylece, validation predictions'ının accuracy'sini max yapacak treshold seçilir.

```
preds,targs = learn.get_preds()
```

```
preds.shape
```

```
torch.Size([2510, 20])
```

```
doc(learn.get_preds)
```

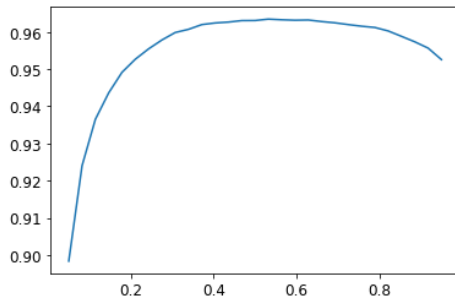
Then we can call the metric directly. Note that by default `get_preds` applies the output activation function (sigmoid, in this case) for us, so we'll need to tell `accuracy_multi` to not apply it:

```
accuracy_multi(preds, targs, thresh=0.9, sigmoid=False)
```

```
TensorImage(0.9567)
```

We can now use this approach to find the best threshold level:

```
xs = torch.linspace(0.05,0.95,29)
accs = [accuracy_multi(preds, targs, thresh=i, sigmoid=False) for i in xs]
plt.plot(xs,accs);
```



Bence bu doğru bir yaklaşım değil, Jeremy de bazıları buna yanlış diyor diyor, çünkü validation performansını etkileyecek bir parametreyi validation set üzerinde deneyerek seçiyoruz, böyle seçilen bir performansa güvenilebilir mi bilmiyorum. Jeremy güvenilir çünkü graph smooth bi graph diyor, her şeyi kurallaştırmayın diyor bana yine de biraz yanlış bir practice gibi geldi ama not ettim.

Regression

Regression Intro

Şimdi önemli bir başka konuya geldik, regression problemini nasıl handle ederiz?

Bu problem için KIWI_HEAD_POSE dataset'i kullanılacak kaggle'dan bulunabilir ama zaten, fastai içerisinde link olarak kayıtlı eskisi gibi datayı indiriyoruz. Ardından `Path.BASE_PATH = path` olarak atanıyor böylece, path'i göster dediğimde base burası olarak alınacaktır.

```
path = untar_data(URLs.BIWI_HEAD_POSE)
```

```
#hide
Path.BASE_PATH = path
```

Bakalım path içinde neler var, görüyoruz ki 24 tane klasör ve her klasör için de bir .obj dosyası var biz .obj dosyaları ile ilgilenmeyeceğiz. Her klasör farklı bir insanı temsil ediyor.

```
path.ls().sorted()

(#50) [Path('01'), Path('01.obj'), Path('02'), Path('02.obj'), Path('03'), Path('03.obj'), Path('04'), Path('04.obj'), Path('05'), Path('05.obj')...]
```

Örneğin '01' directorisinin içine bakalım, görüyoruz ki '01' insanı için klasör içinde bir çok rgb image ve her rgb image'a karşılık bir pose.txt dosyası var, ilgili image'daki kafanın merkez koordinatları pose.txt içerisinde kayıtlı.

```
In [path/'01'].ls().sorted()

: (#1000) [Path('01/depth.cal'), Path('01/frame_00003_pose.txt'), Path('01/frame_00003_rgb.jpg'), Path('01/frame_00004_pose.txt'), Path('01/frame_00004_rgb.jpg'), Path('01/frame_00005_pose.txt'), Path('01/frame_00005_rgb.jpg'), Path('01/frame_00006_pose.txt'), Path('01/frame_00006_rgb.jpg'), Path('01/frame_00007_pose.txt')...]
```

Datanın genel tutulma yapısını anladık, şimdi biliyoruz ki `get_image_files` fonksiyonunu en genel path klasörünü verirsek, bu fonksiyon bize bir liste olarak tüm image'ların isimlerini return edecektir. Daha sonra şunu da biliyoruz ki biz bir image ismini alarak sadece sondaki eklemeleri değiştirerek ilgili image'ın pose.txt dosyasının ismini elde edebiliriz:

```
img_files = get_image_files(path)
def img2pose(x): return Path(f'{str(x)[: -7]}pose.txt')
img2pose(img_files[0])

Path('13/frame_00349_pose.txt')
```

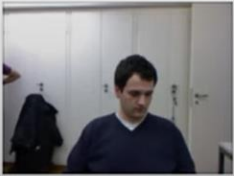
Örneğin yukarıda 13. Klasördeki 349. Image'ın pose.txt'si path olarak elde edilmiş.

Aşağıdaki yapı ile `img_files` listesinin ilk elemanına yani ilk resme bakabiliriz:

We can have a look at our first image:

```
4] im = PILImage.create(img_files[0])
   im.shape
4]: (480, 640)

5] im.to_thumb(160)
```



BIWI dataset'inin açıklamasına bakarsak, `pose.txt`'ye bakarak nasıl koordinatları elde edeceğimiz belirtilmiş, bunu aşağıdaki fonksiyonla yapıyoruz:

The Biwi dataset web site explains 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:

```
6] cal = np.genfromtxt(path/'01'/'rgb.cal', skip_footer=6)
def get_ctr(f):
    ctr = np.genfromtxt(img2pose(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])
```

Sonuçta eğer biz bu fonksiyona bir image beslersek, ilgili image'ın label'ını görebiliriz, label tensor of two items olacak:

This function returns the coordinates as a tensor of two items:

```
7] get_ctr(img_files[0])
7]: tensor([384.6370, 259.4787])
```

Create DataBlock

Şimdi datablock objesini aşağıdaki gibi oluşturabiliriz, burada değişen ilk şey block olacak, eskiden category veya multicategory block olan kısma artık PointBlock yazıyoruz bu point temsil eden 2D tensors için özel kullanılıyor sanıyorum.

get_items=get_image_files diyerek verilecek path'teki her bir image'ın path'i x olarak alınacak, buna karşılık get_y'ye verilen fonksiyon ise x'e alınan her path için çalışacak ve ilgili x'e karşılık y label'ı oluşturulacak.

Bir diğer önemli nokta ise splitter, burada modelin görmediği insanlara da generalize edebilmesi için funcsplitter kullanıyoruz ve örneğin 13. Klasörün tamamını validation olarak ayırıyoruz.

Ardından aug_transforms uyguluyoruz, ve normalization yapıyoruz, normalization zaten yazmasak da otomatik olarak yapılıyor sanıyorum.

```
biwi = DataBlock(
    blocks=(ImageBlock, PointBlock),
    get_items=get_image_files,
    get_y=get_ctr,
    splitter=FuncSplitter(lambda o: o.parent.name=='13'),
    batch_tfms=[*aug_transforms(size=(240,320)),
                Normalize.from_stats(*imagenet_stats)]
)
```

Şimdi oluşturulan datablocks objesi üzerinden data path'ı belirtilerek dataloaders oluşturuluyor ve sonra da show_batch ile örneklerle bakıyoruz, farkındaysan PointBlock kullandığımız için fastai otomatik olarak label'ı image üzerinde bir red dot olarak gösteriyor.

```
dls = biwi.dataloaders(path)
dls.show_batch(max_n=9, figsize=(8,6))
```



Dataloaders üzerinden rastgele bir batch çekelim ve shape'e bakalım olayı aha iyi kavrarız:

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

((64, 3, 240, 320), (64, 1, 2))
```

Görüyoruz ki batch_size=64 olduğu için 64 tane 3 channel'lı 240x320 boyutlarında image xb olarak turuluyor, benzer şekilde yb ise her datapoint için 1,2 boyutlarında coordinates tutan bir tensor olacak.

Labelların birine daha yakından bakalım:

```
yb[0]

TensorPoint([[-0.3375,  0.2193]], device='cuda:6')
```

As you can see, we haven't had 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.

Training Our Model

Learner yaratma süreci de çok benzer olacak, farklı olarak `y_range` kullanılacak `y_range` kullanarak regression output'larımın hangi range'de olmasını istediğimi belirtiyorum, bu belirtmenin temelde yaptığı şey en sona basit bir sigmoid koymak yerine belirtilen range'lerde bir sigmoid koymak olur, yani range 20-50 arası belirtilirse output sigmoidi -sonsuz için 20 verecek, +sonsuz için 50 verecek şekilde set edilir.

```
: learn = cnn_learner(dls, resnet18, y_range=(-1,1))
```

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

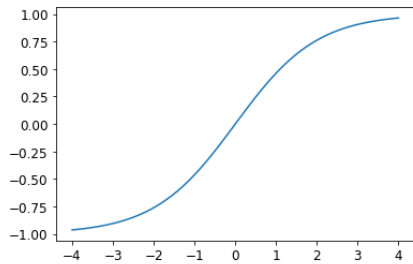
```
: def sigmoid_range(x, lo, hi): return torch.sigmoid(x) * (hi-lo) + lo
```

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 is deprecated and will throw a runtime error in a future release. This warning will appear only once per process. (Triggered internally at /pytorch/aten/src/ATen/native/RangeFactories.cpp:23.)
  x = torch.linspace(min,max)
```



Herhangi bir loss function specify etmediğimize dikkat et, datablock kullanılarak fastai otomatik olarak hangi loss function kullanacağına karar veriyor MSELoss bunu check etmek için `dls.loss_func` diyebiliriz:

```
dls.loss_func
```

```
FlattenedLoss of MSELoss()
```

Bunu değiştirmek istersek `cnn_learner`'un `loss_func` parametresini değiştirebiliriz.

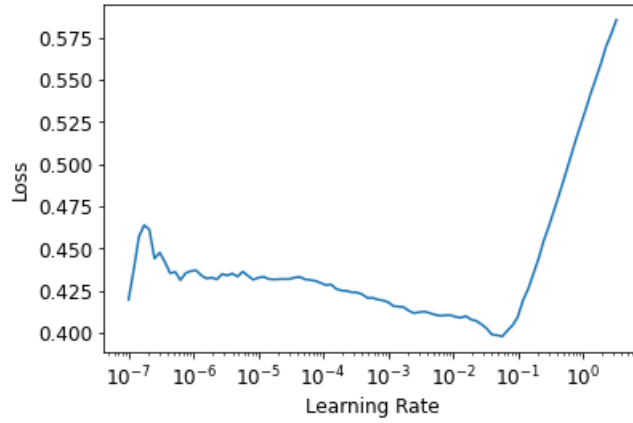
Ayrıca metric belirtmiyoruz çünkü MSELoss metric olarak da kullanılabilir.

Şimdi modeli fine_tune etmeden önce stage1 learning rate'ini bulalım:

Lr 1e-2 olarak seçilmiş, stage1 için bu değer kullanılsın, fine_tune methodunu kullandığımız için stage2 lr'ını kendi ayarlayıyor zaten.

```
learn.lr_find()
```

SuggestedLRs(lr_min=0.005754399299621582, lr_steep=0.033113110810518265)



We'll try an LR of 2e-2:

```
lr = 1e-2  
learn.fine_tune(3, lr)
```

epoch	train_loss	valid_loss	time
0	0.049630	0.007602	00:42

epoch	train_loss	valid_loss	time
0	0.008714	0.004291	00:53
1	0.003213	0.000715	00:53
2	0.001482	0.000036	00:53

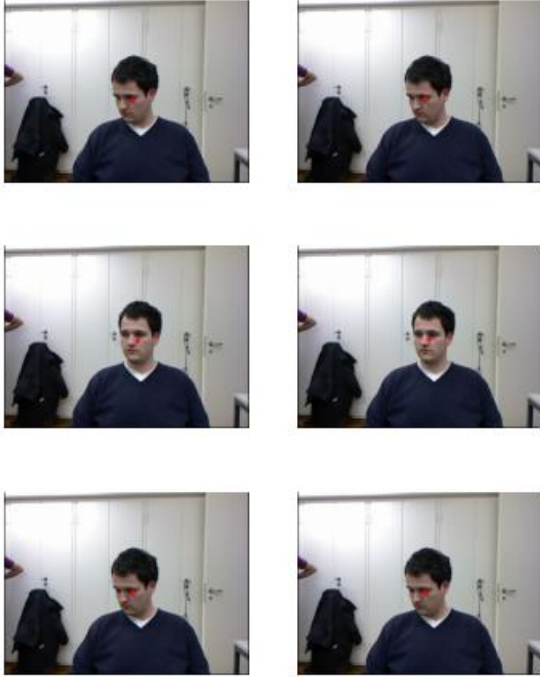
Sonuçta 4 epoch eğitim ile gayet iyi görünen bir model eğittik.

Results

Learn.show_results ile otomatik olarak fastai bize target ve prediction'ı aşağıdaki şekilde gösterir, gayet iyi görünüyor.

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

Target/Prediction



Sonuç olarak önemli olan nokta şu: ASIL OLAY DATAYI HAZIRLAMAKTA VE İLGİLİ DATABLOCK'U ELDE ETMEKTE YATILIYOR KALAN KISIMLAR KOLAYCA HALLEDİLİYOR.
