

```
pip install fastai
```

```
pip install fastbook
```

```
from fastbook import *
```

```
from fastai.vision.all import *
```


Multi-Label Classifications

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

```
path.ls()
```

```
(#8)
[Path('/root/.fastai/data/pascal_2007/segmentation'),Path('/root/.fastai/data/pascal_2007/test'),Path('
```

```
df=pd.read_csv(path/'train.csv')
df
```

	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	
...	
5006	009954.jpg	horse person	True	
5007	009955.jpg	boat	True	
5008	009958.jpg	person bicycle	True	
5009	009959.jpg	car	False	
5010	009961.jpg	dog	False	

5011 rows × 3 columns

Constructing Data Block

```
dblock=DataBlock()
```

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

```
(fname      000005.jpg
 labels      chair
 is_valid      True
Name: 0, dtype: object, fname      000005.jpg
 labels      chair
 is_valid      True
Name: 0, dtype: object)
```

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

```
(4009, 1002)
```

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

```
x,y
```

```
(fname      007097.jpg
 labels     aeroplane
 is_valid   True
 Name: 3586, dtype: object, fname      007097.jpg
 labels     aeroplane
 is_valid   True
 Name: 3586, dtype: object)
```

```
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[0]
```

```
(Path('/root/.fastai/data/pascal_2007/train/005248.jpg'), ['person', 'horse'])
```

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

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

```
dset.train[0]
```

```
(PILImage mode=RGB size=500x334,
 TensorMultiCategory([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0.,
 0.]))
```

tensors are float -> speed up computations as using cross entropy loss()

knowing where true

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

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

▼ Splitting of Data Set

- Use Random if validation set not give not given
- *If Data Set tells which validation to use then you should use it then you may compare your validation result to other**

```
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,
    item_tfms=RandomResizedCrop(128,min_scale=0.35)
)
```

```
dls= dblock.dataloaders(df)
```

```
dsets=dblock.datasets(df)
dsets.train[0],dsets.valid[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., 0.,
0.])),
(PILImage mode=RGB size=500x375,
  TensorMultiCategory([0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.])))
```

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

dog;person



horse;person



chair;person;sofa



chair;diningtable



chair;sofa;tvmonitor



car;horse;person



chair;pottedplant;sofa



dog;person



horse;person



Binary Cross Entropy

- creating Learner
- **returns activations of the model from the final layer**

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

```
/usr/local/lib/python3.7/dist-packages/fastai/vision/learner.py:287: UserWarning: `cnn_learner` has been
warn("`cnn_learner` has been renamed to `vision_learner` -- please update your code")
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:136: UserWarning: Using 'weights' a
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 other t
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/chec
100%
44.7M/44.7M [00:01<00:00, 74.1MB/s]
```

```
x,y=dls.train.one_batch()
activations=learn.model(x)
```

```
activations # predictions from the last layer
```

```
TensorBase([[ -2.4902, -1.8546,  1.4278, ...,  2.6413,  2.0615, -1.5694],
 [  0.2143,  0.6582, -0.3282, ...,  2.9242, -1.7084, -1.3147],
 [ -2.1987,  2.5293, -4.1499, ..., -0.5085, -1.7608, -1.6234],
 ...,
 [ -0.7922,  2.4168, -0.3687, ..., -2.4236, -1.3405,  0.6952],
 [ -1.5835, -1.1215, -0.9287, ..., -1.5578,  1.8463,  1.9448],
```

Always check for shape of Returned Tensor

- 64 -> size of mini batch
- 20 -> probabilities for each of 20 categories (labels)

```
activations.shape
```

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

```
activations[0]
```

```
TensorBase([ -2.4902, -1.8546,  1.4278,  2.7478,  0.1311,  2.6155,  2.2897,  0.8735, -3.9674, -0.1550,
 -0.7420,  1.7110, -0.5503,  0.7111, -2.4944, -0.6024,  2.4099,  2.6413,  2.0615, -1.5694],
 grad_fn=<AliasBackward0>)
```

pretrained model with there random last layer weights

Loss Funtion

- transforming activations between 0-1

self written

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

using pytorch built in

```
loss_func=nn.BCEWithLogitsLoss()
loss=loss_func(activations,y) # comparing activations to our targets
loss

TensorMultiCategory(1.0612, grad_fn=<AliasBackward0>)
```

will use this to train now

- **Not need to tell fastai to use this as a loss funtion - it automatically knows to use this with multilabel problem**

Metric

accuracy or error rate

- only works wiht single layer datasets**
- makes sense if looking for a single maximum thing

accuray_multi

- compare each activation to each activation thresh

We don't actually need to tell fastai to use this loss function (although we can if we want) since it will be automatically chosen for us. fastai knows that the `DataLoaders` has multiple category labels, so it will use `nn.BCEWithLogitsLoss` by default.

One change compared to the last chapter is the metric we use: because this is a multilabel problem, we can't use the accuracy function. Why is that? Well, accuracy was comparing our outputs to our targets like so:

```
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()
```

If we pass `accuracy_multi` directly as a metric, it will use the default value for `threshold`, which is 0.5. We might want to adjust that default and create a new version of `accuracy_multi` that has a different default. To help with this, there is a function in Python called `partial`. It allows us to *bind* a function with some arguments or keyword arguments, making a new version of that function that, whenever it is called, always includes those arguments. For instance, here is a simple function taking two arguments:

▼ Train Multi-Label DataSet

- with accuracy thresh=0.2

```
learn=cnn_learner(dls,resnet50,metrics=partial(accuracy_multi,thresh=0.2))
```

```
/usr/local/lib/python3.7/dist-packages/fastai/vision/learner.py:287: UserWarning: `cnn_learner` has been
warn("`cnn_learner` has been renamed to `vision_learner` -- please update your code")
/usr/local/lib/python3.7/dist-packages/torchvision/models/_utils.py:136: UserWarning: Using 'weights' as
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 other t
warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/chec
100% 97.8M/97.8M [00:01<00:00, 85.0MB/s]
```

```
learn.fine_tune(3,base_lr=3e-3,freeze_epochs=4) # loss funtion set by default seeing datablock
```

epoch	train_loss	valid_loss	accuracy_multi	time
0	0.949114	0.715978	0.229821	10:03
1	0.829925	0.563730	0.288785	09:32
2	0.610766	0.202619	0.821733	09:30
3	0.363862	0.123458	0.943008	09:32
epoch	train_loss	valid_loss	accuracy_multi	time
0	0.132224	0.118181	0.945597	12:28
1	0.115952	0.107362	0.949522	12:32
2	0.095943	0.102443	0.952809	12:27

▼ Picking A Threshold for our Model

- Hit adn Trail
- much faster if we just grab the predictions once: https://github.com/fastai/fastbook/blob/master/06_multicat.ipynb

In this case, we're using the validation set to pick a hyperparameter (the threshold), which is the purpose of the validation set. Sometimes students have expressed their concern that we might be overfitting to the validation set, since we're trying lots of values to see which is the best. However, as you see in the plot, changing the threshold in this case results in a smooth curve, so we're clearly not picking some inappropriate outlier. This is a good example of where you have to be careful of the difference between theory (don't try lots of hyperparameter values or you might overfit the validation set) versus practice (if the relationship is smooth, then it's fine to do this).

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

```
(#2) [0.10244287550449371,0.9329084157943726]
```

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

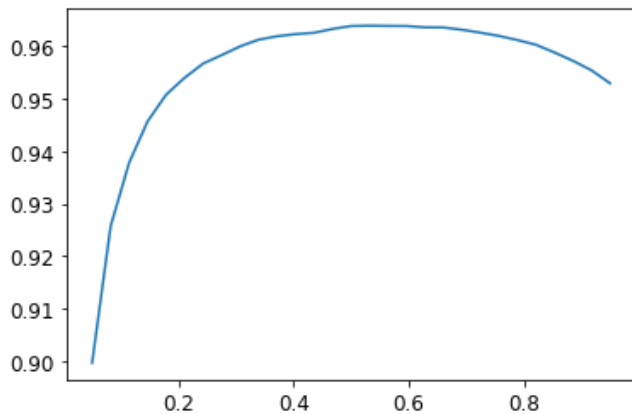
```
(#2) [0.10244287550449371,0.9448006749153137]
```

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

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

```
TensorBase(0.9563)
```

```
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);
```



So a good threshold will be around 0.5

✓ 0s completed at 17:51

