general notes OP

Types of things we used

for loading

- **ImageDataLoaders.from_name_func**
- **SegmentationDataLoaders.from_label_func**
- **TextDataLoaders.from_folder**
- **TabularDataLoaders.from_csv**
- **CollabDataLoaders.from_csv**
- **xyzDataLoaders.from_df** bhi allowed hai #### clearly jo second part hai, wo situation ke hisaab se apply karna

for learning

- **learn = cnn_learner(dls, resnet34, metrics=error_rate)**
- **learn = unet_learner(dls, resnet34)**
- **learn = text_classifier_learner(dls, AWD_LSTM, drop_mult=0.5, metrics=accuracy)**
- **learn = tabular_learner(dls, metrics=accuracy)**
- **learn = collab_learner(dls, y_range=(0.5,5.5))**

Unrelated to the documentation but still very useful: to get help at any point if you get an error, type `%debug` in the next cell and execute to open the Python debugger, which will let you inspect the content of every variable.

> **DataLoaders**: A fastai class that stores multiple DataLoader objects you pass to it, normally a train and a valid, although it's possible to have as many as you like. The first two **(train and valid) are made available as properties.**

To turn our downloaded data into a `DataLoaders` object we need to tell fastai at least four things:

- What kinds of data we are working with
- How to get the list of items
- How to label these items
- How to create the validation set

data block API. With this API you can fully customize every stage of the creation of your `DataLoaders` **.**

In [ ]:
```
animals_datablock=DataBlock(
    blocks=(ImageBlock,CategoryBlock),
    # what types we want for the independent and dependent variables:
    # independent are images, output is category
    get_items=get_image_files,
    # get_image_files function takes a path, and returns a list of all of the images in that path
    splitter=RandomSplitter(valid_pct=0.2,seed=42),
    #splits the dataset into a training set and validation set
    get_y=parent_label,
    # what function to call to create the labels in our dataset
    # parent_label is a function provided by fastai that simply gets the name of the folder a file is in
    item_tfms=RandomResizedCrop(224, min_scale=0.5),
    batch_tfms=aug_transforms())
```

This command has given us a DataBlock object. This is like a template for creating a DataLoaders. We still need to tell fastai the actual source of our data—in this case, the path where the images can be found:

In [ ]:
```
dls=animals_datablock.dataloaders(path)
```

```
dls.valid.show_batch(max_n=4,nrows=1) #for verifying ki sab barobar hai ya nahi
```

datablock direct df se bhi bante hai

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`

In [ ]:
```
# (fname        008663.jpg
#  labels       car person
#  is_valid          False --> aisa hai df poora, images alag file me  stored hai
```

In [ ]:
```
# dblock = DataBlock() #sirf aise hi jaane diya to dsets.train ke elements have same df twice
# doing x,y = dsets.train[0] will give same thing to x,y
# instead use
dblock = DataBlock(get_x = lambda r: r['fname'], get_y = lambda r: r['labels'])
dsets = dblock.datasets(df)
dsets.train[0]
# ('002549.jpg', 'tvmonitor')
```

In [ ]:
```
# or if  independent variable will need to be converted into a complete path,
# so that we can open it as an image
dblock = DataBlock(get_x = lambda r: path/'train'/r['fname'], get_y = lambda r: r['labels'].split(' '))
dsets = dblock.datasets(df)
dsets.train[5] # gives x and y
# (Path('Multicat_Data/train/002868.jpg'), ['cow', 'bird'])
```

To actually open the image and do the conversion to tensors. ImageBlock will work fine again, because we have a path that points to a valid image, but the CategoryBlock is not going to work since it returns a single integer, but if we need to be able to have multiple labels for each item, we use a **MultiCategoryBlock**

In [ ]:
```
def get_x(r): return path/'train'/r['fname']
def get_y(r): return r['labels'].split(' ')
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., 0.]))
```

dataloaders!

In [ ]:
```
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)
```

accuracy_multi since normal accuracy cannot be used (wo single output category ke liye hi bana hai). threshold value is default 0.5 (accuracy_multi me *sigmoid* use hota hai)

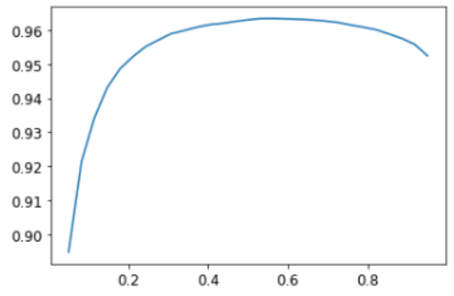
In [ ]:

```
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     0.944744        0.700274        0.236534        00:39
# 1     0.823401        0.554526        0.300239        00:32
# 2     0.601602        0.199432        0.817510        00:32
# 3     0.359271        0.123277        0.945478        00:33
# epoch train_loss      valid_loss      accuracy_multi  time
# 0     0.137921        0.118284        0.945916        00:38
# 1     0.118691        0.104569        0.949681        00:37
# 2     0.099470        0.103237        0.951016        00:38
```

to get best threshold

In [ ]:

```
preds,targs = learn.get_preds()
# preds is (#samples,#categories) jisme prediction probaility values hai
# targets is same shape with 0,1 to show if category is yes or no
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)
```



In this case, we're using the validation set to pick a hyperparameter (the threshold), which is the purpose of the validation set. However, 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)

In [ ]:

In [ ]:

image classification (I)

use accuracy or error rate as metric

item/batch tfms

this randomresizedcrop is the most commonly used one

In [ ]:
```
anim = animals_datablock.new(item_tfms=Resize(128, ResizeMethod.Squish))
# that bracket options has
item_tfms=Resize(128, ResizeMethod.Pad, pad_mode='zeros')
item_tfms=RandomResizedCrop(128, min_scale=0.3)
item_tfms=Resize(128), batch_tfms=aug_transforms(mult=2)
# thoda changed the prev animals batablock
dls = anim.dataloaders(path)
dls.valid.show_batch(max_n=4, nrows=1, unique = True) # could do train also
```

In [ ]:
```
learn = cnn_learner(dls, resnet18, metrics=error_rate)#training our model
learn.fine_tune(4)
# epoch train_loss      valid_loss      error_rate      time
# 0     1.029151        0.248961        0.100000        00:02
# epoch train_loss      valid_loss      error_rate      time
# 0     0.552188        0.134259        0.050000        00:03
# 1     0.373990        0.071003        0.025000        00:03
# 2     0.282097        0.051770        0.025000        00:03
# 3     0.219080        0.035745        0.000000        00:03
```

confusion matrix

In [ ]:
```
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix( figsize=(12,12), dpi=60 )
```

In [ ]:
```
interp.plot_top_losses(5, nrows=1)
interp.most_confused(min_val=5)
```

clean data

In [ ]:
```
cleaner = ImageClassifierCleaner(learn)
for idx in cleaner.delete(): cleaner.fns[idx].unlink()
for idx,cat in cleaner.change(): shutil.move(str(cleaner.fns[idx]), path/cat)
```

In [ ]:
```
# !pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
```

```
from fastbook import *
from fastai.vision.all import *
from jmd_imagescraper.core import *
from pathlib import Path
from jmd_imagescraper.imagecleaner import *
```

In [ ]:

```

```

using regex for datablock

In [ ]:

```
pets = DataBlock(blocks = (ImageBlock, CategoryBlock),
                 get_items=get_image_files,
                 splitter=RandomSplitter(seed=42),
                 get_y=using_attr(RegexLabeller(r'(.+)_\d+\.jpg$'), 'name'),
                 # names were like 'miniature_pinscher_199.jpg', so we get miniature_pinscher
                 item_tfms=Resize(460),
                 batch_tfms=aug_transforms(size=224, min_scale=0.75))
dls = pets.dataloaders(path/"images")
dls, dls.vocab
# (<fastai.data.core.DataLoaders at 0x7f640086fd30>, ['Abyssinian', 'Bengal', 'Birman',...]
```

batch size

In [ ]:

```
dls = pets.dataloaders(path/"images", bs=16) # or 32 or 64
```

Presizing (item_tfms, batch_tfms me jo hai)

Presizing is a particular way to do image augmentation that is designed to minimize data destruction while maintaining good performance.

To implement this process in fastai you use Resize as an item transform with a large size, and RandomResizedCrop as a batch transform with a smaller size. RandomResizedCrop will be added for you if you include the min_scale parameter in your aug_transforms function, as was done in the DataBlock call in the previous section. Alternatively, you can use pad or squish instead of crop (the default) for the initial Resize.

another type of learner!

In [ ]:

```
model = xresnet50(n_out=dls.c) #yaha simply resnet 34 daalte the pehle!
learn = Learner(dls, model, loss_func=CrossEntropyLossFlat(), metrics=accuracy)
```

In [ ]:

```
learn = cnn_learner(dls, resnet34, metrics=error_rate)
# observe we still use error rate kyuki image classification hai
learn.fine_tune(2)
# epoch train_loss      valid_loss      error_rate      time
# 0     1.475844        0.377749        0.115697        01:42
# epoch train_loss      valid_loss      error_rate      time
# 0     0.509520        0.337518        0.102165        02:11
# 1     0.340418        0.236828        0.072395        02:12
```

summary method to check if transforms are working

In [ ]:
```
pets.summary(path/"images")
# will give the entire process step by step
```

cross entropy loss

image classification ke liye it is chosen as default

has two benefits:

- **It works even when our dependent variable has more than two categories.**
- **It results in faster and more reliable training.**

checking labels, activations

In [ ]:
```
x,y = dls.one_batch() #sirf fitting ke baad, not yet trained
len(x), x
# (64,
#  TensorImage([[[[ 5.3097e-01,  5.4482e-01,  5.6834e-01,  ...,  1.4564e-02, -1.8605e-02, -6.0605e-02],
#             [ 5.1453e-01,  5.0940e-01,  5.3032e-01,  ...,  4.9254e-02,  1.8271e-03, -3.5908e-02],
#             [ 5.0639e-01,  4.9789e-01,  4.9978e-01,  ...,  7.6806e-02, -8.7815e-03, -9.6111e-02],
len(y), y
# (64,
# (TensorCategory([ 8, 25, 21, 33,...)
preds,_ = learn.get_preds(dl=[(x,y)])
preds[0], len(preds[0])
# (tensor([1.1741e-02, 6.2871e-05, 1.0333e-02, 2.5886e,.....], 37)
# since we have 37 breeds
```

In [ ]:
```

```

softmax

softmax is the multi-category equivalent of `sigmoid` —we have to use it any time we have more than two categories and the probabilities of the categories must add to 1, and we often use it even when there are just two categories, just to make things a bit more consistent.

the softmax function really wants to pick one class among the others, so it's ideal for training a classifier when we know each picture has a definite label.

log likelihood

Just as we moved from sigmoid to softmax, we need **to extend the loss function** (mnist loss) to work with more than just binary classification—**it needs to be able to classify any number of categories** (in this case, we have 37 categories)

In [ ]:
```
loss_func = nn.CrossEntropyLoss()
loss_func(acts, targ) # tensor(1.8045)
F.cross_entropy(acts, targ) # tensor(1.8045)
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:

In [ ]:

image classification(II)

In [ ]:
```
path = '/Intro_Data/image_classifier/images'
# path basically jaha bhi image file hai
```

The filenames start with an uppercase letter if the image is a cat, and a lowercase letter otherwise.

In [ ]:
```
def is_cat(x): return x[0].isupper()
# is_cat labels cats based on a filename rule provided by the dataset creators
```

We have to tell fastai how to get labels from the filenames, which we do by calling **from_name_func** (which means that labels can be extracted using a function applied to the filename), and passing is_cat, which returns x[0].isupper(), which evaluates to True if the first letter is uppercase (i.e., it's a cat).

ImageDataLoaders The first part of the class name will generally be the type of data you have, such as image, or text.

item_tfms are applied to each item (in this case, each item is resized to a 224-pixel square), while **batch_tfms** are applied to a batch of items at a time using the GPU

In [ ]:
```
dls = ImageDataLoaders.from_name_func(
    path, get_image_files(path), valid_pct=0.2, seed=42,
    label_func=is_cat, item_tfms=Resize(224))
```

Fit the model

provide at least one piece of information: how many times to look at each image (known as number of epochs). 2 steps: pehle one epoch to karna hi hai to adjust the new random layer then, use given number of epochs

When we create a model from a pretrained network fastai automatically freezes all of the pretrained layers for us. When we call the `fine_tune` method fastai does two things:

- **Trains the randomly added layers for one epoch, with all other layers frozen**
- **Unfreezes all of the layers, and trains them all for the number of epochs requested**

Fine-tune Update a pretrained model for a different task

In [ ]:
```
learn = cnn_learner(dls, resnet34, metrics=error_rate)
learn.fine_tune(1)
# create a convolutional neural network (CNN)
```

When using a pretrained model, cnn_learner will remove the last layer, since that is always specifically customized to the original training task (i.e. ImageNet dataset classification), and replace it with one or more new layers with randomized weights, of an appropriate size for the dataset you are working with.

Metrics: error_rate, accuracy (1-error_rate)

working with images

to open image

In [ ]:
```
img = PILImage.create(image_cat())
```

```
img.to_thumb(192)

# OR

im = Image.open('Production_Data/images/bear.jpg')
im.thumbnail((256,256))
im
```

to download and store

```
In [ ]:
ims = ['http://3.bp.blogspot.com/-S1scRCkI3vY/UHzV2kucsPI/AAAAAAAAA-k/YQ5UzHEm9Ss/s1600/Grizzly%2BBear%2BWildlife.jpg']
dest = 'Production_Data/images/grizzly.jpg'
download_url(ims[0], dest)
im = Image.open(dest)
im.to_thumb(128,128)
```

convert image to array

```
In [ ]:
array(im)
```

to see with intensity

```
In [ ]:
im3_t = tensor(im3)
df = pd.DataFrame(im3_t[4:15,4:22])
#df.style.set_properties(**{'font-size':'6pt'}).background_gradient('Greys')
df.style.set_properties(**{'font-size':'6pt'}).background_gradient('Blues')
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
animals=['lion','tiger']
path = Path('Production_Data/animals')
duckduckgo_search(path,"lion_images","lion",max_results=100)
# downloads 100 images of 'cats' and saves it in path/cat
# matlab lion is the search query
```

```
In [ ]:
fns=get_image_files(path)
(#200) [Path('Production_Data/animals/tiger_images/079_c3302824.jpg'),
    Path('Production_Data/animals/tiger_images/004_5c8f6702.jpg'),...]

# OR
(path/'train').ls(), (path/'train').ls().sorted()  #type se bhi wo list milti hai
```

```
failed = verify_images(fns)
(#0) []
failed.map(Path.unlink);
```

In [ ]:
```
is_cat,_,probs = learn.predict(img) #_ has prediction_index
is_cat, _, probs
# ('True', tensor(1), tensor([7.2762e-19, 1.0000e+00]))
```

In [ ]:
```
valid_3_tens = torch.stack([tensor(Image.open(o))
                            for o in (path/'valid'/'3').ls()])
valid_3_tens = valid_3_tens.float()/255
valid_3_tens.shape #gives torch.Size([1010, 28, 28])
def mnist_distance(a,b): return (a-b).abs().mean((-1,-2))
# gives average out of each levels if height(1 only here), so 28*28 shape
mnist_distance(a_3, mean3) #tensor(0.1114)
# for every image, we averaged the intensity of all the pixels in that image.
```

In [ ]:
```
valid_3_dist = mnist_distance(valid_3_tens, mean3)
valid_3_dist, valid_3_dist.shape
#(tensor([0.1787, 0.1422, 0.1412,  ..., 0.1358, 0.1301, 0.1110]), torch.Size([1010]))
```

The magic trick is that PyTorch, when it tries to perform a simple subtraction operation between two tensors of different ranks, will use broadcasting. That is, it will automatically expand the tensor with the smaller rank to have the same size as the one with the larger rank. Broadcasting is an important capability that makes tensor code much easier to write.

SGD

derivatives

In [ ]:
```
xt = tensor([3.,4.,10.]).requires_grad_()
def f(x): return (x**2).sum()

yt = f(xt)
yt.backward()
xt.grad, yt #(tensor([ 6.,  8., 20.]), tensor(125.
```

learning rate

Once picked a learning rate, you can adjust your parameters : **w -= gradient(w) * lr**

full process

In [ ]:
```
dset = list(zip(train_x,train_y))
# A DataLoader can take any Python collection and turn it
# into an iterator over many batches
dl = DataLoader(dset, batch_size=256)
```

```
valid_dl = DataLoader(valid_dset, batch_size=256)
xb,yb = first(dl) # first is literally the first item in the dataloader
xb.shape,yb.shape # (torch.Size([256, 784]), torch.Size([256, 1]))
```

In [ ]:
```
dls = DataLoaders(dl, valid_dl)
```

In [ ]:
```
simple_net = nn.Sequential(
    nn.Linear(28*28,30),
    nn.ReLU(),
    nn.Linear(30,1) )
```

In [ ]:
```
def mnist_loss(predictions, targets):
    predictions = predictions.sigmoid() # so that they are between 0 and 1
    return torch.where(targets==1, 1-predictions, predictions).mean()
```

In [ ]:
```
def batch_accuracy(xb, yb):
    preds = xb.sigmoid()
    correct = (preds>0.5) == yb
    return correct.float().mean()
```

In [ ]:
```
learn = Learner(dls, simple_net, opt_func=SGD,
                loss_func=mnist_loss, metrics=batch_accuracy)
learn.fit(10, lr=lr)

epoch   train_loss      valid_loss      batch_accuracy  time
# 0     0.269239        0.415576        0.504907        00:00
# 1     0.132732        0.215948        0.819921        00:00
# 2     0.076132        0.112346        0.916585        00:00
# 3     0.051601        0.077263        0.940137        00:00
# .
# .
# 36    0.014736        0.021339        0.982336        00:00
# 37    0.014604        0.021188        0.982336        00:00
# 38    0.014476        0.021045        0.982336        00:00
# 39    0.014352        0.020909        0.982336        00:00

# And we can view the final accuracy:
learn.recorder.values[-1][2], learn.recorder.values[-1]
# [0.01435244083404541,0.020909365266561508,0.98233562707901])
```

deeper nn

In [ ]:
```
dls = ImageDataLoaders.from_folder(path)
learn = cnn_learner(dls, resnet18, pretrained=False,
                    loss_func=F.cross_entropy, metrics=accuracy)
learn.fit_one_cycle(1, 0.1)
# epoch train_loss      valid_loss      accuracy        time
# 0     0.156485        0.020939        0.996565        00:20
```

segmentation

Creating a model that can recognize the content of every individual pixel in an image

In [ ]:
```
dls = SegmentationDataLoaders.from_label_func(
    path, bs=8,
    fnames = get_image_files(path/'images'),
    label_func = lambda o: path/'labels'/f'{o.stem}_P{o.suffix}',
    codes = np.loadtxt(path/'codes.txt', dtype=str)
)
# LABEL FUNC IS SIMPLY A FUNC
# THIS IS USING from_labels_func and not from_names_func,
# WAISI HI NEED HOGI BHAI YAHA
```

In [ ]:
```
learn = unet_learner(dls, resnet34)
learn.fine_tune(8)
```

gives

In [ ]:
```
# epoch train_loss      valid_loss      time
# 0     2.669002        2.249490        00:04


# epoch train_loss      valid_loss      time
# 0     1.896676        1.560382        00:04
# 1     1.626508        1.179197        00:04
# 2     1.588799        1.314724        00:04
# 3     1.463475        1.137042        00:04
# 4     1.335701        1.004593        00:04
# 5     1.230620        0.892159        00:04
# 6     1.133300        0.867039        00:04
# 7     1.051864        0.824869        00:04
```

We can visualize how well it achieved its task, by asking the model to color-code each pixel of an image. As you can see, it nearly perfectly classifies every pixel in every object. For instance, notice that all of the cars are overlaid with the same color and all of the trees are overlaid with the same color (in each pair of images, the lefthand image is the ground truth label and the right is the prediction from the model):

In [ ]:
```
learn.show_results(max_n=6, figsize=(7,8))
```

In [ ]:

```

```

In [ ]:

```

```

NLP

In [ ]:

```
from fastai.text.all import *
torch.backends.cudnn.enabled = False
path = untar_data(URLs.IMDB)
```

In [ ]:

```
dls = TextDataLoaders.from_folder(untar_data(URLs.IMDB), valid='test', bs=16)
learn = text_classifier_learner(dls, AWD_LSTM, drop_mult=0.5, metrics=accuracy)
learn.fine_tune(4, 1e-2)
```

In [ ]:

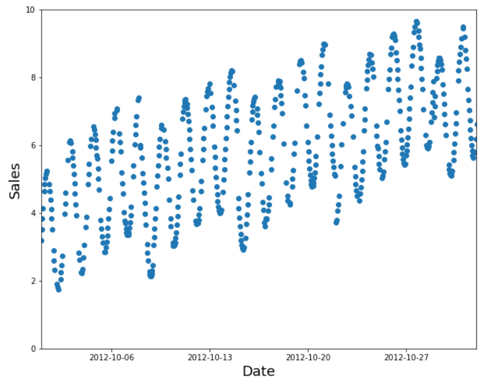
```
learn.predict("I really liked that movie!") #ESSENTIALLY CAPTURING THE SENTIMEMNT OF THE MOVIEW REVIEW
```

In [ ]:

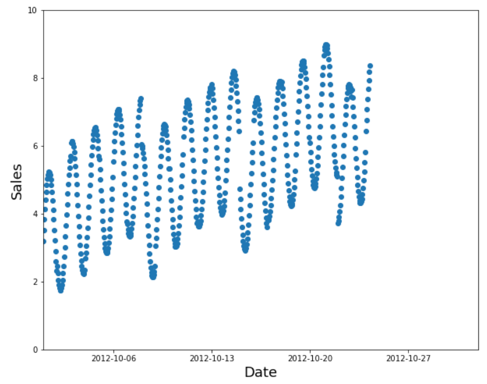
```

```

A **random subset is a poor choice** (too easy to fill in the gaps, and not indicative of what you'll need in production)



Instead, use the earlier data as your training set (and the later data for the validation set)



In [ ]: