W1 NOTES

Genel bir intro yapılıyor, kurs'u bir kitap gibi çalışabilmek için fastbook notebooklarına ulaşabilirsin: *https://github.com/fastai/fastbook*

Ben şuan videoları takip edeceğim, teknik ve önemli gördüğüm, uygulamada işime yarayacak yerleri not etmeyi planlıyorum, en nihayetinde amacım çalışır ve kullanışlı deep learning projeleri ortaya koymak olacak.

Aşağıda örnek bir image classifier eğitilmiş, 0.002 hataya çekilmiş.

```
from fastai.vision.all import *
path = untar_data(URLs.PETS)/'images'

def is_cat(x): return x[0].isupper()
dls = ImageDataLoaders.from_name_func(
    path, get_image_files(path), valid_pct=0.2, seed=42,
    label_func=is_cat, item_tfms=Resize(224))

learn = cnn_learner(dls, resnet34, metrics=error_rate)
learn.fine_tune(1)
```

```
epoch  train_loss  valid_loss  error_rate  time
    0    0.169390    0.021388    0.005413  00:14
epoch  train_loss  valid_loss  error_rate  time
    0    0.058748    0.009240    0.002706  00:19
```

Yukarıdaki classifier kodunu line by line inceleyelim:

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

This gives us all of the functions and classes we will need to create a wide variety of computer vision models.

```
path = untar_data(URLs.PETS)/'images'
```

Downloads a standard dataset from the fast.ai datasets collection (if not previously downloaded) to your server, extracts it (if not previously extracted), and returns a `Path` object with the extracted location:

```
def is_cat(x): return x[0].isupper()
```

We define a function, `is_cat`, labels cats based on a filename rule provided by the dataset creators:

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

Burada dataloaders elde ediliyor, fastai model eğitmek için datayı bu formda istiyor. Farklı dataset tipleri için farklı classlar kullanılır burada datamız images olduğu için ImageDataLoaders kullanıldı.

Verilerden labeling yapmak için de farklı yollar var, burada from_name_funct'ı kullanıyoruz, yani her bir image'ın ismi belirttiğimiz label_func içerisine parametre olarak verilecek ve bu fonksiyona göre labeling yapılacak.

A `Transform` contains code that is applied automatically during training; fastai includes many predefined `Transform`s, and adding new ones is as simple as creating a Python function. There are two kinds: `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, so they're particularly fast (we'll see many examples of these throughout this book).

224'den farklı boyutlar da kullanılabilir daha büyük kullanırsak computational cost artar ama accuracy de muhtemelen artar.

`valid_pct=0.2`. This tells fastai to hold out 20% of the data and *not use it for training the model at all.*

The parameter `seed=42` sets the *random seed* to the same value every time we run this code, which means we get the same validation set every time we run it—this way, if we change our model and retrain it, we know that any differences are due to the changes to the model, not due to having a different random validation set.

```
learn = cnn_learner(dls, resnet34, metrics=error_rate)
```

Training our image recognizer tells fastai to create a *convolutional neural network* (CNN) and specifies what *architecture* to use (i.e. what kind of model to create), what data we want to train it on, and what *metric* to use:

A *metric* is a function that measures the quality of the model's predictions using the validation set, and will be printed at the end of each *epoch*. In this case, we're using `error_rate`, which is a function provided by fastai that does just what it says: tells you what percentage of images in the validation set are being classified incorrectly. Another common metric for classification is `accuracy` (which is just `1.0 - error_rate`).

The concept of a metric may remind you of *loss*, but there is an important distinction. The entire purpose of loss is to define a "measure of performance" that the training system can use to update weights automatically. In other words, a good choice for loss is a choice that is easy for stochastic gradient descent to use. But a metric is defined for human consumption, so a good metric is one that is easy for you to understand, and that hews as closely as possible to what you want the model to do. At times, you might decide that the loss function is a suitable metric, but that is not necessarily the case.

`cnn_learner` also has a parameter `pretrained`, which defaults to `True` (so it's used in this case, even though we haven't specified it), which sets the weights in your model to values that have already been trained by experts to recognize a thousand different categories across 1.3 million photos (using the famous *ImageNet* dataset). A model that has weights that have already been trained on some other dataset is called a *pretrained model*. You should nearly always use a pretrained model, because it means that your model, before you've even shown it any of your data, is already very capable.

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. This last part of the model is known as the *head*.

```
learn.fine_tune(1)
```

tells fastai how to *fit* the model.

As we've discussed, the architecture only describes a *template* for a mathematical function; it doesn't actually do anything until we provide values for the millions of parameters it contains.

In order to fit a model, we have to provide at least one piece of information: how many times to look at each image (known as number of *epochs*). The number of epochs you select will largely depend on how much time you have available, and how long you find it takes in practice to fit your model. If you select a number that is too small, you can always train for more epochs later.

But why is the method called `fine_tune`, and not `fit`? fastai actually *does* have a method called `fit`, which does indeed fit a model (i.e. look at images in the training set multiple times, each time updating the parameters to make the predictions closer and closer to the target labels). But in this case, we've started with a pretrained model, and we don't want to throw away all those capabilities that it already has. As you'll learn in this book, there are some important tricks to adapt a pretrained model for a new dataset—a process called *fine-tuning*.

Fine-tuning: A transfer learning technique where the parameters of a pretrained model are updated by training for additional epochs using a different task to that used for pretraining.

When you use the `fine_tune` method, fastai will use these tricks for you. There are a few parameters you can set (which we'll discuss later), but in the default form shown here, it does two steps:

1.Use **one epoch** to fit just those parts of the model necessary to get the new random head to work correctly with your dataset.

2.Use the number of epochs requested when calling the method to **fit the entire model**, updating the weights of the later layers (especially the head) faster than the earlier layers (which, as we'll see, generally don't require many changes from the pretrained weights).

The *head* of a model is the part that is newly added to be specific to the new dataset. An *epoch* is one complete pass through the dataset. After calling `fit`, the results after each epoch are printed, showing the epoch number, the training and validation set losses (the "measure of performance" used for training the model), and any *metrics* you've requested (error rate, in this case).