

## W2 NOTES

Bu hafta için önemli gördüğüm yerleri not edeceğim. Dersin büyük kısmında genel konseptlerden bahsediyor. Özellikle p-value kavramından ve işe yaramaz olduğundan uzunca bahsetmiş, bu tam olarak ders ile ilgili değil covid ile ilgili olduğu için paylaşmış.

### Intro

Jeremy bazen training ve validation loss'a bakınca overfitting görünse dahi validation accuracy'sinin iyiye gidebileceğini söylüyor o halde overfitting demek her zaman işler kötü gidiyor demek değil, modelimizin performansı iyiyi gittiği sürece loss'un kötüye gitmesi çok da önemli değil, sonuçta bizim için önemli olan error\_metric olacak, model loss'u sadece GD için önemli.

### Fine Tuning

Bir transfer learning tekniğidir. Temelde pretrained modeli biraz daha eğiterek kendi problemimize uygun hale getirmektir. Hem yeni head kısmı eğitilir hem de eskiden kalan body kısmı.

Fastai'in default `fine_tune(n)` methodunu kullanırsak bunun yapacağı işlem şu: yeni eklenen head kısmını bir epoch eğitir. Daha sonra tüm modeli n epoch eğitir, bu tüm modeli eğitme kısmında değişken learning rates kullanılır ve sona yakın layer'lar yüksek lr ile eğitilir.

### Bing Image Search API

Anladığım kadarıyla bu API'ı image'ları indirmek ve kendimize bir dataset oluşturmak için kullanıyoruz. Bunu yapmanın başka yolları da var yapacağın zaman araştıır.

Bing Image Search kullanacaksak, üye olup bir API key alıyoruz ve işlerimizi bu API key ile yapıyoruz. İkinci haftanın notebook'unda gathering data başlığı altında key'i XXX kısmına yazıyoruz artık keyimiz hazır.

Daha sonra `search_images_bing` fonksiyonunu kullanacağız bu fonksiyon bing images ile arama yapıyor ve bulunan resimlerin url'lerini return ediyor. Default olarak 150 url return ediliyor.

Sonuç olarak bu kod bloğuna bakarsın, arama yapıp url'leri kaydediyor, donwload ediyor, bunları klasörler içerisine kaydediyor vesaire ve dataset'imizi hazırlıyoruz.

Şimdi sıradaki adım bu dataset'i kullanarak bir model eğitmek olacak.

## From data to DataLoaders

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

Eğer klasik bir data yapısı ile uğraşıyorsak factory method ile dataset'i dataloaders'a çevirebiliriz ama daha flexible bir yol da var bu da data block API kullanmak. Bu api ile veriyi nereden alacağımızı, nasıl label edeceğimizi, nasıl bir validation set olacağını vesaire hepsini detaylı bir şekilde set edebiliyoruz, ve kendi datasetimizi dataloaders'a çevirebiliyoruz.

Here is what we need to create a `DataLoaders` for the dataset that we just downloaded:

```
bears = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=Resize(128))
```

Bu kodu satır satır inceleyip anlayalım.

Öncelikle independent ve dependent variables için nasıl tipler istediğimizi belirtiyoruz, yani veri tipimiz ve model çıktımız nasıl olacak:

```
blocks=(ImageBlock, CategoryBlock)
```

Modelimiz images alacak ve çıkışında farklı kategorilerlerden birini verecek!

For this `DataLoaders` our underlying items will be file paths. We have to tell fastai how to get a list of those files. The `get_image_files` function takes a path, and returns a list of all of the images in that path (recursively, by default):

```
get_items=get_image_files
```

Validation seti ayırmanın çok farklı yolları vardır, zaten farklı klasör ile ayrılmış olabilir, csv dosyası içerisinde ayrılmış olabilir vesaire. Bu örnekte ayrı bir klasör yok, kendimiz random olarak ayıracağız ve her seferinde aynı ayrılсын diye bir random seed ektik.

```
splitter=RandomSplitter(valid_pct=0.2, seed=42)
```

Here, we are telling fastai what function to call to create the labels in our dataset:

```
get_y=parent_label
```

parent\_label fonksiyonu fastai'ın bir fonksiyonu ve temelde yaptığı şey aldığı image'in folder name'ini label olarak set eder.

---

Son olarak item transforms belirtiyoruz, burada belirtilen transformlar tüm itemlara uygulanacak, bu örnek için sadece resize yapılmış böylece tüm images aynı boyutta olacak bu da mini-batches yaratmakta faydalı olacaktır.

```
item_tfms=Resize(128)
```

---

Sonuçta yukarıdaki kod bloğu ile elimizde bir DataBlock objesi olacak, bunu dataloaders'a çevirmek istiyoruz:

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

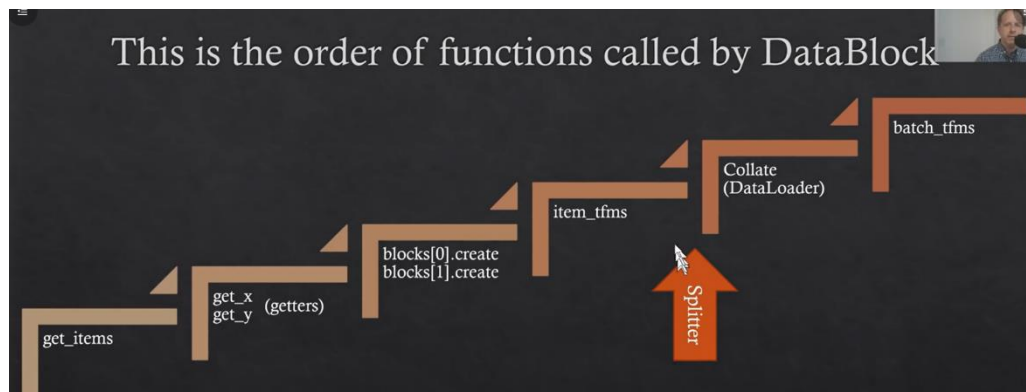
---

**A DataLoaders includes validation and training DataLoaders. DataLoader is a class that provides batches of a few items at a time to the GPU.**

We'll be learning a lot more about this class in the next chapter. When you loop through a DataLoader fastai will give you 64 (by default) items at a time, all stacked up into a single tensor. We can take a look at a few of those items by calling the show\_batch method on a DataLoader.

---

Aşağıda DataBlock api'nin işlemleri hangi sıra ile yaptığı gösterilmiş.



## Show Batch and Transforms (Jeremy buradan sonrasını 3. derste anlatıyor)

Eğer dataloaders'ım oluştuysa artık bu dataloaders'ı modelimi eğitmek için kullanabilirim ancak ondan önce, modele beslenecek verilerime gözetebilirim:



Dataloaders oluştururken Resize transform'unu kullanmıştık bu transform default olarak image'ların içinden square crop eder. Yani size 128 verildiyse resmin tam ortasından 128x128 bir kare crop edilir elbette bunu yapınca image'ın içindeki bazı bilgiler kaybolacaktır. Bu yüzden alternatif olarak padding ya da squish/streching yapabiliriz.

```
[ ] bears = bears.new(item_tfms=Resize(128, ResizeMethod.Squish))  
dls = bears.dataloaders(path)  
dls.valid.show_batch(max_n=4, nrows=1)
```



```
[ ] bears = bears.new(item_tfms=Resize(128, ResizeMethod.Pad, pad_mode='zeros'))  
dls = bears.dataloaders(path)  
dls.valid.show_batch(max_n=4, nrows=1)
```



Fakat bu yaklaşımlar genelde ideal değildir. Bunun yerine her epoch için image'lardan random olarak bir crop alınır, yani bazen tam oradan alınır bazen köşeden alınır vesaire, böylelikle daha çeşitli bir veriseti oluşturmuş oluruz training setimiz genel datayı daha iyi represente edebilir.

Yeni yaklaşımda bir min\_scale parametresi devreye girer, bu parametre her random crop'un image'in en az kaçta kaçını içermesi gerektiğini söyler. Yani eğer buna 0.5 dersek her seferinde orjinal image'in %50'sini içine alacak zoom yaparak croplar alacak.

Meslema aşağıdaki örnekte show\_batch içerisinde unique=True diyerek aynı örneğin farklı croplamalarının nasıl olacağını görüyoruz:

```
[ ] bears = bears.new(item_tfms=RandomResizedCrop(128, min_scale=0.3))
dls = bears.dataloaders(path)
dls.train.show_batch(max_n=4, nrows=1, unique=True)
```



## Data Augmentation

Examples of common data augmentation techniques for images are **rotation**, **flipping**, **perspective warping**, **brightness changes** and **contrast changes**.

Doğal fotoğraflar için aug\_transforms fonksiyonu standard augmentationları sağlıyor.

Item\_transforms tek tek her item üzerine uygulanıyordu, ancak resize ile itemler aynı boyuta getirildikten sonra augmentation için batch\_transforms kullanabiliriz bu transformlar toplu olarak batches üzerinde yapılacak ve çok daha pratik olacak.

Aşağıda aug\_transforms ile default data augmentation uygulanmış, mult=2 diyerek etki default'un 2 katına çıkarılmış, random crop da kullanmadık ki etkiyi daha rahat görebilelim. Aynı image modele her epoch için bambaşka şekillerde input edilecek.



Augmentation sadece training set üzerinde yapılır.

## Train the Model

Yukarıdaki örnekleri transformları ve augmentation'ı anlamak için yaptık, şimdi yeni bir datablock oluşturalım bundan yeni dataloaders oluşturalım ve modelimizi eğitelim:

224x224 random sized crop images kullanılacak:

```
bears = bears.new(  
    item_tfms=RandomResizedCrop(224, min_scale=0.5),  
    batch_tfms=aug_transforms()  
)  
dls = bears.dataloaders(path)
```

Resnet18 architecture kullanılacak, model fine\_tune edilecek önce bir epoch head kısmı sonra tüm model farklı learning rate'ler ile eğitilecek:

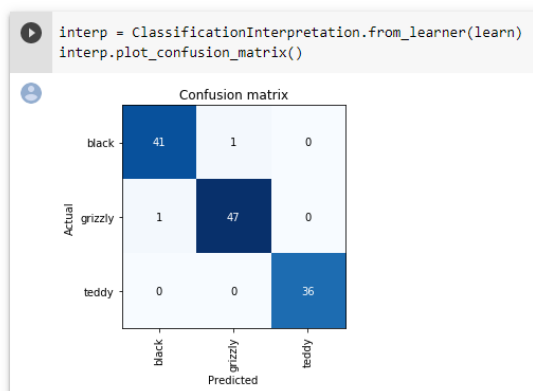
```
[ ] learn = cnn_learner(dls, resnet18, metrics=error_rate)  
learn.fine_tune(4)
```

epoch	train_loss	valid_loss	error_rate	time
0	1.235733	0.212541	0.087302	00:05
epoch	train_loss	valid_loss	error_rate	time
0	0.213371	0.112450	0.023810	00:05
1	0.173855	0.072306	0.023810	00:06
2	0.147096	0.039068	0.015873	00:06
3	0.123984	0.026801	0.015873	00:06

Modelimiz %98.5 accuracy'e ulaşıyor, gayet iyi!

## Model Interpretation

Model performansını daha iyi gözlemlemek için, confusion matrix'ı ve top\_losses'ı plot edebiliriz:





Confusion matrix validation set için tüm sınıflardan kaç örnek olduğunu bunların kaç tanesinin doğru kaç tanesinin yanlış tahmin edildiğini yukarıdaki gibi gösteren önemli bir tablo.

Modeli interpret edebileceğimiz bir diğer yol ise `plot_top_losses()` fonksiyonu:

```
[ ] interp.plot_top_losses(5, n_rows=1)
```

#### Prediction/Actual/Loss/Probability



Model bir örnek için cevabını yüksek confidence ile verip yanılırsa loss o örnek için büyük olur. Doğru cevap verip emin olmamak da görece yüksek loss verir. Emin olmadan yanlış cevap vermek ise emin olarak yanlış cevap vermekten az loss verir.

Yukarıdaki fonksiyon (sanıyorum) validation set için modelin en büyük loss ile tahmin ettiği örnekleri bize gösterir bu da bize bir fikir verir.

## Data Cleaner

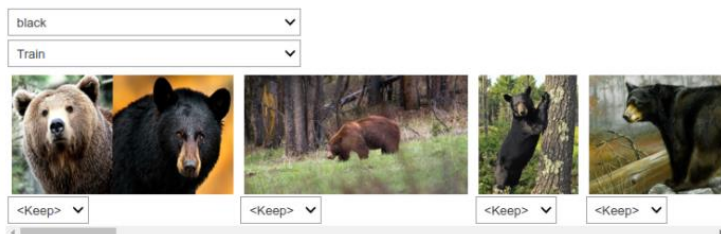
Data cleaning'in en başta yapılacağını düşünebiliriz ancak aslında eğer önce temizlenmemiş dataset ile basit bir model eğitirsek sonrasında bu basit model bize data cleaning yapmak için yardımcı oluyor.

Bu nasıl oluyor? Şöyle ki artık elimizdeki basit model yukarıdaki mantık ile training ve validation set için bize en yüksek loss veren örnekleri gösteriyor. Tüm images'ı higher loss'dan lower loss'a doğru sıralıyor, bu şekilde yanlış label edilmiş veya modelin ayırması zaten imkansız olan dataları ImageClassifierCleaner ile temizleyebiliriz.

Bu GUI kullanılarak istenilirse verilerin label'ı değiştirilir, istenilirse veriler silinir. Ancak unutma ki bu veriler gerçekten silinmiyor veya orjinal dataset klasörlerinde konumu değişmiyor, bu değişim dataloaders üzerinde oluyor!!!

```
[ ] #hide_output
cleaner = ImageClassifierCleaner(learn)
cleaner

VBox(children=(Dropdown(options=('black', 'grizzly', 'teddy'), value='black'), Dropdown(options=('Train', 'Val...
```



Değişiklikleri GUI üzerinde yaptıktan sonra:

to delete (`unlink`) all images selected for deletion, we would run:

```
for idx in cleaner.delete(): cleaner.fns[idx].unlink()
```

To move images for which we've selected a different category, we would run:

```
for idx, cat in cleaner.change(): shutil.move(str(cleaner.fns[idx]), path/cat)
```

Bu tool sayesinde artık cleaned data ile yeni bir model eğitirsek modelimizin accuracy'si %100 olur. Bu tool çok büyük kolaylık sağlar!



## Saving and Loading the Learner

Modelimizi eğittikten sonra production için kullanmak amacıyla bunu kaydedebiliriz. Kaydedeceğimiz iki şey var: architecture ve trained parameters. İkisini de kaydetmek için export methodu kullanılabilir.

Hatta dataloaders'ın nasıl yaratıldığı bile kaydedilir ki yeni bir örnek modele beslendiğinde hangi transformların uygulanacağı belli olsun. Bu amaçla validation set DataLoader kaydedilir. Anladığım kadarıyla validation set'e genelde sadece resize transform'u uygulanır, augmentation uygulanmaz.

Aşağıdaki method çalıştırıldığında "export.pkl" dosyası oluşturulacak ve kaydedilecek.

```
learn.export()
```

Kaydedilen export.pkl dosyası app'imizi deploy etmek için kullanacağımız dosya olacak, bu dosyadan learner'ı geri yükleyip learner'a yeni datayı predict ettirebiliriz!

Aşağıdaki gibi load\_learner methodu ile learner inference'imiz çağırılır, learner inference training için değil sadece predictions için kullanılır.

```
learn_inf = load_learner(path/'export.pkl')
```

Şimdi bu learner inference'e bir prediction yaptıralım:

```
learn_inf.predict('images/grizzly.jpg')
```

Bunun sonucunda predicted category, index of the predicted category ve probability for each category aşağıdaki gibi gösterilir:

```
('grizzly', tensor(1), tensor([9.0767e-06, 9.9999e-01, 1.5748e-07]))
```

Kategorileri görmek için aşağıdaki methodu kullanabiliriz:

```
learn_inf.dls.vocab
```

Artık bir modeli nasıl kaydedeceğimizi ve kaydedilen modeli tekrar nasıl load edip prediction yaptırabileceğimizi gördük.

Bir online app için elimizde önemli toollar var. Şimdi bir online app'i jupyter notebook kullanarak nasıl yapabileceğimizi göreceğiz!

## Turning Our Model into an Online Application

It turns out that we can create a complete working web application using nothing but Jupyter notebooks! The two things we need to make this happen are:

- IPython widgets (ipywidgets)
- Voilà

IPython widgets are GUI components that bring together JavaScript and Python functionality in a web browser, and can be created and used within a Jupyter notebook.

However, we don't want to require users of our application to run Jupyter themselves. That is why Voilà *exists*. It is a system for making applications consisting of IPython widgets available to end users, without them having to use Jupyter at all.

Yani özetle widgets kullanarak notebook üzerinde GUI oluşturacağız, daha sonra Voila kullanarak bunları web applications'a çevirebileceğiz.

Şimdi widgets'ı biraz anlayalım.

### IPython Widgets

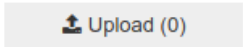
#### FileUpload Widget

Aşağıdaki satır ile bir FileUpload butonu yarattım.

```
btn_upload = widgets.FileUpload()
```

Bunu jupyter notebook üzerinde görüntülemek istersem, aşağıdaki gibi butonun ismini girerim ve bir buton görünür, artık bu butonu file upload için kullanabilirim, upload yapılırca butonun yanındaki 0 sayısı artar.

```
btn_upload
```



Bu widgetların farklı method ve propertyleri vardır, örneğin btn\_upload üzerinden data property'sine ulaşabiliriz. Bu property upload edilen tüm images'i içeren bir arrayden fazlası değil.

Bu arrayin son elemanına ulaşalım ve bunu PILImage.create methoduna besleyelim böylece upload edilen resmi görüntüleyebiliriz:

```
[ ] img = PILImage.create(btn_upload.data[-1])
```



## Output Widget

File upload widget'ını anladık şimdi bir başka widget'a bakalım, **Output** widget'ı. Aşağıdaki şekilde bir output widget'ı yarattım, bunun içine istediğim şeyi daha sonra koyabilirim. Yani output widget'ı bir placeholder gibi, daha sonra içini doldurabileceğim bir yapı:

```
out_pl = widgets.Output()
```

Örneğin bu yapıyı aşağıdaki gibi oluşturduk, içini boşalttık sonra bastırmak istedik diyelim. Ancak şuan jupyter üzerinde hiçbir çıktı alamayız çünkü output widget'ın içi boşdur. Ancak unutma out\_pl nerede yazıldıysa aslında output widget orada duruyor! Sadece içi boş, eğer içi ileride bir zaman doldurulursa, out\_pl'nin yazıldığı yerin hemen altında output'u göreceğiz!!!

```
out_pl = widgets.Output()
out_pl.clear_output()
out_pl
```

Yani diyelim ki bir kaç satır sonra gittik ve out\_pl'nin içerisine 128,128 olacak şekilde upload edilen imageımızı verdik:

```
with out_pl: display(img.to_thumb(128,128))
```

Şimdi image with satırının altında değil üstünde, out\_pl'nin hemen altında görünecek, çünkü widget oraya yerleştirildi!

## Label Widget

Bu da output widget'a benzer şekilde bir placeholder'dır, içi daha sonra doldurulabilir ve doldurulduğu zaman ilk yerleştirildiği yerde bu text inputu bastırılır.

```
#hide_output
lbl_pred = widgets.Label()
lbl_pred.value = 'Please choose an image'
lbl_pred
```

Please choose an image

## General Button

FileUpload button'ı görmüştük, şimdi genel formda bir button nasıl yaratırız ona bakacağız:

```
#hide_output
btn_run = widgets.Button(description='Classify')
btn_run
```

Classify

Şuan yukarıdaki Classify butonu tamamen işlevsiz bir buton, tıkladığımızda hiçbir şey olmaz. Ancak istersek bu butona click event handler eklersek, butona bastığımızda istediğimiz fonksiyonların çalışmasını sağlayabiliriz:

```
def on_click_classify(change):  
    img = PILImage.create(btn_upload.data[-1])  
    out_pl.clear_output()  
    with out_pl: display(img.to_thumb(128,128))  
    pred,pred_idx,probs = learn_inf.predict(img)  
    lbl_pred.value = f'Prediction: {pred}; Probability: {probs[pred_idx]:.04f}'  
  
btn_run.on_click(on_click_classify)
```

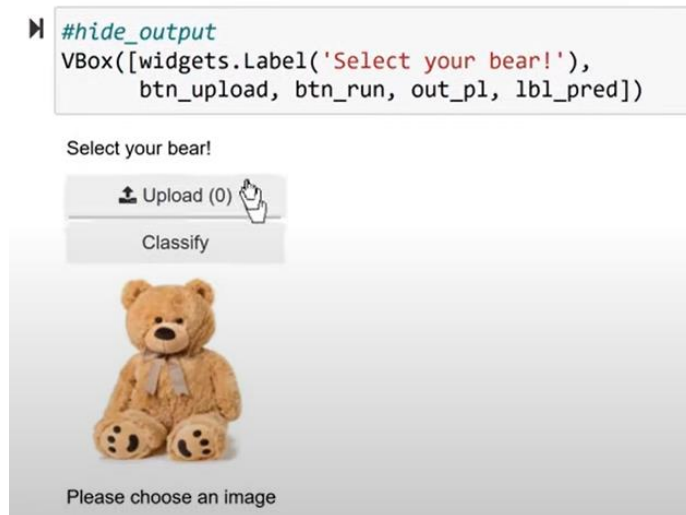
Yukarıda bir fonksiyon tanımlandı ve daha sonra bu fonksiyon Classify butonunun on\_click event'ine callback olarak atandı artık bu butona ne zaman tıklasak ilgili fonksiyon çalışacak.

Peki bu fonksiyonda ne yapılıyor? Öncelikle upload widget'ı kullanılarak data alınıyor, output widget'ının çıktısı temizleniyor, daha sonra içerisine alınan img veriliyor, ardından learn\_inf ile ilgili image modele beslenerek prediction yapılıyor ve son olarak Label widget'ının içi ilgili değerlerle dolduruluyor.

Yukarıdaki fonksiyon için output widget'ının ve label widget'ının dışarıda olduğunu dikkat et ancak bu fonksiyon çağırıldığında output ve label widget'larının içi bir anda doldurulacak.

## VBox

Şimdi, VBox kullanarak widgets'ı aynı yere koyabilirim, önceden label'ım, outputum veya upload butonum farklı farklı yerlerdeydi VBox ile hepsini birleştirebilirim:



Artık Voila kullanarak, bu notebook'u bir webapp'e çevirebilirim bu gerektiğinde 2. haftanın notebook'una bak!

## Not Almak

Bu kısımda son olarak not almanın öneminden bahsedelim, öğrenmek için çok iyi ve bir cv sağlamış oluyoruz. Blogging için fastai/fastpages'a göz at. Bu şekilde jupyter notebooks ile blogging yapabiliriz. Temelde notebooks'u formatted text'e dönüştürüyor anladığım kadarıyla.