# Lesson ‘02\_production.ipynb’

## Provide an example of where the bear classification model might work poorly, due to structural or style differences to the training data.

If there were no black-and-white images in the training data, the model may do poorly on black-and-white images. If the training data did not contain hand-drawn images then the model will probably do poorly on hand-drawn images.

## Where do text models currently have a major deficiency?

Text models don't currently have a reliable way to, for instance, combine a knowledge base of medical information, along with a deep learning model for generating medically correct natural language responses. This is very dangerous, because it is so easy to create content which appears to a layman to be compelling, but actually is entirely incorrect.

## What are possible negative societal implications of text generation models?

The highly compelling responses on social media, even if they are context-appropriate, can be used at massive scale, thousands of times greater than any troll farm previously seen, to spread disinformation, create unrest, and encourage conflict.

## In situations where a model might make mistakes, and those mistakes could be harmful, what is a good alternative to automating a process?

Deep learning should be part of a process in which the model and a human user interact closely. This can potentially make humans orders of magnitude more productive than they would be with entirely manual methods, and actually result in more accurate processes than using a human alone.

For instance, an automatic system can be used to identify potential stroke victims directly from CT scans, and send a high-priority alert to have those scans looked at quickly. There is only a three-hour window to treat strokes, so this fast feedback loop could save lives.

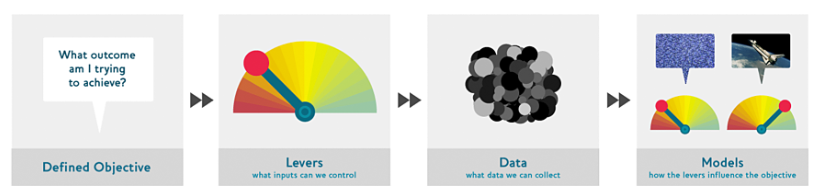
## What kind of tabular data is deep learning particularly good at?

Deep learning is good at data with variety of columns, for example columns containing natural language (e.g. book titles, reviews, etc.), and *high cardinality categorical* columns (i.e. something that contains a large number of discrete choices, such as zip code or product id).

## What's a key downside of directly using a deep learning model for recommendation systems?

Deep learning model only tells you what products a particular user might like, rather than what recommendations would be helpful for a user. Many kinds of recommendations for products a user might like may not be at all helpful, for instance, if the user is already familiar with the products, or if they are simply different packagings of products they have already purchased (such as a boxed set of novels, where they already have each of the items in that set).

## What are the steps of the Drivetrain approach?



## How do the steps of the Drivetrain approach map to a recommendation system?

The *objective* of a recommendation engine is to drive additional sales by surprising and delighting the customer with recommendations of items they would not have purchased without the recommendation. The *lever* is the ranking of the recommendations. New *data* must be collected to generate recommendations that will *cause new sales*. This will require conducting many randomized experiments in order to collect data about a wide range of recommendations for a wide range of customers. This is a step that few organizations take; but without it, you don't have the information you need to actually optimize recommendations based on your true objective (more sales!).

Finally, you could build two *models* for purchase probabilities, conditional on seeing or not seeing a recommendation. The difference between these two probabilities is a utility function for a given recommendation to a customer. It will be low in cases where the algorithm recommends a familiar book that the customer has already rejected (both components are small) or a book that they would have bought even without the recommendation (both components are large and cancel each other out).

## Create an image recognition model using data you curate, and deploy it on the web.

Please refer <https://github.com/JayaParmar/Deep-Learning/blob/master/practice_production.ipynb>

## What is DataLoaders?

DataLoaders is 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 are made available as properties. DataLoaders provide the data to the model.

## What four things do we need to tell fastai to create DataLoaders?

* 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

## What does the splitter parameter to DataBlock do?

The splitter parameter splits the training and validation sets randomly from the dataset.

## How do we ensure a random split always gives the same validation set?

We fix the random seed. Computers don't really know how to create random numbers at all, but simply create lists of numbers that look random. If you provide the same starting point for that list each time—called the *seed*—then you will get the exact same list each time

splitter=RandomSplitter(valid\_pct=0.2, seed=42)

## What letters are often used to signify the independent and dependent variables?

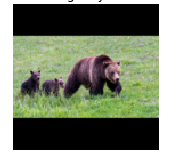
The independent variable is often referred to as x and the dependent variable is often referred to as y.

## What's the difference between crop, pad, and squish resize approaches? When might you choose one over the other?

Crop *crops* the images to fit a square shape of the size requested, using the full width or height. Full height in this image.



Crop can result in losing some important details. Pad the images with zeros (black) to retain the details. See below the bear had two babies. :)



Squish will squish or stretch the image retaining the details but will look a bit unrealistic.



In practice, we randomly select part of the image, and crop to just that part. On each epoch (which is one complete pass through all of our images in the dataset) we randomly select a different part of each image. This means that our model can learn to focus on, and recognize, different features in our images. It also reflects how images work in the real world: different photos of the same thing may be framed in slightly different ways.

## What is data augmentation? Why is it needed?

Dataaugmentation refers to creating random variations of our input data, such that they appear different, but do not actually change the meaning of the data.

Common data augmentation techniques for images are rotation, flipping, perspective warping, brightness changes and contrast changes. It is needed to create lots of data when we have limited number of images in the dataset.

## What is the difference between item\_tfms and batch\_tfms?

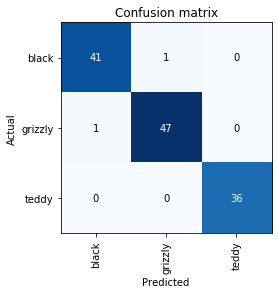
Images (data) input to the model need to be of the same size. We need to add a transform which will resize these images to the same size.

Item transforms are pieces of code that run on each *individual item*, whether it be an image, category. Batch transforms run on a *batch*. It saves time.

## What is a confusion matrix?

It is a matrix to see the mistakes our model made. For example if the model is given a task of classification, the rows of the matrix represent the correct classes in the dataset. The columns of the matrix represent the classes *predicted* by the model. Thus the diagonal would mean correct classes predicted and off diagonal cells would show the mistakes.

In the diagram below, each row represents all the black, grizzly, and teddy bears in our *dataset*, respectively. Each column represents the images which the model *predicted* as black, grizzly, and teddy bears, respectively.



## What does export save?

When the function export is called, fastai will save a pickle file called ‘export.pkl’.

## What is it called when we use a model for getting predictions, instead of training?

Inference

## What are IPython widgets?

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.

## When might you want to use a CPU for deployment? When might GPU be better?

GPUs are only useful when they do lots of identical work in parallel. If you're doing (say) image classification, then you'll normally be classifying just *one user's image* at a time, and there isn't enough work to do in a single image to keep a GPU busy.. So, a CPU is more cost-effective.

GPU is better if your app gets popular enough that it makes clear financial sense for you to do so.

Note: you still train your model on GPU. Above is when the model is in production.

## What are the downsides of deploying your app to a server, instead of to a client (or edge) device such as a phone or PC?

The application will require a network connection, and there will be some latency each time the model is called.

If the app uses sensitive data then users may be concerned about an approach which sends that data to a remote server, so privacy considerations will mean that you need to run the model on the edge device

Managing the complexity and scaling the server can create additional overhead, whereas if your model runs on the edge devices then each user is bringing their own compute resources, which leads to easier scaling with an increasing number of users.

## What are 3 examples of problems that could occur when rolling out a bear warning system in practice?

* Working with video data instead of images
* Handling nighttime images, which may not appear in this dataset
* Dealing with low-resolution camera images
* Ensuring results are returned fast enough to be useful in practice
* Recognizing bears in positions that are rarely seen in photos that people post online (for example from behind, partially covered by bushes, or when a long way away from the camera)

## What is "out of domain data"?

Out of domain data is the data that the model sees in production which is very different to what it saw during training.

## What is "domain shift"?

Domain shift is when the type of data that the model sees changes over time.

For instance, an insurance company may use a deep learning model as part of its pricing and risk algorithm, but over time the types of customers that the company attracts, and the types of risks they represent, may change so much that the original training data is no longer relevant.

## What are the 3 steps in the deployment process?

* Managing multiple versions of deep learning models
* A/B testing
* Refreshing the data (remove the old data as the datasets grow all the time)