

```
#hide  
! [ -e /content ] && pip install -Uqq fastbook  
import fastbook fastbook.setup_book()  
  
In [ ]:  
  
#hide  
from fastbook import *  
from fastai.vision.widgets import *
```

# From Model to Production

モデルから生産まで

The six lines of code we saw in <> are just one small part of the process of using deep learning in practice. In this chapter, we're going to use a computer vision example to look at the end-to-end process of creating a deep learning application. More specifically, we're going to build a bear classifier! In the process, we'll discuss the capabilities and constraints of deep learning, explore how to create datasets, look at possible gotchas when using deep learning in practice, and more. Many of the key points will apply equally well to other deep learning problems, such as those in <>. If you work through a problem similar in key respects to our example problems, we expect you to get excellent results with little code, quickly.

<>で見た 6 行のコードは、ディープラーニングを実際に使うプロセスのほんの一部に過ぎない。この章では、コンピュータビジョンの例を使って、ディープラーニングのアプリケーションを作成するエンドツーエンドのプロセスを見ていきます。具体的には、熊の分類器を作ることになります！その過程で、深層学習の能力と制約、データセットの作成方法、深層学習を実際に使用する際に起こりうる不具合などを議論します。重要なポイントの多くは、<>にあるような他の深層学習の問題にも同様に適用されるでしょう。この例題と同じような問題を解くことで、少ないコードで素早く優れた結果を得られると期待しています。

Let's start with how you should frame your problem.

まずは、問題をどのように組み立てるべきかを考えてみましょう。

# The Practice of Deep Learning

ディープラーニングの実践

We've seen that deep learning can solve a lot of challenging problems quickly and with little code. As a beginner, there's a sweet spot of problems that are similar enough to our example problems that you can very quickly get extremely useful results. However, deep learning isn't magic! The same 6 lines of code won't work for every problem anyone can think of today. Underestimating the constraints and overestimating the capabilities of deep learning may lead to frustratingly poor results, at least until you gain some experience and can solve the problems that arise.

Conversely, overestimating the constraints and underestimating the capabilities of deep learning may mean you do not attempt a solvable problem because you talk yourself out of it.

ディープラーニングは、多くの困難な問題を少ないコードで素早く解決できることを見ました。初心者のうちは、今回の例題と十分に似たような問題で、非常に有用な結果をすぐに得ることができるスイートスポットが存在します。しかし、ディープラーニングは魔法ではありません！同じ 6 行のコードで、今日誰もが思いつくすべての問題に対応できるわけではありません。制約を過小評価し、ディープラーニングの能力を過大評価すると、少なくとも経験を積み、発生した問題を解決できるようになるまでは、イライラするほど悪い結果を招くかもしれません。逆に、制約を過大評価し、深層学習の能力を過小評価すると、解決可能な問題でも、自分自身に言い聞かせて、挑戦しないことになります。

We often talk to people who underestimate both the constraints and the capabilities of deep learning. Both of these can be problems: underestimating the capabilities means that you might not even try things that could be very beneficial, and underestimating the constraints might mean that you fail to consider and react to important issues. 私たちはよく、深層学習の制約と能力の両方を過小評価している人たちと話をします。能力を過小評価すると、非常に有益になる可能性のあることを試みないことになり、制約を過小評価すると、重要な問題を検討し対応することができなくなる可能性があります。

The best thing to do is to keep an open mind. If you remain open to the possibility that deep learning might solve part of your problem with less data or complexity than you expect, then it is possible to design a process where you can find the specific capabilities and constraints related to your particular problem as you work through the process. This doesn't mean making any risky bets — we will show you how you can gradually roll out models so that they don't create significant risks, and can even backtest them prior to putting them in production.

一番良いのは、オープンマインドを保つことです。もしあなたが、深層学習があなたの予想よりも少ないデータや複雑さで問題の一部を解決するかもしれないという可能性に心を開いていれば、プロセスを進めるうちに、あなたの特定の問題に関連する特定の能力と制約を見つけることができるプロセスを設計することができます。これは、リスクの高い賭けをすることを意味するものではありません。私たちは、大きなリスクを生じさせないようにモデルを徐々に展開し、本番に投入する前にバックテストを行う方法についても紹介します。

## Starting Your Project

### プロジェクトの開始

So where should you start your deep learning journey? The most important thing is to ensure that you have some project to work on—it is only through working on your own projects that you will get real experience building and using models. When selecting a project, the most important consideration is data availability. Regardless of whether you are doing a project just for your own learning or for practical application in your organization, you want something where you can get started quickly. We have seen many students, researchers, and industry practitioners waste months or years while they attempt to find their perfect dataset. The goal is not to find the "perfect" dataset or project, but just to get started and iterate from there.

では、ディープラーニングの旅はこれから始めればいいのでしょうか。最も重要なのは、取り組むべきプロジェクトがあることを確認することです。自分のプロジェクトに取り組むことでしか、モデルの構築と使用に関する実際の経験を得ることはできません。プロジェクトを選択する際に最も重要なのは、データの利用可能性です。自分自身の学習のためのプロジェクトであれ、組織での実用的な応用のためのプロジェクトであれ、すぐに始められるものがよいでしょう。私たちは、多くの学生、研究者、業界関係者が、完璧なデータセットを見つけようとするあまり、数ヶ月から数年を無駄にするのを見てきました。目標は「完璧な」データセットやプロジェクトを見つけることではなく、とにかく始めて、そこから反復することです。

If you take this approach, then you will be on your third iteration of learning and improving while the perfectionists are still in the planning stages!

このアプローチをとれば、完璧主義者がまだ計画段階である間に、あなたは 3 回目の学習と改善の繰り返しをすることになります！

We also suggest that you iterate from end to end in your project; that is, don't spend months fine-tuning your model, or polishing the perfect GUI, or labelling the perfect dataset... Instead, complete every step as well as you can in a reasonable amount of time, all the way to the end. For instance, if your final goal is an application that runs on a mobile phone, then that should be what you have after each iteration. But perhaps in the early iterations you take some shortcuts, for instance by doing all of the processing on a remote server, and using a simple responsive web application. By completing the project end to end, you will see where the trickiest bits are, and which bits make the biggest difference to the final result.

つまり、何ヶ月もかけてモデルの微調整をしたり、完璧な GUI を作ったり、完璧なデータセットにラベルを貼ったりするのではなく、合理的な時間でできる限りすべてのステップを完了させ、最後までやり遂げることをお勧めします。例えば、最終的なゴールが携帯電話上で動作するアプリケーションだとしたら、各反復の後にはそれが完成しているはずです。しかし、初期の段階では、例えば、すべての処理をリモートサーバーで行い、シンプルなレスポンシブ Web アプリケーションを使用するなど、いくつかのショートカットを行っているかもしれません。プロジェクトを最後までやり遂げることで、どこが一番難しいのか、どの部分が最終的な結果を大きく左右するのかが見えてくるはずです。

As you work through this book, we suggest that you complete lots of small experiments, by running and adjusting the notebooks we provide, at the same time that you gradually develop your own projects. That way, you will be getting experience with all of the tools and techniques that we're explaining, as we discuss them.

この本を読みながら、私たちが提供するノートブックを実行したり調整したりして、小さな実験をたくさんこなすと同時に、自分のプロジェクトを徐々に発展させていくことをお勧めします。そうすれば、私たちが説明するすべてのツールやテクニックを、私たちが議論しながら経験することができます。

s: To make the most of this book, take the time to experiment between each chapter, be it on your own project or by exploring the notebooks we provide. Then try rewriting those notebooks from scratch on a new dataset. It's only by practicing (and failing) a lot that you will get an intuition of how to train a model.

s: 本書を最大限に活用するために、各章の間に、自分のプロジェクトで、あるいは私たちが提供するノートブックを使って、実験する時間をとってください。そして、そのノートブックを新しいデータセットで一から書き換えてみてください。たくさん練習して（そして失敗して）こそ、モデルの訓練方法の直感を得ることができます。

By using the end-to-end iteration approach you will also get a better understanding of how much data you really need. For instance, you may find you can only easily get 200 labeled data items, and you can't really know until you try whether that's enough to get the performance you need for your application to work well in practice.

また、End-to-End の反復アプローチを用いることで、本当に必要なデータ量をより深く理解することができます。例えば、200 個のラベル付きデータしか簡単に取得できないことが分かっても、それが実際にアプリケーションをうまく動作させるために必要な性能を得るのに十分かどうかは、実際にやってみないと分からないのです。

In an organizational context you will be able to show your colleagues that your idea can really work by showing them a real working prototype. We have repeatedly observed that this is the secret to getting good organizational buy-in for a project.

組織の中では、実際に動くプロトタイプを見せることで、あなたのアイデアが本当に機能することを同僚に示すことができます。私たちは、これがプロジェクトの組織的な賛同を得るために秘訣であることを繰り返し確認してきました。

Since it is easiest to get started on a project where you already have data available, that means it's probably easiest to get started on a project related to something you are already doing, because you already have data about things that you are doing. For instance, if you work in the music business, you may have access to many recordings. If you work as a radiologist, you probably have access to lots of medical images. If you are interested in wildlife preservation, you may have access to lots of images of wildlife.

すでに利用可能なデータを持っているプロジェクトに着手するのが最も簡単なので、つまり、自分がすでにやっていることに関連するプロジェクトに着手するのが最も簡単でしょうということです。例えば、音楽関係の仕事をしている人なら、多くの録音物にアクセスできるかもしれません。放射線科医として働いている人は、たくさんの医療用画像にアクセスできるのではないかでしょうか。野生動物の保護に興味があれば、野生動物の画像にたくさんアクセスできるかもしれません。

Sometimes, you have to get a bit creative. Maybe you can find some previous machine learning project, such as a Kaggle competition, that is related to your field of interest. Sometimes, you have to compromise. Maybe you can't find the exact data you need for the precise project you have in mind; but you might be able to find something from a similar domain, or measured in a different way, tackling a slightly different problem. Working on these kinds of similar projects will still give you a good understanding of the overall process, and may help you identify other shortcuts, data sources, and so forth.

時には、ちょっと工夫が必要なこともあります。Kaggle のコンペティションなど、自分の興味ある分野に関連する過去の機械学習プロジェクトが見つかるかもしれません。時には、妥協しなければならないこともあります。しかし、似たような分野のデータが見つかるかもしれませんし、別の方法で測定して、少し違った問題に取り組めるかもしれません。このような類似のプロジェクトに取り組むことで、全体的なプロセスを理解することができ、他の近道やデータソースなどを見出すことができるようになるかもしれません。

Especially when you are just starting out with deep learning, it's not a good idea to branch out into very different areas, to places that deep learning has not been applied to before. That's because if your model does not work at first, you will not know whether it is because you have made a mistake, or if the very problem you are trying to solve is simply not solvable with deep learning. And you won't know where to look to get help. Therefore, it is best at first to start with something where you can find an example online where somebody has had good results with something that is at least somewhat similar to what you are trying to achieve, or where you can convert your data into a format similar to what someone else has used before (such as creating an image from your data). Let's have a look at the state of deep learning, just so you know what kinds of things deep learning is good at right now.

特にディープラーニングを始めたばかりの頃は、これまでディープラーニングが適用されていないような、全く異なる分野に手を出すのは得策ではありません。というのも、最初にモデルがうまくいかなかった場合、それが自分のミスなのか、それとも自分が解決しようとしている問題が単にディープラーニングでは解決できないのかがわからなくなるからです。そして、どこに助けを求めればいいのかもわからない。ですから、最初は、少なくともあなたが達成しようとしていることに多少なりとも似ているもので、誰かが良い結果を出した例をネットで見つけることができるもの、あるいは、あなたのデータを、誰かが以前に使ったものと同じような形式に変換できるもの（例えば、あなたのデータから画像を作成する）から始めるのがベストです。ディープラーニングが今どんなことを得意としているのかを知るために、ディープラーニングの現状を見てみましょう。

## The State of Deep Learning

### ディープラーニングの現状

Let's start by considering whether deep learning can be any good at the problem you are looking to work on. This section provides a summary of the state of deep learning at the start of 2020. However, things move very fast, and by the time you read this some of these constraints may no longer exist. We will try to keep the [book's website](#) up-to-date; in addition, a Google search for "what can AI do now" is likely to provide current information.

まずは、自分が取り組もうとしている問題に対して、ディープラーニングが少しでも役に立てるかどうかを考えることから始めましょう。このセクションでは、2020年初頭の深層学習の状況をまとめています。しかし、物事は非常に速く動くので、あなたがこれを読む

頃には、これらの制約のいくつかはもはや存在しないかもしれません。本書のウェブサイトでは最新の情報を提供するように努めます。また、Googleで「AIは今何ができるのか」と検索すると、最新の情報を得られる可能性があります。

## Computer vision

### コンピュータビジョン

There are many domains in which deep learning has not been used to analyze images yet, but those where it has been tried have nearly universally shown that computers can recognize what items are in an image at least as well as people can—even specially trained people, such as radiologists. This is known as *object recognition*. Deep learning is also good at recognizing where objects in an image are, and can highlight their locations and name each found object. This is known as *object detection* (there is also a variant of this that we saw in <>, where every pixel is categorized based on what kind of object it is part of—this is called *segmentation*). Deep learning algorithms are generally not good at recognizing images that are significantly different in structure or style to those used to train the model. For instance, if there were no black-and-white images in the training data, the model may do poorly on black-and-white images. Similarly, if the training data did not contain hand-drawn images, then the model will probably do poorly on hand-drawn images. There is no general way to check what types of images are missing in your training set, but we will show in this chapter some ways to try to recognize when unexpected image types arise in the data when the model is being used in production (this is known as checking for *out-of-domain* data).

ディープラーニングが画像解析に使われていない領域は数多くありますが、ディープラーニングが試された領域では、コンピュータが画像に写っているものを、少なくとも人間（放射線科医のような特別な訓練を受けた人）と同じように認識できることがほぼ共通して示されています。これは物体認識と呼ばれるものです。ディープラーニングは、画像内の物体がどこにあるかを認識することにも優れており、その位置を強調表示したり、見つかった物体に名前をつけたりすることができます。これはオブジェクト検出と呼ばれています（<>で見たように、すべてのピクセルがどのようなオブジェクトの一部であるかに基づいて分類される、このようなバリエーションもあります—これはセグメンテーションと呼ばれています）。ディープラーニングのアルゴリズムは、一般的に、モデルの学習に使用した画像と構造やスタイルが大きく異なる画像を認識することは苦手です。例えば、学習データに白黒の画像がない場合、モデルは白黒の画像を苦手とする可能性があります。同様に、トレーニングデータに手描きの画像がなかった場合、モデルは手描きの画像に弱いかもしれません。トレーニングセットに含まれていない画像の種類をチェックする一般的な方法はありませんが、本章では、モデルが実運用されているときに、予期しない画像の種類がデータから発生した場合にそれを認識しようとする方法をいくつか紹介します（これは、領域外データのチェックと呼ばれています）。

One major challenge for object detection systems is that image labelling can be slow and expensive. There is a lot of work at the moment going into tools to try to make this labelling faster and easier, and to require fewer handcrafted labels to train accurate object detection models. One approach that is particularly helpful is to synthetically generate variations of input images, such as by rotating them or changing their brightness and contrast; this is called *data augmentation* and also works well for text and other types of models. We will be discussing it in detail in this chapter.

物体検出システムの大きな課題の一つは、画像のラベリングに時間とコストがかかることがある。現在、このラベリングをより迅速かつ容易にし、正確な物体検出モデルを学習するためには必要な手作業によるラベルを少なくするためのツールに多くの取り組みが行われています。特に便利なのは、入力画像を回転させたり、明るさやコントラストを変えたりして、合成的にバリエーションを生成する方法です。これはデータ増強と呼ばれ、テキストや他のタイプのモデルでも有効です。この章では、この方法について詳しく説明する予定です。

Another point to consider is that although your problem might not look like a computer vision problem, it might be possible with a little imagination to turn it into one. For instance, if what you are trying to classify are sounds, you might try converting the sounds into images of their acoustic waveforms and then training a model on those images.

もう一つ考慮すべき点は、あなたの問題はコンピュータビジョンの問題には見えないかもしれません、少しの想像力でそれを問題に変えることができるかもしれない、ということです。例えば、分類しようとしているのが音であれば、音を音響波形の画像に変換し、その画像でモデルを学習させることができるかもしれません。

## Text (natural language processing)

### テキスト（自然言語処理）

Computers are very good at classifying both short and long documents based on categories such as spam or not spam, sentiment (e.g., is the review positive or negative), author, source website, and so forth. We are not aware of any rigorous work done in this area to compare them to humans, but anecdotally it seems to us that deep learning performance is similar to human performance on these tasks. Deep learning is also very good at generating context-appropriate text, such as replies to social media posts, and imitating a particular author's style. It's good at making this content compelling to humans too—in fact, even more compelling than human-generated text. However, deep learning is currently not good at generating *correct* responses! We don't currently have a reliable way to, for instance, combine a knowledge base of medical information with a deep learning model for generating medically correct natural language responses. This is very dangerous, because it is so easy to create content that appears to a layman to be compelling, but actually is entirely incorrect.

コンピュータは、スパムかスパムでないか、メント（レビューがポジティブかネガティブかなど）、著者、ソースウェブサイトなどのカテゴリに基づいて、短い文書と長い文書の両方を分類することに非常に優れています。私たちは、この分野で人間と比較するための厳密な研究が行われたことを知りませんが、逸話的には、ディープラーニングの性能はこれらのタスクで人間の性能と同様であると思われます。また、ディープラーニングは、ソーシャルメディアの投稿に対する返信など、文脈に応じたテキストを生成したり、特定の著者のスタイルを模倣することも得意としています。このようなコンテンツを人間にとて魅力的なものにすることも得意で、実際、人間が作成したテキストよりも説得力があります。しかし、ディープラーニングは今のところ、正しい回答を生成することは得意ではありません！例えば、医療情報の知識ベースとディープラーニングモデルを組み合わせて、医学的に正しい自然言語応答を生成する信頼できる方法は、今のところありません。素人には説得力があるように見えても、実はまったく正しくないコンテンツを作ることが簡単にできてしまうので、これは非常に危険なことです。

Another concern is that context-appropriate, highly compelling responses on social media could be used at massive scale—thousands of times greater than any troll farm previously seen—to spread disinformation, create unrest, and encourage conflict. As a rule of thumb, text generation models will always be technologically a bit ahead of models recognizing automatically generated text. For instance, it is possible to use a model that can recognize artificially generated content to actually improve the generator that creates that content, until the classification model is no longer able to complete its task.

もう一つの懸念は、ソーシャルメディア上で文脈に適した、非常に説得力のある応答が、偽情報の拡散、不安の創出、紛争の助長に、これまでのトロールファームの何千倍もの大規模な規模で利用される可能性があるということです。経験則から言うと、テキスト生成モデルは、自動生成されたテキストを認識するモデルよりも技術的に常に少し先を行っている。例えば、人工的に生成されたコンテンツを認識するモデルを用いて、そのコンテンツを生成するジェネレーターを実際に改善し、分類モデルがそのタスクを完了できなくなるまで改善することが可能です。

Despite these issues, deep learning has many applications in NLP: it can be used to translate text from one language to another, summarize long documents into something that can be digested more quickly, find all mentions of a concept of interest, and more. Unfortunately, the translation or summary could well include completely incorrect information! However, the performance is already good enough that many people are using these systems—for instance, Google's online translation system (and every other online service we are aware of) is based on deep learning.

ディープラーニングは、ある言語から別の言語への翻訳、長い文書をより速く消化できるように要約すること、関心のある概念に関するすべての言及を見つけることなど、多くの用途に使用することができます。残念ながら、翻訳や要約には完全に間違った情報が含まれる可能性があります！しかし、その性能はすでに十分で、多くの人がこれらのシステムを利用しています。例えば、Google のオンライン翻訳システム（そして私たちが知っている他のすべてのオンラインサービス）は、ディープラーニングに基づいています。

## Combining text and images

### テキストと画像の組み合わせ

The ability of deep learning to combine text and images into a single model is, generally, far better than most people intuitively expect. For example, a deep learning model can be trained on input images with output captions written in English, and can learn to generate surprisingly appropriate captions automatically for new images! But again, we have the same warning that we discussed in the previous section: there is no guarantee that these captions will actually be correct.

テキストと画像を 1 つのモデルに統合するディープラーニングの能力は、一般的に、多くの人が直感的に予想するよりもはるかに優れています。例えば、英語で書かれたキャプションが出力された入力画像に対してディープラーニングモデルを学習させれば、新しい画像に対して驚くほど適切なキャプションを自動生成することができます！しかし、ここでも前項と同じように、「このキャプションが実際に正しいという保証はない」という警告が出ます。

Because of this serious issue, we generally recommend that deep learning be used not as an entirely automated process, but as 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. At the same time, however, all scans could continue to be sent to radiologists in the usual way, so there would be no reduction in human input. Other deep learning models could automatically measure items seen on the scans, and insert those measurements into reports, warning the radiologists about findings that they may have missed, and telling them about other cases that might be relevant.

この重大な問題のために、私たちは一般的に、ディープラーニングを完全に自動化されたプロセスとしてではなく、モデルと人間のユーザーが密接にやり取りするプロセスの一部として使用することを推奨します。そうすることで、人が完全に手作業で行うよりも何桁も生産性が向上し、実際に人間だけで行うよりも正確な処理ができる可能性があります。例えば、CT スキャンから脳卒中の可能性がある人を特定し、優先順位の高いアラートを送信して、そのスキャンを迅速に見てもらうといったことが、自動システムで可能になります。脳卒中の治療には 3 時間しかないため、このような迅速なフィードバックループがあれば、命を救うことができます。しかし、同時に、すべてのスキャン画像は通常の方法で放射線科医に送られ続けるので、人の入力が減ることはない。また、ディープラーニングモデルを使えば、撮影した画像を自動的に測定し、その測定値をレポートに挿入して、放射線科医が見落とした所見を警告したり、関連性のある他の症例を伝えたりすることができます。

## Tabular data

### 表形式データ

For analyzing time series and tabular data, deep learning has recently been making great strides. However, deep learning is generally used as part of an ensemble of multiple types of model. If you already have a system that is using random forests or gradient boosting machines (popular tabular modeling tools that you will learn about soon), then switching to or adding deep learning may not result in any dramatic improvement. Deep learning does greatly increase the variety of columns that you can include—for example, columns containing natural language (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). On the down side, deep learning models generally take longer to train than random forests or gradient boosting machines, although this is changing thanks to libraries such as [RAPIDS](#), which provides GPU acceleration for the whole modeling pipeline. We cover the pros and cons of all these methods in detail in <>chapter\_tabular>>.

時系列や表形式のデータ解析のために、最近、深層学習が大きな成果を上げています。しかし、ディープラーニングは一般的に、複数のタイプのモデルのアンサンブルの一部として使用されます。ランダムフォレストやグラディエントブースティングマシン（近日中に紹介する人気の表形式モデリングツール）を使用しているシステムがすでにある場合、ディープラーニングに変更または追加しても、劇的な改善にはつながらないかもしれません。例えば、自然言語を含む列（本のタイトル、レビューなど）や、高基本度カテゴリ列（郵便番号や製品 ID など、多数の離散的な選択肢を含むもの）などがそうである。一方、ディ

一پラーニングモデルは、一般的にランダムフォレストやグラディエントブースティングマシンに比べて学習に時間がかかりますが、モデリングパイプライン全体の GPU アクセラレーションを提供する RAPIDS などのライブラリのおかげで、この状況は変わりつつあります。これらの手法の長所と短所については、<>で詳しく解説しています。

## Recommendation systems

### レコメンデーションシステム

Recommendation systems are really just a special type of tabular data. In particular, they generally have a high-cardinality categorical variable representing users, and another one representing products (or something similar). A company like Amazon represents every purchase that has ever been made by its customers as a giant sparse matrix, with customers as the rows and products as the columns. Once they have the data in this format, data scientists apply some form of collaborative filtering to *fill in the matrix*. For example, if customer A buys products 1 and 10, and customer B buys products 1, 2, 4, and 10, the engine will recommend that A buy 2 and 4. Because deep learning models are good at handling high-cardinality categorical variables, they are quite good at handling recommendation systems. They particularly come into their own, just like for tabular data, when combining these variables with other kinds of data, such as natural language or images. They can also do a good job of combining all of these types of information with additional metadata represented as tables, such as user information, previous transactions, and so forth.

レコメンデーションシステムは、まさに特殊な表形式のデータである。特に、レコメンデーションシステムは、一般に、ユーザーを表す高基準のカテゴリー変数と、商品（またはそれに類するもの）を表す別の変数を持っています。Amazon のような企業は、顧客がこれまでに購入したすべての商品を、顧客を行、商品を列とする巨大なスペース行列として表現しています。このような形式のデータを手に入れたデータサイエンティストは、何らかの形で協調フィルタリングを適用して行列を埋めていきます。例えば、顧客 A が商品 1 と 10 を購入し、顧客 B が商品 1、2、4、10 を購入した場合、エンジンは A が 2 と 4 を購入するよう推奨します。ディープラーニングモデルは、高基準のカテゴリー変数を扱うのが得意なので、レコメンデーションシステムを扱うのにかなり向いています。特に、表データと同じように、自然言語や画像など他の種類のデータと組み合わせることで本領を発揮します。また、ユーザー情報や過去の取引履歴など、表として表現されるメタデータと組み合わせることも可能です。

However, nearly all machine learning approaches have the downside that they only tell 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, when they already have each of the items in that set). Jeremy likes reading books by Terry Pratchett, and for a while Amazon was recommending nothing but Terry Pratchett books to him (see <>), which really wasn't helpful because he already was aware of these books!

しかし、ほぼすべての機械学習アプローチには、特定のユーザーが好む可能性のある商品しかわからないという欠点があり、むしろ、どのような推奨がユーザーにとって役に立つかを知ることができません。例えば、その商品をすでに知っていたり、すでに購入した商品のパッケージが違うだけだったりする場合（例えば、小説のボックスセットなど、その

セット内の各商品をすでに持っている場合)、ユーザーが好きそうな商品の推奨は、まったく役に立たないことがあります。ジェレミーはテリー・プラチエットの本を読むのが好きで、しばらく Amazon はテリー・プラチエットの本ばかりを薦めていましたが(<>参照)、彼はすでにこれらの本を知っていたので、本当に役に立たなかったようです!

#### Customers Who Bought This Item Also Bought



テリー・プラチエット関連書籍のおすすめ

## Other data types

その他のデータ型

Often you will find that domain-specific data types fit very nicely into existing categories. For instance, protein chains look a lot like natural language documents, in that they are long sequences of discrete tokens with complex relationships and meaning throughout the sequence. And indeed, it does turn out that using NLP deep learning methods is the current state-of-the-art approach for many types of protein analysis. As another example, sounds can be represented as spectrograms, which can be treated as images; standard deep learning approaches for images turn out to work really well on spectrograms.

多くの場合、ドメイン固有のデータ型は既存のカテゴリーにうまく収まることがわかります。例えば、タンパク質鎖は自然言語文書によく似ています。それは、離散的なトークンの長いシーケンスであり、シーケンス全体に複雑な関係や意味があるという点です。そして実際に、多くの種類のタンパク質分析において、NLP のディープラーニング手法を用いることが、現在の最先端アプローチであることが判明しています。別の例として、音はスペクトrogramとして表現され、画像として扱うことができます。画像に対する標準的な深層学習アプローチは、スペクトrogramに対して非常にうまく機能することがわかりました。

## The Drivetrain Approach

ドライブトレインの考え方

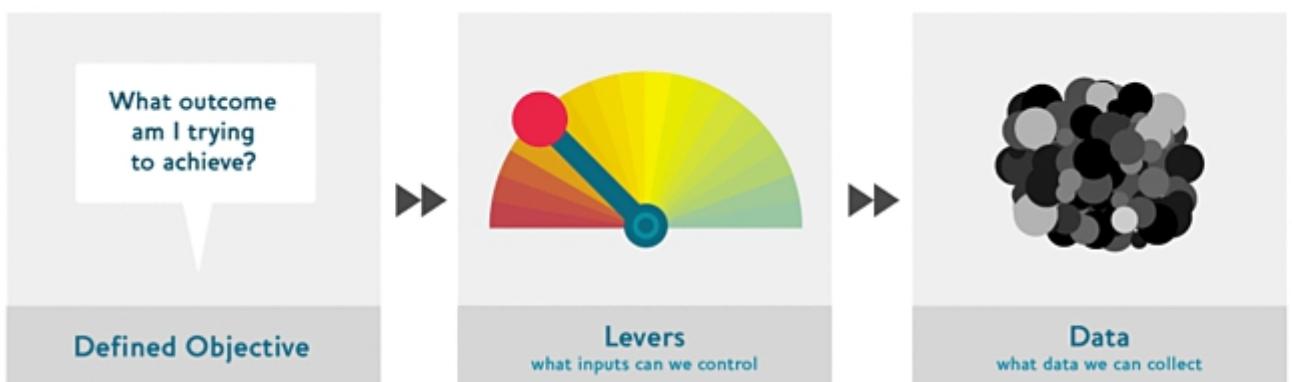
There are many accurate models that are of no use to anyone, and many inaccurate models that are highly useful. To ensure that your modeling work is useful in practice, you need to consider how your work will be used. In 2012 Jeremy, along with Margit Zwemer and Mike Loukides, introduced a method called *the Drivetrain Approach* for thinking about this issue.

誰にも役に立たない正確なモデルもあれば、有用性の高い不正確なモデルも多く存在します。あなたのモデリング作業が実際に役立つようにするには、あなたの作業がどのように使われるかを考える必要があります。2012年、ジェレミーはマーギット・ズウェマーとマイ

ク・ルーキデスとともに、この問題を考えるための「ドライブトレイン・アプローチ」という手法を紹介しました。

The Drivetrain Approach, illustrated in <>, was described in detail in "Designing Great Data Products". The basic idea is to start with considering your objective, then think about what actions you can take to meet that objective and what data you have (or can acquire) that can help, and then build a model that you can use to determine the best actions to take to get the best results in terms of your objective.

<>で例示したドライブトレイン・アプローチは、「優れたデータ製品の設計」で詳しく説明されています。基本的な考え方は、まず目的を考えることから始め、その目的を達成するためにどのような行動が可能か、そのために役立つデータは何か（あるいは取得可能か）を考え、目的の観点から最良の結果を得るために取るべき行動を決めるために使えるモデルを構築する、というものです。



Consider a model in an autonomous vehicle: you want to help a car drive safely from point A to point B without human intervention. Great predictive modeling is an important part of the solution, but it doesn't stand on its own; as products become more sophisticated, it disappears into the plumbing. Someone using a self-driving car is completely unaware of the hundreds (if not thousands) of models and the petabytes of data that make it work. But as data scientists build increasingly sophisticated products, they need a systematic design approach.

自律走行車におけるモデルを考えてみましょう。人間の介入なしに、車が A 地点から B 地点まで安全に走行できるようにしたいですよね。優れた予測モデリングはソリューションの重要な一部ですが、それだけでは成り立ちません。製品が高度化するにつれて、配管の中に消えていってしまうのです。自動運転車を使用している人は、それを機能させている何百（何千とは言わないまでも）ものモデルやペタバイトのデータにはまったく気づかないのです。しかし、データサイエンティストがますます洗練された製品を作るようになると、体系的なデザインアプローチが必要になります。

We use data not just to generate more data (in the form of predictions), but to produce *actionable outcomes*. That is the goal of the Drivetrain Approach. Start by defining a clear *objective*. For instance, Google, when creating their first search engine, considered "What is the user's main objective in typing in a search query?" This led them to their objective, which was to "show the most relevant search result." The next step is to consider what *levers* you can pull (i.e., what actions you can take) to better achieve that objective. In Google's case, that was the ranking of

the search results. The third step was to consider what new *data* they would need to produce such a ranking; they realized that the implicit information regarding which pages linked to which other pages could be used for this purpose. Only after these first three steps do we begin thinking about building the predictive *models*. Our objective and available levers, what data we already have and what additional data we will need to collect, determine the models we can build. The models will take both the levers and any uncontrollable variables as their inputs; the outputs from the models can be combined to predict the final state for our objective.

私たちはデータを、（予測という形で）より多くのデータを生成するためだけでなく、実用的な結果を生み出すために使用します。それが、ドライブトレイン・アプローチの目標です。まず、明確な目的を設定することから始めます。例えば、Googleは最初の検索エンジンを作る際に、"検索クエリを入力するユーザーの主な目的は何か？"を考えました。その結果、"最も関連性の高い検索結果を表示する"という目的を導き出しました。次に、その目的をよりよく達成するために、どんなレバーを引くことができるか（＝どんなアクションを起こすことができるか）を考えることになる。Googleの場合、それは検索結果の順位付けだった。どのページがどのページにリンクしているかという暗黙の情報が、この目的に使えると考えたのです。この最初の3つのステップを経て、初めて予測モデルの構築について考え始めるのです。私たちの目的と利用可能なレバー、すでに持っているデータと追加で収集する必要のあるデータによって、構築できるモデルが決まります。モデルは、レバーと制御不能な変数の両方を入力とし、モデルからの出力を組み合わせて、目的の最終状態を予測することができます。

Let's consider another example: recommendation systems. 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!).

別の例として、レコメンデーションシステムを考えてみましょう。レコメンデーション・エンジンの目的は、レコメンデーションがなければ購入しなかったであろう商品を推奨し、顧客に驚きと喜びを与えることによって、さらなる売上を促進することです。レバーは、レコメンデーションのランキングです。新たな売上を引き起こすレコメンドを生成するためには、新たなデータを収集する必要がある。そのためには、幅広い顧客に対する幅広いレコメンデーションのデータを収集するために、多くの無作為化実験を行う必要がある。このステップを踏む組織はほとんどありませんが、これがなければ、真の目的（より多くの売上！）に基づいてレコメンデーションを実際に最適化するために必要な情報を得ることはできません。

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).

最後に、レコメンドを見た場合と見なかった場合の購入確率について、2つのモデルを構築することができます。この2つの確率の差は、ある顧客に対するレコメンデーションに対する効用関数となります。アルゴリズムが推薦する本が、顧客がすでに拒否した身近な本であったり（両成分は小さい）、推薦がなくても買ったであろう本であったり（両成分は大きく、互いに相殺される）する場合、この値は低くなる。

As you can see, in practice often the practical implementation of your models will require a lot more than just training a model! You'll often need to run experiments to collect more data, and consider how to incorporate your models into the overall system you're developing. Speaking of data, let's now focus on how to find data for your project.

このように、実際には、モデルの実用的な実装には、モデルを訓練するだけでなく、多くのことが必要になることがあります！ 実験をしてデータを集めたり、開発中のシステム全体にモデルをどのように組み込むかを検討したりする必要があることが多いでしょう。データといえば、プロジェクトに必要なデータの探し方に注目しましょう。

## Gathering Data

### データ収集

For many types of projects, you may be able to find all the data you need online. The project we'll be completing in this chapter is a *bear detector*. It will discriminate between three types of bear: grizzly, black, and teddy bears. There are many images on the internet of each type of bear that we can use. We just need a way to find them and download them. We've provided a tool you can use for this purpose, so you can follow along with this chapter and create your own image recognition application for whatever kinds of objects you're interested in. In the fast.ai course, thousands of students have presented their work in the course forums, displaying everything from hummingbird varieties in Trinidad to bus types in Panama—one student even created an application that would help his fiancée recognize his 16 cousins during Christmas vacation!

多くの種類のプロジェクトでは、必要なデータをすべてオンラインで見つけることができるかもしれません。この章で完成させるプロジェクトは、熊探知機です。グリズリー、ブラック、ティディベアの3種類の熊を識別します。インターネット上には、それぞれの種類のクマの画像がたくさんあるので、それを利用することができます。それを見つけてダウンロードする方法が必要なのです。そのために使えるツールを用意しましたので、この章を参考に、興味のある種類の画像認識アプリケーションを自作してみてください。fast.aiコースでは、トリニダードのハチドリからパナマのバスの種類まで、何千人の生徒がコースフォーラムで作品を発表しています。ある生徒は、クリスマス休暇中に婚約者が16人のいとこを認識できるようにするアプリケーションを作成しました！

At the time of writing, Bing Image Search is the best option we know of for finding and downloading images. It's free for up to 1,000 queries per month, and each query can download up to 150 images. However, something better might have come along between when we wrote this and when you're reading the book, so be sure to check out the [book's website](#) for our current recommendation.

この記事を書いている時点では、Bing画像検索は、画像を検索してダウンロードするための最良の選択肢です。月に1,000回まで無料で利用でき、1回のクエリで150枚まで画像をダウンロードできます。しかし、私たちがこの文章を書いたときと、あなたがこの本を読

むときとでは、もっと良いものが登場しているかもしれませんので、この本のウェブサイトで、私たちの現在のおすすめを確認してください。

important: Keeping in Touch With the Latest Services: Services that can be used for creating datasets come and go all the time, and their features, interfaces, and pricing change regularly too. In this section, we'll show how to use the Bing Image Search API available at the time this book was written. We'll be providing more options and more up to date information on the [book's website](#), so be sure to have a look there now to get the most current information on how to download images from the web to create a dataset for deep learning.

important: 最新のサービスに触れ続ける: データセットの作成に利用できるサービスは常に生まれては消え、その機能、インターフェース、価格も定期的に変化しています。このセクションでは、本書が執筆された時点で利用可能な Bing Image Search API の使用方法を紹介します。本書のウェブサイトでは、より多くの選択肢と最新の情報を提供する予定ですので、ディープラーニング用のデータセットを作成するためにウェブから画像をダウンロードする方法に関する最新の情報を得るために、今すぐそちらをご覧になってください。

## Clean

クリーン

To download images with Bing Image Search, sign up at [Microsoft Azure](#) for a free account. You will be given a key, which you can copy and enter in a cell as follows (replacing 'XXX' with your key and executing it):

Bing Image Search で画像をダウンロードするには、Microsoft Azure で無料アカウントにサインアップしてください。キーが配布されるので、それをコピーして以下のようにセルに入力します ('XXX'をキーに置き換えて実行します) :

In [ ]:

```
key = os.environ.get('AZURE_SEARCH_KEY', 'XXX')
```

Or, if you're comfortable at the command line, you can set it in your terminal with:

また、コマンドラインに慣れている方は、ターミナルで次のように設定することもできます:

```
export AZURE_SEARCH_KEY=your_key_here
```

and then restart Jupyter Notebook, and use the above line without editing it.

で、Jupyter Notebook を再起動し、上記の行を編集せずに使用します。

Once you've set key, you can use `search_images_bing`. This function is provided by the small `utils` class included with the notebooks online. If you're not sure where a function is defined, you can just type it in your notebook to find out:

`key` を設定したら、`search_images_bing` を使用することができます。この関数は、オンラインのノートブックに含まれる小さな `utils` クラスで提供されています。どこに関数が定義されているかわからない場合は、ノートブックにその関数を入力すればわかるようになります：

```
search_images_bing
```

Out[ ]:

```
<function fastbook.search_images_bing(key, term, min_sz=128, max_images=150)>
```

In [ ]:

```
results = search_images_bing(key, 'grizzly bear') ims = results.attrgot('contentUrl') len(ims)
```

Out[ ]:

```
150
```

We've successfully downloaded the URLs of 150 grizzly bears (or, at least, images that Bing Image Search finds for that search term).

150 のグリズリーベア(少なくとも、Bing Image Search がその検索語で見つけた画像)の URL をダウンロードすることに成功しました。

**NB:** there's no way to be sure exactly what images a search like this will find. The results can change over time.

We've heard of at least one case of a community member who found some unpleasant pictures of dead bears in their search results. You'll receive whatever images are found by the web search engine. If you're running this at work, or with kids, etc, then be cautious before you display the downloaded images.

注:このような検索でどのような画像が見つかるか、正確に確認する方法はありません。結果は時間の経過とともに変化します。少なくとも1件、コミュニティメンバーが検索結果に熊の死体の不快な写真を見つけたという話を聞いたことがあります。ウェブ検索エンジンが見つけた画像は、すべてあなたに届きます。もしあなたが仕事場や子供と一緒にこれを実行しているならば、ダウンロードした画像を表示する前に慎重になってください。

Let's look at one:

1つ見てみましょう:

In [ ]:

```
#hide ims =
[http://3.bp.blogspot.com/-S1scRCKI3vY/UHzV2kucsPI/AAAAAAA-k/YQ5UzHEm9Ss/s1600/Grizzly%2BBear%2
BWildlife.jpg]
```

In [ ]:

```
dest = 'images/grizzly.jpg' download_url(ims[0], dest)
```

In [ ]:

```
im = Image.open(dest) im.to_thumb(128,128)
```

Out[ ]:

This seems to have worked nicely, so let's use fastai's `download_images` to download all the URLs for each of our search terms. We'll put each in a separate folder:

それでは、fastai の `download_images` を使って、各検索語の URL をすべてダウンロードしましょう。それぞれ別のフォルダに入れることにします：

In [ ]:

```
bear_types = 'grizzly','black','teddy' path = Path('bears')
```

In [ ]:

```
if not path.exists(): path.mkdir() for o in bear_types: dest = (path/o) dest.mkdir(exist_ok=True) results =
search_images_bing(key, f'{o} bear') download_images(dest, urls=results.attrgot('contentUrl'))
```

Our folder has image files, as we'd expect:

私たちのフォルダには、予想通り画像ファイルがあります：

In [ ]:

```
fns = get_image_files(path) fns
```

Out[ ]:

```
(#406) [Path('bears/black/00000149.jpg'), Path('bears/black/00000095.jpg'), Path('bears/black/00000133.jpg'), Path('bears/black/00000062.jpg'), Path('bears/black/00000023.jpg'), Path('bears/black/00000029.jpg'), Path('bears/black/0000094.jpg'), Path('bears/black/00000124.jpg'), Path('bears/black/00000105.jpg'), Path('bears/black/00000046.jpg')...]
```

j: I just love this about working in Jupyter notebooks! It's so easy to gradually build what I want, and check my work every step of the way. I make a *lot* of mistakes, so this is really helpful to me...

j: Jupyter ノートブックで作業することについては、この点がとても気に入っています！自分の欲しいものを少しづつ作って、その都度チェックするのがとても簡単です。私は失敗が多いので、これは本当に助かります…。

Often when we download files from the internet, there are a few that are corrupt. Let's check:

インターネットからファイルをダウンロードしたとき、破損しているものがいくつかあることがよくあります。確認してみましょう：

In [ ]:

```
failed = verify_images(fns) failed
```

Out[ ]:

```
(#11) [Path('bears/black/00000147.jpg'), Path('bears/black/00000057.jpg'), Path('bears/black/00000140.jpg'), Path('bears/black/00000129.jpg'), Path('bears/teddy/00000006.jpg'), Path('bears/teddy/00000048.jpg'), Path('bears/teddy/00000076.jpg'), Path('bears/teddy/00000125.jpg'), Path('bears/teddy/00000090.jpg'), Path('bears/teddy/00000075.jpg')...]
```

To remove all the failed images, you can use `unlink` on each of them. Note that, like most fastai functions that return a collection, `verify_images` returns an object of type L, which includes the `map` method. This calls the passed function on each element of the collection:

失敗した画像をすべて削除するには、それぞれの画像に対して `unlink` を使用すればよい。コレクションを返す多くの fastai 関数と同様に、`verify_images` は `map` メソッドを含む L 型のオブジェクトを返すことに注意してください。これは、コレクションの各要素に対して渡された関数を呼び出すものです：

In [ ]:

```
failed.map(Path.unlink);
```

## Sidebar: Getting Help in Jupyter Notebooks

サイドバー: Jupyter Notebooks でヘルプを得る

Jupyter notebooks are great for experimenting and immediately seeing the results of each function, but there is also a lot of functionality to help you figure out how to use different functions, or even directly look at their source code. For instance, if you type in a cell:

Jupyter ノートブックは、各機能の結果をすぐに確認できる実験に最適ですが、さまざまな機能の使い方を考えたり、そのソースコードを直接見たりするための機能も充実しています。例えば、あるセルに入力すると

```
??verify_images
```

a window will pop up with:

Signature: verify\_images(fns)

Source:

```
def verify_images(fns):
    "Find images in `fns` that can't be opened"
    return [fn for i, fn in
            enumerate(parallel(verify_image, fns)) if not o]
```

File: ~git/fastai/fastai/vision/utils.py

Type: function

This tells us what argument the function accepts (fns), then shows us the source code and the file it comes from. Looking at that source code, we can see it applies the function verify\_image in parallel and only keeps the image files for which the result of that function is False, which is consistent with the doc string: it finds the images in fns that can't be opened.

これは、この関数がどのような引数(fns)を受け取るかを示し、ソースコードとその元となるファイルを表示します。そのソースコードを見ると、verify\_image という関数を並列に適用し、その関数の結果が False の画像ファイルだけを保持していることがわかります。これは doc の文字列と一致しており、fns の中から開くことができない画像を見つけることができます。

Here are some other features that are very useful in Jupyter notebooks:

その他、Jupyter ノートブックで非常に便利な機能をいくつか紹介します:

- At any point, if you don't remember the exact spelling of a function or argument name, you can press Tab to get autocompletion suggestions.

関数名や引数名の正確なスペルを覚えていない場合、Tab キーを押すことで自動補完候補を表示することができます。

- When inside the parentheses of a function, pressing Shift and Tab simultaneously will display a window with the signature of the function and a short description. Pressing these keys twice will expand the documentation, and pressing them three times will open a full window with the same information at the bottom of your screen.

関数の括弧の中にいるとき、Shift と Tab を同時に押すと、関数のシグネチャーと短い説明を表示するウィンドウが表示されます。これらのキーを 2 回押すとドキュメントが拡大され、3 回押すと同じ情報を含むフルウィンドウが画面の下に表示されます。

- In a cell, typing `?func_name` and executing will open a window with the signature of the function and a short description.

セル内で `?func_name` と入力して実行すると、関数のシグネチャと簡単な説明が表示されるウィンドウが表示されます。

- In a cell, typing `??func_name` and executing will open a window with the signature of the function, a short description, and the source code.

セル内で `??func_name` と入力して実行すると、関数のシグネチャと短い説明、そしてソースコードが表示されたウィンドウが開きます。

- If you are using the fastai library, we added a `doc` function for you: executing `doc(func_name)` in a cell will open a window with the signature of the function, a short description and links to the source code on GitHub and the full documentation of the function in the [library docs](#).

セル内で `doc(func_name)` を実行すると、関数のシグネチャ、短い説明、GitHub のソースコードへのリンク、ライブラリの `docs` にある関数の完全なドキュメントが表示されたウィンドウが開かれます。

- 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.

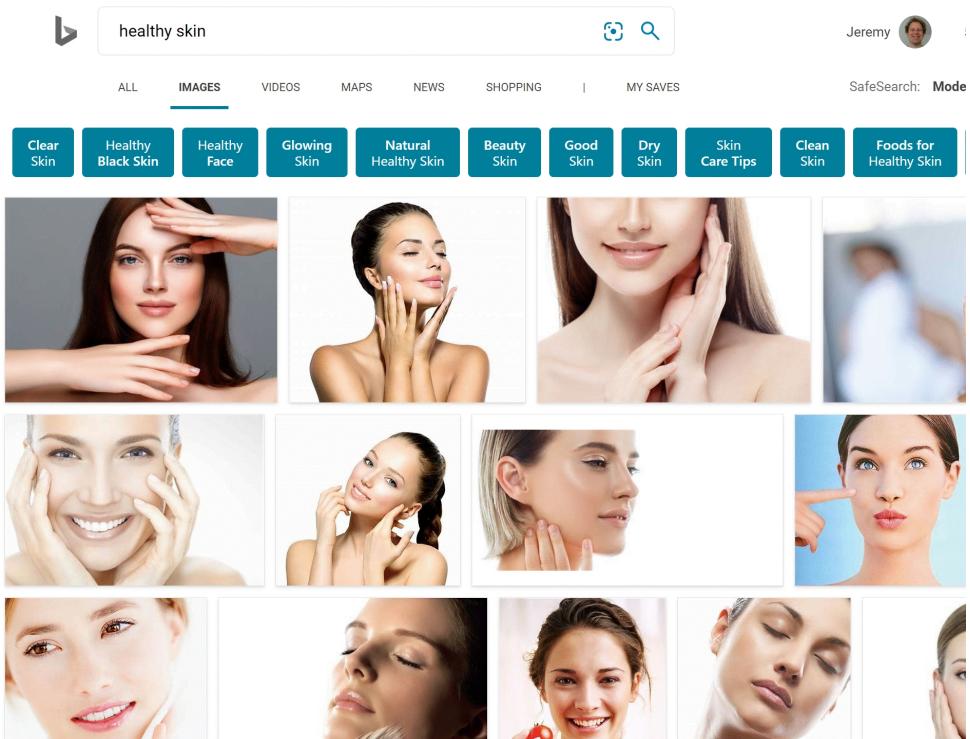
ドキュメントとは関係ありませんが、非常に便利です。エラーが発生したときにいつでもヘルプを得るには、次のセルに `%debug` と入力して実行すると、Python デバッガが開き、すべての変数の内容を調べることができます。

## End sidebar

サイドバーを終了する

One thing to be aware of in this process: as we discussed in <<chapter\_intro>>, models can only reflect the data used to train them. And the world is full of biased data, which ends up reflected in, for example, Bing Image Search (which we used to create our dataset). For instance, let's say you were interested in creating an app that could help users figure out whether they had healthy skin, so you trained a model on the results of searches for (say) "healthy skin." <<healthy\_skin>> shows you the kinds of results you would get.

このプロセスで注意しなければならないのは、<>で説明したように、モデルは訓練に使われたデータしか反映できないことです。そして、世の中には偏ったデータが溢れています。例えば Bing 画像検索（今回のデータセット作成に使用したもの）にも反映されてしまう。例えば、ユーザーが健康な肌かどうかを知るためのアプリを作りたいと考え、"healthy skin" の検索結果でモデルを学習させたとしましょう。<>は、どのような結果を得ることができるかを示しています。



With this as your training data, you would end up not with a healthy skin detector, but a *young white woman touching her face* detector! Be sure to think carefully about the types of data that you might expect to see in practice in your application, and check carefully to ensure that all these types are reflected in your model's source data. footnote:[Thanks to Deb Raji, who came up with the "healthy skin" example. See her paper "[Actionable Auditing: Investigating the Impact of Publicly Naming Biased Performance Results of Commercial AI Products](#)" for more fascinating insights into model bias.]

これをトレーニングデータとして使用すると、健康な肌の検出器ではなく、若い白人女性が顔を触っている検出器になってしまうのです！あなたのアプリケーションで実際に目にできる可能性のあるデータの種類をよく考え、これらの種類がすべてモデルのソースデータに反映されていることを慎重に確認してください。脚注：[「健康な肌」の例を思いついた Deb Raji に感謝します。彼女の論文「Actionable Auditing」を参照： モデルの偏りに関するより魅力的な洞察については、「Actionable Auditing: Investigating the Impact of Publicly Naming Biased Performance Results of Commercial AI Products」]を参照してください。

Now that we have downloaded some data, we need to assemble it in a format suitable for model training. In fastai, that means creating an object called DataLoaders.

さて、データをダウンロードしたら、モデルのトレーニングに適した形式に組み立てる必要があります。fastai では、DataLoaders と呼ばれるオブジェクトを作成することになります。

## From Data to DataLoaders

データから DataLoaders へ

DataLoaders is a thin class that just stores whatever DataLoader objects you pass to it, and makes them available as train and valid. Although it's a very simple class, it's very important in fastai: it provides the data for your model. The key functionality in DataLoaders is provided with just these four lines of code (it has some other minor functionality we'll skip over for now):

DataLoaders は、あなたが渡した DataLoader オブジェクトを保存し、train と valid として利用できるようにするだけの薄いクラスです。非常にシンプルなクラスですが、fastai では非常に重要です： モデルのためのデータを提供します。DataLoaders の主要な機能は、この 4 行のコードだけで提供されます（他にも細かい機能がありますが、今は省略します）：

```
class DataLoaders(GetAttr):
    def __init__(self, *loaders): self.loaders = loaders
    def __getitem__(self, i): return self.loaders[i]
    train,valid = add_props(lambda i, self: self[i])
    jargon: 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 are made available as properties.
```

専門用語： DataLoaders（データローダー）です： Fastai クラスは、渡された複数の DataLoader オブジェクト（通常は train と valid ですが、いくつでも持つことが可能です）を格納します。最初の 2 つはプロパティとして利用可能です。

Later in the book you'll also learn about the Dataset and Datasets classes, which have the same relationship.

この本の後半では、同じような関係を持つ Dataset と Datasets クラスについても学びます。

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

ダウンロードしたデータを DataLoaders オブジェクトにするために、少なくとも 4 つのことを fastai に伝える必要があります：

- What kinds of data we are working with  
扱うデータの種類
- How to get the list of items  
アイテム一覧の取得方法
- How to label these items  
これらのラベルの貼り方について
- How to create the validation set  
バリデーションセットの作成方法

So far we have seen a number of *factory methods* for particular combinations of these things, which are convenient when you have an application and data structure that happen to fit into those predefined methods. For when you don't, fastai has an extremely flexible system called the *data block API*. With this API you can fully customize

every stage of the creation of your DataLoaders. Here is what we need to create a DataLoaders for the dataset that we just downloaded:

これまで、これらの組み合わせのためのファクトリーメソッドをいくつか見てきましたが、アプリケーションとデータ構造がたまたまこれらの定義済みメソッドに適合する場合に便利です。そうでない場合のために、fastaiはデータブロック API という非常に柔軟なシステムを持っています。この API を使えば、DataLoaders を作成するすべての段階を完全にカスタマイズすることができます。ここでは、先ほどダウンロードしたデータセットの DataLoaders を作成するために必要なものを示します：

```
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))
```

Let's look at each of these arguments in turn. First we provide a tuple where we specify what types we want for the independent and dependent variables:

では、これらの議論を順番に見ていきましょう。まず、独立変数と従属変数の種類を指定するタプルを用意します：

### blocks=(ImageBlock, CategoryBlock)

The *independent variable* is the thing we are using to make predictions from, and the *dependent variable* is our target. In this case, our independent variables are images, and our dependent variables are the categories (type of bear) for each image. We will see many other types of block in the rest of this book.

### blocks=(ImageBlock, CategoryBlock)

独立変数とは、予測をするために使うもので、従属変数とは、私たちのターゲットです。この場合、独立変数は画像で、従属変数は各画像のカテゴリ（熊の種類）です。この本の残りの部分では、他にも多くの種類のブロックを見ることになります。

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):

この DataLoaders の場合、基礎となるアイテムはファイルパスです。これらのファイルのリストを取得する方法を fastai に伝えなければなりません。`get_image_files` 関数はパスを受け取り、そのパスに含まれるすべての画像のリストを返します（デフォルトでは再帰的に）：

### get\_items=get\_image\_files

Often, datasets that you download will already have a validation set defined. Sometimes this is done by placing the images for the training and validation sets into different folders. Sometimes it is done by providing a CSV file in which each filename is listed along with which dataset it should be in. There are many ways that this can be done, and fastai provides a very general approach that allows you to use one of its predefined classes for this, or to write

your own. In this case, however, we simply want to split our training and validation sets randomly. However, we would like to have the same training/validation split each time we run this notebook, so 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):

ダウンロードしたデータセットには、すでに検証セットが定義されていることがよくあります。この場合、トレーニングセットとバリデーションセットの画像を別々のフォルダに入れることで対応できることがあります。また、CSVファイルでファイル名を列挙し、それがどのデータセットに含まれるべきかを示すこともあります。このような方法はたくさんありますが、fastai は非常に一般的な方法で、あらかじめ定義されたクラスを使用したり、独自のクラスを作成したりすることができます。しかし、今回のケースでは、単にトレーニングセットと検証セットをランダムに分けたいだけです。しかし、このノートブックを実行するたびに同じトレーニング/バリデーション分割を行いたいので、ランダムシードを固定します（コンピュータは乱数の作り方を全く知らないのですが、単にランダムに見える数字のリストを作成します：）

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

The independent variable is often referred to as *x* and the dependent variable is often referred to as *y*. Here, we are telling fastai what function to call to create the labels in our dataset:

独立変数はしばしば *x* と呼ばれ、従属変数はしばしば *y* と呼ばれます。ここでは、データセットのラベルを作成するために呼び出す関数を fastai に指示しています：

```
get_y=parent_label
```

*parent\_label* is a function provided by fastai that simply gets the name of the folder a file is in. Because we put each of our bear images into folders based on the type of bear, this is going to give us the labels that we need.

*parent\_label* は fastai が提供する関数で、単純にファイルの入っているフォルダの名前を取得します。熊の画像を熊の種類に応じたフォルダに入れるので、これで必要なラベルを得ることができます。

Our images are all different sizes, and this is a problem for deep learning: we don't feed the model one image at a time but several of them (what we call a *mini-batch*). To group them in a big array (usually called a *tensor*) that is going to go through our model, they all need to be of the same size. So, 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, or so forth. fastai includes many predefined transforms; we use the *Resize* transform here:

画像はすべて異なるサイズですが、これはディープラーニングの問題で、モデルに一度に 1 つの画像を与えるのではなく、複数の画像を与えます（ミニバッチと呼びます）。モデルに与える画像は 1 枚ではなく、複数の画像（ミニバッチと呼ばれるもの）です。これらを大きな配列（通常はテンソルと呼ばれる）にまとめ、モデルを通過させるためには、すべての画像を同じサイズにすることが必要です。そこで、これらの画像を同じサイズにリサ

イズするトランスフォームを追加する必要があります。アイテム変換は、画像やカテゴリなど、個々のアイテムに対して実行されるコードの断片です。fastaiには多くの定義済み変換が含まれていますが、ここでは Resize 変換を使用します：

### item\_tfms=Resize(128)

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:

このコマンドで、DataBlock オブジェクトができました。これは、DataLoaders を作成するためのテンプレートのようなものです。この場合、fastai に実際のデータソース（この場合、画像を見つけることができるパス）を伝える必要があります：

```
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:

DataLoaders には、検証用 DataLoaders とトレーニング用 DataLoaders があります。DataLoader は、一度に数個のアイテムのバッチを GPU に提供するクラスです。次の章でこのクラスについてもっとたくさん学びます。DataLoader をループすると、fastai は一度に 64 個の（デフォルトで）アイテムを提供し、すべて 1 つのテンソルに積み重ねます。DataLoader の show\_batch メソッドを呼び出すことで、これらのアイテムのいくつかを見てみることができます：

In [ ]:

```
dls.valid.show_batch(max_n=4, nrows=1)
```



By default Resize *crobs* the images to fit a square shape of the size requested, using the full width or height. This can result in losing some important details. Alternatively, you can ask fastai to pad the images with zeros (black), or squish/stretch them:

デフォルトでは、リサイズは、幅または高さをフルを使って、要求されたサイズの正方形の形状に合うように画像を切り取ります。その結果、重要なディテールが失われることがあります。また、fastai に頼んで、画像をゼロ（黒）で埋めるか、縮小・伸張させることができます：

In [ ]:

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



In [ ]:

```
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)
```



All of these approaches seem somewhat wasteful, or problematic. If we squish or stretch the images they end up as unrealistic shapes, leading to a model that learns that things look different to how they actually are, which we would expect to result in lower accuracy. If we crop the images then we remove some of the features that allow us to perform recognition. For instance, if we were trying to recognize breeds of dog or cat, we might end up cropping out a key part of the body or the face necessary to distinguish between similar breeds. If we pad the images then we have a whole lot of empty space, which is just wasted computation for our model and results in a lower effective resolution for the part of the image we actually use.

これらのアプローチは、いずれも無駄が多く、問題があるように思えます。画像を縮小したり引き伸ばしたりすると、非現実的な形になってしまい、物事が実際と異なって見えることを学習するモデルになってしまい、精度が落ちることが予想されます。また、画像を切り抜くと、認識を行うための特徴が失われます。例えば、犬や猫の品種を認識しようとした場合、似たような品種を区別するために必要な体や顔の重要な部分を切り抜いてしまうかもしれません。このような画像をパッド化すると、空いたスペースが多くなり、モデルの計算が無駄になり、実際に使用する部分の解像度が低くなってしまいます。

```
bears = bears.new(item_tfms=Resize(128, ResizeMethod.Pad, pad_mode='zeros'))  
dls = bears.dataloaders(path)
```



We used `unique=True` to have the same image repeated with different versions of this `RandomResizedCrop` transform. This is a specific example of a more general technique, called data augmentation.

`unique=True` を使用して、この `RandomResizedCrop` 変換の異なるバージョンで同じ画像を繰り返し表示させています。これは、データ拡張と呼ばれる、より一般的な手法の具体例です。

## Data Augmentation

### データ拡張

*Data augmentation* refers to creating random variations of our input data, such that they appear different, but do not actually change the meaning of the data. Examples of common data augmentation techniques for images are rotation, flipping, perspective warping, brightness changes and contrast changes. For natural photo images such as the ones we are using here, a standard set of augmentations that we have found work pretty well are provided with the `aug_transforms` function. Because our images are now all the same size, we can apply these augmentations to an entire batch of them using the GPU, which will save a lot of time. To tell fastai we want to use these transforms on a batch, we use the `batch_tfms` parameter (note that we're not using `RandomResizedCrop` in this example, so you can see the differences more clearly; we're also using double the amount of augmentation compared to the default, for the same reason):

データ補強とは、入力されたデータをランダムに変化させることで、見た目は異なるが、実際にはデータの意味を変えないようにすることである。画像のデータ補強の例としては、回転、反転、遠近法、明るさの変化、コントラストの変化などがあります。今回使用するような自然な写真画像については、`aug_transforms` 関数で、かなりうまく機能する標準的な補強のセットが提供されていることが分かっています。画像はすべて同じサイズなので、GPUを使ってこれらの拡張をバッチ全体に適用することができ、時間を大幅に節約することができます。バッチに対してこれらの変換を使いたいことを fastai に伝えるには、`batch_tfms` パラメータを使います(この例では `RandomResizedCrop` を使っていないので、違いがより明確にわかると思います：)

In [ ]:

```
bears = bears.new(item_tfms=Resize(128), batch_tfms=aug_transforms(mult=2)) dls = bears.dataloaders(path)  
dls.train.show_batch(max_n=8, nrows=2, unique=True)
```



Now that we have assembled our data in a format fit for model training, let's actually train an image classifier using it.

モデル学習に適した形でデータが揃ったので、実際にそれを使って画像分類器を学習してみましょう。

## Training Your Model, and Using It to Clean Your Data

モデルの学習とデータクリーニングに利用する

Time to use the same lines of code as in <<chapter\_intro>> to train our bear classifier.

それでは、<>と同じコードを使って、熊の分類器を訓練してみましょう。

We don't have a lot of data for our problem (150 pictures of each sort of bear at most), so to train our model, we'll use RandomResizedCrop with an image size of 224 px, which is fairly standard for image classification, and default aug\_transforms:

この問題のデータはそれほど多くないので（せいぜい 150 枚）、モデルを訓練するために、RandomResizedCrop を使い、画像サイズは 224px、これは画像分類としてはかなり標準的で、デフォルトの aug\_transforms を使うことにします：

In [ ]:

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

We can now create our Learner and fine-tune it in the usual way:

これで Learner を作成し、通常の方法で微調整を行うことができるようになりました：

In [ ]:

```
learn = vision_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

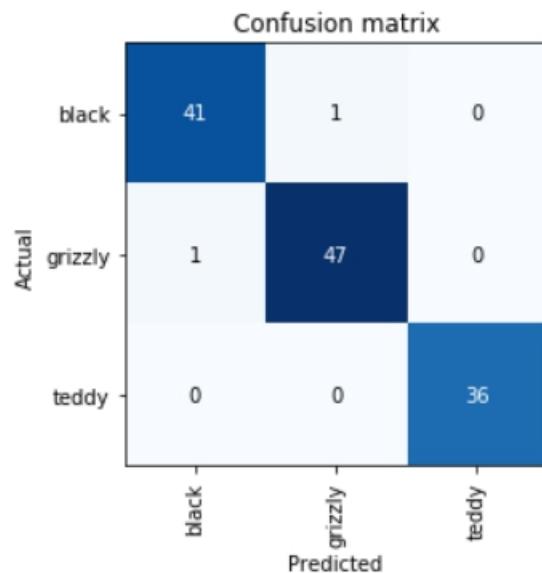
epoch	train_loss	valid_loss	error_rate	time
0	0.213371	0.112450	0.023810	00:05
1	0.173855	0.072306	0.023810	00:06
2	0.147096	0.039068	0.015873	00:06
3	0.123984	0.026801	0.015873	00:06

Now let's see whether the mistakes the model is making are mainly thinking that grizzlies are teddies (that would be bad for safety!), or that grizzlies are black bears, or something else. To visualize this, we can create a *confusion matrix*.

では、このモデルが犯している間違いは、主にグリズリーをteddy（安全上よくない！）と考えているのか、グリズリーをツキノワグマと考えているのか、それとも別のことなのかを見てみましょう。これを視覚化するために、混乱マトリックスを作成します：

In [ ]:

```
interp = ClassificationInterpretation.from_learner(learn) interp.plot_confusion_matrix()
```



The rows represent all the black, grizzly, and teddy bears in our dataset, respectively. The columns represent the images which the model predicted as black, grizzly, and teddy bears, respectively. Therefore, the diagonal of the matrix shows the images which were classified correctly, and the off-diagonal cells represent those which were classified incorrectly. This is one of the many ways that fastai allows you to view the results of your model. It is (of course!) calculated using the validation set. With the color-coding, the goal is to have white everywhere except the diagonal, where we want dark blue. Our bear classifier isn't making many mistakes!

行はデータセットに含まれるすべての黒熊、グリズリー熊、teddy bearをそれぞれ表す。列は、モデルがそれぞれ黒熊、グリズリー、teddy bearと予測した画像を表している。したがって、行列の対角線は正しく分類された画像を示し、対角線外のセルは間違って分類された画像を示しています。これは、fastai がモデルの結果を表示するための多くの方法の1つです。これは（もちろん！）検証セットを使って計算されたものです。色分けでは、対角線を除くすべての場所を白にするのが目標で、ここでは濃い青にします。このクマ型分類器は、あまり間違いを犯しません！

It's helpful to see where exactly our errors are occurring, to see whether they're due to a dataset problem (e.g., images that aren't bears at all, or are labeled incorrectly, etc.), or a model problem (perhaps it isn't handling images taken with unusual lighting, or from a different angle, etc.). To do this, we can sort our images by their *loss*.

データセットの問題（例：クマではない画像、誤ったラベル付けなど）なのか、モデルの問題（異常な照明、異なる角度から撮影された画像などを処理できていない）なのかを確認するためには、どこでエラーが発生しているのかを正確に把握することが有効です。そのために、画像を損失額でソートすることができます。

The loss is a number that is higher if the model is incorrect (especially if it's also confident of its incorrect answer), or if it's correct, but not confident of its correct answer. In a couple of chapters we'll learn in depth how loss is calculated and used in the training process. For now, `plot_top_losses` shows us the images with the highest loss in our dataset. As the title of the output says, each image is labeled with four things: prediction, actual (target label), loss, and probability. The *probability* here is the confidence level, from zero to one, that the model has assigned to its prediction:

損失とは、モデルが不正解である場合（特に不正解にも自信がある場合）、または正解であるが正解に自信がない場合に高くなる数値です。数章で、損失がどのように計算され、トレーニングプロセスで使用されるかを深く学びます。とりあえず、`plot_top_losses` は、データセットの中で最も損失が大きい画像を示しています。出力のタイトルにあるように、各画像には、予測、実際（ターゲットラベル）、損失、確率の4つがラベル付けされています。ここでいう確率とは、モデルが予測に割り当てた信頼度（0から1まで）のことです：

In [ ]:

```
interp.plot_top_losses(5, nrows=1)
```

#### Prediction/Actual/Loss/Probability



This output shows that the image with the highest loss is one that has been predicted as "grizzly" with high confidence. However, it's labeled (based on our Bing image search) as "black." We're not bear experts, but it sure looks to us like this label is incorrect! We should probably change its label to "grizzly."

この出力から、最も損失が大きい画像は、高い信頼性で「グリズリー」と予測されたものであることがわかります。しかし、この画像は（Bing の画像検索に基づき）"black"とラベル付けされています。私たちは熊の専門家ではありませんが、このラベルは確かに間違っているように見えます！このラベルを "グリズリー"に変えるべきでしょう。

The intuitive approach to doing data cleaning is to do it *before* you train a model. But as you've seen in this case, a model can actually help you find data issues more quickly and easily. So, we normally prefer to train a quick and simple model first, and then use it to help us with data cleaning.

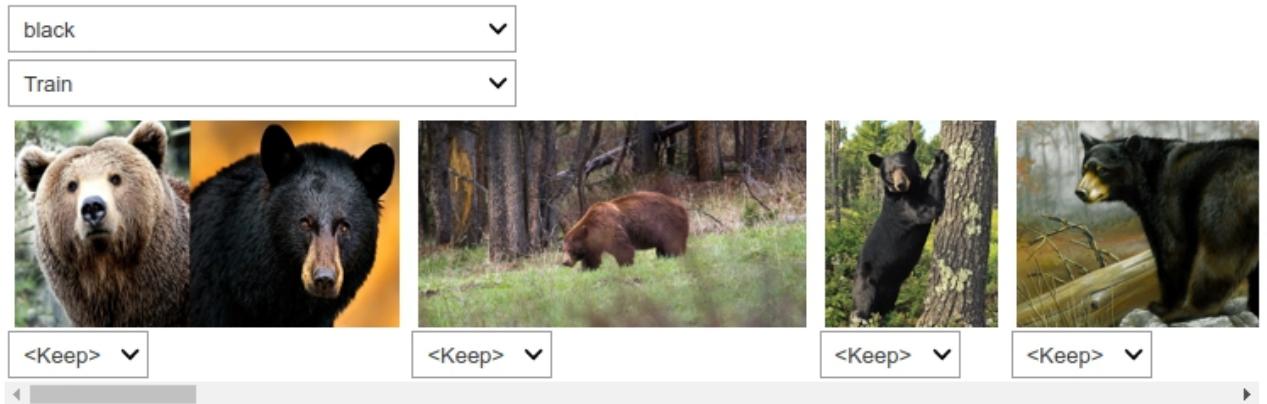
データクリーニングの直感的なアプローチは、モデルを訓練する前に行うことです。しかし、このケースでお分かりのように、モデルは実際にデータの問題をより迅速かつ簡単に見つけるのに役立ちます。ですから、私たちは通常、まず素早くシンプルなモデルをトレーニングし、それをデータクリーニングに役立てることを好みます。

fastai includes a handy GUI for data cleaning called `ImageClassifierCleaner` that allows you to choose a category and the training versus validation set and view the highest-loss images (in order), along with menus to allow images to be selected for removal or relabeling:

fastai には `ImageClassifierCleaner` というデータクリーニングのための便利な GUI があり、カテゴリとトレーニング対検証セットを選択し、最も損失の大きい画像を（順番に）表示できるほか、画像を選択して削除や再ラベリングができるメニューが用意されています：

In [ ]:

```
#hide_output cleaner = ImageClassifierCleaner(learn) cleaner
VBox(children=(Dropdown(options=('black', 'grizzly', 'teddy'), value='black'),
Dropdown(options=('Train', 'Val...')))
```



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

We can see that amongst our "black bears" is an image that contains two bears: one grizzly, one black. So, we should choose <Delete> in the menu under this image. ImageClassifierCleaner doesn't actually do the deleting or changing of labels for you; it just returns the indices of items to change. So, for instance, to delete (unlink) all images selected for deletion, we would run:

黒熊」の中に、グリズリーと黒熊の 2 頭の熊を含む画像があることがわかります。そこで、この画像の下にあるメニューからを選択します。 ImageClassifierCleaner は、実際にラベルの削除や変更を行うわけではなく、変更する項目のインデックスを返すだけです。そのため、例えば、削除対象として選択されたすべての画像を削除（リンク解除）する場合は、次のように実行します：

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

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

違うカテゴリを選択した画像を移動するには、次のように実行します：

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

s: Cleaning the data and getting it ready for your model are two of the biggest challenges for data scientists; they say it takes 90% of their time. The fastai library aims to provide tools that make it as easy as possible.

s: データのクリーニングとモデルの準備は、データサイエンティストにとって最大の課題の 2 つで、時間の 9 割を占めると言われています。fastai ライブライアリは、それをできるだけ簡単にためのツールを提供することを目的としています。

We'll be seeing more examples of model-driven data cleaning throughout this book. Once we've cleaned up our data, we can retrain our model. Try it yourself, and see if your accuracy improves!

本書では、モデル駆動型データクリーニングの例をもっとたくさん見ていきます。データをきれいにしたら、モデルを再トレーニングすることができます。自分でも試してみて、精度が上がるかどうか確認してみてください！

**note:** No Need for Big Data: After cleaning the dataset using these steps, we generally are seeing 100% accuracy on this task. We even see that result when we download a lot fewer images than the 150 per class we're using here. As you can see, the common complaint that *you need massive amounts of data to do deep learning* can be a very long way from the truth!

**note:** ビッグデータは必要ない： この手順でデータセットをクリーニングした後、一般的にこのタスクの精度は 100%になります。また、1 クラスあたり 150 枚という少ない枚数の画像をダウンロードした場合でも、この結果を得ることができました。このように、ディープラーニングを行うには大量のデータが必要だという一般的な苦情は、真実から非常に遠いところにあることがわかります！

Now that we have trained our model, let's see how we can deploy it to be used in practice.

さて、モデルの学習が終わったところで、実際に使用するためにどのように展開するかを見てみましょう。

## Turning Your Model into an Online Application

モデルをオンラインアプリケーションにする

We are now going to look at what it takes to turn this model into a working online application. We will just go as far as creating a basic working prototype; we do not have the scope in this book to teach you all the details of web application development generally.

これから、このモデルを実際に動作するオンラインアプリケーションにするために必要なことを見ていきます。この本では、ウェブアプリケーション開発の詳細をすべて教えることはできませんので、基本的な動作するプロトタイプを作成するところまでとします。

## Using the Model for Inference

推論のためのモデルの使用

Once you've got a model you're happy with, you need to save it, so that you can then copy it over to a server where you'll use it in production. Remember that a model consists of two parts: the *architecture* and the trained *parameters*. The easiest way to save the model is to save both of these, because that way when you load a model you can be sure that you have the matching architecture and parameters. To save both parts, use the export method.

満足のいくモデルができたら、それを保存して、本番で使用するサーバーにコピーする必要があります。モデルは、アーキテクチャと学習済みパラメータの2つの部分から構成されていることを忘れないでください。モデルを保存する最も簡単な方法は、この2つを保存することです。そうすれば、モデルをロードするときに、アーキテクチャとパラメータが一致していることを確認することができるからです。両方の部分を保存するには、`export` メソッドを使用します。

This method even saves the definition of how to create your DataLoaders. This is important, because otherwise you would have to redefine how to transform your data in order to use your model in production. fastai automatically uses your validation set DataLoader for inference by default, so your data augmentation will not be applied, which is generally what you want.

このメソッドでは、DataLoaders の作成方法に関する定義も保存されます。これは重要なことです。そうしないと、モデルを本番で使うために、データの変換方法を再定義しなければならなくなるからです。fastai はデフォルトで、推論に検証セットの DataLoader を自動的に使うので、データの増強は適用されません。これは一般的に望むことです。

When you call `export`, fastai will save a file called "export.pkl":

`export` を呼び出すと、fastai は "export.pkl" という名前のファイルを保存します:

In [ ]:

```
learn.export()
```

Let's check that the file exists, by using the `ls` method that fastai adds to Python's Path class:

Python の Path クラスに fastai が追加した `ls` メソッドを使って、ファイルが存在するかどうか確認してみましょう:

In [ ]:

```
path = Path().path.ls(file_exts='.pkl')
```

Out[ ]:

(#1) [Path('export.pkl')]

You'll need this file wherever you deploy your app to. For now, let's try to create a simple app within our notebook.

このファイルは、アプリをデプロイする場所に必要です。とりあえず、ノートブックの中に簡単なアプリを作ってみましょう。

When we use a model for getting predictions, instead of training, we call it *inference*. To create our inference learner from the exported file, we use `load_learner` (in this case, this isn't really necessary, since we already have a working Learner in our notebook; we're just doing it here so you can see the whole process end-to-end):

学習ではなく、予測値を得るためにモデルを使用することを推論と呼びます。エクスポートしたファイルから推論学習器を作成するには、`load_learner` を使用します（この場合、ノートブックにすでに学習器があるので、これはあまり必要ではありません：）

In [ ]:

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

When we're doing inference, we're generally just getting predictions for one image at a time. To do this, pass a filename to predict:

推論を行う場合、一般的には一度に 1 つの画像の予測値を得るだけです。これを行うには、predict にファイル名を渡します：

In [ ]:

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

Out[ ]:

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

This has returned three things: the predicted category in the same format you originally provided (in this case that's a string), the index of the predicted category, and the probabilities of each category. The last two are based on the order of categories in the *vocab* of the DataLoaders; that is, the stored list of all possible categories. At inference time, you can access the DataLoaders as an attribute of the Learner:

これは、最初に提供したのと同じ形式の予測カテゴリ（この場合は文字列）、予測カテゴリのインデックス、および各カテゴリの確率の 3 つを返しました。最後の 2 つは、DataLoaders の vocab におけるカテゴリの順序、つまり、すべての可能なカテゴリの保存されたリストに基づいています。推論時に、学習者の属性として DataLoaders にアクセスすることができます：

In [ ]:

```
learn_inf.dls.vocab
```

Out[ ]:

```
(#3) ['black','grizzly','teddy']
```

We can see here that if we index into the vocab with the integer returned by predict then we get back "grizzly," as expected. Also, note that if we index into the list of probabilities, we see a nearly 1.00 probability that this is a grizzly.

predict が返す整数を用いて語彙にインデックスを付けると、予想通り「grizzly」が返されることがわかります。また、確率のリストにインデックスを付けると、これがグリズリーである確率はほぼ 1.00 であることがわかります。

We know how to make predictions from our saved model, so we have everything we need to start building our app. We can do it directly in a Jupyter notebook.

保存したモデルから予測を行う方法がわかったので、アプリを作り始めるのに必要なものはすべてそろっています。Jupyter ノートブックで直接行うことができます。

## Creating a Notebook App from the Model

モデルからノートアプリを作成する

To use our model in an application, we can simply treat the predict method as a regular function. Therefore, creating an app from the model can be done using any of the myriad of frameworks and techniques available to application developers.

このモデルをアプリケーションで使用するには、predict メソッドを通常の関数として扱えばよいのです。したがって、モデルからアプリケーションを作成するには、アプリケーション開発者が利用できる無数のフレームワークやテクニックのいずれかを使用することができます。

However, most data scientists are not familiar with the world of web application development. So let's try using something that you do, at this point, know: 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:

しかし、ほとんどのデータサイエンティストは、ウェブアプリケーション開発の世界に精通しているわけではありません。Jupyter ノートブックだけで、完全に動作するウェブアプリケーションを作成できることがわかったのです！これを実現するために必要なものは 2 つです：

- 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. For instance, the image cleaner that we saw earlier in this chapter is entirely written with IPython widgets. However, we don't want to require users of our application to run Jupyter themselves.

IPython ウィジェットは、JavaScript と Python の機能を Web ブラウザ上で実現する GUI コンポーネントで、Jupyter ノートブック内で作成・利用することが可能です。例えば、本章の前半で見た画像クリーナーは、すべて IPython ウィジェットで書かれています。しかし、私たちのアプリケーションの利用者に Jupyter を自分で起動することを求めたいとは思いません。

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. Voilà is taking advantage of the fact that a notebook *already is* a kind of web application, just a rather complex one that depends on another web application: Jupyter itself. Essentially, it helps us automatically convert the complex web application we've already implicitly made (the notebook) into a simpler, easier-to-deploy web application, which functions like a normal web application rather than like a notebook.

そのために Voilàが存在するのです。IPython ウィジェットで構成されたアプリケーションを、エンドユーザーが Jupyter を全く使わなくても利用できるようにするためのシステムです。Voilàは、ノートブックがすでに一種のウェブアプリケーションであるという事実を利用しています： Jupyter そのものです。基本的には、私たちが暗黙のうちに作った複雑

な Web アプリケーション（ノートブック）を、よりシンプルでデプロイしやすい Web アプリケーションに自動的に変換してくれるもので、ノートブックのような機能ではなく、通常の Web アプリケーションのように機能します。

But we still have the advantage of developing in a notebook, so with ipywidgets, we can build up our GUI step by step. We will use this approach to create a simple image classifier. First, we need a file upload widget:

しかし、ノートブックで開発するという利点は残っているので、ipywidgets を使えば、GUI をステップバイステップで作り上げていくことができます。この方法を用いて、簡単な画像分類器を作ってみます。まず、ファイルアップロードウィジェットが必要です：

In [ ]:

```
#hide_output  
btn_upload = widgets.FileUpload()  
btn_upload  
FileUpload(value={}, description='Upload')  

```

Now we can grab the image:

これで、画像を取り込むことができます：

In [ ]:

```
#hide  
# For the book, we can't actually click an upload button, so we fake it  
btn_upload = SimpleNamespace(data = ['images/grizzly.jpg'])  
img = PILImage.create(btn_upload.data[-1])
```

In [ ]:



We can use an Output widget to display it:

Output ウィジェットを使って表示させることができます：

In [ ]:

```
#hide_output  
out_pl = widgets.Output()  
out_pl.clear_output() with
```

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



Then we can get our predictions:

そして、予測値を得ることができます：

In [ ]:

```
pred,pred_idx,probs = learn_inf.predict(img)
```

and use a Label to display them:

Label を使用して表示します：

In [ ]:

```
#hide_output
lbl_pred = widgets.Label()
lbl_pred.value = f'Prediction: {pred}; Probability: {probs[pred_idx]:.04f}'
lbl_pred
```

Label(value='Prediction: grizzly; Probability: 1.0000')

Prediction: grizzly; Probability: 1.0000

We'll need a button to do the classification. It looks exactly like the upload button:

分類を行うためのボタンが必要です。アップロードボタンと全く同じように見えます：

In [ ]:

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

Button(description='Classify', style=ButtonStyle())

We'll also need a *click event handler*; that is, a function that will be called when it's pressed. We can just copy over the lines of code from above:

また、クリックイベントハンドラーも必要で、つまり、押されたときに呼び出される関数が必要です。上のコードの行をそのままコピーすればいい：

In [ ]:

```
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)
```

You can test the button now by pressing it, and you should see the image and predictions update automatically!

ボタンを押すと、画像と予測が自動的に更新されるのがわかると思いますので、これでテストできます！

We can now put them all in a vertical box (VBox) to complete our GUI:

あとは、これらを縦長のボックス（VBox）に入れて、GUI を完成させましょう：

In [ ]:

```
#hide
#Putting back btn_upload to a widget for next cell
btn_upload = widgets.FileUpload()

#hide_output
VBox([widgets.Label('Select your bear!'),
      btn_upload, btn_run, out_pl, lbl_pred])
```

In [ ]:

VBox(children=(Label(value='Select your bear!'), FileUpload(value={}, description='Upload'), Button(description='...')))

Select your bear!

 Upload (0)

Classify



Prediction: grizzly; Probability: 1.0000

We have written all the code necessary for our app. The next step is to convert it into something we can deploy.  
アプリに必要なコードはすべて書きました。次のステップは、これをデプロイできるようなものに変換することです。

## Turning Your Notebook into a Real App

ノートを本物のアプリにする

In [ ]:

```
#hide  
#!pip install voila  
!jupyter serverextension enable --sys-prefix voila
```

Now that we have everything working in this Jupyter notebook, we can create our application. To do this, start a new notebook and add to it only the code needed to create and show the widgets that you need, and markdown for any text that you want to appear. Have a look at the `bear_classifier` notebook in the book's repo to see the simple notebook application we created.

さて、この Jupyter ノートブックですべてが動作するようになったので、アプリケーションを作成することができます。新しいノートブックを作成し、必要なウィジェットを作成・表示するためのコードと、表示させたいテキストのマークダウンのみを追加してください。この本のレポにある `bear_classifier` ノートブックで、私たちが作った簡単なノートブック・アプリケーションを見てみてください。

Next, install Voilà if you haven't already, by copying these lines into a notebook cell and executing it:

次に、まだインストールされていない方は、以下の行をノートブックのセルにコピーして実行することで、Voilàをインストールします：

```
!pip install voila  
!jupyter serverextension enable --sys-prefix voila
```

Cells that begin with a ! do not contain Python code, but instead contain code that is passed to your shell (bash, Windows PowerShell, etc.). If you are comfortable using the command line, which we'll discuss more later in this book, you can of course simply type these two lines (without the ! prefix) directly into your terminal. In this case, the first line installs the voila library and application, and the second connects it to your existing Jupyter notebook.

で始まるセルには Python のコードは含まれておらず、代わりにシェル（bash、Windows PowerShell など）に渡されるコードが含まれています。本書の後半で詳しく説明しますが、コマンドラインの使用に慣れているのであれば、もちろん、この 2 行をターミナルに直接

入力することもできます（接頭辞の「！」を除きます）。この場合、1行目は voila ライブ リーとアプリケーションをインストールし、2行目はそれを既存の Jupyter ノートブックに接続します。

Voilà runs Jupyter notebooks just like the Jupyter notebook server you are using now does, but it also does something very important: it removes all of the cell inputs, and only shows output (including ipywidgets), along with your markdown cells. So what's left is a web application! To view your notebook as a Voilà web application, replace the word "notebooks" in your browser's URL with: "voila/render". You will see the same content as your notebook, but without any of the code cells.

Voilàは、今使っている Jupyter ノートブックサーバと同じように Jupyter ノートブックを実行しますが、同時にとても重要なことを行います：セルの入力をすべて削除し、マークダウンセルと一緒に出力（ipywidgets を含む）だけを表示するのです。つまり、残るはウェブアプリケーションです！あなたのノートブックを Voilà ウェブアプリケーションとして表示するには、ブラウザの URL で "notebooks" という単語を次のように置き換えてください："voila/render" と書き換えてください。あなたのノートブックと同じ内容が表示されますが、コードのセルはありません。

Of course, you don't need to use Voilà or ipywidgets. Your model is just a function you can call (`pred,pred_idx,probs = learn.predict(img)`), so you can use it with any framework, hosted on any platform. And you can take something you've prototyped in ipywidgets and Voilà and later convert it into a regular web application. We're showing you this approach in the book because we think it's a great way for data scientists and other folks that aren't web development experts to create applications from their models.

もちろん、Voilà や ipywidgets を使う必要はありません。モデルは単に呼び出せる関数（`pred,pred_idx,probs = learn.predict(img)`）なので、どんなフレームワークでも、どんなプラットフォームでホストされていても使うことができます。また、ipywidgets や Voilà でプロトタイプを作成したものを、後で通常の Web アプリケーションに変換することも可能です。本書でこの方法を紹介しているのは、データサイエンティストやウェブ開発の専門家ではない人たちが、モデルからアプリケーションを作成するのに最適な方法だと考えているからです。

We have our app, now let's deploy it!  
アプリができたので、デプロイしてみましょう！

## Deploying your app

アプリをデプロイする

As you now know, you need a GPU to train nearly any useful deep learning model. So, do you need a GPU to use that model in production? No! You almost certainly *do not need a GPU to serve your model in production*. There are a few reasons for this:

もうご存知の通り、ほぼすべての有用なディープラーニングモデルをトレーニングするためには、GPU が必要です。では、そのモデルを本番で使うのに GPU は必要なのでしょう

か？ いいえ！ 本番でモデルを使うのに GPU が必要なことはほとんどありません。これにはいくつかの理由があります：

- As we've seen, 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 normally enough work to do in a single image to keep a GPU busy for long enough for it to be very efficient. So, a CPU will often be more cost-effective.

これまで見てきたように、GPU が有用なのは、同一の作業を多数並行して行う場合だけです。例えば、画像分類を行う場合、通常は一度に 1 人のユーザーの画像を分類することになります。そして、通常、1 つの画像には、GPU を効率よく長時間働かせるのに十分な量の作業がありません。そのため、CPU の方が費用対効果が高いことが多いでしょう。

- An alternative could be to wait for a few users to submit their images, and then batch them up and process them all at once on a GPU. But then you're asking your users to wait, rather than getting answers straight away! And you need a high-volume site for this to be workable. If you do need this functionality, you can use a tool such as Microsoft's [ONNX Runtime](#), or [AWS Sagemaker](#)

何人かのユーザーが画像を投稿するのを待ち、それをバッチ処理して GPU で一度に処理する、という方法もあります。しかし、それではユーザーに待ってもらうことになり、すぐに答えを得ることができません！ また、この機能を実現するためには、大容量のサイトが必要です。この機能が必要な場合は、Microsoft の ONNX Runtime や AWS の Sagemaker のようなツールを使用することができます。

- The complexities of dealing with GPU inference are significant. In particular, the GPU's memory will need careful manual management, and you'll need a careful queueing system to ensure you only process one batch at a time.

GPU 推論を扱うには、複雑な作業が必要です。特に、GPU のメモリは手動で慎重に管理する必要があり、一度に 1 つのバッチしか処理しないように、慎重な待ち行列システムが必要です。

- There's a lot more market competition in CPU than GPU servers, as a result of which there are much cheaper options available for CPU servers.

GPU サーバーよりも CPU サーバーの方が市場競争が激しく、その結果、CPU サーバーの方がはるかに安いオプションが用意されています。

Because of the complexity of GPU serving, many systems have sprung up to try to automate this. However, managing and running these systems is also complex, and generally requires compiling your model into a different form that's specialized for that system. It's typically preferable to avoid dealing with this complexity until/unless your app gets popular enough that it makes clear financial sense for you to do so.

GPU サーバーは複雑なため、これを自動化しようとするシステムが多く登場しています。しかし、これらのシステムの管理・運用も複雑で、一般的には、そのシステムに特化した別

の形式にモデルをコンパイルする必要があります。このような複雑な処理は、アプリが十分に普及し、経済的な意味が明らかになるまで、あるいはそうでない限り、通常は避けることが望ましい。

For at least the initial prototype of your application, and for any hobby projects that you want to show off, you can easily host them for free. The best place and the best way to do this will vary over time, so check the [book's website](#) for the most up-to-date recommendations. As we're writing this book in early 2020 the simplest (and free!) approach is to use [Binder](#). To publish your web app on Binder, you follow these steps:

少なくともアプリケーションの初期プロトタイプや、自慢したい趣味のプロジェクトについては、簡単に無料でホスティングすることができます。最適な場所や方法は時代によって変わるので、この本のウェブサイトで最新の推奨事項を確認してください。私たちがこの本を書いているのは 2020 年初頭なので、最もシンプルな（そして無料の！）アプローチは Binder を使うことです。Binder で Web アプリを公開するには、以下のステップを踏みます：

1. Add your notebook to a [GitHub repository](#).

ノートブックを GitHub のリポジトリに追加する。

2. Paste the URL of that repo into Binder's URL, as shown in <>.

Binder の URL に、<>のようにそのレポの URL を貼り付けます。

3. Change the File dropdown to instead select URL.

File のドロップダウンを変更し、代わりに URL を選択します。

4. In the "URL to open" field, enter /voila/render/name.ipynb (replacing name with the name of for your notebook).

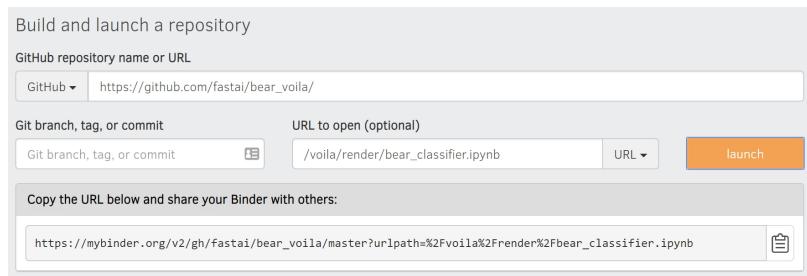
URL to open の欄に、/voila/render/name.ipynb（name はノートブックの名前に置き換えてください）と入力します。

5. Click the clickboard button at the bottom right to copy the URL and paste it somewhere safe.

右下のクリックボードボタンをクリックして URL をコピーし、安全な場所に貼り付けてください。

6. Click Launch.

Launch をクリックします。



The first time you do this, Binder will take around 5 minutes to build your site. Behind the scenes, it is finding a virtual machine that can run your app, allocating storage, collecting the files needed for Jupyter, for your notebook, and for presenting your notebook as a web application.

最初にこれを行うと、Binderはあなたのサイトを構築するために5分程度かかります。舞台裏では、アプリを実行できる仮想マシンを見つけ、ストレージを割り当て、Jupyterやノートブックに必要なファイルを収集し、ノートブックをWebアプリケーションとして表示するための作業を行っています。

Finally, once it has started the app running, it will navigate your browser to your new web app. You can share the URL you copied to allow others to access your app as well.

最後に、アプリの実行を開始すると、ブラウザを新しいウェブアプリにナビゲートします。コピーしたURLを共有することで、他の人があなたのアプリにアクセスできるようにすることができます。

For other (both free and paid) options for deploying your web app, be sure to take a look at the [book's website](#).

Webアプリケーションをデプロイするための他のオプション（無料と有料の両方）については、この本のウェブサイトを参照してください。

You may well want to deploy your application onto mobile devices, or edge devices such as a Raspberry Pi. There are a lot of libraries and frameworks that allow you to integrate a model directly into a mobile application. However, these approaches tend to require a lot of extra steps and boilerplate, and do not always support all the PyTorch and fastai layers that your model might use. In addition, the work you do will depend on what kind of mobile devices you are targeting for deployment—you might need to do some work to run on iOS devices, different work to run on newer Android devices, different work for older Android devices, etc. Instead, we recommend wherever possible that you deploy the model itself to a server, and have your mobile or edge application connect to it as a web service. アプリケーションをモバイルデバイスや Raspberry Pi のようなエッジデバイスにデプロイしたいと思うかもしれません。モバイルアプリケーションにモデルを直接組み込むことができるライブラリやフレームワークがたくさんあります。しかし、これらのアプローチは、多くの余分なステップや定型文を必要とする傾向があり、モデルが使用する可能性のあるすべての PyTorch や fastai レイヤーを常にサポートしているわけではありません。さらに、iOS デバイスで動作させるために必要な作業、新しい Android デバイスで動作させるために必要な作業、古い Android デバイスのために必要な作業など、どのようなモバイルデバイスをターゲットとして展開するかによって、作業内容が変わってきます。その代わりに、

可能な限り、モデル自身をサーバーにデプロイし、モバイルアプリケーションやエッジアプリケーションがウェブサービスとしてそれに接続することをお勧めします。

There are quite a few upsides to this approach. The initial installation is easier, because you only have to deploy a small GUI application, which connects to the server to do all the heavy lifting. More importantly perhaps, upgrades of that core logic can happen on your server, rather than needing to be distributed to all of your users. Your server will have a lot more memory and processing capacity than most edge devices, and it is far easier to scale those resources if your model becomes more demanding. The hardware that you will have on a server is also going to be more standard and more easily supported by fastai and PyTorch, so you don't have to compile your model into a different form.

この方法には、多くの利点があります。最初のインストールは簡単で、小さな GUI アプリケーションをデプロイするだけで、サーバーに接続してすべての重要な作業を行うことができるからです。さらに重要なのは、コアロジックのアップグレードを、すべてのユーザーに配布するのではなく、サーバー上で行うことができることです。サーバーには、多くのエッジデバイスよりも多くのメモリと処理能力があり、モデルの要求が高くなった場合にリソースを拡張することがはるかに容易です。また、サーバーに搭載されるハードウェアは標準的なもので、fastai や PyTorch がより簡単にサポートできるようになっているので、モデルを別の形にコンパイルする必要がありません。

There are downsides too, of course. Your application will require a network connection, and there will be some latency each time the model is called. (It takes a while for a neural network model to run anyway, so this additional network latency may not make a big difference to your users in practice. In fact, since you can use better hardware on the server, the overall latency may even be less than if it were running locally!) Also, if your application uses sensitive data then your users may be concerned about an approach which sends that data to a remote server, so sometimes privacy considerations will mean that you need to run the model on the edge device (it may be possible to avoid this by having an *on-premise* server, such as inside a company's firewall). Managing the complexity and scaling the server can create additional overhead too, 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 (also known as *horizontal scaling*).

もちろん、デメリットもあります。アプリケーションにはネットワーク接続が必要で、モデルが呼び出されるたびに多少の待ち時間が発生します。(ニューラルネットワークのモデルを実行するには時間がかかるので、このようなネットワークの待ち時間が増えることは、実際にはユーザーにとって大きな違いにはならないかもしれません。実際、サーバーではより良いハードウェアを使用できるため、全体的なレイテンシーはローカルで実行する場合よりも少なくなる可能性があります!) また、アプリケーションが機密データを使用する場合、ユーザーはそのデータをリモートサーバーに送信するアプローチに懸念を抱く可能性があります。一方、エッジデバイス上でモデルを実行する場合、各ユーザーが自分のコンピュートリソースを持ち込むことになるので、ユーザー数の増加に伴うスケーリングが容易になります(水平スケーリングとも呼ばれる)。

A: I've had a chance to see up close how the mobile ML landscape is changing in my work. We offer an iPhone app that depends on computer vision, and for years we ran our own computer vision models in the cloud. This was the only way to do it then since those models needed significant memory and compute resources and took minutes to process inputs. This approach required building not only the models (fun!) but also the infrastructure to ensure a certain number of "compute worker machines" were absolutely always running (scary), that more machines would automatically come online if traffic increased, that there was stable storage for large inputs and outputs, that the iOS app could know and tell the user how their job was doing, etc. Nowadays Apple provides APIs for converting models to run efficiently on device and most iOS devices have dedicated ML hardware, so that's the strategy we use for our newer models. It's still not easy but in our case it's worth it, for a faster user experience and to worry less about servers. What works for you will depend, realistically, on the user experience you're trying to create and what you personally find is easy to do. If you really know how to run servers, do it. If you really know how to build native mobile apps, do that. There are many roads up the hill.

A: 私は仕事柄、モバイル ML がどのように変化しているかを間近で見る機会があります。私たちはコンピュータビジョンに依存する iPhone アプリを提供していますが、何年もの間、私たち自身のコンピュータビジョンモデルをクラウド上で実行していました。なぜなら、これらのモデルはかなりのメモリと計算資源を必要とし、入力を処理するのに数分かかるからです。このアプローチでは、モデル（楽しい！）だけでなく、一定数の「コンピュートワーカーマシン」が絶対に常に稼働していること（怖い）、トラフィックが増加すれば自動的にマシンが増えること、大きな入力と出力に対して安定したストレージがあること、iOS アプリが自分の仕事がどうなっているかを知ってユーザーに伝えられることなど、インフラの構築も必要でした。現在では、Apple がモデルをデバイス上で効率的に実行できるように変換するための API を提供し、ほとんどの iOS デバイスには ML 専用のハードウェアが搭載されています。それでもまだ簡単ではありませんが、私たちの場合、より速いユーザ体験とサーバーの心配を減らすために、その価値はあります。何が効果的かは、現実的には、作ろうとしているユーザ体験と、個人的にやりやすいと思うことによるでしょう。もし、あなたが本当にサーバーを動かす方法を知っているなら、それを実行すればいい。ネイティブのモバイルアプリの作り方を本当に知っているのなら、それをやればいい。坂の上にはたくさんの道があります。

Overall, we'd recommend using a simple CPU-based server approach where possible, for as long as you can get away with it. If you're lucky enough to have a very successful application, then you'll be able to justify the investment in more complex deployment approaches at that time.

全体として、可能な限り、シンプルな CPU ベースのサーバーアプローチを使用することをお勧めします。幸運にもアプリケーションが大成功を収めれば、その時点でより複雑なデプロイメントアプローチへの投資を正当化することができるでしょう。

Congratulations, you have successfully built a deep learning model and deployed it! Now is a good time to take a pause and think about what could go wrong.

おめでとうございます、あなたはディープラーニングモデルを構築し、それをデプロイすることに成功しました！今、一旦立ち止まって、何が間違っているのかを考える良い機会です。

## How to Avoid Disaster

### 災害を回避する方法

In practice, a deep learning model will be just one piece of a much bigger system. As we discussed at the start of this chapter, a data product requires thinking about the entire end-to-end process, from conception to use in production. In this book, we can't hope to cover all the complexity of managing deployed data products, such as managing multiple versions of models, A/B testing, canarying, refreshing the data (should we just grow and grow our datasets all the time, or should we regularly remove some of the old data?), handling data labeling, monitoring all this, detecting model rot, and so forth. In this section we will give an overview of some of the most important issues to consider; for a more detailed discussion of deployment issues we refer to you to the excellent [Building Machine Learning Powered Applications](#) by Emmanuel Ameisen (O'Reilly)

実際には、ディープラーニングモデルは、より大きなシステムの 1 ピースに過ぎないでしょう。本章の冒頭で述べたように、データプロダクトは、構想から本番での使用まで、エンドツーエンドのプロセス全体について考える必要があります。この本では、複数のバージョンのモデルの管理、A/B テスト、カナリアリング、データのリフレッシュ（データセットを常に大きくしていくべきか、古いデータの一部を定期的に削除すべきか）、データレベルの取り扱い、これらすべての監視、モデルの腐敗の検出など、展開されたデータ製品の管理の複雑さをすべてカバーできるわけではありません。このセクションでは、考慮すべき最も重要な問題の概要を説明します。デプロイメントに関するより詳細な議論については、Emmanuel Ameisen による優れた [Building Machine Learning Powered Applications](#) (O'Reilly) を参照してください。

One of the biggest issues to consider is that understanding and testing the behavior of a deep learning model is much more difficult than with most other code you write. With normal software development you can analyze the exact steps that the software is taking, and carefully study which of these steps match the desired behavior that you are trying to create. But with a neural network the behavior emerges from the model's attempt to match the training data, rather than being exactly defined.

ディープラーニングモデルの挙動を理解しテストすることは、他の多くのコードを書く場合よりもはるかに難しいということが、考慮すべき最大の問題の1つです。通常のソフトウェア開発では、ソフトウェアが取っている正確なステップを分析し、これらのステップのどれが、作成しようとしている望ましい動作と一致するかを慎重に検討することができます。しかし、ニューラルネットワークの場合、動作は正確に定義されるのではなく、モデルが学習データと一致させようとすることで出現するのです。

This can result in disaster! For instance, let's say we really were rolling out a bear detection system that will be attached to video cameras around campsites in national parks, and will warn campers of incoming bears. If we used a model trained with the dataset we downloaded there would be all kinds of problems in practice, such as:

その結果、災難に見舞われることもあります！例えば、国立公園のキャンプ場周辺のビデオカメラに取り付けて、熊の侵入をキャンプ客に警告する熊検知システムを開発するとしましょう。もし、ダウンロードしたデータセットで学習させたモデルを使うとしたら、実際には以下のような様々な問題が発生することでしょう：

- Working with video data instead of images  
画像ではなく、動画データを扱う
- Handling nighttime images, which may not appear in this dataset  
本データセットに掲載されていない可能性のある夜間画像の取り扱いについて
- Dealing with low-resolution camera images  
低解像度のカメラ画像への対応
- Ensuring results are returned fast enough to be useful in practice  
実務に役立つ十分な速さで結果が返ってくるようにすること
- 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)  
ネットにアップされる写真ではあまり見かけない位置にいるクマを認識する（後ろから、茂みに覆われている、カメラからかなり離れている、など）。

A big part of the issue is that the kinds of photos that people are most likely to upload to the internet are the kinds of photos that do a good job of clearly and artistically displaying their subject matter—which isn't the kind of input this system is going to be getting. So, we may need to do a lot of our own data collection and labelling to create a useful system.

インターネットにアップロードされる写真は、被写体をはっきりと芸術的に表現しているものが多いので、このシステムにはそのような写真は入ってこないというのが大きな問題です。ですから、便利なシステムを作るためには、私たち自身が多くデータを収集し、ラベリングする必要があるかもしれません。

This is just one example of the more general problem of *out-of-domain* data. That is to say, there may be data that our model sees in production which is very different to what it saw during training. There isn't really a complete technical solution to this problem; instead, we have to be careful about our approach to rolling out the technology.

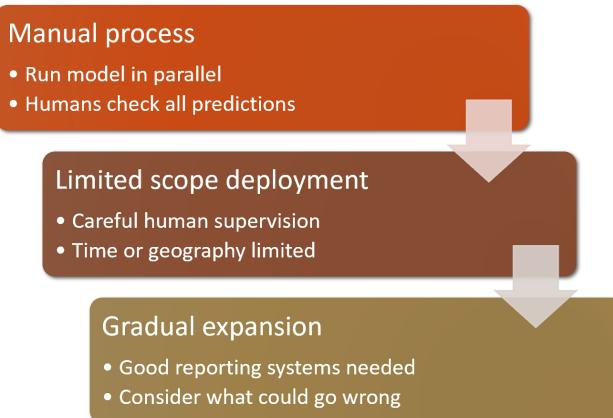
これは、より一般的な「領域外データ」の問題の一例です。つまり、モデルが生産現場で見るデータは、トレーニング中に見たものとは全く異なるということです。この問題に対する完全な技術的解決策はないのですが、その代わりに、技術を導入する際のアプローチに注意しなければなりません。

There are other reasons we need to be careful too. One very common problem is *domain shift*, where the type of data that our 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.

注意しなければならない理由は他にもあります。よくある問題のひとつに、ドメインシフトというものがあります。例えば、保険会社が価格設定やリスクのアルゴリズムの一部としてディープラーニングモデルを使用することができますが、時間の経過とともに、その会社が集める顧客の種類やリスクの種類が大きく変化し、元のトレーニングデータはもはや適切ではなくなってしまうかもしれません。

Out-of-domain data and domain shift are examples of a larger problem: that you can never fully understand the entire behaviour of your neural network. They have far too many parameters to be able to analytically understand all of their possible behaviors. This is the natural downside of their best feature—their flexibility, which enables them to solve complex problems where we may not even be able to fully specify our preferred solution approaches. The good news, however, is that there are ways to mitigate these risks using a carefully thought-out process. The details of this will vary depending on the details of the problem you are solving, but we will attempt to lay out here a high-level approach, summarized in <>deploy\_process>, which we hope will provide useful guidance.

ドメイン外データとドメインシフトは、より大きな問題の一例です。つまり、ニューラルネットワークの動作全体を完全に理解することはできません。ニューラルネットワークにはあまりにも多くのパラメータがあり、可能なすべての動作を分析的に理解することはできません。これは、ニューラルネットワークの最大の特徴である柔軟性がもたらす自然なマイナス面であり、そのおかげで、私たちが望ましい解決方法を完全に特定することさえできないような複雑な問題を解決することができます。しかし、このようなリスクを軽減するために、入念に練られたプロセスが存在します。その詳細は、解決しようとする問題の詳細によって異なりますが、ここでは、<>に要約される高レベルのアプローチを整理することを試み、有用な指針になることを願っています。



Where possible, the first step is to use an entirely manual process, with your deep learning model approach running in parallel but not being used directly to drive any actions. The humans involved in the manual process should look at the deep learning outputs and check whether they make sense. For instance, with our bear classifier a park ranger could have a screen displaying video feeds from all the cameras, with any possible bear sightings simply highlighted in red. The park ranger would still be expected to be just as alert as before the model was deployed: the model is simply helping to check for problems at this point.

可能であれば、最初のステップは完全に手動プロセスを使用することで、ディープラーニングモデルアプローチは並行して実行されますが、アクションを駆動するために直接使用されることはありません。手動プロセスに関わる人間は、ディープラーニングの出力を見て、それが意味を持つかどうかをチェックする必要があります。例えば、熊の分類を行う場合、パークレンジャーがすべてのカメラの映像をスクリーンに映し出し、熊が目撃された可能性がある場合は赤色でハイライト表示することができます。パークレンジャーは、モデルを導入する前と同じように警戒することが期待されますが、モデルはこの時点で問題をチェックするのに役立つだけです。

The second step is to try to limit the scope of the model, and have it carefully supervised by people. For instance, do a small geographically and time-constrained trial of the model-driven approach. Rather than rolling our bear classifier out in every national park throughout the country, we could pick a single observation post, for a one-week period, and have a park ranger check each alert before it goes out.

第二のステップは、モデルの範囲を限定し、人の手で注意深く監視してみることです。例えば、地理的・時間的に制約のある中で、モデル駆動型アプローチの小規模なトライアルを実施します。全国の国立公園でクマ分類器を展開するのではなく、1つの観測所を選び、1週間かけて、パークレンジャーに各アラートをチェックしてもらってから発信するのです。

Then, gradually increase the scope of your rollout. As you do so, ensure that you have really good reporting systems in place, to make sure that you are aware of any significant changes to the actions being taken compared to your manual process. For instance, if the number of bear alerts doubles or halves after rollout of the new system in some location, we should be very concerned. Try to think about all the ways in which your system could go wrong, and then think about what measure or report or picture could reflect that problem, and ensure that your regular reporting includes that information.

そして、徐々に範囲を広げていくのです。その際、報告システムをきちんと整備し、手動で行っていた作業と比較して、実施されるアクションに大きな変化があったことを確認するようにします。例えば、ある場所で新システムを導入した後、ベアアラートの数が2倍または半分になった場合、私たちは非常に心配になるはずです。システムがうまくいかない可能性のあるあらゆる方法を考え、その問題を反映できる指標や報告書、写真は何かを考え、定期的な報告書にその情報が含まれるようにするのです。

J: I started a company 20 years ago called *Optimal Decisions* that used machine learning and optimization to help giant insurance companies set their pricing, impacting

tens of billions of dollars of risks. We used the approaches described here to manage the potential downsides of something going wrong. Also, before we worked with our clients to put anything in production, we tried to simulate the impact by testing the end-to-end system on their previous year's data. It was always quite a nerve-wracking process, putting these new algorithms into production, but every rollout was successful.

J: 私は 20 年前に Optimal Decisions という会社を立ち上げ、機械学習と最適化を用いて巨大な保険会社の価格設定を支援し、数百億ドルのリスクに影響を与えるようにしました。私たちは、ここで紹介したようなアプローチで、何かがうまくいかなかったときに起こりうるデメリットを管理していました。また、クライアントと協力して何かを本番稼働させる前に、エンドツーエンドのシステムをクライアントの前年度のデータでテストすることで、影響をシミュレーションするようにしました。新しいアルゴリズムを本番に導入するのは、常に神経をすり減らす作業でしたが、すべての展開が成功しました。

## Unforeseen Consequences and Feedback Loops

### 不測の事態とフィードバックループ

One of the biggest challenges in rolling out a model is that your model may change the behaviour of the system it is a part of. For instance, consider a "predictive policing" algorithm that predicts more crime in certain neighborhoods, causing more police officers to be sent to those neighborhoods, which can result in more crimes being recorded in those neighborhoods, and so on. In the Royal Statistical Society paper "[To Predict and Serve?](#)", Kristian Lum and William Isaac observe that: "predictive policing is aptly named: it is predicting future policing, not future crime." モデルを展開する際の最大の課題の 1 つは、モデルがその一部であるシステムの挙動を変えてしまう可能性があることです。例えば、「予測警察」アルゴリズムが、特定の地域で犯罪が増えると予測し、その地域に多くの警察官を派遣することになり、その結果、その地域でより多くの犯罪が記録される、といったようなことを考えてみましょう。王立統計学会の論文 "[To Predict and Serve?](#)" の中で、Kristian Lum と William Isaac は次のように観察している： 「予測的取り締まりは、将来の犯罪を予測するのではなく、将来の取り締まりを予測するという、適切な名称である。

Part of the issue in this case is that in the presence of bias (which we'll discuss in depth in the next chapter), *feedback loops* can result in negative implications of that bias getting worse and worse. For instance, there are concerns that this is already happening in the US, where there is significant bias in arrest rates on racial grounds. [According to the ACLU](#), "despite roughly equal usage rates, Blacks are 3.73 times more likely than whites to be arrested for marijuana." The impact of this bias, along with the rollout of predictive policing algorithms in many parts of the US, led Bär Williams to [write in the New York Times](#): "The same technology that's the source of so much excitement in my career is being used in law enforcement in ways that could mean that in the coming

years, my son, who is 7 now, is more likely to be profiled or arrested—or worse—for no reason other than his race and where we live."

この場合の問題の一つは、バイアス（次の章で詳しく説明します）が存在する場合、ファイードバックループによって、そのバイアスがもたらすネガティブな影響がますます悪化する可能性があることです。例えば、人種を理由とした検挙率に大きな偏りがあるアメリカでは、すでにこのようなことが起きていることが懸念されている。ACLUによると、"使用率がほぼ同じにもかかわらず、黒人は白人に比べて大麻で逮捕される確率が 3.73 倍も高い"という。この偏りの影響は、米国の多くの地域で予測的な取り締まりアルゴリズムの展開とともに、Bári Williams が New York Times に書いたようになりました： 「私のキャリアにおいて大きな興奮の源となっている同じテクノロジーが、法執行機関において、今後数年のうちに、7歳になる私の息子が、人種や住んでいる場所以外の理由なく、プロファイリングや逮捕、あるいはそれ以上の事態に陥る可能性があるという意味で使われているのです」。

A helpful exercise prior to rolling out a significant machine learning system is to consider this question: "What would happen if it went really, really well?" In other words, what if the predictive power was extremely high, and its ability to influence behavior was extremely significant? In that case, who would be most impacted? What would the most extreme results potentially look like? How would you know what was really going on?

重要な機械学習システムを展開する前に、この質問を考えてみるのも有効な手段です："本当に、本当にうまくいったらどうなるのか？"。つまり、予測力が極めて高く、行動に影響を与える能力が極めて大きいとしたらどうだろうか。その場合、誰が最も影響を受けるのだろうか？最も極端な結果はどのようなものになる可能性があるのだろうか。何が起きているのか、どうやって知ることができるのだろう？

Such a thought exercise might help you to construct a more careful rollout plan, with ongoing monitoring systems and human oversight. Of course, human oversight isn't useful if it isn't listened to, so make sure that there are reliable and resilient communication channels so that the right people will be aware of issues, and will have the power to fix them.

このような思考訓練は、継続的な監視システムと人間の監視を伴う、より慎重な展開計画を構築するのに役立つかかもしれません。もちろん、人による監視は、それに耳を傾けなければ意味がありません。そのため、信頼性が高く弾力性のあるコミュニケーションチャネルを確保し、適切な人が問題を認識し、それを解決する力を持つようにしましょう。

## Get Writing!

書いてみる！

One of the things our students have found most helpful to solidify their understanding of this material is to write it down. There is no better test of your understanding of a topic than attempting to teach it to somebody else. This is helpful even if you never show your writing to anybody—but it's even better if you share it! So we recommend that, if you haven't already, you start a blog. Now that you've completed Chapter 2 and

have learned how to train and deploy models, you're well placed to write your first blog post about your deep learning journey. What's surprised you? What opportunities do you see for deep learning in your field? What obstacles do you see?

私たちの生徒が、この教材の理解を深めるために最も役に立ったことのひとつは、書き出すことです。誰かに教えることほど、そのトピックの理解度を試すのに最適なことはありません。書いたものを誰かに見せることはなくとも、共有することで、より理解を深めることができます！ですから、まだの方は、ブログを始めることをお勧めします。第2章を終え、モデルの訓練と配備の方法を学んだあなたは、ディープラーニングの旅について最初のブログ記事を書くのに適しています。驚いたことは何ですか？あなたの分野では、深層学習のどんな機会があると思いますか？また、どのような障害があると思いますか？

Rachel Thomas, cofounder of fast.ai, wrote in the article "[Why You \(Yes, You\) Should Blog](#)":

fast.ai の共同創業者であるレイチェル・トマスは、「なぜあなたは（Yes, You）ブログを書くべきなのか」という記事で、こう書いています：

asciidoc

---

The top advice I would give my younger self would be to start blogging sooner. Here are some reasons to blog:

私が若い頃の自分にしたいアドバイスは、もっと早くブログを始めることです。ブログを書く理由は以下の通りです：

\* It's like a resume, only better. I know of a few people who have had blog posts lead to job offers!

\* 履歴書のようなもので、より良いものです。私は、ブログの投稿が仕事のオファーにつながった人を何人か知っています！

\* Helps you learn. Organizing knowledge always helps me synthesize my own ideas. One of the tests of whether you understand something is whether you can explain it to someone else. A blog post is a great way to do that.

\* 学ぶことができる。知識を整理することは、常に自分の考えをまとめるのに役立ちます。自分が何かを理解しているかどうかのテストのひとつは、それを誰かに説明できるかどうかです。ブログ記事は、そのための素晴らしい方法です。

\* I've gotten invitations to conferences and invitations to speak from my blog posts. I was invited to the TensorFlow Dev Summit (which was awesome!) for writing a blog post about how I don't like TensorFlow.

\* 私は、自分のブログ記事からカンファレンスへの招待や講演の依頼を受けたことがあります。TensorFlow が好きではないというブログ記事を書いたことで、TensorFlow Dev Summit に招待されました（最高でした！）。

\* Meet new people. I've met several people who have responded to blog posts I wrote.

\* 新しい人に会う。私が書いたブログ記事に反応してくれた人たちに何人か会いました。

\* Saves time. Any time you answer a question multiple times through email, you should turn it into a blog post, which makes it easier for you to share the next time someone asks.

\* 時間を節約することができます。メールを通じて何度も質問に答えるときは、それをブログ記事にしておくと、次に誰かが質問してきたときに共有しやすくなります。

---

Perhaps her most important tip is this:

彼女の最も重要なヒントはこれかもしれません：

: You are best positioned to help people one step behind you. The material is still fresh in your mind. Many experts have forgotten what it was like to be a beginner (or an intermediate) and have forgotten why the topic is hard to understand when you first hear it. The context of your particular background, your particular style, and your knowledge level will give a different twist to what you're writing about.

: 自分の一歩後ろにいる人を助けるのに最も適しているのは、あなたです。あなたの頭の中には、まだ教材が新鮮に残っているのです。多くの専門家は、初心者の頃（あるいは中級者）のことを忘れ、初めて聞く話題がなぜ理解しにくいのかを忘れてしまっています。あなたの特別な背景、あなたの特別な

スタイル、あなたの知識レベルという文脈は、あなたが書いていることに違ったひねりを与えてくれます。

We've provided full details on how to set up a blog in <>. If you don't have a blog already, take a look at that now, because we've got a really great approach set up for you to start blogging for free, with no ads—and you can even use Jupyter Notebook!

ブログの開設方法については、<>で詳しく解説しています。もしあなたがまだブログを持っていないのであれば、今すぐ見てみてください！

## Questionnaire

アンケート

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

学習データの構造やスタイルの違いにより、熊の分類モデルが本番でうまく機能しない可能性がある例を挙げてください。

2. Where do text models currently have a major deficiency?

テキストモデルには現在、どこに大きな欠陥があるのか？

3. What are possible negative societal implications of text generation models?

テキスト生成モデルの社会的な負の影響として考えられるものは何か？

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

モデルが間違いを犯す可能性があり、その間違いが有害である可能性がある状況において、プロセスを自動化する良い代替案は何か？

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

ディープラーニングが特に得意とする表形式のデータにはどのようなものがありますか？

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

推薦システムにディープラーニングモデルを直接使用することの重要な欠点は何ですか？

7. What are the steps of the Drivetrain Approach?

Drivetrain Approach のステップとは？

8. How do the steps of the Drivetrain Approach map to a recommendation system?

Drivetrain Approach のステップは、どのようにレコメンデーションシステムにマッピングされるのでしょうか？

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

キュレーションしたデータを使って画像認識モデルを作成し、Web 上に展開する。

10. What is DataLoaders?

DataLoaders とは何ですか？

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

DataLoaders を作成するために fastai に伝える必要がある 4 つのことは？

12. What does the splitter parameter to DataBlock do?

DataBlock の splitter パラメータは何をするのか？

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

ランダムな分割で常に同じ検証セットが得られるようにするにはどうすればよいですか？

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

独立変数と従属変数を表すのによく使われる文字は何ですか？

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

クロップ、パッド、スクイッシュのリサイズアプローチの違いは何ですか？どのような場合に、他の方法より 1 つを選択するのでしょうか？

16. What is data augmentation? Why is it needed?

データ補強とは何ですか？なぜ必要なのでしょうか？

17. What is the difference between item\_tfms and batch\_tfms?

item\_tfms と batch\_tfms の違いは何ですか？

18. What is a confusion matrix?

コンフュージョンマトリックスとは何ですか？

19. What does export save?

エクスポートセーブとは何ですか？

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

学習ではなく、予測値を得るためにモデルを使用することを何と呼ぶか？

21. What are IPython widgets?

IPython の ウィジエットとは何ですか？

22. When might you want to use CPU for deployment? When might GPU be better?

デプロイメントに CPU を使いたいのはどんな時ですか？ GPU の方がいいのはどんなときですか？

23. 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?

アプリを電話や PC などのクライアント（またはエッジ）デバイスではなく、サーバーにデプロイすることのデメリットは何ですか？

24. What are three examples of problems that could occur when rolling out a bear warning system in practice?

熊出没注意システムを実際に展開する際に起こりうる問題の例を 3 つ教えてください。

25. What is "out-of-domain data"?

ドメイン外データ」とは何ですか？

26. What is "domain shift"?

ドメインシフト」とは何ですか？

27. What are the three steps in the deployment process?

展開の 3 つのステップとは？

## Further Research

さらなる研究

1. Consider how the Drivetrain Approach maps to a project or problem you're interested in.

ドライブトレインアプローチが、あなたの興味のあるプロジェクトや問題にどのように対応するか考えてみてください。

2. When might it be best to avoid certain types of data augmentation?

ある種のデータ補強を避けた方が良いのはどんな場合か？

3. For a project you're interested in applying deep learning to, consider the thought experiment "What would happen if it went really, really well?"

ディープラーニングを適用することに興味があるプロジェクトについて、"本当に、本当にうまくいったらどうなるか？"という思考実験について考えてみる。

4. Start a blog, and write your first blog post. For instance, write about what you think deep learning might be useful for in a domain you're interested in.

ブログを始めて、最初のブログ記事を書いてみてください。例えば、あなたが興味のある領域で、深層学習が役に立つと思うことについて書いてみてください。