Deep Learning for Image Analysis

2019 Edition
Deep Learning for Image Analysis

2019 Edition
To the future—
There has been an explosion of industry interest in deep learning in the last five years, in particular in image analysis (IA), making this area of machine learning even more relevant to enabling operations and creating differentiation in business activities today. This interest has fed on both academic and corporate research advancements, fueled in part by the remarkable public advances in fields such as medical imaging, autonomous vehicles, news and media (including manipulation), and art.

With this update to our 2016 Deep Learning: Image Analysis report, we revisit the state of the art in deep learning for IA, exploring the evolving landscape of IA algorithms and open source IA toolkits and examining the emerging concrete standards of practice in development. This report aims to offer the technical background and practical guidance data scientists and business stakeholders need to develop image analysis products that are beneficial as well as ethical. With our accompanying prototype, ConvNet Playground, we demonstrate how deep learning models can be applied to the task of semantic image search and provide tools that help build intuition on how the models work.
What Can Deep Learning Do?

Image analysis is just one area where deep learning has had a significant impact in recent years, and is poised to do more. When tasks are appropriately framed as learning problems, we’ve witnessed leading-edge results: we can now infer depth information and reconstruct 3D maps from 2D images without additional context or metadata, giving new potential to urban planning as well as entertainment experiences; we can perform pixel-level separation of objects in images and video, with applications ranging from public safety to medical robotics; we can automatically identify defects in manufactured items, reducing costs associated with quality assurance (QA) processes; and we can perform “super resolution” on photo images (upscaling an image up to 10x), reconstructing and filling in information that would be omitted and lost using a standard digital zoom and thus sharpening the result.

The velocity of change in this field means that tools for the design, implementation, and deployment of deep learning models that worked well even in 2016 seem relatively basic today. To address this, ongoing community efforts have led to the creation of libraries that abstract the most important blocks of deep learning into usable methods, and the emergence of standards for their use in real-world problem solving.

The Risks

We also bring a word of caution: reliance on commercial solutions may have an impact on the uniqueness of your data product. With image analysis increasingly being used in diverse industrial applications, inexperienced IT/data teams and business leaders are increasingly at risk of overusing
commoditized solutions without understanding their function or really understanding what makes the most sense for their needs. Incorporation of the tooling has become easier, but it’s not yet trivial to fold these products into diverse workflows directly off the shelf—and the impression of "ease" creates potential blind spots as to how these solutions could fail, how they may enable competitors to improve their results based on your data, or how they may cause you to miss out on opportunities to build more differentiated or advantageous capabilities that meet your unique business, safety, and ethical requirements. The perception of ease of use may also open users up to potentially large financial costs associated with server time for training and testing large deep learning models.

Although the deep learning libraries of frameworks like PyTorch and TensorFlow have advanced massively, constructing models—particularly using the newest and best-performing deep learning architectures—is by no means plug-and-play. In addition, many of the novel architectures exist only in the form of scientific papers, openly available to support model reconstruction but not yet available as downloadable packages.

The aim of this report is to help you understand the landscape of available architectures and tools, tradeoffs between these architectures, and approaches for debugging and understanding IA models, so that you can navigate these challenges effectively. We will also explore the ethical issues associated with deep learning for IA.
Image Analysis Use Cases

To make the abstract algorithm and software discussions more concrete, consider the following examples of fictional companies with realistic image analysis use cases.

Chipset Inspection

ChypCon Industries is a manufacturer of chipsets for embedded devices that aims to make its “Chypsets” ubiquitous across millions of Internet of Things (IoT) products. The company’s production objectives are twofold: 1) build and maintain a reputation as reliable makers of very high quality chips, and 2) explore opportunities for efficiency via reliable automation. As such, ChypCon has instituted a multi-stage quality control process, including a visual inspection of the chips for any observed defects that cannot be identified by other tests. However, the visual inspection creates a production bottleneck, given the amount of time required and the few engineers available (most of whom have other responsibilities as well). Recently, fatigued engineers and human error have led to several false positives. Given that quality control is a sensitive aspect of ChypCon’s business, such false positives can incur huge financial costs.

ChypCon is looking for a solution that allows it to automate the visual inspection step in its QA process. The system should have very high accuracy, but does not need to have high throughput (there is a cap on the amount of chips available for inspection each hour). It should also be explainable (i.e., there should be information that helps ChypCon understand when and how the model might fail).

To achieve this, ChypCon can use an image segmentation
model (see 3.1.2 Image Segmentation) to predict which pixels in images of its Chypsets are likely to contain defects. The defects identified by the model can then be confirmed or rejected by an engineer. To ensure high accuracy (see 3.3 Picking a Good Model: Accuracy vs. Latency), ChypCon should consider the use of a high-accuracy model and invest in the acquisition of a large training dataset.

**The Blazing Mine**

Ozzie Mines is in the iron ore mining business in Western Australia, where temperatures during the summer are some of the highest in the world. Temperatures are so extreme that it is difficult to find workers willing to brave the elements. To cope with this shortage in manpower, Ozzie Mines has
codeveloped and deployed an extensive fleet of robots to automate some of the tasks associated with mining. These include driverless trains for shipping products between mines and distribution centers, as well as trucks, drills, and loaders that can be controlled remotely.

Currently, all sensor data (2 terabytes of data per day) collected from the fleet of robots is sent to remote servers for processing. Maintaining this data is expensive. Furthermore, as more robots are added to the fleet, this will introduce additional costs in bandwidth for communication, as well as data storage costs. The IT team at Ozzie Mines are interested in a solution that allows on-device processing, removing or minimizing the need for data transfer and storage. They have identified autonomous driving as the first area of investigation. Given that decisions regarding navigation and site

![Figure 1.2 Deep learning models allow for the implementation of autonomous driving capabilities that can help autonomous vehicles transport products in regions challenging for human habitation.](image-url)
safety need to be made in real time, a strong requirement of the solution is that the system exhibits low latency within some acceptable error limit.

To achieve these goals, Ozzie Mines can explore a combination of object detection and object segmentation models (see 3 Applied Image Analysis: Tasks and Models) to accurately create a model of the driving scene; results from the model can then be integrated into the navigation control system. To meet their latency objectives, they can focus on model architectures that maintain good accuracy while remaining small in size (fewer parameters) (see 3.3 Picking a Good Model: Accuracy vs. Latency).

The Intelligent Fashioneer

MazingCorp has been in the e-commerce business for over two decades, helping small businesses set up online storefronts. It provides multiple services to businesses—including logistics support for product delivery, payment processing, content hosting, and brand management—and earns a commission on each purchase based on client agreements.

Each day, MazingCorp receives thousands of images that are uploaded by its customers to showcase the products in their storefronts. In the e-commerce business, research has shown that the quality of the images uploaded for a product can be just as important as reviews for encouraging a purchase. Thus, the MazingCorp team are interested in a solution that helps them automatically quantify the visual quality of each product image. This information can then be used to optimize how images are displayed to visitors, so as to improve
the odds of purchase. Ideally, they would like to scale this to their entire platform, spanning millions of images.

To achieve these objectives, MazingCorp can leverage a convolutional neural network that has been trained to predict visual quality given the content (pixels) of an image. Typically, this training process will require a dataset of images annotated with a visual quality score and may be initialized with information learned during other standard training processes, such as image classification (see 3 Applied Image Analysis: Tasks and Models).

**FIGURE 1.3** Deep learning models can be used to automatically assess the quality of images. This information can be leveraged for optimizing the placement of images in e-commerce storefronts, which in turn can help encourage purchasing.
Deep Learning for Image Analysis: A Brief Background

Deep learning applications for image analysis have a long and rich history, and a majority of the advances in this field have been enabled by convolutional neural networks (CNNs, or ConvNets). CNNs are a class of deep neural networks composed of basic building blocks such as convolutional layers, pooling layers, and fully connected layers. The LeNet model,\(^1\) a classic CNN architecture introduced in 1998 by Yann LeCun et al., is widely credited as one of the earliest success stories for deep learning applied to solving commercial image-related problems. This model had seven layers (alternating convolutional layers and pooling layers, and a single fully connected layer) and was widely deployed for recognizing handwritten digits on bank cheques in the United States.

Over the last decade, multiple deep neural network architecture variants have emerged to tackle image processing tasks. Much of this work has been done within the context of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC),\(^2\) an annual contest introduced in 2010 to benchmark progress in image analysis research. To participate in

---

the challenge, researchers and practitioners are invited to submit software applications that correctly classify images and detect objects in scenes based on the ImageNet dataset (composed of over 1 million images spread across 1,000 object categories).

In 2012, a contest submission named AlexNet based on convolutional neural networks achieved an error rate of 15.3%, more than 10% lower than the closest runner-up, marking neural networks as the leading approach for image analysis tasks.

Following AlexNet, the ImageNet challenge continued to be dominated by increasingly more sophisticated and complex (i.e., more layers) CNN models. Error rates for the leading models have continued to improve: in 2014 VGG16 featured an error rate of 7.3% and GoogLeNet (InceptionV1) reached 6.7%; ResNet lowered the error rate to just 3.57% in 2015, and

FIGURE 2.1 A neural network — similar to the LeNet model — for classification of vehicles in images.
in 2016 InceptionV4 boasted an error rate of just 3.08%. Two important theoretical insights enabled researchers to successfully train these complex models. The first was the use of skip connections, as seen in the ResNet model, and the second was the batch normalization technique introduced in the InceptionV2 model. More recently, there have been efforts that allow machines to automatically discover architecture hyperparameters that are fine-tuned for specific objectives (e.g., accuracy on a given dataset, number of parameters, latency on specific hardware, FLOPs, etc.). This area of research is

FIGURE 2.2 All of the top-performing models on the ImageNet classification task are convolutional neural networks. Recently, AutoML approaches have yielded the models with the best performance (accuracy and parameter efficiency).

usually referred to as neural architecture search or automatic machine learning (AutoML).

**Reusing Representations Learned by CNNs: Transfer Learning**

When we train a CNN model on a large, diverse dataset such as ImageNet, the layers within the model tend to learn filters or patterns that are relevant to the current task. For example, when trained on the task of image classification, we observe that CNNs learn hierarchical representations: early layers learn to detect simple patterns such as colors, lines, and edges, while later layers learn to detect complex patterns such as textures and parts of objects. Importantly, these learned patterns can be reused for other tasks that require an understanding of the content of images. For example, consider the use case of building a model for the task of detecting defects on surfaces. Information on how to detect lines, edges, and textures learned while training an image classification model on the ImageNet dataset (called a pretrained model) can be reused in training a new CNN for detecting defects. To implement this, it is typical to instantiate the new model with a subset of the weights and architecture of the pretrained model, and adapt these to fit the new task. This process, known as transfer learning, enables us to obtain good accuracy for image analysis tasks with relatively small datasets.

**Recommendation** For image analysis tasks that benefit from an understanding of the patterns/features within an image, and where there is limited training data, we recommend transfer learning as a standard approach. We also

5 https://ieeexplore.ieee.org/abstract/document/5288526
recommended the exploration of AutoML architectures which achieve high accuracy with less parameters.

**Hardware Developments**

Deep learning models are composed of stacks of interconnected neurons which represent units of compute operations. These operations have parameters known as *weights*, which are stored in matrices. As a model is trained, simple linear algebra operations are applied to update these weights based on the available data. It turns out that performing operations on matrices can be done more efficiently with graphics processing units (GPUs) than the traditional CPUs standard in most computers — that is, leveraging GPUs often results in faster training and inference times. This realization has inspired efforts to create specialized hardware and software libraries that accelerate linear algebra and neural network operations on various hardware platforms. For example, many citizen data scientists work on GPU-boosted “gaming” computers, while companies like our own enterprise data cloud company, Cloudera, and other cloud services providers offer the ability to spin up and allocate GPU-based virtual machines for use for machine learning and data analytics work, on demand.

Over the years, companies such as NVIDIA, AMD, and Intel have released a range of GPUs in parallel with multi-core CPU chips, initially to support compute-heavy graphics applications and gaming. However, chipmakers are now making specific efforts to design chips for neural network applications, and are creating corresponding libraries that programmers can use to access full GPU capabilities. These include the
CUDA Toolkit\textsuperscript{6} to enable deep learning on the NVIDIA GPU line of hardware devices and the OpenVINO Toolkit\textsuperscript{7} released by Intel. In a similar vein, Google has created in-house what it terms a tensor processing unit (TPU),\textsuperscript{8} an application-specific integrated circuit (ASIC) complement to the TensorFlow library. The TPU seeks to trade off the computational precision required of modern graphics for an emphasis on speed in core calculations; unfortunately, at this time it is accessible only through Google’s own cloud offering.

**The Evolving Field**

Today, the emergence of deep learning for image analysis as a standardized tool for modern data science is being driven in part by the availability of open source deep learning libraries (see \textsuperscript{6}Deep Learning in Industry Today) and educational materials (datasets, tutorials, MOOC courses and sample code on GitHub, etc.). We recommend learning resources such as CS231n: Convolutional Neural Networks for Visual Recognition from Stanford University, Fast.ai, and Deep Learning Specialization on Coursera, as well as the offerings on pay-as-you-go and subscription course sites like Udemy, Lynda, Cloudera, and MathWorks.

Given the availability of these resources, we see opportunities for differentiation in the use of custom datasets and custom model architectures fine-tuned for a given business problem domain. In the next chapter, we discuss a set of

\begin{itemize}
  \item \textsuperscript{6} https://developer.nvidia.com/cuda-toolkit
  \item \textsuperscript{7} https://software.intel.com/en-us/openvino-toolkit
  \item \textsuperscript{8} https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip
\end{itemize}
image analysis tasks, relevant architectures that can serve as a starting point for customization, and tools for understanding and debugging models.
The set of image analysis tasks that are amenable to deep learning methods can be broadly categorized based on the approach used to train the underlying model. For each of these categories, we identify influential models that have been used to address the corresponding tasks and provide pointers to open source implementations where available. Finally, we offer guidance on how to select a model with consideration of the tradeoffs between accuracy and latency.

**Supervised Learning Tasks**

Supervised learning requires the existence of a dataset of labeled or annotated examples. These examples serve as the “ground truth” signal used in training a model. For example, in the case of an image classification task, an image of a dog is labeled with metadata such as its general class (dog) and its breed (e.g., Pomeranian). During model training, these labels are used as signals for adjusting the model’s parameters — i.e., adjusting neuron weights to make sure that the model output classifies that image as a dog/Pomeranian. Several deep learning for image analysis tasks fall under this domain.

**Image Classification**

Image classification refers to the task of identifying the
primary subject (e.g., a person, animal, scene, building) within an image. Given an input image, the model is tasked with assigning the image to one of several discrete class labels on which it has been trained. As an example, a model trained on the ImageNet dataset is able to classify a new image (with some level of probability) as belonging to 1 of the 1,000 class labels in that dataset. For the task of image classification, large convolutional neural network architectures provide the best results (as evidenced by their performance in the ImageNet challenge).

It is important to note that model architectures trained on an image classification task are frequently reused for other tasks (as mentioned in Chapter 2, this is known as transfer learning). For example, it is common for object detection models to rely on features extracted using a model pretrained on the ImageNet classification task.

**Image Segmentation**

Image segmentation (also known as semantic segmentation) involves predicting the class labels for pixels or groups of pixels within an image. That is, rather than assigning one or a few labels to the entire image, image segmentation assigns labels to the individual pixels in the image, such that pixels with similar labels share some semantic characteristics.

As an example of image segmentation in practice, consider the autonomous driving domain, where a machine makes driving decisions (e.g., about speed and steering) based on information from its camera and other sensors. For such a system, it is critical to quickly process which aspects of an image comprise the road ahead and lane lines vs. a curb, a shoulder,
grass, traffic signs, sky, and other things that should be avoided, like pedestrians and buildings. Each of these objects is predicted as a polygon (or map), and their positions can be used to inform driving decisions or as input to safety systems.

Image segmentation has also been successfully applied in the medical domain. Given an image of cells or cell components, a longstanding challenge for biomedical researchers has been to automatically detect cell types or elements within a cell. When trained on data containing labeled cell segments the U-Net model,\(^9\) based on a fully convolutional network architecture, enables highly precise relative location of cell components within images; it has been applied to brain image segmentation, liver image segmentation, and cancer/tumor segmentation tasks, among other medical research applications.

The task of object detection focuses on predicting the occurrence and position of objects or features within an image. Typically, an object detection model is trained to predict bounding box coordinates \((x,y)\) — height and width) and a class label for each object in an image. To achieve this feat, the model learns to perform two tasks: first, it generates candidate region proposals for where objects might reside in the image; second, it assigns each of these regions to a given class. The more efficient models perform both tasks simultaneously. Object detection is recommended for use cases where it is desirable to learn the location, type, or count of objects in an image. For example, object detection can be applied in agricultural sorting facilities to count the number of products on a conveyor belt or to automatically sort products by type.

**Video Classification**

Video classification focuses on identifying the activity occurring in a video segment. Automatically classifying video segments as containing specific activities — sports, cooking, construction, etc. — is useful because these classifications are

---

**FIGURE 3.2 U-Net cellular-level segmentation of insect brain cells [courtesy of NIH.gov].**
often helpful for downstream tasks (such as video search and summarization). Several factors make video classification a much harder task than image classification. First, the naive approach of simply classifying each video frame individually doesn’t do as well as expected because of the large amount of noise in videos (motion blur, out-of-frame objects, etc). In addition, video analysis is complicated by the temporal characteristics of video content — activity in a video is not just dependent on the content of each video frame, but also on the temporal changes between each frame. This is particularly challenging in longer videos, where scenes, camera angles, or actions are likely to change frequently. Additionally, representing an input video as a space-time volume (number of images * number of frames) creates very high-dimensional input that can be prohibitively expensive to use for training. Recently introduced neural networks address these challenges with selective downscaling of input, use of 3D convolutions, and explicit extraction of temporal motion (optical flow maps).

It should also be noted that the quantity of precisely labeled video suitable for image analysis training in numerous fields is still relatively small, though this is beginning to change with the advent of video datasets like Moments in Time, YouTube M8, UCF101, and Sports 1M. Interest in self-driving cars and surgical robotics has driven the growth of private labeled video libraries too, but these are not likely to be shared outside of the creator companies and their partners; in addition to being potentially costly to create (consider a surgeon watching, segmenting, and labeling each frame of 12 hours’ worth of video of a single procedure, then multiply this by many such procedures), the datasets themselves
create a competitive advantage for very lucrative future markets. However, we expect that in a matter of one or two years there will be a larger open source pool of pretrained models available for video classification, if not data, that may be used for transfer learning for different tasks (just as we saw in the past for standard image classification).

**Unsupervised Learning Tasks**

While supervised learning models have been responsible for most of the recent advances in deep learning, there are situations where massive amounts of unlabeled data exist and these techniques cannot be applied. In these contexts, unsupervised learning approaches (which focus on extracting any underlying structure within data) provide an avenue to learn independently from data without any labels.

**Image Generation**

Given a known dataset, image generation is the task of synthesizing new images that do not exist in the dataset but belong to the same distribution. For example, given a dataset containing images of houses, an image generation model would be tasked with synthesizing new house images that are not present in the dataset, but can be clearly identified as houses. Note that each house in the dataset does not need to be labeled or annotated. Deep learning models for image generation typically adopt generative approaches, where the task is to learn the parameters of the source data distribution so that novel images can be sampled from this distribution. These approaches have been applied for data augmentation (generating synthetic data used for training CNN models), generative
art (generating images that imitate the artistic style found in other images, generating novel image or art samples, music generation), and low-resolution video generation.

The following table gives pointers to our current recommendations for various IA tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Recommended Model Architectures</th>
<th>OSS Frameworks with Official Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Detection</td>
<td>R-FCN, Faster R-CNN, Yolo, SSD</td>
<td>TensorFlow Github</td>
</tr>
<tr>
<td>Video Classification</td>
<td>I3D, T3D, Attention Clusters</td>
<td>TensorFlow Hub</td>
</tr>
<tr>
<td>Image Generation</td>
<td>DCGAN, CycleGAN, ProGAN, BigGAN, Super resolution: ESRGAN</td>
<td>-</td>
</tr>
</tbody>
</table>

**TABLE 3.1 A summary of recommended models for image analysis tasks, with pointers to official implementations within model repositories supported by leading deep learning frameworks.**

**Recommendation** Note that these repositories are constantly evolving (new implementations of the latest model architectures are added as they become available). In addition, many researchers provide a GitHub repository containing code that implements their model and replicates their results.
The reader is encouraged to explore these resources as a first step in applying these models.

**Picking a Good Model: Accuracy vs. Latency**

Given the plethora of neural network architectures that exist, selecting the right model for your specific image analysis problem can be a challenge. Perhaps the most salient factor in choosing a model is something we will refer to as the **accuracy-latency tradeoff**. At the core of the tradeoff is the accuracy gain associated with computationally expensive model architectures compared to “good enough” results produced by less complicated architectures.

In general, the accuracy of deep learning models increases with the number of layers (depth) in the model architecture (see **Figure 3.3**). The intuition here is that additional layers within a model allow it to approximate even more complex non-linear functions and learn features at various levels of abstraction. However, this increase in performance comes at a cost: in addition to the requirement of a larger dataset, each additional layer introduces additional parameters that need to be trained, weights that need to be stored, and compute operations that must be performed during inference. Overall, it takes more compute resources (faster GPUs, longer training time, larger datasets) to train deeper models, and they have higher latency when deployed for inference.¹⁰

To address use cases where low latency is a strong

¹⁰ For an extensive treatment on benchmarking the efficiency of image classification models, see the paper “Benchmark Analysis of Representative Deep Neural Network Architectures” by Simone Bianco et al. [https://arxiv.org/pdf/1810.00736.pdf].
requirement (e.g., for IoT and edge devices), there have been efforts to introduce model architectures optimized to have fewer parameters, low latency, low storage requirements, and low power requirements (see FIGURE 3.4). The reduction in parameters is enabled mainly by the use of fewer layers (e.g., SqueezeNet) and the use of alternatives to standard convolution layers, which are expensive to compute (e.g., the use of depthwise separable convolutions in the MobileNet architecture). More recently, AutoML approaches focused on automatically identifying fast but highly accurate models have yielded impressive results (e.g., NASNet, MnasNet, FBNet, and EfficientNet). Other approaches to improving latency include efforts in model compression (where the focus is on reducing the number of parameters in a model) and model quantization (where the focus is on reducing the precision requirements for a model, e.g., from 32 bits to 8 bits).

**FIGURE 3.3** There is a tradeoff between accuracy and latency for deep learning models. Large models are more accurate but incur higher latency due to the large number of parameters.
We offer the following guidelines on when to choose large models with high accuracy vs. small models with low latency.

**High-parameter models** (high accuracy) are preferred when:

- The task is risk-averse and sensitive to error (e.g., medical imaging, where there is significant risk to human well-being in the event of errors).
- The task does not have real-time requirements and can be run in batch mode (e.g., a periodic batch process where a retail firm uses deep learning to generate visual quality scores or captions for images in their storefront).
- There are no strict budget concerns (time or financial) with respect to training or retraining the model for

**Figure 3.4** Examples of models typically used in low-latency scenarios, given their relatively fewer number of parameters. To further reduce the latency associated with these models, we encourage the reader to explore model compression and quantization methods.
production deployment. 

Low-parameter models (low latency) are preferred when:

- The model needs to run on resource-constrained devices (limited power, memory, compute, or storage).
- The model needs to run in real time.
- The model needs to be frequently distributed to target devices.
- Training or retraining the model is a frequent requirement (e.g., for online systems) and incurs significant costs, including limiting compute resources available to other systems.

**Recommendation** While the best performing models boast high accuracy, they may not be the best fit for your use case given the resources required to deploy them in production. The relationship between the number of parameters and the accuracy of models is parabolic: small increments in accuracy come at a cost of large increments in the number of parameters. Thus, we recommend initial experimentation with smaller models before progressing to larger models; small AutoML models (see **Figure 3.4**) balance the accuracy-latency tradeoff well and should be considered.
A deep learning approach to image analysis has two important benefits: it allows a machine to automatically approximate complex functions that map the inputs of the model to its outputs, and the model automatically learns to identify the features needed to predict outputs, eliminating the need to manually engineer these features.

While this works very well in practice, a limitation of neural networks is that there is not a way to inspect the structure of the function learned by the model; there is no intuitive relationship between the thousands or more learned weights and the function being approximated. With deep models, it is hard — almost impossible — to intuit how changes to specific parameters or input segments affect the output. For example, how does the 10th neuron weight, in a model comprised of 8 million weight parameters impact output classification after training against 8,000 images?

For these reasons, deep learning models are referred to as black box models. This is in contrast to a non-black box model (such as linear regression), where the importance of each parameter is explicitly specified and can be easily understood. As deep learning models are increasingly deployed, several questions related to their interpretability arise: How can we explain the model’s decisions? How will the model behave
when it encounters edge cases? What biases does the model have? When can we trust the model? How can we improve the model?

These questions are even more important in sensitive applications, such as medicine, finance, and fully autonomous systems such as self-driving cars. The search for answers underpins a growing and active field of research focused on the interpretability and explainability of models. The solutions that have been proposed can be categorized into visualization methods and attribution methods.

**Visualization Methods**

Broadly speaking, visualization methods seek to understand what a deep learning model "sees." For example, some visualization methods have sought to understand which patterns each filter in a convolutional network layer has learned to recognize. One way of achieving this is to search for images

---

**FIGURE 4.1** An image containing random noise is optimized with the objective of activating a neuron in the GoogLeNet model. It turns out this neuron has learned to detect mesh patterns. From [https://distill.pub/2017/feature-visualization/](https://distill.pub/2017/feature-visualization/)
that trigger the maximum response from certain neurons. These images are generated through an optimization approach which begins with random noise that is iteratively modified (based on gradients) to maximize the activation of a neuron or layer.

The following code snippet provides an example of how to visualize the pattern learned by the first neuron in the conv5_3/conv5_3 layer of a VGG16 network.

---

import numpy as np
import tensorflow as tf

from lucid.misc.io import show, load
import lucid.optvis.objectives as objectives
import lucid.optvis.param as param
import lucid.optvis.render as render
import lucid.optvis.transform as transform

# Lucid's modelzoo can be accessed as classes in vision_models
import lucid.modelzoo.vision_models as models

_ = render.render_vis(model, "conv5_3/conv5_3:0")

**FIGURE 4.2**, created using the Lucid library, shows a visualization of patterns learned by two layers in the 16-layer VGG16 model: conv1_1, which is the first convolutional layer in the model, and conv5_3, which is the last convolutional layer in the model. We see that early layers (e.g. the first convolutional layer) learn low-level features like textures, while later layers (e.g. the last convolutional layer) focus on shapes and objects.

**Attribution Methods**

While feature visualization helps us understand what the network sees, it does not provide much information on how the output of each neuron or layer contributes to later decisions or why each decision was made. Attribution methods expand on these areas by exploring the relationships between neurons. A common approach is the use of saliency maps, which are heat maps that indicate the pixels within an image that contribute the greatest parts of the classification decision of a model. Several other popular attribution methods have
been proposed, including Grad-CAM, SmoothGrad, occlusion, Layerwise Relevance Propagation, LIME, and DeepLIFT.

The following code snippet shows how to generate heat maps that highlight which pixels influence the decision of a classifier trained on images of dogs for the class boxer. It is implemented using the keras-vis library. FIGURE 4.3 shows the parts of the image that the classifier focuses on.

```python
import numpy as np
import matplotlib.cm as cm
from matplotlib import pyplot as plt
from vis.visualization import visualize_cam, overlay
from vis.utils import utils
from keras import activations
from keras.applications import VGG16

# Build VGG16 network with ImageNet weights
model = VGG16(weights='imagenet', include_top=True)
# Get the index of the last layer in VGG16
layer_idx = utils.find_layer_idx(model, 'predictions')

# Swap softmax with linear activation
model.layers[layer_idx].activation = activations.linear
model = utils.apply_modifications(model)

tens = np.zeros((1, 3, 224, 224))
tens[0, :, :, 0] = utils.load_img('dog1.jpg', target_size=(224, 224))
tens[0, :, :, 1] = utils.load_img('dog2.jpg', target_size=(224, 224))

plt.rcParams['figure.figsize'] = (18, 6)
plt.figure()
f, ax = plt.subplots(1, 4)
for i, img in enumerate([tens[0, :, :, i] for i in range(3)]):
    ax[i].imshow(img)
```

Interpreting Models  39
# 242 is the ImageNet index corresponding to `boxer`  
grads = visualize_cam(model, layer_idx, filter_indices=242,  
seed_input=img, backprop_modifier="guided")

# Generate heat map using gradients  
jet_heatmap = np.uint8(cm.jet(grads) * 255)[:, :, :, 0]  
ax[i*2].imshow(img)  
ax[i*2 + 1].imshow(overlay(jet_heatmap, img))

**FIGURE 4.3** Here, the keras-vis library has been used to identify the pixels in the image that contribute the most to the classification `boxer`. It focuses on the dog's face and legs where available.

**Open Source Libraries for Model Interpretability**

As an example of how deep learning has become more standardized, there are now libraries (see the following table) that allow end users to inspect their models using these methods.

**Recommendation** As data scientists build custom models or apply pretrained models to various problem domains, we recommend the periodic use of the interpretability methods discussed in this section for inspecting models. Frequently, insights from model inspection can provide valuable feedback on how to further optimize the model or point to a need for additional data collection. For example, it is important to verify that a model trained to detect currency notes
for individuals with visual impairment is leveraging salient information (e.g., landmarks on the bill, numbers, and markings) as opposed to spurious characteristics (color and edges).

<table>
<thead>
<tr>
<th>Interpretability Library</th>
<th>Description</th>
<th>Supported Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucid</td>
<td>A collection of infrastructure and tools for research in neural network interpretability</td>
<td>Feature visualization</td>
</tr>
<tr>
<td>DeepExplain</td>
<td>A unified framework for state-of-the-art gradient and perturbation-based attribution methods</td>
<td>Feature attribution: saliency maps, gradient input, integrated gradients, DeepLIFT, e-LRP, occlusion</td>
</tr>
<tr>
<td>Keras Visualization Toolkit</td>
<td>A high-level toolkit for visualizing and debugging trained Keras neural net models</td>
<td>Feature visualization: activation maximization; Feature attribution: saliency maps, Grad-CAM</td>
</tr>
<tr>
<td>LIME</td>
<td>Explaining the predictions of any machine learning classifier</td>
<td>Feature attribution</td>
</tr>
<tr>
<td>SHAP</td>
<td>A unified approach to explain the output of any machine learning model</td>
<td>Feature attribution</td>
</tr>
</tbody>
</table>

**TABLE 4.1** A list of open source libraries for model interpretability.
42 Interpreting Models
ConvNet Playground, the prototype created for this report, allows users to explore representations learned by a CNN model. It has two main parts. The first part — Semantic Search — demonstrates an implementation of content-based (semantic) search using modern pretrained CNN models. The intuition here is that various layers in a pretrained CNN will have learned important concepts that allow them to extract meaningful representations which can be leveraged in computing the similarity between images. The second part of the prototype — Model Explorer — is a visualization tool that allows the user to inspect what features have been learned by the layers in a CNN, and in so doing build better intuition on how CNNs work.

**Semantic Search**

We define the task of semantic search as follows: > Given a dataset of existing images, and a new arbitrary image, find a subset of images from the dataset that are most similar to the new image.

Semantic search is implemented as a three-step process. First, a pretrained CNN model is used to extract features (represented as vectors) from each image in the dataset. Next, a distance metric is used to compute the distance between each
Finally, to return results for a search query, we retrieve the pre-computed distance values between the searched image and all other images, sorted in the order of closest to farthest. The semantic search interface allows the user to perform a search query by selecting (clicking on) an image in the provided dataset.

In practice, there are many choices to be made while implementing a similarity search use case based on convolutional neural networks. An appropriate model architecture needs to be selected, as well as appropriate layers from the model and an appropriate distance metric. We have precomputed the extracted features from images in four datasets, using eight different models and eight different layers from each model. We have also computed the similarity between all of these features using four different similarity metrics. The prototype allows the user to explore and ask questions about the results of these computations.

**FIGURE 5.1** To implement semantic search, we use a CNN model to extract features (embeddings) from images in our dataset and compute similarity as the distance between these embeddings.
Datasets

The four datasets used for the prototype are described here.

Iconic200

This is a dataset of real-world images collected from Flickr (Creative Commons). It contains images spanning 10 keyword searches (arch, banana, volkswagen beetle, eiffel tower, empire state building, ferrari, pickup truck, sedan, stonehenge, and tractor), with 20 images per category.

These image categories were chosen deliberately with conceptual overlaps (several car brands, similar colors across

**Figure 5.2** The basic interaction flow of the ConuNet Playground Semantic Search module. The user can select an image and view a list of results which the model retrieves as the most similar.
classes) to highlight how various models perform in correctly representing similarity. For example, models with more capacity (layers), such as InceptionV3, are better able to model the notion of similarity for some overlapping categories and hence present better results.

**Figure 5.3** Example images from the Iconic200 dataset.

**Fashion200**

Fashion200 is a collection of 200 images (10 categories, 20 images per category) of real fashion models from the Kaggle Fashion Product Images dataset. Images have a maximum width of 300px. Categories include flipflops, menjeans, menshirt, sandals, sportshoe, womenheels, womenjeans, womenshirt, and womentshirt. Again, concept overlaps exist to allow the user to interactively explore how well various models/layers correctly represent and distinguish each category.

---

FIGURE 5.4 Example images from the Fashion200 dataset.

Tinyimagenet
This dataset contains a subset of 200 64px * 64px images from the Tiny ImageNet Visual Recognition Challenge dataset. It consists of images from 10 categories (arch, bottle, bridge, bus, face, frog, goldfish, sandals, teapot, and tractor).

FIGURE 5.5 Example images from the Tinyimagenet dataset.

13 https://tiny-imagenet.herokuapp.com/
**Cifar10**

This is a subset of the popular CIFAR10 dataset containing 20 images from 10 randomly selected classes (*airplane, car, bird, cat, deer, dog, frog, horse, ship and truck*). Each image is 32px * 32px in dimension.

![Example images from the CIFAR10 dataset.](image)

**FIGURE 5.6 Example images from the CIFAR10 dataset.**

**Models and Layers**

We provide results from nine models (*vgg16, vgg19, mobilenet, efficientnetb0, efficientnetb5, xception, resnet50, inceptionv3, and densenet121*) and a selection intermediate models, using eight layers from each model. We use only eight layers to reduce the viewer’s cognitive burden and to enable easy visual comparisons. To select which layers to use, we focused on convolutional layers with trainable parameters. We included the first and last convolutional layer in each model, then selected a random sample of six convolutional layers in between. The models are presented in order of increasing complexity (number of parameters) and show marked differences in their ability to generate features that correctly identify similar images. For convenience and reproducibility, we use implementations of pretrained models from the Keras Applications package.
**Distance Metrics**

We provide results from the use of four distance metrics in measuring the similarity between features extracted from all images in each dataset. The metrics used are the Cosine, Euclidean, Squared Euclidean, and Minkowski distances.

**Model Explorer**

![Model Explorer Interface](image)

**FIGURE 5.7** The Model Explorer interface allows the user to select a pretrained model and view visualizations of channels (groups of neurons) within layers in the model. Each image is an example of the patterns or features which the channel has learned to detect.

The second part of the prototype is a visualization interface for exploring the features or representations learned by
each layer in a pretrained CNN. We show visualizations for nine models (vgg16, vgg19, mobilenet, mobilenetv2, xception, resnet50, inceptionv3, densenet121, and nasnetmobile), each pretrained on the ImageNet dataset using weights from the official Keras Applications package.

**Emergent Design Principles**

ConvNet Playground was designed with several overarching goals: to demonstrate a concrete capability enabled by CNNs (semantic image search), to provide a learning experience where the user is introduced to concepts that help build intuition on how CNNs work, and to support users of varying levels of expertise (novices as well as experts). The following design principles were valuable in achieving these goals.

**Selective Revelation of Complexity**

We understand that our prototype will be used by individuals with varying levels of technical expertise and have adopted the selective reveal principle to accommodate all user types. At the start of interaction, the user is presented with a basic interaction flow which allows them to perform a single search task (click an image, view the top most similar results). Following this, the user can initiate an advanced interaction flow by selecting the advanced options toggle. This allows them to modify the search configuration (dataset, model, layer, and distance metric), view visualizations of embeddings extracted using each model, and compare the performance of each model for a given search query. Appropriate visual cues (connector lines, highlights) are integrated to suggest a meaningful sequence of actions.

50 Prototype
**Multimodal Interaction**

To help build intuition, we provide multiple scaffolds that help the user make sense of the presented semantic search results. For example, each search query reveals a top results panel with similarity scores displayed for each result. This panel is further summarized in the form of a “weighted” search score for each search. The search score is designed to communicate information on the performance of the current search configuration (selected model/layer) for the given search query. It is calculated as the percentage of returned results that belong to the same category as the selected image, weighted by position in the result list.

Next, users can view a visualization (UMAP) of the features extracted by using a particular layer in a given model. They can observe that models which perform relatively well have excelled at correctly separating the individual categories into delineated clusters. Users can also observe important correlations between search result quality and the _shape_ of the feature clusters generated by a given layer. Furthermore, as users hover over images in the dataset, their respective positions in the UMAP visualization space are highlighted.

Finally, the Model Explorer module lets users inspect what features/patterns are learned by each layer and may provide explanations of why some layers provide certain results. For example, when a search query for a banana is performed using the first layer in the VGG16 model, the results also contain a yellow Volkswagen Beetle. On face value, it is not immediately clear why this type of mistake is made. However, by reviewing patterns learned by the first layer in VGG16 (it learns...
to detect colors), it becomes more apparent that the layer returned a yellow Volkswagen Beetle car mainly because it is the same color as the search query (a banana). This observation also hints at the relevance of early layers for search queries where color is a relevant aspect of similarity.

**Implementation Details**

Additional details on the user interface and backend system for ConvNet Playground is provided below.

**User Interface**

The prototype is designed in React.js as a static web application where the content of the application is precomputed and loaded at runtime.

**Generating Images of Neurons and Layers**

Each model architecture shown (vgg16, vgg19, etc.) has been pretrained on the ImageNet dataset. The images/visualizations of neurons represent an example of what the given neurons have learned to look for. They are generated using an iterative optimization process which synthesizes input that causes the neurons to have high activation. The process begins with random noise (an image that looks like 1980s TV static). This image is then shown to the channel, and on its gradients (derivatives) the image pixels are updated to arrive at a final image that maximally excites the channel.

*Note* While the resulting visualizations may not all correspond to identifiable objects/concepts, we consistently see increasingly complex patterns as we progress through the layers in the model. We recommend the work by Olah.
et al. on feature visualization\textsuperscript{14} as further reading on the topic of visualizing neurons in a model.

Images that represent layers and neurons are generated using the Lucid library, which implements optimization-based feature visualization for neurons, channels, logits, and layers of a neural network. For each model we select a subset of layers to display, and for each layer we select a subset of channels. Only layers with trainable parameters and activations are visualized (a requirement for optimization-based feature visualization). For convenience, 30 channels are sampled at random from each layer. We used the lucid4keras package to easily import Keras models to a format that can be processed by the Lucid framework.

\textit{Generating Similarity Metrics and UMAP Embeddings}

For each layer in the models we support, we extract embeddings (representations of the images) from images in our datasets. Similarity computation is performed with four similarity metrics using the \texttt{scipy} library. The results from these processes are stored in \texttt{json} files and subsequently loaded by the prototype interface. Using the same process, we also precompute UMAP embeddings of features for each image in each dataset.

\textit{Beyond the Prototype}

In practice, there are numerous use cases for which it is beneficial to extract representations of an image to be used for downstream tasks. In our prototype, we leverage these

\texttt{14 https://distill.pub/2017/feature-visualization/}
embeddings as they are for similarity search and make a few assumptions (i.e., that the searched image is arbitrary and may or may not be similar to any other existing image in our dataset). While this approach is simple, it does excel in demonstrating how much value is immediately available from pretrained models.

In the real world, there may be additional information that can be exploited for even better search results. For example, an e-commerce retail shop with a constrained set of search queries can fine-tune its search by using a specialized neural network trained to identify similarity for a specific dataset (see use of triplet loss\textsuperscript{15} and siamese networks\textsuperscript{16}). Furthermore, it may be desirable to search within parts of images in the dataset as opposed to matching the entire image. For this, a two-stage approach for matching multiple objects in the search query (extract object crops and use these as search queries) may be more appropriate.

**Recommendation** The overall approach of utilizing features extracted from images can be extended to a broad range of tasks, such as organizing images of documents, curating unlabeled image datasets, content recommendation on e-commerce platforms, object detection, defect detection, video analysis, and more. As the user navigates our prototype, we hope it provides insights into the performance of various models/layers for different dataset types, and intuition on when to apply them.


\textsuperscript{16} Siamese Neural Networks for One-shot Image Recognition ([https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf](https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf)).

54 Prototype
In this chapter, we review the landscape of deep learning for image analysis to help the reader sort through the variety of relevant offerings as a service and open source tools available today.

**Deep Learning for Image Analysis as a Service**

If you are considering using deep learning but don’t currently have the resources to develop and train your own models, there are several well-known companies that offer pretrained model services that you can integrate into your products. As each service is continually refining its offerings, the key deciders for your application may be pricing for the service delivery or your comfort with the platform ecosystem. Many of these pretrained services have comparable APIs covering tasks such as generic image classification, custom image classification, image captioning, and video captioning, amongst others. In some cases, the pretrained APIs made available may be too restrictive, fail to fit your specific problem, or be inappropriate for complex research use cases. For these scenarios, we recommend Machine Learning as a Service (MLaaS) offerings, which provide a managed environment (hardware, software frameworks) to run machine learning code/jobs—but as a short-term solution only. Some
leading providers that offer both pretrained APIs and MLaaS include Cloudera Data Science Workbench (CDSW), Google Cloud AI, IBM Watson AI, Amazon AI Services, and Microsoft Azure AI.

**Open Source Neural Network Libraries**

The field of deep learning has been driven by the contribution of open source frameworks and tools for both production and research purposes. These tools have evolved to support the diversity of use cases, with most making tradeoffs between simplicity and extensibility. It is attractive for a framework to allow new users to intuitively and rapidly prototype ideas, as well as supporting robust large-scale production deployment. But these requirements are challenging, and various frameworks satisfy them to different extents.

The landscape for deep learning libraries is dynamic, and several of these tools will likely become either outdated or highly specialized as popularity shifts and performance gains are made using new approaches. As such, we generally recommend against data product organizations solely relying on a single (niche) tooling framework or platform provider, as future changes (or discontinuation of support for a library) may incur costs in R&D, production, and maintenance of models.

**TensorFlow**

TensorFlow from Google is a Python-based library for dataflow and differentiable programming, and it is currently the fastest-growing deep learning framework. In the years since its initial release, TensorFlow has grown beyond a library to become a multifunction platform and ecosystem (TensorFlow
Extended) that supports the pipelines, design, training, and deployment of machine learning models. As the primary tool used internally by Google for its machine learning production workflows, TensorFlow was built for large-scale deployment across servers, on the web, and on mobile/embedded devices.

TensorFlow originally followed a “define and run” paradigm with computation represented using a dataflow graph and a “session” created to run parts or all of the specified graph. This approach provided benefits such as parallelism (easily identifying components that can execute in parallel), distributed execution, fast compilation, and portability (graphs are language-independent), but also made it less intuitive and challenging to debug.

The June 2019 TensorFlow 2.0 release aims to address the latter usability issues with the introduction of eager execution, an imperative programming approach that evaluates operations immediately, without building graphs. It also includes tight integration with Keras (see 6.4 Abstractions on Top of Deep Learning Libraries), as well as implementation of well-known computer vision models made available in the TensorFlow model repository, TensorFlow Hub. TensorFlow is preferred for native development for production on the Google Cloud Platform.

**PyTorch**

PyTorch is a Python-based deep learning framework that provides support for computer vision through the Torchvision project. While PyTorch natively supports building arbitrary models for computer vision, Torchvision predefines some of the most common architectures and supports pretrained
versions of them, trained on popular open datasets made available through the PyTorch Hub. The Torchvision project also provides several types of data loaders for common computer vision tasks, including classification, segmentation, and object detection.

Models built with PyTorch and Torchvision can be deployed through standard PyTorch toolchains. PyTorch supports the ONNX library for model export, while the recently merged Caffe2 library has strong support for model deployment. Many senior developers who cut their ML teeth on the command line in the 2000s and 2010s will be very familiar with PyTorch’s tooling and simple premise—an effective library to build what you need—which makes it particularly useful for research.

**Apache MXNet**

Apache MXNet ships with the Gluon library, which provides a simple “building blocks” Python interface to the framework’s core tooling. Like with PyTorch, arbitrary computer vision components can be built using just the tools provided by Gluon, but the closely related Gluon-CV (Computer Vision) project should be preferred. Gluon-CV provides state-of-the-art pretrained models and the necessary tools to train them for several of the most popular computer vision tasks: classification, object detection, semantic and instance segmentation, and even pose estimation. Image augmentation and transformation is available for both inputs and labels, which is required for more advanced tasks. Several premade Python scripts are available that allow users to train these standard models on their own datasets. However, if users
desire to experiment with customized loss functions or use nonstandard models, it will still require writing a lot of code from scratch.

MXNet supports model export through the ONNX interoperability framework (see Abstractions) and production deployment through a C++ API. MXNet emphasizes speed of development and deployment of large-scale, deep neural networks—including multi-GPU training and optimized pre-defined layers. It is most suitable for native development for production on Amazon Web Services and (increasingly, following the spring 2019 end of Microsoft’s support to its home-grown CNTK framework) Microsoft Azure.

Chainer

Chainer is a Python framework that uses dynamic computational graphs. For image analysis applications, it supports several image classification algorithms, reinforcement learning, and generative model approaches and can do so leveraging scalable, distributed, multi-GPU setups. Many of these capabilities come as add-on modules which enable use of popular algorithms to reproduce results described in research papers. Other modules add development tools like visualization and hyperparameter optimization, or research field-specific tooling for use within biochemistry, for example. While researchers still find use in this contender for the fastest-performing tool for deep learning modeling, Chainer’s popularity (and uniqueness) has diminished drastically in the last two years relative to the previously mentioned frameworks.
**Deeplearning4j (DL4J)**

Eclipse Deeplearning4j is an open source deep learning library and framework designed for use with Java and for compatibility via API with Clojure, Scala, and Python (via Keras; see Abstractions). The tooling enables use of distributed resources, including distributed GPUs, and offers multiple options for model and results visualization. The resources are also composable, allowing small networks and machines to be aggregated into deeper neural networks like building blocks. It has been particularly popular with frontend designers used to using Java for production systems, and is often preferred for Android deployment.

**ML in the Browser**

There are several open source libraries which provide a simple JavaScript API that allows users to build and train machine learning models in the browser. These include the formal TensorFlow.js effort as well as numerous small-scale projects, such as ConvNetJS, Synaptic, and Neataptic.

As of now, TensorFlow.js is clearly the most mature API in terms of maintenance, integration with the broader ML ecosystem, and community adoption. The tool consists of two sets of APIs: the Ops API, which provides lower-level linear algebra operations (e.g., matrix multiplication, tensor addition, etc.), and the Layers API, modeled after Keras, which provides higher-level model building blocks and best practices with emphasis on neural networks.

Running inside the browser, TensorFlow.js is able to utilize the GPU of the host machine (with a CPU fallback), while on the server side TensorFlow.js is able to grab full access to core
TensorFlow. Via a converter connection to the full ecosystem developers can build, train, optimize, and test their models in TensorFlow (Python) and then export the resulting models for use in the browser.

**Abstractions on Top of Deep Learning Libraries**

As part of efforts to make deep learning research and applications more accessible, several libraries have been introduced that provide abstractions for high-level tasks in building neural networks. While hiding complex internals by abstraction can come at the price of flexibility and optimization, the rapid prototyping qualities of these tools are often appreciated by the community.

**Fast.ai**

Fast.ai is an opinionated wrapper for PyTorch that aims to simplify the use of neural networks by providing the user with best-practices presets and workflows out of the box. While its speed of evolution has led to outdated or missing documentation, the structure does deliver on computer vision tasks, among others. For example, using generative models, Fast.ai provides state-of-the-art deep learning tools to colorize black-and-white pictures and movies and generate super-resolution images.

**Keras**

Keras is a high-level Python deep learning API specification for building and training neural networks. The API is very well designed and allows for a user-friendly, modular, composable, and easily extensible interface. Several frameworks
(TensorFlow, DL4J, MXNet) have adopted the Keras API standard and offer “backends” that allow you to write Keras specification code that is executed by the respective framework.

Depending on the underlying framework, Keras supports both static and dynamic computation graphs, and it is excellent for rapid prototyping. While abstracting away the complexities of the underlying libraries means it is not ideal for research purposes, it is often considered a best tool for beginners in deep learning model building.

**Sonnet**

Sonnet is a TensorFlow wrapper developed by DeepMind, an Alphabet (Google) company, because existing libraries were judged insufficiently flexible for the DeepMind use case (where extensive use is made of weight sharing). Despite its specialization, we mention it here as the name does pop up regularly in relation to ML research.

**Core ML**

Core ML serves as a layer for bringing models built in other frameworks to production on Apple devices. While Core ML has generally not been used for standalone research and application development, it is important to leverage for delivering applications to iOS devices. For image analysis function development Core ML includes a built-in Vision framework and augmented reality kit.

**ONNX**

Developed in a continuing partnership between Microsoft, Amazon, and Facebook to support interoperability and help
developers avoid lock-in to a single ecosystem, ONNX enables the development of models using one architecture and deployment in another. It supports the export of models first to a serialized format before delivering them to accomplish their predictive/inferential task(s) in the production system.

**Choosing a Deep Learning Framework**

![Deep Learning Libraries 2019](image)

**FIGURE 6.1** A listing of statistics (stars, watching, forks, dependent repos) on the GitHub repositories for several deep learning frameworks.

Most of the frameworks presented here are written in Python and are open source, which makes them available to all users for free. They are all also rapidly evolving to fit a streamlined set of important capabilities (support for multiple hardware architectures, support for distributed/parallel computations, intuitive APIs, visualization tools, etc.). However, the
nature of a particular use case, the user’s level of expertise, and the future prognosis for each framework can be important factors when choosing which one to use.

For projects where large-scale production workloads are expected, we recommend TensorFlow, as it has been designed to handle complex production requirements — real-time and batch processing, deployment to multiple platforms (web, mobile, servers) — and has an ecosystem of libraries for this purpose (TensorFlow Extended). For teams that are focused on research and rapid prototyping, PyTorch has been a community favorite for these sort of explorations. However, the path to deployment across different platforms is less clear and straightforward for models developed in PyTorch. For teams who are new to machine learning and want to rapidly try out ideas or apply well-known ML models, we recommend the Keras and Fast.ai wrapper libraries. Finally, of all the frameworks presented, we find that TensorFlow and PyTorch show the most promise for future longevity, based on the size of their respective communities (see FIGURE 6.1) and the overall health of the projects (in terms of active contributions).

Repositories for Pretrained ML Models

Some of the frameworks mentioned above offer repositories of open source pretrained models (also known as model zoos) that can be easily imported into a user’s workflow. Short descriptions and pointers to these model repositories are provided below.

64 Deep Learning in Industry Today
### Deep Learning for Image Analysis Products

There are a variety of commercial image analysis products that successfully bundle deep learning models into usable capabilities. For example, Topaz Labs offers several products that leverage GANs in providing impressive super-resolution (up to 6x!), noise removal, and image upscaling capabilities. Let’s Enhance provides similar image generation capabilities and supports batch processing workloads. We also like Runway ML, a tool that allows artists and creators to apply a suite of machine learning models and provides open source interfaces.

<table>
<thead>
<tr>
<th>Supported Framework</th>
<th>Description</th>
<th>Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>TensorFlow Hub: A library for reusable machine learning modules. At the time of writing, provides 75 models trained on NLP tasks, 71 models for image processing tasks, and 2 models for video processing tasks.</td>
<td>Image, NLP</td>
</tr>
<tr>
<td>Keras</td>
<td>Keras Applications: Applications module of the Keras deep learning library. Provides model definitions and pretrained weights for nearly 30 popular architectures, such as VGG16, ResNet50, Xception, and MobileNet.</td>
<td>Image</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Torchvision.Models: Subpackage containing definitions and pretrained models for the AlexNet, VGG, ResNet, SqueezeNet, DenseNet, InceptionV3, and GoogLeNet model architectures. The PyTorch Hub contains a more general set of pretrained models also covering image analysis models.</td>
<td>Image</td>
</tr>
</tbody>
</table>
plug-ins for integration with game development and animation toolkits, Arduino, Python, Ruby, Adobe Creative Suite, and more. Bear in mind, however, that these products, while impressive, are generally not designed for incorporation into large-scale workflows and may also leverage user image data to enhance their own model capabilities.

Lastly, please feel free to run TensorFlow, MXNet, Keras, DL4J, and other tools from the landscape on CDSW, a bring-your-favorite-editor, browser-based environment for ML and deep learning application development geared for production. CDSW provides secure model sharing among teams and secure connectivity to your organization’s data, whether in the cloud or on premises.
The promise of harnessing deep learning for image analysis in solving large-scale problems is attractive and full of potential. However, there are important ethical issues that arise. In this section, we explore significant ethical pitfalls related to biases in training data, privacy, misinformation, and environmental issues.

**Training Dataset Bias**

Deep learning is driven by the availability of large datasets used to train models. In theory, a requirement for good model accuracy is that the data used to train a model should be representative of the application scenarios where the model will be deployed. For example, a model meant to detect events during a lawn tennis match (e.g., a serve, a fault, etc.) should use a video dataset of athletes playing lawn tennis as opposed to athletes playing badminton or beach tennis. In scenarios where the application area is of critical importance to human life and well-being (medicine, autonomous driving, human resources, etc.), the impact of a mismatch between the dataset and its application area can have even more severe implications beyond accuracy. These mismatches or biases can result in real-world harm and promote inequality.

For example, consider a scenario where a dataset of human
images that may overinclude and underinclude people of various skin colors, genders, or shapes is used to train models. A resulting security system might recognize people of one skin color better than another. An attention detection system might not notice attention from people in some groups. An emotion detection system might misunderstand people with certain facial features or fail to account for cultures where a head-nod signals the negative instead of the much more common affirmative—or simply be based on a flawed premise.17

Data biases could also cause harm within other types of image data. For example, cancer lesion detection models built using medical image samples from individuals in North America will not work well for individuals in parts of India and Africa. Self-driving car models trained on North American roads may struggle in parts of Africa and Asia. This could mean that life-saving cancer diagnosis software will be unavailable to certain parts of the world, or that the safety and efficiencies provided by autonomous vehicles won't be available to less-developed countries.

Finally, while the debate among data scientists on whether to support military development of next-gen weapons (aka “deterrents”) may be considered a personal decision by some, even those who feel they would support hunter/killer-style robotics efforts18 need to acknowledge that image analysis developments so far haven’t solved the issue of bias, and that that presents literal dangers in the deployment of these future capabilities.

17 https://journals.sagepub.com/eprint/SAUES8UM69EN8TSMUGF9/full
Privacy

Public surveillance is already a well-understood concern. Improvements in image recognition enable even more and better surveillance. This not only further curtails the general "right to be left alone" but also deepens particular privacy concerns. It becomes easier to track a specific person and identify the racial or political group they belong to. Better interpretation of satellite image data (in visible and non-visible spectrums) allows easier surveillance of individuals’ or groups’ activities even while on their own property.

Preventing the obvious harms that can flow from these capabilities requires a joint effort between policymakers and data practitioners to create and respect privacy standards that meet the expectations of the public in terms of scope and fairness. Recognizing the lack of consent of the general public in general surveillance, the City of San Francisco in May 2019 became the first US city to ban public use of facial recognition. However, the UK and Chinese governments (among others) are moving in the opposite direction with regard to views on privacy in public image recognition. Multiple police forces in the UK, already known for its abundance of public security cameras, are testing facial recognition systems to identify “criminals,” even though they haven't proven the systems to be robust enough to appropriately recognize ethnic minorities. In China, documentation has been collected (by New York Times journalists) of the government using fa-

19 https://www.bbc.co.uk/news/technology-48276660
20 https://www.bbc.co.uk/news/technology-48222017
cial recognition specifically to target the Uighurs ethnic minority group. This targeting has even mobilized commercial software companies to design systems to alert the police if six or more Uighurs are identified in a given area within a given span of time, and represents among the most oppressive uses of image analysis capabilities.

Domestically in the US, a broader need for commercially respected standards has been highlighted by recent scandals, like the one caused by online photo storage firm Ever. In May 2019, it was revealed that the company had decided to change business models and use, instead of simply store, customers’ private photos in its new AI-linked business without notifying customers or asking for additional consent, arguing the fine print of a complex privacy agreement gave it permission.22 Around the same time, IBM was caught publishing a dataset of images scraped from photo sharing site Flickr without users’ permission.23 While website scraping has been a norm of data science practice for years, the advent of privacy and data use regulations like the EU’s General Data Protection Regulation (GDPR) should give one pause before basing a new data product on others’ personal images. More simply, if a business use case is tangential to the original intent of the acquired personal data, it’s important to consider carefully the ethical implications of the development as well as any business consequences for violations of any new regulations.

22 https://www.nbcnews.com/tech/security/millions-people-uploaded-photos-ever-app-then-company-used-them-n1003371
Manipulating Reality

Image generation techniques make it simpler for people without special training to generate image and video content, and seeing is believing. This is great news for creativity and entertainment purposes, but it can also be used — intentionally or accidentally — to confuse or mislead people, or outright manufacture facts.

![Diagram showing image generation process](http://grail.cs.washington.edu/projects/AudioToObama/siggraph17_obama.pdf)


Images or videos created using the image generation approaches discussed in this report have become known as "deepfakes". It immediately became clear that deepfakes of politicians’ speeches and activities could cause trouble for political parties and governments. Politicians are highly visible, which means there is ample training data available for image generation. This could be applied to “make” a politician say or do arbitrary things; e.g., taking contrary or

---

immoral positions, or declaring an evacuation, or issuing police or military orders. States have recently begun regulating deepfakes, while (in the context of foreign government interference in US elections) the US House Intelligence committee has taken up these concerns with the expectation that the 2020 US presidential race will face realistic use of deepfake videos and imagery in an attempt to influence the outcome.

The deepfake trouble may soon extend beyond government and famous figures, though. With teams working across the globe, connected primarily digitally (as is the case for the group writing this report), there is more and more video data available to create fake video, and increased reliance on video as a means of working remotely. This means there’s a risk that we may soon see manipulated videos allowing a malicious party to simulate instructions from corporate officers authorizing a bank transfer or other significant corporate action.

And generating video to mislead humans is not the entire story. Subtle changes to street markers that autonomous vehicles use could cause chaos on the roads, as with this fake-driving lane hack. It’s also possible to trick and misguide image analysis-controlled computers, where the worst-case scenario, whether on an individual or international scale, is limited only by hackers' access to the right digital resources.

---

As we noted in an earlier report (see interpreting_models), advances in interpreting image models like those used for image analysis can assist in avoiding this kind of interference.

**Environmental Considerations**

A final ethical consideration for deep learning model development and use, whether for image analysis or other tasks, has to do with the environment: training and deploying a model consumes a massive amount of power and creates a proportional amount of pollution. This MIT Technology Review article’s title alone conveys the base concern: "Training a single AI model can emit as much carbon as five cars in their lifetimes".  

There are also real business costs to training and running massive AI models, whether in the cost of electricity, cost of

---

keeping your data servers cool, or cost of leveraging someone else’s cloud services for significant periods of time—hours, days, or even weeks. While model training takes the most resources, we note that model prediction and inference can also be power-hungry processes that need to be optimized. For online systems that support real-time predictions, it may be valuable to explore power-aware model quantization approaches and robust caching to reduce the overall power footprint. Being efficient in the case of deep learning isn’t just smart data science or a good-natured business practice, but a true operational expense consideration.
From Supervised to Unsupervised Learning

The task of extracting value from large troves of unstructured, unannotated data can be prohibitively expensive with the common supervised learning approaches in use today. In most cases the labeling of new datasets literally requires a human to add labels or append metadata to individual images in sets of thousands (or more). In the case of medical/surgical robotics, for example, this may require a highly trained medical professional to review single frames of videos taken from surgeries lasting 8 to 12 hours — which is likely not what they signed up for.

There has been progress in research that allows us to work with limited labeled data (see FF10: Learning with Limited Labeled Data), and we expect to see an increase in the flexibility of data science modeling in adapting to unlabeled data. We also expect to see more specialized and private image and video libraries created specifically to support emerging image analysis applications.

But there’s a danger in relying solely on purchased access to an existing dataset or on black box models developed by a third party (at least, as anything more than a short-term bridge). Such reliance removes organizations’ ability to fully leverage the unique advantages their own data could provide,
in addition to offering competitors insight into (if not outright outsourcing) key components of their business. Specifically, both you and your competitors could find yourselves building models based upon the same commercially available data resources, negating any modeling advantage you may have had over each other. Further, feeding your own image data through a third-party system to provide more tailored models for your own firm may ultimately support and enhance the models made available to other clients—including direct competitors. This may be legally permissible if the third party isn’t "sharing your data," and is instead simply "updating" or "supplementing" its own model(s) on offer.

We have always held the mantra that you cannot outsource your core business when it comes to use of data in your organization, and going forward we see this increasingly becoming a point of caution in commoditized ML—and particularly in deep learning for image analysis. While cost-challenging now, we expect that the organizations that are going to be the most successful and develop the most unique capabilities in vision-based automation in the long run are going to be those that have an internal data science capability, allowing them to fold together proprietary and publicly available data resources to create effective models tailored to specific company data product needs.

**From Complex Networks to Simpler Networks**

We are seeing a transition in deep learning from image analysis networks with enormously deep and complex architectures to ones that are simpler and more task-specific. Once key tasks have been modeled, the models could perhaps be
retailed with multi-task learning approaches (see FF08: Multi-Task Learning), or other ensemble modeling approaches to enable broader solutions.

For example, in lieu of a massive, general deep learning model to simultaneously predict who is doing what in a video, an efficient future model will focus on precise body pose estimation/identification in the video without including facial recognition. The recognition aspect of the application may be developed separately (to its own high level of accuracy) for potential integration.

Furthering the push to lighter-weight and more focused deep learning, 2019 research out of MIT recognizes that most neural networks begin larger/deeper than they need to be, and some features of these deep general neural networks don’t factor significantly into specific task results. Such insights will only further accelerate the push to faster, more capable, and more secure mobile image analysis (and other neural network-based) applications.

**External Factor Training of Deep Learning: Image Analysis Systems**

Future image analysis will also step away from pure reliance on information from only within a given image to create classifications/predictions. For example, folding measured human stress responses into various driving situations is aiding recognition training of self-driving cars. Similarly, companies working on fully automated surgical robotics systems are correlating real-time external "view" CAT scan

30 [https://openreview.net/pdf?id=rI-Lb3RcF7](https://openreview.net/pdf?id=rI-Lb3RcF7)

31 [https://openreview.net/forum?id=SyNvti09KQ](https://openreview.net/forum?id=SyNvti09KQ)
data as well as positional and telemetry information from the surgical tools themselves to enhance critical segmentation and identification tasks. Meanwhile, dual-imaging processes using electron microscopy have begun enabling molecular biologists and biochemists with much more effective identification, modeling, and functional understanding of cellular components by layering different types of imaging data on top of each other.\(^{32}\)

In the next couple of years we expect to see even more development in deep learning and image analysis, featuring incorporation of multiple image sources and complementary data to create highly accurate mappings of visual objects. There will certainly be expansion in driving, surgical, and biological sciences applications, but we also expect to see growth in areas such as enhanced facial recognition\(^ {33}\) and oceanography,\(^ {34}\) and perhaps in the further future leveraging positive reactions to images to enhance artistic applications.

**Evolving Hardware**

The push to more versatile applications of neural networks will continue to bring advances in the development of specialized hardware for deep learning on resource-constrained (low power) and mobile devices.

The cellular market is diving headfirst into the specialized “AI” chip space with GPUs and multi-core processors on


phones. While applications thus far have generally focused on video and gaming uses, federated approaches to leveraging mobile resources and specialized deep learning model advancements are enabling more diverse applications. There are opportunities for image analysis portability in areas like insurance assessments, public safety, retail assistance, medical devices, virtual and augmented reality, and assistive technology for people with disabilities (see FF09: Federated Learning).

Intel, NVIDIA, and Google each produce small standalone devices with built-in image analysis capabilities for incorporation into other hardware. Though currently limited in the types of image analysis models they can run, and considered underpowered for general deep learning applications, these types of devices will gain more traction as both the models and the hardware gain efficiency—and with the spread of greater data connectivity via 5G.

Meanwhile, IBM is advancing a 4,096-core chip with a million neurons in configurable layers, scalable via the addition of more chips. The promise for use with deep learning and image analysis is high, and these restricted-availability and unique-language chips should support massive and more general neural networks. However, the commercial world outside of the efforts of DARPA and university cosponsors seems aimed at lighter-weight, tailored, and maximum-efficiency development. We expect these latter open efforts to be much more viable options for even large organizations.
“It’s really great to see you two again. And Sarah, I love that dress on you.” These words fell by way of a goodbye, and Sarah and Mark watched as Diana stepped gracefully into the crowd loudly laughing near the bar.

“I didn’t quite feel the sincerity in that exit,” Mark mused.

“Don’t be so dour — it’s an art exhibit and she’s legitimately one of the best critics in the world. I imagine she was happy to chat with us, if only for a few moments’ respite from cheek kisses and pretending to be happy to see everyone.”

“How do you know she was happy to see us, the ugly ducklings in this black-tie sea of swans?” Mark quipped, entertaining himself.

“Because we were friends with her before she lost her sight and gained fame with her new ‘vision.’ Besides, she knows the hors d’oeuvres won’t go to waste with us here," she added playfully.

“I still find it fascinating that she’s become a bigger success after the accident — a bit like Van Gogh pre- and post-ear.”

A wry smile passed over Sarah’s lips. “She explained it to me once. What she sees. And you’re right. It’s fascinating. The image processing in the multiple cameras she wears. The various perspectives integrated across dozens of chips, analyzing and sharing colors, textures, shapes, and structure.”

Remembering the conversation, Sarah continued, “It started with a question that was odd to hear from Diana — from anyone really — who loves art as she does. In her apartment after her physical therapy one night, she noticed me looking
at her Chagall. She asked, ‘What would it take to ruin that painting?’” Sarah’s head cocked slightly to one side, looking for Mark’s reaction. “I was confused and I’m sure I stuttered something about not understanding. I seriously thought she was thinking about damaging it, and that this was the first sign of her giving up on life post-crash.”

Her voice softened. “But she explained to me what she meant, that with her new eyes, the cameras, whatever you want to call them, she’s able to view every painting so incredibly precisely, in ways she never could with her human eyes. She can look at a new work and understand the alignment of each curve and brushmark, and make comparisons in a moment with every other work she has seen. She can know if the lines are coherent and in the style of a particular artist. Her new eyes can ask how photographic is the work, like a Vermeer, or how consistent are the strokes, like a Monet. And that’s just with surface details. When she asked about ruining the work, she meant that she can see every tiny detail and understands how much room for error there was in each brush stroke.” Sarah was beaming now, and her enthusiasm was drawing attention. A man who had overheard stepped closer, joining the conversation.

Glancing at the visitor, Sarah continued, “When Diana starts to look with her mechanical eyes at the art’s higher-level themes, she said she’s able to extract a richness so subtle it’s lost on most of us. It’s not just that she can see the colors and richness of a Kandinsky or a Pollock; those ‘eyes’ can process and understand the sentiment — the abstract horror — Picasso saw in the Guernica, and see that terror and pain repeated in a Goya or a Kahlo.”
“And she hasn’t left room for the rest of us,” the visitor tersely interjected. “She’s seen and reviewed all those works so authoritatively with her fake eyes, that the rest of our views barely carry any meaning anymore!” Exasperation poured through the speaker’s words. “My own critiques have become practically irrelevant.”

“Sir, I’m sorry. I don’t imagine she’s intentionally taking work to spite you, and everything in her ‘fake,’ as you say, eyes—the code, the art on which she has trained—is open and available for others to learn and build from. Have you asked her for help?”

“Ha, I’m old and have no plans to attach robots to my face—but I do know the beauty I have seen in art, and I also know what’s crap and what’s borderline for an artist right on the verge. And I see it better than she ever could before! Even if she’s sharing the technology, it’s not like I—or anyone else—could match what she does now.” Clearly frustrated, the stranger’s voice had sharpened.

“Hello, Charlie.” Sarah’s eyes widened as she watched the stranger turn to find Diana directly behind him.

“Diana, we were just speaking of you!” Charlie’s cheeks flushed the color of his wine.

“Ha,” Diana laughed. “I can imagine, Charlie. I’m glad you met these two—we go back almost as long as you and I do.” She paused. Mark waved.

“Charlie, we have always shared a passion for art. But I have to tell you, it was hard, really hard, coming back after my accident—even so-called friends here at this ‘gala’ could barely look at me without gasping. It was even worse in public... sneers, heads turning with disgust, pitying smiles, a few
hopeful smiles and beautiful faces of caring, and others of awe and terror. I really had to push myself back out into the world and I wouldn't wish the experience on anyone.” She glanced down and touched the side of her faceplate self-consciously before a small motor refocused her gaze back on Charlie.

“But I realized when I went back out into the streets where people could see me, and back into the libraries and museums where I’d lived my whole life — I saw people differently.”

“I learned from those reactions, Charlie. These eyes showed me what ‘body language’ really meant. I saw millions of variations of feelings dripping from thousands of real faces and half that number of painted ones and I would say I’m now the only person in the world who can tell you what the Mona Lisa is thinking, where the David is going next, and why Aphrodite covers herself so.”

A crowd had followed her, and gathered to listen in. Diana gestured toward them. “I think many of those in the art world have come to appreciate what I’m able to contribute.” Several nods came from the circle.

“But for all I can do with these eyes, though — so much else slips through the cracks. I need help. I need your help, Charlie.”

“My help? For what?”

“To see what I cannot. To help me figure out what else I should learn from. These eyes can only help me make assessments based on what they’ve seen before, whether masterful or crap — while your eyes, your real eyes, can sort the ones that are truly new and original and right on the edge of being interesting. Maybe something’s good enough... maybe not?
How close is the piece or the series to being great, what would it take to ruin it?"

She smiled — or seemed to do so around the eyes. "I’d love to work with you, Charlie. These machines can’t do everything yet."

84 Future
Deep learning for image analysis has been a game-changer for the evolution of modern technology. A growing portion of the world, from mobile phone security to social media and self-driving cars, now leans heavily on this tooling, and its value in the future is only going to increase. Academia and industry alike continue to push the limits of possibility for effective image recognition, classification, segmentation, and generation, creating new models and approaches to solving increasingly specific and differentiated application problems.

As noted with the ChypCon, Mazing Corp, and Ozzie Mines examples, "image analysis" is now about much more than the photo recognition algorithms of just a decade ago. It’s growing into a real business driver for quality assurance processes, automation, the increasingly online retail world, and so much more.

A key limiting factor for tailored model and data product development is still the relatively low availability of correspondingly specialized labeled datasets from which new models can learn. As shortcuts and efficiencies in these areas continue to evolve, an organization that invests in developing and maintaining data resources tailored to its specific business needs, while building a parallel internal data science capability to grow models’ effectiveness, will have a good
opportunity to step past the competition to become a leader in its field.

While similar things may be said of most applications of machine learning, the value of early investment in image analysis resources is greater because obtaining good labeled photo/video training data can be so much more resource-consuming than for other data types—and because, until that investment happens, competitors will be working from the same data and general models optimized for a different purpose or managed by someone else. Organizations don’t want to be caught out when the firms behind outsourced image analysis products change their terms of agreement, business plans, or pricing, after having grown themselves off those organizations’ work pipelines. Further, your company definitely won’t want to be among the last firms working only from general image models and public data when the rest of the industry has specialized.

The further future specialization of image analysis carries both concern and significant promise for data science and society. Certainly technologies such as augmented reality will grow past pure entertainment into near-horizon applications like infrastructure maintenance, production operations, and medicine. In the meantime, the field will be most publicly notable in the imminent conversations around “deepfake” news and influence, as well as real ethical challenges in public surveillance and other “security” and military/drone/robotics developments—particularly as image recognition and the supporting data pipelines are still far from perfect.

The field of image analysis does have an incredible opportunity to complement the advancing labor, work-efficiency,
and societal landscapes. We’re looking forward to so much more that can be creatively done with the technology in opportunity areas that are still somewhat fringe in the field, like agriculture, environmental quality, assistive technology for people with disabilities, refugee work, and voting. We’ll certainly continue to report on what develops in future updates and in conversations along the way.
About Cloudera Fast Forward Labs

Cloudera Fast Forward Labs is an applied machine learning research group. We help organizations recognize and develop new product and business opportunities through emerging technologies.


http://fastforwardlabs.com
Convolutional Neural Networks (CNN) excel at learning meaningful representations of features and concepts within images. These capabilities make CNNs extremely valuable for solving problems in domains such as medical imaging, autonomous driving, manufacturing, robotics, and urban planning. In this report, we show how to select the right deep learning models for image analysis tasks and techniques for debugging deep learning models.