VISION INSIGHTS

VISION SYSTEMS DESIGN®

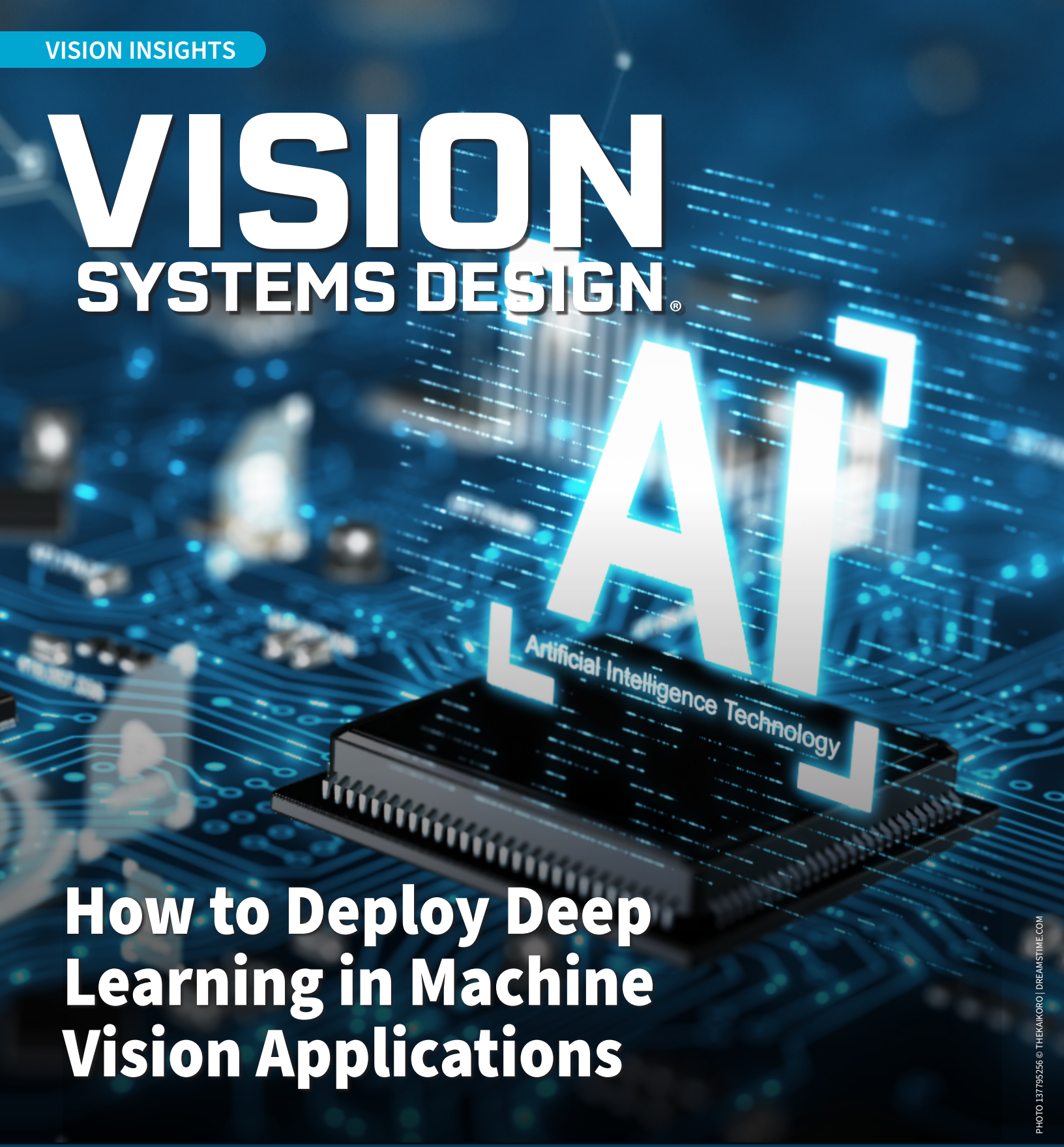

PHOTO 137795256 © THEKAIKORO | DREAMSTIME.COM

How to Deploy Deep Learning in Machine Vision Applications

How to Deploy Deep Learning in Machine Vision Applications

In this edition of Vision Insights, Vision Systems Design offers practical advice on building and deploying deep learning in machine vision systems.

To do so, we discuss what neural networks are and how they work, how to set up and train neural networks, and how to deploy them to automate product inspection.

Whether you work at an OEM, system integrator or end-user organization, we've provided insights here to help fuel your next project.

Linda Wilson,
EDITOR IN CHIEF
VISION SYSTEMS DESIGN

What is deep learning and how do I deploy it in imaging?

A subset of machine learning, deep learning offers promising enhancements in accuracy and efficiency in machine vision.

PERRY WEST

What is deep learning? Within the area of artificial intelligence (AI) exists machine learning, and within that, is the area of deep learning. Many different types of artificial intelligence exist. AI is any technique that imitates, at least in part, what a human would do with their mind. It is the ability of machines to mimic human behavior, and usually integrates perception, prediction, reasoning, and decision making, according to Yann LeCun, vice president and chief AI scientist at Facebook.

Deep learning (Figure 1) can be thought of as "programming with data." In a virtual machine setup with a neural network, the network is initially not programmed to do anything. When this virtual machine is given images, it begins to program itself to perform whatever the task is that is being set up. Deep learning can be used in imaging in many ways, including the recognition of objects in images, flaw detection, sorting and grading products, facial recognition, self-driving cars, and denoising images, among others.

Neural networks are modeled after the human brain. The brain consists of around 100 billion brain cells—neurons. Each neuron has dendrites (inputs), a nucleus, and axons (outputs). The connection between an axon of one neuron and a dendrite of another neuron is a synapse. The synapse contains a small gap separating the axon and dendrite, and when things are going right, there is dopamine in the synapse that strengthens the electrochemical connection the two neurons.

Something similar exists in a computer model of the neuron (Figure 2), with the nucleus and the dendrites (inputs) and the weights associated with each input, which represents the synapses—how tightly coupled the inputs are to preceding neurons. There are also the axons (outputs from the neurons). One important aspect of the computer simulation of a neuron is the activation function, which introduces non-linearity

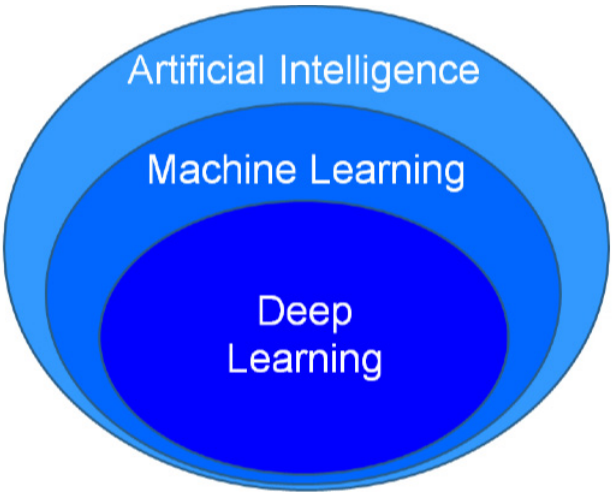

FIGURE 1: Deep learning exists within machine learning, which exists within the larger category of artificial intelligence (AI), which refers to techniques that imitate what humans would do with their mind.

to neural networks. Activation functions include the sigmoid, hyperbolic tangent, rectified linear unit (ReLU), and leaky ReLU functions.

The simple neural network proposed many years ago consists of three layers. The input layer did nothing but distribute inputs and had no activation function or weights. Then there was the hidden layer in the middle,

and the output layer. This was the minimal neural network architecture.

To use a neural network in deep learning, it must be trained to minimize errors over the training set. The training technique is backpropagation. When using the network, input data (images or features) advance through the neural network to reach the outputs. Network training involves starting from the output and propagating the error backwards through network (backpropagation). Network training uses the steepest gradient descent, which means using the derivative for the activation function and the chain rule in Calculus to move the errors back through the network regardless of the number of layers.

One learning control parameter, the learning rate, controls the magnitude of the weight adjustment based on the area and the steepness of the gradient. Starting with a very-high learning rate and trying to make big jumps in learning typically results in big weight changes causing the network to diverge from a solution and perform worse instead of better. When using a high learning rate, some learning takes place, but typically the network doesn't improve to the point of being useful. Setting the learning rate low results in a long training period. A learning rate that brings the error down as quickly as possible is ideal, but ultimately, developers must experiment and configure the system to find the best learning rate.

In convolutional neural networks (CNN), an input (image), first goes into layers performing convolution with the convolution weights set through training. The next layers use an activation function, usually followed by a pooling layer, where groups of four pixels reduce to one, thus reducing the overall size of the image and making it easier for the network to extract features. Pooling can be 2 x 2 to 1, it can be 3 x 3 to 1, and it can be the maximum, the average, or the median, but the most common pooling function involves getting one pixel from the maximum of four pixels.

In the CNN, an image goes through a number of two-dimensional convolutional layers and then is flattened to create a one-dimensional feature vector from the image. The feature vector runs through additional network layers to create classification outputs (Figure 3). The next step involves non-maximum suppression, an important post-processing step. If the neural network says cat 0.8 and dog 0.3, and the application only requires choosing between cat or dog, non-maximum suppression helps determine that—based on the difference in values—a cat is in the image. An issue presents itself when the network says cat 0.51 and dog 0.49. Here, the non-maximum suppression algorithm must be strong enough to render such values as indeterminate.

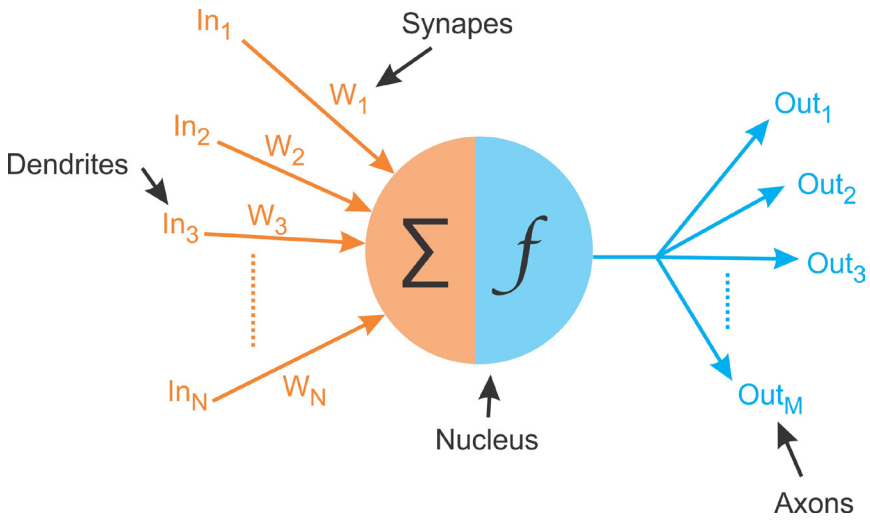

FIGURE 2: In a computer model of a neuron, there exists a nucleus, dendrites (inputs), weight associated with each input, and axons (outputs from the neurons).

Deep Learning Techniques and Implementation

Techniques for using deep learning in imaging include supervised learning, the most common method, where a system trains with labeled images and is then tested. If the testing results are adequate, the deep learning-based system is ready to use. Transfer learning methods involve using supervised learning along with a pretrained neural network. In this method, retraining is only required on the later layers in the network for the new application. When it is appropriate, transfer learning is much faster than training a network from scratch.

Unsupervised learning, or predictive learning, defines classes based on clustering of input data without labeling. Reinforcement learning uses a reward—positive or negative—for each output during training. The learning rate is much slower than supervised learning and requires much more data. Lastly, adversarial training involves the use of two neural networks working against each other.

The first step in implementing deep learning into an imaging system involves determining the types and numbers of layers, the number of inputs (usually the number of pixels in the input images), and the number of outputs (usually dependent on the number of classes required). Inputs, images, outputs, and labels for the images then go into a model under training. Training compares the network's outputs to the labels to generate an error signal which feeds back into the model under training to adjust the weights (coefficients), a process repeated until the error over all the training images reaches an acceptable minimum value.

When this happens, the network is tested using images not used for training to verify the network's outputs have a sufficiently low error. After a successful test, this neural network can be exported with the trained weights and becomes the model to use. At runtime, the model loads into an inference engine—a computer running the neural network—where the trained network operates.

Deep learning requires an abundance of labeled training data. This data will be split up into three classes: training data, test data, and validation data. Training data is used exclusively for network training, with each pass through the training data representing an epoch. After each epoch, the network's performance is evaluated using the validation data and the error on the validation data is tracked as an indication of how well training is progressing. After training—with the error on the training data sufficiently low—the network is tested using the test data to ensure acceptable results. Having a separate test set on which the network doesn't train enables developers to independently test a trained network—providing the only evidence that a network performs reasonably well.

Several methods for speeding up training exist, such as initially training with a subset of the training data. Such a method helps get adequate weights. Another option, transfer learning, involves taking a CNN previously trained on a similar problem and only training the last few layers of the network for the new application. With a similar enough network, the features extract the same, removing the need to train the convolutional layers—it is only necessary to train the last few layers from the feature vector to output.

Fine tuning—another method similar to transfer learning to speed up training—involves using a new, small set of training images with a trained network. To use fine tuning, the output layer of the CNN must be replaced, with training performed with a small training data set. Pre-trained networks for transfer learning and fine tuning exist, including VGG-16, ResNet 50, DeepNet, and AlexNet by ImageNet.

Several considerations must be made when implementing a deep learning system, including the fact that a fast system requires parallel processing.

A CPU can perform initial tests but will be very slow in training and execution. Useful tools for parallel processing include graphics processing units (GPU), field-programmable gate arrays (FPGA), and digital signal processors (DSP). Chips such as the Apple A12 bionic chip, which has two CPUs and a GPU, plus some additional circuitry specially designed to execute deep learning networks, as well as special processing chips like the Intel Movidius and Intel Nervana, also provide processing capabilities.

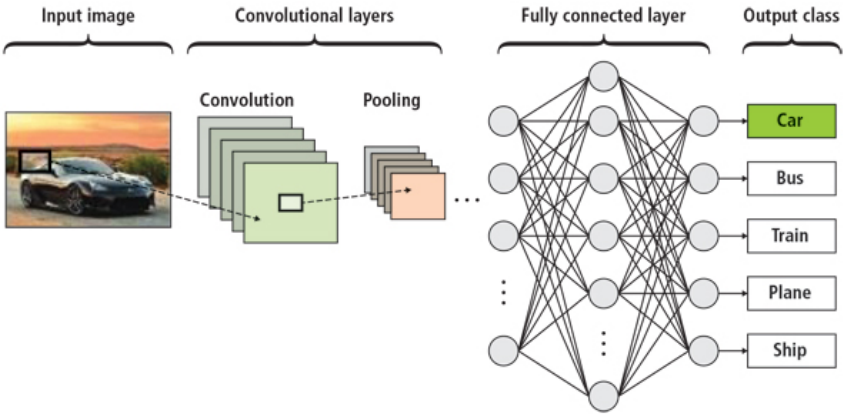

FIGURE 3: In CNNs, convolutional layers are used to perform feature extraction, just as convolution operators are used to find features such as edges. In conventional image processing, image filters such as Gaussian blurring and median filtering perform this task. CNN architectures, on the other hand, emulate the human visual system (HVS) where the retinal output performs feature extraction such as edge detection. (Figure courtesy of MIT).

Data must also be considered, including data for training, validation and testing. More training data means better results. Implementing a successful deep learning system necessitates a lot of time and effort spent on gathering and labeling data.

Major Challenges and Pitfalls to Avoid

One of the major challenges in deep learning involves real-time processing of many pixels, which means a lot of data, which requires a lot of computational power in both training and execution. This necessitates a very large set of pre-labeled, training images, but in some cases, labeled training datasets can be purchased. Another problem exists with the possibility of mislabeling of training data. Labeling just one image wrong results in degradation in performance of a trained neural network.

Bias in the training data represents another possible pitfall. More samples of one class than the other leads to difficulty in achieving good results on the classes with fewer examples. Data storage is another factor, including the image format and image storage options. When storing images on disks, it takes a long time for retrieval. Solid-state disks become preferable, but if there are a million RGB, megapixel images, storage requirements become enormous.

In using machine learning without deep learning, network inputs won't be the images, but features extracted from the images. Selected features must be normalized in such scenarios. With one feature going from 0 to 1 and another feature that goes from 0 to 100, the latter dominates. Normalizing scales the features, ensuring that they all fall within the same range. Otherwise, the network will fail to meet performance expectations.

Data augmentation—a method for reducing the cost of gathering and labeling the data—improves network reliability. Augmenting a picture of a cat by creating several other pictures from it with the same label represents one example. Further methods include rotation, scale, shear, obscuration (e.g. haze), color change, flip/mirror, crop, addition of noise, and varying the background scene.

Choosing the right network configuration poses potential problems. If a network is too complex, training time will be longer than necessary and the network may not generalize the features that it needs to recognize and may perform poorly.

One option to deal with long training time is renting cloud processing time from Microsoft or Amazon for deploying deep learning applications, as the cloud pulls in many processors and GPUs to get training done more quickly. Determining the initial conditions of the network also creates a challenge, as does the selection of the learning rate and training parameters.

Pragmatic challenges in deep learning exist as well, including the input of an image during execution that is not representative of any training input (an anomaly). Security must also be considered, in terms of what would happen when a network fails to give the right answer or if someone hacks into the network.

Philosophical challenges must be considered as well. CNNs lack creativity or abstract thinking, have no intuition, and cannot synthesize input from a specified output, unlike people. For example, it is not possible to tell a network "I want to recognize a cat, what do I look for?" Other questions to consider are: If the network fails, who is responsible, and will they be able to explain why?

Supervised learning and reinforcement learning are not sufficient for real world problems. Unsupervised learning is required, where the network can not only recognize the cat and the dog, but also look at a picture of a camel and determine that it is not a cat and not a dog.

Several methods for improving efficiency in deep learning exist, including experimentation with floating point weights and subsequent reduction to integer (fixed point) weights, which leads to speed increases but a slight decrease in accuracy. Using binary weights (0 or 1) also leads to a further increase in speed, but a decrease in accuracy.

In a network with lots of branches with weights, some of these weights become infinitesimally small. Pruning these weights out is possible, with only a small decrease in accuracy. Transfer learning can also lead to a significant reduction in training time.

Tips for Successful Implementation

Regarding the successful implementation of a deep learning system, here are several tips to consider. First, watch out for overfitting, which happens when a neural network essentially "memorizes" the training data. Overfitting results in great performance on training data, but the network's model becomes useless for out-of-sample prediction. Larger networks are more powerful, but it becomes easier to overfit. A system trying to learn a million parameters from 10,000 examples is not ideal. If parameters are greater than examples, this leads to trouble. More data is almost always better because it helps fight overfitting.

Validation—and one should never train on the validation or test data—really helps when the network isn't behaving right, and validation doesn't track the testing results. When this happens, stop and make changes to the network rather than continuing with disappointing results.

Furthermore, train over multiple epochs. With learning rates of 1/10th or 1/20th, many passes are required to get the gradient down to the minimum or near the minimum. Lastly, stacking layers can help as well. The first layer can be a convolutional layer or multiple convolutional layers doing different things.

When approached with great attention to detail and an understanding of deep learning architectures, functionality, and system requirements, the technology offers great enhancements in efficiency and accuracy in machine vision processes of all kinds. Additionally, breakthroughs should coming in network topology, in ways to speed up training, in ways to reduce the number of labeled images needed to train, and even in new types of networks for learning.

Perry West *is the founder and president of Automated Vision Systems, Inc. (San Jose, CA, USA).*

Five steps for building and deploying a deep learning neural network

Accelerating machine vision implementation with deep learning is nothing to fear.

BRIAN CHA

Free tools and training data, easy-to-find tutorials, and low hardware costs have made deep learning no longer a method available only to researchers or people with highly specialized skills and/or big budgets.

This presents both opportunities and threats as new players emerge to disrupt established names and spur innovation. It also provides opportunities for a machine vision system to do things previously unimaginable. Deep learning can recognize unexpected anomalies, typically very difficult or almost impossible to achieve with traditional rules-based coding, for example.

Deep Learning Fundamentals

Deep learning, a subset of machine learning inspired by how the human brain works, takes place in two stages: training and inference.

The training phase involves defining the number of neurons and layers that will comprise the neural network and exposing the network to labeled training data, usually good images of objects that would pass inspection and bad images of objects that would fail.

The neural network then figures out properties of each grade, such as size, shape, color, consistency of color, and so on. Manual definition of these characteristics or programming the parameters of a good or bad product are not required. The neural network trains itself.

In the inference phase the trained neural network, when presented with fresh images, will provide an inference as to the quality of the object and the neural network's confidence in its assessment.

Step 1 - Identify The Appropriate Deep Learning Function

Four of the most common deep learning tasks include classification, detection and localization, segmentation, and anomaly detection.

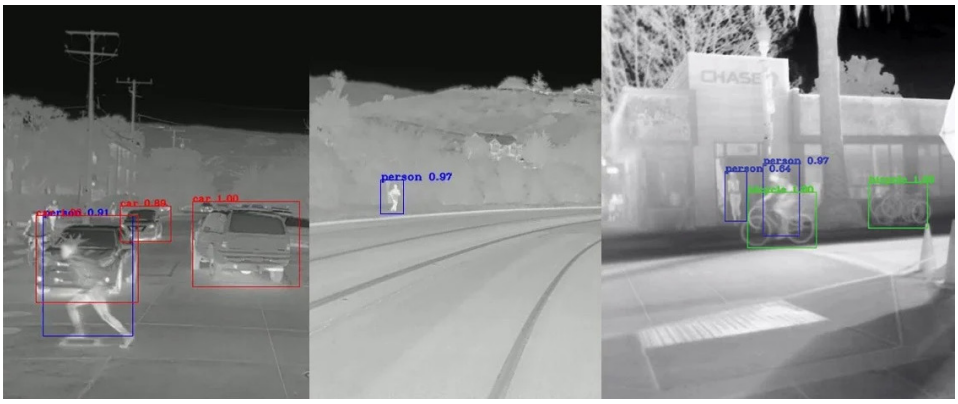

FIGURE 1: Advanced driver assistance systems are appropriate for deep learning segmentation routines that locate and identify objects, for instance pedestrians and other vehicles.

Classification involves sorting images into different classes and then grouping images based on common properties, most often into categories of pass and fail. Any item classified pass continues down the production line. An item classified fail does not.

Detection and localization can identify features in an image and draw a bounding box around those features

to determine their position and size. This function can provide a more detailed assessment of why an item deserves a fail classification, for example, by detailing the location of the fault.

Segmentation identifies which pixels in an image belong to which corresponding objects, to determine the context of an object and its relationship to other objects. Advanced driver assistance systems (ADAS) use segmentation routines to identify cars, street signs, or other objects while the car moves.

Anomaly detection functions can identify regions on an image that do not match a pattern. For instance, a deep learning system could process an empty shelf in a grocery store as an anomaly compared to nearby, full shelves, and mark the empty shelf as requiring a restock.

Step 2 - Select a Framework

A framework, or a toolset used to develop a neural network, usually includes a starter neural network and tools for training and testing the network. Free, easy to use frameworks like PyTorch, TensorFlow, and Caffe2 provide great documentation and include examples to allow novice users to train and deploy neural networks with minimum effort.

PyTorch (https://pytorch.org/), an open source solution now part of Facebook (Menlo Park, CA, USA; www.facebook.com), is simple and easy to use and employed in many research projects, but not commonly used for large deployments and only fully supported for the Python programming language.

TensorFlow (www.tensorflow.org) by Google (Mountain View, CA, USA; https://about.google/) has a large userbase supported with good documentation. It offers scalable production and deployment and supports mobile deployment. It has a higher learning curve compared to PyTorch, however.

Caffe2 (https://caffe2.ai/) by Facebook, a lightweight option, translates to efficient deployment. One of

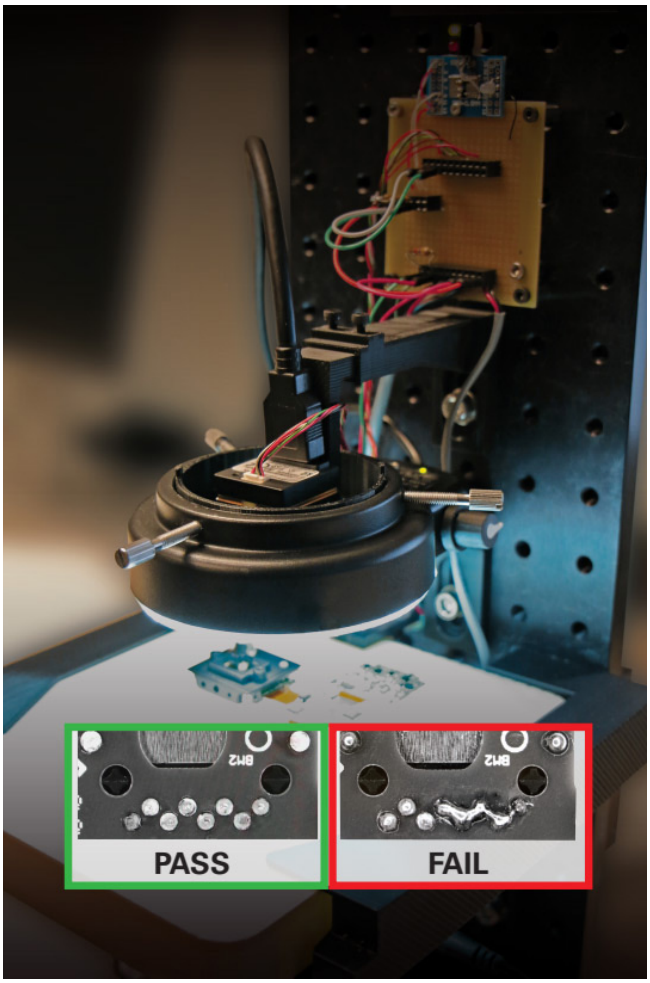

FIGURE 2: A deep learning system, once trained to recognize what a good image looks like (left), can identify objects with defects as bad images (right) and designate a failed inspection.

the oldest frameworks, Caffe2 has widely supported libraries for convolutional neural networks and computer vision applications and is best suited for mobile devices using OpenCV.

The optimal framework for a task ultimately depends on complexity and required inference speed. The more layers a neural network has, the slower the inference.

Step 3 - Preparing Training Data for the Neural Network

The number of images required for training depends on the type of data a neural network will evaluate. Generally, every characteristic and every grade of that characteristic

the neural network must assess, requires a set of training images. The more images provided for each category, the more finely the neural network can learn to assess those categories.

For common use cases, free or purchasable pre-labelled datasets that match specific requirements may exist online. Companies such as Cvedia (Arlington, VA, USA; www.cvedia.com) can create synthetic datasets annotated and optimized for neural network training. In the absence of other options, self-produced and -labeled images may need creating. Turning a single image into many images through rotating, resizing, stretching, and brightening or darkening can save time.

Several developers in the deep learning market open source their image labeling solutions and share them for free. LabelImg (bit.ly/VSD-LBMG), particularly useful for unlabeled datasets, provides a graphical image annotation tool that helps label objects into bounding boxes within images. Alternatively, third parties can handle the labeling process. Preparing training data can become even more important in light of specific hardware limitations or preferences, as some deep learning tools support only a finite set of hardware.

Step 4 - Train and Validate the Neural Network to Ensure Accuracy

This stage involves configuring and running the scripts on a computer until the training process delivers acceptable levels of accuracy for a specific use case. Separating training and test data ensures a neural network does not accidentally train on data used later for evaluation. Taking advantage of transfer learning or utilizing a pre-trained network and repurposing it for another task, can accelerate this process. A neural network already trained for feature extraction, for example, may only need a fresh set of images to identify a new feature. Frameworks like Caffe2 and TensorFlow provide pre-trained networks for free.

In the absence of coding expertise for neural network training, several graphical user interface-based software options exist, like Matrox Imaging Library X, or MIL (bit.ly/VSD_MILX) from Matrox Imaging (Dorval, QC, Canada; www.matrox.com) which work with different frameworks and make the training and deployment process very intuitive, even for less experienced users.

Step 5 - Deploy the Neural Network and RunI Inference on New Data

The last step entails deployment of a trained neural network on the selected hardware to test performance and collect data in the field. The first few phases of inference, ideally used in the field to collect additional test data, may provide training data for future iterations.

Cloud deployment offers significant savings on hardware cost and the ability to scale up quickly and deploy and propagate changes in several locations. Internet connection issues can cause critical failures, however, and cloud deployment has high latency compared to edge deployment.

Edge deployment on a highly customizable PC suits high performance applications. Selected PC components may fit a specific application, which makes pricing flexible. Edge deployment still has a higher cost than other options and the footprint of the needed hardware requires consideration.

Edge deployment on ARM, FPGA, or inference cameras like the Firefly DL camera from FLIR Machine Vision (Arlington, VA, USA; www.flir.com) requires less power than other options, offers savings in peripheral hardware, and has high reliability. This creates a secure system isolated from other hardware, or an ideal compact application, but may not handle computationally demanding tasks effectively

Potential Shortcomings of Deep Learning

Deep learning, a black box for the most part, can make explaining how a neural network arrives at its decisions difficult to illustrate. While inconsequential for some applications, companies in the medical, health, and life sciences field have strict documentation requirements for the product approval by the FDA or its counterparts in other regions. Full awareness of how deep learning software functions and potential requirements to document the entire operation in fine detail are necessary in some cases.

Optimizing a neural network in a predictable manner may present an issue. Many neural networks take advantage of transfer learning to retrain existing networks, while very little optimization occurs.

Even minor errors in labeling training data can throw off the accuracy of the neural network. Debugging the problem becomes extremely tedious, if review of all training data individually to find the incorrect label becomes necessary.

In addition to these shortcomings, logic-based solutions better suit some applications. For instance, logic-based solutions may provide better results for a well defined, deterministic, and predictable problem compared to deep learning-based solutions. Typical examples include barcode reading, part alignment, and precise measurements.

Conclusion

Even with some of the shortcomings, for certain applications the potential benefits accrued from deep learning like rapid development, ability to solve complex problems, and ease of use and deployment, outweigh the negatives. Deep learning also continually improves to account for these shortcomings.

Also, with wider adoption many companies now develop their own neural networks instead of relying on transfer learning which improves performance and customizes the solution for a specific problem.

Even in applications well-suited for logic-based programming, deep learning can assist the underlying logic to increase overall accuracy of the system. As a parting note, it's getting easier and cheaper than ever before to get started on developing a deep learning system (bit.ly/VSD-DLCS).

Brian Cha *is a technical product manager at FLIR Systems, Inc. (Arlington, VA, USA)*

Designing effective traditional and deep learning based inspection systems

When best practices are followed, machine vision and deep learning-based imaging systems are capable of effective visual inspection and will improve efficiency, increase throughput, and drive revenue.

DAVID L. DECHOW and **ANDREW NG**

For decades, machine vision technology has performed automated inspection tasks—including defect detection, flaw analysis, assembly verification, sorting, and counting—in industrial settings. Recent computer vision software advances and processing techniques have further enhanced the capabilities of these imaging systems in new and expanding uses. The imaging system itself remains a critically important vision component, yet its role and execution can be underestimated or misunderstood.

Without a well-designed and properly installed imaging system, software will struggle to reliably detect defects. For example, even though the imaging setup in Figure 1 (left) displays an attractive image of a gear, only the image on the right clearly shows a dent. When best practices are followed, machine vision and deep learning-based imaging systems are capable of effective visual inspection and will improve efficiency, increase throughput, and drive revenue. This article takes an in-depth dive into the best practices for iterative design and provides a roadmap for success for designing each type of system.

Is Your Imaging System Good Enough?

Lighting, optics, and cameras comprise an imaging system, and these components must be carefully specified and implemented to ensure high-quality parts images. "High quality," in this context, means images with sufficient contrast to highlight unacceptable features (such as dents) compared with those with a normal or expected appearance. The images must also have adequate resolution to show differences between features.

If a human inspector examining an image produced by an inspection system cannot confidently identify a

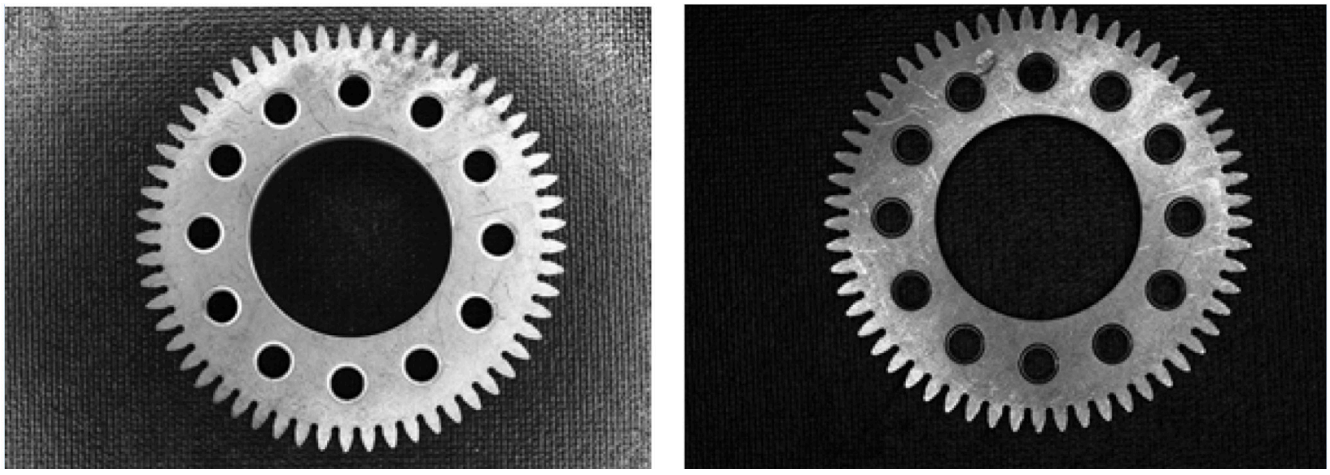

Figure 1: While the image on the left shows a flawless gear, the image on the right clearly shows a defect. *(Photos courtesy of Landing AI.)*

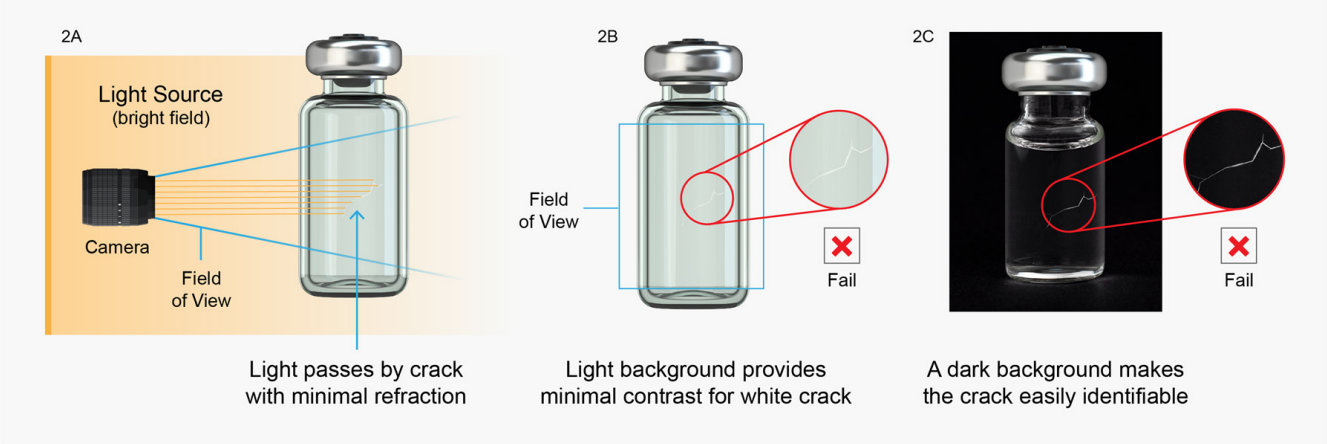

Figure 2: The image on the left illustrates a challenging detection problem as compared with the image in the middle, while the image on the right shows the defect even more clearly.

defect, it's unlikely the software will be able to either. Conversely, in circumstances where a human inspector can identify a defect in an image, there's no guarantee that an imaging technique will produce reliable and repeatable detection of similar target defects during operation. Situations that indicate an imaging system (and not the software) needs work include:

- An inspector looking at a physical part can reliably judge that something is defective but can't be sure when looking only at the image captured.
- Two inspectors looking at a physical part generally agree in their assessment, but an inspector looking at the physical part often disagrees with a different inspector looking only at the image.

One common misconception is that if a human inspector can see a feature with the naked eye, an imaging system can be designed to produce an image that successfully captures the same feature. But, a human inspector can view a part from multiple orientations and under different lighting conditions to make a quality judgment, while a static imaging system cannot necessarily capture a similarly large range of orientations and illumination variations. Therefore, it might have trouble highlighting features that a human inspector holding the same object would highlight. And, in cases such as detecting scratches in transparent parts, the imaging system's challenges might become even more complex.

Over millennia, the human visual system has evolved to be very efficient and accurate at processing image data. Building a software system capable of beating a person at processing images is an incredibly difficult task, as is building a software system that can detect defects that an inspector cannot. Even the most advanced vision system is not magic. If given a sufficiently blurry and fuzzy image, no vision system can reliably make a defect judgment.

Traditional Imaging System Design Checklist

Systems integrators and OEMs must consider several factors when designing an effective imaging system. These factors include:

Contrast: Creative use of dedicated illumination and optics specifically selected for an application and the types of features that produce contrast represent an important element in machine vision.

Spatial resolution: Spatial resolution in an imaging system refers to the number of pixels that span a feature, such as a defect. With too few pixels, it is impossible to reliably detect the feature relative to the

part surface. Assuming that an image is well focused, we recommend having at least a 5-pixel width of the smallest defect the system is expected to detect.

Image consistency: In an automated process, many factors can cause variation in an image, including part positional variation and variation in the parts themselves. In some scenarios, these variations can cause glare or dropout from the illumination source that obscures features. In other cases, part variation might cause reflections that could be mistaken for flaws or defects. If a machine vision system inspects a transparent automotive headlight for defects, for example, different lighting conditions will produce different amounts of glare. The more the system can capture images from the same angle, with the same lighting and the same background, the easier it is to build software to detect defects.

Exposure: Over or underexposed images lose a lot of fine detail. The appropriate level of exposure should allow the system to capture clear images of defects.

The Iterative Process of Designing an Imaging System

Specifying an imaging architecture for a machine vision system is just one critical step in the overall integration process. Successful automation vision system integration requires thorough and competent analysis and planning prior to component design and specification, followed by efficient installation, configuration, and system start-up.

Software must be considered during imaging system design as well. In some cases, an image that will be used with traditional rules-based machine vision algorithms might be different from an image that would be appropriate for a system using deep learning algorithms. Figure 2 *(left)* shows a significantly more challenging detection problem than the better illuminated and lit image in Figure 2 *(middle)*. The darker backdrop in Figure 2 *(right)* illuminates the defect even more. In this case, better image design would make implementation of either inspection system significantly more reliable.

Designing an imaging system is a highly iterative process; the best machine vision solutions evolve and grow more reliable and stronger over time. Designing a system around the "perfect" lighting and camera and then building it in advance may not be possible. But with thorough analysis of an application's needs — along with some knowledge of imaging components and techniques—developers can produce a good initial design.

When developing software systems, an integrator or OEM should collect sample images—even with a smartphone camera in the first few days—to get initial data to validate the feasibility of the software. Whether this proof of concept produces positive or negative results, bear in mind that a separate, production-ready imaging system must be designed. A smartphone camera's capabilities, such as quickly moving to multiple angles, might not be feasible in a production system. Working with sample parts with representative defects using a static imaging setup may work, but the final imaging system configuration must still be considered.

Testing software with "perfect" images might not truly represent actual capability in the production setting. When designing a production-ready imaging system, a thoughtful design will produce longer-term success. In a typical process, one should:

- Develop a specification for the types of features/objects/defects to be imaged, considering the automation and handling limitations of the part. Considerations might involve fast-moving parts, parts that change in appearance based on viewing orientation, and parts that show glare.
- Collect defective and acceptable part samples.
- Design an initial imaging system that meets the

needs of the parts to be detected and the physical constraints and specifications of the production environment.
- Run the sample of parts through the system and check that all defects are imaged clearly in a way that will be suitable for the targeted software solutions.
- Iterate steps 3 and 4 until performance is satisfactory.

Similar steps must be made when developing a deep learning-based imaging system or implementing deep learning capabilities into an existing machine vision system, with the exception of some key considerations. The next section provides a plan for getting started with deep learning in your imaging system.

Deep Learning Development Checklist

In several scenarios, discrete analysis-based machine vision algorithms may not suffice. These include semiconductor and electronics inspection, steel inspection, welding inspection, and any other inspection task where defects may be hard to find or where the appearance of "good" parts or items varies. Developing a deep learning software solution may be similar to building a traditional rules-based system, with the exception of some key considerations. These include:

*Deploying clean data***:** "Garbage in, garbage out" is the saying. Data represent the food that nourishes an artificial intelligence (AI) system, so it is imperative that quality data are used to train a deep learning model. Even the most well-conceived model produces subpar results when consuming inaccurate or incomplete information. A quality deep learning software solution should continuously collect data while allowing the data and each software component to be systematically developed, deployed, tracked, maintained, and monitored with tools that help developers access and control AI model evolution. The data should include information on products, defects, labels or tags, data consistency, and associated models.

Defining defects: In many industrial settings, companies that rely on human inspectors often keep a written log of defined part defects. In training a deep learning system, these defects must also be defined up front so that the software can recognize a defective part.

Tagging and labeling: Companies looking to deploy deep learning must accurately label and tag data. When done inconsistently, this step can lead to inaccurate AI models. With clear defect definitions and clear, unambiguous labels on a representative data set, companies can proceed with visual projects with small amounts of data. Internal experts must collaborate to assign, manage, execute, and review tasks to ensure quick and accurate labeling to produce more accurate models.

Iterative improvement: The best AI models should be evaluated against expert human inspectors to prove value before deploying to a production line, especially if the line serves as a test for global deployments. Deep learning software should have tools for evaluating a model's performance, identifying data that can result in losses in model accuracy, and evaluating new data sets to improve and expand existing models to reach success metrics. The software should also feature tools to prevent overfitting and to evaluate the performance of a trained model.

Common Pitfalls and Challenges

Imaging presents many challenges, so systems integrators and OEMs should consider a few of the most fundamental and elementary pitfalls and address these up front in system design. These include:

Ambient light: Illumination from sources other than the dedicated lighting components designed for an imaging system is considered ambient and can

introduce inconsistency and failure into the system. Sunlight and even overhead illumination must be controlled either through shielding or optical filtering where possible. In one example, a change in color of the uniforms of manufacturing personnel near the inspection system caused additional reflected light that impacted inspection results. In most cases, it is relatively simple to mitigate ambient light in imaging system design.

Mechanical stability: Factory vibration can shake the optics in imaging systems loose, and changes in camera position, lighting components, or even lens settings can cause unreliable imaging.

Varying appearances: Materials, design, and overall appearance of the parts being inspected may change, and without the vision system owner being aware of them. For example, a manufacturing engineering team decides to change the metal alloy on a screw because it's cheaper. Functionally, the part will work the same, but the appearance may be altered. Such external influences can cause a system's performance to degrade, sometimes silently. Software that checks for this drift can signal to the operations team when to carry out vision system maintenance in a timely way.

Machine Vision and Deep Learning Evolved

A visual inspection system, whether traditional or deep learning-based, can help industries and companies of all types keep up with customer demand while ensuring product quality, improving productivity, and bringing down costs. Whether you are a company looking to automate more processes or an integrator or OEM facing the specification, design, and installation of your next system, consider the fact that all visual inspection systems require testing, iteration, and continuous improvement.

Following best practices and considering contrast, spatial resolution, image consistency, and exposure will aid in the design of an effective imaging system. On the deep learning side, factoring in the need for clean data, agreement-based labeling, tagging and labeling, and iterative model improvement will help produce a high-quality AI visual inspection system. With ongoing improvement, your visual inspection system will continue to add value and allow your business to grow into the future.

David L. Dechow *is owner of Machine Vision Source (Salisbury, NC, USA) and Andrew Ng is the CEO and founder of Landing AI (Palo Alto, CA, USA).*