Applied Intelligence (2023) 53:15923–15945
https://doi.org/10.1007/s10489-022-04278-6

Front-end deep learning web apps development and deployment: a review

Hock-Ann Goh[1] · Chin-Kuan Ho[2] · Fazly Salleh Abas[1]

Accepted: 17 October 2022 / Published online: 30 November 2022

Abstract

Machine learning and deep learning models are commonly developed using programming languages such as Python, C++, or R and deployed as web apps delivered from a back-end server or as mobile apps installed from an app store. However, recently front-end technologies and JavaScript libraries, such as TensorFlow.js, have been introduced to make machine learning more accessible to researchers and end-users. Using JavaScript, TensorFlow.js can define, train, and run new or existing, pre-trained machine learning models entirely in the browser from the client-side, which improves the user experience through interaction while preserving privacy. Deep learning models deployed on front-end browsers must be small, have fast inference, and ideally be interactive in real-time. Therefore, the emphasis on development and deployment is different. This paper aims to review the development and deployment of these deep-learning web apps to raise awareness of the recent advancements and encourage more researchers to take advantage of this technology for their own work. First, the rationale behind the deployment stack (front-end, JavaScript, and TensorFlow.js) is discussed. Then, the development approach for obtaining deep learning models that are optimized and suitable for front-end deployment is then described. The article also provides current web applications divided into seven categories to show deep learning potential on the front end. These include web apps for deep learning playground, pose detection and gesture tracking, music and art creation, expression detection and facial recognition, video segmentation, image and signal analysis, healthcare diagnosis, recognition, and identification.

Keywords Deep learning web apps · TensorFlow.js · Front-end deep learning · Browser-based deep learning · Client-side deep learning

1 Introduction

Since the early 2010s, the discipline of deep learning has made astounding progress, resolving previously unsolved issues and generating intriguing new possibilities. Deep learning development is fueled by algorithmic breakthroughs enabled by a stack of open source tools, hardware advancements, greater availability of labeled data, and algorithmic advancements [17].

Deep learning (DL) models can be deployed on various platforms for real-world use after thorough validation and testing. One approach is to host the DL model on a server or in the cloud. This method allows developers to launch DL-enabled services by contacting API endpoints. Frameworks (for example, TF Serving[1]) and cloud platforms (for example, Google Cloud Machine Learning Engine[2]) may

Chin-Kuan Ho and Fazly Salleh Abas contributed equally to this work.

✉ Hock-Ann Goh
hagoh@mmu.edu.my

Chin-Kuan Ho
ckho@apu.edu.my

Fazly Salleh Abas
fazly.salleh.abas@mmu.edu.my

[1] Faculty of Engineering and Technology, Multimedia University, Jalan Ayer Keroh Lama, Bukit Beruang, 75450, Melaka, Malaysia

[2] Asia Pacific University of Technology and Innovation, Jalan Teknologi 5, Technology Park Malaysia, 57000, Kuala Lumpur, Malaysia

[1] https://www.tensorflow.org/tfx/guide/serving

[2] https://cloud.google.com/gcp

aid with this deployment. Apart from online cloud frameworks, the model can also be served on a smaller scale, using specialized back-end technologies such as Django and Flask. However, since these deployments need a cloud service or a dedicated computer to execute the model, these back-end deployment methods are prohibitively expensive.

The introduction of deep learning libraries that specifically cater to client-side deployment has changed the landscape [78]. Previously, it was believed that computation-intensive deep learning models could only be effectively run on platforms with GPU support and that they could not be successfully deployed on browsers or mobile platforms owing to the client-side platforms' limited processing power. However, this has changed. With the front-end option becoming available, the deployment process has shifted its attention to adapting deep learning models to the front-end platform.

According to a study on the challenges of deep learning deployment [21], the query about deployment has grown in popularity, based on relevant postings from 2015 to 2019 on Stack Overflow, a developer-focused Q&A website. The questions about front-end deployments only surfaced in 2018 and remain relatively few. By 2019, the volume of users and queries about front-end deployment was only a quarter of back-end and mobile deployment. The researcher concurred with Ma et al. [78] that deep learning in browsers is still in its infancy. According to the study, it takes approximately 3 times longer to answer deployment questions than non-deployment questions (404.9 min vs. only 145.8 min). These findings show that issues surrounding the deployment of deep learning software are difficult to address, partially confirming a prior result in [4] that model deployment is the most challenging step of the machine learning life cycle. While deployments are the most time-consuming to resolve, browser-based or front-end deployments seem to have faster median response times. The median response time for server/cloud, mobile, and browser-related queries, respectively, was 428.5, 405.2, and 180.9 min [21]. These findings suggest that the front end may be an excellent choice for deep learning deployments.

This paper aims to discuss and review the deployment stack for front-end deep learning and the development process that specifically targets this deployment. No other study has explicitly addressed and reviewed deep learning web apps development and deployment for the front end. In [29], however, a review of deep learning on mobile devices has been provided. In [109], a series of 52 review papers on deep learning are presented. The papers are grouped into topics, such as 'computer vision, forecasting, image processing, adversarial cases, autonomous vehicles, natural language processing, recommender systems, and big data analytics'. These review papers focused on particular topics and not on model deployment. To close the gap, this paper reviews the development and deployment of deep learning models on the front-end, client-side browser.

Due to the length of content, Table 1 presents the paper's organization and section summary, which helps the reader's navigation and highlights key points.

2 Front-end deep learning deployment stack

Machine learning was previously restricted to only experts in research laboratories; today, it is widely accessible. Now, it is much simpler for machine learning experts and others to get started by leveraging popular front-end technologies. This section goes through the reasoning for developing real-world deep learning apps using the browser, JavaScript, and TensorFlow.js as the front-end stack. These technological stacks are built on top of one another to form an ecosystem for front-end deep learning deployment.

2.1 Why not deploy deep learning on the client-side browser?

Once trained, the machine-learning model should be deployed to generate predictions on actual data. It is meaningless to train a model if it will not be used [17]. Based on the benefits mentioned later, the front-end may be a viable deployment option.

The conventional approach is deploying a deep learning model on the server backend and accessing it through complicated API calls sent via HTTP requests using JavaScript. Because of their semantics and requirements, correct and efficient use of machine learning APIs is challenging. Accuracy-performance tradeoffs must also be carefully considered. In [123], a research study of 360 apps that use Google or AWS cloud-based ML APIs was performed. It was found that 70% of these applications include API misuses that impair the functional, performance, or economic quality of the program.

The mobile platform is also a potential deployment platform. However, creating mobile AI apps that are cross-platform compatible is not simple. It is incredibly challenging to develop and maintain versions for iOS and Android, as well as test and submit the mobile apps for distribution on an app store [78]. A taxonomy of 23 types of fault symptoms is given in [22], showing that deploying deep learning models on mobile devices covers a wide variety of problems.

However, with advancements in front-end development, researchers now have the option of deploying on the front-end, which is much easier [21], thus migrating away from back-end data centers and toward clients. Front-end deep learning applications offer the following advantages: *wide*

Table 1 Paper organization, descriptions and highlights

Section 1.	Introduction
	Section 1 introduces front-end deep learning deployment as a viable alternative to back-end deployment. It also outlines the objective and highlights the content of the paper
Section 2.	Front-end deep learning deployment stack
	Section 2 discusses why the ideal technology stack for front-end deep learning deployment includes browsers (front end), JavaScript, and TensorFlow.js. It also includes the constraints and considerations to be taken when adopting the stack
	Section 2.1 Why not deploy deep learning on the client-side browser?
	Section 2.2 Why is JavaScript the language of choice?
	Section 2.3 Why is TensorFlow.js ideal for front-end deployment?
Section 3.	Front-end deep learning development approach
	Section 3 covers the most commonly used approaches to build deep learning models suitable for front-end deployment, namely reusing, customizing, converting, and optimizing. Constraints and considerations for each approach are also highlighted here.
	Section 3.1 Reusing pre-trained TensorFlow.js models
	Section 3.2 Customizing existing models using transfer learning
	Section 3.3 Converting Python models into TensorFlow.js models
	Section 3.4 Optimizing the model prior to deployment
Section 4.	Front-end deep learning web apps
	Section 4 provides samples of research work that leverages front-end deep learning web apps for development and deploying their work. These web apps are grouped into seven research categories
	Section 4.1 Deep-learning playground apps
	Section 4.2 Body pose detection and gesture tracking apps
	Section 4.3 Music and art creation apps
	Section 4.4 Facial expression detection and recognition apps
	Section 4.5 Video, image, and signal analysis apps
	Section 4.6 Scientific computation, diagnosis, and optimization apps
	Section 4.7 Classification and detection apps
Section 5.	Conclusions
	Section 5 wraps up the paper with a summary of the key points presented in the paper

reach, *reduced server costs*, *timely response*, and *privacy* [17, 111].

Wide reach The web environment is the most commonly used deployment platform for applications, with a growing number of customers expecting to use machine learning web applications while surfing the web [16].

Reduced server cost Server cost is often an issue in deep learning deployments. Accelerating deep-learning models often requires GPU acceleration. Models implemented on Google Cloud or Amazon Web Services virtual machines with CUDA GPUs may be costly. The cost increases with traffic, as does scalability and server architectural complexity. A simplified deployment stack using the front-end lowers server expenses and developer concerns. For example, in [13], the front-end deployment method was selected over cloud deployment to save server costs.

Timely response Running from the client-side is essential for low latency applications, such as real-time audio, image, and video data. Consider what happens if an image needs to be sent to the server for inference. With client-side inference, the data and processing remain on the device, reducing latency and connection issues. Due to the proximity of data, the front-end browser is the perfect platform for interaction and visualization while working with deep learning applications. In [19], for example, a real-time and interactive online experience is provided by creating models on the front-end, avoiding the size and latency limitations associated with conventional neural audio synthesis models.

Data privacy Data privacy is a must for certain applications, such as health-related deep learning models inferred from medical data. Using just the client-side to execute the model effectively addresses this issue since no data is sent or kept outside the user's device, guaranteeing the privacy of sensitive or personal data. For example, FreddieMeter[3]

[3] https://experiments.withgoogle.com/freddiemeter

is an artificial intelligence-powered singing challenge that determines how well a singer's voice (timbre, pitch, and melody) resembles Freddie Mercury's while protecting user privacy.

Constraints and considerations The deployment of deep learning models to mobile devices and browsers has been discovered to have compatibility and dependability problems [45], with variations in prediction accuracy and performance between models trained on PC platforms and those deployed to mobile devices and browsers. To bridge the gap, software design tools for development and deployment on the target platform are suggested.

The security of the deep learning model is one major drawback of using the browser to generate predictions. When the model is in the user's possession, it may be saved and used for unintended purposes. A secure way of distribution would be to keep the model on the cloud or servers and offer services through APIs. Many production use cases, for example, require that the model be securely sent to the client so that it cannot be copied and used on other websites. This is especially true if the model has financial value.

While traditional cybersecurity methods can safeguard deep learning models on back-end servers, deep learning models on mobile devices and browsers pose new security challenges. An adversarial attack on neural networks may cause misclassifications, and the deep learning model's hidden states and outputs may be exploited to reconstruct user inputs, possibly compromising user privacy [59]. Thus, additional concerns regarding protecting models against attacks are needed to securely and discreetly deploy deep learning models.

2.2 Why is JavaScript the language of choice?

Python is the preferred language for most machine learning projects. JavaScript, however, may be more practical for browser-based machine learning web applications because of its ease of using front-end components.

Although alternative front-end programming languages such as Typescript, CoffeeScript, and JQuery exist, these languages are compiled into JavaScript to be executed on the browser. The JavaScript engine exists in all browsers and is the one responsible for creating a responsive, interactive environment.

The benefits of developing deep learning applications with JavaScript include *universality*, *user interaction*, *no installation*, and *direct input access*.

Universality JavaScript has been the most widely used open-source programming language for nine years[4] and is often considered a universal language for browsers [67]. It has a thriving ecosystem with broad client and server applications and has grown rapidly. Unlike backend web development, which uses Python, Java, C#, or PHP, frontend web development uses JavaScript. JavaScript allows users to execute code on almost any device, virtually unchanged while providing an intrinsically linked environment with direct access to different resources. Unlike native mobile apps, web apps may be deployed, regardless of the underlying hardware or operating system.

User interaction JavaScript enhances the user experience by adding responsive, interactive components to web pages. It can update webpage content, animate graphics, and manage multimedia. This kind of user engagement is critical in deep learning applications. For example, in a semantic image search web app,[5] an INCEPTIONV3 convolutional neural network is used to enhance user interactions. When a picture is chosen, the neural network examines the content of all the photos in the dataset and displays the top 15 most similar images.

No installation The simplicity with which JavaScript code and static resources may be shared with the help of a Content Delivery Network (CDN) and performed without installation is one of the main advantages of the JavaScript ecosystem [17]. This feature eliminates potentially tedious and error-prone installation processes and potentially dangerous access control when installing new software. This is in contrast to the Python environment, where installation becomes difficult due to dependencies. Developers may also make use of a large variety of JavaScript APIs, such as the WebGL API. For example, in [43], DeepFrag,[6] a 3D CNN model, has been deployed as a web app to allow broader usage by researchers to conduct optimization experiments in their browsers. This is done without submitting potentially proprietary structures to the server or installing any additional software. Another example is the Cell Profiler Analyst Web [9], which highlights that no installation or software update is necessary to use the tool.

[4] https://insights.stackoverflow.com/survey/2021

[5] https://convnetplayground.fastforwardlabs.com/

[6] http://git.durrantlab.com/jdurrant/deepfrag-app

Direct input access Web browsers offer an extensive set of tools for managing and displaying text, audio, and visual data. These data, produced and made accessible to the client, may be used directly with permission in the browser by deep learning models with the user's consent. Aside from these data sources, modern web browsers may access sensor data, providing exciting possibilities for many deep learning applications. For example, in [77], face recognition on the front end allowed developers to run models on the client-side, reducing latency and server load.

Constraints and considerations Deploying deep learning models on browsers presents developers with unique programming difficulties, such as converting the models to the formats required by the target platforms. The key to deploying deep learning models is the interplay between deep learning knowledge and web development. As a result, deploying deep learning models requires developers to understand both disciplines, making this a complex process.

Because machine learning development environments are incompatible with deployment environments, transferring them requires much time and effort to accommodate the new environment. The reason is that deep learning frameworks are often developed on powerful machines with GPUs, which speeds up training but may hinder inference during deployment on low-end devices.

Web applications written in JavaScript are often challenging to create for developers that prefer Python, which may introduce additional challenges to overcome. To make matters worse, only a few publicly accessible deep learning packages and built-in functions for JavaScript are available to choose from compared to Python. This lack of library support may add to the complexity and lengthen development time.

2.3 Why is TensorFlow.js ideal for front-end deployment?

TensorFlow.js is a JavaScript library for creating machine learning models compatible with TensorFlow Python APIs [111], allowing it to benefit from its strengths. However, unlike TensorFlow Python APIs, TensorFlow.js may be easily shared and performed on any platform without installation, making it more powerful. TensorFlow.js also provides high-level modeling APIs that support all Keras layers [106], making it easy to deploy pre-trained Keras or TensorFlow models into the browser and run them with TensorFlow.js.

2.3.1 Hardware acceleration

Apart from offering the flexibility required for web-based machine learning applications, TensorFlow.js also performs well. Since the TensorFlow.js library is linked with the acceleration mechanism offered by current web browsers, the TensorFlow.js platform enables high-performance machine learning and numerical computing in JavaScript [17].

TensorFlow.js's initial hardware acceleration method is WebGL [111]. Modern web browsers have WebGL APIs, which were initially intended to accelerate the rendering of 2D and 3D visuals in web pages. However, it has since been repurposed for parallel numerical computing in neural networks in TensorFlow.js [111]. This implies that the integrated graphics card may speed deep learning operations, eliminating the requirement for dedicated graphics cards, such as those from NVIDIA, needed by native deep learning frameworks. While WebGL-based acceleration is not on a par with native GPU acceleration, it nevertheless results in orders of magnitude speedups compared to a CPU-only backend. Tapping into these performance improvements enables real-time inference for complex tasks such as PoseNet extraction of body posture.

TensorFlow.js launched the WebAssembly backend (WASM) in 2020.[7] WASM is a cross-browser, portable assembly, and binary format optimized for the web that enables near-native code execution. Although WebGL is quicker than WASM for most models, WASM may outperform WebGL for small models because of the fixed overhead costs associated with WebGL shader execution. Thus, when models are tiny or when low-end devices lack WebGL support or have less capable GPUs, WASM is an excellent option used in place of JavaScript's vanilla CPU and WebGL-accelerated backend.

TensorFlow.js assigns a priority to each backend and automatically picks the one best supported in an environment. Presently, WebGL takes precedence over WASM and subsequently the vanilla JS backend.[8] An explicit call is needed to use a particular backend. Apart from hardware acceleration through WebGL and WebAssembly, WebGPU is still in its experimental phase as of June 2022.[9]

2.3.2 Robust ecosystem

The TensorFlow.js ecosystem supports all major processes [17], such as training and inference, serialization, deserialization, and conversion of models. It also supports a wide variety of environments [17, 106], including browsers, browser extensions, web servers (Node.js), cloud servers, mobile apps (ReactNative), desktop apps (Electron.js), platforms for app plugins, and single-board computers. It also

[7] https://blog.tensorflow.org/2020/03/introducing-webassembly-backend-for-tensorflow-js.html

[8] https://www.tensorflow.org/js/guide/platform_environment

[9] https://github.com/tensorflow/tfjs/tree/master/tfjs-backend-webgpu

has built-in functionality for data input and a visualization API [106]. For example in [95], a browser extension for online news credibility assessment tools[10] was developed using a collection of interactive visualizations that explain the reasoning behind the automated credibility evaluation to the user. As another example, in [11], a neural network was developed and exported to the Arduino Nano Module for geometric object identification by their acoustic signatures, which attempts to replicate the way dolphins and bats perceive by using sonic waves.

2.3.3 Alternative library for front-end deep learning

Several JavaScript-based deep learning frameworks enable deep learning in browsers. In [78], deep learning in browsers was studied to evaluate how well these frameworks perform, but the work does not consider recent TensorFlow.js speed gains. They assessed seven JavaScript-based deep learning frameworks to see how far browsers have supported deep learning tasks and compared the performance of these frameworks on various deep learning tasks. They found ConvNetJS[11] excels in both training and inference. TensorFlow.js is recommended in place of ConvNetJS, which is no longer being developed, because of its feature set and performance. Libraries such as ConvNetJS, Keras.js,[12] and Mind[13] are no longer supported, while WebDNN[14] only supports inference tasks. For training tasks, Brain.js[15] supports DNN, RNN, LSTM, and GRU, while Synaptic[16] supports first-order or even second-order RNN.

In [94], a series of deep learning frameworks tailored to the browser environment, such as TensorFlow.js, Brain.js, Keras.js, and ConvNet, are reviewed and compared for object detection. For fast inference with acceleration support, frameworks like TensorFlow.js and WebDNN were recommended. Integration of low-level programming (such as mathematical operations and data manipulation) with higher-level programming (such as deep learning model development, training, and execution in the browser) makes TensorFlow.js a more resilient framework.

According to npmtrends.com, which monitors package download counts over time, TensorFlow.js is well ahead of other ML frameworks in terms of popularity. The other frameworks (Brain.js, ConvNet.js, synaptic, and WebDNN), including the Onnx.js and Neataptic.js libraries, have not yetX reached 10% of TensorFlow.js's weekly download rate of 75 thousand.[17]

2.3.4 Available high-level API extensions

Several deep learning high-level libraries are built based on TensorFlow.js, which allows them to take advantage of its extensive library and hardware-accelerated inference. These libraries are highlighted in Table 2. The table shows the functionality and use cases of these libraries.

2.3.5 Constraints and considerations

Although TensorFlow.js is a practical and accessible deep learning framework, it has certain drawbacks. TensorFlow.js is not a framework aimed at resource-constrained IoT and embedded devices [26]. TensorFlowLite, another library from the TensorFlow framework, is more appropriate.

In TensorFlow.js, memory management is essential and tensors that are no longer required must be manually cleaned up by using the `tidy()` and `dispose()` functions [101, 111]. If they are not explicitly removed, the tensors will continue to use memory, resulting in memory leaks. The JavaScript engine in browser execution environment is also restricted and single-threaded. Thus, computationally intensive tasks may cause the UI to stall [111]. The asynchronous function should be used for computationally demanding operations [101]. These environmental issues only apply to the TensorFlow.js environment and must be considered.

3 Front-end deep learning development approach

Researchers often face new challenges and constraints when deploying deep learning models in the browser because these models may not have been explicitly developed for client-side execution. Furthermore, the high-performance back-end server with hardware acceleration such as the GPU differs from the front-end environment in the browser with limited resources. Because of these environmental differences, the emphasis has shifted to adapting models for effective deployment and improved user experience.

When a model is deployed on the front-end, the user experience is a crucial aspect to consider. As a result, much thought goes into making models that are compact and can be executed quickly [17]. In fact, model loading performance exceeds inference task performance because

[10] https://github.com/piotrmp/credibilator

[11] https://github.com/karpathy/convnetjs

[12] https://github.com/transcranial/keras-js

[13] https://github.com/stevenmiller888/mind

[14] https://mil-tokyo.github.io/webdnn/

[15] https://brain.js.org/

[16] https://github.com/cazala/synaptic

[17] https://www.npmtrends.com/@tensorflow/tfjs-vs-brain.js-vs-convnetjs-vs-onnxjs-vs-synaptic-vs-webdnn-vs-neataptic 19 June 2022

Table 2 High-level libraries that are built based on TensorFlow.js

Library	Functionality and use cases
`ml5.js`[a]	This library offers artists, creative programmers, and students with high-level access to machine learning techniques and models. Examples of pre-trained models and use cases include object detection [94, 100, 132], detecting human postures [15, 80], generating text, drawing pictures, creating music, pitch recognition, and common English language word associations
`magenta.js`[b]	Using the pre-trained Magenta models in the browser, the API can generate music and art. It contains VAE and RNN models for musical note-based models, sketch drawing models (including SketchRNN), and image style transfer models (including Arbitrary Style Transfer). Examples of use cases include [19, 33, 35, 53, 102, 106]
`face-api.js`[c]	This is a high-level API for face detection, face landmark detection, face recognition, facial expression recognition, age estimation, and gender recognition. Examples of applications include [8, 30, 38, 46, 49, 60, 71, 75, 77]
`handtrack.js`[d]	This real-time hand detection library frames hand tracking as an object detection problem and predicts bounding boxes for the position of hands in an image using a trained convolutional neural network. Example works include [58] and [120]
`machinelearn.js`[e]	This is a library for machine learning algorithms, similar to `scikit-learn` from Python. An example can be found in [106]
`Danfo.js`[f]	Inspired by Pandas, it provides a high-performance, intuitive, and easy-to-use API for manipulating and processing structured data such as arrays, JSON Objects and Tensors

[a] https://ml5js.org

[b] https://magenta.tensorflow.org/

[c] https://github.com/justadudewhohacks/face-api.js/

[d] https://github.com/victordibia/handtrack.js/

[e] https://github.com/machinelearnjs/machinelearnjs/

[f] https://danfo.jsdata.org

the process of loading and warming up the deep learning model takes longer than performing the inference job [78]. For example, if the object detection model size is larger than 10MB, it will take a long time to load, slowing down the website considerably.

Creating models from the ground up, particularly for large and complicated models, is not advisable for frontend deep learning. Because of the limitations of the browser environment, browser models must be small, have fast inference, ideally in real-time, and be as easy to train as possible [17]. However, decreasing the model's size may cause reduced accuracy, but most models will work adequately well in a browser [45]. Generally, when accuracy is weighed against user experience, a minor loss of precision is acceptable, since the user experience is emphasized.

Researchers often choose one of many options when confronted with implementing a deep learning challenge in the browser. The most straightforward and most commonly used approach is to use pre-trained TensorFlow.js models that are ready to deploy. A model that has previously solved a similar issue may also be reused by retraining the model using transfer learning to adapt it for its particular application. However, if an existing model is built in Python, conversion to a TensorFlow.js model is required. Before deployment, it is also critical to test and optimize the model. These points are addressed in more detail in the following sections.

3.1 Reusing pre-trained TensorFlow.js models

The creation of machine learning models, which are at the heart of AI software development, is not a simple task. Significant technical skills and resources are required to build, train, and deploy modern deep learning models. Special abilities in reading and comprehending professional AI literature are needed to apply deep learning algorithms. Training machine learning models also takes considerable resources. For instance, models that perform complicated tasks like image classification, object recognition, or text embedding need extensive calculations. The tasks also take a long time to train on large-scale datasets, using a significant amount of computing resources.

The complexity of creating machine learning models drives the effective reuse of machine learning models. To help with model discovery and reuse, public machine learning package repositories that collect pre-trained models may be employed. Once suitable models are found, these models may also serve as a testing ground for researchers to see whether a concept is viable before delving further and developing a model of their own.

Table 3 TensorFlow.js pre-trained models

	TensorFlow.js pre-trained models[a] and use cases
Object-utility models	
`mobilenet`	Trained using the ImageNet database, this image classification model has 1.2 million training images and 1,000 object classes. It is applied in [1, 42, 73, 89, 106, 129]
`coco-ssd`	This object detection model is trained on the COCO dataset and can classify items into 80 distinct categories
`deeplab`	This model can perform semantic segmentation, which is the process of comprehending a picture at the pixel level and then labeling each pixel by generating a two-dimensional tensor with class labels
`knn-classifier`	This is a tool for building a K-Nearest Neighbors classifier, commonly used with activations from another model, such as with transfer learning. Application of this model can be found in [73]
Face and pose models	
`blazeface`	This is a face detection model based on Single Shot Detector architecture that can detect one or more faces inside a photograph taken with a smartphone camera
`HandPose`	This lightweight machine learning pipeline is composed of two models: a palm detector and one that tracks the fingers on the hand-skeleton. It predicts 21 3D hand keypoints for each identified hand. The applications of this model can be found in [42, 116]
`Pose-detection`	This is a unified pose detection toolkit that makes use of one of three models (`MoveNet`, `BlazePose`, and `PoseNet)` to identify atypical postures and rapid body movements in real-time. MoveNet is a lightning-fast and highly accurate model that detects the body's 17 key points; BlazePose can detect 33 keypoints, and PoseNet can detect multiple poses, each of which includes 17 key points. Works on `Pose-detection` can be found in [20, 32, 81, 85, 92, 96, 96, 98, 106, 116, 124]
`BodyPix`	This is a body segmentation model that segments 24 body components from a background image or video in real-time, and it also works for multiple people. Its design is based on either MobileNetV1, a smaller but less precise model, or ResNet50, a bigger but more precise model. Use cases on `BodyPix` can be found in [91], and [41]
Text and language models	
`speech-commands`	This is a speech commands recognition model used to categorize audio clips from the dataset of voice commands. It can recognize spoken instructions composed of basic isolated English words from a limited vocabulary
`universal-sentence-encode`	This model can compress text into a 512-dimensional embedding that can be used for natural language processing tasks like textual similarity and sentiment classification. Applications of this model can be found in [42, 101]
`toxicity`	This is a model trained using the civil comments dataset, which includes over 2 million comments, that can classify text as 'Very poisonous' to 'Very healthy'. Use case for this model can be found in [42, 73]
`qna`	This is a BERT[b] natural language question-answering model using the SQuAD 2.0 dataset[c] that responds to queries about the content of a particular text excerpt

[a] https://github.com/tensorflow/tfjs-models/tree/master/

[b] BERT, or Bidirectional Encoder Representations from Transformers, is a technique for pre-training language representations that achieves state-of-the-art performance on a broad range of Natural Language Processing tasks

[c] The Stanford Question Answering Dataset, or SQuAD, is a reading comprehension dataset comprised of Wikipedia articles and a collection of question-answer pairings for each article

3.1.1 Using models from the TensorFlow.js library

TensorFlow.js includes a collection of Google's pre-trained models. These pre-trained models, as shown in Table 3, are ready to execute tasks like object identification, picture segmentation, voice recognition, and text toxicity categorization. The models may be used directly or customized using transfer learning. TensorFlow.js pre-trained models may be classified according to whether they are object-utility models, face and pose models, or text and language models.

3.1.2 Using models from online model repositories

TensorFlow Hub[18] is a massively scalable, open repository and library for reusable machine learning algorithms. The TensorFlow Hub included trained machine learning models that were ready for fine-tuning and could be deployed anywhere. The TensorFlow Hub provides a central location

[18] https://tfhub.dev/

for searching and discovering hundreds of trained, ready-to-deploy models. With a few lines of code, the most recent trained models, such as BERT and Faster R-CNN, can be downloaded and reused. There are thousands of models available, with an increasing number in each of the four input domains: text, image, video, and audio. Apart from filtering by domain, models may be filtered by formats: TensorFlow.js, TFLite, coral, TF1, and TF2. Models may also be filtered by architectural type, including BERT, EfficientDet, Inception, MobileNet, ResNet, Transformer, and VGG-style.

Another repository that is linked to their framework includes the PyTorch Hub,[19] which presently includes packages that may be accessed through the PyTorch framework's APIs. Other such repositories include Microsoft cognitive toolkit,[20] Caffe/Caffe2 model zoo,[21] and MXNet model zoo.[22]

Two repositories that are not connected to any specific framework are Model Zoo[23] and ModelHub.[24] Model Zoo curates and hosts a repository of open-source deep learning code and pre-trained models for various platforms and applications. The repositories also offer filtering capabilities to assist users in locating the models they need. TensorFlow, Keras, PyTorch, and Caffe are all included in this framework package. ModelHub is another self-contained deep learning model repository. It is a crowdsourced platform for scientific research that highlights current developments in deep learning applications and aims to encourage reproducible science.

The following repositories focus on certain fields:

- SRZoo[25] in [24] is a centralized repository for super-resolution tasks that collects super-resolution models in TensorFlow.
- ARBML[26] in [6] showcased a series of TensorFlow.js NLP models trained for Arabic, which support the NLP pipeline development.
- EZ-MMLA toolkit[27] in [51] was created as a website that makes it simple to access machine learning algorithms in TensorFlow.js for collecting a series of multimodal data streams.

Constraints and considerations Using an open-source pre-trained model may be very simple and effortless. However, some models do not disclose how they were created, the dataset on which they were trained, or even the method employed. These "black box" models can be a problem, as they can lead to legal liability [42] when an explanation of why the model made a certain prediction is required.

3.2 Customizing existing models using transfer learning

Although TensorFlow.js is not suggested for intensive training, it is suitable for small-scale interactive learning and experimenting. By integrating available pre-trained models into an appropriate use case, the development process may be significantly accelerated.

Transfer learning is a method that enables us to apply previously learned models to our unique use cases. It is repurposing a trained model for a second similar job. Transfer learning enables the combination of pre-trained models with customized training data. This implies that by simply adding custom samples, the functionality of a model may be leveraged without having to recreate everything from the start.

Most projects use transfer learning to achieve the following recurring benefits: obtaining a solution with little data, getting a solution quicker, and reusing a time-tested model structure [101]. Transfer learning has the major advantage of using less training data to develop an effective model for new classes. Rather than performing time-consuming relearning, previously acquired features, weights, or biases may be transferred to another situation. Modern, state-of-the-art models often include millions of parameters and train slowly. Transfer learning simplifies this training process by reusing a model learned on one task for a second related task. For instance, if an image classification model has been trained on thousands of pictures, rather than starting from scratch, fresh, unique image samples may be merged with the pre-trained model to produce a new image classifier using transfer learning. This feature enables the user to quickly and easily create a more personalized classifier.

Transfer learning has been applied in a variety of use cases for front-end deep learning. For example, in [118], a convolutional neural network (CNN) computer vision model was trained on approximately 10,000 pictures (5,500 snails and 5,100 cercariae). Because the image dataset was small, transfer learning was used by using seven pre-trained CNN models, where InceptionResNetV2 was found to be the best pre-trained model. After developing and training the CNN model, the best performing Keras model was converted to TensorFlow.js to be deployed. Other examples of work that applies transfer learning include [89] which did

[19] https://pytorch.org/hub/

[20] https://www.microsoft.com/en-us/cognitive-toolkit/features/model-gallery/

[21] https://github.com/caffe2/models/

[22] https://mxnet.apache.org

[23] https://modelzoo.co/

[24] http://modelhub.ai/

[25] https://github.com/idearibosome/srzoo

[26] https://github.com/ARBML/ARBML

[27] https://mmla.gse.harvard.edu/

transfer learning on InceptionV3, Resnet50, MobileNet, and [32] on PoseNet.

Transfer learning is accomplished by retraining selected parts of previously trained models [101]. This may be done by substituting new layers for the last layers of the pre-trained model and training the new, much smaller model on top of the original truncated model using freshly tailored data. For example, in [1], transfer learning is accomplished using TensorFlow's MobileNet. The MobileNet architecture's last completely linked layer has been deleted. To categorize the dataset, the higher-level characteristics are input into machine learning classifiers such as the Logistic Regression model, the Support Vector Machine model, and the Random Forest model.

Because of the variety of model implementation types, there are also different ways to access them for transfer learning purposes. For instance, the downloaded bundle from TensorFlow Hub does not include the whole model but rather truncates it into a feature vector output that may be connected to exploit the model's learned features. Fortunately, TensorFlow's environment is sufficiently flexible to support transfer learning. Understanding the model and adequate planning on reusing the underlying model is required to implement this correctly.

Constraints and considerations Two primary considerations must be addressed before initiating transfer learning. First, it is essential to validate the data's quality. If the data used in training is of poor quality, the result of the training will be worthless [73]. For this to work, the model properties gained in the first task must also be transferrable. In short, the features should be suitable for both the first and second tasks.

When the machine-learning challenge is unique, transfer learning is not suitable, and developing a model from scratch is best. However, in other instances, the problem is generic for which pre-trained models exist that either precisely fit the need or can fulfill the requirements with minimal modification. Sharing and repurposing deep learning models and resources is expected to continue to increase in popularity [17]. As modern deep-learning models are becoming more solid and broader, reusing pre-trained models for direct inference or transfer learning is becoming more practical and cost-efficient.

3.3 Converting Python models into TensorFlow.js models

While there are many open-source pre-trained models accessible online, most of these models are trained and available in TensorFlow and Keras Python formats, compared to TensorFlow.js open-source pre-trained models. These pre-trained Python models can actually be converted to TensorFlow.js for front-end deployment. For example, in [127], the model is trained in Python using the TensorFlow library and then transformed into a layer model using TensorFlow.js.

The TensorFlow.js framework provides a converter tool[28] that enables direct conversion of models trained using Python frameworks, such as TensorFlow and Keras, allowing for direct inference and transfer learning in web pages [73]. TensorFlow.js supports the following model formats: TensorFlow SavedModel, Keras model, and TensorFlow Hub module [106]. When the TensorFlow.js tools converted the model, they produced a JSON file containing the model's information and a binary file holding the weights and biases.

While conversion is workable for TensorFlow SavedModel and Keras models, the conversion will fail if the model contains operations that are not supported by TensorFlow.js and modification of the original model needs to be done. The full list of TensorFlow.js operations is available here.[29] For example, CamaLeon [30] was trained in Keras before being converted to TensorFlow.js for browser-based inference. After many optimization rounds, the model was reported to have 485,817 parameters and a weight size of just 2MB when converted to TensorFlow.js. Other examples of conversions from Keras for usage in web applications are reported in [6, 88, 99].

In addition, models created using PyTorch may be used, but the models must go through an extra conversion before they can be used for inference with TensorFlow.js. It must first be converted to Keras or Onnx format before being converted to TensorFlow.js. Again, changes may be required if any operations are not supported because of differences in library support across models. For example, in [108], the conversion process was accomplished via the conversion of PyTorch to the Onnx standard and then to TensorFlow and TensorFlow.js.

Constraints and considerations Converting models poses a challenge when the model is not compatible with the TensorFlow.js library and changes to the API calls need to be made. In [45], compatibility and reliability issues were highlighted, as well as accuracy loss in certain instances when moving and quantizing a deep learning model from one platform to another. They suggested implementing platform-agnostic deep learning solutions, particularly for mobile and browser platforms, to tackle the problem.

3.4 Optimizing the model prior to deployment

Before deploying TensorFlow.js models to production, testing and model optimization are strongly recommended.

[28] https://www.tensorflow.org/js/guide/conversion

[29] https://github.com/tensorflow/tfjs/blob/master/tfjs-converter/docs/supported_ops.md

Optimizing the download and inference speeds is critical for the client-side deployment of TensorFlow.js models to succeed [17].

The TensorFlow.js converter supports both optimization methods:[30] graph-model conversion to improve inference speed, and post-training weight quantization to reduce the model size.

- Inference speed is optimized via the use of graph optimization, which simplifies computation graphs and decreases the amount of computation needed for model inference.
- Model size is optimized through weight quantization, which reduces the model weight to a lower size. Weight quantification may not result in a significant decrease in forecast accuracy. In most instances, this phase has a minimal effect on total model correctness [45]. However, if accuracy falls, repeatedly omitting or adding tensors from quantization may help identify the tensors that cause the model's accuracy to decrease after their values have been quantized. As a result, it is critical to ensure that the model keeps an appropriate level of accuracy following quantization.

Optimization has been used successfully in several use cases. As an example, the original `ssd_mobilenet_v2 _cocomodel` COCO-SSD object identification model is 187.8 MB in size. In comparison with the original model, the TensorFlow.js version of the model is very lightweight and optimized for browser execution. The TensorFlow.js `lite_mobilenet_v2` model is less than 1MB and has the quickest inference performance. As another example, in [53], by using post-training weight quantization, the downloaded weights are compressed, resulting in a 400KB payload size with no discernible loss of quality. By using dilated depth-wise separable convolutions and fusing procedures, the researchers decreased the model's run-time for one harmonization from 40s to 2s.

Research on model optimization Because models must be both compact and powerful, how to construct a neural network with the lowest possible size while still completing tasks with an accuracy equal to that of a larger neural network, is a prominent subject of research in deep learning.

A method for expanding and describing neural net information in a model based on hierarchical choices is suggested in [40]. In addition, to identify performance bottlenecks and aid system design, a thorough characterization of the results for a selection of paradigmatic deep learning workloads was provided in [44].

To produce a smaller model and low-latency inference, context-aware pruning was introduced in [55] that takes into consideration latency, network state, and computational capability of the mobile device. A binary convolutional neural network was later introduced in [56]. The updated approach reduces 'model size by 16x to 29x' and reduces 'end-to-end latency by 3x to 60x' compared to existing methods.

A progressive transfer framework for deep learning models is introduced in [76] by using the principle of transferring large image files over the web. The framework enables a deep learning model to be divided and progressively transferred in stages. Gradually, each part added builds a better model on the user device.

Constraints and considerations While model optimization is beneficial, the best approach to guarantee that the model works effectively is to design it from the start with resource restrictions. This means avoiding excessively complicated designs and, where feasible, reducing the number of parameters or weights.

A possible fallback alternative if the front end does not perform well would be to use back-end services. For example, in [53], support for Tensor Processing Units (TPU) on Google Cloud was also introduced, besides front-end inferences for preparation of large-scale deployment. Here, a speed test is run to see whether the user's device can run the model in the browser. Otherwise, the requests were routed to distant servers.

4 Front-end deep learning web apps

Web browsers have drawn the interest of AI researchers because they offer a cross-platform computing target for deep learning on the client's side. Privacy, accessibility, and low latency interactions are just a few advantages of using web browsers for deployment. The browser's access to components, including the web camera, microphone, and accelerometer, allows for simple integration of deep learning models and sensor data. Because of this connectivity, user data may be kept on the device while maintaining user privacy, allowing personalized deep learning apps in fields like medicine and education.

Deep learning improves many existing solutions while also bringing new practical applications. Sometimes, an entirely new area of application is introduced. A few applications of TensorFlow.js are available in TensorFlow.js's gallery of applications.[31]

This section focuses on research that leverages front-end deep learning web apps. It explores the question of what front-end deep learning can do by providing a glimpse of the work that has been deployed on the front end. Instead of delving deeply into any one field, the section concentrates

[30] https://github.com/tensorflow/tfjs/tree/master/tfjs-converter

[31] https://github.com/tensorflow/tfjs/blob/master/GALLERY.md

on how working on the front end may help solve user problems.

The collected works have 18 different fields, and they are grouped into 7 categories based on the purpose of the web apps. To show how researchers construct front-end deep learning web applications, the section also provides links to resources, such as the website where it is deployed or the GitHub repository where the source code is located.

To help navigate the broad list of diverse topics, Fig. 1 has been designed to outline and highlight the major groupings of applications that will be covered. For example, if gesture tracking is of interest, following the section number (Section 4.2) will lead to examples of what has been done and how web apps are used to achieve gesture tracking.

4.1 Deep-learning playground apps

Education playground Teachable Machine,[32] in [18], is a web app that enables non-programmers to train their machine learning classification models using videos, pictures, or sounds from their devices. To build a model, it uses transfer learning to uncover the trends and patterns within images or sound samples. The trained model can also be downloaded locally, eliminating the need to save and save huge files, datasets, or models on the cloud. Teachable Machine offers expandable panels for hyperparameter tweaking and model assessment visualizations for customers who desire greater control over model training.

Two other attempts have been made to create online AI learning applications comparable to Google's Teachable Machine. In [107], the aim is to develop a web-based learning application[33] to assist novices in comprehending deep learning processes and resolving real-world issues. In [90], the aim is to create a web app for an open artificial intelligence platform that helps preprocess input data, trains artificial neural networks, and sets up future actions based on inference findings.

For children's education, an interactive online explanation using picture recognition is suggested in [100]. When prompted by the website's instructional video, the user will take a picture of the item, and the website will identify it using a deep learning algorithm created using ml5.js.

Visualization playground GAN Lab,[34] in [66], is an interactive visual learning and experimentation app for Generative Adversarial Networks (GANs). Users may train GANs interactively and visually examine the model training process to grasp how GANs function during training and inference. In a follow-up study, an in-person observational study was conducted to investigate how GAN Lab is used and what users gain from it. Design considerations and difficulties associated with interactive teaching systems for deep learning have also been highlighted in [65].

The same research group presented CNN Explainer,[35] in [126] and [125], an interactive visualization application that helps explore convolutional neural networks. This app imports the pre-trained Tiny VGG model and uses TensorFlow.js to calculate results in real-time. They also presented a review of visualization and visual analytics in deep learning research, using a human-centered, interrogative approach [52].

Anomagram[36] is another interactive visualization app for studying deep learning models, and it looks at how an autoencoder model may be used for anomaly detection. The ECG5000 dataset was used for interactive training and testing, in which the autoencoder model predicts whether an ECG signal sample is normal or abnormal.

Two other studies that introduced visualization for learning deep learning include TensorSpace,[37] a framework for visually representing pre-trained neural network models in three dimensions, and LEGION,[38] in [28], a graphical analytic app that enables users to compare and choose regression models that have been created either via hyperparameter tweaking or feature engineering.

Development playground Milo[39] in [97] is a web-based visual programming environment for data science. It provides an abstraction for language-specific implementations of machine learning principles using graphical blocks (Blocky[40]) and the creation of interactive visualizations. Similarly, DeepScratch,[41] in [5], is a new Scratch programming language extension that adds strong language features to aid in the development and exploration of deep learning models. Likewise, Marcelle[42] is another toolkit that enables interactive machine learning development by composing or customizing component-based machine learning workflows based on the Tensorflow.js library [39].

To employ gamification to facilitate deep learning, a series of tasks[43] was presented in [105]. These tasks assess the human-interpretability of generative model representations, in which users change model parameters

[32] https://teachablemachine.withgoogle.com/

[33] http://ai.uol.de

[34] https://github.com/poloclub/ganlab/

[35] https://github.com/poloclub/cnn-explainer

[36] https://anomagram.fastforwardlabs.com/

[37] https://tensorspace.org/

[38] https://gtvalab.github.io/projects/legion.html

[39] https://miloide.github.io/

[40] https://developers.google.com/blockly/

[41] https://github.com/Noufst/DeepScratch

[42] https://github.com/marcellejs/marcelle

[43] https://hreps.s3.amazonaws.com/quiz/manifest.html

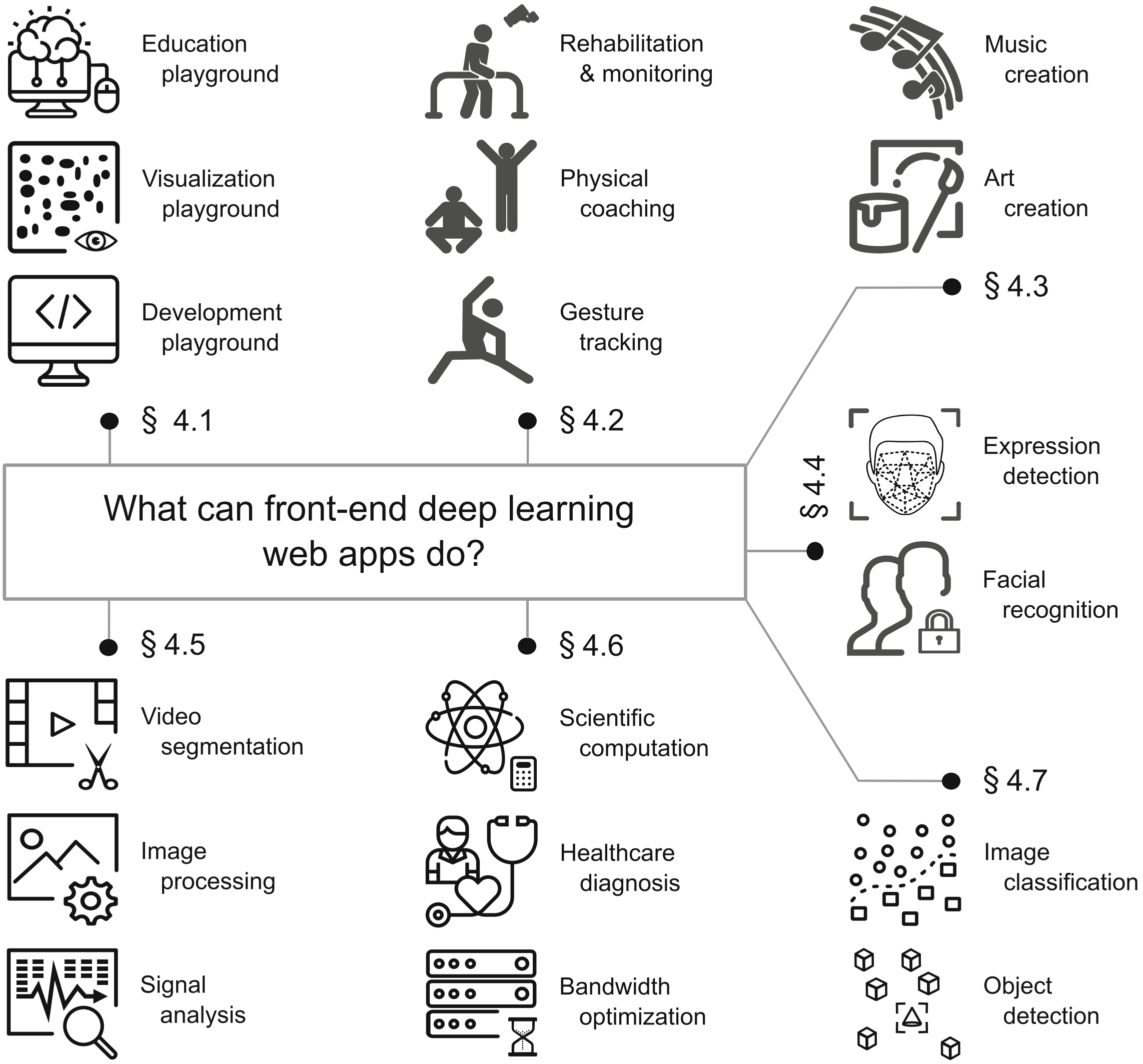

Fig. 1 Examples of front-end deep learning web apps

interactively to recreate target instances. Similarly, in [64], students were given 30 'half-baked' artificial intelligence projects[44] in the Snap! block language programming system to explore and improve.

4.2 Body pose detection and gesture tracking apps

Rehabilitation and monitoring To facilitate fall risk assessment and rehabilitation monitoring for the elderly, a web app was introduced in [98]. A similar home care monitoring system was reported in [20], where it can assess the quality of in-home postural alterations, such as posture conversion, body movement, and positional changes.

In a similar work, a fall detector is proposed in [7] using convolutional neural networks and recurrent neural networks. The suggested technique is appropriate for real-time detection of a single individual's fall in a controlled setting, even on low-end computers.

Most recently, in [25], PoseNet was used for in-home rehabilitation where its skeletal tracking was used to identify and monitor patients' angular motions as they conduct rehabilitation activities in front of a webcam. After patients have completed their rehabilitation activities, doctors may examine and assess the deviation rate of their angular motions across various days to find out their rate of recovery.

Physical activity coaching In [104], web-based video annotations were presented that support multimodal annotations and can be used in a variety of scenarios, including dance rehearsals. Pose estimation is used to detect a human skeleton in video frames, giving the user an app for locating potential annotations. To make it an interactive tool, voice integration was later added by using the ML5 speech and sound classifier to provide human-computer interaction [103].

[44] https://ecraft2learn.github.io/ai/

In another study, the PoseNet machine learning model from ml5.js was used to create an exercise tracking system that uses two cameras to track and assess the difficulty of an exercise while providing feedback [96]. According to them, a single RGB camera may miss certain essential information, and incorrect body movement during workouts may also be overlooked. A self-monitoring and coaching system for online squat fitness is also designed in [124].

In [32], a Digital Coaching System uses PoseNet to provide real-time feedback on exercise performance to the trainee by comparing keypoints from two video feeds, one from the trainer (using ResNet50) and one from the trainee (using MobileNet). The trainer's video feed is pre-recorded and processed with a pre-trained pose estimation model to generate keypoints that will be compared to the keypoints from the trainee's live video feed.

An assessment of pre-trained CNN models for player detection and motion analysis in pre-recorded squash games was carried out in [15]. One of the ml5.js-based algorithms demonstrated the fastest inference by using a 28-layer deep MobileNetV1 architecture, with the least depth of the CNNs evaluated but the lowest accuracy across all threshold stages in all videos.

To estimate human pose and identify anatomical points, a computer vision-based mobile tool is proposed in [82] for assessing human posture.

Gesture tracking HeadbangZ,[45] in [81], is a web-based game that shows a 3D gesture-based input using deep posture estimation and user interaction. Likewise, Scroobly[46] and PoseAnimator[47] map the user's actual motion to their animations using Facemesh and PoseNet machine learning models. When the user moves, the machine learning algorithm changes the animation shown on the screen.

TensorFlow.js pose estimation was also used in the development of Learn2Sign [92], which provides feedback for learning signed languages. In contrast, a real-time translation of American Sign Language (ASL) into text is available in [86].

4.3 Music and art creation apps

Music creation Magenta.js [102] is an open-source toolkit with a simple JavaScript API that abstracts away technical complexities, allowing application developers to build new interfaces for generative models. While Magenta.js's goal is far broader than music, the suite's first package, @magenta/music, contains several state-of-the-art music-generating models.

Trained on Magenta's MusicVAE's latent space, MidiMe,[48] in [33], is a compact Variational Autoencoder (VAE) that enables artists to summarize the musical characteristics of the input and create new customized samples based on them. Again powered by Magenta.js, Bach Doodle,[49] in [53], attempts to make music creation more accessible to amateurs and professionals alike. Users may compose their tune and have it harmonized in baroque-style counterpoint composition using an easy sheet music interface supported by a machine learning model capable of filling in arbitrarily incomplete scores. In addition, Tone Transfer,[50] in [19], is an interactive online app that allows users to convert any audio input into various musical instruments using differentiable digital signal processing (DDSP) models.

In a similar work, Rhythm VAE[51] [122] is a system for exploring musical rhythms' latent spaces. It uses minimal training data, which allows for quick customization and exploration by individual users. Likewise, JazzICat,[52] in [72], a deep neural network LSTM model, shows how it may simulate a jazz improvisation scenario involving a human soloist and an artificial accompaniment in a real-time environment. In addition, Essentia.js,[53] in [27], is a collection of publicly available pre-trained TensorFlow.js models for music-related tasks such as signal analysis, signal processing, and feature extraction.

Art creation Using the results of picture recognition in Sketcher,[54] a convolutional neural network model that recognizes drawings was implemented in [36], to offer human-like complimenting feedback.

In contrast, a web app for collaborative sketching is presented in [35]. It leverages Magenta.js' sketch-rnn to be both cooperative and adaptable to the actions of its human partner. During collaborations, this app may suggest logical extensions of incomplete drawings.

Finally, also following earlier work, this time using Sketch-rnn and Sketch-pix2seq, an image-to-sequence VAE model is constructed to guide the user via an interactive GUI environment using conditionally produced lines [127].

4.4 Facial expression detection and recognition apps

Expression detection A comparison study of two open-source emotion recognition software packages was performed under a range of lighting and distance conditions, in

[45] https://github.com/dermotte/headbangz

[46] https://experiments.withgoogle.com/scroobly

[47] https://github.com/yemount/pose-animator/

[48] https://midi-me.glitch.me/

[49] https://magenta.tensorflow.org/coconet

[50] https://magenta.tensorflow.org/ddsp

[51] https://github.com/vigliensoni/R-VAE-JS

[52] https://github.com/kosmasK/JazzICat

[53] https://mtg.github.io/essentia.js/

[54] https://zaidalyafeai.github.io/sketcher/

which they found that Face-api.js outperforms CLMTrackr with 64% vs. 28% average accuracy [8].

An intelligent environment capable of detecting students' learning-related emotions was introduced in [99], allowing different interventions to be carried out automatically in response to students' emotional states. In addition, an online report and dashboard were also suggested in [46] that offer instructors an assessment of students' involvement and emotional states while using an online tutoring system. Likewise, a head pose app that predicts students' real-time head direction and reacts to potential disengagement was also proposed in [128]. The app also includes a facial expression dashboard that detects students' emotional states and delivers this information to instructors to help instructors assess student development and identify students who need more help.

To assist with emotional healthcare monitoring during mental treatment, a video analysis app on facial expression and emotion visualization was created in [48] and [49]. Similarly, a WebRTC-based real-time video-conferencing application that can detect facial emotions by reading the participants' facial expressions was presented in [34].

Interestingly, the relationship between human mental fatigue level as measured by bio-signals (i.e., eye blink data) and plant health was studied in [68]. Face-api.js is also used to conduct neuromarketing research by identifying respondents' emotions after seeing advertisements [38].

Facial recognition A 68-point facial landmark predictor that is under 200kb in size and capable of matching the predicted 68-point landmarks with conventional face recognition landmarks has been presented in [77]. In addition, to prevent unauthorized access, a system was created that uses facial recognition and landmark detection techniques to perform identity verification and liveness detection tasks [75]. Similarly, a smart identification system is featured in [2] for video conferencing applications based on face-api.js, and a distributed hash table using blockchain technology.

4.5 Video, image and signal analysis apps

Video segmentation To transform the backdrop of the distant peer's webcam feed into one that matches the receiver's, a browser-based application in [30] uses UNet and TensorFlow.js to perform real-time machine vision. In a similar work, TensorFlow.js's BodyPix model is used to segment the backdrop and the individual for enhancing remote user presence. This is done by replacing the distant user's backdrop with a real-time acquired picture of the region behind the receiving side [41].

A real-time painting algorithm is proposed in [12] that can remove unwanted human objects from the video by performing segmentation on the video frame by frame using BodyPix and TensorFlow.js. Instead of altering the backdrop, Invisibility Cloak[55] is a real-time person removal system that runs in the browser.

To predict the areas and landmarks of the faces for real-time face augmentation, a pre-trained FaceMesh model was used in [115] to overlay various augmented reality filters and effects over the identified face regions.

Interestingly, rather than adding or removing video regions of the face and body, a face-touching web application has been developed in [91]. By using BodyPix 2.0, the application accepts real-time input from the built-in camera and identifies face-touching using the intersection of the hands and facial areas, delivering a warning notice to the user.

Image processing Front-end deep learning web apps have also been used to provide a centralized tool that enables individuals working in disparate places and industries to interact.

An online Integrated Fingerprint Image system was created with web-based tools that can view, edit, apply pattern recognition, analyze, and interact with pictures in real-time [54]. The system can extract characteristics and discriminate between normal and abnormal regions of the image, while allowing for comments. In addition, preprocessing of the input picture has also been shown in [62] to significantly affect the quantity of input data that must be sent to the cloud service, which subsequently improves server-side processing.

Cell Profiler Analyst Web (CPAW)[56] in [9] allows users to examine image-based data and categorize complex biological traits via an interactive user interface, allowing for better accessibility, faster setup, and a simple workflow. During the active learning phase, cells are fetched to be trained and categorized by the machine learning classifier into their appropriate class using TensorFlow.js. MedSeg[57] is another clinical web app that allows simple volume segmentation of organs, tissue and pathologies for radiological images. The segmentation of the images can be done manually or automatically using deep learning models.

Signal analysis A virtual laboratory for EEG data analysis that enables data analysis, pre-processing, and model development was built with TensorFlow.js in [3]. Using real-time sensor measurements of rib-cage movement, a separate clinical study in [108] found that an LSTM network model can predict physical effort by comparing how much air a person breathes in a minute to their perceived workout intensity.

[55] https://github.com/jasonmayes/Real-Time-Person-Removal

[56] https://mpsych.github.io/CellProfilerAnalystWeb/

[57] https://www.medseg.ai/

In [70], using captured audio data during class observation, deep learning models were built to classify classroom activities. The CNN classifier model outperformed the KNN and Random Forest models in this study.

Using vibrational signals, a real-time classification model for rolling bearing diagnostics based on MobileNet was built in [129] and tested on a range of fault types. In the study, the improved ReLU is superior to the standard ReLU activation function.

4.6 Scientific computation, diagnosis and optimization apps

Scientific computation MLitB [79] is the first prototype machine learning framework written completely in JavaScript without Tensorflow.js. It can conduct large-scale distributed computing with diverse classes of devices using Web browsers.

JSDoop[58] is a high-performance volunteer-based web computing library introduced in [84] that splits a problem into tasks and distributes the computation using various queues. TensorFlow.js is used as a proof-of-concept to train a recurrent neural network that produces or predicts the following letter of an input text. In a follow-up study [83], a federated learning system that is dynamic and adaptable has been implemented and evaluated to train common machine learning models. The system has been evaluated with up to 24 desktops working together via web browsers to train common machine learning models, while allowing users to join or leave the computation and keeping personal data stored locally.

For exploratory data analysis of high-dimensional data, a T-distributed Stochastic Neighbor Embedding (t-SNE) algorithm[59] is proposed in [93]. Their technique decreases the computational cost by orders of magnitude while maintaining or improving the accuracy of previous methods. In contrast, LatentMap [61], a GAN-based method, was developed to analyze the latent space of density maps. The method can be applied to any density map in spatiotemporal visualization. It may speed up front-end loading, complete or anticipate stream data information, and visualize missing data.

DeepFrag is a deep learning application presented in [43] for lead optimization, an important step in early-stage drug development, which involves making chemical modifications to a small-molecule ligand to improve features such as binding affinity. In another study to help predict drug responsiveness, a deep learning model[60] was used on unlabeled cell culture images in [23]. Transfer learning was applied using the MobileNetV2 architecture and converted to TensorFlow.js format before deployment in the browser.

Healthcare diagnosis For healthcare diagnosis, front-end deep learning web apps have been used for a wide variety of use cases.

ImJoy,[61] in [88], is a versatile and open-source app that enables the broad reuse of deep learning methods and plugins for interactive image analysis and genomics. For instance, the Skin-Lesion-Analyzer plugin uses a deep convolutional network in TensorFlow.js to categorize images of skin into seven distinct kinds of (possibly malignant) skin lesions. Similarly, a web application was created in [113] that allows users to submit images and analyze them for skin melanoma lesions using a convolutional neural network previously trained on Google Teachable Machine. In [31], a classification of skin cancer lesions was presented using two different implementations, a basic CNN model and a transfer learning model implemented using ResNet50 pre-trained with ImageNet. The transfer learning model was found to give higher accuracy.

To train the model for COVID-19 Chest X-ray Diagnosis, CustomVision from Microsoft Azure Cognitive Services was used and exported to TensorFlow.js in [14]. Similarly, to detect and monitor abnormal shapes of the skull in children, the Google Inception V3 model was trained using a transfer learning approach and converted to TensorFlow.js before deployment [117].

In [121], an ear infection classifier[62] was implemented and trained with published otoscopic images, from which transfer learning on MobileNetV2 was found to outperform Inception-V3, ResNet-50, and InceptionResnet-V2. In [57], the viability of Google's Teachable Machine was assessed against the diagnosis of tooth-marked tongue, a condition in which tooth traces develop on the tongue.

In [110], to assess public perceptions of physical distance, social media content posted during the COVID-19 epidemic was categorized using gated recurrent unit GRU-based recurrent neural network models. In [13], an online platform that enables users' annotations, promotes active learning, and offers model inference calculated immediately in the web browser is presented. For the case study, breast cancer tissue microarray images and COVID-19 computed tomography images were used. TensorFlow.js was used to deploy the model, which was originally built using Google Cloud AutoML.

Lastly, in [37], a remote healthcare cyber-physical systems (CPS) application that employs TensorFlow.js to

[58] https://github.com/jsdoop/

[59] https://github.com/tensorflow/tfjs-tsne

[60] https://bioanalysis-79545.web.app/main/images/a549

[61] https://imjoy.io/

[62] https://headneckml.com/tympanic.html

allow machine learning algorithms to work on collected data in a variety of applications, such as image recognition and audio classification, has been proposed.

Bandwidth optimization Video streaming and cloud gaming services can be improved by reducing delays or latency between user input and feedback.

An adaptive video streaming solution was suggested in [10], which includes bandwidth prediction and model auto-selection methods tailored especially for low-latency live streaming. In [114], Deep Reinforcement Learning-based algorithms were used for low-latency live video streaming adaptation that could choose the video rate at each request. A networking application was used to show how domain knowledge in networking can enhance the robustness of the systems for dynamic adaptive video streaming [131].

To compensate for the high latency of online video games, artificial neural networks have been introduced in [50] to predict the player's next move and implement it in the game before receiving the actual user input. This effectively reduces the time required to process the feedback loop between player and game, thus improving user performance and user experience.

4.7 Classification and detection apps

Image classification Image classification has a wide range of applications. Here, the works are arranged chronologically.

For image forensic analysis, a Deep Convolutional Neural Network with transfer learning was suggested in [1], where the model can accurately classify various classes of camera models using the Open-Image dataset.

To reduce the computing load associated with image processing, a web application that enables users to evaluate Image Visual Quality (IVQ) and conduct image enhancement by comparing it to IVQ prediction was developed using neural networks and TensorFlow.js in [119].

A low-cost augmented reality system for pre-school books was described in [74], which uses convolutional neural networks (CNN) models to recognize pages and select which 3D graphics to display. By producing unique 3D views of pages under a range of lighting conditions and camera navigations on each page, the method claimed to require less time and effort to generate datasets.

To automate the encoding of early seventeenth-century music prints, an online Optical Music Recognition (OMR[63]) system was created in [112]. The system can process pictures of written music and classify these instances using supervised learning with convolutional neural networks and TensorFlow.js.

To analyze unsteady gas flows, a neural network was built to classify the shadowgraph picture dataset and identify images with shock waves [132]. A separate CNN model was also developed to perform the regression job and describe the location of shock waves.

To classify medically essential snails and their parasitic equivalents, CNN models were employed in image identification tasks in [118], and the CNN's classification performance was comparable to that of expert parasitologists.

To support post-earthquake damage inspection and investigation, the CNN model was trained with a transfer learning approach using 1780 manually labeled images of structural damage in [87]. Of the six classic pre-trained models, MobileNet's fine-tuned model proves to be superior to the others and is further developed as a web-based application.

Object detection A dynamic object classification technique was developed and validated using low-quality webcam images in [63]. They examined several designs and discovered that although well-known baseline architectures such as Xception, DenseNet, and ResNet perform well on high-quality ImageNet datasets, they fall short on low-quality image datasets. MobileNet, however, works better with low-quality pictures. Similarly, a garbage classification system that can dynamically self-learn using real-time data has been created in [130] to assist consumers in classifying domestic waste correctly.

An object-based sound feedback system was developed in [85] that uses the SSD-MobileNetV2 object detection algorithm to assist visually impaired individuals in comprehending their environment via sound production. In [89], for Indian sign language recognition, transfer learning was used where pre-trained MobileNet model output features were fed to a KNN classifier to identify actions and predict respective words.

To distinguish chosen invasive plants from comparable non-invasive species, the TensorFlow.js' MobileNets model was retrained using pictures of invasive plants and their characteristics in [69]. This is done in a variety of light situations and stages of the plant's life cycle. Likewise, preliminary detection and sexing of cricket species was performed using a model generated by Google Teachable Machine that uses transfer learning on MobileNet [47]. Similarly, Plant AI[64] is a web app that identifies diseases in plants.

[63] https://github.com/jjstoessel/IntelliOMR_2020_prototype_release

[64] https://github.com/Rishit-dagli/Greenathon-Plant-AI

5 Conclusions

The most challenging aspect of the deep learning process is deployment. However, front-end deployment is more straightforward than back-end server, and mobile app deployment [21]. In addition, front-end deployment works best when the model has to work interactively and access resources on the front end without installation while preserving data privacy and offering real-time inference speed.

Although in-browser deep learning is still in its infancy, progress in front-end technologies has helped drive the development of deep learning web apps. This is especially true for TensorFlow.js and JavaScript in the browser, which have become a viable front-end stack for deep learning research and deployment. However, the front-end stack poses concerns on model security, IP issues, and potential attacks, all of which must be considered before deployment. Further studies on web security are needed to tackle these concerns. During front-end web development, it is also necessary to consider releasing used resources using garbage collection, calling, and releasing the CPU thread to improve performance and prevent blocking. The challenge is that front-end deep learning deployment needs both deep learning knowledge and web development (software engineering) know-how.

Development for the front-end Deep learning differs from the back end. The model must be small, have fast inference, and user experience must be considered. A typical starting point for front-end development and deployment is to reuse an existing model from one of the many repositories available. However, background research on the model is recommended to prevent the black-box model issue. In addition, transfer learning may be used to extend or customize an existing model. However, compatibility between the current training task and the new task is needed so that the previous task can be transferable to the new task. If an existing model is converted, compatibility issues must be addressed by rewriting certain functions. Finally, optimization of inference speed and model size is critical before deployment to maximize user experience.

To understand better how it could be used in a broad range of disciplines, the article also discusses reported efforts that use deep learning on the front end while using the principles summarized here. The article explores how web apps are used as playgrounds, with a focus on artificial intelligence learning and the creation of music and art. To improve human wellbeing, front-end deep learning provides interesting human-centered apps such as pose detection, gesture tracking, expression detection and facial recognition. In addition, the paper examines how deep learning web apps may assist in video, image and signal processing, scientific computation, healthcare diagnosis, bandwidth optimization, image classification and object detection in the real world.

Finally, this paper provides many links and citations to examples and codebases, which may assist individuals interested in front-end deep learning in overcoming entrance hurdles.

References

1. Al Banna MH, Ali Haider M, Al Nahian MJ et al (2019) Camera model identification using deep CNN and transfer learning approach. In: 2019 international conference on robotics, electrical and signal processing techniques (ICREST). IEEE, Dhaka, pp 626–630. https://doi.org/10.1109/ICREST.2019.8644194
2. Alizadeh M, Andersson K, Schelén O (2022) DHT- and blockchain-based smart identification for video conferencing. Blockchain: Res Appl 3(2):100,066. https://doi.org/10.1016/j.bcra.2022.100066
3. Alphonse J, Diwakar S (2020) Deploying a web-based electroencephalography data analysis virtual laboratory. Procedia Comput Sci 171:2420–2425. https://doi.org/10.1016/j.procs.2020.04.261
4. Alshangiti M, Sapkota H, Murukannaiah PK et al (2019) Why is developing machine learning applications challenging? a study on stack overflow posts. In: 2019 ACM/IEEE international symposium on empirical software engineering and measurement (ESEM). IEEE, Porto de Galinhas, pp 1–11. https://doi.org/10.1109/ESEM.2019.8870187
5. Alturayeif N, Alturaief N, Alhathloul Z (2020) DeepScratch: scratch programming language extension for deep learning education. IJACSA 11(7). https://doi.org/10.14569/IJACSA.2020.0110777
6. Alyafeai Z, Al-Shaibani M (2020) ARBML: democritizing arabic natural language processing tools. In: Proceedings of second workshop for NLP open source software (NLP-OSS). Association for Computational Linguistics, Online, pp 8–13. https://doi.org/10.18653/v1/2020.nlposs-1.2
7. Apicella A, Snidaro L (2021) Deep neural networks for real-time remote fall detection. In: Del Bimbo A, Cucchiara R, Sclaroff S et al (eds) Pattern recognition. ICPR international workshops and challenges. Springer, Cham, pp 188–201. Lecture notes in computer science. https://doi.org/10.1007/978-3-030-68790-8_16
8. Aranha RV, Casaes AB, Nunes FLS (2020) Influence of environmental conditions in the performance of open-source software for facial expression recognition. In: Proceedings of the 19th Brazilian symposium on human factors in computing systems. ACM, Diamantina, pp 1–10. https://doi.org/10.1145/3424953.3426630
9. Baidak B, Hussain Y, Kelminson E et al (2021) CellProfiler analyst web (CPAW) - exploration, analysis, and classification of biological images on the web. In: 2021 IEEE visualization conference (VIS). IEEE, New Orleans, pp 131–135. https://doi.org/10.1109/VIS49827.2021.9623317
10. Bentaleb A, Begen AC, Harous S et al (2021) Data-driven bandwidth prediction models and automated model selection for low latency. IEEE Trans Multimed 23:2588–2601. https://doi.org/10.1109/TMM.2020.3013387
11. Bertemes-Filho P, Gandolphi de Almeida MP (2020) Acquisition and recognition of ultrasonic signatures using multi-layer neural network. IJBSBE 6(3):70–73. https://doi.org/10.15406/ijbsbe.2020.06.00190
12. Bharathi Kannan B, Daniel A, Pandey DK et al (2021) Real-time person removal from video. In: Prateek M, Singh TP, Choudhury T et al (eds) Proceedings of international conference

on machine intelligence and data science applications. Springer, Singapore, pp 295–298. Algorithms for intelligent systems. https://doi.org/10.1007/978-981-33-4087-9_26

13. Bhawsar PS, Abubakar M, Schmidt M et al (2021) Browser-based data annotation, active learning, and real-time distribution of artificial intelligence models: from tumor tissue microarrays to COVID-19 radiology. J Pathol Inform 12(1):38. https://doi.org/10.4103/jpi.jpi_100_20
14. Borkowski A (2020) Using artificial intelligence for COVID-19 chest X-ray diagnosis. Fed Pract 37(9):398–404. https://doi.org/10.12788/fp.0045
15. Brumann C, Kukuk M, Reinsberger C (2021) Evaluation of open-source and pre-trained deep convolutional neural networks suitable for player detection and motion analysis in squash. Sensors 21(13):4550. https://doi.org/10.3390/s21134550
16. Cai CJ, Guo PJ (2019) Software developers learning machine learning: motivations, hurdles, and desires. In: 2019 IEEE symposium on visual languages and human-centric computing (VL/HCC). IEEE, Memphis, pp 25–34. https://doi.org/10.1109/VLHCC.2019.8818751
17. Cai S, Bileschi S, Nielsen ED et al (2020) Deep learning with JavaScript: neural networks in Tensorflow. Js Manning Publications Co., Shelter Island
18. Carney M, Webster B, Alvarado I et al (2020) Teachable machine: approachable web-based tool for exploring machine learning classification. In: Extended abstracts of the 2020 CHI conference on human factors in computing systems. ACM, Honolulu, pp 1–8. https://doi.org/10.1145/3334480.3382839
19. Carney M, Li C, Toh E et al (2021) Tone transfer: in-browser interactive neural audio synthesis. In: Joint proceedings of the ACM IUI 2021 workshops, vol 2903. CEUR Workshop Proceedings, College Station
20. Chen S, Saiki S, Nakamura M (2020a) Nonintrusive fine-grained home care monitoring: characterizing quality of in-home postural changes using bone-based human sensing. Sensors 20(20):5894. https://doi.org/10.3390/s20205894
21. Chen Z, Cao Y, Liu Y et al (2020b) A comprehensive study on challenges in deploying deep learning based software. In: Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. ACM, Virtual Event USA, pp 750–762. https://doi.org/10.1145/3368089.3409759
22. Chen Z, Yao H, Lou Y et al (2021) An empirical study on deployment faults of deep learning based mobile applications. In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE). IEEE, Madrid, pp 674–685. https://doi.org/10.1109/ICSE43902.2021.00068
23. Cho K, Choi ES, Kim JH et al (2022) Numerical learning of deep features from drug-exposed cell images to calculate IC50 without staining. Sci Rep 12(1):6610. https://doi.org/10.1038/s41598-022-10643-9
24. Choi JH, Kim JH, Lee JS (2020) Srzoo: an integrated repository for super-resolution using deep learning. In: ICASSP 2020–2020 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, Barcelona, pp 2508–2512. https://doi.org/10.1109/ICASSP40776.2020.9054533
25. Chua J, Ong LY, Leow MC (2021) Telehealth using PoseNet-based system for in-home rehabilitation. Future Internet 13(7):173. https://doi.org/10.3390/fi13070173
26. Cornetta G, Touhafi A (2021) Design and evaluation of a new machine learning framework for IoT and embedded devices. Electronics 10(5):600. https://doi.org/10.3390/electronics10050600
27. Correya A, Marcos-Fernández J, Joglar-Ongay L et al (2021) Audio and music analysis on the web using essentia.js. Trans Int Soc Music Inf Retr 4(1):167–181. https://doi.org/10.5334/tismir.111
28. Das S, Endert A (2020) LEGION: visually compare modeling techniques for regression. In: 2020 visualization in data science (VDS). IEEE, Salt Lake City, pp 12–21. https://doi.org/10.1109/VDS51726.2020.00006
29. Deng Y (2019) Deep learning on mobile devices: a review. In: Mobile multimedia/image processing, security, and applications 2019, vol 10993. International Society for Optics and Photonics, Maryland, p 109930A. https://doi.org/10.1117/12.2518469
30. Denoue L, Carter S, Kim C (2019) CamaLeon: smart camera for conferencing in the wild. In: Proceedings of the 27th acm international conference on multimedia. ACM, Nice France, pp 1038–1040. https://doi.org/10.1145/3343031.3350583
31. Devarapalli DJ, Mavilla VSD, Karri SPR et al (2021) Classification of skin cancer lesions using deep neural networks and transfer learning. In: Saini HS, Sayal R, Govardhan A et al (eds) Innovations in computer science and engineering. Springer, Singapore, pp 259–268. Lecture notes in networks and systems. https://doi.org/10.1007/978-981-33-4543-0_28
32. Díaz RG, Laamarti F, El Saddik A (2021) DTCoach: your digital twin coach on the edge during COVID-19 and beyond. IEEE Instrum Meas Mag 24(6):22–28. https://doi.org/10.1109/MIM.2021.9513635
33. Dinculescu M, Engel J, Roberts A (2019) MidiMe: personalizing a MusicVAE model with user data. In: Workshop on machine learning for creativity and design. NeurIPS, Vancouver
34. Eltenahy SAM (2021) Facial recognition and emotional expressions over video conferencing based on web real time communication and artificial intelligence. In: Hassanien AE, Darwish A, Abd El-Kader SM et al (eds) Enabling machine learning applications in data science. Springer, Singapore, pp 29–37. Algorithms for intelligent systems. https://doi.org/10.1007/978-981-33-6129-4_3
35. Fan JE, Dinculescu M, Ha D (2019) Collabdraw: an environment for collaborative sketching with an artificial agent. In: Proceedings of the 2019 on creativity and cognition. ACM, San Diego, pp 556–561. https://doi.org/10.1145/3325480.3326578
36. Fang Z, Paliyawan P, Thawonmas R et al (2019) Towards an angry-birds-like game system for promoting mental well-being of players using art-therapy-embedded procedural content generation. In: 2019 IEEE 8th global conference on consumer electronics (GCCE). IEEE, Osaka, pp 947–948. https://doi.org/10.1109/GCCE46687.2019.9015247
37. Fiaidhi J, Mohammed S (2021) Virtual care for cyber– physical systems (VH_CPS): NODE-RED, community of practice and thick data analytics ecosystem. Comput Commun 170:84–94. https://doi.org/10.1016/j.comcom.2021.01.029
38. Filipovic F, Despotovic-Zrakic M, Radenkovic B et al (2019) An application of artificial intelligence for detecting emotions in neuromarketing. In: 2019 international conference on artificial intelligence: applications and innovations (IC-AIAI). IEEE, Belgrade, pp 49–494. https://doi.org/10.1109/IC-AIAI48757.2019.00016
39. Françoise J, Caramiaux B, Sanchez T (2021) Marcelle: composing interactive machine learning workflows and interfaces. In: The 34th annual ACM symposium on user interface software and technology, UIST '21. Association for Computing Machinery, New York, pp 39–53. https://doi.org/10.1145/3472749.3474734
40. Frosst N, Hinton G (2017) Distilling a neural network into a soft decision tree. In: Besold TR, Kutz O (eds) Proceedings of the first international workshop on comprehensibility and explanation in AI and ML 2017, CEUR workshop proceedings, vol 2071. CEUR, Bari
41. Furuya Y, Takashio K (2020) Telepresence robot blended with a real landscape and its impact on user experiences. In:

2020 29th IEEE international conference on robot and human interactive communication (RO-MAN). IEEE, Naples, pp 406–411. https://doi.org/10.1109/RO-MAN47096.2020.9223346

42. Gerard C (2021) Practical machine learning in JavaScript: TensorFlow.js for web developers. Apress, Berkeley. https://doi.org/10.1007/978-1-4842-6418-8
43. Green H, Durrant JD (2021) DeepFrag: an open-source browser app for deep-learning lead optimization. J Chem Inf Model 61(6):2523–2529. https://doi.org/10.1021/acs.jcim.1c00103
44. Guignard M, Schild M, Bederián CS et al (2018) Performance characterization of state-of-the-art deep learning workloads on an ibm" minsky" platform. In: Proceedings of the 51st Hawaii international conference on system sciences
45. Guo Q, Chen S, Xie X et al (2019) An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms. In: 2019 34th IEEE/ACM international conference on automated software engineering (ASE). IEEE, San Diego, pp 810–822. https://doi.org/10.1109/ASE.2019.00080
46. Gupta A, Menon N, Lee W et al (2021) Affective teacher tools: affective class report card and dashboard. In: Roll I, McNamara D, Sosnovsky S et al (eds) Artificial intelligence in education, vol 12748. Springer International Publishing, Cham, pp 178–189. https://doi.org/10.1007/978-3-030-78292-4_15
47. Gupta YM, Homchan S (2021) Short communication: Insect detection using a machine learning model. Nusantara Biosci 13(1). https://doi.org/10.13057/nusbiosci/n130110
48. Hadjar H, Lange J, Vu B et al (2020) Video-based automated emotional monitoring in mental health care supported by a generic patient data management system. In: Gigliotta O, Ponticorvo M (eds) Proceedings of the second symposium on psychology-based technologies, CEUR workshop proceedings vol 2730. CEUR Workshop Proceedings, Naples
49. Hadjar H, Reis T, Bornschlegl MX et al (2021) Recognition and visualization of facial expression and emotion in healthcare. In: Reis T, Bornschlegl MX, Angelini M et al (eds) Advanced visual interfaces. Supporting artificial intelligence and big data applications, vol 12585. Springer, Cham, pp 109–124. https://doi.org/10.1007/978-3-030-68007-7_7
50. Halbhuber D, Henze N, Schwind V (2021) Increasing player performance and game experience in high latency systems. Proc ACM Hum-Comput Interact 5(CHI PLAY):1–20. https://doi.org/10.1145/3474710
51. Hassan J, Leong J, Schneider B (2021) Multimodal data collection made easy: the EZ-MMLA toolkit: a data collection website that provides educators and researchers with easy access to multimodal data streams. In: LAK21: 11th international learning analytics and knowledge conference. ACM, Irvine, pp 579–585. https://doi.org/10.1145/3448139.3448201
52. Hohman F, Kahng M, Pienta R et al (2019) Visual analytics in deep learning: an interrogative survey for the next frontiers. IEEE Trans Visual Comput Graphics 25(8):2674–2693. https://doi.org/10.1109/TVCG.2018.2843369
53. Huang A, Hawthorne C, Roberts A et al (2019a) Bach doodle: approachable music composition with machine learning at scale. In: Proceedings of the 20th international society for music information retrieval conference (ISMIR), pp 793–800
54. Huang CY, Liu L, Chen YL (2019b) An online integrated fingerprint image system. IJMLC 9(1):51–56. https://doi.org/10.18178/ijmlc.2019.9.1.764
55. Huang Y, Qiao X, Tang J et al (2020) DeepAdapter: a collaborative deep learning framework for the mobile web using context-aware network pruning. In: IEEE INFOCOM 2020 - IEEE conference on computer communications. IEEE, Toronto, pp 834–843. https://doi.org/10.1109/INFOCOM41043.2020.9155379
56. Huang Y, Qiao X, Ren P et al (2021) A lightweight collaborative deep neural network for the mobile web in edge cloud. IEEE Trans Mobile Comput:1–1. https://doi.org/10.1109/TMC.2020.3043051
57. Jeong H (2020) Feasibility study of google's teachable machine in diagnosis of tooth-marked tongue. J Dent Hyg Sci 20(4):206–212. https://doi.org/10.17135/jdhs.2020.20.4.206
58. Ionescu TB (2021) Adaptive simplex architecture for safe, real-time robot path planning. Sensors 21(8):2589. https://doi.org/10.3390/s21082589
59. Isakov M, Gadepally V, Gettings KM et al (2019) Survey of attacks and defenses on edge-deployed neural networks. In: 2019 IEEE high performance extreme computing conference (HPEC). IEEE, Waltham, pp 1–8. https://doi.org/10.1109/HPEC.2019.8916519
60. Jayaswal R, Dixit M (2020) comparative analysis of human face recognition by traditional methods and deep learning in real-time environment. In: 2020 IEEE 9th international conference on communication systems and network technologies (CSNT). IEEE, Gwalior, pp 66–71. https://doi.org/10.1109/CSNT48778.2020.9115779
61. Jiang S, Li C, Wang L et al (2021) LatentMap: effective auto-encoding of density maps for spatiotemporal data visualizations. Graphics and Visual Computing 4:200,019. https://doi.org/10.1016/j.gvc.2021.200019
62. Juranek L, Stastny J, Skorpil V et al (2019) Acceleration of server-side image processing by client-side pre-processing in web application environment. In: 2019 42nd international conference on telecommunications and signal processing (TSP). IEEE, Budapest, pp 127–130. https://doi.org/10.1109/TSP.2019.8768889
63. Kabir MM, Ohi AQ, Rahman MS et al (2020) An evolution of CNN object classifiers on low-resolution images. In: 2020 IEEE 17th international conference on smart communities: improving quality of life using ICT, IoT and AI (HONET). IEEE, Charlotte, pp 209–213. https://doi.org/10.1109/HONET50430.2020.9322661
64. Kahn K, Winters N (2021) Learning by enhancing half-baked AI projects. Künstl Intell 35(2):201–205. https://doi.org/10.1007/s13218-021-00732-8
65. Kahng M, Chau DHP (2020) How does visualization help people learn deep learning? evaluating GAN lab with observational study and log analysis. In: 2020 IEEE visualization conference (VIS). IEEE, Salt Lake City, pp 266–270. https://doi.org/10.1109/VIS47514.2020.00060
66. Kahng M, Thorat N, Chau DHP, et al (2019) GAN Lab: understanding complex deep generative models using interactive visual experimentation. IEEE Trans Visual Comput Graphics 25(1):310–320. https://doi.org/10.1109/TVCG.2018.2864500
67. Kanber B (2018) Hands-on machine learning with JavaScript: solve complex computational web problems using machine learning. Packt Publishing, Birmingham
68. Kanda M, Kunze K (2021) Tranquillity at home: designing plant-mediated interaction for fatigue assessment. In: Augmented humans conference, vol 2021. ACM, Rovaniemi, pp 292–294. https://doi.org/10.1145/3458709.3458978
69. Kasthurirathna D, Lokuge K, Mendis R et al (2020) Invasive plant detection and management platform. In: 2020 IEEE international conference on environment and electrical engineering and 2020 IEEE industrial and commercial power systems europe (EEEIC/I&CPS Europe). IEEE, Madrid, pp 1–6. https://doi.org/10.1109/EEEIC/ICPSEurope49358.2020.9160590
70. Khan MS (2020) Using convolutional neural networks for smart classroom observation. In: 2020 international conference on artificial intelligence in information and communication

(ICAIIC). IEEE, Fukuoka, pp 608–612. https://doi.org/10.1109/ICAIIC48513.2020.9065260

71. Klym H (2020) Face detection using an implementation running in a web browser. In: 2020 IEEE 21st international conference on computational problems of electrical engineering (CPEE). IEEE, Pińczów, pp 1–4. https://doi.org/10.1109/CPEE50798.2020.9238754
72. Kritsis K, Kylafi T, Kaliakatsos-Papakostas M et al (2021) On the adaptability of recurrent neural networks for real-time jazz improvisation accompaniment. Front Artif Intell 3:508,727. https://doi.org/10.3389/frai.2020.508727
73. Laborde G (2021) Learning TensorFlow.js. O'Reilly Media Inc., Sebastopol
74. Le H, Nguyen M, Nguyen Q et al (2020) Automatic data generation for deep learning model training of image classification used for augmented reality on pre-school books. In: 2020 international conference on multimedia analysis and pattern recognition (MAPR). IEEE, Ha Noi, pp 1–5. https://doi.org/10.1109/MAPR49794.2020.9237760
75. Lee DJ, Pan TY, Hu MC (2020) Design of identity recognition and liveness detection system for mobile phones. In: 2020 Indo – Taiwan 2nd international conference on computing, analytics and networks (Indo-Taiwan ICAN). IEEE, Rajpura, pp 113–118. https://doi.org/10.1109/Indo-TaiwanICAN48429.2020.9181332
76. Lee Y, Yun S, Kim Y et al (2021) Progressive transmission and inference of deep learning models. In: 2021 20th IEEE international conference on machine learning and applications (ICMLA). IEEE, Pasadena, pp 271–277. https://doi.org/10.1109/ICMLA52953.2021.00049
77. Li C (2019) Web front-end realtime face recognition based on TFJS. In: 2019 12th international congress on image and signal processing, biomedical engineering and informatics (CISP-BMEI). IEEE, Suzhou, pp 1–5. https://doi.org/10.1109/CISP-BMEI48845.2019.8965963
78. Ma Y, Xiang D, Zheng S et al (2019) Moving deep learning into web browser: how far can we go? In: The world wide web conference, WWW '19. ACM, New York, pp 1234–1244. https://doi.org/10.1145/3308558.3313639
79. Meeds E, Hendriks R, Al Faraby S et al (2015) MLitB: machine learning in the browser. PeerJ Comput Sci 1:e11. https://doi.org/10.7717/peerj-cs.11
80. Milkes Espinosa S, Graves J, Towery J (2021) What the flock?: fostering collaborative active breaks for online education. In: Extended abstracts of the 2021 CHI conference on human factors in computing systems, vol 499. ACM, New York, pp 1–6
81. Moll P, Leibetseder A, Kletz S et al (2019) Alternative inputs for games and AR/VR applications: deep headbanging on the web. In: Proceedings of the 10th ACM multimedia systems conference. ACM, Amherst Massachusetts, pp 320–323. https://doi.org/10.1145/3304109.3323832
82. Moreira R, Fialho R, Teles AS et al (2022) A computer vision-based mobile tool for assessing human posture: a validation study. Comput Methods Programs Biomed 214:106,565. https://doi.org/10.1016/j.cmpb.2021.106565
83. Morell JÁ, Alba E (2022) Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices. Futur Gener Comput Syst 133:53–67. https://doi.org/10.1016/j.future.2022.02.024
84. Morell JA, Camero A, Alba E (2019) JSDoop and TensorFlow.js: volunteer distributed web browser-based neural network training. IEEE Access 7:158,671–158,684. https://doi.org/10.1109/ACCESS.2019.2950287
85. Nguyen H, Nguyen M, Nguyen Q et al (2020) Web-based object detection and sound feedback system for visually impaired people. In: 2020 international conference on multimedia analysis and pattern recognition (MAPR). IEEE, Ha Noi, pp 1–6. https://doi.org/10.1109/MAPR49794.2020.9237770
86. Njazi S (2021) Veritas: a sign language-to-text translator using machine learning and computer vision. In: 2021 the 4th international conference on computational intelligence and intelligent systems. ACM, Tokyo, pp 55–60. https://doi.org/10.1145/3507623.3507633
87. Ogunjinmi PD, Park SS, Kim B et al (2022) Rapid post-earthquake structural damage assessment using convolutional neural networks and transfer learning. Sensors 22(9):3471. https://doi.org/10.3390/s22093471
88. Ouyang W, Mueller F, Hjelmare M et al (2019) ImJoy: an open-source computational platform for the deep learning era. Nat Methods 16(12):1199–1200. https://doi.org/10.1038/s41592-019-0627-0
89. Ozarkar S, Chetwani R, Devare S et al (2020) AI for accessibility: virtual assistant for hearing impaired. In: 2020 11th international conference on computing, communication and networking technologies (ICCCNT). IEEE, Kharagpur, pp 1–7. https://doi.org/10.1109/ICCCNT49239.2020.9225392
90. Park HJ, Lee K (2020) Implementation of an open artificial intelligence platform based on web and tensorflow. J Inf Commun Converg Eng 18(3):176–182. https://doi.org/10.6109/JICCE.2020.18.3.176
91. Patel S, Madhani H, Garg S et al (2021) An ai-based solution to reduce undesired face-touching as a precautionary measure for COVID-19. In: Chaubey N, Parikh S, Amin K (eds) Computing science, communication and security. Springer International Publishing, Cham, pp 30–45. Communications in computer and information science. https://doi.org/10.1007/978-3-030-76776-1_3
92. Paudyal P, Lee J, Kamzin A et al (2019) Learn2Sign: explainable ai for sign language learning. In: 2019 joint ACM IUI workshops, ACMIUI-WS, 2019, vol 2327. CEUR-WS, Los Angeles, p 7
93. Pezzotti N, Thijssen J, Mordvintsev A, et al (2020) GPGPU linear complexity t-SNE optimization. IEEE Trans Visual Comput Graphics 26(1):1172–1181. https://doi.org/10.1109/TVCG.2019.2934307
94. Pournaras X (2020) Deep learning on the web: state-of-the-art object detection using web-based client-side frameworks. In: 2020 11th international conference on information, intelligence, systems and applications (IISA). IEEE, Piraeus, pp 1–8. https://doi.org/10.1109/IISA50023.2020.9284358
95. Przybyła P, Soto AJ (2021) When classification accuracy is not enough: explaining news credibility assessment. Inf Process Manag 58(5):102,653. https://doi.org/10.1016/j.ipm.2021.102653
96. Ranasinghe I, Dantu R, Albert MV et al (2021) Cyber-Physiotherapy: rehabilitation to training. In: 2021 IFIP/IEEE international symposium on integrated network management (IM). IEEE, Bordeaux, pp 1054–1057
97. Rao A, Bihani A (2018) Milo: a visual programming environment for data science education. In: 2018 IEEE symposium on visual languages and human-centric computing (VL/HCC). IEEE, Lisbon, pp 211–215. https://doi.org/10.1109/VLHCC.2018.8506504
98. Rick SR, Bhaskaran S, Sun Y et al (2019) NeuroPose: geriatric rehabilitation in the home using a webcam and pose estimation. In: Proceedings of the 24th international conference on intelligent user interfaces: companion. ACM, Marina del Ray, pp 105–106. https://doi.org/10.1145/3308557.3308682
99. Ríos Félix JM, Zatarain Cabada R, Barrón Estrada ML et al (2020) An intelligent learning environment for computational thinking. CyS 24(3). https://doi.org/10.13053/cys-24-3-3480

100. Risal MF, Sukaridhoto S (2019) Web explainer for children's education with image recognition based on deep learning. In: 2019 international electronics symposium (IES). IEEE, Surabaya, pp 406–410. https://doi.org/10.1109/ELECSYM.2019.8901627
101. Rivera JDDS (2020) Practical TensorFlow.js: deep learning in web app development. Apress, Berkeley. https://doi.org/10.1007/978-1-4842-6273-3
102. Roberts A, Hawthorne C, Simon I (2018) Magenta.js: a JavaScript API for augmenting creativity with deep learning. In: Joint workshop on machine learning for music (ICML)
103. Rodrigues R (2022) Interactive intelligent tools for creative processes using multimodal information. In: 27th international conference on intelligent user interfaces. ACM, Helsinki, pp 134–137. https://doi.org/10.1145/3490100.3516479
104. Rodrigues R, Madeira RN, Correia N et al (2019) Multimodal web based video annotator with real-time human pose estimation. In: Yin H, Camacho D, Tino P et al (eds) Intelligent data engineering and automated learning – IDEAL 2019. Springer International Publishing, Cham, pp 23–30. Lecture notes in computer science. https://doi.org/10.1007/978-3-030-33617-2_3
105. Ross A, Chen N, Hang EZ et al (2021) Evaluating the interpretability of generative models by interactive reconstruction. In: Proceedings of the 2021 CHI conference on human factors in computing systems. ACM, Yokohama, pp 1–15. https://doi.org/10.1145/3411764.3445296
106. Sasaki K (2019) Hands-on machine learning with TensorFlow. Js: a guide to building ml applications integrated with web technology using the TensorFlow.Js library. Packt Publishing, Birmingham
107. Schultze S, Gruenefeld U, Boll S (2020) Demystifying deep learning: a learning application for beginners to gain practical experience. In: Hansen C, Nürnberger A, Preim B (eds) Mensch und computer 2020 - workshopband. Gesellschaft für Informatik e.V., Bonn. https://doi.org/10.18420/muc2020-ws111-334
108. Sen S, Bernabé P, Husom EJB (2020) DeepVentilation: learning to predict physical effort from breathing. In: Proceedings of the twenty-ninth international joint conference on artificial intelligence. International Joint Conferences on Artificial Intelligence Organization, Yokohama, pp 5231–5233. https://doi.org/10.24963/ijcai.2020/753
109. Sengupta S, Basak S, Saikia P et al (2020) A review of deep learning with special emphasis on architectures, applications and recent trends. Knowl-Based Syst 194:105,596. https://doi.org/10.1016/j.knosys.2020.105596
110. Sesagiri Raamkumar A, Tan SG, Wee HL (2020) Use of health belief model–based deep learning classifiers for COVID-19 social media content to examine public perceptions of physical distancing: model development and case study. JMIR Public Health Surveill 6(3):e20,493. https://doi.org/10.2196/20493
111. Smilkov D, Thorat N, Assogba Y et al (2019) TensorFlow.Js: machine learning for the web and beyond. In: Proceedings of the 2nd SysML conference, Palo Alto
112. Stoessel J, Collins D (2020) Using optical music recognition to encode 17th-century music prints: the canonic works of Paolo Agostini (c.1583–1629) as a test case. In: 7th international conference on digital libraries for musicology. ACM, Montréal, pp 1–9. https://doi.org/10.1145/3424911.3425517
113. Stratulat-Diaconu A, Cocu A (2020) Classifying skin moles using convolutional neural networks. The annals of "Dunarea de Jos" university of Galati. Fascicle IX, Metallurgy Mater Sci 43(2):9–13
114. Sun L, Zong T, Wang S et al (2021) Towards optimal low-latency live video streaming. IEEEACM Trans Netw:1–12. https://doi.org/10.1109/TNET.2021.3087625
115. Sun TR (2020) FaceAUG: a cross-platform application for real-time face augmentation in web browser. In: 2020 IEEE international conference on artificial intelligence and virtual reality (AIVR). IEEE, Utrecht, pp 290–293. https://doi.org/10.1109/AIVR50618.2020.00058
116. Suryadevara NK (2021) Beginning machine learning in the browser: quick-start guide to gait analysis with JavaScript and TensorFlow.js. Apress, Berkeley. https://doi.org/10.1007/978-1-4842-6843-8
117. Tabatabaei SAH, Fischer P, Wattendorf S et al (2021) Automatic detection and monitoring of abnormal skull shape in children with deformational plagiocephaly using deep learning. Sci Rep 11(1):17,970. https://doi.org/10.1038/s41598-021-96821-7
118. Tallam K, Liu ZYC, Chamberlin AJ et al (2021) Identification of snails and schistosoma of medical importance via convolutional neural networks: a proof-of-concept application for human schistosomiasis. Front Public Health 9:900. https://doi.org/10.3389/fpubh.2021.642895
119. Tsekhmystro R, Oliinyk V, Proskura G et al (2020) Web assembled benchmark for image visual quality assesment, prediction and improvement. In: 2020 IEEE 15th international conference on advanced trends in radioelectronics, telecommunications and computer engineering (TCSET). IEEE, Lviv-Slavske, pp 791–795. https://doi.org/10.1109/TCSET49122.2020.235543
120. Tsuji M, Kubo H, Jayasuriya S et al (2021) Touch sensing for a projected screen using slope disparity gating. IEEE Access 9:106,005–106,013. https://doi.org/10.1109/ACCESS.2021.3099901
121. Tsutsumi K, Goshtasbi K, Risbud A et al (2021) A web-based deep learning model for automated diagnosis of otoscopic images. Otol Neurotol 42(9):e1382. https://doi.org/10.1097/MAO.0000000000003210
122. Vigliensoni G, McCallum L, Fiebrink R (2020) Creating latent spaces for modern music genre rhythms using minimal training data. In: International conference on computational creativity (ICCC). Goldsmiths University of London, Coimbra
123. Wan C, Liu S, Hoffmann H et al (2021) Are machine learning cloud APIs used correctly? In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE). IEEE, Madrid, pp 125–137. https://doi.org/10.1109/ICSE43902.2021.00024
124. Wang T, Kamon M, Okada S et al (2021a) Design and evaluation of an online squat fitness system: lessons learned during the early COVID-19 pandemic in japan. Front Digit Health 3:55. https://doi.org/10.3389/fdgth.2021.679630
125. Wang ZJ, Turko R, Shaikh O, et al (2020) CNN 101: interactive visual learning for convolutional neural networks. In: Extended abstracts of the 2020 CHI conference on human factors in computing systems, CHI EA '20. ACM, New York, pp 1–7. https://doi.org/10.1145/3334480.3382899
126. Wang ZJ, Turko R, Shaikh O, et al (2021b) CNN explainer: learning convolutional neural networks with interactive visualization. IEEE Trans Visual Comput Graphics 27(2):1396–1406. https://doi.org/10.1109/TVCG.2020.3030418
127. Wu SJ, Lin PS, Huang PC et al (2020) Variational-autoencoder-based environment for interactive sketch tutoring aiming for kids. In: 2020 2nd international workshop on artificial intelligence and education. ACM, Montreal, pp 12–17. https://doi.org/10.1145/3447490.3447493
128. Yu H, Gupta A, Lee W, et al (2021) Measuring and integrating facial expressions and head pose as indicators of engagement and affect in tutoring systems. In: Sottilare RA, Schwarz J (eds) Adaptive instructional systems. Adaptation strategies and methods. Springer International Publishing, Cham, pp 219–233. Lecture notes in computer science. https://doi.org/10.1007/978-3-030-77873-6_16

129. Yu W, Lv P (2021) An end-to-end intelligent fault diagnosis application for rolling bearing based on MobileNet. IEEE Access 9:41,925–41,933. https://doi.org/10.1109/ACCESS.2021.3065195

130. Zhaojie D, Chenjie Z, Jiajie W et al (2020). In: 2020 15th international conference on computer science & education (ICCSE). IEEE, Delft, pp 349–352. https://doi.org/10.1109/ICCSE49874.2020.9201690

131. Zheng Y, Chen H, Duan Q et al (2021) Leveraging domain knowledge for robust deep reinforcement learning in networking. In: IEEE INFOCOM 2021 - IEEE conference on computer communications. IEEE, BC, pp 1–10. https://doi.org/10.1109/INFOCOM42981.2021.9488863

132. Znamenskaya I, Doroshchenko I, Tatarenkova D (2020) Edge detection and machine learning approach to identify flowstructures on schlieren and shadowgraph images. In: Proceedings of the 30th international conference on computer graphics and machine vision (GraphiCon 2020). CEUR Workshop Proceedings, Saint Petersburg, pp 15–1 to 15–14. https://doi.org/10.51130/graphicon-2020-2-3-15

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.