Rethinking Model Architecture and Means of Delivery to get Video Interaction with Generative Neural Networks Online

Josh Murr

The lines between scientific researcher and artist are continually being blurred as artists adopt machine learning methods[30], or researchers apply their tools in novel and interesting ways[26] rethinking how cutting-edge research can be applied to create image generation tools or interrogate the tools themselves. *Learning to See* by Memo Akten[1] is one such seminal artwork which does both. Reframing the tools which we are in fact living with day-to-day in the context of the gallery provides an opportunity to think deeper about how a neural network interprets the data it is given and to witness the capabilities and limitations of such a system. *Learning to See* in particular has a uniquely simple interaction allowing the visitor to play with the network in real time. This work shows that simply shrinking the Pix2Pix[20] model which powers *Learning to See*, and a custom made WebGL wrapper for TensorflowJS to make a pipeline more suited to video achieves useable results when served online, even on commonplace integrated graphics processing units. This brings the unique interaction developed by Akten *et al*[2] into the homes of many people and offers exciting possibilites to future video based interaction with machine learing on the internet. The work is currently online here: https://learning-to-learn-to-see.netlify.app/, and code available here: https://github.com/joshmurr/cci-pix2pix-demo/.

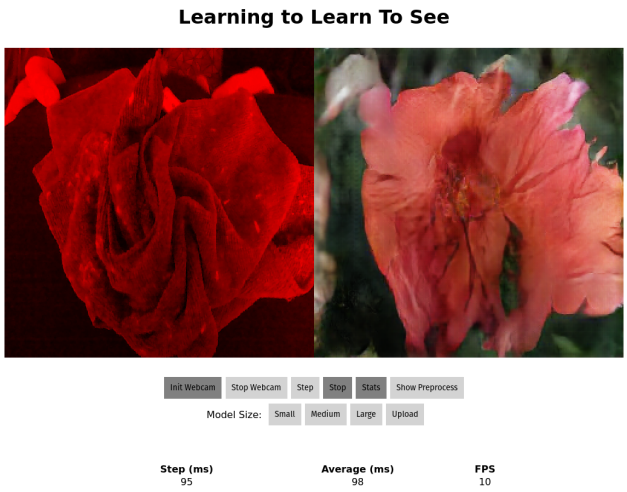

Fig. 1. The online platform running the largest model trained on the *102 Category Flower Dataset*[43] for 250 epochs.

1 Introduction

Machine Learning (ML) is a part of our everyday lives. For the technical among us it is currently easy enough to see the patterns in the outputs of these black-box systems—as the end of our sentence is predicted as we type, or an item we did not know we wanted is offered to us as we shop. There are however many non-technical users of ML systems who can only see the black-box, or perhaps nothing at all; as Arthur C. Clarke famously said *"Any sufficiently advanced technology is indistinguishable from magic."*[42] It is often the case that exciting new technologies are adopted by artists as readily as industry because the gallery provides a unique environment to present back the to the user one of these black-boxes in a drastically different context. Artists such as Mario Klingemann[23], Terence Broad[5] or Anna Ridler[30] are all excellent examples of artists who have done just this.

Learning to See by Memo Akten[1] is another example, but unique in its own way in that it gives the user an interactive experience of the artwork; an immediate image translation of arbitrary objects placed on a tabletop to a rich image of waves crashing, the night sky or ever changing flowers. The user is able to manipulate the input data and witness how the model interprets this to produce an output image in front of them. Although the model still exists as a black box, being able to see how input-affects-output in real time is extremely powerful.

Witnessing the relationship between input and output is in many ways more vital to understanding the function of a Neural Network (NN) than understanding what a neuron is and how backpropagation works. The role of the artist is important here as these artworks have done a huge amount to shed light on dataset bias[10], stimulate conversation about ownership and copyright[31] and show both the capabilities of ML systems *and* the limitations. It is therefore of great importance to make these stimulating artworks accessible to many more people, which is not as simple as it might seem.

Fig. 2. A frame from Learning to See using a model trained on ocean waves. — ©Memo Akten,2017

Unfortunately the gallery in itself is a black box to many, entry restricted by a fee or simply by the location of the gallery. Moving such interactive experiences online is a clear next step as once something is online it is immediately available to anyone with an internet connection — this then poses numerous challenges. Even a modestly sized NN can contain many millions of parameters. Modern computers are able to perform millions of floating point operations per second (FLOPS) but advances in ML research have shown that *deeper* models and more data produces better results[7]; as such, without expensive hardware acceleration from discrete graphics processing units (GPU) or even more modern tensor processing units (TPU), an average modern computer cannot crunch the numbers needed to run a large, deep NN. Even if one *could* perform the necessary FLOPS, storing this number of parameters can take up gigabytes of memory and so downloading the weights and biases of a pre-trained model can be a time consuming process detracting from the real-time focus.

This paper aims to explore each of these challenges and to look at each through the perspective of the artworks previously described with the aim of getting a piece like *Learning to See* online and useable by most with access to the internet.

2 Background Context

Getting NNs working online and on the clients machine is not a new thing and with modern CPUs it has certainly been shown possible to train and run a simple enough model in the browser[22, 4, 8, 25, 9] and many tools exist which aim to visualise the architecture of a model and show the training process[6, 35, 21]. Such tools are invaluable as learning resources for those dedicated to learning about machine learning. However they are often tailored to that use-case expecting the user to have higher motivation for understanding the process, more than the average online 'passer-by'.

There is an increasing number of tools available online which use an interactive machine learning approach allowing the user to retrain a model in real-time to create unique interactions of the users specification[14, 16, 15]. Again this is valuable work and useful to many, but often requires active engagement from the user and some knowledge of the process.

Learning to See provides an instinctive way for the user to interact with machine learning, avoiding technical terms or a prerequisite knowledge which is simply not seen online (at the time of writing). This is largely down to the issues described above, and these are especially applicable to image generative models which will be discussed further below.

Learning to See is an adaptation of the Pix2Pix model[20] which is a Conditional Generative Adversarial Network (cGAN)[12]. Unlike a conventional GAN[13], a cGAN takes in two data samples as input, in the case of the Pix2Pix model it is a *target* image and an *input* image. Pix2Pix also learns a structured loss which means it is able to penalize any possible structure that differs between output and target[20]. This model architecture is extremely flexible and has been shown to be useful for many use cases as detailed in the original paper. However neither the original Pix2Pix paper nor *Learning to See* had an incentive to create a small model; a good result is simply a sharp image with a predictable output from a given input. As stated by the Pix2Pix authors measuring the results of an image generating GAN is an open issue[33, 20] and in an artistic context quantifiying the outcome is largely pointless — a good result is one which delivers the message of the artist.

It is worth noting that other generative models of similar architecture to Pix2Pix have been developed since, notably CycleGAN[48] and GauGAN[28]. However these models are also significantly more complex and did not seem conducive to a lightweight and performant system with the given requirements. Image generative models are notoriously unstable in training, intuition (and Jonathan Hui) notes that it is easier to recognise an image than it is to create one[18]. Therefore traditional NN compression[27] is often unsuited to GANs. There has been work in GAN compression which has yielded impressive results, but it is a quite esoteric, unconventional compression involving programmatically finding alternate model architectures with neural architecture search (NAS) and teacher–student model pairing to distill knowledge into a smaller architecture[24]. Simplicity was the first port-of-call for this work thus methods shown in the aforementioned works remain untested in this context for now.

Designing and training a NN is of course not limited to the big name ML frameworks like Tensorflow[38] or PyTorch[29], but given the modular nature of a NN and the want in most cases to utilise the GPU for faster training and inference, it is no surprise that a limited number of ML frameworks have become the go-to to create and iterate on a novel ML model and to take care of interfacing with a highly optimized backend and the GPU at a lower level. Tensorflow offers a relatively full end-to-end pipeline from designing and training a model in Python, to saving and converting a model into something can be run in the browser using TensorflowJS[40]. A trained model is simply a collection of weights and biases and to use this model on a different device or context, these weights and biases need to be loaded into a model of the same architecture as was used to train the model. The pipeline from training-to-browser is largely about keeping data structures in order and consistent such

Fig. 3. The output of a 'large' model trained on the *102 Category Flower Dataset*[43] for 250 epochs.

that the next process knows how to work with it. As part of the process Tensorflow has developed the TensorflowJS Converter[41] which converts a saved model into a a series of binary files which hold the weights and a `.json` file which is the blueprints of the model. When uploading a model TensorflowJS dynamically creates a succession of routines in the chosen backend (WebGL, WebAssembly, WebGPU, etc.) to take care of specific tasks such as 2D convolutions, batchnorm, max pooling etc.

There exists nothing even similar to *Learning to See* in the catalog of examples provided by TensrflowJS — i.e. a model which takes video as input and produces video as output. Thus the uniqueness of this use case has yielded a number of hurdles to overcome to achieve something usable on the web for *most users* (users with a reasonably modern laptop, produced in the past 5 years).

3 Method

TensorflowJS is currently the most developed framework for deploying models in the browser so working with the Tensorflow pipeline made the most sense. The model for this work was based on the Tensorflow implementation[39] and adjusted according to the *Learning to See* paper: the input and target images are the same, only the input goes through a series of augmentations, details of which can be found in the original paper[2].

The data augmentation requires the generator to take a single channel input. Augmenting the dataset at runtime during training makes forming a dataset very easy as in reality a full dataset is simply a directory of many similar images. As stated earlier, evaluating the model quantitatively by some distance metric has been shown to be of little use, and particularly in this case the quality of the outcome can only really be measured by how well it performs when interacted with. Thus the common process of splitting a dataset into *training* and *testing* data at a 4:1 split was not as important; however approximately 5% of the dataset was reserved for testing after training for visual inspection.

It was found that for a model that takes an input image size of 256×256 pixels and produces an output of the same size, a dataset of 800–1000 images uniformly shuffled with a batch size of 4 was sufficient to produce *interesting* results — in this case *interesting* means: varied enough to provide unique experiences whilst producing a output which is representative of the input data. Differences between the model sizes can be seen in 4. The models were trained on a NVIDIA Quadro RTX 6000 with 24GB memory locally, or a Tesla P100-PCIE with 16GB memory using Google Colab. In either case a usable model will train in a few hours allowing for a relatively quick development cycle.

Generator Layer	Pix2Pix	Our Small	Our Medium
1	64	4	8
2	128	8	16
3	256	16	32
4	512	32	64
5	512	64	128
6	512	128	256
7	512	256	512
8	512	512	512
9	512	256	512
10	512	128	256
11	512	64	128
12	512	32	64
13	512	16	32
14	256	8	16
15	128	4	8

Table 1. Number of filters per layer in the generator. TODO: Replace table with graphic of model architecture, showing difference between model sizes.

The Pix2Pix generator is an Autoencoder similar to the "U-Net" architecture[32] but differs in some key ways, and we offer some further changes to drastically reduce the size. The generator consists of 8 2D convolution layers downsampling the input data in width and height, and 7 transpose convolutions upsampling the data back to original size. Each layer is batchnormalized[19] apart from the first, and layers 8–10 have dropout[37] applied at a rate of 50%. The Pix2Pix architecture includes no avg- or max-pooling as all filter kernels are 4×4 and a stride of 2 is used. This has the effect of downsampling the layer input by half in width and height, thus circumventing the need for any kind of pooling, otherwise known as an *all convolutional net*[36]. The difference proposed in this work is to greatly reduce the number of filters used in each layer. The majority of trainable parameters are found in the middle of the generator as a result of the structure described. By reducing the number of filters in each layer by up to a factor of 8, very usable results were obtained whilst drastically reducing the model size — more discussion on the impacts of a smaller model are found in Results.

Once a model is trained, it is saved using the `HDF5` format (`.h5`) and converted using the TensorflowJS Converter. The Converter provides options for compressing the model of varying levels of severity. It was found that the compression of the model:

1. Does a great deal to reduce the size (in memory) of the model.
2. Does very little to improve performance in inference.
3. Does very little to affect the output of the model.

Therefore the highest compression of `uint8_affine_quantize` can be used to reduce the size by 4×, while maintaining only a very slightly compromised output.

Finally, a bespoke WebGL wrapper consisting of a number of shader programs was created to handle the input data (pre-augmentation to match that in training) and the output data to render the image to the screen as video. It was found that the image handling capabilities of TensorflowJS seem to be tailored to static images. In inference, the data is of the *tensor* data structure and is transformed and manipulated by the model, but at each iteration of the draw cycle a frame of video must be preprocessed and then converted into a format which can be digested by the model, and then the output of the model must be converted back into a data structure which the browser can handle to display as an image, or in our case, video. The internal computation at inference is handled by WebGL shader programs compiled by TensorflowJS, but the input/output was a rather slow process of converting the tensor data back to pixels value-by-value. The input shader programs receives data from the webcam as a 16×16 image and scales it up to 256×256 while applying three gaussian blurs — slightly different to the training process but the effect is the same. The data is extracted using `<WebGL>.readPixels()`, converted to a tensor object and passed to the model to process. The output of the model is a tensor object but already of the structure of a 3-channel image as a result of the architecture of the model itself. The data is easily extracted as a 32 bit float array and passed into a very simple shader program as an RGB texture to be rendered to a canvas element in the browser.

4 Results

Fig. 4. The large model in use running on an NVIDIA RTX 2070.

Model	HDF5 Size (mb)	Uint8 Compressed Size (mb)
Small	25.3	6.3
Medium	67.3	16.9
Large	217.8	54.5

A test-bed was created with the aforementioned processes to test three sizes of model. The project is currently online at https://learning-to-learn-to-see.netlify.app/.

As already stated, the focus was on finding a sufficiently responsive interaction for the user while maintaining enough variety in the output to maintain engagement. The target audience is someone with a relatively modern computer and not necessarily someone with a computer with a high specification GPU. For testing the user is able to download a model size at their choosing or upload their own, and different statistics are displayed.

In the table below *D-GPU* means *Discrete-GPU* and *I-GPU* means *Integrated-GPU*. *FPS* is rounded to the nearest number. All figures are taken from the online platform.

b¿X s¿X 0.8bsss

Machine	Small	Medium	Large
System76 Laptop 2020, D-GPU	50	41	10
System76 Laptop 2020, I-GPU	26	12	0
Macbook Pro 2020, I-GPU	25	15	1
Macbook Pro 2017, D-GPU	40	27	0
Macbook Pro 2017, I-GPU	26	14	0
Macbook Pro 2015, I-GPU	30	18	0
Macbook Pro 2015, D-GPU	27	18	1
MSI Laptop 2016, I-GPU	18	12	1
Pixel 3A Phone	3	-	-
Samsung Galaxy A20e Phone	4	-	-

In general the results are very promising. The average frame rate on an integrated GPU is ∼14FPS which suggests that the average 'everyday' computer will be able to run the *medium* sized model at a useable speed while a gaming-laptop with a discrete GPU can observe good results even on the largest model.

It can be observed that models trained with fewer parameters struggle to generalise. The *102 Category Flower Dataset*[43] has proven to be a useful dataset for testing as it is quite varied; the subject is not always centered in the image, the colours vary drastically and there is great variation in tone and texture throughout. A full sized *large* model is capable of capturing a lot of this texture, however with fewer filters in the smaller models, artefacts of the kernels themselves often appear in the output, as seen in Figure 5.

In training the model is attempting to learn a mapping from a blurry greyscale image to a well defined 3-channel image. As a user interacts using the online platform one can quickly see the effects of bright or dark areas of such an abstract input, the model appears to associate these bright spots with key areas of interest in the output.

Fig. 5. A medium model trained for 200 epochs on the *102 Category Flower Dataset*[43] showing artefacts of the inner filters, perhaps evidence of overfitting given the smaller model size.

With all this in mind one can train a model with a less varied dataset (such as daytime clouds) which still produces compelling results even in the smallest model as the model needs to generalise less. A dataset produced from a timelapse of the Aurora Borealis was used for the smallest model. The smallest model does not produce particularly exciting results, but still does a good job of illustrating the relationship between input and output, and has even been shown to work on mobile devices.

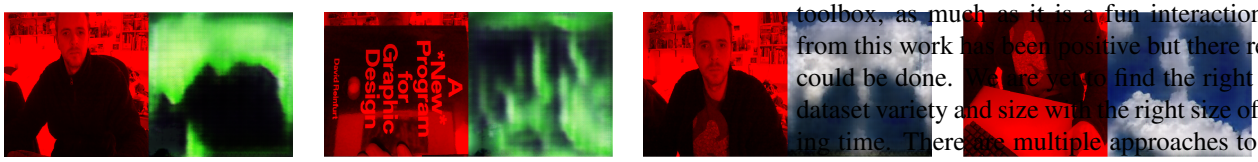

Fig. 6. Small model trained on images of Aurora Boreals (left), medium model trained on images of clouds (right).

5 Discussion

The adoption of ML research into the art world offers a reframing of the application of ML research, which in turn reframes the approach to research itself. The U-Net style, all convolutional network used for the generator in the Pix2Pix model is a particularly graceful NN which at its core is *only* a series 2D convolutions (omitting batchnormalisation and skip connections). This work shows that the network itself need not be complex or large to still produce results which are useable, visually interesting and accessible. As stated, if this work reaches and excites people who previously saw generative ML as off limits, then it is considered a success. Beyond that this work could lead to the development even more artistic tools in the same way that ML5[3], RunwayML[44] and Wekinator[11] have become tools for experimentation among artists and creators.

Given the smaller scale model with this workflow, the development process becomes faster and less daunting, allowing one to engage with the entire process more fully. Neural networks as a *tool*, just like a paintbrush or a camera, is an exciting future in art. The iterative process involved in this work meant creating many datasets and training many models and assessing the outcome qualitatively, this helps develop a sense of *how the model will react to a certain dataset*, and equally an *intuition of the output*. The ability to iterate quickly is how, as a practitioner or an artist, one develops an intuition of ones tools and thus how one learns to control the medium — a painter is not born with the ability to accurately mix color and control a brush, but it is learnt and honed over time.

This work is not only about creating art, or tools for artists. Democratising the process from start-to-finish is important to making the process accessible to all. We currently live with machine learning and artificial intelligence; as there appears to be no limits to the possibilities there is no doubt that it will find its way into even more aspects of our lives. Therefore it is imperative that the field of machine learning becomes more accessible and transparent. The first automobile was met with distrust[**find citation*] and seen as magic, the hope of this work is to dispel some myths of ML by giving the user their first ride in the car.

6 Conclusion

The goal of this work is to inspire intrigue and excitement in users to take up ML as a new tool in their creative toolbox, as much as it is a fun interaction. The feedback from this work has been positive but there remains more that could be done. We are yet to find the right balance between dataset variety and size with the right size of model and training time. There are multiple approaches to this, but a semi-automated training process while qualitatively assessing the results seems the most exciting.

We are also yet to see the perfect means of NN deployment on the internet. TensorflowJS provides great flexibility, but at some performance cost. Tools are being developed to allow internet browsers the ability to utilise low level performance enhancements specific to ML such as XNNPACK[47] while SIMD operations are now available to WebAssembly[34] — both features being utilised to perform background blurring and substitution in-browser and on mobile[17]. WebGPU[45] and WebNN[46] also lie on the horizon showing great promise as web specific ML tools able to handle the data structures of ML. As it stands however WebGL remains the strongest web API to interface with the GPU and with simple models such as the *all convolutional net* it is simply a matter of rethinking tensor data structure into 2D textures to allow for efficient GLSL shader programs to operate on the data. A sample NN of similar structure to the Pix2Pix generator was created to test this hypothesis and shows potential. Early tests show that the running time of a full size Pix2Pix generator model on an integrated GPU is the same as the TensorflowJS model with the WebGL wrapper as used in this work. The advantages of keeping the data structures suited to WebGL means it is trivial to display the weights and biases and possibly create novel real-time interactions with even deeper aspects of the model.

ML on the web is growing rapidly and this work shows that seeking inspiration from unconventional research seen in the art world can lead to exciting findings.

References

[1] Memo Akten. *Learning to See*. 2017. URL: http://www.memo.tv/works/learning-to-see/.

[2] Memo Akten, Rebecca Fiebrink, and Mick Grierson. "Learning to See: You Are What You See". In: (2020). DOI: 10.1145/3306211.3320143. eprint: arXiv:2003.00902.

[3] NYU Interactive Telecommunications/Interactive Media Arts. *ML5*. 2020. URL: https://ml5js.org/.

[4] *Brain JS*. 2018. URL: http://brain.js.org/.

[5] Terence Broad. *Terence Broad*. 2020. URL: https://terencebroad.com/.

[6] Terence Broad. *Topological Visualisation of a Convolutional Neural Network*. 2016. URL: https://blog.terencebroad.com/archive/convnetvis/vis.html.

[7] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. eprint: arXiv:2005.14165.

[8] Juan Cazala. *Synaptic*. 2017. URL: http://caza.la/synaptic/.

[9] Leon Chen. *Keras JS*. 2017. URL: https://transcranial.github.io/keras-js/.

[10] Kate Crawford and Trevor Paglen. *Excavating AI: The Politics of Images in Machine Learning Training Sets*. Sept. 2019. URL: https://www.excavating.ai/.

[11] Rebecca Fiebrink. *Wekinator*. 2009. URL: http://www.wekinator.org/.

[12] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2016. eprint: arXiv:1701.00160.

[13] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. eprint: arXiv:1406.2661.

[14] Google. *Magenta*. 2019. URL: https://magenta.tensorflow.org/.

[15] Google. *Teachable Machines*. 2019. URL: https://teachablemachine.withgoogle.com/.

[16] Mick Grierson. *Mimic*. 2019. URL: https://mimicproject.com/.

[17] Tingbo Hou and Tyler Mullen. *Background Features in Google Meet, Powered by Web ML*. 2020. URL: https://ai.googleblog.com/2020/10/background-features-in-google-meet.html.

[18] Jonathan Hui. "GAN — Why it is so hard to train Generative Adversarial Networks!" In: (2018). URL: https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b.

[19] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. eprint: arXiv:1502.03167.

[20] Phillip Isola et al. "Image-to-Image Translation with Conditional Adversarial Networks". In: (2016). eprint: arXiv:1611.07004.

[21] Minsuk Kahng et al. *ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models*. 2017. eprint: arXiv:1704.01942.

[22] Andrej Karpathy. *ConvNetJS*. 2016. URL: https://cs.stanford.edu/people/karpathy/convnetjs/.

[23] Mario Klingemann. *Mario Klingemann*. 2020. URL: http://quasimondo.com/.

[24] Muyang Li et al. *GAN Compression: Efficient Architectures for Interactive Conditional GANs*. 2020. eprint: arXiv:2003.08936.

[25] Steven Miller. *Mind*. 2016. URL: http://caza.la/synaptic/.

[26] Alexander Mordvintsev. *Alexander Mordvintsev*. 2020. URL: https://znah.net/.

[27] James O' Neill. *An Overview of Neural Network Compression*. 2020. eprint: arXiv:2006.03669.

[28] Taesung Park et al. "Semantic Image Synthesis with Spatially-Adaptive Normalization". In: (2019). eprint: arXiv:1903.07291.

[29] PyTorch. *PyTorch*. 2019. URL: https://pytorch.org/.

[30] Anna Ridler. *Anna Ridler*. 2020. URL: http://annaridler.com/.

[31] Aja Romano. *A guy trained a machine to "watch" Blade Runner. Then things got seriously sci-fi.* 2016. URL: https://www.vox.com/2016/6/1/11787262/blade-runner-neural-network-encoding.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. eprint: arXiv:1505.04597.

[33] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. eprint: arXiv:1606.03498.

[34] WebAssembly SIMD. *WebAssembly SIMD*. 2020. URL: https://v8.dev/features/simd.

[35] Daniel Smilkov and Shan Carter. *Tensorflow Playground*. 2016. URL: http://playground.tensorflow.org/.

[36] Jost Tobias Springenberg et al. *Striving for Simplicity: The All Convolutional Net*. 2014. eprint: arXiv:1412.6806.

[37] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[38] Tensorflow. *Tensorflow*. 2019. URL: https://www.tensorflow.org/.

[39] Tensorflow. *Tensorflow Pix2Pix*. 2019. URL: https://www.tensorflow.org/tutorials/generative/pix2pix.

[40] TensorflowJS. *TensorflowJS*. 2019. URL: https://www.tensorflow.org/js.

[41] TensorflowJS. *TensorflowJS Converter*. 2019. URL: https://github.com/tensorflow/tfjs/tree/master/tfjs-converter.

[42] Alvin comp. Toffler. *The futurists. Edited with an introd. by Alvin Toffler*. Random House, 1972.

[43] Oxford University. *102 Category Flower Dataset*. 2008. URL: https://www.robots.ox.ac.uk/~vgg/data/flowers/102/index.html.

[44] Cristóbal Valenzuela. *RunwayML*. 2020. URL: https://runwayml.com/.

[45] GPU for the Web Community Group. *WebGPU*. 2020. URL: https://gpuweb.github.io/gpuweb/.

[46] Machine Learning for the Web Community Group. *Web Neural Network API*. 2020. URL: https://webmachinelearning.github.io/webnn/.

[47] XNNPACK. *XNNPACK*. 2020. URL: https://github.com/google/XNNPACK.

[48] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2017. eprint: arXiv:1703.10593.