Visualizing Machine Learning Models on Alphanumeric Characters

Ajay Sarjoo
SBU ID: 11162378
ajay.sarjoo@stonybrook.edu

ABSTRACT

As humans, we recognize handwritten items simply by looking at them, which involves coordination between our eyes, optic nerves, and brain. Machine Learning models are able to achieve this as well, though they are more mysterious to non-technical folks. This project aims to demystify how machine learning models see and process information through an interactive webpage that visualizes this information. Users will be able to draw (either with a trackpad or mouse) a single handwritten keyboard character (entire alphanumeric alphabet) on a drawing area, which will then be sent to a backend that houses a LeNet-backed neural network trained on a dataset to process the prediction. This will be sent back to the user, where they will be able to see how the model visualized their input at each layer of the network, what the final prediction was, how confident it was (ex. accuracy), and what other predictions it made.[1]

INTRODUCTION

Neural networks are an advancement in deep learning techniques in which a network of functions are used to understand and translate a data input of one form into a desired output. Inspired by human biology and the way neurons in the human brain work together, they have many applications in speech and image recognition, spam filtering, medicine, finance, and more [2]. The way neural networks work is by learning from label examples during a training phase and formulates a means of using the characteristics of the input to generate the output. Once enough examples have been trained on the network, it can then evaluate new or unseen inputs and generate predictions from it.

In this work, we look at one particular type of neural network called a convolutional neural network (CNN). CNNs follow the basic principle of neural networks in general, but its properties lend itself to image tasks. CNNs take images as input, learn the characteristics of the input image by assigning learnable weights and biases to various aspects of the image. This is done by passing an input image through a series of layers and/or filters to predict the output. This means that CNNs can capture the spatial and temporal dependencies in an input image through these aspects.

To a technical person that has knowledge of deep learning, the information presented thus far would be considered trivial. However, to the common person that may not be technically fluent, or to one that is not familiar with CNNs at all, this would be considered far less trivial. Wouldn't it be great if we had a tool to bridge the gap in knowledge between non-technical users and how CNNs make predictions on inputs through an interactive interface? Performing such a feat would be difficult, as its not like we can pry open a CNN to see what goes on under the hood. In this work, we are going to show you how to overcome that by visualizing what a CNN sees while making predictions at a per-layer view in the neural network, as well as what top prediction was made, and what probabilities are associated with the top predictions. This will all be made interactive through a canvas in the webapp that will allow the user to draw an alphanumeric character of their choice, and see what the CNN predicts after visualizing all the layer outputs.

RELATED WORKS

Visualizing layer outputs from neural networks is something that has been experimented on before. However, we will analyze several works that have explored this topic.

The first is by Qin et. al. in [4]. In their work, they look at how CNNs learn their input features to understand the inner workings of various established architectures, such as AlexNet, VGG, ResNet, among others. The work explores several ways to visualize the predictions of these CNNs by looking at several visualization methods, such as Activation Maximization, Network Inversion, Deconvolutional Neural Networks, and Network Dissection based visualizations. They were able to show that through these visualizations that CNNs have a feature representation mechanism that resembles the organization of the human visual cortex, and were successful in their approach. This provided a starting point for how the model was to be implemented.

Another work that looks at how CNN layers process inputs is by Yosinski et. al. in [5]. Similar to [4], they also look at the visualizations of the intermediate layers, but they specifically focus on two aspects of CNN predictions using an AlexNet-based CNN. The first is the activations produced in each layer as it is processed by video/photo input. The second is the vi-

[1]The code for this project can be found here: https://github.com/asarj/CSE518-Project

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

sualized features of each layer via regularized optimization in the image space. By analyzing the AlexNet inputs using both tools, they were able to produce new insights into how they are processed. In this work, we leverage a similar philosophy to visualizing our images via activations per layer.

One thing that the previous works didn't look at was how CNNs visualized unseen inputs and how the intermediate layers assign importance of individual pixels with respect to the classification decision. The authors of [3] leveraged several methods to generate heatmaps of the predicted images to precisely determine what made the neural network arrive at a particular classification decision as opposed to a sensitivity-based approach or a deconvolutional-based method. As a result, the authors were able to quantify the quality of the heatmaps generated from the model predictions.

METHODOLOGY

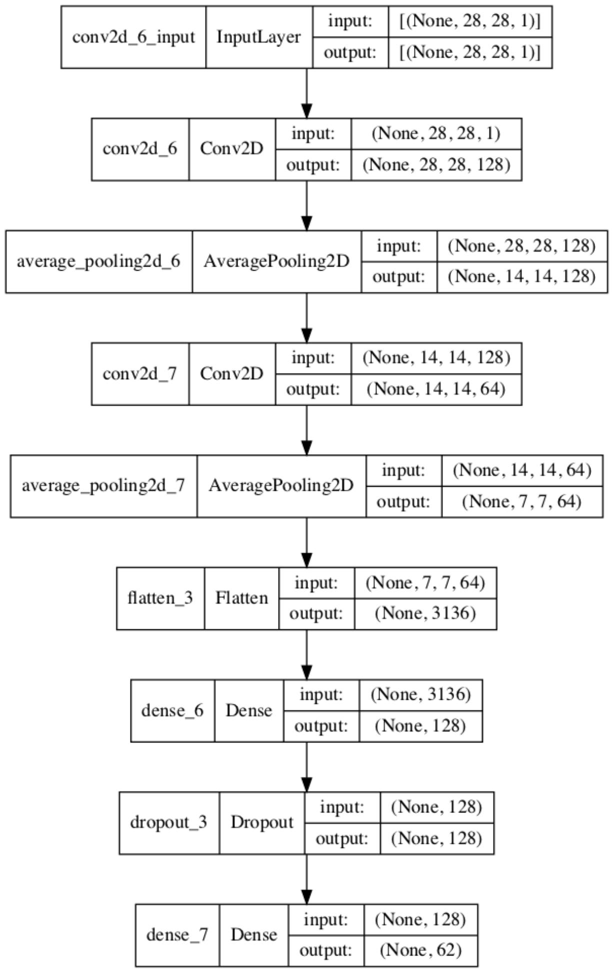
The project was built entirely through the use of the Python programming language. The libraries used were Streamlit, TensorFlow, Numpy, Pillow, and Matplotlib. We will first cover the details of the dataset used, the CNN itself, and integration of human-computer interaction with the webapp interface

Dataset

To achieve the ability of being able to predict handwritten alphanumeric characters in the project, we used the EMNIST dataset from the National Institute of Standards and Techology [1]. In particular, we use the ByClass split of the dataset, which provides 814,255 handwritten characters and 62 classes, and contains handwritten characters from 0-9, a-z, and A-Z.

Model

The CNN used to make predictions on the EMNIST dataset used a LeNet architecture. The LeNet architecture is described in Figure 1.



conv2d_6_input | InputLayer | input: [(None, 28, 28, 1)] | output: [(None, 28, 28, 1)]
conv2d_6 | Conv2D | input: (None, 28, 28, 1) | output: (None, 28, 28, 128)
average_pooling2d_6 | AveragePooling2D | input: (None, 28, 28, 128) | output: (None, 14, 14, 128)
conv2d_7 | Conv2D | input: (None, 14, 14, 128) | output: (None, 14, 14, 64)
average_pooling2d_7 | AveragePooling2D | input: (None, 14, 14, 64) | output: (None, 7, 7, 64)
flatten_3 | Flatten | input: (None, 7, 7, 64) | output: (None, 3136)
dense_6 | Dense | input: (None, 3136) | output: (None, 128)
dropout_3 | Dropout | input: (None, 128) | output: (None, 128)
dense_7 | Dense | input: (None, 128) | output: (None, 62)

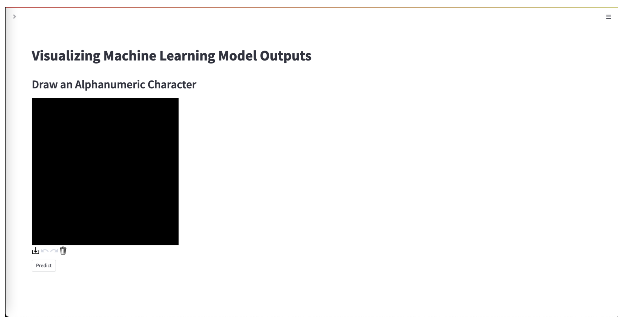
Figure 1. LeNet Architecture

It was constructed using the Keras API in TensorFlow 2.0. During model training, we normalized the pixels in the image by dividing the Numpy image array by 255, and we also extended the EMNIST dataset with augmented images that help with making the model more robust to classifications on unseen inputs. Augmentations to the dataset included shifting the pixels in all directions by 0.2 to account for different ways the user might draw the character on the webapp canvas.

We also use an Adam optimizer and Sparse Categorical Crossentropy for the loss function on this model. We performed a train/test split, and trained it on the extended EMNIST dataset with augmented images for 20 epochs (at which point it early stopped), and reported a training accuracy of 0.8578 and loss of 0.4056. On the testing set, the model achieved an accuracy of 0.8671 and loss of 0.3797, which tells us that our model is performing very well.

User Interface

The user interface is the main aspect of this project. This was built purely through the use of the Streamlit library. Upon visiting or running the Streamlit web application in their browser via localhost, the user will be greeted by the following webapp shown in Figure 2.



Visualizing Machine Learning Model Outputs
Draw an Alphanumeric Character
Predict

Figure 2. Starting Screen of the Webapp

From this interface, the user can draw with their trackpad or mouse a singular alphanumeric character in handwritten form. This drawing area was created from the `streamlit.st_canvas` object provided by the component library. There are also controls for undoing, redoing, and clearing the entire canvas should the user make any mistakes.

The canvas and the ink have also been fixed to be black-and-white in order to match the inputs that the LeNet classifier was trained on. An example drawing is shown in Figure 3.

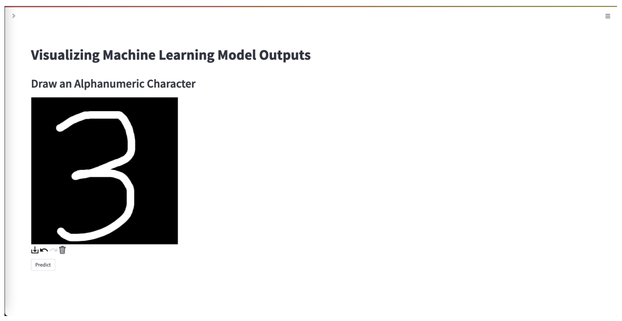
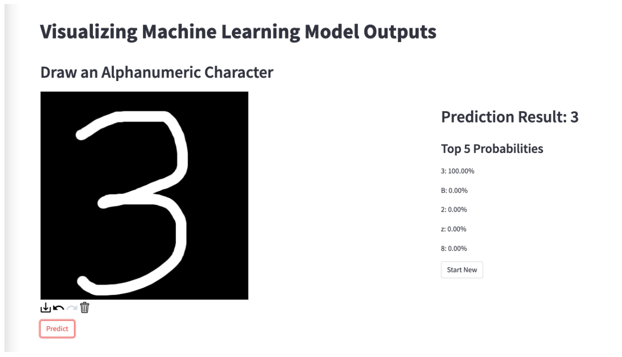

Visualizing Machine Learning Model Outputs
Draw an Alphanumeric Character
Predict

Figure 3. Example of drawing a handwritten character

Once the user clicks on the "Predict" button, the webapp will send the hand-drawn input to the LeNet model that was trained on the EMNIST dataset, get the predictions and feature maps, and send this back to the webapp interface to show the results.

The first thing the user will see is what the LeNet model predicted, as well as the top 5 predictions for the given input, as shown in Figure 4. The prediction is calculated by calling `model.predict` using the Keras API. The top 5 predictions are done by using the `tf.nn.top_k` function call, and specifying $k = 5$ as part of the function parameters. By providing near-immediate feedback with the results, we are able to convey that the model shows the correct output based on their hand-drawn input.


Visualizing Machine Learning Model Outputs
Draw an Alphanumeric Character
Predict
Start New

Prediction Result: 3

Top 5 Probabilities

3: 100.00%

B: 0.00%

2: 0.00%

z: 0.00%

8: 0.00%

Figure 4. Prediction of the LeNet model with top predictions

Upon user scrolling, we then show the detailed activations of the LeNet model layer-by-layer in the order shown of the original LeNet architecture, as shown in Figure 1. In particular, we visualize the feature maps within each layer to give the user a better understanding of how their hand-drawn input is being processed. We leverage the viridis color map to show the intensity of what each layer is analyzing based on the feature map. These feature maps were also generated through the use of the `streamlit.pyplot` module in the component library, and the user can scroll to see the visualizations of each layer in order of the original architecture in the webapp[2].

Figure 5 shows the input layer of the hand-drawn input before passing the data into the next layer.

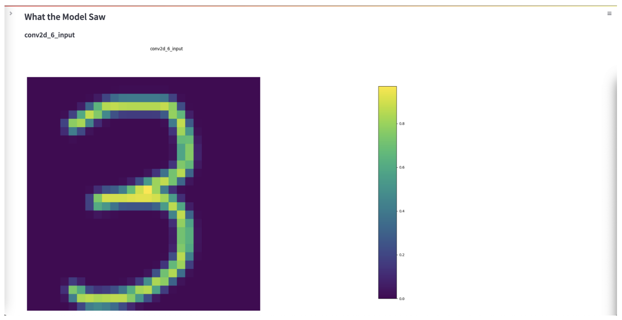


Figure 5. Visualization of the Input Layer

Below this, the user can scroll down to see the feature maps for the next layer in the model, which is a convolutional layer, shown in Figure 6.

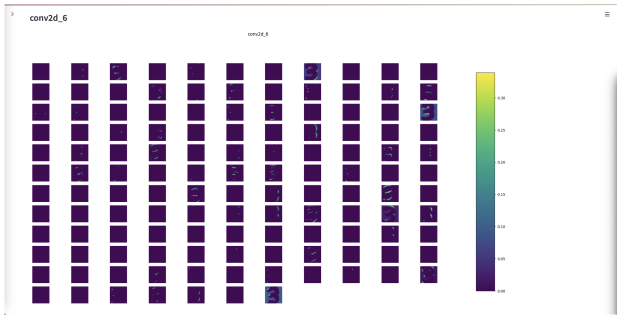


Figure 6. Visualization of the First Convolutional Layer

After this, we show the feature maps of the average pooling layer, in which the user can observe that the feature maps are halved to increase the size, which is also known as downsampling. The resulting feature maps can be seen in Figure 7.

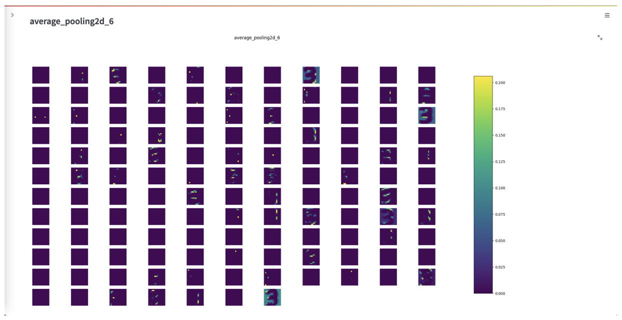


Figure 7. Visualization of the First Average Pooling Layer

Continuing with the LeNet architecture, next is another convolutional layer, which is shown in Figure 8.

[2]If the images for each of these layers are not visible, feel free to visit `https://github.com/asarj/CSE518-Project/tree/master/report` for higher-res images of the same figures

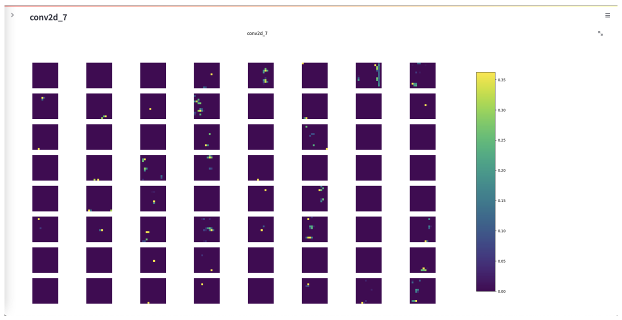


Figure 8. Visualization of the Second Convolutional Layer

Following this second convolutional layer is another average pooling layer, shown in Figure 9.

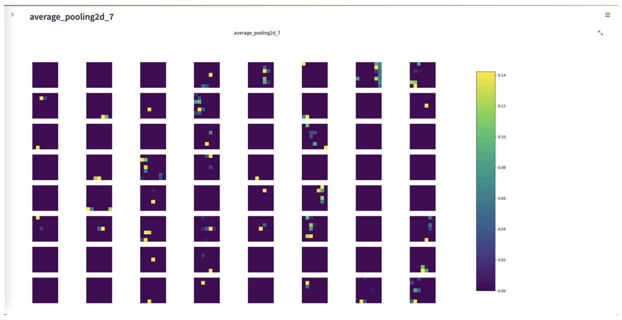


Figure 9. Visualization of the Second Average Pooling Layer

Part of the LeNet architecture includes a flattening layer, which flattens all of the feature maps into a 1-dimensional vector so that this can be fed into successive layers to generate the prediction. It may look like a solid line from this view, but really this is a bunch of features compressed into a really long vector that is 128 units wide, as shown in Figure 10.

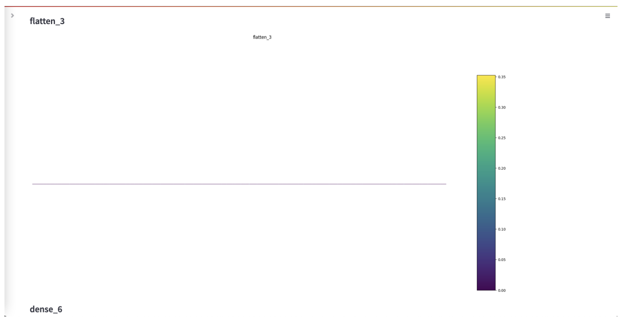


Figure 10. Visualization of the Flattened Layer

After the feature maps have been flattened, the data is then passed into a dense layer, which as a specified number of units to help filter these neurons down into a singular prediction. The output of the dense layer is shown in Figure 11.

As an intermediate step, we use a dropout layer to prevent overfitting the model prediction, which simply drops neurons that would contribute to this based on a proportion specified during the model architecture, which is shown in Figure 12.

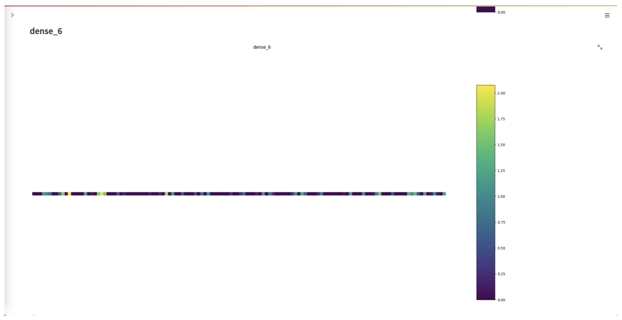


Figure 11. Visualization of the First Dense Layer

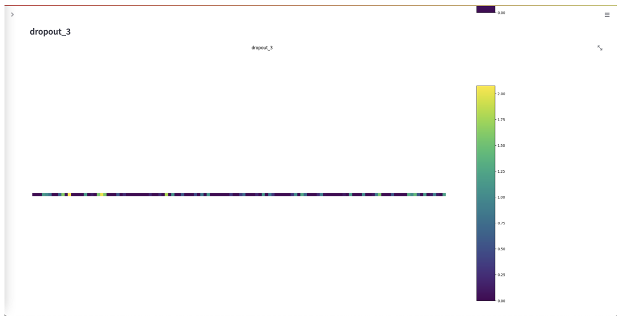


Figure 12. Visualization of the Dropout Layer

Finally, Figure 13 shows last layer in the series of visualizations, which is another dense layer. This layer in particular is special because it has the same number of neurons as the number of classes for the dataset that this model is trained on. As a result of applying the softmax activation on this layer, we see that one position in this feature vector has a value of 1, as indicated by the yellow box, and the others have been squashed to 0. In the case of the initial prediction made, this yellow box corresponds to the number 3 that the model successfully predicted.

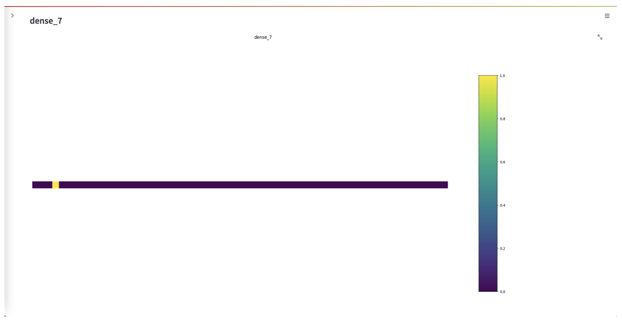


Figure 13. Visualization of the Output Dense Layer

Should the user wish to experiment more with the webapp on more inputs, they can scroll up and click on the "Start Over" button and clear the canvas to continue.

EVALUATION

To assess the success of this design, we performed a heuristic evaluation. By performing such an evaluation, we can better understand how this project influences the understanding of actual users that would stand to benefit from a project such as this to further their understanding of how neural networks work. The results from this study would help us identify minor and major usability issues, and rectifying them to improve upon the existing design.

Evaluation Process

To start, we narrowed down what features of the webapp we wanted to evaluate. We primarily focused on the interfaces ability to clearly visualize what the LeNet model saw in each layer of the model. We sampled 10 students at Stony Brook University that had some familiarity of neural networks, but not to a deep extent. The criteria that each student used to evaluate the interface was based on the heuristics taught in class, which were:

- H2-1:Visibility of system status
- H2-2: Match system and real world
- H2-3: User control and freedom
- H2-4: Consistency and standards
- H2-5: Error prevention
- H2-6: Recognition rather than recall
- H2-7: Flexibility and efficiency of use
- H2-8:Aesthetic and minimalist design
- H2-9: Help users recognize, diagnose and recover from errors
- H2-10: Help and documentation

While performing evaluations on all of these heuristics, the students ranked the severity of each usability heuristic based on the following scale:

- 0 - don't agree that this is a usability problem
- 1 - cosmetic problem
- 2 - minor usability problem
- 3 - major usability problem, important to fix
- 4 - usability catastrophe; imperative to fix

Evaluation Results

After conducting the evaluations with all students, I consolidated the feedback based on the severity ratings of each heuristic. After identifying and ranking the severity of our evaluated problems, we focused on the the problems with the highest severity ratings in order to develop a set of results that we can use to improve the webapp.

Finding #1: No help/documentation
Violation: H2-10, Severity: 3

One thing that all of the participants commented on was the lack of documentation detailing how to use the webapp. While it didn't take them long to figure out how to use it and what it shows, they all agreed that guided instructions would be a nice to have

Fix: To address this problem, we added in minor instructions to the UI to help guide what the user should be doing

Finding #2: Disallowing Non-Alphanumeric Characters
Violation: H2-5, Severity: 2

Some participants tried drawing inputs that weren't alphanumeric characters, or tried to "hack" the model input and only pass in a dot on the canvas or something equivalent. While the backend certainly does not crash, the model will report predictions on an input that it was not trained on, which could confuse some users

Fix: To address this problem, we checked the probability of the predicted class for the given hand-drawn sketch, if that probability did not meet a certain threshold of confidence, then we report that the image is not a valid input to the user in the frontend.

Finding #3: No Dark Mode
Violation: H2-8, Severity: 1 While the users had no complaints about the user interface and its usage, some participants preferred a dark mode to use, given that it would be easier on the eyes and look nicer to some

Fix: To address this problem, I added a dark mode option in Streamlit for the user to be able to toggle in the application settings in order to suit their preferences.

All violations aside, all participants reported to have gained a deeper understanding into how CNNs "see" and process inputs to generate a prediction. Most of the positive feedback came from consistency, UI choices, and quality of feature maps shown. We have also implemented the fixes to the above problems, thus improving the webapp's overall design.

CONCLUSION

Visualizing the inner workings of a neural network, much like understanding how a brain processes what it sees, are often mysterious to non technical folks. Through this project, we have demonstrated that it is possible to construct a medium that allows for non-technical folks to see what neural networks see in order to make predictions. We have incorporated elements from Human-Computer Interaction and deep learning to create a drawable interface based on an optical character recognition task to help end users understand how machine learning models work on unseen data. By showing the feature maps of each layer in the model, we are able to convey how a LeNet CNN transforms its input into an actionable prediction. We also performed a heuristic evaluation on this project with a select group of participants, and we were able to successfully improve upon the existing project to make it more informative and easier to use.

REFERENCES

[1] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373

[2] https://deepai.org/machine-learning-glossary-and-terms/neural-network

[3] W. Samek, A. Binder, G. Montavon, S. Lapuschkin and K. Müller, "Evaluating the Visualization of What a Deep Neural Network Has Learned," in IEEE

Transactions on Neural Networks and Learning Systems, vol. 28, no. 11, pp. 2660-2673, Nov. 2017, doi: 10.1109/TNNLS.2016.2599820.

[4] Qin, Zhuwei, et al. "How convolutional neural network see the world-A survey of convolutional neural network visualization methods." arXiv preprint arXiv:1804.11191 (2018). https://arxiv.org/pdf/1804.11191.pdf.

[5] Yosinski, Jason, et al. "Understanding neural networks through deep visualization." arXiv preprint arXiv:1506.06579 (2015).