# Final Report
Neural Network visualization and implementation

Černý Prokop, Kopecká Eliška

## 1 Introduction

Neural networks have gained significant popularity over the last decade, thanks to their prowess in a variety of machine learning tasks. But understanding why a network produced a particular output proves challenging, and neural networks are therefore often treated as black boxes. In spite of the fact that methodologies of how such networks are trained (e.g. gradient descent) are well understood, the sheer amount of learned parameters and their often complex non-linear relations proves very hard to decipher.

There has been a large amount of research into visualizing neural networks, with the goal of gaining insight about multitude of things about a neural network model, from visualizing the training process to aid model creation, to helping us better understand the decision process used by a trained model, to either debug the model or explain why it made a particular decision.

The goal of our project is to visualize neural network which will classify handwritten digits. More precisely we will implement a neural network which will recognize digits on handwritten images and design and implement insightful visualizations of the network.

In this project, we shall first explore current methods of neural networks visualization, overviewing where, why and how will visualizing help us understand a neural network. Then we will analyze and describe visualization methods which would best help us visualize how neural network for recognizing handwritten digits works. Finally, we will implement our neural network, which will recognize handwritten digits and design and implement visualizations using some of the analyzed methods.

### 1.1 MNIST Dataset

We will explore a neural network trained on the MNIST dataset [1], which is a collection of 60 000 greyscale images of handwritten digits. It is a post-processed subset of the much larger NIST dataset, where each digit has been normalized and centered, to enable beginner machine learning practitioners learn how to create ML models on real-world data.

All images have uniform dimensions of 28x28 pixels, with each pixel represented as an 8-bit intensity value.

## 2 Neural networks visualization goals

When creating a visualization, it is important to distinguish, what purpose should the visualization serve. Based on surveys done by Hohman, Kahng, Pienta and Chau [2] and Choo and Liu [3], we present several main strategies for visualizing neural networks.

A common and already mentioned strategy is focusing on helping the user understand the model's decision-making process better. This increases model transparency and helps to open the black box, which neural network can seem like. Understanding the model is especially important if it is used to evaluate or classify people, because the decision process of the network might be discriminating, such as in the notorious Apple Card case [4].

Other common strategy is visualizing a network for educational purposes. The audience of these visualizations are people possibly with technical background but without proper neural network knowledge, therefore it should visualize how a neural network works at a high-level.

Next frequent goal is to help the user identify and address issues the model has - debug the model. This could for example mean uncovering why a model does not successfully converge or helping to increase the network's performance. The audience are model developers, who have strong neural network understanding and therefore these visualizations can be very technical and provide low-level understanding, such as the loss value, activation and the gradient value of a particular node.

Last strategy we considered was a refinement strategy which focuses on improving and steering the neural network's decision-making process. These visualizations require in depth understanding of the domain and they provide an intuitive overview of the network and its decision-process, while offering interactive tools to steer the network.

### 2.1 Our strategy

Since we are interested in visualizing how a neural network for recognizing handwritten digits works, our focus will be on model debugging and understanding. Therefore we will further explore only techniques which correspond with our focus.

## 3 Neural networks visualization methods

### 3.1 Weight visualization

One of the first steps to visualizing a neural network is to visualize the learned model parameters. A neural network can be thought of as a directed acyclic graph, with neurons of a layer being vertices of a graph and the learned weights being edges connecting the neurons. In this representation, we can visualize the learned weights as the thickness of an edge between neurons and/or as the color of the edge (fig. 1). This can be useful for detecting important neurons, as well as for finding unused (so called dead) neurons.

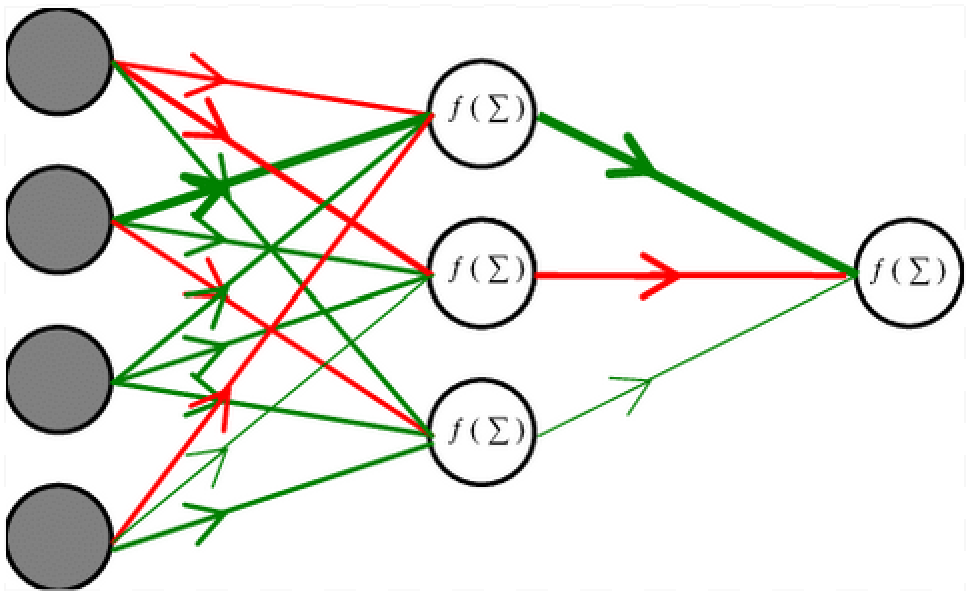


Figure 1: Learned weights visualized by edge thickness and color indicating positivity of the weight.

Such visualization is fine for smaller networks, however, for larger networks, it can quickly become illegible as the number of neurons and connections between them grows. This problem can be mitigated by allowing interactive zooming on the graph, as well as expanding neurons on demand, so the number of visible connections at any one time is manageable.

### 3.2 Visualization of neuron activations

After investigating the learned structure of the model, we can investigate neuron activations to aid our model understanding. This can be done for a single input, or computing average neuron activation for a group of inputs. This kind of visualization should help us identify patterns in the activations. For example, we would expect that activation patterns for different images of the digit 1 would be quite similar, while being a bit less similar to activations for the digit 7, and be quite dissimilar to for example 0.

One approach to visualize the activations is to use the networks graph structure again, same as when visualizing the learned weights, but instead of focusing on the edges, the computed activations can map to color and/or size of the neurons (vertices in the graph).

The same problem with potential visual clutter can also occur here as well. Except for zooming and panning across the network, an alternative can be used here to visualize the neuron activations just for a selected layer (fig. 2), greatly reducing the visual clutter by narrowing down the visualization

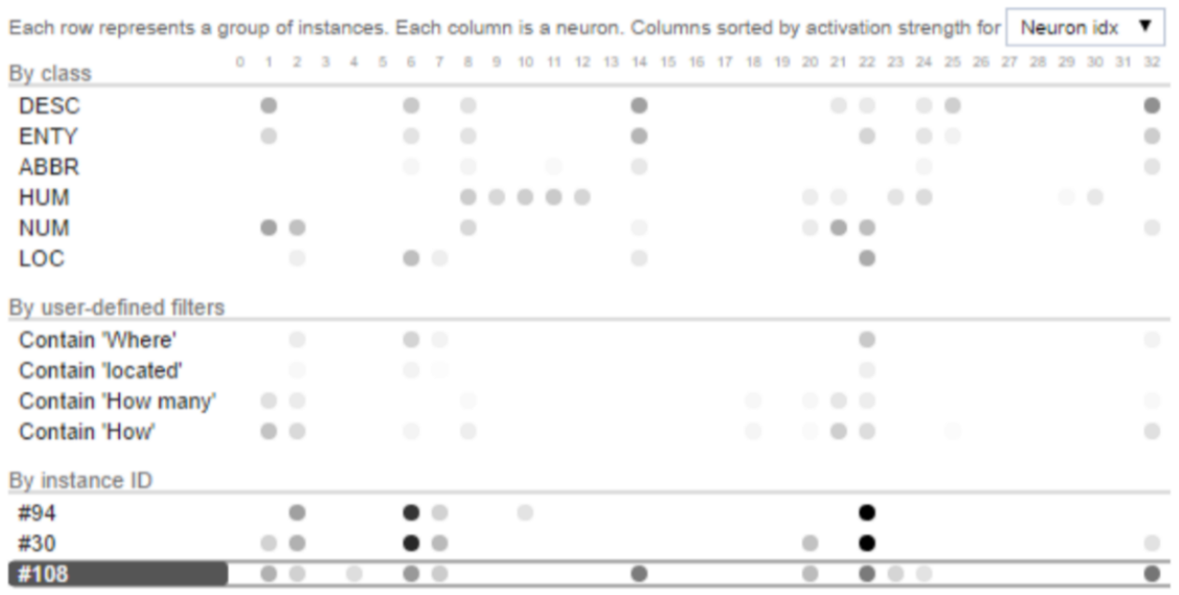

Each row represents a group of instances. Each column is a neuron. Columns sorted by activation strength for Neuron idx

Figure 2: Neuron activations visualized for a single layer in ActiVis [5]. Each column corresponds to a neuron, and each line corresponds to average activations for either a subset of inputs or activations for a particular input, with strength of the activation indicated by saturation of the dots.

When visualizing a single layer, the neurons can be laid out in a line, or even a 2D image as an activation map, although care is needed with fully connected layers because the resulting activation image does not correspond in structure to the original image input of the network and can be confounding, as can be seen in fig. 3.

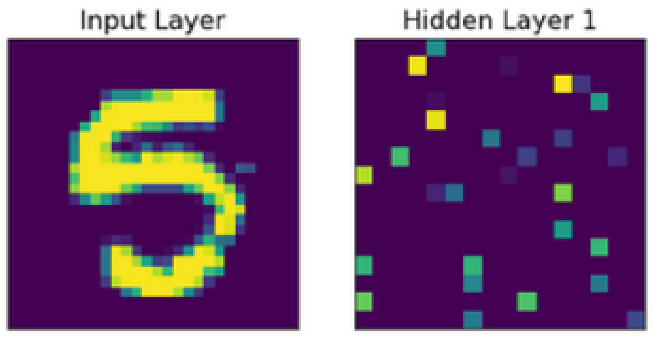


Figure 3: Example of visualizing a fully connected hidden layer with 256 neurons as a 16 by 16 activation grid (illustration credit [6]). As each neuron is connected to all input pixels, any possible structure in the layout of the activation map is not spatially related to the input picture.

## 3.3 Visualization of layer output

In addition to visualizing activations and weights, we are interested in examining what has a network learned at a given layer. A layer's output can be interpreted as a point in high dimensional data, specifically 10-dimensional for our digit-classifying network's output layer, where each input image maps to a particular point. To visualize such spaces we can use the well-known techniques of dimensionality reduction, such as PCA, LDA, Isomap or t-SNE, to project the data to more human readable two or three dimensions.

To get a glimpse into a layer's learned space after dimensionality reduction, we can plot the resulting data as a scatter plot, where each point corresponds to one input image from the dataset. Using this knowledge we can color each projected point using it's label (fig. 4), to see whether the same digits form clusters in the network's learned space. Low separation of the clusters or even a lack of any clusters can indicate that a network did not have enough time to learn.

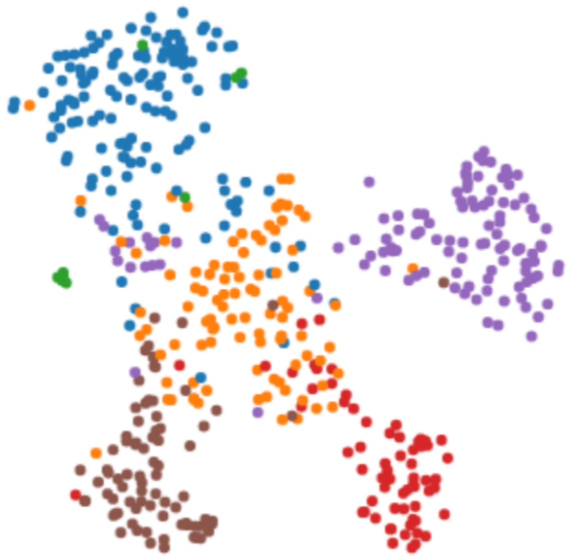


Figure 4: Example of t-SNE projection of a classification problem with 6 classes into 2D space. Points of the same class share color. It can be seen that the network learned satisfactory clusters for most of the classes.

## 3.4 Convolutional neural network visualization

The neural network in our project should process images. For analyzing visual inputs, such as image recognition, convolutional neural networks are frequently used. Since we do not know yet which type of neural network we will use in the project, it is interesting to look at problematics and methods for visualizing convolutional neural networks specifically.

Convolutional neural networks contain hidden convolutional layers, in which each neuron represents a filter. Filter is represented by a weight vector of specified size, which provides a measure for how well a patch of input resembles a pattern/feature. Input for each filter is the output of the previous layer and the filter convolves across each filter-sized block of pixels to locate searched pattern in the input. From visualization perspective we therefore face another challenge, how to visualize each filter - the feature it captures, its occurrence in the data and effect on the result.

### 3.4.1 Feature visualization by optimization

The main idea of this method is to find and show what kind of input causes a particular behaviour - whether that behaviour is final result or an internal neuron firing. This method helps us understand what a model is really looking for, because it isolates the causes of behavior from mere correlations (see fig. 5). This can be achieved by finding the inputs which maximize a given neuron's activation.

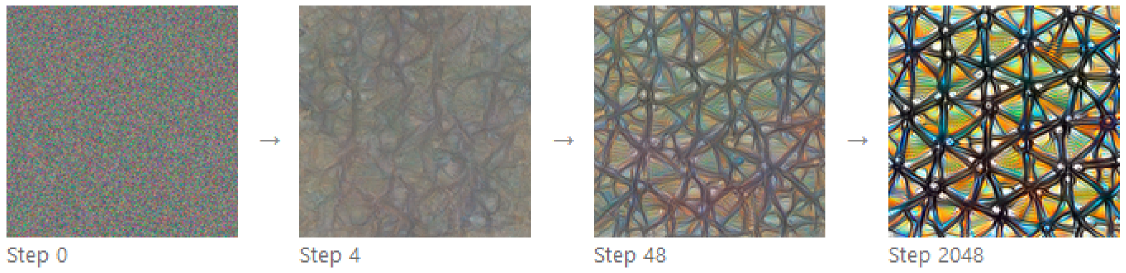


Figure 5: Starting from random noise, we optimize an image to activate a particular neuron.

Basic concept could use direct gradient ascent, but generated images are not very informative, because this approach is sensitive to noise and high-frequency patterns. Therefore we use different techniques to get useful visualizations.

There are multiple methods which reduce high frequencies. As demonstrated by C. Olah, A. Mordvintsev, L. Schubert in [7], frequency reduction can be done either in the visualization or in the gradient using either by one of various model regularization techniques (such as frequency penalization, transformation robustness or learned priors) for reducing high frequencies in the visualization, or by preconditioning for gradient transformation.

These methods can also be combined for even better results, as is demonstrated in figures fig. 6 and fig. 7 from the same resource. There is a visible difference between basic gradient ascent and combination of regularization technique and preconditioning.

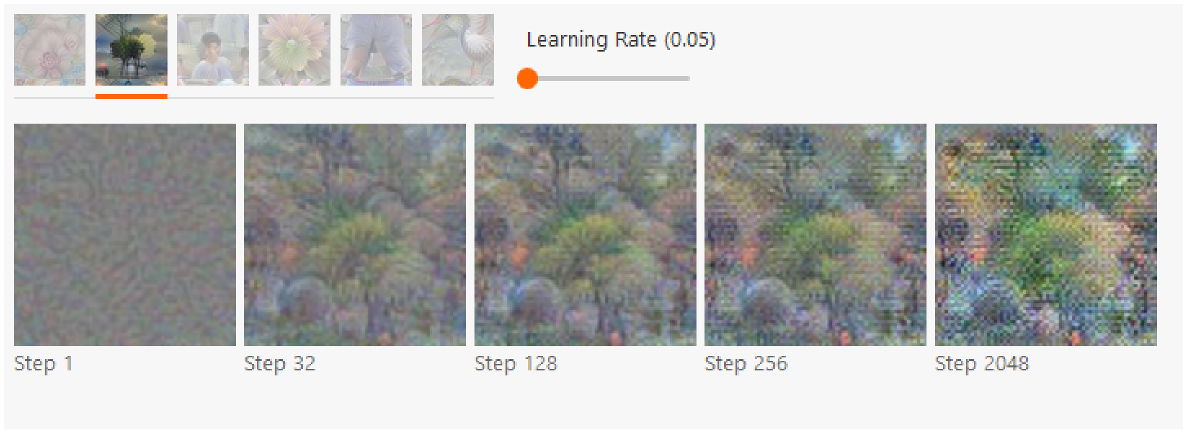


Figure 6: Optimization results using simple gradient ascent.

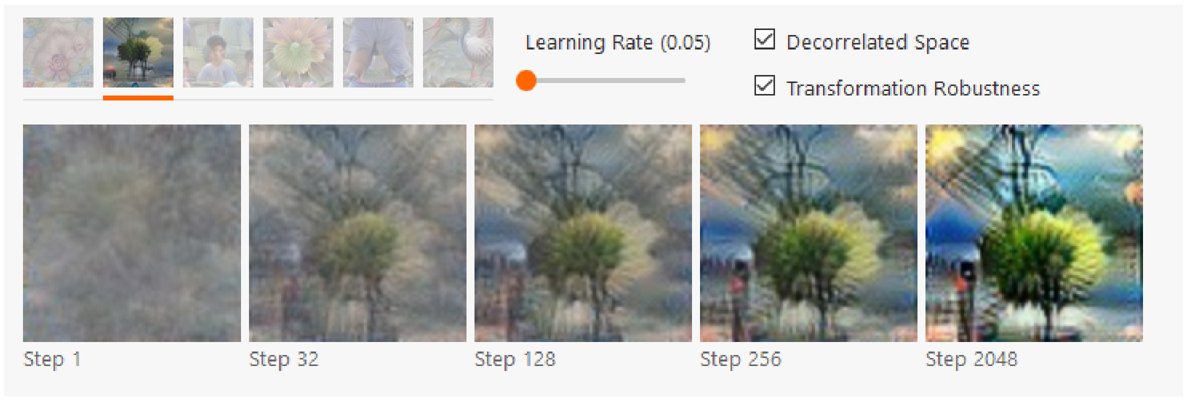


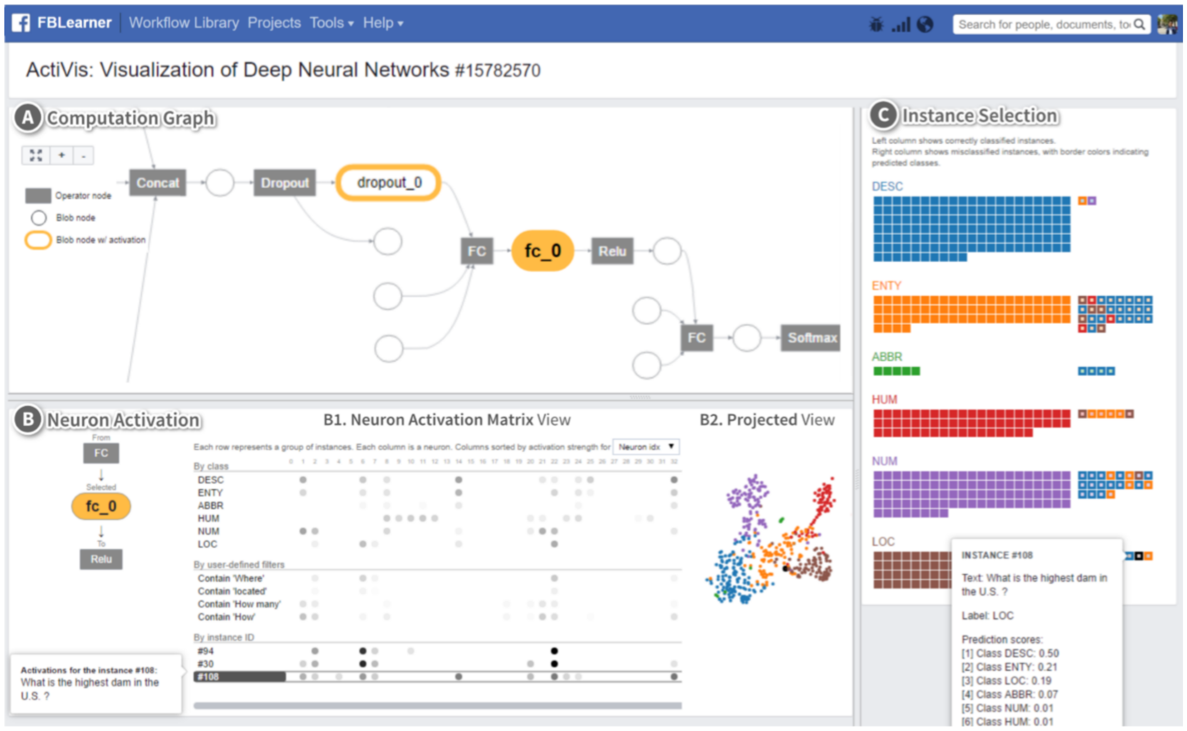
Figure 7: Optimization results using combination of preconditioning and transformation robustness (regularization technique).

## 4 Existing neural network visualization tools

### 4.1 ActiVis

One of the visualization tools designed to help understand how complex machine learning models work is ActiVis [5]. It is an internal tool at Facebook designed to help examine model creators how their models work and perform. The tool is designed not just for deep neural models, but also for general machine learning models, provided they implement an interface for ActiVis.

This tool is focused on examining the workings of a model provided a set of testing data from the user. When using the tool, user can select particular instances from their testing data, or even whole subgroups of instances to see how they flow through the model. The model is presented to the user as a computation graph (fig. 8), which shows the flow of data through the model and the operations performed on the data. User can select nodes corresponding to outputs of operations in the model to examine what happens to the data after each part of the model. Selecting a node in neural network models shows an activation visualization, as we've discussed in section 3.2, specifically fig. 2, as well as a projection of the data in the current layer to 2D, as was discussed in section 3.3.



Each row represents a group of instances. Each column is a neuron. Columns sorted by activation strength for

Activations for the instance #108:
What is the highest dam in the U.S. ?

Left column shows correctly classified instances.
Right column shows misclassified instances, with border colors indicating predicted classes.

INSTANCE #108

Text: What is the highest dam in the U.S. ?

Label: LOC

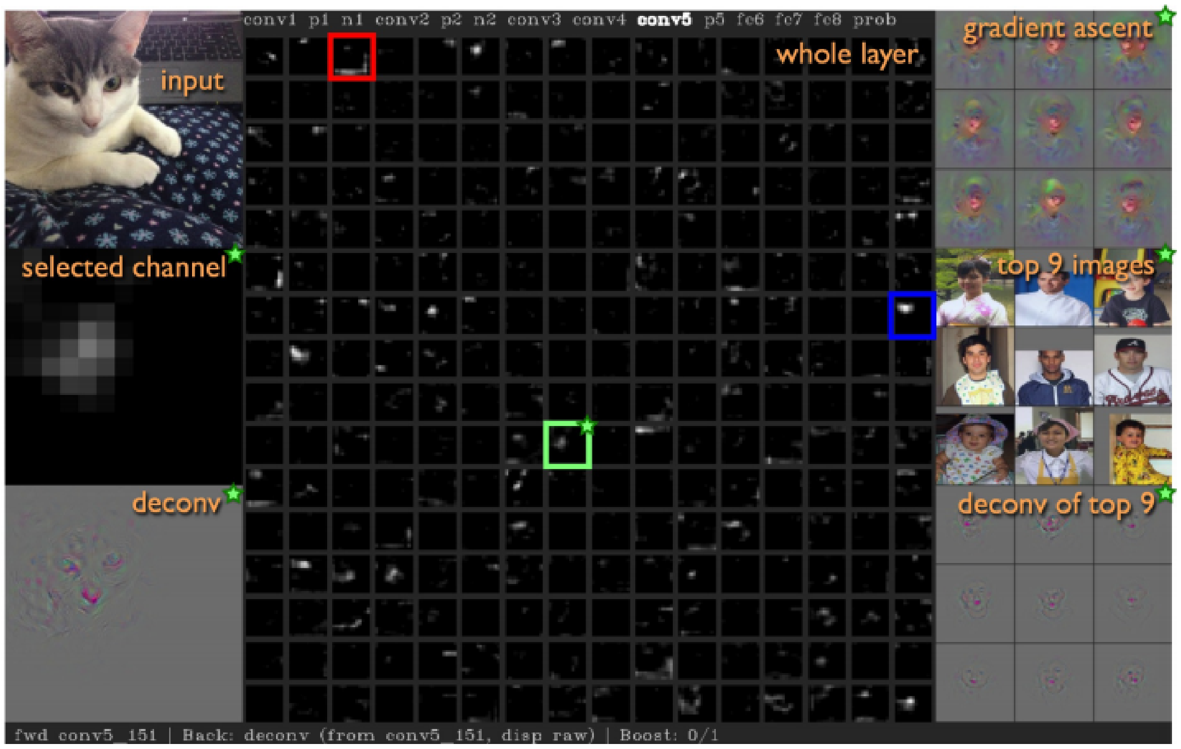
Prediction scores:
[1] Class DESC: 0.50
[2] Class ENTY: 0.21
[3] Class LOC: 0.19
[4] Class ABBR: 0.07
[5] Class NUM: 0.01
[6] Class HUM: 0.01

Figure 8: Overview of ActiVis's interface, showing the structure of a model as a computation graph (**A**), visualizing activations in for selected data operation (**B**), here visualizing activations of a fully connected layer, as well as a projection of the learned space at the current layer. Finally, the right-side panel (**C**) provides an overview of the testing data and enables selecting particular test instances or groups of instances for closer inspection.

ActiVis's approach enables model designers to quickly see what instances were classified correctly or incorrectly and quickly examine activation patterns for them to identify what might have caused the error, be it underfitting, high similarity in structure to a different class or other causes.

## 4.2 DeepVis

A tool [8] that offers two main functionalities. It dynamically visualizes the activations produced on each layer of a trained convolutional neural network as it processes an image or video in real life. And it also enables visualization of features at each layer of the network by applying regularized optimization method discussed above.



```
fwd conv5_151 | Back: deconv (from conv5_151, disp raw) | Boost: 0/1
```

Figure 9: Screenshot from DeepVis [8]. The webcam *input* is shown, along with the *whole layer* of conv5 activations. The *selected channel* pane shows an enlarged version of the 13x13 $\text{conv5}_{151}$ channel activations. Below it, the *deconv* starting at the selected channel is shown. On the right, three selections of nine images are shown: synthetic images produced using the regularized gradient ascent methods as mentioned above, the *top 9 image* patches from the training set (the images from the training set that caused the highest activations for the selected channel), and the *deconv of the those top 9 images*. All areas highlighted with a green star relate to the particular selected channel, here $\text{conv5}_{151}$; when the selection changes, these panels update.

# 5 Implementation

We have decided to implement our project in Python and the resulting product is split into two parts - technical part and user interface.

Technical part contains the logic of the project logic and its computations, such as data preparation, implementation and execution of our neural network, as well as processing of the results, by which we mean extraction of the network's features, weights and predictions and their transformation, such that they can be visualized.

User interface is then a standalone application, which displays the results of the project in form of visualisations.

Since the technical part contains heavy computations and can therefore be expensive to execute, it can be found in enclosed Jupyter notebook and we hand it in already executed. It is therefore possible to run the user application even without executing the whole project code base and for this reason, we also decided to structure the rest of the report such that it is intuitive for the application user and we describe all visualizations in the fitting chapter of application section (section 5.3).

## 5.1 Neural network

As was already mentioned, we have decided to implement our project in Python. To create the neural network we have made use of the TensorFlow Neural Network backend to facilitate computation and training. To create the network we have used the Keras Deep Learning frontend.

As our task is to recognize handwritten digits, we have decided to use a partially convolutional neural network, as these excel in image recognition task. The structure of our network is as follows

1. Input layer - takes in images of size 28x28
2. Convolutional layer 1
    - 16 kernels of size 3x3.

- No padding.
- Outputs 16 26x26 images for each image.
- ReLU activation.

3. Convolutional layer 2
    - 32 kernels of size 3x3.
    - No padding.
    - Outputs 32 24x24 images for each input image.
    - ReLU activation.
4. Maxpooling layer
    - 2x2 maxpooling
    - Reduces each non-overlapping 4-region into its max value.
    - Outputs 12x12 images from its 24x24 inputs.
5. Fully Connected layer 1
    - Flattens each input sample into a 1D vector, loses spatial information.
    - Contains 64 neurons.
    - ReLU activation.
6. Fully Connected layer 2
    - 64 neurons.
    - ReLU activation.
7. Output layer
    - Fully connected output layer.
    - 10 neurons, one for each target digit.
    - Softmax activation function, so its output is a probability distribution.

Our network is able to achieve almost a 99% accuracy on test instances, which are 10000 images from the MNIST dataset that were not available in the training subset, e.g. images which the network hasn't observed before.

As a part of this project, we have decided to visualize activations of neurons in fully connected layers as was presented in section 3.2, as well as visualize the output spaces of layers projected into 2D, as we presented in section section 3.3. We have also decided that feature visualization in convolutional layers, as described in section 3.4, would not be in scope of the project, because we do not find it feasible to implement all visualizations mentioned in section 3 in sufficient quality and in time. We think that activation of neurons together with outputs of each layer will provide more insights for decision making then visualization of filters would.

## 5.2 Output data format

As we have decided to separate the neural network component and the visualization into two separate parts, we needed to create a format to exchange data needed for visualization. To that end, we have decided to use the JSON data format, which easily maps to python dictionaries. The toplevel JSON object contains the following keys and respective described values.

- digit_to_instances
  - A dictionary mapping each digit (0 through 9) to a list of strings of instance names of that digit.
- prediction_results
  - A dictionary mapping instance names to a tuple of their real label and the label the network predicted
- activations
  - a nested dictionary from layer names (strings) to a dictionary of activations for the given layer

  - structure of the nested dictionary has instance names (strings) as keys and an array of activations (real numbers) of neurons of that specific instance in the respective layer.
  - It is supported to only include activations for some layers. In our case, as we visualized neuron activations only for fully connected layers, activation data for instances in convolutional layers is not present.
  - all included layers have to include activation data for all instances present in the file, e.g. the application does not support missing data for some instances, if a layer has been included in activations.

- projections
  - a nested dictionary from layer names (strings) to a dictionary of projections into 2D for the given layer
  - the nested dictionary has instance names as keys and maps to projected 2D coordinates of that instance in that layer
  - It is supported to only include projections for some subset of layers, which can differ from the subset of layers included in activations section.
  - Unlike activations, each layer projections can contain data only for some subset of instances.

- images
  - dictionary mapping instance names (strings) to 2D arrays of dimensions 28x28.
  - the arrays are the images from the mnist dataset the given instance corresponds to, or even aggregate instances (we included average digit instance for each digit)
  - the image data has to be a single floating point number representing the luminosity of a given pixel, scaled to the [0, 1] interval.
  - has to contain an image for any instance name used anywhere in the file.

This JSON is then saved into a file named 'NN_data.json', which has to be put into the same directory as the main GUI application .py file.

## 5.3 Application

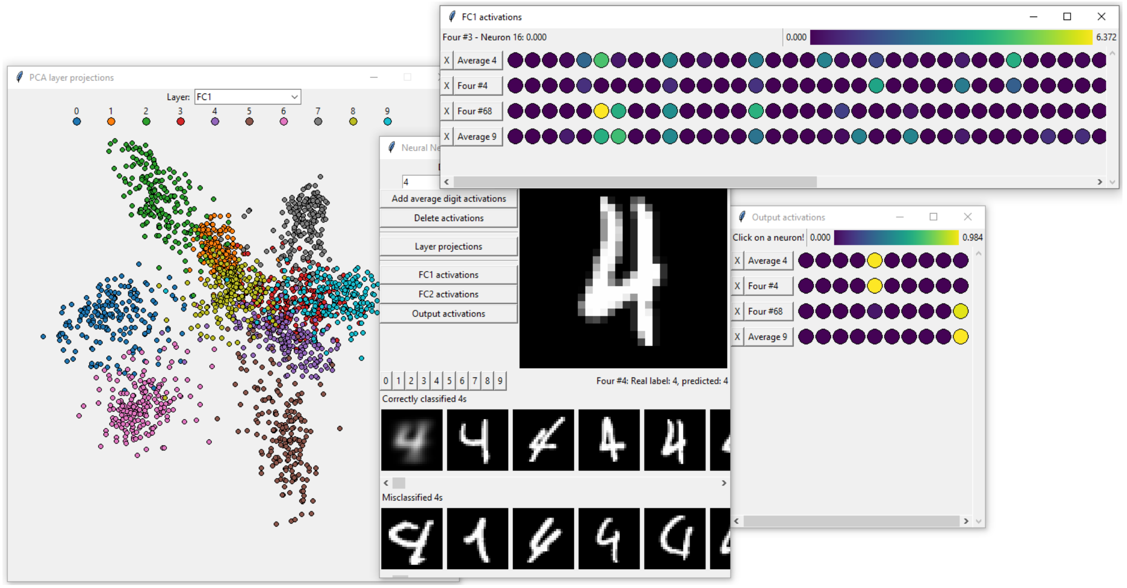


Figure 10: Whole application with all sections open.

We implemented our application in Python, using the built-in crossplatform Tkinter GUI framework. We have decided to structure the application creating separate windows for each visualization activity to offer the user flexibility to structure their view into the data.

Our application dynamically adapts to which data is available in the JSON data file (as described in section 5.2) from the neural network. That means it does not have problems if data for activations is only available for some layers, and projection data is available for some other layers, as nothing is hardcoded, except top level names, as in the data file specification.

### 5.3.1 Main window

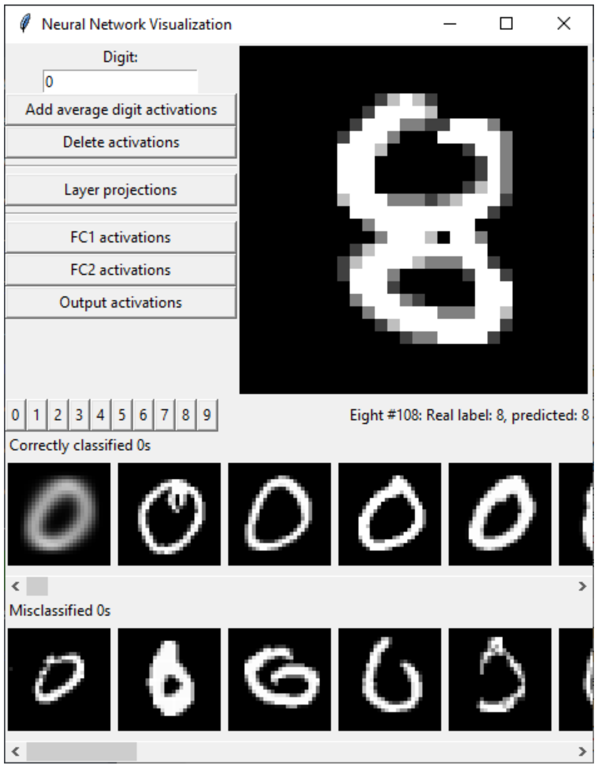


Figure 11: Main window of the app showing the menu for opening particular visualization subwindows on the left side, large preview of last selected digit on the right side, and an image selection gallery on the bottom.

When starting the app, the user is greeted with this window, which serves as a main hub for the application.

At the left side, it contains a menu to open the desired visualization, such as the layer projections window (section 5.3.2), for visualizing the learned feature space of a given layer, and layer neuron activation windows (section 5.3.3), which allow comparing specific neuron activations for different instances (images) from the dataset.

At the right side there is a window showing an enlarged view of the currently selected instance, either from the gallery below, or when clicking on an instance in any subwindows. Underneath this preview there is a label describing the currently selected instance, e.g. its name, real label and the prediction of the network.

At the bottom of the window, there is a gallery for browsing the possible input images, grouped by their real label (the digit contained in the image). For visualization purposes, we have only included a randomly selected subset of 200 instances for each digit for clarity, as the MNIST dataset contains 70000 images in total. In addition to the random subset, we have also included all misclassifications for each digit, as there are not many of them, and are the point of interest for inspecting how and when the neural network fails to make the correct prediction.

The gallery has two rows, first contains the correctly classified instances by the neural network for the currently selected digit, except the first image in this row, which is a proxy instance corresponding to the average for the given digit, more on that in section 5.3.2.

The second row contains all misclassified instances of the currently selected digit, e.g. instances where the real label is the currently selected digits, and the network predicted that it is a different digit.

Single click on any image in the gallery selects the given instance to be previewed in the top-right side of the windows, as well as shows its description right below the large preview. Double click on any image adds it to the instances which are shown in layer activations windows.

### 5.3.2 Layer Projections window

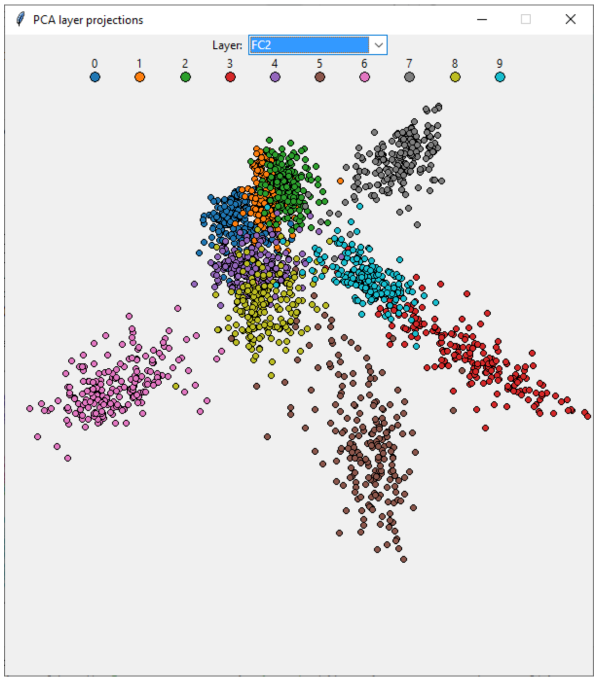


Figure 12: Window for showing 2D projections of learned feature space of the selected layer. At the top there is a dropdown selector of the layer which the user wants to project into 2D, and below there is a scatterplot showing the 2D projection of the possibly very high dimensional feature space.

This window facilitates the projections of learned feature spaces of layers of the neural network as described in section 3.3.

Feature spaces of layers can have very high dimensions. In fully connected layers, the dimensionality corresponds to the amount of neurons in a layer, while in convolutional layers, the dimensionality roughly corresponds to the amount of kernels times the width times the height of the input images, although the particular dimensionality is not of much importance to us, only that it is very high.

To be able to visualize the learned feature spaces we need to perform dimensionality reduction on them into two or three dimensions, which are most human interpretable. We have chosen 2D projections for our visualization.

For dimensionality reduction technique, we decided to use principal component analysis (PCA), which is a well-known dimensionality reduction method. Its main idea is to find principal components – new orthogonal axes, which have direction of the highest variance, and then project the data on to those axes (we can also see it as fitting a hyperellipsoid to the data).

In terms of implementation, there are two approaches: using properties of eigenvector decomposition or more general solution using singular value decomposition(SVD). We chose implementation with SVD because its execution is faster.

After we have performad dimensionality reduction, we are able to view it as a scatterplot in this window. The user selects the desired layer to visualize using the dropdown menu at the top, which is then shown in the scatterplot below.

Displaying all of the 70000 instances available in the dataset results in too much visual clutter, so we have opted to only plot 200 randomly selected instances for each digit, as we have described in section 5.3.1. For the scatterplot to be representative of the whole dataset, we have decided to not include all of the misclassifications, to not skew the perceived distribution of predictions in the scatterplots.

The scatterplot was created using the Tkinter built-in canvas widget, where we started by centering the origin in the center of the visible portion of the canvas (by default it is in the top-left corner). Then we found out the maximum absolute value $m$ in any axis of the 2D projected data, to create the scale multiplier $s$ as

$$s = \frac{\min(\frac{\text{width}}{2}, \frac{\text{height}}{2}) - 2 \times \text{padding}}{m}$$

so the data would fit onto a canvas of dimensions width $\times$ height with origin in the center and selected padding. All of the projected data was then multiplied by this scale factor $s$ to project it into the canvas space, and lastly we had to reproject the $y$-axis data because Tkinter has the y-direction increasing in the downwards direction by using $\forall y \in Y : y := \max(Y) + \min(Y) - y$, where $Y$ is the set of all $y$-axis coordinates.

Each instance is then drawn as a single circle, with the instances belonging to the same real digit class having the same fill color. Clicking on a circle selects the instance for display in the main window of the app, as well as shows its information in the main window. Doubleclicking a circle adds the particular instance into layer activation windows.

As can be seen in the app and also partly at fig. 12 separation of digits is improving with each layer, while after convolutional layers, instances of different digits are very much mixed, after first fully connected layer, we can already see well separated groups (such as for digits 0, 6, 2, 5 or 7), even though we can also see that certain groups are more difficult to distinguish, such as 3 and 9. These results could have been more visible, had we used t-Distributed Stochastic Neighbor Embedding (t-SNE) as a dimensionality reduction technique.

### 5.3.3 Layer Activations window

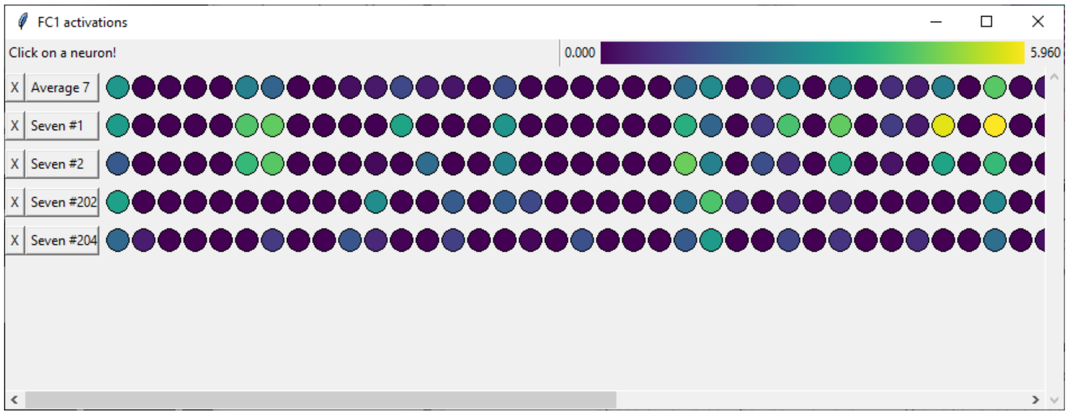


Figure 13: In the window, we can see which neurons are active in the first fully-connected layer for classification of digit seven. We can see activations for an average seven as well as activations for two correctly classified and two incorrectly classified images of seven. In the picture, we can clearly see which neurons the network expects to be active for digit seven and how much they actually are activated for correctly and incorrectly classified instances.

This window facilitates visualization of neuron activation for particular instances or grouped subsets of instances as described in section 3.2. Note that the application does not provide functionality for the user to create their own subsets of instances, so the only grouped activation visualization is available for whole digit classes (average activations for digit 0, digit 1, etc.).

We enable visualization of neuron activations only for fully connected layers, as our selected visualization does not scale well for a huge amount of neurons, but enables good inspection and comparison for examining a layer's behaviour on different instances side by side, which is why we selected this technique.

We open these windows from the main menu, where each layer has its own button.

When an instance or instance group is selected anywhere in the application by double clicking, it gets added to layer activation windows. Each instance is visualized as a line of circles, where each circle represents a neuron in the layer corresponding to the current window, and the color corresponds to the activation of that particular neuron.

For the color of the neurons, we have selected the Viridis color scheme, because of its perceptual uniformity, and because the activation function mostly used for fully connected layers is the ReLU activation function, which is linear for all positive inputs and 0 for all negative inputs, so a perceptually uniform colormap is appropriate. The Viridis colormap is appropriate even for the output layer, which uses a nonlinear Softmax activation function, but which outputs a probability distribution, and differences in probabilities can be well-interpreted by a perceptually uniform colormap.

The user can quickly glance at the strength of activations of neurons in the layer by their color, and comparing it to the colorscheme legend in the top right corner, which shows the colormap gradient and the corresponding min and max values at the appropriate sides.

Each instance is visualized in a single line. The first element in each line is an 'X' button, which removes the given instance from layer activations, the next element is the instance name, which can be clicked to show the input image corresponding to the instance in the main window, and then there is the neuron activation visualization on the line.

As each instance is visualized as a line of neurons, by adding multiple instances after each other into the visualization, we can compare how do these activations differ (or agree) for different instances, and we might for example understand why some instance of digit 7 is classified as a 1 by the network by adding that misclassified 7 into the layer activation window and adding some instance (or instances) of digit 1, or even the averaged activations for digit 1, and we might see that this particular instance of digit 7 fires similar neurons as digits 1.

By allowing having multiple windows of layer activations open at the same time, a user can inspect the instance's activation through the layers at the same time, to help gain an understanding of why the network made a particular classification.

# 6 Conclusion

The goal of this project was to visualize neural network which recognizes handwritten digits. Throughout the project, we initially explored neural network visualization options, later we implemented the network and finally designed and implemented visualizations of the network.

During our analysis stage we initially created an overview of main goals of neural network visualizations and set goal for our visualization. Then, we found and described visualization methods which could be used to achieve the goal for different types of neural networks and finally we presented a few existing tools which use described methods.

In the implementation stage, we explored architectural options for our neural network and finally decided to use both convolutional and fully connected layers. This led to the next design decision, since we had to determine which of the explored visualizations would be most suitable for our needs and should be included in our project. We had a good understanding of our options thanks to prior theoretical preparation and we were also aware that we could not implement all previously mentioned visualization techniques, since it would not be feasible to implement the solution in sufficient quality and in time.

Eventually, we decided to visualize activation of neurons for fully connected layers and output of all layers reduced to 2D and we designed and implemented mapping for these visualisation.

Using these techniques, a user is able to see browse instances, especially misclassifications, see them in projected spaces to judge how the network believes they are similar to nearby instances, and even explore how they activate specific neurons in the network and compare their activation pattern to other instances.

## References

[1] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[2] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau, “Visual analytics in deep learning: An interrogative survey for the next frontiers,” *IEEE transactions on visualization and computer graphics*, vol. 25, no. 8, pp. 2674–2693, 2018.

[3] J. Choo and S. Liu, “Visual analytics for explainable deep learning,” *IEEE computer graphics and applications*, vol. 38, no. 4, pp. 84–92, 2018.

[4] Taylor Telford, The Washington Post, “Apple card algorithm sparks gender bias allegations against goldman sachs,” 2019, [Accessed: 12-April-2020]. [Online]. Available: https://www.washingtonpost.com/business/2019/11/11/apple-card-algorithm-sparks-gender-bias-allegations-against-goldman-sachs/

[5] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. P. Chau, “Activis: Visual exploration of industry-scale deep neural network models,” *IEEE transactions on visualization and computer graphics*, vol. 24, no. 1, pp. 88–97, 2017.

[6] M. Mommert, “Mnist neural network visualization,” 2019. [Online]. Available: https://www.kaggle.com/mommermi/mnist-neural-network-visualization

[7] C. Olah, A. Mordvintsev, and L. Schubert, “Feature visualization,” *Distill*, 2017. [Online]. Available: https://distill.pub/2017/feature-visualization

[8] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization,” in *Deep Learning Workshop, International Conference on Machine Learning (ICML)*, 2015.