# ConvLens: Visualizing Inner Components of Convolutional Neural Networks

Mahdi Ghodsi and Hooman Shariati

**Abstract** Convolutional Neural Networks (CNNs) are one of the most popular approaches for object recognition problems. Despite their increasing popularity, they are widely known as black boxes. This is mainly due to their complex architecture and the high number of none-linear parameters within a CNN. Recently, a lot of attention has been given to exposing these black boxes by visualizing their learnable parameters. This project proposes a solution that interactively visualizes the architecture of the network and allows the user to investigate the parameters of the network layer by layer. The convolutional layers of the network, which contain learnable parameters, are the main focus of this tool. We visualize these learnable parameters by taking advantage of multiple recently published methods. Along with the visualization tool, this project provides software to automatically replicate some of the existing methods for visualizing the learnable parameters and outputs the results in a compatible format for ease of use in our visualization tool. **Our tool is, currently, available for use at http://35.163.48.45:9374/.**

***Index Terms*—Visual Analytics, Deep Learning, Convolutional Neural Networks, Information Visualization, Machine Learning**

## I. Introduction

Recently, Deep Neural Networks (DNNs) have gained much attention due to their success in tasks such as image classification and speech recognition. Among different DNN approaches, Convolutional Neural Networks (CNNs) are extremely popular, in particular, due to their outstanding capacity to utilize spatial information. Currently, there has been a shift in applying bioinformatics data to CNNs [1]. While much success has been achieved in the biomedical-imaging domain [2] using CNNs, there is ongoing research in other bioinformatics domains such as genomic sequence, [3] motifs, and EEG [4] as well.

Despite the encouraging success of CNNs, many still see CNNs as promising black boxes with little insight into the behaviour of their internal components. This fact leaves many researchers relying on trial and error to achieve better performance and fine-tune the parameters involved. Large modern neural networks are even harder to study because of their size. For example, understanding the widely used AlexNet DNN [5] involves making sense of the values taken by the 60 million trained network parameters [6, 7].

Consequently, the goal of this project is to create a visualization tool that allows domain experts to gain a better understanding of the inner components of any given CNN for their given input.

In order to achieve this goal, one naive approach is to feed forward the input up to the desired component inside a CNN, and then visualize what that component is outputting. Thankfully, since the input of a CNN is an image, for the components of the network that keeps the spatial information of the input, it is possible to produce an output that is understandable by humans. This output is taken and converted to a gray scale image. Then we can gain some insight by a simple side-by-side comparison to the original input.

However, as described by Mathew Zeiler and Rob Fergus [7], this naive approach fails to produce easy to understand images for the learnable parameters located deep inside the network. Therefore, they propose a more sophisticated approach, in which visualizing these parameters is possible. This is done by building a Deconvolutional network, which takes the raw result of the naive approach (described above) and projects the result of each component back to the input pixel space. In this project, an improved version of Deconvnet called Guided Back Propagation (GBP) [8] is used as well as the naive approach to visualize the learnable parameters contained in CNNs.

Consequently, we make the following contributions in this project:

1. Provide a clutter-free visualization of the over-all structure of the CNN, which allows users to interactively investigate different inner components of the network for a given input.
2. Allow for side-by-side comparison of multiple techniques for visualizing the learnable parameters inside CNNs.
3. Provide a set of data generation script, which can convert any CNN structure to the format that is consumable by our tool.
4. Provide a web-application deployment so that researchers can easily use our tool with their own CNNs and training/testing datasets.

The rest of this report is broken up into the following sections: Related Works, Domain Background, Data and Task Abstractions, Solution, Interfaces, Implementation,

Results, Discussion and Future Works, Conclusions, and Bibliography.

## II. Related Work

Visualizing learnable parameters is the most common approach to better understanding CNNs [9]. In an initial effort, Erhan and Bengio [10] found that visualizing learned features based on the concept of activation maximization could be an effective approach. This concept is based on looking for an input image from a large data set of images or a patch from one input image that maximizes the activation of a given neuron in a layer. This idea is further advanced by multiple research groups leading to three branches: Input Modification, Deconvolution, and Input Reconstruction methods [9].

Input modification methods are based on modifying the input image and monitoring the changes in the activation of a given neuron. Changes in the value of activation help identify parts of the image that are the most important for a given neuron [10, 11]. Zeiler and Fergus [7] achieved this by creating a gray square to occlude one part of the image at a time. By sliding this occlusion square they obtain a heat-map that is identical in size to the input pixel space that encodes the values of activations of a given neuron.

Input reconstruction methods break down the image into smaller patches and reconstruct it by either re-ordering the patches or replacing the patches of the original image with other images [12, 13]. The goal of these methods is to create new images that maximize the activation of a given neuron. Observing reconstructed images helps users gain better insight about learned features by a neuron. These methods have recently gained much attention by several research groups that propose modification to the original approach with the goal of keeping reconstructed images human interpretable [6].

Deconvolutional networks (described in Introduction), also widely known as Transposed Convolutional networks, refer to a multi-layered Deconvolutional network to project the feature activations back onto the input pixel space for a trained network. This technique reveals the input stimuli that excite individual feature maps at any layer in the model [14]. This method has been further improved to accommodate more models and produce easily interpreted images. GBP is a state of the art methodology within the family of Deconvnets [8]. We implement this method in our tool.

Up to this point, we have discussed the state of the art methods of visualizing learnable parameters within convolutional layers of CNNs. While feature visualization remains the most common approach in visualizing CNNs, less attention has been given to visualizing the overall structure of CNNs and the relationship between neurons within the network. Google's Tensorflow (an open source software library for machine learning) is complemented by a visualization tool, called Tensorboard, which visualizes the overall structure and relationships within the network [14].

CNNVis [15], ReVACNN [16], and a 3D tool proposed by Harley [17] are some other visualization tools that are capable of visualizing the structure of the networks, the relationship between the neurons, and the learnable parameters in each layer. Despite the valuable insights that these tools provide, each of them suffer from at least one major weakness that discourages users from taking advantage of their strengths. It is also worth mentioning that none of these tools are capable of including more than one of the methods discussed in this section for visualizing learnable parameters.

A major weakness of both Tensorflow and ReVACNN is the presence of visual clutter due to the use of node-link diagrams to show relationships between all of the neurons. CNNVis is not scalable for large networks since the images produced for visualizing learnable parameters are too small to view. Finally, Harley's tool is a 3D visualization of the network. This tool suffers from occlusion due to the nature of 3D visualizations. Rotating the view (to see the structure from different angles) results in occlusion of inner layers by outer layers. Furthermore, interacting with the 3D objects, in a lot of cases, leads to the user losing context of where they are within the overall structure of the network.

Since one of the goals of this project (as described in our Data and Task Abstraction section) is to visualize how the CNN reacts to a given input, Deconvolutional method is the only technique that satisfies our requirements. In this project, after careful review of these methods and existing tools, we decided to use GBP and Filter Map activations. The rationale behind these choices is explained in more details in the Solution section of this report.

## III. Domain Background

This section aims to deliver a brief summary of the architecture of CNNs and the types of data that CNNs contain. The contents described in this section are the summary of an online course offered by Stanford University, CS231n: Convolutional Neural Networks for Visual Recognition [18].

There are several main operations that take place within a CNN, all in a specific order. In a modular design of CNNs, each of these operations lies within a module, which is referred to as a **layer**. However, in the literature, there is an inconsistency in the grouping of different operations into layers. Some publications combine several operations in a single layer, while others dedicate a distinct layer to each operation. In the context of this project, in order to avoid any confusion, we will dedicate a distinct layer for each operation.

Each layer has a set of parameters. Some of the layers have fixed parameters while others have learnable ones. The following section briefly describes the common layers in CNNs.

### *A. Convolutional*

This layer is responsible for convolutional operations, which are performed using grid-like windows of weights that slide across the input image. This window is commonly referred to as a convolutional **filter**. The filter slides across

the image with a predefined step size. This step size is referred to as **stride**. The output of this operation is the result of the dot product between the entries of the filter and the pixel values of the input image. The result of this operation is called the **feature map or activations** of a filter.

CNN designers have the freedom to choose the number of filters in each layer, the window size, the stride, and the **padding** size, which as the name suggests refers to the operation of adding zero padding around the input image. Fig. 1 [19] illustrates the operation of convolution on an input image of size 5x5, by a filter of size of 3x3, stride size of 1 and padding size of 0. The green area illustrates the image. The yellow area is the filter window containing the small red values indicating the filter weights. Finally, the pink window, labeled as "convolved feature", is the result of the convolution, which is basically the feature map or activation of the filter. It is, also, worth noting that all the weights in each filter are learned during the training process.

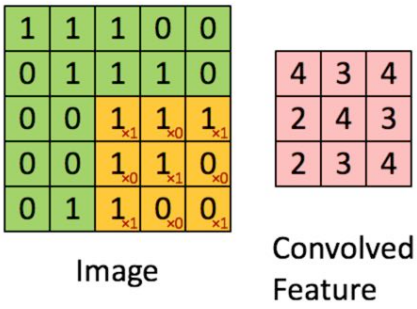


Fig. 1. Illustration of convolution[1]

### B. Non Linearity

This layer is also commonly referred to as the activation function. The most popular activation function used for CNNs is the rectified linear unit (**ReLU**); an element-wise activation function that sets the negative values to 0 and retains the positive values.

$$f(x) \quad \max(0, x) \qquad (1)$$

This layer comes after the convolutional layer and is applied to every element/weight in each window. There is no learnable parameter in this layer.

### C. Pooling

Pooling layer reduces the dimension of the input. The most popular pooling is maximum pooling. In this operation a fixed window size is defined, and then an empty window of this size is slid over the input to cover the entire pixel space. For each stride in this process, the maximum value within the window is extracted. Fig. 2 illustrates the process of max pooling for an input of window size 4x4, pooling window of 2x2, and stride of 2. The colour-coding of red, green, yellow, and blue indicate when the pooling window is placed on top of the input. For each coloured window, the maximum value is picked. The result of this operation down samples the original input from window size of 4x4 down to window size of 2x2. Finally, the CNN designer chooses the pooling window size, and the stride. There are no learnable parameters in this layer.

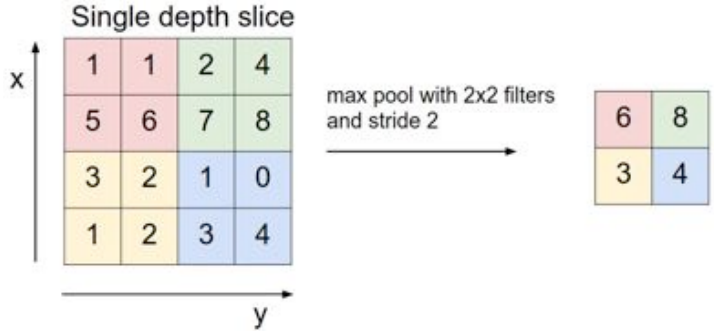


Fig. 2. Illustration of Max Pooling[2]

### D. Fully Connected

This layer acts just like traditional neural networks. Within each layer, there exists a set of blocks. These blocks are commonly referred to as **neurons**. Each neuron contains a vector of size equal to the number of inputs. The output of each neuron is the sum of dot product between the input values and the entries of the vector. The CNN designer chooses the number of neurons in each fully connected layer. In addition, all the weights in each neuron are learnable.

### E. Normalization

This layer, as the name suggests, normalizes the input. This operation happens for a given window size. The designer can choose to have the operation happen in one selected window or across multiple windows. In this operation all the elements in a given window are replaced with a normalized value, obtained using the elements in the normalization window.

### F. Dropout

This is a very simple but effective layer in the architecture. The role of this layer is to avoid memorizing (also known as over fitting) the examples during the training process. This layer only takes a probability value. According to the probability value, it sets the value of the inputs to zero. The CNN designer sets this value.

### G. Softmax and Classification

Softmax and Classification layers are usually the last two layers in the architecture. These two layers work together

[1] Retrieved from Deep Learning Tutorial available at http://ufldl.stanford.edu/wiki/images/6/6c/Convolution_schematic.gif.

[2] This image was retrieved from an online course, *Convolutional Neural Networks*, Stanford University. Retrieved From http://cs231n.github.io/convolutional-networks/

to produce the final output for a network that approximates a classification problem. The Softmax layer is responsible for assigning the probability of input x belonging to class i. The Classification layer picks the highest probability as the final output and attaches the class name to it. In the pre-trained implementation of AlexNet for Matlab these two layers are named "**Prob**" and "**Out**" respectively.

## IV. DATA AND TASK ABSTRACTIONS

### *A. Data: Domain-Specific*

The data that we need to visualize in this project is a Hierarchical Network, where each layer corresponds to a specific operation performed on input images that is supplied from the previous layer. There are 8 layer types in our data. Every layer has a specific type, input size, an output size; and depending on its type, anywhere from zero to nine other quantitative and sequential attributes.

In addition, as described in the previous section, Convolutional and Fully Connected layers contain learnable parameters that are important for feature visualization. These, quantitative and diverging parameters are real numbers stored as weights in a network of nodes and edges.

In order to visualize the activation of the filters inside Convolutional layers, we derive additional data from these weights using two different techniques, both producing gray scale images. The second method we use is Forward Activation [9], which keeps the weights as diverging (with both positive and negative values) when producing gray-scale images. The second method that we use, is GBP [8], which sets all the negative weights to 0 (converts our weight attribute from diverging to sequential) when producing gray-scale images.

### *B. Data: Domain-Independent*

In domain-independent language, our data can be abstracted as a Hierarchical Network of nodes and edges, where each layer in the hierarchy of the network has:

- 1 categorical attribute
- 0 to 9 quantitative and sequential attributes
- 0 to thousands of quantitative and diverging attributes
- **Derived data**: 0 to thousands of grey-scale images

### *C. Tasks: Domain-Specific*

In order to obtain a list of domain tasks for this project, two main steps were taken. First, we interviewed Dr. Ali Bashashati, who agreed to meet with us on regular basis to collaborate as the domain expert in this project. The domain-specific tasks that we identified during our interviews are as follows:

- Understand the high-level architecture of the network. Specifically, understand the type and order of different layers.
- Understand the activation of each filter inside Convolutional layers.
- Understand the over-all performance of a Convolutional layer based on the quality of the activations of its filters.

In a second step, we compared above tasks with the ones from a study by Liu that surveyed researchers and identified the following requirements for a CNN specific visualization tool [9]:

1. Providing an overview of the learned features of filters in convolutional layers and neurons in fully connected layers.
2. Interactively modifying the neuron/filter clustering results.
3. Exploring multiple facets of neuron/filter.
4. Revealing how low-level features are aggregated into high-level features.
5. Examining the debugging information.

Note that Lui's requirements are obtained for a visualization tool that visualizes the network during training with access to all the inputs from the training set. This is considerably different for this project, as this project is aiming to visualize a pre-trained network's response to a given input.

Lui's access to training information allows him to cluster neurons/filters. Also, training information allows him to encode debugging information in his visualization tool. Hence, requirements 2 and 5 from his list are not applicable for this project as this project's goal is to visualize pre-trained networks with no access to their training information. Moreover, there is no method for visualizing neurons in fully connected layers for a given input in a meaningful way. This is due to the nature of fully connected layers, in which spatial information of the input images are lost. Therefore, in this project visualizing convolutional layers remain as the only appropriate layers to visualize for a given input.

Finally, after comparing the relevant requirements from Lui's paper and combining them with our domain tasks, we added the following domain tasks to our list:

- Identify the effect of applying different inputs to the system on the activation of the filters inside Convolutional layers.
- Compare different techniques for visualizing the activations of the filters inside Convolutional layers to find the best technique for the specific dataset under study.

### *D. Tasks: Domain-Independent*

Following the 9-stage design study methodology framework [16], we summarized our domain-specific tasks

into the following abstract tasks:

**Explore → Summarize**
- Overall Architecture of ConvNet
- Activations of the filters in each Convolutional layer

**Explore → Compare**
- Different techniques for visualizing the activations of the filters inside Convolutional layers

**Locate → Identify**
- Filters that have learned useful features
- Filters that are useless

## V. Solution

Our final solution was designed to achieve the tasks that were described in the previous section. Our over-arching goal is to help researchers tune the structure and parameters of Convolutional Neural Networks to improve their performance. At a lower level, our goal is to help researchers understand the high-level structure and activations of the filters inside Convolutional layers of the Convolutional Neural Networks they are researching.

Consequently, we spent a lot of time on making sure that our tool works with all Convolutional Neural Networks. In addition, we provide our tool in a web-application, which allows researchers to upload their own network structure and filter activation to our server and immediately begin using our tool on their own network and training data.

To avoid visual clutter in our design, we do not visualize the connections between layers, or the nodes and edges of the network inside Convolutional and Fully Connected layers. Discussion with our domain experts determined that information about the connection between layers or inside Convolutional and Fully Connected layers is not really important in tuning network parameters to optimize over-all performance. The following section will describe the technique we use for visualizing the activations of the filters inside Convolutional layers.

### *A. Visualizing Learnable Parameters*

We have decided to include two techno feature visualization methods. The most common way to do this is by visualizing the activations of each filter in convolutional layers. This method was our design choice as a naive way of visualizing learned parameters in the network. This choice was made for two main reasons. First, this method is very intuitive to understand, and relative easy to produce. In addition, it requires significantly less computation compared to more sophisticated methods. For instance, running this method for all of the convolutional layers in AlexNet, on a 2.4 GHz Intel Core i5, takes only 28 seconds.

The other method we selected was GBP [8]. This method belongs to the family Deconvolutional networks, which was introduced for visualizing activations of filters in deeper layers. Among this family, a few different methods have been introduced in the past a few years. After reviewing original Deconvnet [7], back propagation [12], and GBP, We chose GBP as we observed this method to produce slightly more clear results. Fig. 3 compares the result of our comparison for the activation map of filter 20 in 5$^{th}$ convolution layer of AlexNet. All three images were retrieved from the same filter and all other variables remained constant. As illustrated in this figure, GBP gives the best result.

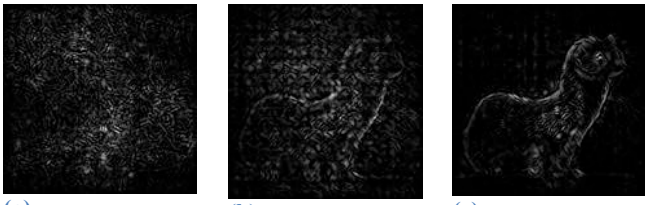


Fig. 3. Illustration of our comparative analysis on the family of Deconvnet visualization method: (a) Back Propogation; (b) Deconvnet; (c) Guided Back Propagation

Our findings matched the claims for two different papers [9]. Also, computationally, all three methods are comparable. For us, running this method for all of the convolutional layers in Alexnet, on a 2.4 GHz Intel Core i5, takes only 920 seconds.

## VI. Interfaces

The visualization tool was carefully designed to meet all the requirements of this project. This tool is deployed as website application that users can start using after uploading the necessary files mentioned above, or by using our default files to launch a demo. **The tool can be accessed at http://35.163.48.45:9374/**. Fig. 4 illustrates an overview of our visualization tool for AlexNet and an input image of a weasel, retrieved from Imagenet dataset [21].

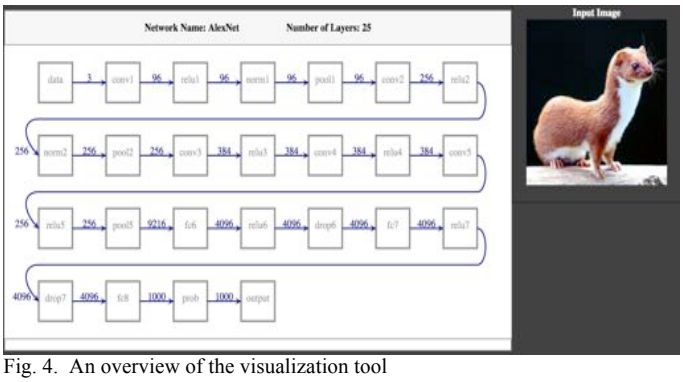


Fig. 4. An overview of the visualization tool

The visualization tool is split into 3 main views. Initially, the system was designed in a single view as opposed to multiple views showing the top level of hierarchy where the architecture was the only visible view. The system was designed based on the details on demand idiom, where the user could see the next level of hierarchy by clicking on the desired layer from the architecture diagram. The problem with the original design was its heavy reliance on user's cognitive abilities to remember the architecture, the selected layer, and the rank of selected layer in the architecture. Also

for selecting another layer, the user had to go back to the original view, and then select another layer to investigate. This extra effort required for viewing another layer caused the users to lose context in terms of where they are in the overall network.

This design was then replaced with a multiple view design, where each view is designed for a specific purpose. The following section describes the three views that we have in our final design.

### *1) Architecture Canvas*

This canvas contains the general information about the network. Fig. 5 illustrates this canvas in detail for AlexNet. On the very top the name and the number of layers are encoded as text fields. Underneath, the overview of the architecture of the network is displayed. The architecture is encoded as a chain of node-link diagram. Each node represents a layer. The type of the layer is encoded inside each node as plain text. The link between layer A and B is encoded as a directional line mark. This encodes the output of layer A is fed to layer B. The number of outputs is encoded as numerical text values above each link.

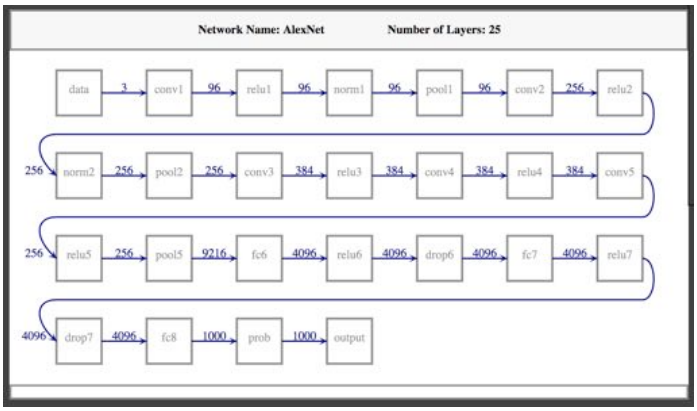


Fig. 5. Illustration of the Architecture Canvas

Also, as mentioned earlier in the report, one design choice for this canvas that we made was about whether to aggregate any of the layers in one node. Some of the existing CNN visualization tools aggregate some of the layers between two convolutional layers. We believe that this might be misleading to a novice user. For instance, aggregating convolutional and ReLU layers in one node might lead to a user thinking that ReLU is not a distinct layer.

The last encoding for this canvas lies inside the nodes representing convolutional layers. These nodes contain a stacked bar summarizing the result of the annotation. This bar is placed at bottom of these nodes. Annotation is one of the features described in the interaction section. The stacked rectangles in the stacked bar are color coded in three different colors. Blue section indicates Good, red indicates Bad, and gray is used for filters that do not have any annotations.

### *2) Details Canvas*

This canvas contains the detail on demand for selected layers in the Architecture Canvas. The selected layer determines the details displayed in this canvas. Therefore, the encoding differs based on the layer type.

Fig. 6 illustrates the Details canvas for the second convolutional layer of AlexNet. In this view, switching between two different activation visualization methods and the parameters is possible by selecting the corresponding radio button. The activations are visualized as a grid of images.

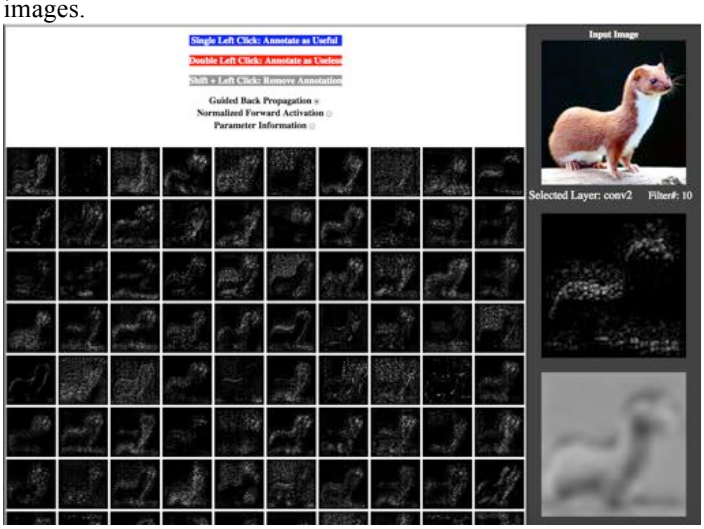


Fig. 6. Illustration of Details Canvas for a Convolutional Layer

For probability layer, the classification results are visualized in a sorted bar chart. The bar chart filters the values shown to only the ones that are larger than a fixed threshold. The threshold is set to be 0.1. In the bar chart, the y-axis encodes the probability in percentage from 0 - 100. The x-axis illustrates the classes that achieved a score larger than the fixed threshold of 0.1. Fig. 7 illustrates the Details canvas for the "Prob" layer of AlexNet for the input image illustrated in Fig. 6. In this case "weasel" has the highest probability and is located on the left hand side of the x-axis.

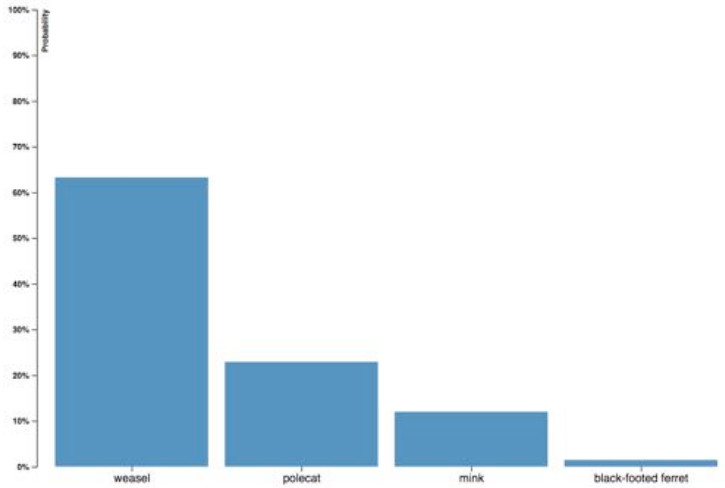


Fig. 7. The Details Canvas for a Probability layer

For output layer, we show the confidence level of the network in classifying the input as one of the classes in a bar chart. Additionally, the top image on Flickr with the tag matching the name of the output class is shown. We believe this would be useful as the user might not be familiar with the name of the output class. For instance, if the network takes an image of a German shepherd and misclassifies the

input image of to be an image of Bedlington terrier, this feature will help the user to compare the similarities between the two breeds. Fig. 8 illustrates the Details canvas for the output layer of AlexNet. This figure shows the result for an input image of a weasel.

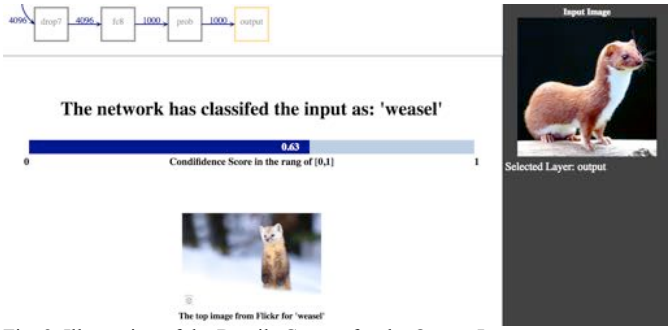


Fig. 8. Illustration of the Details Canvas for the Output Layer

The pooling layer parameters are encoded to visually represent the window size of the parameters. All the parameters in this layer could be encoded as a grid of squares. We believe showing a 3x3 grid of 9 squares is easier to grasp for the user rather than just showing the numerical values of this layer as plain text. Finally, all the other layers contain a set of parameters and they are encoded as plain text.

### *3) Side Canvas*

This canvas contains the input image on the very top. The space below this image is reserved for labels and activation images to be played when the user interacts with the system. This canvas is designed to be on the right side of the screen at all times. It illustrates the selected layer and the selected texts. It is also locked to the same fixed location even during scrolling. We believe this is necessary because when the user is scrolling through the Details canvas, the user needs to see the input image for comparison. Allowing this canvas to scroll will push the input image off the screen if the user scrolls down. Fig. 9 illustrates an example of the Side canvas for an input. This screen shot was taken while the user was hovering over the activation of filter 20$^{th}$ of the 5$^{th}$ convolutional layer in AlexNet. This will become clearer when we discuss in the next section where we discuss the possible user interactions in our system.

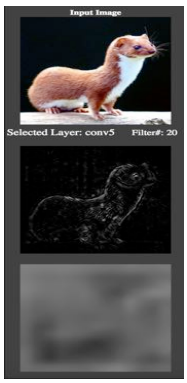


Fig. 9. Illustration of the Side Canvas

## *B. User Interactions*

In this section, the possible interactions of the user with our system are described. There are some obvious interactions such as highlight on hover for layers, or select and change border on click for all the nodes representing the layers in the main canvas. Also, upon clicking each layer, the name of the selected layer is added to the side canvas underneath the input image. However, we believe there are some other annotations that need to be explained further. The following sections categorize and describe the ways users can interact with our system in detail.

### *1) Annotation*

In Convolutional layers, we visualize the activations of each filter for the input. These activations might indicate that a filter is not extracting beneficial features, or possibly extracts no features at all. Also, keep in mind that the user has the freedom in choosing the number of filters for a given convolutional layer. Since our tool could be used for improving the performance, a user might want to annotate the filters as useful/good, useless/bad, or leave them un-annotated. For this reason, we have included this feature in our design. Fig. 10 illustrates an example of this feature in action. In this case the user has already annotated 42 filters.

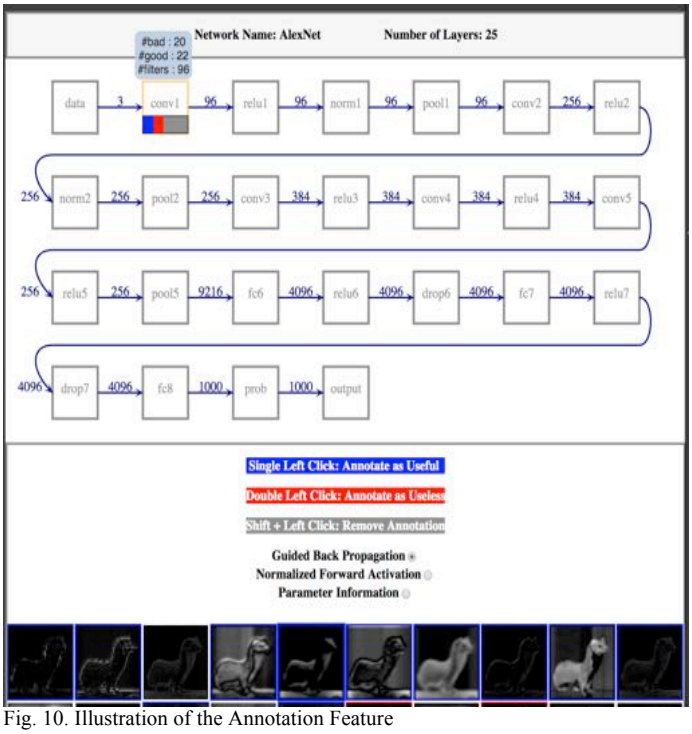


Fig. 10. Illustration of the Annotation Feature

The user can annotate a filter to be good by a single left click, bad by double click, and undo the annotation by holding the shift key before a single left click. The summary of these annotations is contained inside the corresponding node representing the layer.

### *2) 6.3.2 Details on Hover*

This feature allows the user to get extra detail about a specific visible element on mouse over. We have added tooltips for this purpose for the stacked bars summarizing

the annotations in convolutional layers and for the bar chart representing the probabilities in the Prob layer.

#### *3) 6.3.3 Zoom on Hover*

This feature is available for all of the filters inside convolutional layers. The user can hover over an image representing the activation of a filter and see the large version of the image displayed on the right hand side, inside the Side canvas, juxtaposed under the input image for comparison. This allows easy comparison to the input image. Additionally, the label representing the identification number of the filter of interest is indicated above the enlarged version of the image. In Fig. 6, an illustration of this interaction is shown.

## VII. IMPLEMENTATION

This project is Mahdi's MEng main project. Therefore, he was in control of this project and prepared all the data required for the visualization tool. That includes selecting the method for deriving the data and comparing and contrasting the available methods for visualizing activations.

That being said, both authors contributed equally in coding the visualization tool.

This section is divided into two portions. The first one (Data Generation) describes the tools we are providing our users for modifying their CNNs to make them usable in our system. The second section (Data Visualization Tool) describes the actual implementation of our tool (ConvLens).

### *A. Data Generation*

In order to meet the requirements of this project, our visualization tool needs to be dynamic. This implies that our tool should be able to visualize any CNN that is fed into it. Therefore, careful considerations were taken into account for portability and generalizability of the data generation phase. We will explain this in more detail in the next section (Data Visualization Tool).

Four data files are required to take advantage of the full capability of our visualization tool: a JSON file containing Architecture Information, a folder containing images of feed forward activations, a folder containing images of back propagation, and finally a JSON file of the output of the network for a given input.

It is worth mentioning that in order for our tool to visualize the overall architecture and the numerical parameters of the network only the first file is required. However, for our tool to visualize the response of the network to a given input at least one of the other three files is required. In the following section our method for producing the data and what they are is described in detail.

#### *1) Architecture Information*

This file contains general information about the architecture of the visualized CNN. It needs to be saved in JSON text format and with the specific structure that is compatible with the visualization tool. For ease of use we have provided a sample version of this file. The user may modify the sample and manually or produce the file automatically.

For this project, we have automated this process using Matlab and JavaScript. In order to achieve this we had to modify the Matlab's deep learning library's source code. we have made the script available in this project's package. Instructions for how to produce this data are included in the software manual.

#### *2) Forward Activations*

As mentioned earlier, this method is a naive way of visualizing the result of applying each filter on the input in convolutional layers. For this project, we wanted to visualize this effect for every filter in every convolutional layer in the network. Fortunately, recent release of Matlab's deep learning library includes an "activation" function that returns the numerical values of any filter in any layer; after it is applied to the input. By manipulating this function, we can produce images that resemble the effect of each filter on the input.

These images are produce by following the steps mentioned below:

1. Obtain the values of applying the activation function for filter 1 in convolutional layer.
2. Normalize the result of 1, between [0,255] to produce a gray scale image. (This is needed because the filter weights are diverging values from $(-\infty, +\infty)$ and multiplying these values by the input (which is an image) values scales the diverging values of the filter. Therefore, we need to normalize the smallest value to be 0 and the largest value be 255 to produce a meaningful gray scale image. In Matlab, mat2grey function takes care of this normalization operation.)
3. Save image to file with the name of "LayerNameFilterNumber.format".
4. Repeat step 1 to 3 until we reach the last filter in the last convolutional layer.

We have written a script in Matlab that follows the above algorithm and produces all the images automatically. The script is included in the project package and the manual describes how to use the script.

#### *3) Guided Back Propagation*

Guided Back Propagation is the more sophisticated method of visualizing the result of applying each filter on the input in convolutional layers. Unfortunately, there are no native Matlab functions to support some of the required operations for this method. However, MatConvNet, a third party library for Matlab, released by Andrea Vedaldi from University of Oxford [22], makes this process easier to implement. Felix Grün has released an open source library, FeatVis that builds on top of MatConvNet. FeatVis supports Guided Back Propagation.

It is necessary to note that CNNs are defined in a different manner in MatConvNet compared to Matlab's native deep learning library. The user needs to convert the format of the CNN to the format compatible for MatConvNet following the guidelines and manuals

provided for this library.

For the purpose of this project, we used FeatVis to produce the images for Guided Back Propagation to visualize the features from the input image that activated a particular filter the most. We have a written a separate scripts to automatically produce all of the images for all the filters in the network. This script takes care of converting the input image to the correct format for FeatVis and MatConvNet. We have also modified the source code of FeatVis to make this process automated and faster. The instructions for using the script are included in the manual for this project.

#### 4) Output of Class Scores

This file is a simple JSON text file that contains an array "S" of scores. Each element "$s_i$" in the array with index "we" contains the scores of the input being classified to belong to class "$c_i$". The length of the array is equal to the number of possible classes of output.

This file is simply achieved by using Matlab's native deep learning library. This is done by using the built in function of "classify" and saving the result as a JSON text file. A sample of this file has been provided in the package.

### B. Data Visualization Tool

Our tool (ConvLens) is designed as a web application, implemented using HTML, CSS, D3.js, and jQuery. Our design contains the following main views.

#### 1) Architecture Canvas

This canvas was a designed using D3.js and raw JavaScript. All the code for this canvas is written from scratch. No other external libraries or code from other sources were used to implement this canvas

The width of this canvas remains constant but the height of the canvas is adjusted dynamically, depending on the number of layers in the architecture.

In the implementation, we found seven layers per row, and square side of length 60px to be reasonable values to choose. These values were chosen based on trial and error.

The stacked bars are contained within each node representing the layers. The stack bars are placed at bottom one third of the area of each convolutional layer. The stack bars become visible after the user starts annotating a layer. No external library or code from other sources was used to implement this canvas

#### 2) Details Canvas

This canvas displays the detail on demand based on the user's interaction with the tool. The selected layer determines the details displayed in this canvas. Therefore, the implementation of this canvas is different based on the type of the selected layer. Aside from ReLU, Normalization, Drop Out, and Fully Connected layers, where simple text is displayed, visualizing other layers required more development effort. In the following section we discuss the implementation details for each of our layer types.

##### a) Convolutional Layer Details

This layer reveals the effect of applying filters on the input. In addition, numerical values of all the parameters of these layers are displayed for this layer type. On top of the canvas, the radio buttons are implemented in HTML.

A grid of SVG images using D3 was implemented to show the images of post-filtered input. The height of canvas is adjusted based on the number of filters in each layer. The elements within the grid have constant width and height. The size of widths and heights are determined based on trial and error. The parameters are displayed as plain text in HTML.

##### b) Pooling Layer Details

The encoding in this layer was coded in D3 and draws basic grids to show window size of all three parameters of this layer.

##### c) Probability Layer Details

The sorted bar chart for this layer was implemented by modifying the given example by the author of D3.js [23]. A threshold of 0.1 was set to filter the shown data points in this chart.

##### d) Output Layer Details

This layer displays a bar chart of the confidence in classifying the output. This bar chart is coded in D3. Below the bar chart, the image is an SVG image that is queried from Flickr by modifying a code from Stack Overflow [24]

#### 3) Side Canvas

This canvas is another SVG canvas that illustrates the input image. The canvas and all the details illustrated in it are all coded using D3.

## VIII. Results

In this section, we discuss two scenarios that aim to demonstrate how our tool can be used to complete the domain tasks discussed earlier in this report.

### A. Exploring the input to the network and the effect of applying convolution throughout the system

Donald, a hobbyist, takes a picture from himself and wonders what if a well-known CNN, called AlexNet could classify him as a person. He, then, uses Matlab to feed AlexNet his picture. Surprisingly, AlexNet, classifies him as a neck brace. Donald knows that deep layers in the network extract high level features. He then wonders if he could explore every filter in the last convolutional layer to see what features from his picture excited the filters in that layer the most.

Donald runs our data generation Matlab script and feeds the results into ConvLens. He then looks for the last convolutional layer in AlexNet. By looking at the overview

of AlexNet's architecture visualized in ConvLens, Donald, quickly spots that "conv5" is the last convolutional layer in the network. He also observes from the shown architecture that this layer has an output value of 256. Donald knows that the number of filters in Convolutional layers is equal to the number of outputs; therefore, he expects to investigate 256 images in this layer. He then selects this layer. Fig. 5, illustrates this architecture.

The Details canvas loads all the 256 images produced using GBP method. From the overview, he spots that most of the images emphasize the bottom of the image. He picks a few of the images to investigate further. He first hovers on the output of filter 256. By musing over the desired filter, a larger version of the image is displayed under the input image, which allows Donald to compare the result of filter 10 to the original image. This process for a different input is shown in Fig 6. Finally, repeating this process for all the filters, allows Donald to confirm that about one third of the filters are extracting his neck area, which explains why his AlexNet classified him as a neck brace.

### *B. Optimizing the Number of Filters*

Yann, a biomedical engineering student, recently has learned about CNNs in his CPSC540 course. He implements a CNN that takes an image and classifies it is an image of a dog or a cat. In his implementation, Yann chooses to include 3 convolutional layers in his architecture. For the number of filters in each convolutional layer, he decides to include an arbitrary number of 256. He chooses this value by comparing his network to AlexNet. Unfortunately, Yann's initial attempts at classifying common objects result in poor classification accuracy. He then wishes to gain a better understanding of how these filters respond to his set of test images. Yann, knew his classmates in CPSC547 have created ConvLens, a tool that could help him with this task. Therefore, he decides to give ConvLens a try.

He sets up ConvLens to understand how the filters in the convolutional layers respond to a few test images. For a test image, Yann investigates the 3 convolutional layers in his network. He uses the annotation feature of ConvLens to annotate each filter as either useful or useless by visually examining the activations of each filter for the input. He marks the filters that produce noisy images, or pure black, as useless, by double-clicking on each image; and marks the rest as useful by a single click on the image.

Following this step, Yann records the overview of his annotation results by taking a look at the stacked bar charts contained inside the nodes that represent convolutional layers. Fig 10 illustrates the same task for a different CNN and different input. We have not implemented the functionality to export the annotations yet; so, currently Yann has to record the stack chart results himself. Subsequently, He re-runs this experiment for a few other test images and concludes that for all the test images, the 3rd convolutional layer contains over 80 useless filters. Yann then reduces the number of filters and in this layer by one third and verifies that this process has improved the classification accuracy of the network.

## IX. Discussion and Future Work

Even though we did not perform a formal user evaluation of our tool in time for this report, we did perform an informal evaluation by showing our tool to our domain experts and several none-expert individuals. Based on our observations during these demos, we believe that our solution helps users achieve the domain tasks described in this paper. Furthermore, after every single demo, participants showed interest in exploring our tool further and possibly using it in their own research. In several instances, we were asked whether ConvLens is open sourced, where the Github repository is, and whether we are going to keep hosting it at the current address or possibly on a new domain comparison.

In with the most common tools that are currently available for visualizing CNNs, ConvLens has three major strengths: it provides an architectural overview of the network without any visual clutter; it provides multiple methods for visualizing the learned parameters inside Convolutional layers; and, most importantly, it can be easily used with any CNN and any training/ test data.

We have decided to allow our users to upload the structural information of their CNNs to our server and we make them available to the public because these files are relatively small in size; pose less privacy concerns compared to the images visualizing the learnable parameters; and, most importantly, can be of great value to other researchers or hobbyists.

On the other hand, uploading the images that visualize the learnable parameters inside CNNs onto a server might be difficult due the sheer number of these images (even though their file sizes are usually small) and privacy concerns. Consequently, our tool only requires a link to the directory that contains these images on user's private computer (as a local host) or any other servers. Finally, it is important to point out that the implementation of our tool (ConvLens) allows it to visualize the learnable parameters using any technique that produces an image. In other words, nothing in our implementation limits the user to use only Forward Activation or GBP for creating images that visualize the filters inside Convolutional layers.

In terms of future works, we are hoping to have a more dynamic front-end in terms of screen aspect ratios. Currently, the user needs to adjust the zoom of the browser to be at 75% for the tool to fit entirely on the screen. Also, the overall appearance of the tool and the layout requires a fair amount of polishing.

On the backend side, we are working on providing the users with the functionality to export their annotations, and possibly perform quick statistical analysis on them.

On the visualizing side, it will be extremely useful to rank the filters (inside convolutional layers) for each input image based on the relative importance of the filter. There could be several measures to determine the importance of a filter, but, so far, we are planning to use the average weight of the activation map for each filter. Moreover, while we

couldn't find any appropriate methods for visualizing the Fully Connected layers for a given input, further investigation into this matter is could be fruitful. Finally, we are planning to integrate our Matlab and Java visualization scripts that are used for data generation into our tool, which would further streamline the process for researchers to use our tool with their own network and data.

## X. Conclusion

In this project, we designed and implemented an interactive tool that is capable of visualizing any convolutional neural networks, with any training/ test sets, and multiple methods for visualizing the learnable parameters. Our tool (ConvLens) is currently served on the web as a fully functional application. The main contributions of this project are divided into two categories of data visualization and data generation.

For the data generation phase, we have reviewed the state of the art method of visualizing the learnable parameters within convolutional layers of CNNs. We have selected two different methods and provided the software to replicate these two methods on other CNNs and datasets. In addition, we have provided the software to export CNNs architecture into a compatible JSON text file for our visualization tool.

For the data visualization phase, we created a standalone tool that could work with the data produced in any software. Our tool (ConvLens) visualizes an overview of the architecture, the activation map of convolutional filters for an input using multiple methods, the tunable parameters of all the layers, and the classification results of the network for the input image. Additionally, our tool allows the user to annotate the filters in convolutional layers to keep track of the layers that require fine-tuning of the number of filters.

## References


[1] S. Min, B. Lee and S. Yoon, "Deep learning in bioinformatics", *Briefings in Bioinformatics*, p. bbw068, 2016.

[2] H. R. Roth *et al.*, "DeepOrgan: Multi-level Deep Convolutional Networks for Automated Pancreas Segmentation," in *Medical Image Computing and Computer-Assisted Intervention -- MICCAI 2015*, 2015, pp. 556–564.

[3] J. Lanchantin, R. Singh, B. Wang, and Y. Qi, "Deep Motif Dashboard: Visualizing and Understanding Genomic Sequences Using Deep Neural Networks," *Pac Symp Biocomput*, vol. 22, pp. 254–265, 2016.

[4] P. Bashivan, WE. Rish, M. Yeasin, and N. Codella, "Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks," *arXiv:1511.06448 [cs]*, Nov. 2015.

[5] A. Krizhevsky, WE. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[6] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding Neural Networks Through Deep Visualization," *ArXiv e-prints*, vol. 1506, p. arXiv:1506.06579, Jun. 2015.

[7] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *arXiv:1311.2901 [cs]*, Nov. 2013.

[8] J. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," presented at the ICLR (workshop track), 2015.

[9] F. Grün, C. Rupprecht, N. Navab, and T. Federico, "A Taxonomy and Library for Visualizing Learned Features in Convolutional Neural Networks," presented at the International Conference on Machine Learning, New York, NY, USA, 2016, vol. 48.

[10] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing Higher-Layer Features of a Deep Network," Universite de Montreal, Dept. IRO, Technical Report 1341, 06 2009.

[11] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object Detectors Emerge in Deep Scene CNNs," *ArXiv e-prints*, vol. 1412, p. arXiv:1412.6856, Dec. 2014.

[12] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps," *ArXiv e-prints*, vol. 1312, p. arXiv:1312.6034, Dec. 2013.

[13] J. L. Long, N. Zhang, and T. Darrell, "Do Convnets Learn Correspondence?," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1601–1609.

[14] "TensorBoard: Visualizing Learning," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/get_started/summaries_and_tensorboard. [Accessed: 28-Apr-2017].

[15] S. Chung, S. Suh, C. Park, K. Kang, J. Choo, and B. C. Kwon, "ReVACNN: Real-Time Visual Analytics f or Convolutional Neural Network," presented at the Workshop on Interactive Data Exploration and Analytics, San Francisco, CA, USA., 2016.

[16] A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks," in *Advances in Visual Computing*, 2015, pp. 867–877. Springer International Publishing.

[17] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu, "Towards Better Analysis of Deep Convolutional Neural Networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 91–100, Jan. 2017.

[18] A. Karpathy and J. Johnson, "CS231n Convolutional Neural Networks for Visual Recognition." [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 28-Apr-2017].

[19] A. Ng, "Deep Learning Tutorial, Computer Science Department, Stanford University." [Online]. Available: http://ufldl.stanford.edu/. [Accessed: 28-Apr-2017].

[20] M. Sedlmair, M. Meyer, and T. Munzner, "Design Study Methodology: Reflections from the Trenches and the Stacks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2431–2440, Dec. 2012.

[21] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[22] A. Vedaldi and K. Lenc, "MatConvNet - Convolutional Neural Networks for MATLAB," *ArXiv e-prints*, vol. 1412, p. arXiv:1412.4564, Dec. 2014.

[23] M. Bostock, V. Ogievetsky, and J. Heer, "D3; Data-Driven Documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[24] "Stack Overflow, JavaScript - Random photos from Flickr JSON feed - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/16566260/random-photos-from-flickr-json-feed. [Accessed: 28-Apr-2017].