Exercise 4: Dashboard using two XAI techniques

Data Visualization with and for AI

Simon Vermeir

1 Requirements

The goal of this lab assignment is to learn how to make a dashboard in Dash that allows us to use XAI techniques. The dashboard has to have 2 main features: Show a 2D data overview of our data using dimensionality reduction and explain prediction of a machine learning model.

To enable this, a dataset of choice can be picked, and an accompanying model. The dimensionality reduction can in the first place be applied to the samples themselves. To learn even more, we should also push the samples through the model and capture the activation at different places in the model and apply dimensionality reduction on the activations. This way To learn more from our data.

2 Introduction

A natural way for me to achieve this assignment would have been to extend the features of my dashboard of exercise 4. I could have stayed with the MNIST dataset, picked a model and also show the dimensionality reduction for different layers of the model by way of a dropdown.

However, this didn't excite me. I wanted to create something bold, and new, and I wanted to challenge myself. There were three things that I wanted to do:

- Work with a big dataset that is relevant.
- Use a big, state-of-the-art model.
- Make the big model as accessible as possible to the end user by using highly visual and interactive elements.

The initial inspiration for my creation stemmed from a video that we have seen in class where researches had developed a dashboard where you could input pictures into a model and observe activations, and explanations for relevant layers. This dashboard operated on the live model an worked incredibly smooth.

I wanted to create something similar but from a different angle. The live GPU implementation of them did not seem relevant to me in this context, so these were not taken into account. Displaying the activation per layer, and the sense of overview you had was something I wanted to carry over. Combined with the requirements of this assignment, and my desires, I came up with the following requirements for my dashboard:

- As a model use VGG16 a big state-of-the-art model for ImageNet. That can identify 1000 classes efficiently.
- Use the ImageNet dataset.
  - Using the full dataset was after discovering this was 120GB impossible. After a lot of research, I found two suitable subsets that contained each 10 classes. This solved two problems: size and also the amount of labels to be displayed in the dimensionality reduction.
- Use different amount of samples, different type of data, different settings for the dimensionality reduction etc. Just like in assignment 3.
- Be able to visualize the model architecture, and interact with this visualization to select the layer you are interested in.

  - This was the hero feature for me. The model is big and contains 30+ layers. Since I really hated the idea of having a dropdown with 30 different options, I needed a solution. Humans are not capable of easily picking from such a long list, but we are pattern recognizing, so if we would display the model and be able to select the layer we wanted directly we could quickly select the layer and be able to constantly keep in mind where we are looking in the model.

- When hovering over samples in the dimensionality reduction, you can view relevant info:
  - The picture
  - The top 10 prediction
  - Explainability of the prediction.

- When clicking the sample be able to inspect the indivudal activations within the model at thay layer, just like the dashboard shown in class.

3 The product

In the end, the dashboard ImageNet data insights with VGG16 was born. The dashboard enables you to gain insights into ImageNet data by leveraging the power of a neural network trained for classification on the ImageNet dataset. By exploring representations beyond simple RGB images, the goal is to uncover new insights into ImageNet data. For example, discovering interesting properties inherent to the data, such as distinct clusters based on uniquely shared features between groups of pictures.

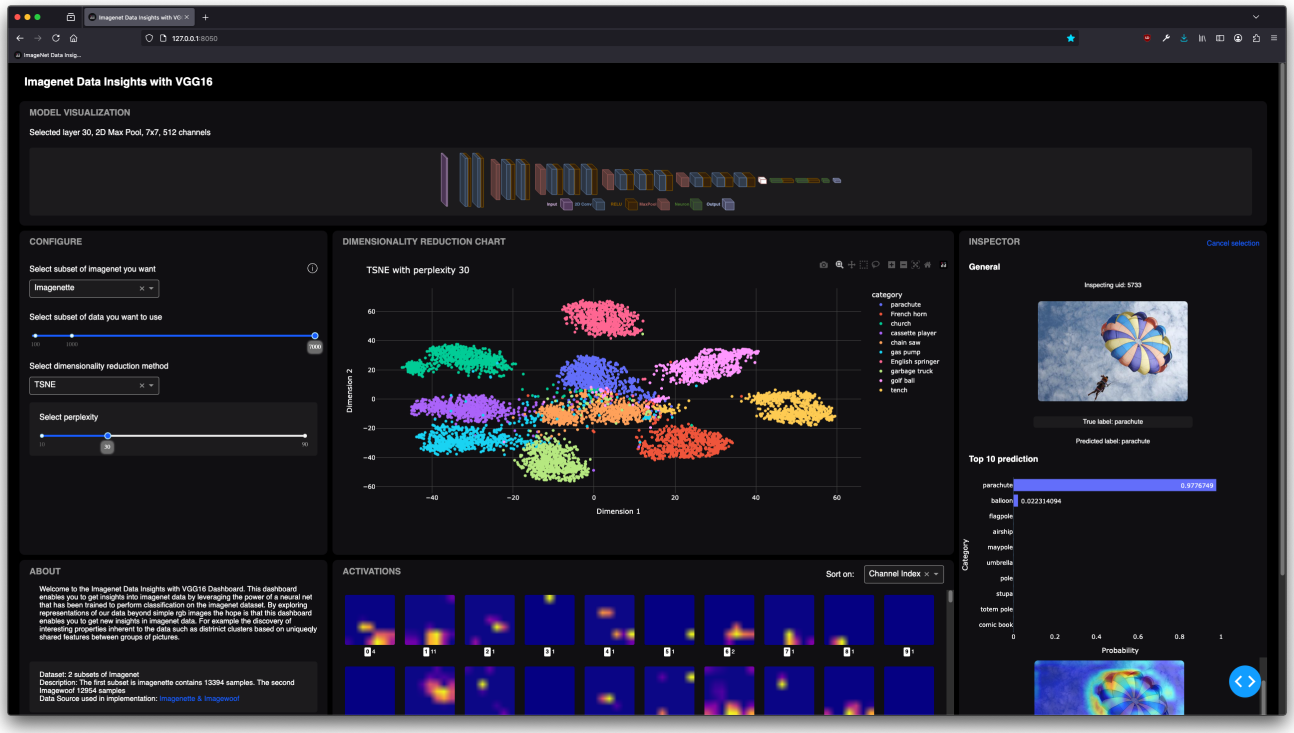

Figure 1: Dashboard using dash: ImageNet data insights using dash

4 Design and interactions

Since the dashboard is quite extensive, a guided tour is available. This takes you through the features of the dashboard. This can be watched via guided-tour.mov . The implemented dashboard is shown in Figure 1.

Careful attention was made to create a clear, aesthetically pleasing and self-explanatory UI. The choice was made to work with a dark color scheme since it is easier on the eyes. The design

was inspired by my own style preferences, modern web applications of today and a bit of Star Trek UI even. There were no UI frameworks used and everything was designed and implemented from scratch (except for the plotly charts) (The css file alone contains over 800 lines).

The dashboard is divided into different sections.

- Model visualization: select the layer to inspect, get a sense of where you are looking in the model.
- Control: control relevant parameters such as number of samples.
- Inspector: show detailed information about the sample. Such as image, prediction, explainability
- Dimensionality reduction: show the dimensionality reduction for the selected samples applied on the activations of the selected layer.
- About: A bit of info about the dashboard
- Activations: Show individual activations for the individual channels

Model Visualization At the top you have the model visualization section. Here you can see a visual representation of VGG16. The selected layer is highlighted in white, and some text gives some relevant info about this layer. This isometric representation represents the true operation of vgg16. To the left you have the input image you can see it is a big cube with little depth representing the big spatial size of the image (224x224) with little channels (3), as you progress through the model the spatial size get smaller and the number of channels get bigger. The model presses the image as it were into a flat vector with a dimension of 1000 corresponding to the amount of channels.

You can interact with this visualization. When you hover over the different layers, you can see them highlight. When you click on a layer, you are now inspecting the dimensionality reduction and activations for that layer.

Control On the center left you have the control section. This section allows you to select which data and which dimensionaluty reduction method is used when using the rest of the dashboard.

- "Select subset"
  - This selects the subset of ImageNet you want. Imagenette is a subset of ImageNet with 10 diverse classes: tench, English springer, cassette player, chainsaw, church, French horn, garbage truck, gas pump, golf ball, and parachute. Imagewoof is a subset of ImageNet that only contains dog breeds. It includes 10 different breeds: Australian terrier, Border terrier, Samoyed, Beagle, Shih-Tzu, English foxhound, Rhodesian ridgeback, Dingo, Golden retriever, and Old English sheepdog.
- "Select subset of data you want to use"
  - This allows you to select different subset of data. You can select 100, 1000 or, 7000 samples.
- "Select dimensionality method":
  - This allows you to select the dimensionality reduction technique. Either TSNE (using openTSNE) or PCA (using sci-kit learn)
- "Select perplexity" - shown when TSNE is selected
  - When TSNE is selected, you can modify the perplexity being used. The perplexity is a parameter that is being used by the algorithm to determine the similarity between points. For a small perplexity value, only a small neighborhood of points will be used around each one. Whereas a large one with a large perplexity. Thus, a low perplexity value will focus more on local relationships, while a larger value will focus more on global ones. The intricacies of perplexity are more complicated than this, and for more information I advise looking at literature on TSNE.

    Low values are often cited as being in the 5-30 range and high ones in the 30-100 range. Therefore, I made values 10, 30 and 90 available.

Dimensionality reduction The middle center section is the dimensionality reduction section. This is the heart of our dashboard and shows a scatter plot of the dimensionality reduction technique being applied on the selected subset of the selected dataset.

The scatter plot has a title to show what technique is being applied. In the case of TSNE it also shows what perplexity value is being used.

The marks of the scatter plot are points that corresponds with one sample out of the dataset. There are two main channels: position and color. The position determines the location of where a point is drawn. A position consist out of an x and y coordinate. For both TNSE and PCA we project our data on two dimensions. Each coordinate corresponds to one of the dimensions. The x to the first dimension and the y to the second dimension. The color channel is discrete and is used on the dots to differentiate between the different classes/labels in the dataset. These are also displayed by a legend to the right of the scatter plot.

The positions describe the relationship that data points have to each other. In this way, it is possible to identify interesting clusters in the data.

Inspector The inspector section in the right allows you to inspect individual data points. You can hover over data points or click on a data point in the dimensionality reduction section and inspect the corresponding sample.

The inspector has different subsections:

1. The corresponding image, it's uid, the true label, and the predicted label
2. A histogram of the top10 predictions for this sample.
3. Explainability on why this label was predicted.
    - A class activation map. This highlights what parts of the image were important in determine this label. 'Where the model was looking'. Uses global average pooling mechanism, a more spatially meaningful explanation.
    - A guided back prop image. This does a similar thing, (it restricts the flow of gradients through the neurons that have low activations). More on a local pixel level.
    - The combination of the guided back prop and class activation map. Local + Spatial.

Activations The activation section under the dimensionality section allows you to inspect the activations of the current layer for a selected sample.

Depending on where you are looking in the model, and depending on if you have viewed the activation before, you have a few different possibilities:

- Nothing selected or in hover mode: The activation view is empty.
- Selected a sample while inspecting a convolutional layer
    - When the activations have not been computed, it will show a compute button. You can press the compute button and the activations for this sample will be computed and save in the background. This might take a minute or so.
    - When the activation has been computed. The activations for each channel will be visible. They will be represented as a grid of thumbnails, where each thumbnail corresponds to the activation of a particular channel. Under each thumbnail there is a highlighted number in white, this is the channel index. The other number is the score of this activation. There is also a legend visible.

      You can sort the channels either on score or on the channel index with te dropdown in the top right of the section. This only has an effect for the convolutional layers.

      The score is a way to identify interesting, better yet to group similar activations together. The score is calculated by multiplying the entropy and the magnitude of the channel. The magnitude is the average activation value across the channel. The entropy is calculated by flattening the 2D activation, and then binning the values in 10 different bins and calculating the entropy (discrete). The hope is that channels that this way we can identify interesting activations more easily. Interesting activations might be where there is a lot going on or where certain activations really stuck out. Thus, when there a lot of different activations (high entropy) and/or high magnitude, the score will be high and vice verca. Low entropy can also be useful, and you can simply look for the low scores too. In fact actually the score groups similar channels

together, and we can decide with our own pattern recognizing with score range is useful to inspect. This way the score becomes another powerful tool in this dashboard.

Additionally, the dashboard has one more trick on up its sleeve. If you click on a channel, a modal will open. This will show the activation for the selected channel via a plotly heatmap. Now you can zoom in to the individual activation neurons. This really shows of the granularity of this dashboard, we can go from a high level overview to the smallest details. It is as if we are performing neuroscience on a digital brain. To cancel the modal, you can click on the background.

- Selected a sample while inspecting a neuron layer
  - The activation for that layer will be displayed in the form of a heatmap. A neuron layer is flat and contains a for example 512 dimensions. To represent this efficiently, the neurons are displayed in a rectangular heatmap. You can zoom in to see individual values.

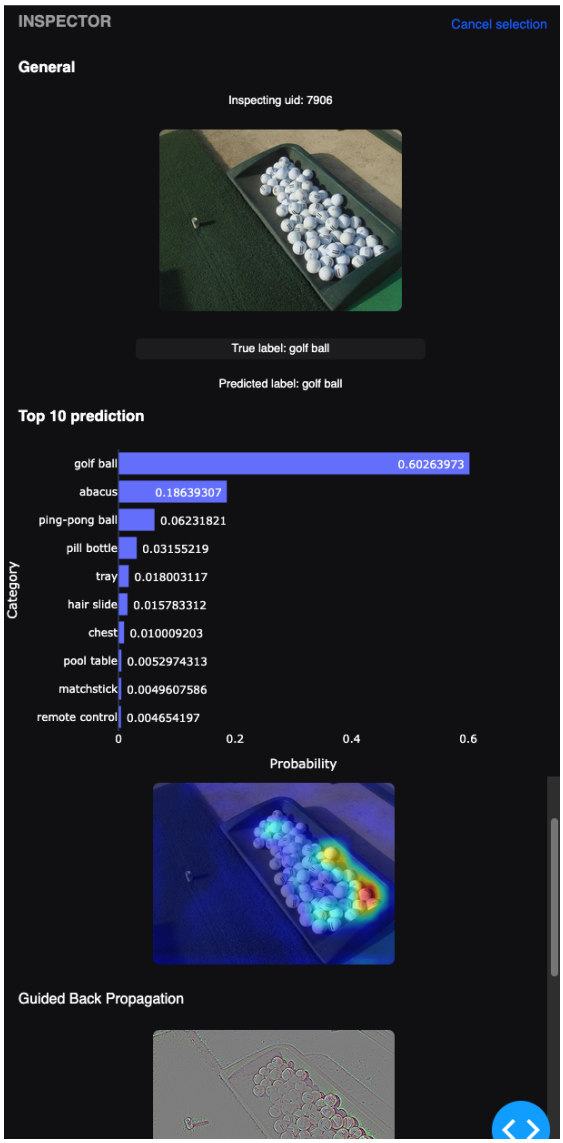

Figure 2: The inspector

Finally, at the bottom left of the page, an about section is available. This presents some general info about the dashboard, and also what the data source being used is.

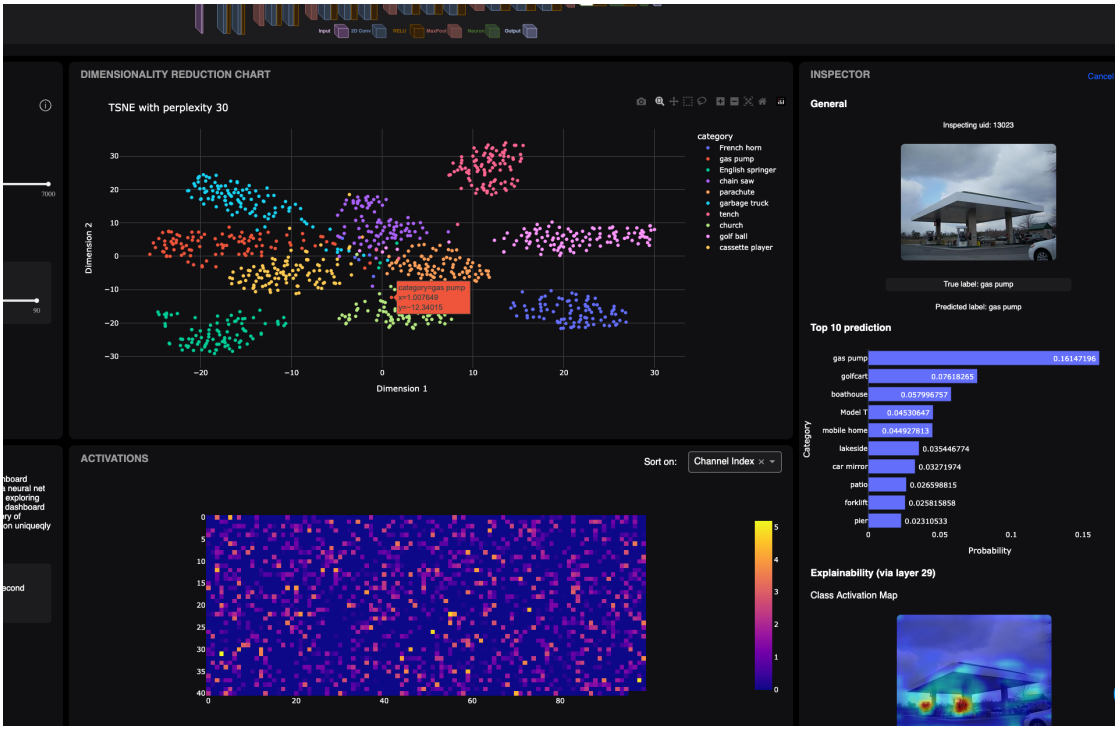

Figure 3: Viewing the neurons of a fully connected layer.

5 Implementation

5.1 General

The implementation of this dashboard turned out to be a momentous task. The combination of a large model, a lot of samples, high resolution samples, and the wish to implement an advanced dashboard with many features made this exponentially more difficult than the last assignment.

5.2 Data pipelines

At first, I thought using the data would be simple, this was not the case. First, using the full ImageNet dataset was intractable. So I resorted to using a subset of ImageNet. However, this data did not match one to one the way the model operated, so a lot of processing needed to be done before even using the data inside the model.

In the end there are 4 data pipelines. One that processes the data from the datasets. One that processes the basic data for the dasboard. One that does the dimensionality reduction, and finally one that does the calculation of the activation values when the dashboard is running. At first, I started out with one, but this was too big and not usable, this quickly became a distirbuted approach.

5.2.1 Processing of the dataset

First the imagenette and imagewoof two subsets of imagenet were downloaded. These contained only sysnetid's as labels. In imagenet you have three types of labels you could say. I treat them as the following:

- Labels: This is the sysnet id: for example n03445777 (this corresponds to golf ball)
- Label ids: This corresponds to the index of the predicted class on the 1000 dimensional output vector of the model. For example, 574 corresponds to golf ball.
- Category: This would be the text label thus 'golf ball'

I made scripts to do the mappings between the three, and searched for the correct mappings online. This was a tedious process but necessary to get everything working.

To handle all of this, a custom data loader for Pytorch was made. The data loader loads in the image and also attaches the correct label, label_id and category to each label. It also keeps

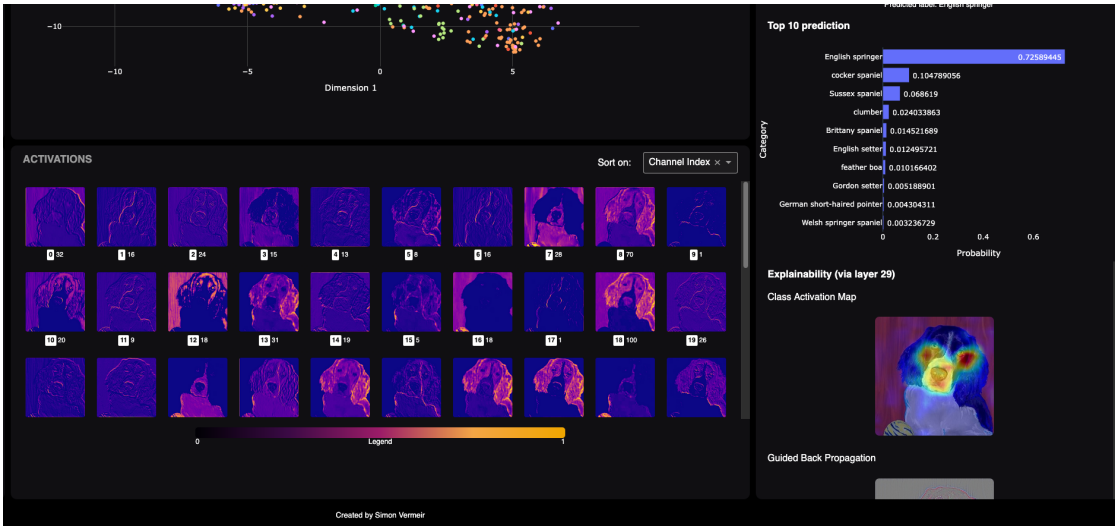

Figure 4: Viewing the activations of a conv layer

track of where the image was coming from and gives it a unique id. This is very important in dealing with the data later in the dashboard.

Take a look at process_annnotations.py to see the code.

5.2.2 Preprocessing data for the dashboard

The pre-processing pipeline, first creates the directory structure for the dashboard to use. This directory structure is key. For each dataset there is a folder, and within a folder each layer of the model has its own folder. Within each folder the corresponding tsne outputs, activation outputs etc. can be found. Take a look at the assets/data/imagenette/1000 folder to get an idea. Although this seems simple, I spend a lot of time thinking about the best structure, this allowed me to create advanced mappings within the UI and the pipelines in a natural way.

Next this pipeline will copy the images to the folder, create an index of all the samples, do the predictions with the model, create the top10 predictions for all samples, do explainability on each prediction. All of this is saved in a structured way in this directory structure.

The explainability on each sample are done via the gradCAM library.

Take a look at data_processing.py to see the code.

5.2.3 Dimensionality reduction

The dimensionality reduction pipeline will push the selected samples through the network. It will capture the activation values at each layer and perform dimensionality reduction on them.

The proved to be another technical challenge on its own. At first a naive approach was taken. Simply capturing the values and performing TSNE or PCA directly on them. However, I quickly discovered this was intractable, this is due to the big spatial size of the samples and the size of the model. For example, at layer 0 we have 224x224x64, that is a vector with a dimensionality of over 3 million that needs to be rescued to 2.

After some studying, I found a hierarchical approach:

1. First the 2D outputs are spatially downsampled by a factor of 4 (8 for the 7000 samples dataset). (Only for the conv layers)
2. Next PCA is performed and 90% of the variance is retained
3. Finally, the TSNE or PCA to 2D is performed. Now its input is around 300-1500 much more manageable!

Take a look at dimensionality_reduction.py to see the code.

5.2.4 Online activation processing

At first, I wanted to pre-compute the activations for all samples on all the layers. But after some experimentation, I discovered that I would have over 1TB of activation data. This would be

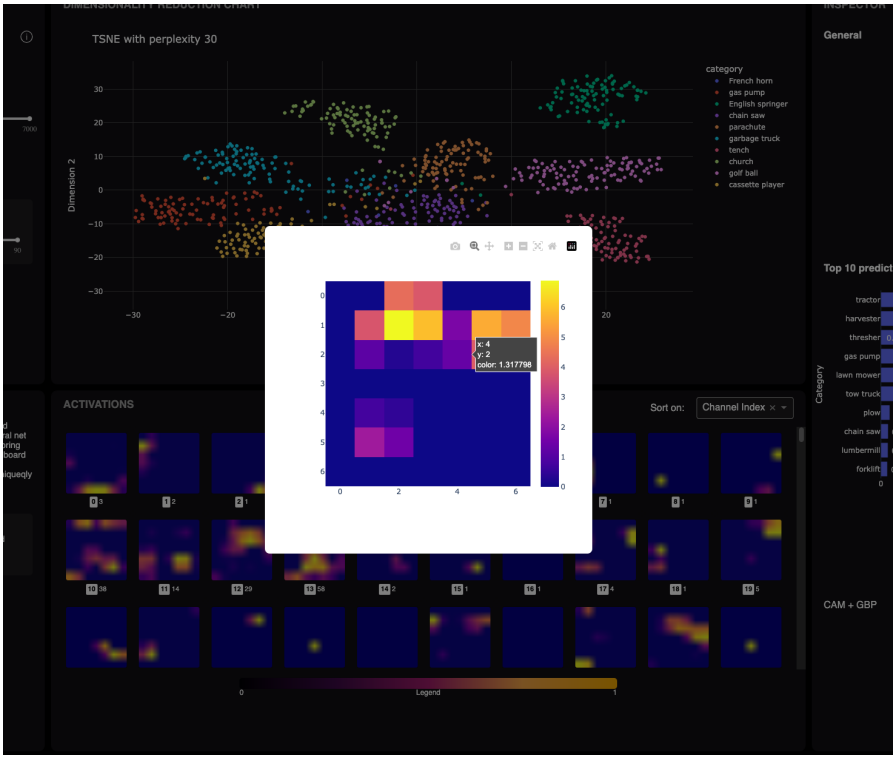

Figure 5: Inspecting an individual channel using the modal pop up

too big to turn in as an assignment (my pc even has only 1TB). Thus I developed an approach where you can select a sample in the UI. Then, depending on the situation, you can either press compute to compute the activations if they are not available. If they have been computed earlier it will show them directly, and if they are from the flat neuron layers it will also show them immediately since these are pre-computed.

For the neuron activations, I could simply capture the values and show them via plotly heatmap.

However, for the activations of the 2D conv layers that have multiple channels, a more advanced approach was needed. I first capture the layers, then I turn this into a heatmap for each channel and this is then exported as a .png. These in fact become little thumbnails. These can then be displayed in a grid and in this way you can see all channels and there activations with one look. This was not possible in plotly directly, therefore I needed this custom approach. A cool addition I also exported the raw activation values of each channel, so if you click on a channel you get a heatmap of that channel, you can zoom on this to really inspect the activation on its lowest level.

Moreover, for each channel, a score is also calculated. This score is the entropy x magnitude. This is the entropy of the flattened channel together with the average magnitude of the activations. This score is stored inside the filename in a special format and extracted by the UI. I explained the idea behind this after the exam session.

To make this on the fly computing all work I needed to redo a lot my implementation, and a lot of mechanisms became much more complicated but in the end it worked.

Take a look at data_manager.py to see the code and main.py for the UI mechanisms.

5.3 The Dashboard

The dashboard was implemented in main.py. A lot of tricks were used to achieve the desired interactions. Although Dash is a fantastic framework for creating basic dashboards quickly, in my opinion if you want to do custom things, or complex interaction mechanisms that are multi layered this framework is not suited. I had to take a lot of shortcuts and write a lot of unnecessary code to catch all the dash quirks. I understand the callback mechanism and used it quite effectively in the 3th assignment, however, here because of the multi layered interactions this was difficult to deal with.

5.3.1 VGG16 Visualization

Creating the visualization of the vgg16 model was a project in itself. Before starting this, I thought it would be relativly simple. I would simply draw the model in a vector program and then export it as SVG. Then i would target the different svg parts with css.

Firstly, drawing this took a lot of time. Next targeting the different attributed was not easy and in the end I needed to painstakingly map all the different parts of the drawn SVG to my layers.

Finally, I could not import the SVG inside Dash. Since I discovered that Dash does not support SVG directly. So I needed to create a custom dash component. However, the current version of their component creator does not compile. Luckily, I created a custom dash component in the past. So I recovered this code and completely changed it to work for my visualization.

This dash component displays the SVG and makes it interactive. This is done in React (The code can be found in 'custom_components/vgg16_svg_model_viewer/src/lib/components/Matrix.react.js'). In a sense, the component takes in which layer to select and will select that layer, and it also outputs which layer is currently selected. You can capture what layer is selected and if another layer gets selected you can fire a callback inside dash.

6 Running the dashboard

To run the dashboard:

1. Open the folder DVAI_lab4_DL_dashboard in your favorite IDE. (I used Pycharm)
2. Copy the assets folder from the assets.zip into this folder.
3. Create a venv using Python 3.12, and install the packages from the requirements.txt file.
   - pip3 install -r requirements.txt
4. Additionally, you need to install the custom Dash component for displaying the model. The code can be found in 'custom_components/vgg16_svg_model_viewer/src/lib/components/Matrix.react.js'. The rest of the code in this folder custom_components is mainly auto generated.
   - pip3 install ssp_matrix-0.0.1.tar.gz
5. Run main.py file. It will take a while to start up. All data is preloaded in the background. Please wait around 30 seconds before running the browser.
6. You will be prompted with a URL, open this url.
7. It is possible that the css is still compiling in the background. Please wait another 30 seconds and reload then the dashboard should load.
8. You will now see the dashboard. Please use a big enough browser window to get the full experience. On a large screen monitor it is best I used 2500x1600.

Important to note:

- Please note when hovering for the first time over a data point it takes a while to cache the necessary data, therefore it might take a while for the inspector to show the data point. However, subsequent hovers will perform smoothly.
- Because of limited compute, some shortcuts were taken with regard to the data shown. The UI could have let you know this more gracefully, but implementing this would take a lot of time this is a proof of concept, therefore I mention it here:
  - For the 100 samples datasets, the actual perplexity values do not correspond to the ones shown. Since there are only 100 points, they are at least an order of magnitude smaller. Around the 1 - 5 range.‘
  - For the imagewoof dataset, the 7000 samples dataset is actually the same as the 1000. Computing the 7000 for the imagenette dataset ImageNetdy more than 24 hours. In total, probably around 30-40 hours of computation was needed to achieve the processed datasets (on CPU).

  - The model consists of 37 layers (input and output included). We have the ability to select each layer and inspect a dimensionality reduction, these are a lot of reductions. Therefore, some shortcuts were taken to limit the compute needed:
    * The dimensionality reduction for the conv layers are copies of the dimensionality reduction of the Relu activation layer that follows them. After some observation, it seemed that almost all relu activation were the same as the conv outputs (pointing towards the fact that probably negative values did not flow through the network maybe due to normalization etc., this was not further studies). Therefore these were copied since it cut computation time almost in half.
    * For the 7000 dataset were still too many reductions needed to be done (still 20+ layers). Therefore, there was only one reduction done per convolutional block on the last relu layer. A block can easily be identified via the model visualization. So after the input layer 0 (conv), layer 1 (conv), and layer 3 (relu) form a block. Reduction is done on the relu layer and all other layers take the same reduction.
    * For the 7000 dataset, all perplexity values corresponds to the 47 perplexity value. Since doing this for 10, 30, and 90 would require 3 times the compute time.
  - If I would make this for production, I would naturally not do these shortcuts.

- If you play a lot with the UI it is possible to find small bugs. the many interaction possibilities cause a lot of edge cases. Catching these all requires a lot of work and testing. This was outside the scope of this project.

- The UI is not always ultra fast. A lot is happening in the background without you even noticing (for the user, all the complexity is hidden as much as possible). A lot of it has to do with the complex interaction mechanisms and the multi layered interactions I have, such as the activations view. Dash is not suited for a lot of customization, and a lot of tricks were needed. Optimization is possible but would take a lot of time.

- Please note that when running dash, sometimes it might struggle to reload its CSS file in the background. The layout will look broken then. However, when starting up and just running and using the dashboard it should not happen, but during development and with auto compile it might. Just reloading the page does the trick to fix it. A similar thing might happen with the style of the dropdown, this dropdown is the dash standard component, but is heavily modified (doing this directly through dash was not an option, and it needed to be targeted via CSS).

If you have any trouble getting the dashboard to work. Don't hesitate to contact me.

7 Conclusion

During this lab the ImageNet data insights dashboard using VGG16 was created. A powerful way to get insight in ImageNet data and the vgg16 model. Thanks to the extensive features, you are able to get a lot of new and interesting perspectives on how neural nets work and the data they use.

In the end, I think I achieved the goals that I put fort. However, it took a lot more work than I thought. Working with the large file sizes, and bigmodel,l brought on a lot of unexpected design challenges. A lot of engineering tricks needed to be applied to get it to work. Therefore, the scope of this project far exceeds what was needed for this lab.

However, I am happy I created this. I think this can not only be useful as a lab assignment, but can be used for other projects too. Moreover, with some additional work, I can make things like the vgg16 visualization dash component and even the dashboard itself available to the public. I believe this really can give you an appreciation for what these networks actually do. It did indeed impress me the vast amount of data that actually flow through these models, and they really showcase our human ability.