Model Interpretability and Visualization: Looking Inside the Neural Network Black Box

Avinash Follow
Apr 15, 2019 · 9 min read

Introduction: What is Deep Learning?

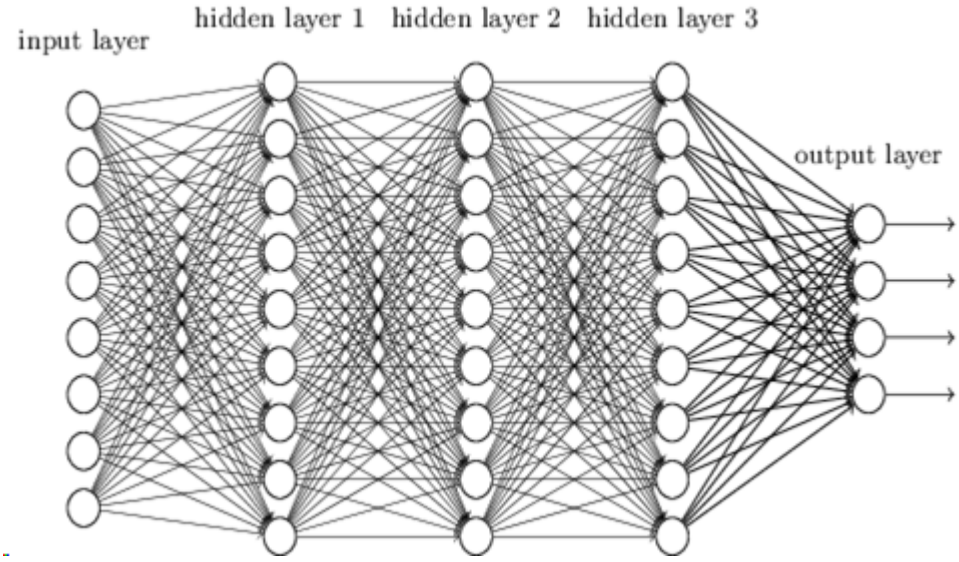
There has been tremendous development and progress over the past few years in how some technologies have impacted and blended into our human lives. Deep learning has been at the forefront of such technologies and has found applications in many industries — retail, health, automobiles, finance, and transportation to name a few.

Deep learning uses artificial neural networks to learn representations and patterns in provided data and comes up with task-specific outputs. The cabs we take to reach the office, the algorithms that sort out our emails, the recommendation engines behind apps like YouTube, Netflix, Amazon, and many more applications like these are powered by deep learning.

Given enough data, deep learning models learn from the examples provided and automatically extract patterns that exist in the data, which in turn helps in performing various tasks.

Some of these tasks include object detection, image classification, language understanding, speech recognition, and more. The amount of data available has grown exponentially in the past decade. That along with high computational capabilities have fueled the application and growth of deep learning.

The artificial neural networks used in deep learning are inspired by human biological brains. The way we perceive things around us starts with taking inputs from sensory organs, which get passed through layers of neurons. Deep learning models have a similar layered structure in which learning happens incrementally, with each layer building on top of the layer before it.



Pic Credits

. . .

Machine learning models don't have to live on servers or in the cloud — they can also live on your smartphone. And Fritz AI has the tools to easily teach mobile apps to see, hear, sense, and think.

. . .

Deep Learning vs Standard Algorithms

In a nutshell, deep learning models take some input, use their layers to process the input information, and provide task-specific outputs.

But isn't that what the traditional algorithms do as well? On a very high level, these models, are like any algorithms we use to automate a task. The main difference between the two is how they internally process the data and operate on it.

Here we have an example of a binary search algorithm. It takes in a list of numbers and a query item. It has hand-coded logic that works on the data and returns whether the query item is present in the list. As we can see, every part of the algorithm is interpretable and we can easily understand what the algorithm does:

```
def binarySearch(alist, item):
    first = 0
    last = len(alist)-1
    found = False

    while first<=last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
```

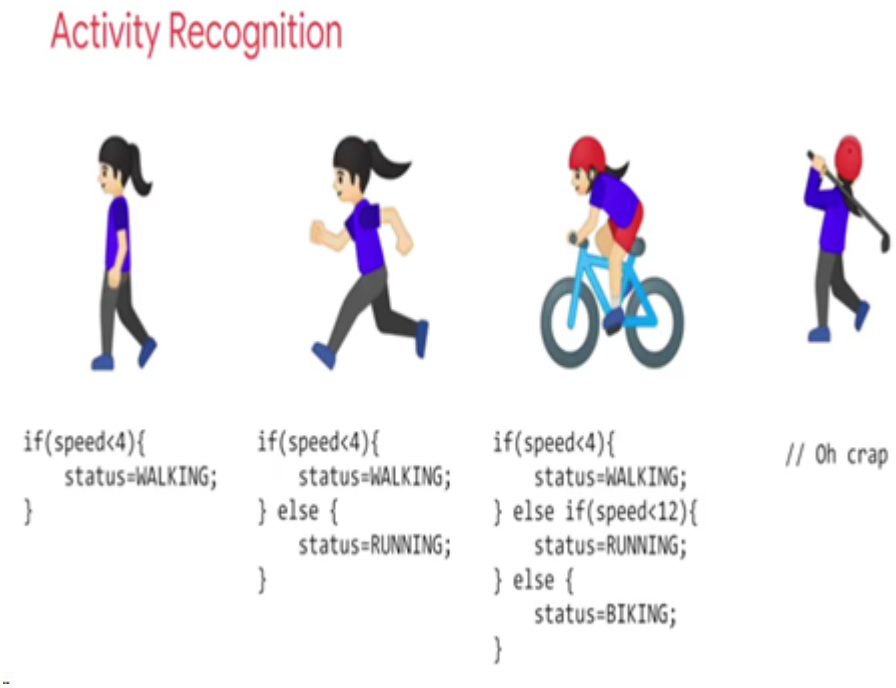
```
    if item < alist[midpoint]:
        last = midpoint-1
    else:
        first = midpoint+1

return found
```

Pic Credits

Lets take another example. Given data like the speed at which a person is moving, an algorithm might have to recognize the person's activity. For a few possible outcomes like recognizing whether the person is walking, running, or cycling, the rule-driven approach works perfectly.

But what if we have another activity like playing golf? How can the same algorithm recognize this new activity? More rules? Perhaps, but it's not hard to see the problem or flaw in this approach. More complicated tasks ask for more and more rules. Deriving these rules manually can be time consuming, error prone, inefficient, and (most importantly) it makes the algorithm less generic. It is also practically impossible to manually derive rules for more complicated tasks.



Pic Credits

The image below helps in summarizing all of these key differences. While traditional algorithms take data along with hand-coded rules to produce the output, ML models take samples of the data along with the correct answers. They then use this to understand the patterns in the data to automatically derive the rules.

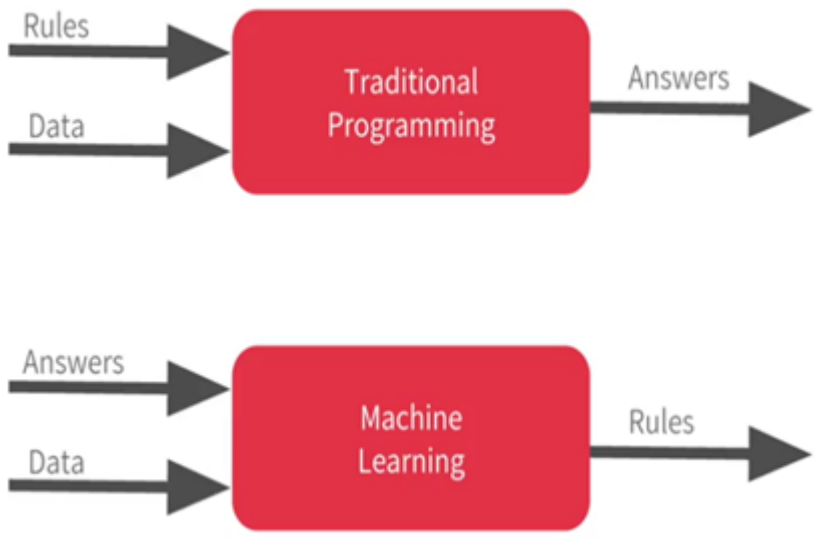
. . .

> The future of machine learning is on the edge. Subscribe to the Fritz AI Newsletter to discover the possibilities and benefits of embedding ML models inside mobile apps.

. . .

Model Interpretability and Visualization

So how do neural networks manage to find patterns in data? How do they learn from existing data, and what do the inner layers of these artificial neural networks look like? All of this is studied under fields referred to as model interpretability and visualization, which are currently very active areas of research.



Pic Credits

To get an overview of what model interpretability is and why it's critical, let's take an example problem. The first thing we need to solve any problem is data. For this example, we'll use the dataset called Food-101, which has 101,000 images in total, with 101 types of food.

Each type of food has 1000 example images—750 of them for training and 250 for testing. It's a normal procedure in machine learning to split data we have into training and test sets so that we can train our model on training data and use the test data for evaluating a model's performance.

We'll be doing the same for this example, in which we'll build and train a model for multi-class food classification. What does it exactly do? Well, we're basically going to extend upon the famous hotdog/not hotdog example from HBO's Silicon Valley and make it work on 101 types of food instead of just one!

Training Our Model

The entire code for Food-101 Multi-class Food Classification is published here as a Kaggle Kernel. Once extracted, the dataset contains a folder with 101 sub directories.

Each sub directory contains images of a specific food class.

Food-101 Dataset snapshot

Using `train.txt` and `test.txt`, which contains the file names that belong to train and test data respectively, I split the data into train and test folders using the helper method `prepare_data()`. At the end of this step, we'll have a train folder with 75,750 images, and a test folder with 25250 images.

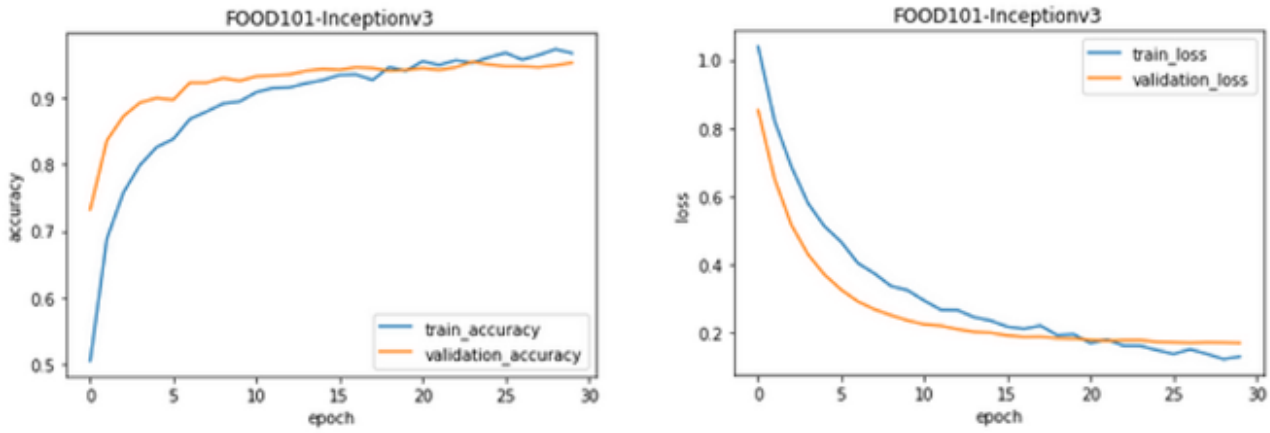
Working with the whole dataset would require a lot of training hours. Since I was using Google Colab for model training, I decided to go with a subset of data. To start with, I chose 3 food classes — apple pie, pizza, and omelette—to run my experiments and train the model. The idea is to extend this to more classes (11 classes) once the model is performing well.

We now have `train_data` and `test_data` with only 3 classes of food for model training. Instead of building and training a model from scratch, I used the Inception V3 pre-trained model as this helps in faster training. After training the model for 30 epochs, we see a validation accuracy of 95%.

Below are the plots showing accuracy and loss when the model is trained on the aforementioned 3 classes of data. As a next step, I repeated the training by extending the data to 11 classes and the results were similar. When I tried to train the model on the whole dataset, it took 50–55mins per epoch on Colab.

Though I didn't run the model for enough epochs on the whole dataset, the model would give more or less the same kind of performance when trained on whole data. This may not be the case for all datasets, though. In this case, the ImageNet dataset on which

the pre-trained model is trained on already has classes of data that are similar to that of Food-101. Hence the pre-trained model was providing good results.



Accuracy and Loss plots

How Neural Networks Learn

Now that we have a model that's giving us 95% accuracy, does that mean we are good to go ahead and deploy the trained model into production? This could have been the case if we were dealing with standard algorithms.

But with neural networks, it's not enough that the model is giving expected and correct results. It's also necessary and critical to know how it's doing it. To understand this better, let's put aside our food classifier for a moment and think about other applications where deep learning is used. For example, self-driving cars, cancer detection, face recognition, etc.

In all these scenarios, it's critical to make sure that the neural networks are learning the right features/patterns. This isn't only useful in improving the model performance, but more importantly, it's critical to know the "why" and "how" as deep learning gets adapted to industries like healthcare, autonomous vehicles, the legal system etc.

This is where model interpretability and visualization come into play in better understanding neural networks' outputs.

Neural networks learn incrementally. The first layer looks for edges and gradients. The second layer builds on top of it and looks for curves and simple patterns. We can see

below how the visual interpretability is lost as we go deeper into the neural network. That's because the deeper layers, instead of looking for edges and curves, try to find class-specific and abstract features.

To better understand this, think about how we would hand-sketch a dog or a cat on a piece of paper. In order to draw the ear, snout, or tail of a dog, we'd need to first start with lines. We'd then put those lines together with curves and other patterns to show more abstract features. Similarly, neural networks learn to recognize/learn features in this way.

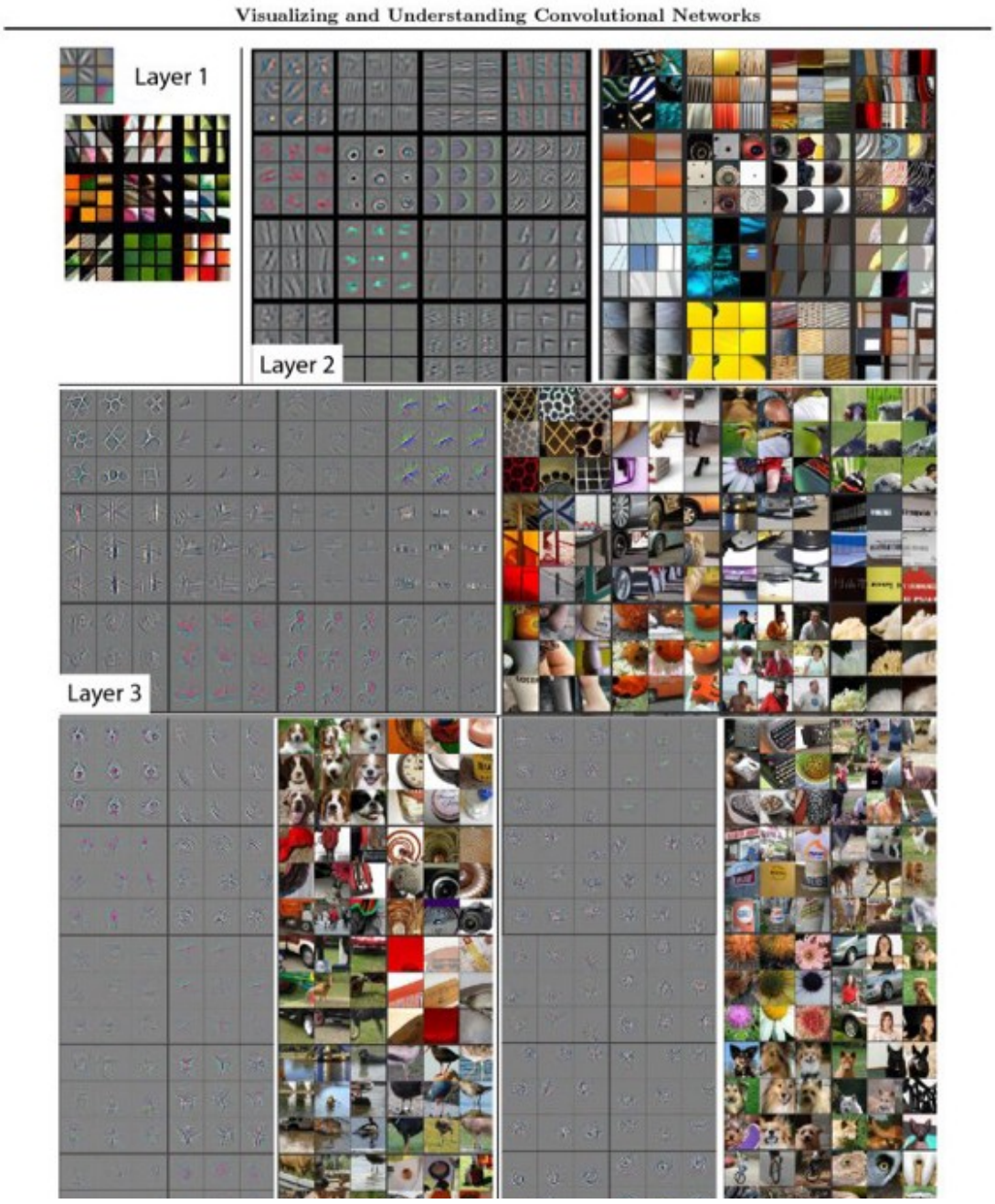


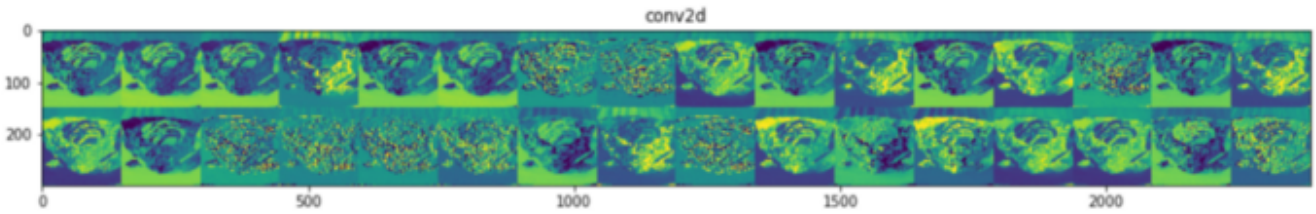
Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.

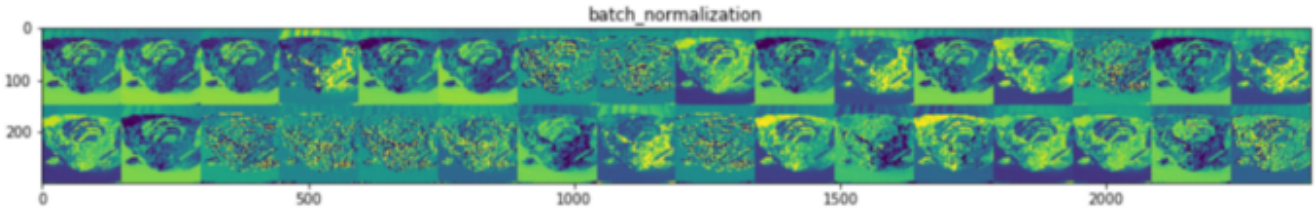
Pic Credits

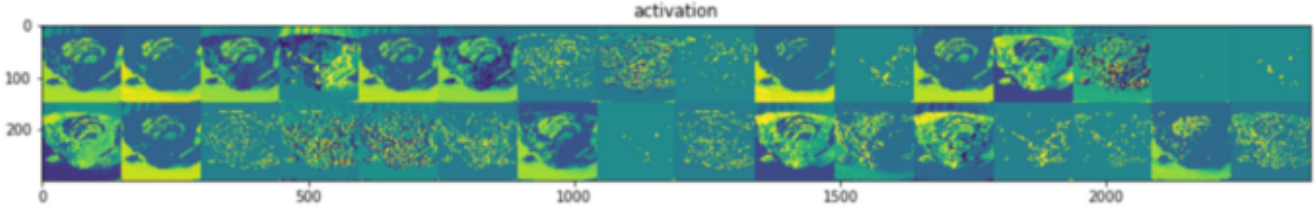
Visualizing Our Model

Let's now return to the model we trained to visualize the inner layer activations. The first layer in Inception V3 is a convolution layer with 32 filters (**conv2d** in the below image). The input given is an image of an apple pie.

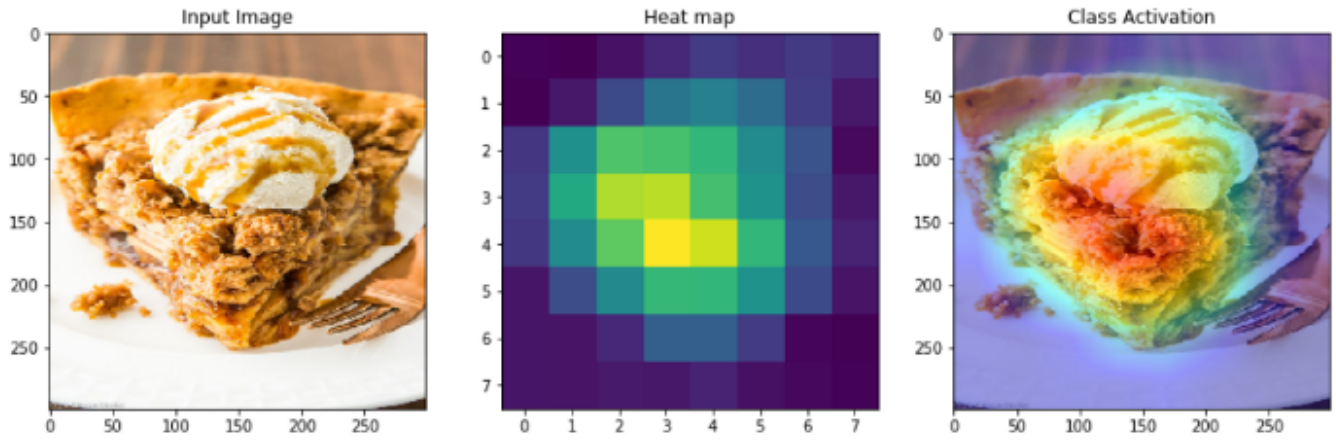
Each cell in the grid of images below is an activation. An activation is the output obtained after a filter is applied on the input from the previous layer. While all of the activations are visibly interpretable as apple pie in the first **conv2d** layer, we can see some sparse activations in the third layer (**activation)**. As we go deeper into the neural network, we'll have more of these sparse activations.



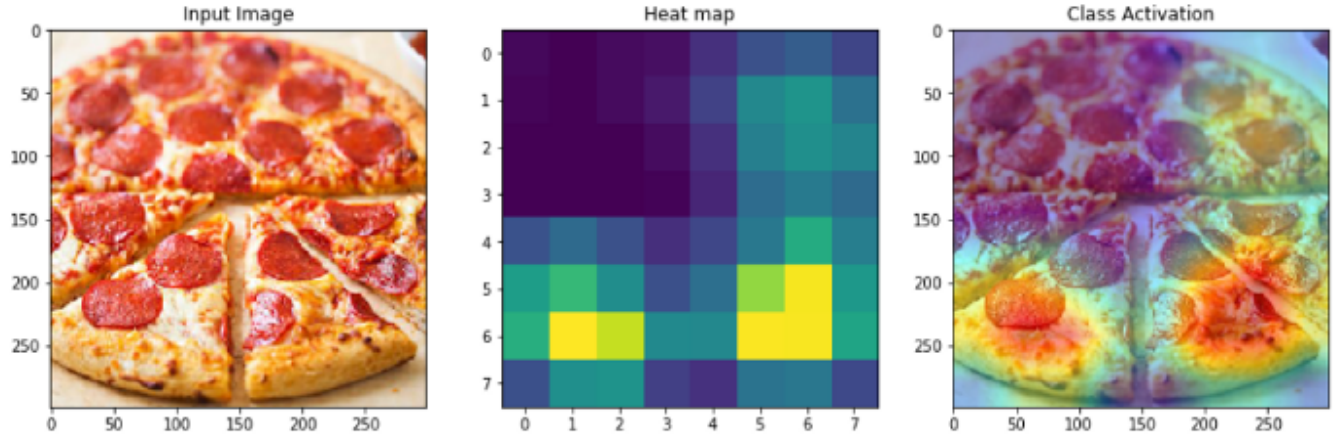




Activation map visualization helps us understand how the input is transformed from one layer to another as it goes through several operations. We can also generate heat maps using gradients to find out which regions in the input images were instrumental in determining the output class. As we see in the below two images, the class activation maps are different for different input images. This helps us understand what a neural network looks for in an image to classify it as an apple pie or a pizza.



Class Activation Map for apple pie



Class Activation Map for pizza

Conclusion

Using these methods along with several others, we can get a glimpse of a neural network's black box and its inner layers. We can use these techniques to verify whether the model is learning the right features for classifying images correctly. When there's bias in data, the model might fail to classify or learn the right features.

While performance metrics like accuracy and loss can give us an indication of bias in data, model interpretability can provide further evidence of what's going wrong with the data.

With deep learning becoming ubiquitous, it's critical to understand what's going on inside neural networks. There's still a lot of progress to be made in this direction, and I hope this blog gave you a quick overview of what model interpretability is.

References:

Deep Learning with Python by Francois Cholett — a must read!

Building Powerful Image Classification Models

How Convolutional Neural Networks See the World

The Building Blocks of Interpretability

Feature Visualization

> ***Did you find this post useful? Feel free to leave any feedback/comments. Thanks for reading!!***

To connect : LinkedIn, Twitter and my Blog.

. . .

Editor's Note: ***Heartbeat*** *is a contributor-driven online publication and community dedicated to exploring the emerging intersection of mobile app development and machine learning. We're committed to supporting and inspiring developers and engineers from all walks of life.*

Editorially independent, Heartbeat is sponsored and published by ***Fritz AI****, the machine learning platform that helps developers teach devices to see, hear, sense, and think. We pay our contributors, and we don't sell ads.*

*If you'd like to contribute, head on over to our **call for contributors**. You can also sign up to receive our weekly newsletters (**Deep Learning Weekly** and the **Fritz AI Newsletter**), join us on **Slack**, and follow Fritz AI on **Twitter** for all the latest in mobile machine learning.*

Thanks to Austin Kodra.

Machine Learning | Deep Learning | TensorFlow | Guides And Tutorials | Heartbeat