Visualizing and Understanding CNN

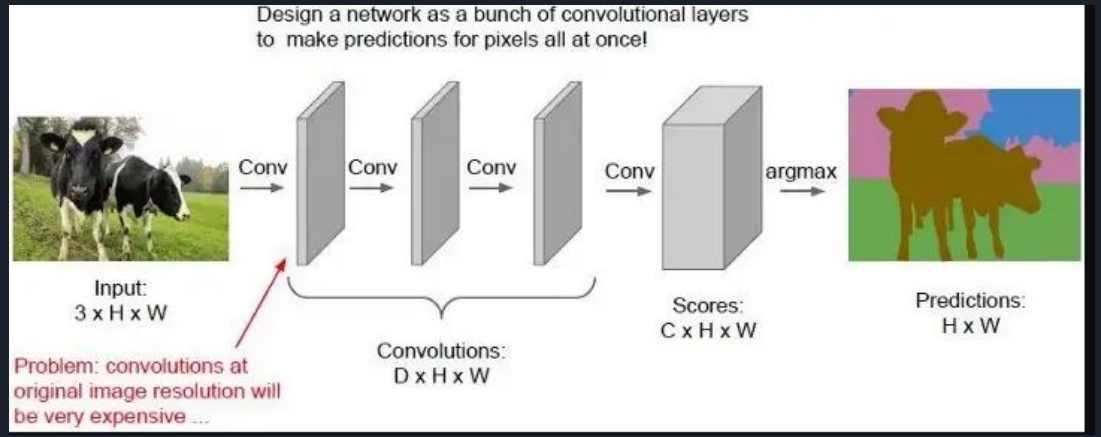
Team - 49

Adhiraj Deshmukh - 2021121012
Shikhar Saxena - 2021121010
Shreya Patil - 2021121009
Shreyansh Agarwal - 2020101013

Motivation

In the last few years CNNs (Convolutional Neural Networks) have started to grow in popularity especially in the field of image recognition. However, this has always been a sort of Black-Box and nobody knows how they work internally. To try to understand this, we have tried to make a Deconvolution network so that we can convert the feature map to human recognisable images.

This helps us not only to visualise the inner workings of CNNs but may also help us to tweak certain parameters in our model to make it perform better.

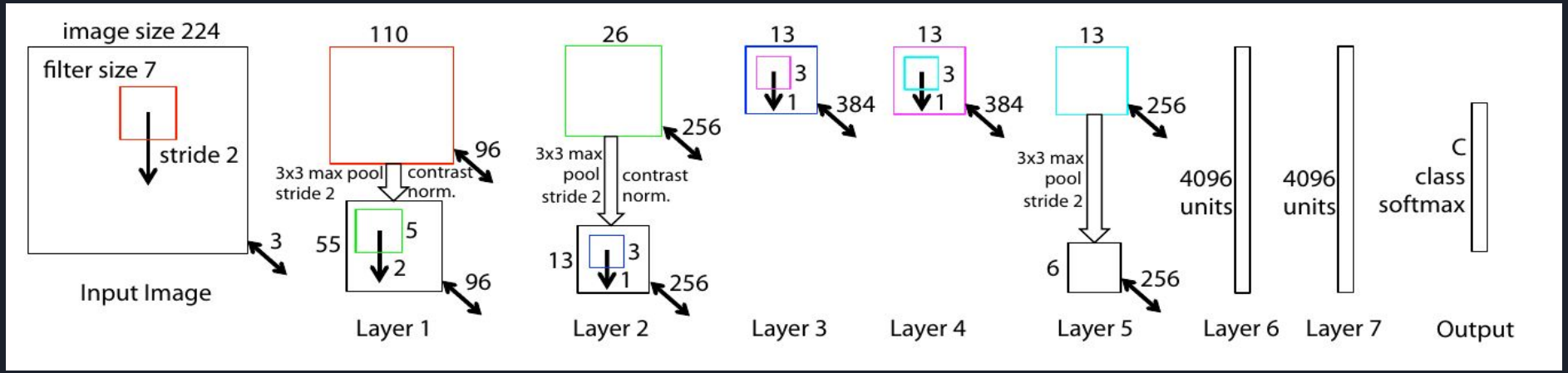


Convnet Architecture

We first made a CNN which gives the probability of an image belonging to a particular class. This is similar to the model proposed by Krizhevsky et al. in 2012 (AlexNet).

There are multiple layers in the model, each layer containing -

- A Convolution of the previous layer output.
- Passing it through a rectified linear function *Relu(x)*.
- Optional max pooling
- Optional local contrast operation that normalizes the responses across feature maps.



Deconvolution using Transposed Convolution

We use the transpose of the learned filters in the Convnet to get the inverse in the deconvnet stage. Below is a small representation of the same.

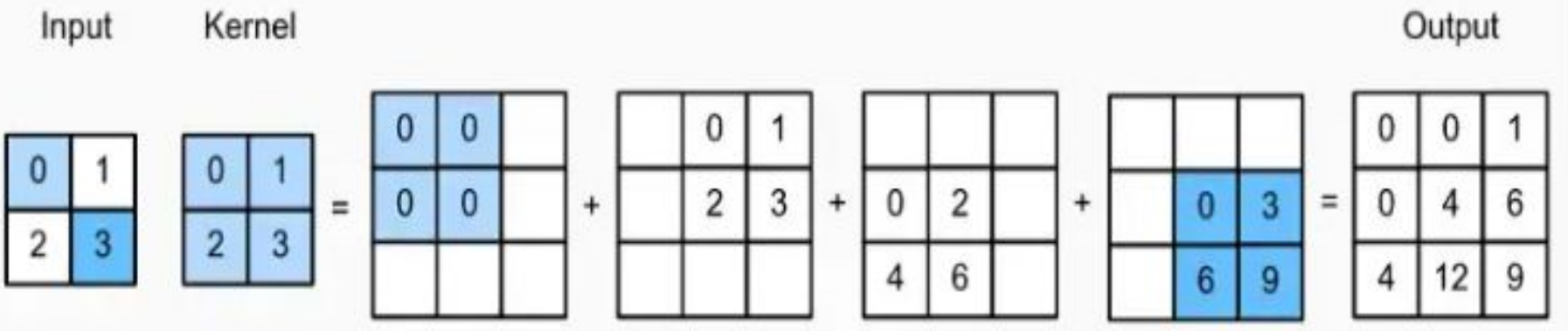


Figure 12. The Complete Transposed Convolution Operation

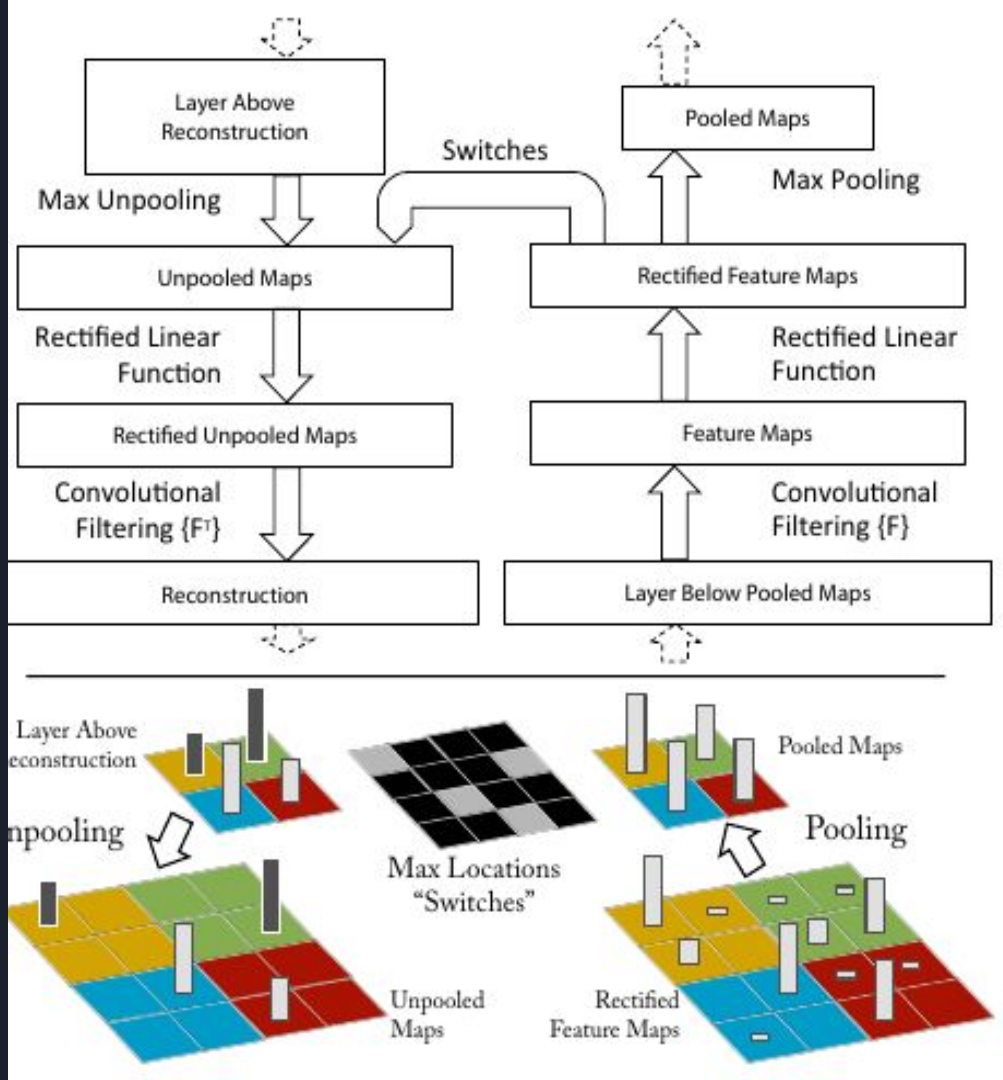
Deconvnet

The Deconvnet can be thought of as a reverse of the whole Convolution part of the model. We do all the operation one by one, but in reverse order. All the weights of the convolutional layers are used by the deconvnet and there is no need to train it. Hence, it used only as a probe of an already trained convnet. The various step are -

- **Unpooling** - The max-pooling operation is non-invertible, but, we get an approximate inverse by recording the location of the maxima (switches) and using them to reconstruct the layer.
- **Rectification** - We pass the reconstructed feature maps through ReLu non-linearity.
- **Filtering** - The deconvnet uses transposed versions of the filters learned in convnet, to the rectified maps to generate a reconstruction of the layer.

Structure of Deconv

- To examine a given convnet activation, we set all other activations in the layer to zero and pass the feature maps as input to the attached deconvnet layer. Then we successively unpool, rectify and filter to reconstruct the activity in the layer beneath that gave rise to the chosen activation.



Training Details

- We tried to train on a subset of the ImageNet dataset, using 10 and 100 classes.
- The images have a dimension of 256 x 256, but all the images were sub-cropped to 224 x 224 during learning.
- We used a stochastic gradient descent with a mini-batch size of 8 to update the parameters.
- The learning rate was started from 0.01, and then annealed manually whenever the error was very large.
- The training was done for 70 epochs

Attempted Implementation for Architecture

Attempt #1 : Paper Implementation (AlexNet)

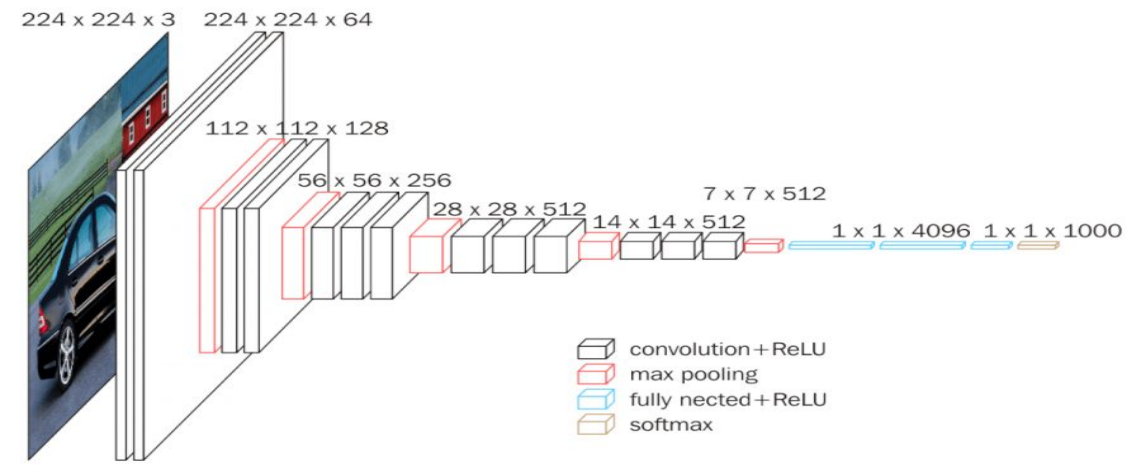
- Our first attempt was training the ConvNet based on paper implementation (AlexNet).
- The ConvNet had 5 convolution layers and were trained on ImageNet dataset with 1000 classes, the MaxPooling index were stored in switches.
- DeconvNet used the similar convolutional architecture but with transposed filters.
- The learned weights were used to initialize the weights of DeconvNet Transpose Convolution Filters, and the switches were used for MaxUnpooling.

- Unfortunately due to resource constraints and some unknown problems we were unable to train this architecture properly.
- And thus we used another Model architecture with pre-trained weights to construct the DeconvNet for visualization of our experiments.

Attempted Implementation for Architecture

Attempt #2 : VGG16 with Pretrained Weights

- For out attempt 2, we decided to select VGG16 which had one of the better accuracies on ImageNet dataset with a relatively simple architecture.
- Similar to our previous model, VGG16 also just has series of Convolutional layers and Maxpooling layers, followed by Densely connected linear layers for learning the extracted features.
- We used pre-trained weights from standard `pytorch` library, trained on ImageNet2012 Dataset.
- For DeconvNet, we used these weights to create the Transpose Convolutional Layers and had similar implementation as our previous attempt.

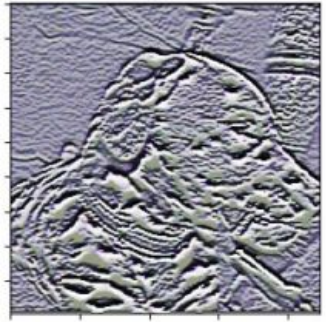


Feature Visualization

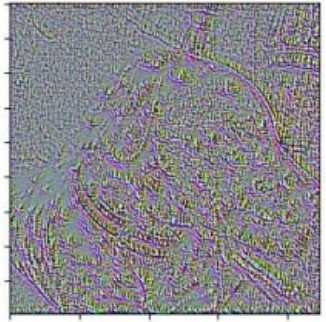
We visualized the feature map after each layer.

- The first few layers were all very simple and only focused on simple geometry like lines and or simple curve.
- However, the later layers can identify complex non-linear geometries like spirals, circles, and even faces.

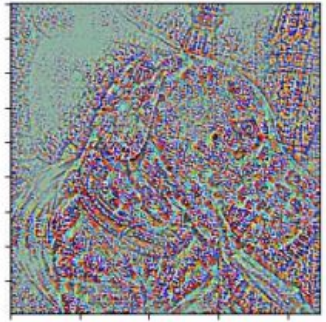
Layer 1



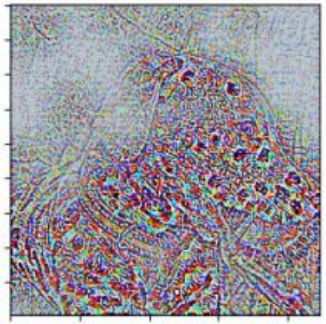
Layer 2



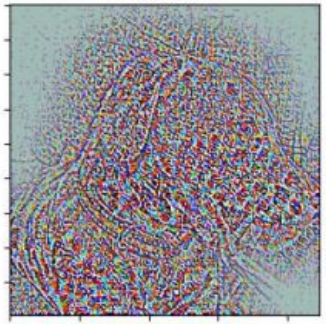
Layer 3



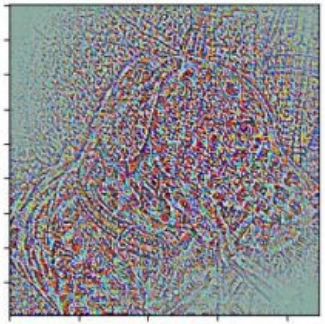
Layer 4



Layer 5



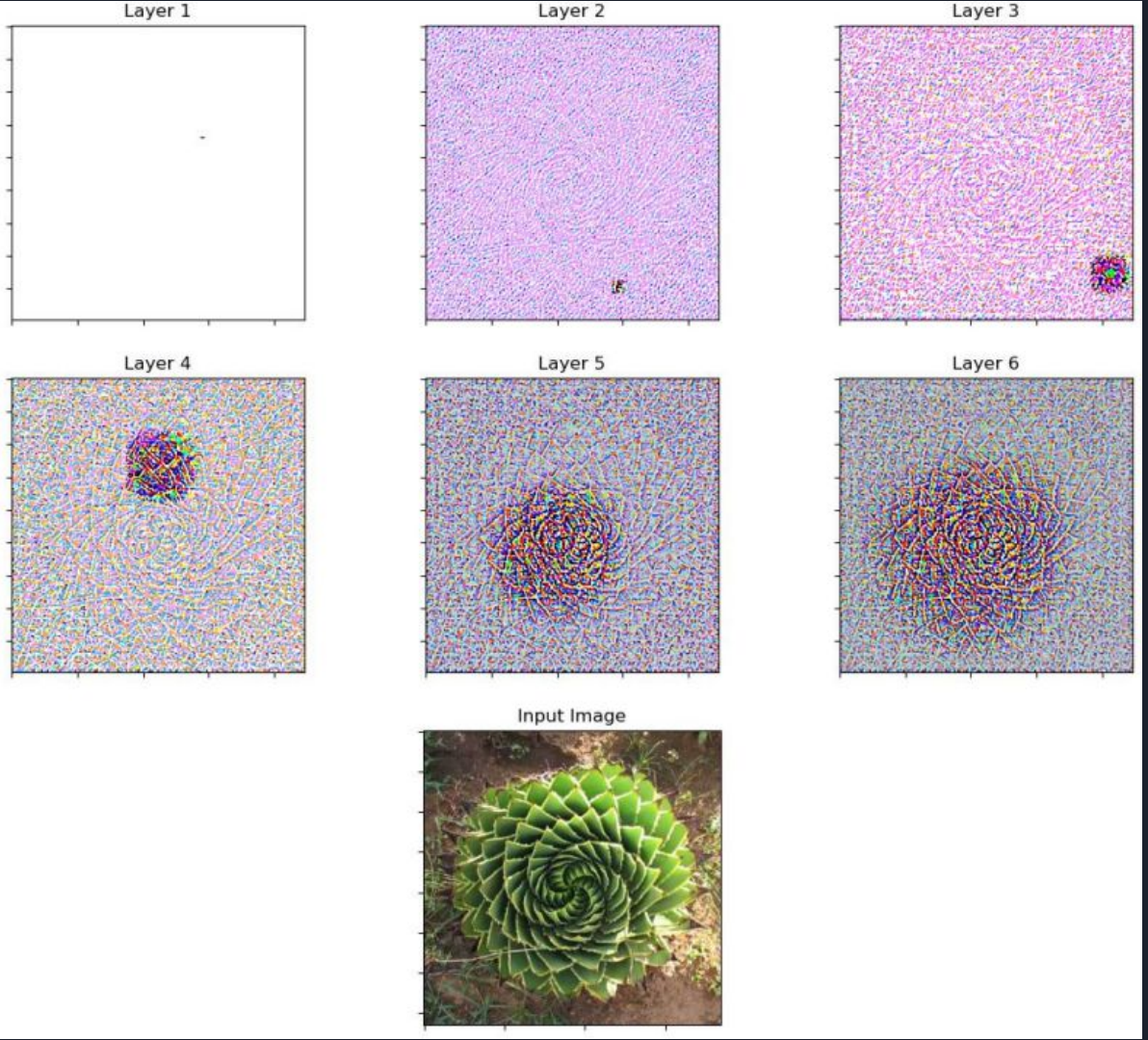
Layer 6



Input Image

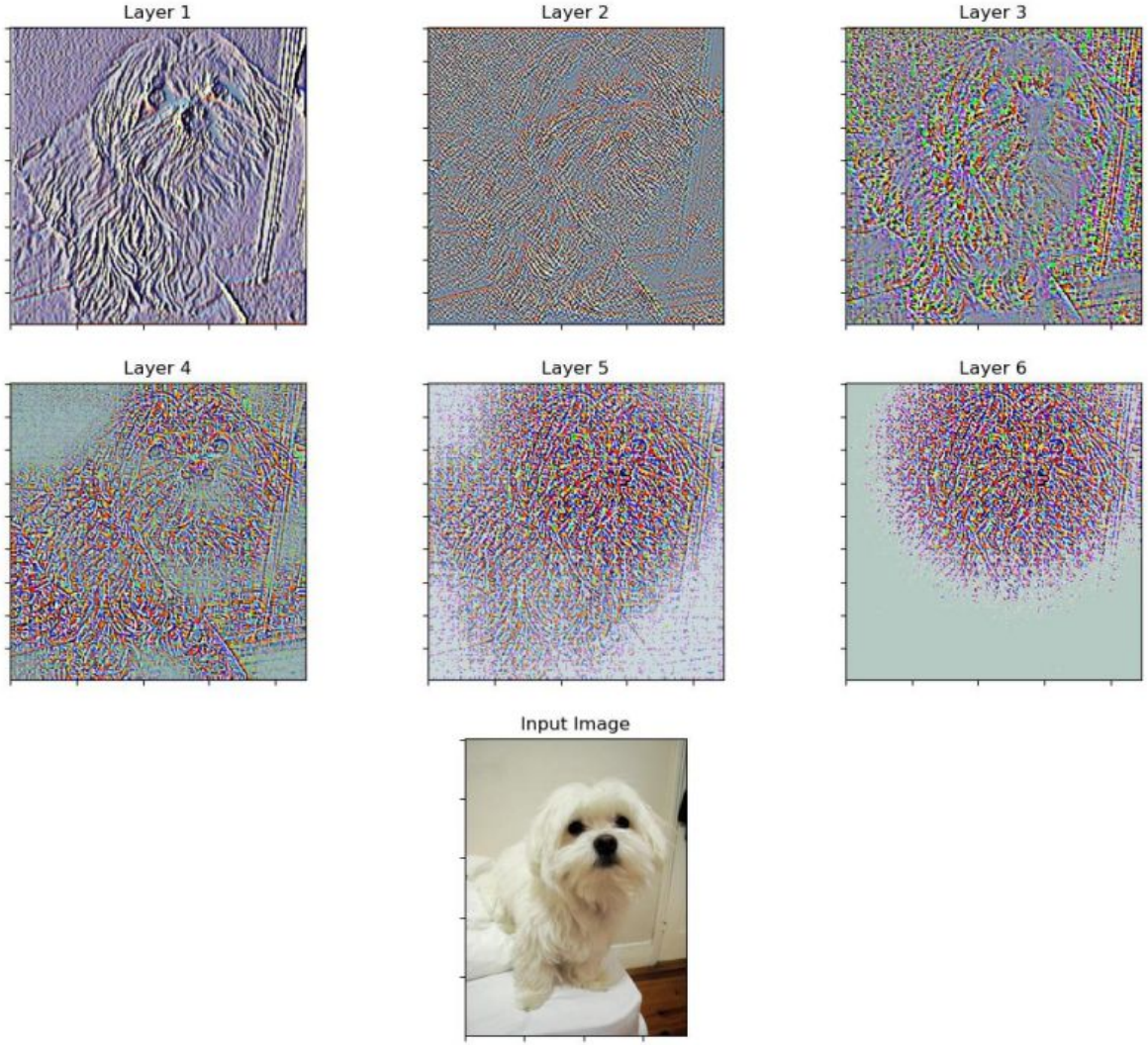
As we can see that the initial layers are basically just edge detection and the later layers focus more on more complex features.

Like spots in layer-4.



In this case we can see the model struggles to comprehend the whole image and the spiral cactus, especially in the earlier layers.

However, in the later layers, it can visualize the more complex geometry and can identify the whole spiral.



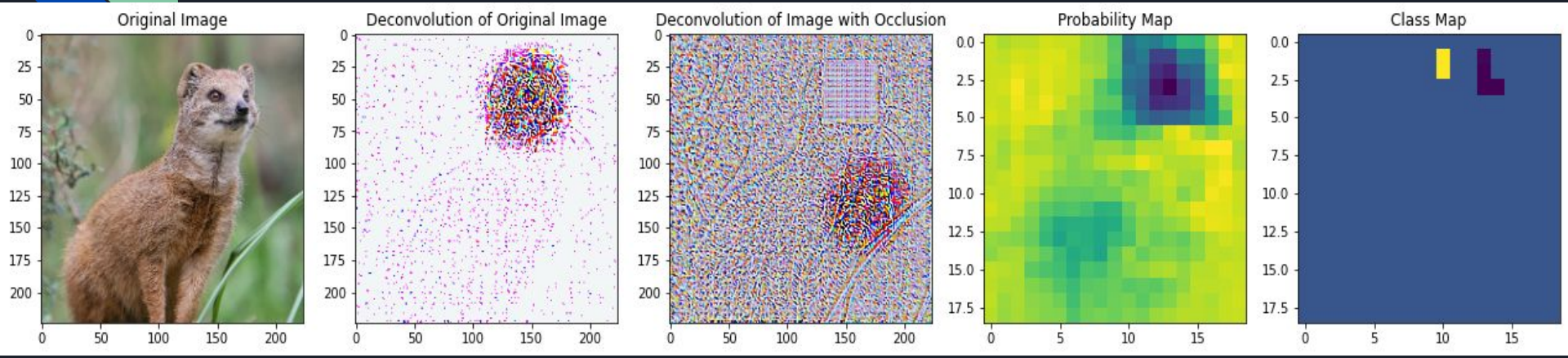
In this case we can see that as we go to higher layers, it focuses more on the face of the dog and less on the whole image. Indicating that the model slowly learns the important features from the image (in this case face).

Occlusion Sensitivity

In this we occlude different parts of the image with a grey square and see how the output of the classifier changes. This also helps us answer the question whether the model is actually identifying the object in any location or just using surrounding context to make its prediction.

We observed that the prediction becomes worse when we cover the most prominent parts of the image (Ex - the face in case of a dog). This means we can conclude that the model is actually learning about the features and using them to make predictions and not using just the surrounding context.

Occlusion

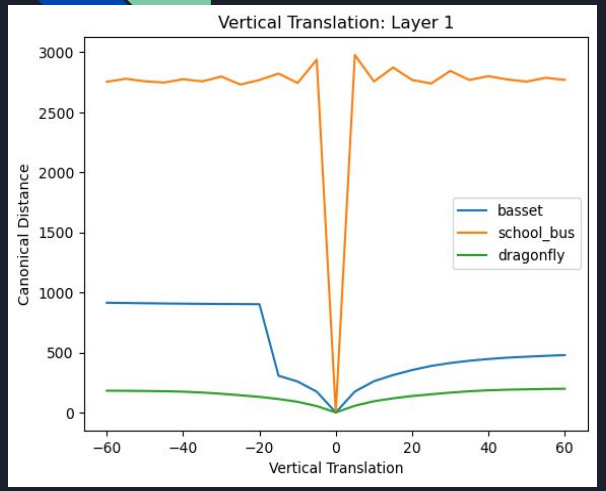


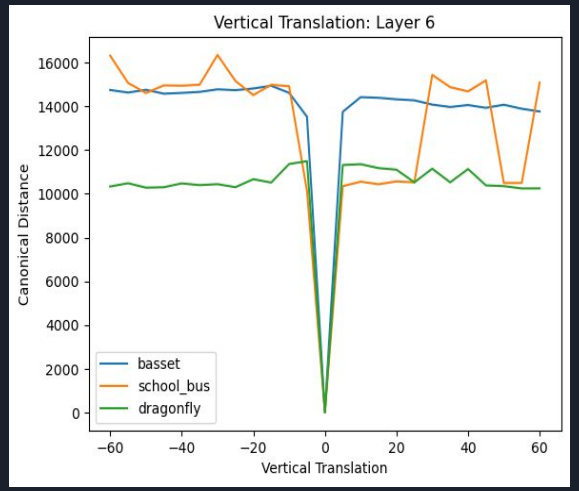
As we can see here, when the image is occluded with a grey square, the model fails to identify it correctly if if is occluded near its face, indicating that the model is actually learning about the Mongoose's face and not merely classifying it on the basis of the background context.

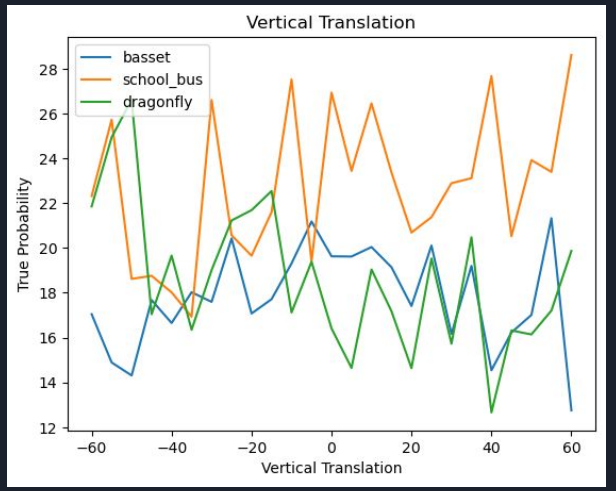
Feature Invariance

We chose some sample images and rotated, translated, and scaled them and looked at the changes in the feature vectors. The feature vectors were drastically different for the first few layers, however they were similar in the later layers. Implying that the model is stable to translation and scaling. However, the model is not so stable with rotation.

Vertical Translation

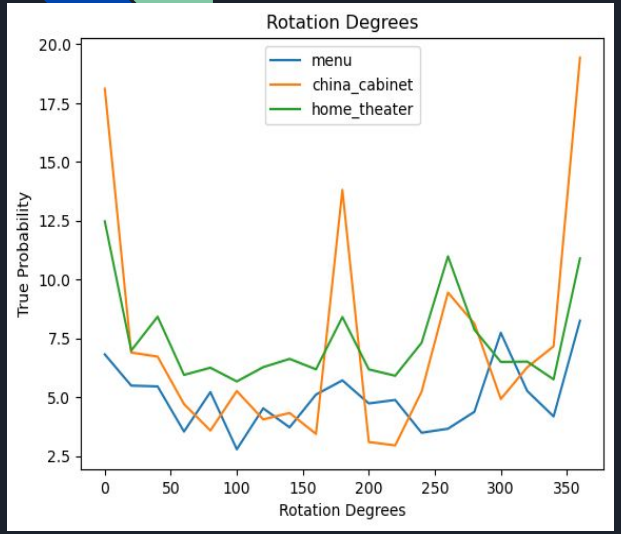


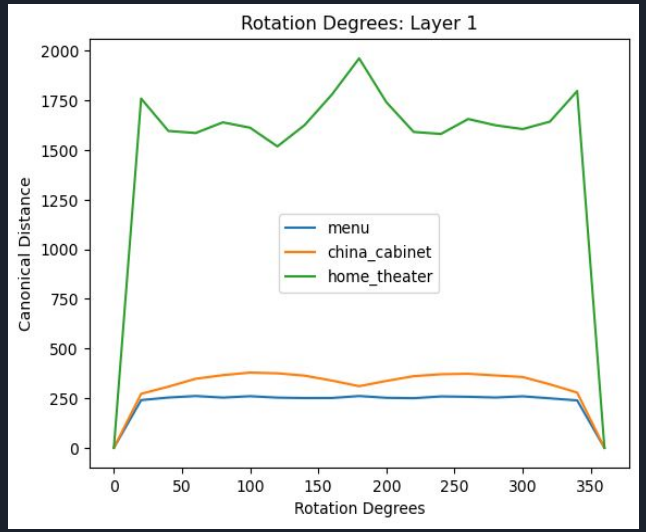


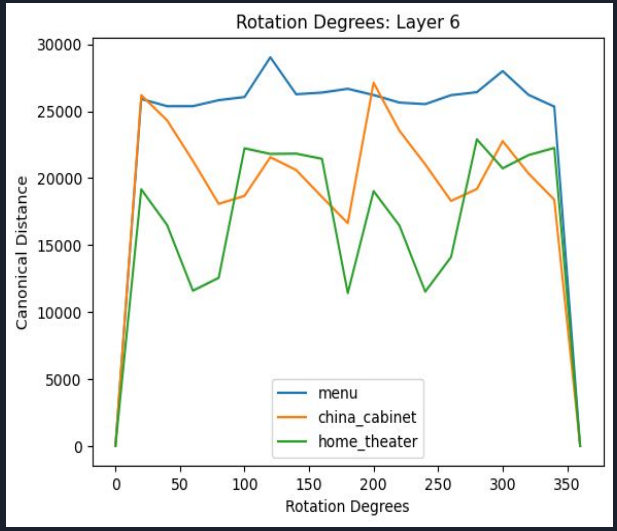


As we can see the model is more or less stable with respect to vertical translation. This shows that the model is actually learning features and making predictions based off of these features rather than just from background context.

Rotation

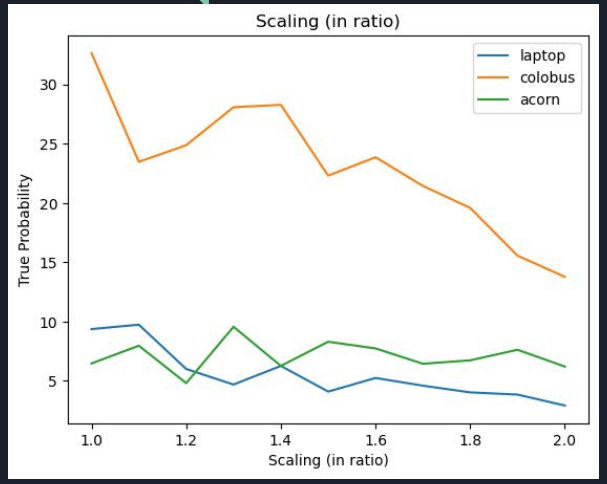


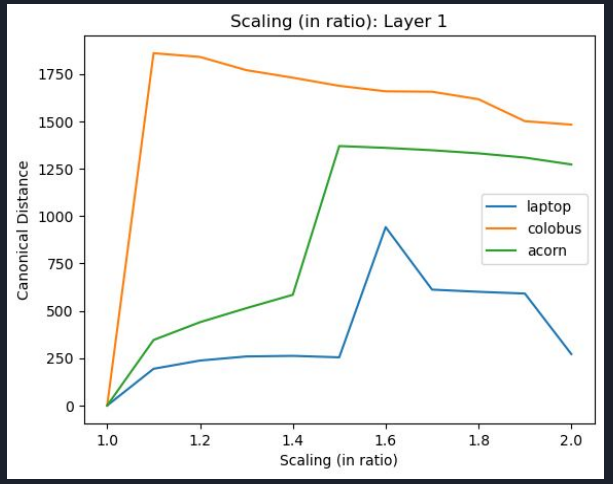


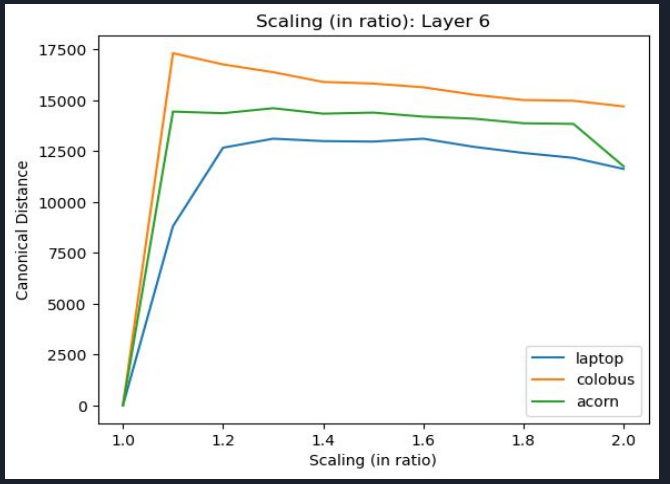


As we can see that the model is not stable with respect to rotation. This is because the features are dependent on rotation and the filters rely heavily on the fact that the images must be correctly oriented.

Scaling







The model is relatively stable with respect to scaling. This means that the model is invariant to changes in the size of the features.

Contributions

Adhiraj: Model Implementation and Training

Shikhar: Experiments and Report

Shreya: Model Implementation and Training

Shreyansh: Experiments and Report