Note : The code for this project can be found here.

Introduction

Deep Convolutional Neural Networks (CNNs) has been one the most successful and versatile class of networks that has successfully been applied to a variety of different Computer Vision tasks ever since it came into prominence due to its performance in Imagenet challenge(2012). But very little is understood what happens behind the hood of the models. This is a significant challenge in Research. Strong insights in our understanding of CNN's can lead to development of even stronger models and in this project we perform a series of experiments with the aim of gaining deeper insight into the working of the CNNs. We leverage Deconvolutional Networks the perform visualizations of the model's feature map and activations in the image space.

DeConvolution Networks

For this experiment, we visualize activation maps using DeconvNets for certain input images of the convolutional Layers of a VGG16 model trained on ImageNet.

The DeconvNets attempt to transform the feature maps back into the pixel space for purposes of visualization.

The DeconvNets are built by essentially building the original CNN in reverse, using DeConv2d layers in place of Conv2d layers, MaxUnpool layers in place of MaxPool layers and keeping ReLU unchanged.

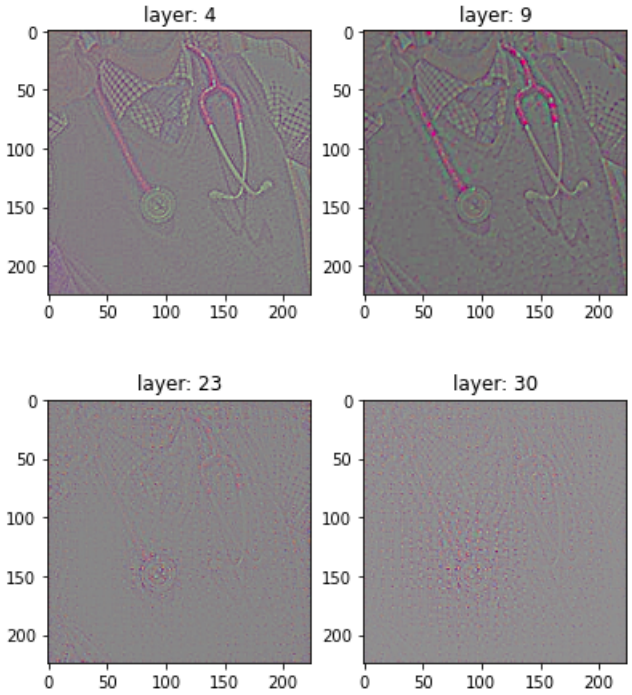
Convolution: The convolution is inverted using convolving the feature map with the transpose of the forward kernel that was used to generate the feature map.

ReLU: The Inverse of ReLU is ReLU itself

Max Pooling : Out of the 3 Max pooling is the most complex.Its impossible to fully invert the output of the Max pooling layer,but partial reconstruction is possible by using switch to remember the location(indices) of the cells which the max pool feature map got the values.

For the input image:

The following activation maps were observed:



Observations:

As the layers increase you can see the model focus more narrowly.

The initial layer retains most of the features of the original image,whereas the later layer loses most of the details.

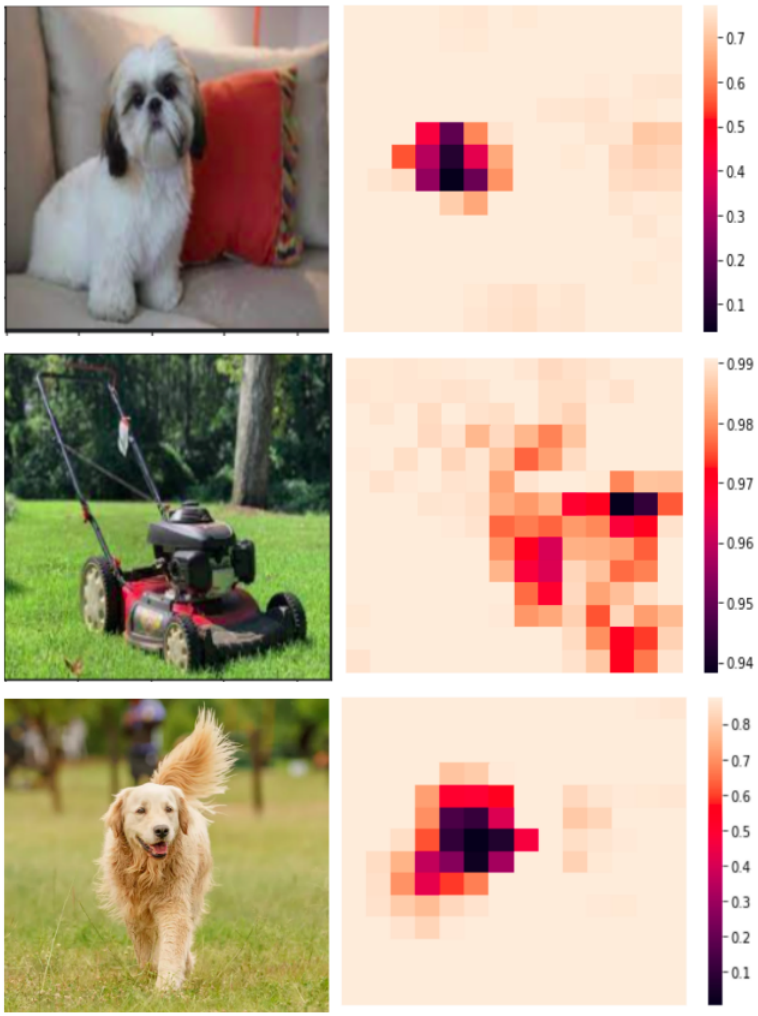
Occlusion Sensitivity

When dealing with image classification, An important question that arises is the importance that one part of the image has on its classification.

We are able to obtain useful information from this by systematically occluding or covering a portion of the image with a gray value, and calculating the confidence of the model on the resulting image with that label. The value in the heatmap, represents the confidence of the detection when that part of the image is occluded.

This analysis was made on VGG16 trained on imagenet.

Results:



These examples clearly show the model is localizing the objects within the scene, as the probability of the correct class drops significantly when the object, or parts of it are occluded.

Correspondence Analysis

Deep Models like convolutional neural networks are opaque in the sense it is unclear how or what exact features that they rely on to make their decisions. Do CNN's focus on the eyes or the nose while trying to detect faces? Since we are not explicitly inserting them in our design of ML model, so are these objects parts being implicitly learned? This is one of the questions that correspondence analysis aims to answer. To answer this we designed experiments that establishes correspondence between specific parts of the image and the resulting feature vector. One of the ways we do this is by seeing the changes in the features responses when certain objects or object parts within an image are occluded.

we took 5 random dog images with frontal pose and systematically masked out the same part of the face in each image. For the first experiment we occluded the right eye in all the dogs. Then for each image i we calculate
$\epsilon_i^l = x_i^l - \hat{x}_i^l$
where x and $\hat{x}$ are feature vectors at layer l for the original and occluded images respectively. We then measure the consistency of this difference ϵ vector between all related image pairs $(i, j) : \Delta_l = \sum_{i,j=1,i\neq j}^{5} H(sign(\epsilon_i^l), sign(\epsilon_j^l))$ where H is Hamming distance.

A lower value indicates greater consistency in the change resulting from the occlusion operation. This means there is high correspondence between the same object parts in different images. In other words blocking same parts in different images changes feature response of the CNN in a similar way indicating some implicit defination of the part. we compare the Δ score for two parts of the face (left eye, right eye) to random parts of the object (occluding random parts in different images).The lower score for these parts, relative to random object regions shows that the model does establish some degree of correspondence.

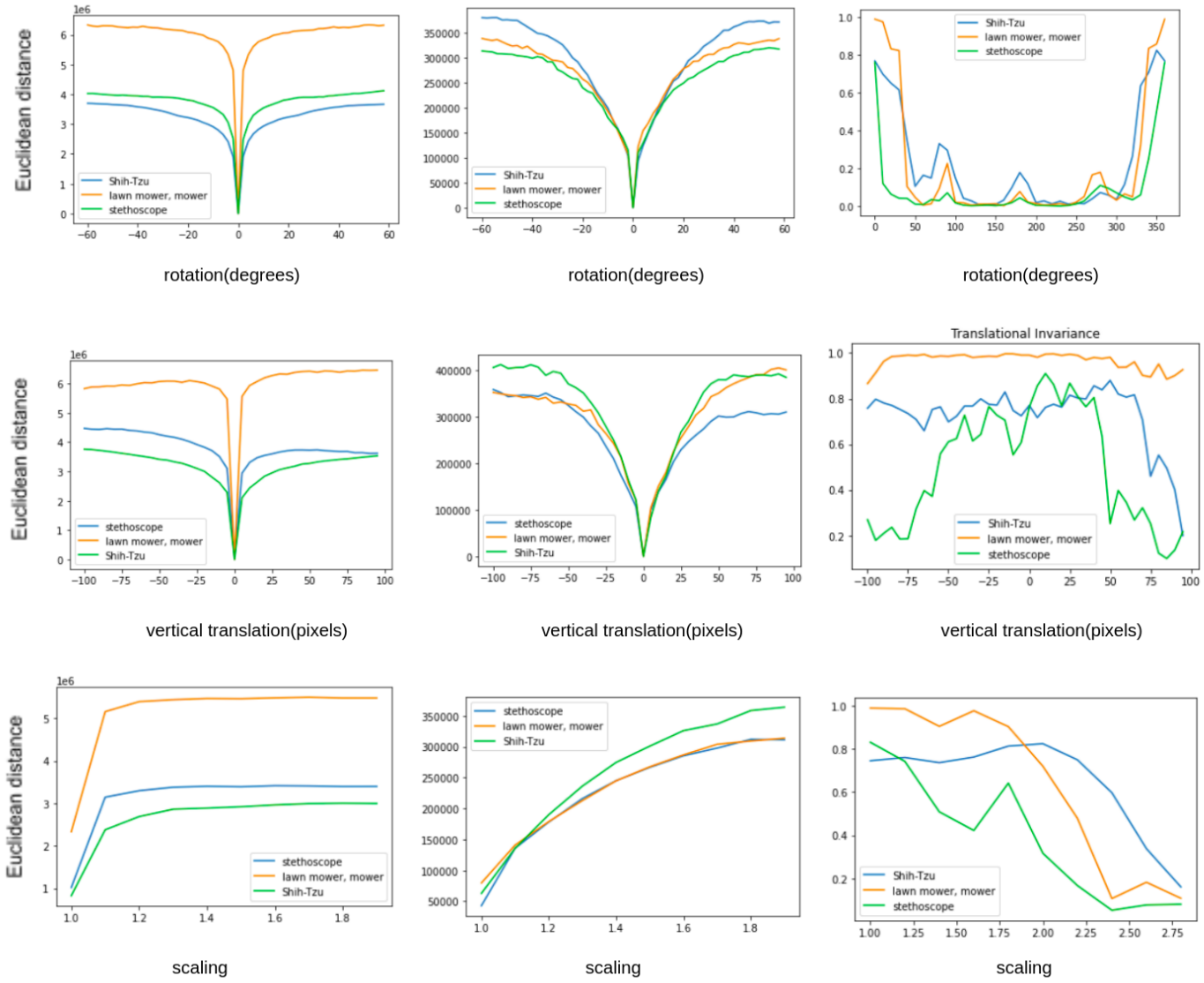
Occlusion Location	Δ value
Right Eye	0.4990
Left Eye	0.5021
Random	1.0056

Feature Invariance

An ideal classifier would be invariant to transformations like scaling, transformation or rotation. small transformations have a dramatic effect in the first layer of the model, but a lesser impact at the top feature layer, being quasi-linear for translation scaling.

The below figure (in columns) The euclidean distance between the feature vectors of the original and transformed in the image in the 2nd and 27th layer, and then the accuracy. (in columns): Rotation, translation, scaling. The network seems to me more invariant towards scaling and translation and scaling than rotation.

This analysis was made on VGG16 trained on imagenet.



Architecture Selection

Tuning parameters of a CNN's architecture like the sizes of filters, padding sizes, strides, etc. can be quite a difficult task.

We can use visualizations of the activations in each layer to get a better understanding of what exactly a layer "sees" in the image and use that understanding to tune the model's architecture to improve the accuracy of a model.

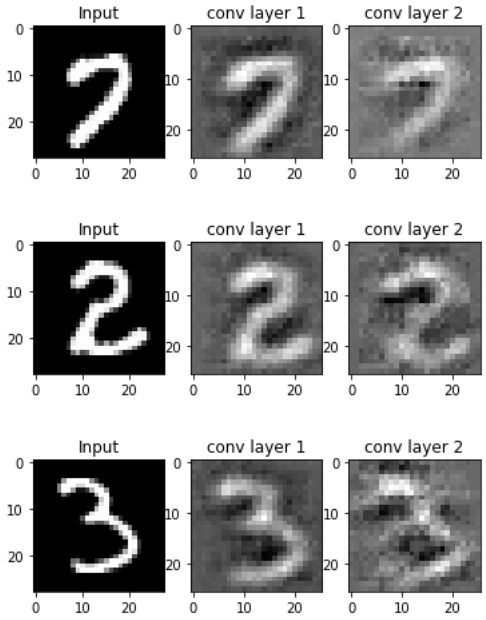
For this experiment, we first define a CNN model and train it on the MNIST dataset.

The model was as follows:

```
CNN1(
  (features): Sequential(
    (0): Conv2d(1, 16, kernel_size=(10, 10), stride=(4, 4), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(16, 32, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Linear(in_features=32, out_features=10, bias=True)
)
```

After training for 5 epochs, we get a test accuracy of about 97.2 %

Some of the visualizations from this model are shown below:



Similar to the original paper, we face a problem of aliasing artifacts.

- Prominent aliasing artifacts are visible in the features of both convolution layers, especially the second layer.

- Ideally, we would want that the feature activations are strong in the region of the hand-written digits' strokes and very weak in the region outside of the stroke with the differences between these two being high. However we see that there is a clear aliasing effect giving the impression of softer boundaries.

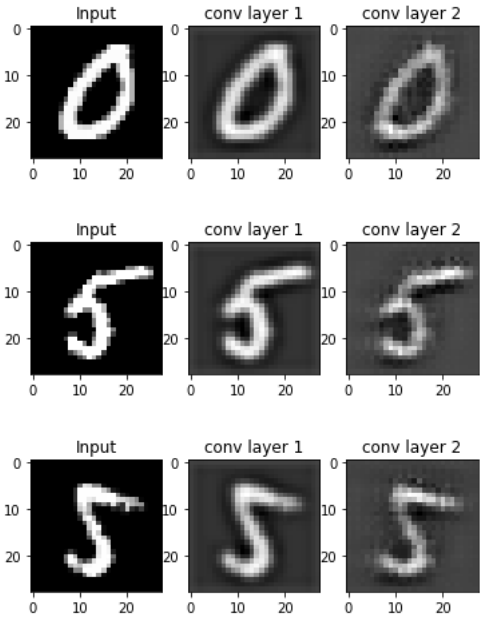
- This is caused by large strides and we attempt to fix that by reducing the strides in the improved model. We also reduce kernel size.

We define an improved CNN Model taking this into consideration.

```
CNN2(
  (features): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Linear(in_features=1568, out_features=10, bias=True)
)
```

After training for 5 epochs, we get a test accuracy of about 98.7 % which is an improvement over the original model.

Some of the visualizations from this model are shown below:



We see a reduction in the aliasing effect in both layers with the boundary between the stroke and background being much less smoother.

Also, we see an improvement in the model accuracy which confirms that our tuning of architecture parameters was a step in the right direction.

Feature Evolution

The aim of feature evolution is to visualize the different feature maps across training and understand how it evolves across training. Each Layer of the network outputs N feature maps depending on the architecture.For any layer k,which outputs a N feature maps,s feature maps are randomly chosen. So for any epoch for which we want to visualize the feature maps of

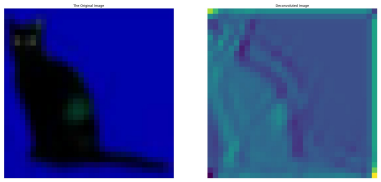
layer k,for all the s randomly chosen feature map of layer k,we show the highest the one with the activation across all the training examples. As the model learns we would then be able to visualize which training example most excites the model,and what part of the image the model focuses on.

The Dataset used is CIFAR-10. As one can see from the images, for any activation layer, the training image which excites it the most often changes accross training.The patterns of images are more visible in the lower layers while at the higher layers the pattern becomes less discernable.

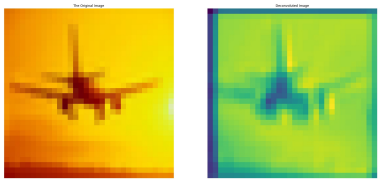
The Architecture of the Model:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 32, 32]             448
              ReLU-2           [-1, 16, 32, 32]               0
         MaxPool2d-3  [[-1, 16, 16, 16], [-1, 16, 16, 16]]
            Conv2d-4           [-1, 32, 16, 16]           4,640
              ReLU-5           [-1, 32, 16, 16]               0
         MaxPool2d-6  [[-1, 32, 8, 8], [-1, 32, 8, 8]]
            Conv2d-7             [-1, 32, 8, 8]           9,248
              ReLU-8             [-1, 32, 8, 8]               0
            Linear-9                  [-1, 128]         262,272
         Dropout-10                   [-1, 128]               0
           Linear-11                    [-1, 10]           1,290
================================================================
Total params: 277,898
Trainable params: 277,898
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 159.59
Params size (MB): 1.06
Estimated Total Size (MB): 160.66
----------------------------------------------------------------
```

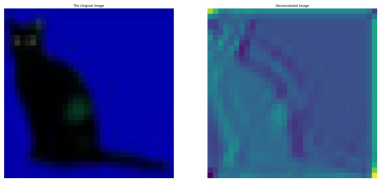
The Deconvoluted Image along with original Image for epoch 1,layer 1 and activation index 10



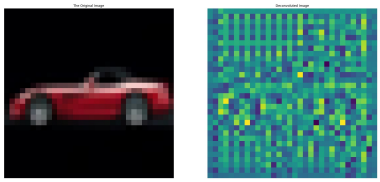
The Deconvoluted Image along with original Image for epoch 1,layer 2 and activation index 3



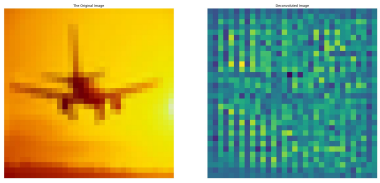
The Deconvoluted Image along with original Image for epoch 1,layer 2 and activation index 10



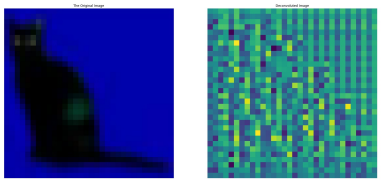
The Deconvoluted Image along with original Image for epoch 1,layer 4 and activation index 3



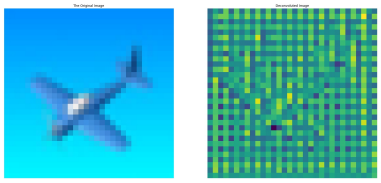
The Deconvoluted Image along with original Image for epoch 1,layer 4 and activation index 10



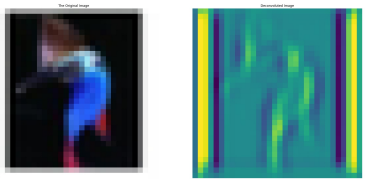
The Deconvoluted Image along with original Image for epoch 1,layer 5 and activation index 3



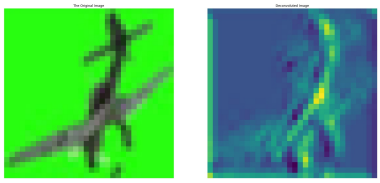
The Deconvoluted Image along with original Image for epoch 1,layer 5 and activation index 10



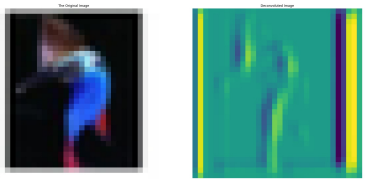
The Deconvoluted Image along with original Image for epoch 6,layer 1 and activation index 3



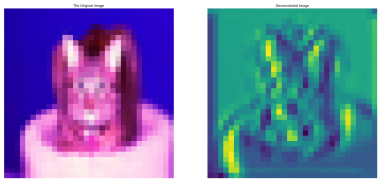
The Deconvoluted Image along with original Image for epoch 6,layer 1 and activation index 10



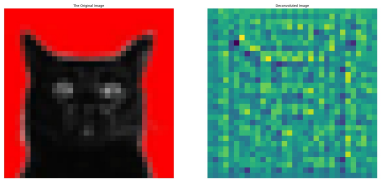
The Deconvoluted Image along with original Image for epoch 6,layer 2 and activation index 3



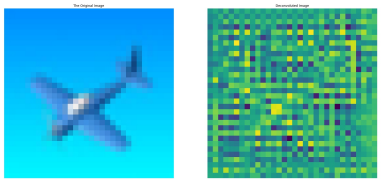
The Deconvoluted Image along with original Image for epoch 6,layer 2 and activation index 10



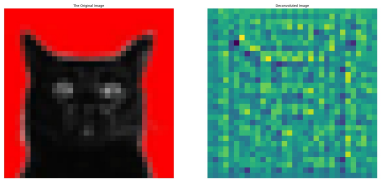
The Deconvoluted Image along with original Image for epoch 6,layer 4 and activation index 3



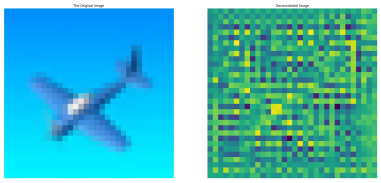
The Deconvoluted Image along with original Image for epoch 6,layer 4 and activation index 10



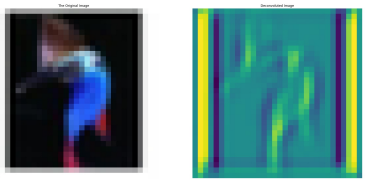
The Deconvoluted Image along with original Image for epoch 6,layer 5 and activation index 3



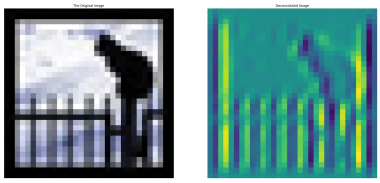
The Deconvoluted Image along with original Image for epoch 6,layer 5 and activation index 10



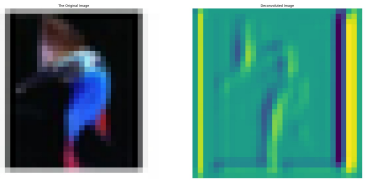
The Deconvoluted Image along with original Image for epoch 16, layer 1 and activation index 3



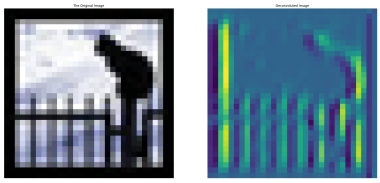
The Deconvoluted Image along with original Image for epoch 16, layer 1 and activation index 10



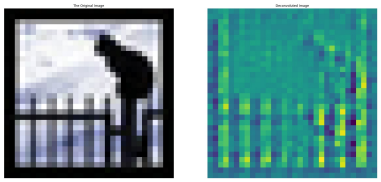
The Deconvoluted Image along with original Image for epoch 16, layer 2 and activation index 3



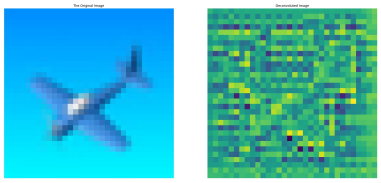
The Deconvoluted Image along with original Image for epoch 16,layer 2 and activation index 10



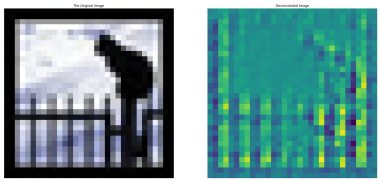
The Deconvoluted Image along with original Image for epoch 16,layer 4 and activation index 3



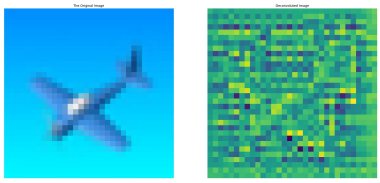
The Deconvoluted Image along with original Image for epoch 16,layer 4 and activation index 10



The Deconvoluted Image along with original Image for epoch 16,layer 5 and activation index 3



The Deconvoluted Image along with original Image for epoch 16,layer 5 and activation index 10



Division of Work

Boilerplate code - Saravanan Senthil and Abhijit Manatkar
Feature evolution - Tathagato Roy and Kushal Jain
Occlusion Sensitivity - Saravanan Senthil
Correspondence Analysis - Kushal Jain and Tathagato Roy
Feature Invariance - Saravanan Senthil
Architecture Selection - Abhijit Manatkar