SMAI Project Mid-Eval Report

Team Name : cassata

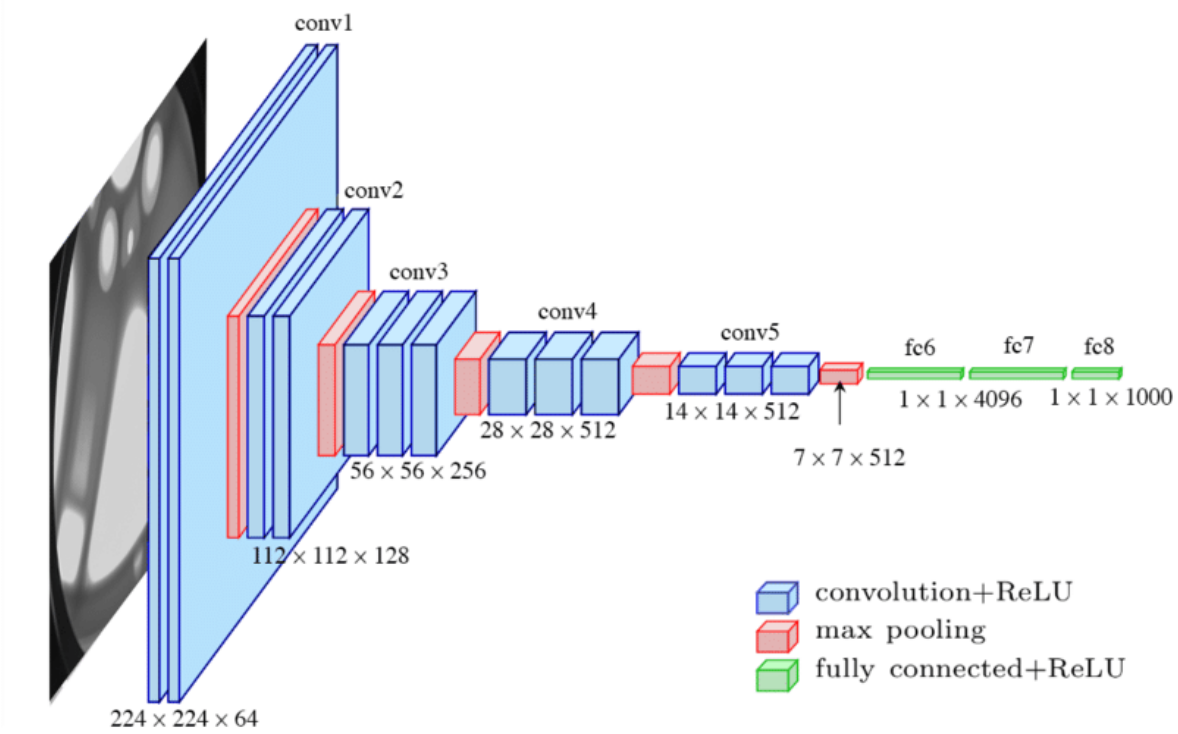
Team No : 1

Team members:

- 2019101108 Abhijit Manatkar
- 2019111001 Kushal Jain
- 2019101016 Saravanan Senthil
- 2019111020 Tathagatho Roy

We decided to use the VGG network to visualise the inner workings of CNN's. While other iterations of AlexNet (previous best method) focused on smaller window sizes and strides in the first convolutional layer, VGG addresses another very important aspect of CNNs: depth. Architecture of

VGG :



The training images are passed through a stack of convolution layers. There are a total of 13 convolutional layers and 3 fully connected layers in VGG16 architecture. VGG has smaller filters (3*3) with more depth instead of having large filters. The network consists of two Fully Connected layers with 4096 channels each which is usually followed by another fully connected layer with 1000 channels to predict 1000 labels but we downscaled it to 10 for our CIFAR10 dataset.
We trained it only for 8 epochs (due to hardware issues) but got a decent accuracy on the test set of 77.6%.

VGG architecture in code :

```
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1)
        self.conv1_2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)

        self.conv2_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.conv2_2 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)

        self.conv3_1 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)
        self.conv3_2 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
        self.conv3_3 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)

        self.conv4_1 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=3, padding=1)
        self.conv4_2 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.conv4_3 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)

        self.conv5_1 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.conv5_2 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)
        self.conv5_3 = nn.Conv2d(in_channels=512, out_channels=512, kernel_size=3, padding=1)

        self.maxpool = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(25088, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, 10)

    def forward(self, x):
        x = F.relu(self.conv1_1(x))
        x = F.relu(self.conv1_2(x))
        x = self.maxpool(x)
        x = F.relu(self.conv2_1(x))
        x = F.relu(self.conv2_2(x))
        x = self.maxpool(x)
        x = F.relu(self.conv3_1(x))
        x = F.relu(self.conv3_2(x))
        x = F.relu(self.conv3_3(x))
        x = self.maxpool(x)
        x = F.relu(self.conv4_1(x))
        x = F.relu(self.conv4_2(x))
        x = F.relu(self.conv4_3(x))
        x = self.maxpool(x)
        x = F.relu(self.conv5_1(x))
        x = F.relu(self.conv5_2(x))
        x = F.relu(self.conv5_3(x))
        x = self.maxpool(x)
        x = x.reshape(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, 0.5) #dropout was included to combat overfitting
        x = F.relu(self.fc2(x))
        x = F.dropout(x, 0.5)
        x = self.fc3(x)
        return x
```

Feature Evolution:

The aim of feature evolution is to visualize the different feature maps across training and understand how it evolves across epochs.

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 focusses on.

The initial plan for this half was to prototype the code,to see its correctness and understand it logic ,then extend it to a larger model and dataset perform the full visualization of the feature evolution .

But for purposes of the prototyping we have chosen a smaller model and the dataset MNIST.
Here is the architecture of the smaller model.

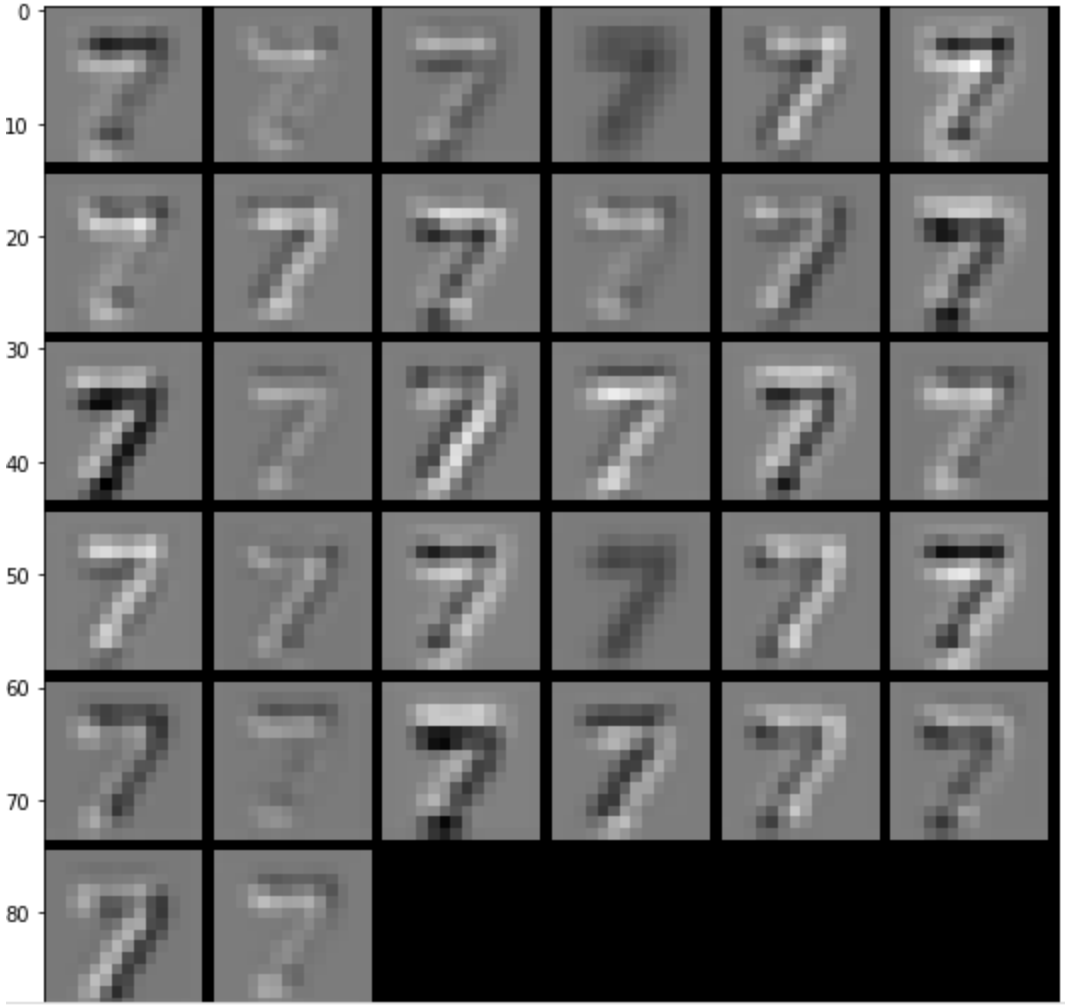
```
    (features): Sequential(
      (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU()
      (2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU()
      (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
      (5): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (6): ReLU()
      (7): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (8): ReLU()
      (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (classifier): Sequential(
      (0): Linear(in_features=1568, out_features=512, bias=True)
      (1): ReLU()
      (2): Linear(in_features=512, out_features=10, bias=True)
    )
  )
```

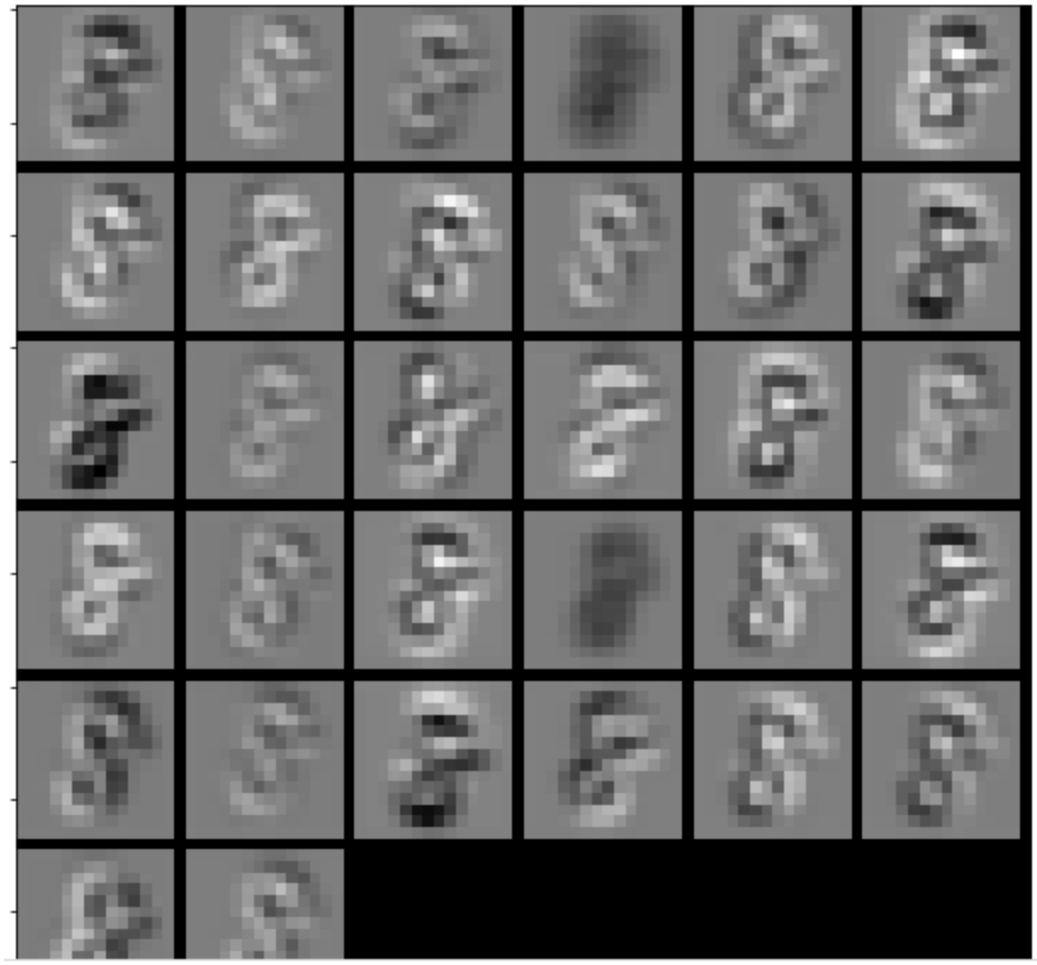
And the corresponding Deconvolutional Network :

```
(features): Sequential(
  (0): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
  (1): ReLU()
  (2): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU()
  (4): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (5): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
  (6): ReLU()
  (7): ConvTranspose2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU()
  (9): ConvTranspose2d(32, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
)
```

As one can see,this a very simple CNN with 2 Blocks each with two Convolutional layer , 2 ReLU layer and 1 Max pooling layer.
This 2 Convolutional block is followed by two fully connected Linear layer over which a softmax layer is used for classification

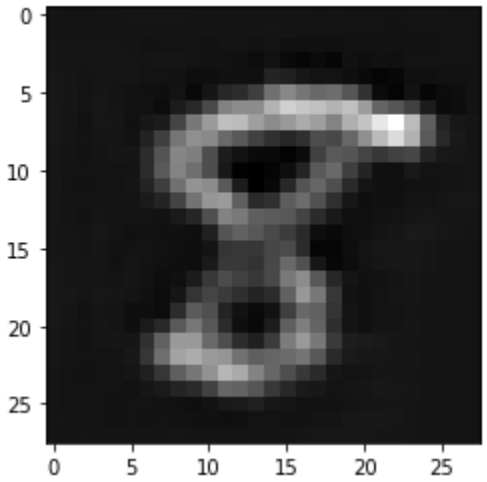
This are 3 random visualization of the feature maps for a random subset of images in Layer 1.

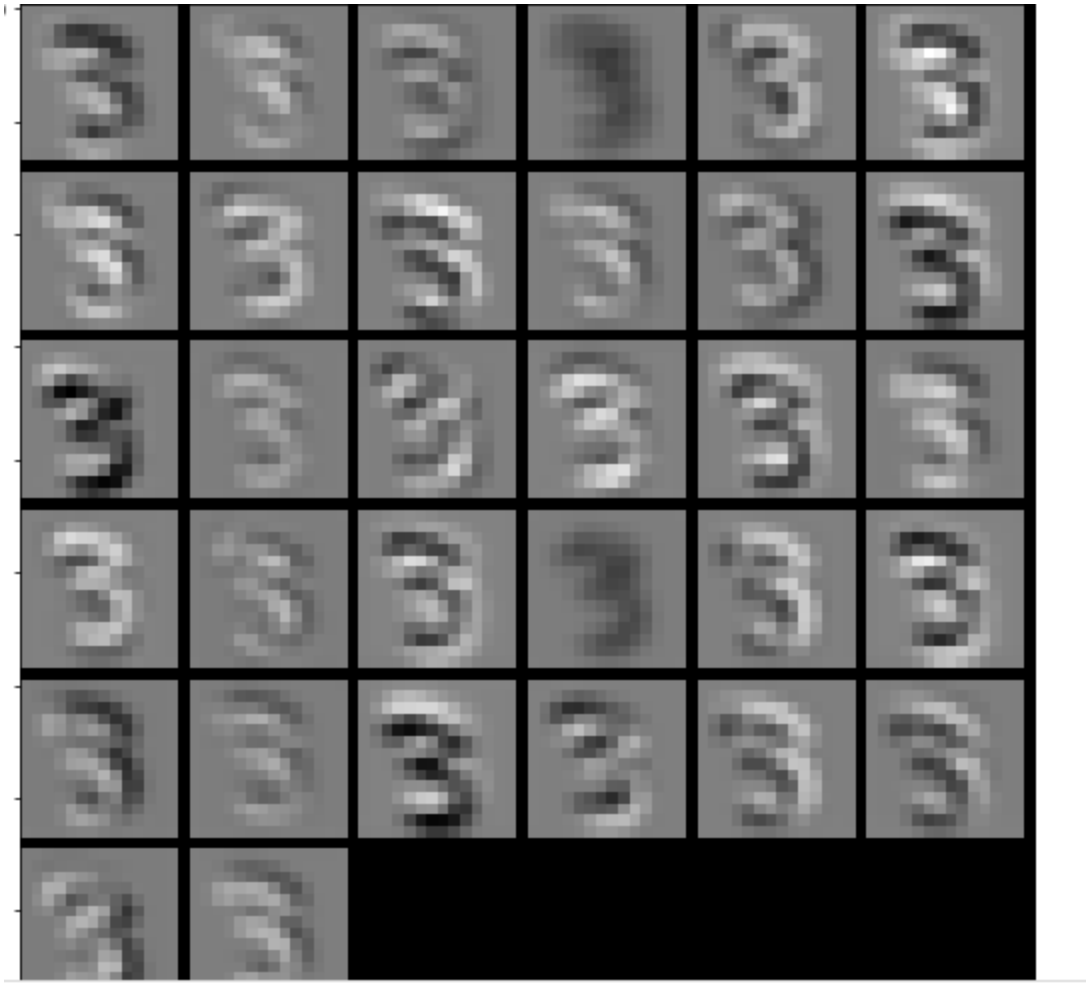

0
10
20
30
40
50
60
70
80



This is the output when the above image is deconvolutated through the Deconv Network.

```
<matplotlib.image.AxesImage at 0x7fba539fd99
```





Unfortunately we have faced an undiagnosed error which is causing the Collab runtime to crash everytime we are running the last cell in the attached ipynb file called “Feature Evolution”.
We were unable to fix that by deadline and hence couldn’t submit a more complete and demonstrative outputs for our work.
Hopefully this would be fixed soon enough and we would be able to fully demonstrate feature evolution.

Boilerplate code for the project:

Construction of Deconvolution Network :
The Deconv Network attempts to transform the feature maps back into the pixel space for purposes of visualization.

The Deconv Network attempts to invert the operations like Convolutions,ReLu and Max Pooling.

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:

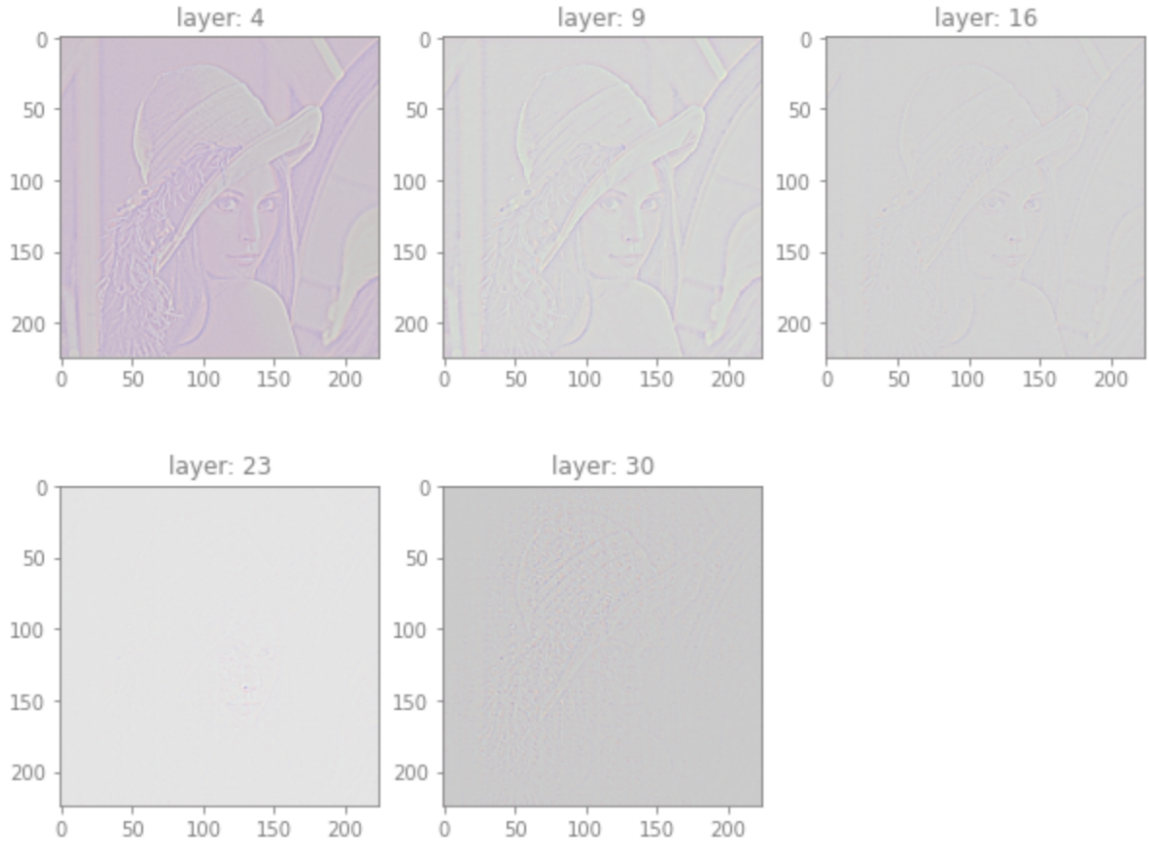
The Inverse of ReLU is ReLU itself

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.

The boiler plate is the construction of a Deconv net for VGG16 model.

Result from sample image and pretrained vgg16 model:



The code generates a model to deconvolute images, and output the images at every max pooling layer.

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.

Next Steps :

As our boiler plate code is mostly done,we plan to run the experiments in the next steps,and perform the necessary analysis.

We hope to do it for both pre trained and custom-trained model.