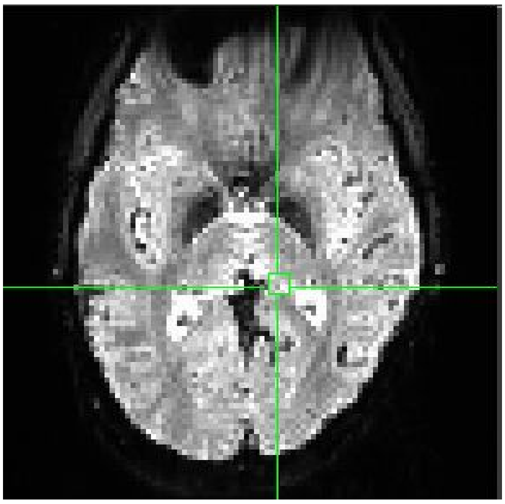
Submission for Final Project

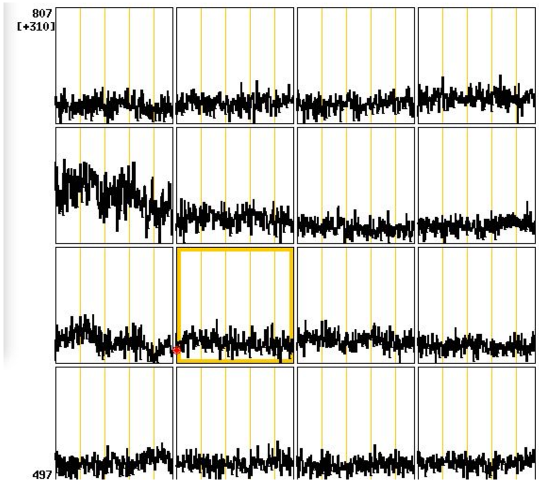
The aim of this project was to recreate the image as seen by the user using fMRI scans and BOLD signals. The submission of the project needed to include

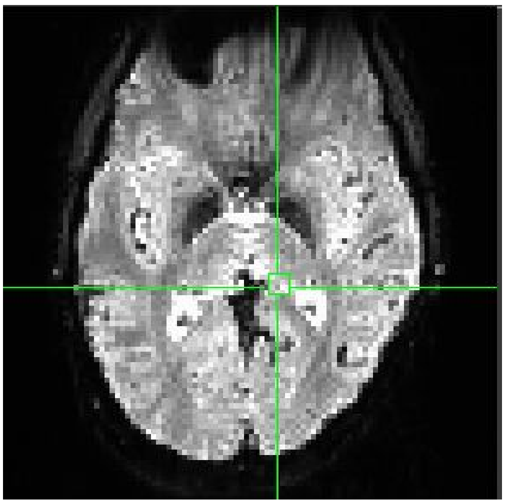
a) BOLD run and the series obtained
b) The model used
c) The images obtained (5 each for each category) and
d) The source code of the algorithm used.

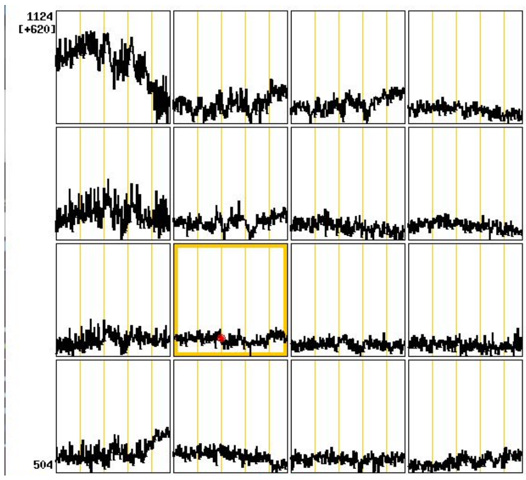
The submission is hereby divided in the aforementioned sections.

BOLD runs



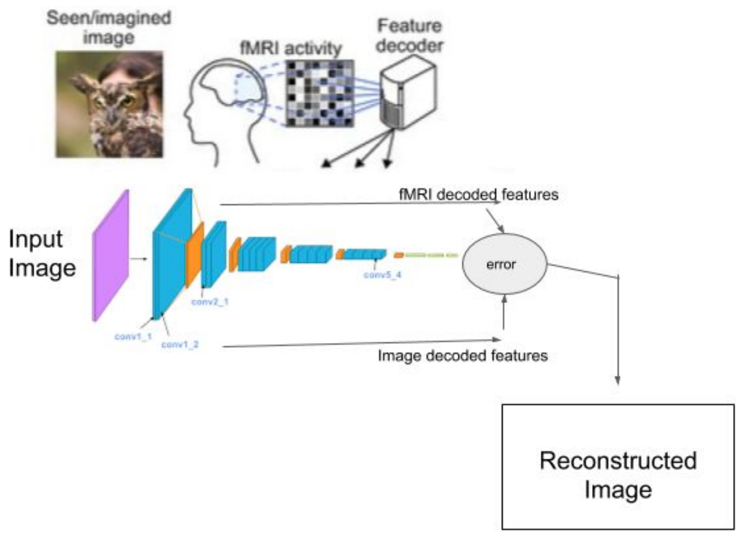






Model

We use a CNN model to replicate the visual cortex behavior. VGG 19 was used as our base CNN model upon which we further developed our algorithm. VGG 19 has been shown to have mimicked the activity of the visual cortex and for the same reason we choose this model. BOLD signal features were passed through the CNN to obtain a representation of the image. This was in turn compared with the feature maps obtained from the original stimulus image itself. Based on this, an error was calculated and the optimization was done on these features obtained from the CNN. These features were optimized using the scipy library's optimize function and an output was obtained. All outputs obtained were from the features obtained from the first convolution to the third convolution layer. This was done so because of the limitation of compute and data storage. The results obtained are shown in the subsequent sections.



Model Used

Generated Images

The different responses for the two subjects are shown in the following subsections for each of the three categories that exist i.e. Alphabets, Natural and Artificial responses.

Alphabets

The responses for both the subjects for alphabet stimulus are compared side by side.

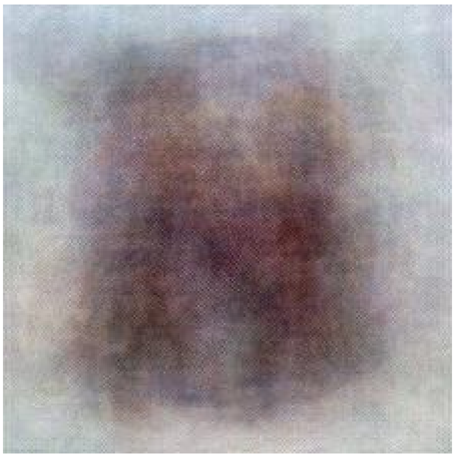


Fig 1.A Response
From first subject

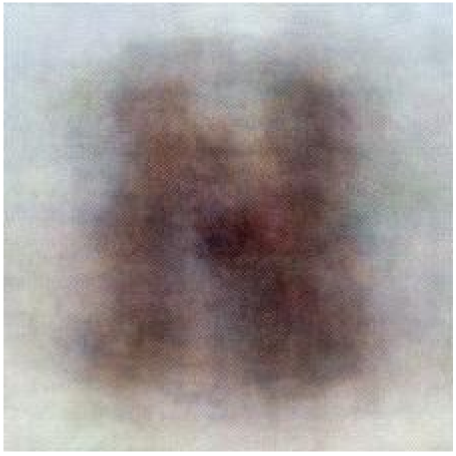


Fig 1.B Response
from second subject

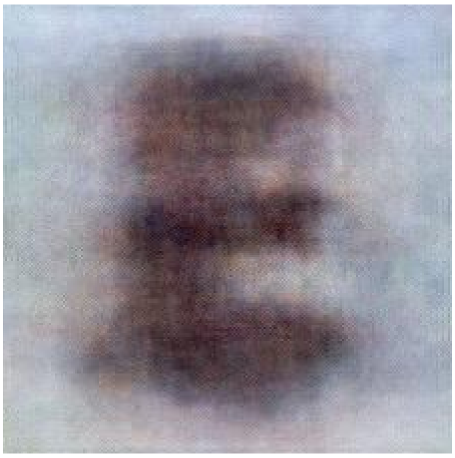


Fig 2.A Response
From first subject

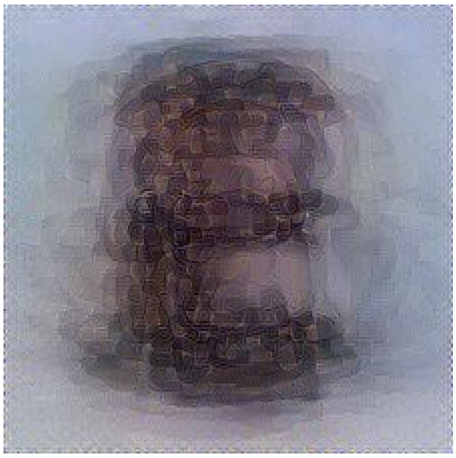


Fig 2.B Response
from second subject

Fig 3.A Response
From first subject

Fig 3.B Response
from second subject

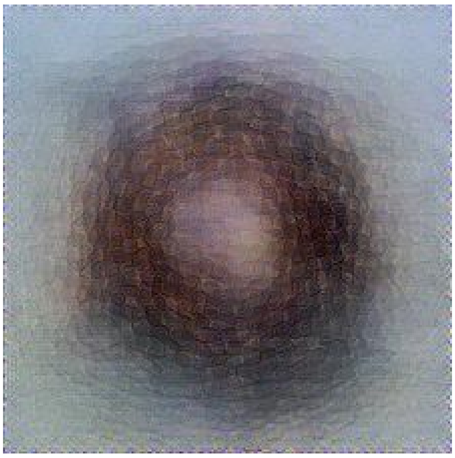


Fig 4.A Response
From first subject

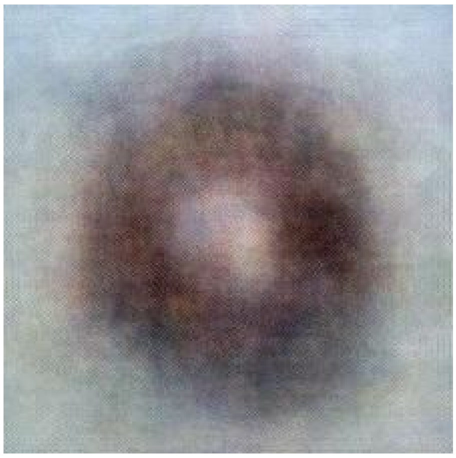


Fig 4.B Response
from second subject

Fig 5.A Response
From first subject

Fig 5.B Response
from second subject

Artificial

The responses for both the subjects for artificial stimulus are compared side by side.

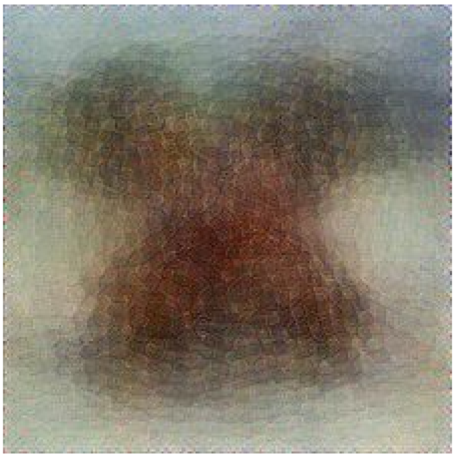


Fig 6.A Response
From first subject

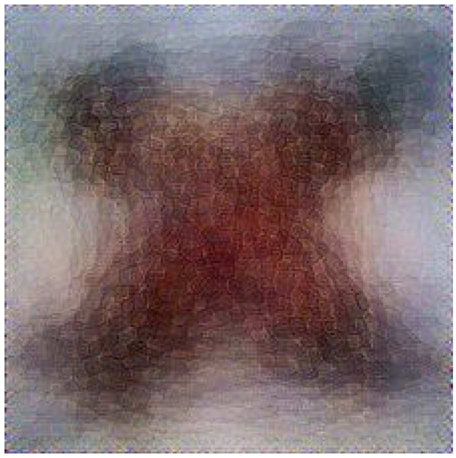


Fig 6.B Response
from second subject

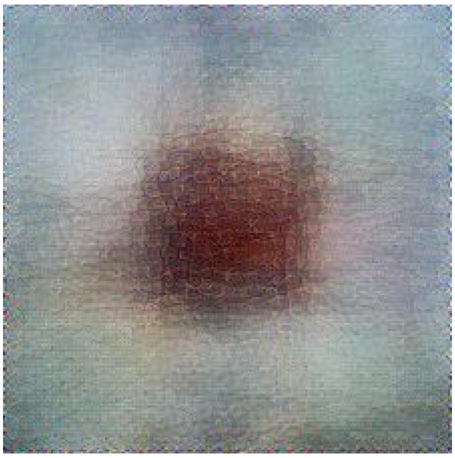


Fig 7.A Response
From first subject

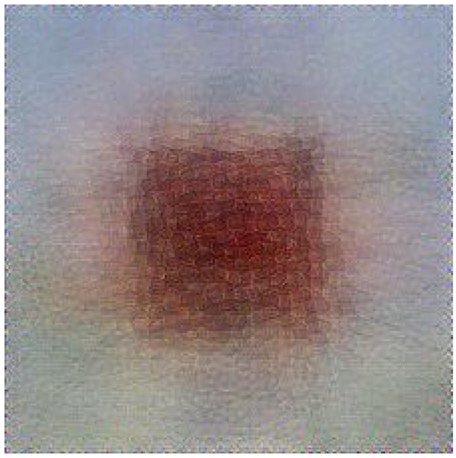


Fig 7.B Response
from second subject

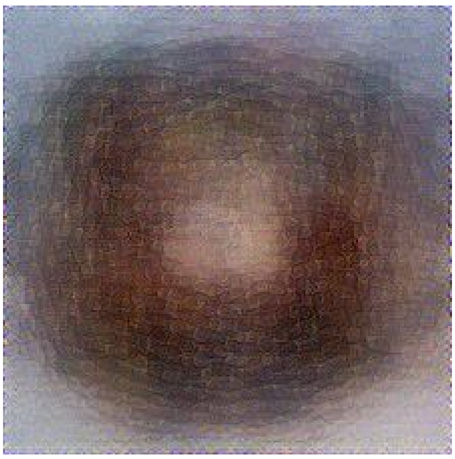


Fig 8.A Response
From first subject

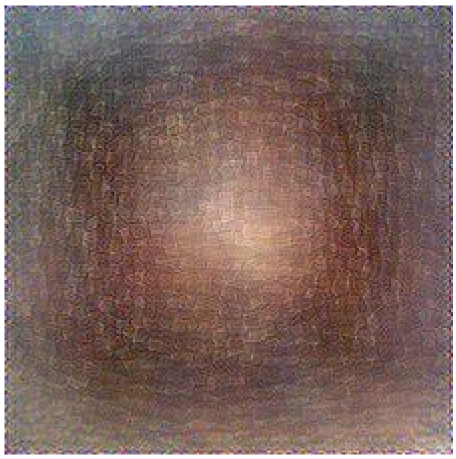


Fig 8.B Response
from second subject

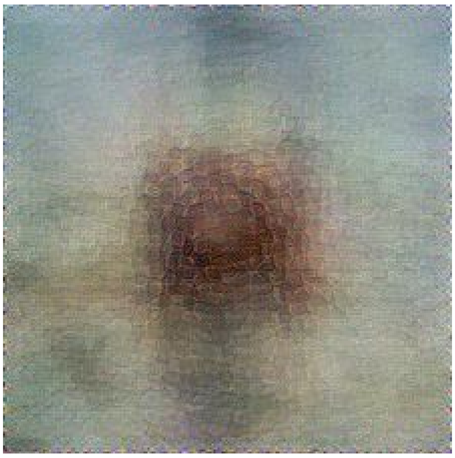


Fig 9.A Response
From first subject

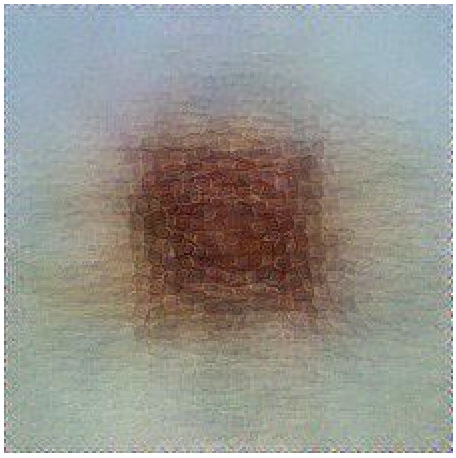


Fig 9.B Response
from second subject

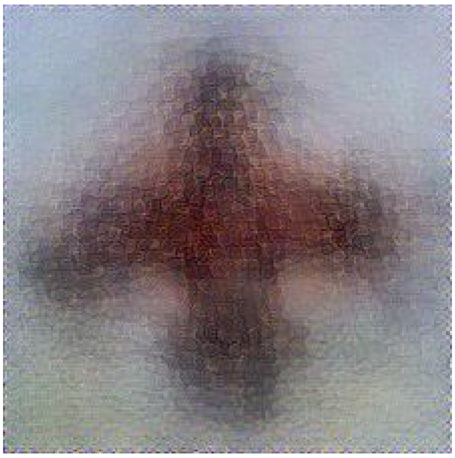


FIg 10.A Response
From first subject

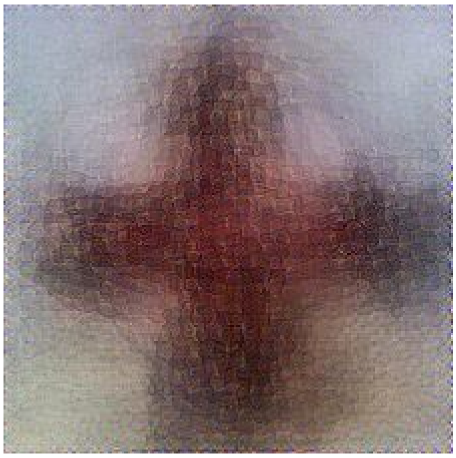
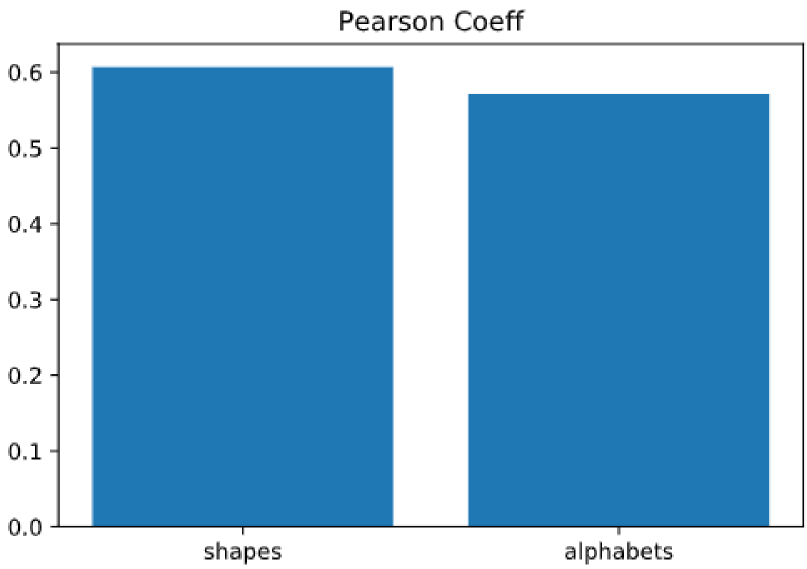


Fig 10.B Response
from second subject



Pearson Coefficients for alphabets and shapes

Natural

The responses for both the subjects for natural stimulus are compared side by side.

Fig 11.A Response
From first subject

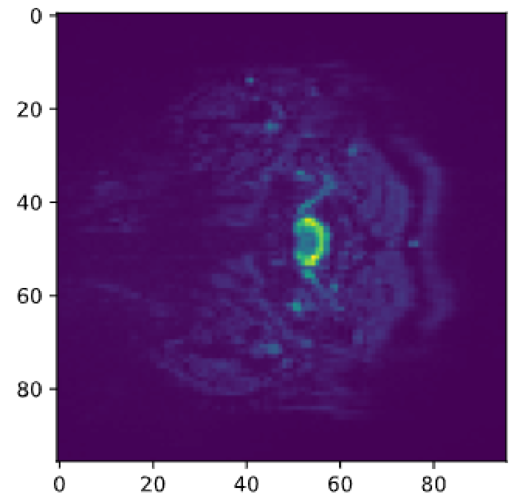


Fig 11.B BOLD info for
given response

Fig 12.A Response
from second subject

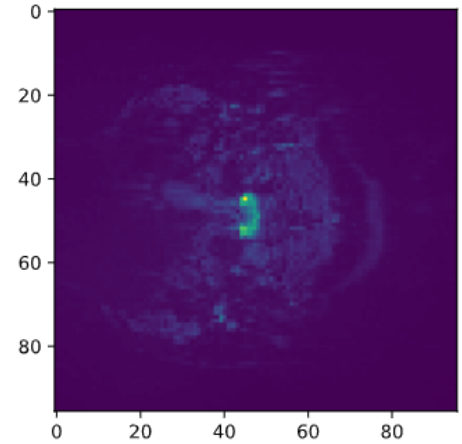


Fig 12.B BOLD info for
given response

Fig 13.A Response
From first subject

Fig 13.B Response
from second subject

Fig 14.A Response
From first subject

Fig 14.B Response
from second subject

Fig 15.A Response

Fig 15.B Response

Fig 16.A Response
From first subject

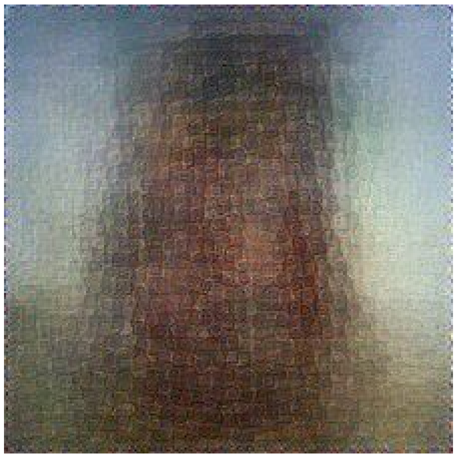


Fig 16.B Response
from second subject

Code

The algorithm used for implementation is explained below.

```
def reconstruct_img(features, net,
                    layer_weight=None, channel=None, mask=None, initial_image=None,
maxiter=500, disp=True,save_intermediate_every=1, save_intermediate=False,
loss_type='gram',

save_intermediate_path=None,save_intermediate_ext='jpg',save_intermediate_postpr
ocess=normalise_img):
          # loss function
          loss_fun = switch_loss_fun(loss_type)

          # make dir for saving intermediate
          if save_intermediate:
          if save_intermediate_path is None:
          save_intermediate_path = os.path.join('./recon_img_lbfgs_snapshots' +
datetime.now().strftime('%Y%m%dT%H%M%S'))
          if not os.path.exists(save_intermediate_path):
          os.makedirs(save_intermediate_path)

          # get img size, #of pixel and mean of img
          img_size = net.blobs['data'].data.shape[-3:]
          num_of_pix = np.prod(img_size)
          img_mean = net.transformer.mean['data']


          # initial image
          if initial_image is None:
          initial_image = np.random.randint(0, 256, (img_size[1], img_size[2], img_size[0]))
          if save_intermediate:
          save_name = 'initial_img.png'

PIL.Image.fromarray(np.uint8(initial_image)).save(os.path.join(save_intermediate_pat
h, save_name))

          # preprocess initial img
          initial_image = img_preprocess(initial_image, img_mean)
          initial_image = initial_image.flatten()

          # layer_list
          layer_list = list(features.keys())
```

```
        print("layer list : "+ str(layer_list))
        layer_list = sort_layer(net, layer_list)
        print("layer list sorted : "+ str(layer_list))

        # number of layers
        num_of_layer = len(layer_list)

        # layer weight
        if layer_weight is None:
        weights = np.ones(num_of_layer)
        weights = np.float32(weights)
        weights = weights / weights.sum()
        layer_weight = {}
        for i, lyr in enumerate(layer_list):
        layer_weight[lyr] = weights[i]

        # feature mask
        feature_masks = create_feature_masks(features, masks=mask,
channels=channel)

        # optimization
        loss_list = []
        res = minimize(obj_fun, initial_image, args = (net, features, feature_masks,
layer_weight, loss_fun, save_intermediate, save_intermediate_every,
save_intermediate_path, save_intermediate_ext,
                                        save_intermediate_postprocess, loss_list),
                method='L-BFGS-B', jac=True, options= {'maxiter': maxiter})

        # recon img
        img = res.x
        img = img.reshape(img_size)

        # return img
        return img_caffe_deproc(img, img_mean), loss_list


def obj_fun(img, net, features, feature_masks, layer_weight, loss_fun,
save_intermediate, save_intermediate_every, save_intermediate_path,
save_intermediate_ext, save_intermediate_postprocess, loss_list=[]):
        # reshape img
        img_size = net.blobs['data'].data.shape[-3:]
        img = img.reshape(img_size)

        # save intermediate image
        t = len(loss_list)
        if save_intermediate and (t % save_intermediate_every == 0):
        img_mean = net.transformer.mean['data']
```

```
save_path = os.path.join(save_intermediate_path, '%05d.%s' % (t,
save_intermediate_ext))
if save_intermediate_postprocess is None:
snapshot_img = img_caffe_deproc(img, img_mean)
else:
snapshot_img = save_intermediate_postprocess(img_caffe_deproc(img,
img_mean))
PIL.Image.fromarray(snapshot_img).save(save_path)

# layer_list
layer_list = features.keys()
layer_list = sort_layer(net, layer_list)

# num_of_layer
num_of_layer = len(layer_list)

# cnn forward
net.blobs['data'].data[0] = img.copy()
net.forward(end=layer_list[-1])

# cnn backward
loss = 0.
layer_start = layer_list[-1]
net.blobs[layer_start].diff.fill(0.)
for j in range(num_of_layer):
layer_start_index = num_of_layer - 1 - j
layer_end_index = num_of_layer - 1 - j - 1
layer_start = layer_list[layer_start_index]
if layer_end_index >= 0:
layer_end = layer_list[layer_end_index]
else:
layer_end = 'data'
feat_j = net.blobs[layer_start].data[0].copy()
feat0_j = features[layer_start]
mask_j = feature_masks[layer_start]
layer_weight_j = layer_weight[layer_start]
loss_j, grad_j = loss_fun(feat_j, feat0_j, mask_j)
loss_j = layer_weight_j * loss_j
grad_j = layer_weight_j * grad_j
loss = loss + loss_j
g = net.blobs[layer_start].diff[0].copy()
g = g + grad_j
net.blobs[layer_start].diff[0] = g.copy()
if layer_end == 'data':
net.backward(start=layer_start)
else:
net.backward(start=layer_start, end=layer_end)
net.blobs[layer_start].diff.fill(0.)
```

```
grad = net.blobs['data'].diff[0].copy()


grad = grad.flatten().astype(np.float64)
loss_list.append(loss)

return loss, grad
```