Technische Universität Hamburg
Institut für Mathematik
Lehrstuhl Numerische Mathematik

Jens-Peter M. Zemke
Jonas Grams

Mathematics of Neural Networks
winter semester 2021/2022
exercise sheet 8

Exercise 1: (4 points)

a) Implement the computation of CAM.

b) (Optional) Implement the computation of GradCAM.

Use the following skeleton and use the test as provided in the skeleton. Exchange `truck.jpg` to try different images.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow import keras as K
from scipy.ndimage import zoom
from matplotlib import cm

preprocess_input   = K.applications.resnet50.preprocess_input
decode_predictions = K.applications.resnet50.decode_predictions

# Load the ResNet50 network
def get_model():

  return K.applications.resnet50.ResNet50()

"""
Load an image from a given path with a given size
as TensorFlow Tensor.
"""
def get_image(img_path, size):

  # Load image
  img = K.preprocessing.image.load_img(img_path, target_size=size)
  # Cast img to a numpy array with shape (3, size[0], size[1])
  img = K.preprocessing.image.img_to_array(img)
  # Transform img to a 4-tensor of shape (1, 3, size[0], size[1])
  img = tf.expand_dims(img, axis=0)
  # Cast to float32, if not done yet
  img = tf.cast(img, dtype=tf.float32)

  return preprocess_input(img)

"""
For a given model with last convolutional layer
last_conv_layer and given input x get the output
of the model and the last convolutional layer
"""
```

```python
def get_output(model, last_conv_layer, x):

  get = K.Model([model.layers[0].input],
                [last_conv_layer.output,
                 model.layers[-1].output])

  return get(x)

# analyze the predictions of the net
def analyze_predictions(predictions):
    predicted_class = predictions.argmax()
    top5 = decode_predictions(predictions, top=5)[0]
    print("=========== resulting top predictions: ===========")
    for i in range(5):
        print("{}: probability {:6.2f}% for the class {}"\
              .format(i + 1, 100 * top5[i][2], top5[i][1]))
    return predicted_class

"""
Get the CAM heatmap for a given model, where the name
of the last convolutional layer is last_conv_layer_name
"""
def get_cam_heatmap(x, model, last_conv_layer_name):

  last_conv_layer = model.get_layer(last_conv_layer_name)

  # Get the output of the last convolutional layer and the
  # predicted class
  last_conv_out, predictions = get_output(model, last_conv_layer, x)
  predicted_class = analyze_predictions(predictions.numpy())

  # Get the weights of the prediction layer
  W, b = model.layers[-1].get_weights()

  """
  TODO Sum the output of the last convolutional layer
       over the channels. Scale each channel with the
       corresponding weight of the predicted class
       (See lecture notes, p.129 (version from 11.01.21)).
  """
  heatmap = None

  # The returned heatmap should have shape (h,w), where h is the
  # height of the output of the last convolutional layer, and w is
  # its width.
  """
  TODO Assure that the heatmap is a 2D numpy array. You can convert
       tensorflow tensors to numpy arrays with the numpy method.
       If you want a numpy array from the tensor x, call c.numpy().
       If you have a 4D array x of shape (1, h, w, 1), you can
       replace it by x[0, :, :, 0] to obtain a 2D array.
  """

  return heatmap

def get_gradient_cam_heatmap(img, model, last_conv_layer_name,
```

```
                             classifier_layer_names):

  last_conv_layer = model.get_layer(last_conv_layer_name)

  # Splitting the network into convolutional and classifier part
  # Model for the convolutional part
  convolutional_model = K.Model(model.inputs, last_conv_layer.output)

  # Model for the classifier part
  classifier_input = K.Input(shape=last_conv_layer.output.shape[1:])
  x = classifier_input
  for layer_name in classifier_layer_names:
    x = model.get_layer(layer_name)(x)

  classifier_model = K.Model(classifier_input, x)

  # Track gradients with GradientTape()
  with tf.GradientTape() as tape:

    # Call the convolutional part with img as input
    last_conv_out = convolutional_model(img)

    # Track the derivatives with respect to the output of the
    # last convolutional layer.
    tape.watch(last_conv_out)

    # Call the classifier part with last_conv_out as input
    predictions = classifier_model(last_conv_out)

    # Analyze the predictions and get winning class data
    predicted_class = analyze_predictions(predictions.numpy())
    top_class_channel = predictions[:,predicted_class]

  # Get the derivatives of the predicted class channel w.r.t. the output
  # of the last convolutional layer
  gradient = tape.gradient(top_class_channel, last_conv_out)

  # gradient is of shape (1, oh, ow, c) where oh, ow are the height and
  # width of the outout of the last convolutional layer. The average over
  # the first three axes has to be taken.
  pooled_gradient = tf.reduce_mean(gradient, axis=(0,1,2))

  """
  TODO Sum the output of the last convolutional layer over the channels.
       Scale each channel with the corresponding pooled gradient
       (see lecture notes, p.129 (version from 01.11.21)).
  """
  heatmap = None

  # The returned heatmap should have shape (h,w), where h is the
  # height of the output of the last convolutional layer, and w is
  # its width.
  """
  TODO Assure that the heatmap is a 2D numpy array. You can convert
       tensorflow tensors to numpy arrays with the numpy method.
       If you want a numpy array from the tensor x, call c.numpy().
```

```
      If you have an 4D array x of shape (1, h, w, 1), you can
      replace it by x[0, :, :, 0] to obtain a 2D array.
  """

  return heatmap

def superimpose_heatmap(img_path, heatmap, alpha=.7):

  # load image, e.g., float array of shape (465, 621, 3)
  image_np = plt.imread(img_path).astype(np.float32)
  heatmap_uint8 = np.uint8(np.maximum(heatmap, 0) / heatmap.max() * 255)
  cm_jet = cm.get_cmap("jet")
  jet_colors = cm_jet(np.arange(256))[:, :3]
  heatmap_jet = jet_colors[heatmap_uint8]
  # scale color heatmap to shape (465, 621, 3)
  target_h, target_w, _ = image_np.shape
  h, w, _ = heatmap_jet.shape
  heatmap_scaled = zoom(heatmap_jet, (target_h/h, target_w/w, 1))
  heatmap_scaled_uint8 = np.uint8(np.maximum(heatmap_scaled, 0)
                                  / heatmap_scaled.max() * 255)
  # superimpose image and heatmap
  return np.uint8(image_np * (1 - alpha) + heatmap_scaled_uint8 * alpha)

# show image using matplotlib
def show_image(image_path, title=None):
    image = plt.imread(image_path)
    plt.imshow(image)
    plt.title(title)
    plt.show()

# show heatmap using matplotlib and cm
def show_heatmap(heatmap, title=None):
    plt.imshow(heatmap, cmap='jet')
    plt.title(title)
    plt.show()


if __name__ == '__main__':

  img_path = 'truck.jpg'
  resnet_size = (224,224)

  model = get_model()

  last_conv_layer_name = 'conv5_block3_out'
  classifier_layer_names = ['avg_pool',
                            'predictions']

  img = get_image(img_path, resnet_size)

  heatmap_cam = get_cam_heatmap(img, model, last_conv_layer_name)
  heatmap_grad_cam = get_gradient_cam_heatmap(img, model,
                                              last_conv_layer_name,
                                              classifier_layer_names)

  image_cam = superimpose_heatmap(img_path, heatmap_cam)
```

```
image_grad_cam = superimpose_heatmap(img_path, heatmap_grad_cam)

plt.axis('off')
plt.imshow(image_cam)
plt.savefig('truck_cam.pdf')
plt.close()

plt.axis('off')
plt.imshow(image_grad_cam)
plt.savefig('truck_grad_cam.pdf')
plt.close()
```

Exercise 2: (2 points) Let $\mathbf{B}_0$ be symmetric. Let $\mathbf{y}_k, \mathbf{s}_k \in \mathbb{R}^n$ be given for all $k \in \mathbb{N}_0$. Prove that all BFGS approximations

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^\mathsf{T}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^\mathsf{T} \mathbf{B}_k^\mathsf{T}}{\mathbf{s}_k^\mathsf{T} \mathbf{B}_k \mathbf{s}_k}, \quad k \in \mathbb{N}_0$$

to the Hessian are symmetric and satisfy the secant condition $\mathbf{B}_{k+1}\mathbf{s}_k = \mathbf{y}_k$.

Exercise 3: (2+2*+2 points)

a) Let regular $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{U} \in \mathbb{R}^{n \times k}$, $\mathbf{V} \in \mathbb{R}^{k \times n}$ be given. Prove the Sherman-Morrison-Woodbury formula

$$(\mathbf{A} + \mathbf{U}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I}_k + \mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1}$$

for regular $\mathbf{I}_k + \mathbf{V}\mathbf{A}^{-1}\mathbf{U}$.

b) Use the Sherman-Morrison-Woodbury formula to derive the update

$$\mathbf{H}_{k+1} = \left(\mathbf{I} - \frac{\mathbf{s}_k \mathbf{y}_k^\mathsf{T}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k}\right) \mathbf{H}_k \left(\mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^\mathsf{T}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k}\right) + \frac{\mathbf{s}_k \mathbf{s}_k^\mathsf{T}}{\mathbf{y}_k^\mathsf{T} \mathbf{s}_k}$$

of the inverse $\mathbf{H}_k = \mathbf{B}_k^{-1}$ of the approximation $\mathbf{B}_k$ of the Hessian in BFGS from Exercise 2.

c) Suppose that the vectors $\mathbf{s}_k$, $\mathbf{y}_k$ of Exercise 2 satisfy additionally the curvature condition $\mathbf{s}_k^\mathsf{T}\mathbf{y}_k > 0$ and that $\mathbf{B}_0$ is symmetric positive definite. Prove that then all $\mathbf{H}_k$ (and thus all $\mathbf{B}_k$) are symmetric positive definite[1].

Exercise 4: (3 points) Implement Newton's method, BFGS, and L-BFGS for the minimization of the function

$$f : \mathbb{R}^n \to \mathbb{R}, \quad f(\mathbf{x}) = \frac{1}{2}\left((x_1 - 1)^2 + \sum_{k=2}^{n} k(x_k - x_{k-1})^2\right)$$

How well do these methods work? For L-BFGS compute the update with the following algorithm. Use $\gamma \in \{1, \mathbf{s}_{k-1}^\mathsf{T}\mathbf{y}_{k-1}/\mathbf{y}_{k-1}^\mathsf{T}\mathbf{y}_{k-1}\}$.

[1] A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is symmetric positive definite when it is symmetric and for all nonzero vectors $\mathbf{v} \in \mathbb{R}^n$ we have $\mathbf{v}^\mathsf{T}\mathbf{A}\mathbf{v} > 0$.

Algorithm 1 Iterative computation of the update in L-BFGS

$\mathbf{p} \leftarrow -\mathbf{g}_k;$
for $j = k-1 : k-m : -1$ **do**
    $\alpha_j \leftarrow \rho_j \mathbf{s}_j^{\mathsf{T}} \mathbf{p};$
    $\mathbf{p} \leftarrow \mathbf{p} - \mathbf{y}_j \alpha_j;$
end for
$\mathbf{p} \leftarrow \mathbf{p}\gamma;$
for $j = k-m : k$ **do**
    $\alpha_j \leftarrow \alpha_j - \rho_j \mathbf{y}_j^{\mathsf{T}} \mathbf{p};$
    $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{s}_j \alpha_j;$
end for

Hint: Since f is quadratic, the gradient $\nabla f(\mathbf{x})$ is linear in each component. The Hessian $\nabla^2 f(\mathbf{x})$ is constant, symmetric, and positive definite.