SGN-41007 Assignment Task 1

Predict competition classes using a pre-trained convnet and ***scikit-learn*** *tools for classification.*

- ▷ The first assignment task uses scikit-learn tools with a pretrained convnet for feature extraction. This is not the most usual way to use convnets, but can be a good approach if the amount of data is too small for proper training of convnet (which we will do in the second assignment).
- ▷ Here, create a convnet, pass all images through it, fit a sklearn model and predict the classes for test data. Finally, we write the results in a submission file.
- ▷ A written report of the group solution is required; due date Sunday 24.11.2019 at 23:59. Return to Moodle (`moodle.tuni.fi`).
- ▷ If you need help, we have a helpdesk session in TC303 on Monday, 18.11. at 12:15 - 13:45.
- ▷ See also video lecture of the assignment (25 min): `https://panopto.tuni.fi/Panopto/Pages/Viewer.aspx?id=4abc9bf8-cb57-4dcc-bda4-ab0500927778`.

1. *Load data.*

   If you work on your own computer, download the data from Kaggle. If you use the computers in classroom TC303, then the competition data is in folder `C:\Work\sgndataset\`.

2. *Create an index of class names.*

   The class names can be collected from the training data folder using `os.listdir`. To guarantee unique ordering, it is safe to sort the labels in alphabetical order:

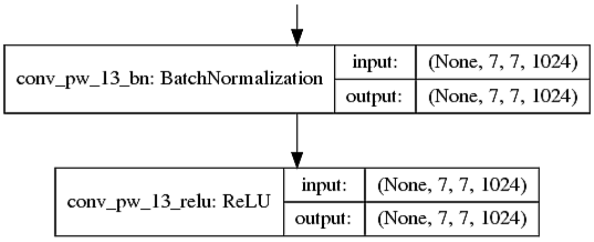
```
class_names = sorted(os.listdir(r"C:\Work\sgndataset\train\"))
```

3. *Prepare a pretrained CNN for feature extraction.*

   We will take the last layer output of an Imagenet-trained mobilenet and use that as input for sklearn classifiers. First, we instantiate the network:

```
base_model = tf.keras.applications.mobilenet.MobileNet(
                input_shape = (224,224,3),
                include_top = False)
```

   You can get a listing of the network structure by `base_model.summary()`, or `tf.keras.utils.plot_model`. At the end of the network, we have the following modules.



In other words, the base model outputs a 1024 channel stack of 7×7 images. We will vectorize this stack by computing the average of each 7×7 matrix. To this aim, we need to construct another model from `base_model` using Keras' functional interface. In practice, the base model can be extended by additional layers as follows.

```
in_tensor = base_model.inputs[0]   # Grab the input of base model
out_tensor = base_model.outputs[0] # Grab the output of base model

# Add an average pooling layer (averaging each of the 1024 channels):
out_tensor = tf.keras.layers.GlobalAveragePooling2D()(out_tensor)

# Define the full model by the endpoints.
model = tf.keras.models.Model(inputs  = [in_tensor],
                              outputs = [out_tensor])

# Compile the model for execution. Losses and optimizers
# can be anything here, since we don't train the model.
model.compile(loss = "categorical_crossentropy", optimizer = 'sgd')
```

4. *Load all images and apply the network to each.*

Next step is to load all images with their respective classes and push them through the network (generating a list of 1024-dimensional feature vectors). Below is a template how this can be done.

```
# Find all image files in the data directory.

X = []       # Feature vectors will go here.
y = []       # Class ids will go here.

for root, dirs, files in os.walk(r"C:\Work\sgndataset\train\"):
    for name in files:
        if name.endswith(".jpg"):

            # Load the image:
            img = plt.imread(root + os.sep + name)

            # Resize it to the net input size:
            img = cv2.resize(img, (224,224))

            # Convert the data to float, and remove mean:
            img = img.astype(np.float32)
            img -= 128
```

```
            # Push the data through the model:
            x = model.predict(img[np.newaxis, ...])[0]

            # And append the feature vector to our list.
            X.append(x)


            # Extract class name from the directory name:
            label = name.split(os.sep)[-2]
            y.append(class_names.index(label))

# Cast the python lists to a numpy array.
X = np.array(X)
y = np.array(y)
```

5. *Try different models.*

   After the feature extraction procedure, experiment with the following classifiers (additional ones are welcome, too); measure the accuracy on a 20% validation split and report.

   a) Linear discriminant analysis classifier.
   b) Support vector machine (linear kernel).
   c) Support vector machine (RBF kernel).
   d) Logistic regression.
   e) Random forest.

6. *Create submission file.*

   Select the best of the above classifiers, fit that with all training data (80% + 20%), and predict classes for the test data.

   Create a submission file like in the following template.

```
with open("submission.csv", "w") as fp:

    fp.write("Id,Category\n")

    for <image file name> in test files:
        # 1. load image and resize
        # 2. vectorize using the net
        # 3. predict class using the sklearn model
        # 4. convert class id to name (label = class_names[
            class_index])

        fp.write("%d,%s\n" % (i, label))
```

7. *Submit.*

   Log in to Kaggle and submit your solution. Describe in your report what was your Kaggle score and how it differs from what you estimated locally using the 20% validation set.