# Resit Assignment

**Deadline: Monday, July 16th (23:59)**

June 21, 2018

## Question 1: Neural Codes & Nearest Neighbor retrieval (33pt)

**Data: Caltech256**

The Caltech256 dataset consists of images from 256 different object classes, as well as an extra clutter class with random images. In this question you will develop an image retrieval system using image representations (neural codes) learned with a deep convolutional neural network and a given distance metric.

The Caltech256 dataset can be downloaded from: `http://www.vision.caltech.edu/Image_Datasets/Caltech256/`. *NOTE: this is NOT the same dataset as Caltech101, which was used for Assignment 2 during the regular course time.* Further info about the dataset (e.g. differences with Caltech101) can also be found there.

**Retrieval score**

For any given query image, we can define an image retrieval score ranging from 0 to 10 as follows:

- Obtain the 10 images from the dataset most similar to the query image (excluding the query image itself!), according to some predetermined distance metric.
- Count how many of those 10 images have the same class label as the query image, this number will be the retrieval score.

*NOTE: the "clutter" set is NOT an object class, make sure to separate it from the rest of the data and do not use it for retrieval.*

### Task 1.1: Distance measure: Neural codes

For the retrieval score described above, we need to define some distance metric. For this, we will use a neural network to compute some representation (called "neural code") for each of the images; our distance metric will then be the L2-distance (Euclidean distance) between those features.

a) From Keras, obtain the VGG19 (*not VGG16!*) network pre-trained on ImageNet to compute "neural codes" for all the images in the Caltech256 dataset. I.e. remove the final layer(s) of the model and obtain features from the "fc2"-layer. Save these features in a single Pickle file.

b) For each of the Caltech256 images, obtain its retrieval score using L2-distance on "fc2" neural codes. Report the average retrieval score of the entire dataset, as well as the average retrieval scores within each of the 256 classes.

c) Consider the class with the highest average retrieval score, as well as the class with the lowest average retrieval score. For each of these classes, do the following:

- Visualise a few query images, along with its 10 most similar images (according to L2-distance on "fc2" neural codes). Also show the image retrieval score for this query image, as well as the correct classes of the retrieved images.
- Mention a few possible reasons why image retrieval works well or not for this class, using this retrieval method.

Also relate the "good" class to the "bad" class, point out the differences between the classes that likely lead to the difference in performance.

### Task 1.2: Clutter images

Besides the regular 256 classes, Caltech256 also contains an extra clutter "class" of images that are not necessarily related to each other. We can thus not reasonably evaluate the same image retrieval score as before, since we do not consider clutter images to be related to each other in the first place.

Given a query image from this clutter set however, it is still possible to obtain the 10 most similar images from the regular Caltech256 dataset. We could then evaluate each image by hand, and decide whether it is relevant or not. This is a lot of manual work however, which we typically try to avoid.

a) For each image from the clutter set, obtain the 10 most similar images in the original Caltech256 dataset (again using L2-distance on "fc2" neural codes).

b) Think of a way to quantitatively assess the performance of this kind of image retrieval, without needing to manually check the images. In other words, devise an image retrieval score for a query image from the clutter set, using only the class information from the 10 retrieved images from Caltech256. Briefly describe and motivate your scoring system.

c) Implement and evaluate the image retrieval scoring metric you defined in b) on all images from the clutter set. Visualise the distribution of scores (e.g. using a histogram or boxplot). Interpret/motivate your results.

d) Visualise the query image (from the clutter set) with the worst score (as defined in b)), as well as the one with the best score. For each, also visualise its 10 most similar images from Caltech256. Given these two examples, would you say your image retrieval scoring metric is reasonable?

## Question 2: Siamese Networks & One Shot Learning (33pt)

**Data: Caltech101 Silhouettes**

We will now work with a dataset based on Caltech101, that consists of binary (black and white) images of outlines/silhouettes of Caltech101 images. For this question we will use the 28x28 images from this dataset, and train a Siamese network for metric learning on 81 of its classes, which we will then use for one-shot learning on the remaining 20 classes.

The dataset, as well as more information about it, can be found here: `https://people.cs.umass.edu/~marlin/data.shtml`.

### Task 2.1: Obtain and process data

- Download the `caltech101_silhouettes_28.mat` images, and convert them to a format usable with Python/Keras.
- For each class, only keep the first 20 images, in order to simulate a situation in which limited data is available per class.

- Split the data into two sets: the first 81 classes as a training set for a Siamese Network, and the remaining 20 classes as a test set for one-shot learning.

### Task 2.2: Train Siamese Network and evaluate one-shot accuracy

- Design a suitable Siamese Network for the training dataset obtained above (the first 81 classes of Caltech101 silhouettes). Train it to predict the probability that a pair of images belongs to the same class, using the label information as ground truth.
- Implement 20-way one-shot learning tasks for the remaining 20 classes. Evaluate this accuracy periodically while training the Siamese Network. Report your results.

### Task 2.3: Motivation

a) Motivate the architecture of your Siamese Network.

b) Discuss the one-shot performance of the Siamese Network. Does it perform reasonable well? Is this a suitable method for this task?

c) Given the data selection from Task 2.1, another way of performing one-shot learning would be to train a classifier neural network on the first 81 classes, and using its neural codes with nearest neighbor matching for one-shot learning. Without implementing it, assess this method. Would it work well for this data? Would it outperform Siamese Networks? Why, or why not?

## Question 3: Emojifier – Classifying emotions with RNN (33pt)

Sentiment analysis allows computer to identify human emotions from raw text information. Traditional approach mainly relies on count-based algorithm, i.e. by counting *happy* or *sad* words in observed text, and external knowledge base sentiment corpus (wordNet, sentiNet) – as compared to neural-based methods we study in this course.

Through Practicals 5-6 and Assignment 3 of the course, you have learned how to construct a Recurrent Neural Networks (RNN) model for a binary "Sentiment Classification" task. This RNN model is used as an end-to-end application to detect sentiment polarity of text reviews, i.e. whether a statement can be categorized as *positive* or *negative* sentiment.

Instead of detecting sentiment polarity (positive or negative), you will build a RNN model for classifying human emotions in a more expressive message, i.e. by using emojis. Emoji is a graphical symbol (ideogram) of human's feelings, moods, and emotions – which is mostly available in social network platforms, website, and mobile phone applications. Using the constructed model, you can express text with its predicted emoji label. You can further create an emoji sentiment score based on emotions found in corpus - or - you can use available emoji sentiment ranking to classify and analyse the sentiment polarity of text input. An advanced work [1] further utilizes this sentiment spectrum of emotions to detect fake news or illegitimate reviews.

In this assignment, you need to complete the following tasks:

1. Preprocess data into training, validation, and test set, including creating the word vocabulary index and functions to transform raw emotion labels into numerical vectors and emojis. Briefly discuss the statistics of raw data (the statistical occurrences of emotion labels in observed corpus).

[1] https://github.com/tlkh/text-emotion-classification

2. Construct a model to transform text input into ideograms (emoji) symbolizing emotions or moods of the corresponding text. Train and validate the model. Discuss the model performance during training and validation stage.

3. Compute evaluation metrics: Confusion matrix, Accuracy, Precision, Recall, and F1-score from data set 1 (test set of "emotion" data).

4. Discuss type of emotions in which the model is under-performed and its possible reasons. Give a running examples (i.e. text input and expected emoji vs. predicted emoji)

5. Create a prototype of emoji sentiment ranking from data set 2 ("sentiment labelled sentences" data).

## Data description

1. Twitter emotion sentiment [2], can be downloaded from `https://storage.googleapis.com/trl_data/emotion.zip`. This twitter data set contains 40.000 raw tweets and its emotion labels (13 labels). Use this data for training, validating, and testing the constructed model. Choose a reasonable partition number to split the data.

   Note: You can use preprocessing code for twitter data, as in Practical 5.3 and 5.4. You will also need to transform 13 classes into 5 classes as a comparison task to analyse model performance.

2. Sentiment labelled sentences [3], i.e. sentences labelled as positive and negative, extracted from Amazon reviews, IMDB reviews, and Yelp. Each consists of 500 positive and 500 negative sentences. Data can be downloaded from `https://storage.googleapis.com/trl_data/sentiment_sentences.zip`. Use this data for the last task, creating a prototype of emoji sentiment ranking.

3. Glove pretrained embedding from twitter data set, can be downloaded from `https://nlp.stanford.edu/projects/glove/`. Use this pretrained embedding as non-trainable weights for embedding layer in the constructed model.

## Model architecture

Build emojifier - a RNN model that processes the text and produces a classification output. You can either choose to use LSTM/GRU layer(s). Justify your chosen model architecture.

## Evaluation metrics

Compute evaluation metrics (i.e. Accuracy, Precision, Recall, F1-Score) on two (2) preprocessed data, i.e. original data with 13 classes, and data with 5 classes to analyse model performance on two (2) settings of multi-class classification training and evaluation.

Note: This requires you to train the model on data with 5 classes as well.

## Sentiment spectrum of emotions/emojis

Create a prototype of emoji sentiment ranking, as exemplified in `http://kt.ijs.si/data/Emoji_sentiment_ranking/` from "sentiment labelled sentences" data set.

- Predict emotion labels (13 classes) of corresponding data sets (Amazon, IMDB, Yelp) using the model trained on twitter emotion data.

[2] https://github.com/tlkh/text-emotion-classification
[3] https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences

- Assuming your model prediction is nearly accurate (you do not have gold standard of "emotion labels" in this data set - and thus, we refer this as *a prototype*), use the results (predicted emotion labels / emojis) to calculate the occurrences of each emotion label in positive and negative sentiments. Use the statistics to compute sentiment spectrum (percentage score) of each emoji / emotion label.

Note: You do not need to create similar statistics. Use your own table presentation. The required information need to be present: emojis, emotion labels (text), total occurrences, positive occurrences, negative occurrences, sentiment bar/spectrum.