# Homework 5

All assignments need to be submitted via github classroom:
https://classroom.github.com/g/F65AmlrV
and on gradescope. You can submit in groups of maximum size two.

The homework is due 05/01/19 at 1pm.

All the tasks are to be completed using the keras Sequential interface. It's recommended that you run your code on GPU using google colab:
https://colab.research.google.com/
You can find a starter notebook here if you haven't used colab before:
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb

You can enable GPU support at "runtime" -> "change runtime type".
Feel free to experiment with TPU support if you're adventurous.
You can use any other resources at your disposal if you prefer. You should not use k-fold cross-validation for any of these tasks. You can use StratifiedShuffleSplit to create a single train-validation split for use with GridSearchCV.
Use of GridSearchCV might not be the best option for any task but task1, though.

## Task 1 [10 Points]

Run a multilayer perceptron (feed forward neural network) with two hidden layers and rectified linear nonlinearities on the digits dataset from sklearn using the keras Sequential interface. Include code for selecting L2 regularization strength and number of hidden units using GridSearchCV and evaluation on an independent test-set.

## Task 2 [35 Points]

Train a multilayer perceptron (fully connected) on the Fashion MNIST dataset using the traditional train/test split as given by fashion_mnist.load_data in keras. Use a separate 10000 samples (from the training set) for model selection and to compute learning curves (accuracy vs epochs, not accuracy vs n_samples). Compare a "vanilla" model with a model using drop-out and evaluate if using drop-out allows you to learn a bigger network. Then, compare to a model using batch normalization. Visualize learning curves for all models.

# Task 3 [55 Points]

Train a convolutional neural network on the following dataset:
https://lhncbc.nlm.nih.gov/publication/pub9932
The goal is to classify cells infected with malaria against those that are not.

**3.1** Start with a convolutional model without residual connections (using batch normalization is likely to be helpful and you should try it, whether you use dropout is your choice).

**3.2** Augment the data using rotations, mirroring and possibly other transformations. How much can you improve your original model by data augmentation?

**3.3** Build a deeper model using residual connections. Show that you can build a deep model that would not be able to learn if you remove the residual connections (i.e. compare a deep model with and without residual connections while the rest of the architecture is constant). Feel free to reuse existing architectures from the literature or use them as inspiration for your own. You can find commonly used architectures here:
https://keras.io/applications/
However, the point of the exercise is to learn the weights from scratch, so please do not reuse the weights shipped with these applications.

3.4 BONUS / Optional: Transfer learning
Reuse an existing architecture from keras (https://keras.io/applications/) and compare retraining only the densely connected layer with fine-tuning the whole network.

**Hint**: Make sure you are doing the reshape for the training set correctly. A direct reshape might give you garbled images. Display an image after reshaping to make sure they are correct.

# Some additional advice to help you along:

- Make sure all your code is running on GPU. You can use `tf.debugging.set_log_device_placement(True)` to see which device is being used and confirm it is the device you intended.
- Preprocess the images before training a model.
- Test your code on a small part of the data before training the model. You don't want your code to fail on a print statement after waiting for the network to train.