DL Exercise 5: Tensorflow and Classification Challenge

Pattern Recognition Lab, Friedrich-Alexander University of Erlangen-Nürnberg

January 21, 2019

Goal of this exercise

- Get to know a widely used deep learning framework: TensorFlow
- Implement & train two widely architectures: AlexNet and ResNet18
- Classification on **real** data: Images from solar panels

Goal of this exercise

- Get to know a widely used deep learning framework: TensorFlow
- Implement & train two widely architectures: AlexNet and ResNet18
- Classification on **real** data: Images from solar panels
- **Challenge yourself & your colleagues!**

Organizational

Part I: Classification with Tensorflow - Mandatory

- Implementation & training of TensorFlow architectures
- No oral presentation, BUT: submission of trained models in submission system (later more)
- Goal: reach mean F1 score of > 0.60 for both architectures
- **Deadline: Feb. 15**

Organizational

Part II: Challenge - Optional, but highly encouraged

- Try to find & train the best architecture & model for this task!
- Compete with your colleagues!
- **Deadline: March 1**
- Announcement of winners & prices: March 4 - 8

Source: Designed by Freepik

Data set: Identification of defects in solar panels

- Solar moduls are composed of cells
- Are subject to degradation (transport, wind, hail, ...)
- Different defects, e.g., cracks or inactive regions

Data set: Identification of defects in solar panels

- Solar moduls are composed of cells
- Are subject to degradation (transport, wind, hail, ...)
- Different defects, e.g., cracks or inactive regions
- **Task:** Automatically determine which defect(s) a module has
- Panel can have no or multiple defects ➔ **multi-label problem**!

Data set: Identification of defects in solar panels

- Solar moduls are composed of cells
- Are subject to degradation (transport, wind, hail, ...)
- Different defects, e.g., cracks or inactive regions
- **Task:** Automatically determine which defect(s) a module has
- Panel can have no or multiple defects ➔ **multi-label problem**!

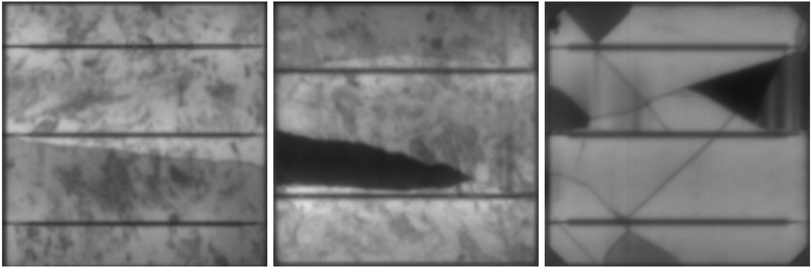


Figure: Left: Crack on a polycristalline module; Middle: Inactive region; Right: Cracks and inactive regions on a monocristalline module

TensorFlow: General introduction

- Open source software library for numerical computations
- Using data flow graphs:
  - Nodes = mathematical operations (e.g. convolutions)
  - Edges = multidimensional data arrays (=tensors)
  - Implemented in C++, different APIs
  - We use the Python API

TensorFlow: General introduction

- Open source software library for numerical computations
- Using data flow graphs:
  - Nodes = mathematical operations (e.g. convolutions)
  - Edges = multidimensional data arrays (=tensors)
  - Implemented in C++, different APIs
  - We use the Python API
- Python API provides higher- and lower-level functionalities
  - We suggest to go somewhat “middle-level”, e.g., tf.nn/tf.layer/tf.losses
  - Takes care of most variables definitions
  - We will handle data input, training iterations, etc. ourselves

TensorFlow: General introduction

- Open source software library for numerical computations
- Using data flow graphs:
  - Nodes = mathematical operations (e.g. convolutions)
  - Edges = multidimensional data arrays (=tensors)
  - Implemented in C++, different APIs
  - We use the Python API
- Python API provides higher- and lower-level functionalities
  - We suggest to go somewhat “middle-level”, e.g., tf.nn/tf.layer/tf.losses
  - Takes care of most variables definitions
  - We will handle data input, training iterations, etc. ourselves
- **Online documentation**: https://www.tensorflow.org/api_docs/python/

TensorFlow: General introduction

- Simple example:

```python
import tensorflow as tf
# We define two constant nodes (=tensors)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
# And define some operation on these nodes
node3 = tf.subtract(node1, node2)
# Only now we instantiate the graph and execute it
with tf.Session() as sess:
    sess.run(node3)
```

TensorFlow: General introduction

- Simple example:

```
import tensorflow as tf
# We define two constant nodes (=tensors)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
# And define some operation on these nodes
node3 = tf.subtract(node1, node2)
# Only now we instantiate the graph and execute it
with tf.Session() as sess:
    sess.run(node3)
```

TensorFlow: General introduction

- Simple example:

```
import tensorflow as tf
# We define two constant nodes (=tensors)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
# And define some operation on these nodes
node3 = tf.subtract(node1, node2)
# Only now we instantiate the graph and execute it
with tf.Session() as sess:
    sess.run(node3)
```

TensorFlow: General introduction

- Simple example:

```python
import tensorflow as tf
# We define two constant nodes (=tensors)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
# And define some operation on these nodes
node3 = tf.subtract(node1, node2)
# Only now we instantiate the graph and execute it
with tf.Session() as sess:
    sess.run(node3)
```

TensorFlow: General introduction

- Simple example:

```python
import tensorflow as tf
# We define two constant nodes (=tensors)
node1 = tf.constant(3.0, dtype=tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
# And define some operation on these nodes
node3 = tf.subtract(node1, node2)
# Only now we instantiate the graph and execute it
with tf.Session() as sess:
    sess.run(node3)
```

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables
3. Define loss function and optimizer

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables
3. Define loss function and optimizer
4. Define graph consisting of all operations (forward and backward pass)

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables
3. Define loss function and optimizer
4. Define graph consisting of all operations (forward and backward pass)
5. Run all operations within a TensorFlow Session

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables
3. Define loss function and optimizer
4. Define graph consisting of all operations (forward and backward pass)
5. Run all operations within a TensorFlow Session
6. Train graph:
   - Compute forward pass
   - Compute loss value
   - Compute backward pass and adjust weights

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables
3. Define loss function and optimizer
4. Define graph consisting of all operations (forward and backward pass)
5. Run all operations within a TensorFlow Session
6. Train graph:
   - Compute forward pass
   - Compute loss value
   - Compute backward pass and adjust weights
7. Validate/test graph:
   - Freeze all values in graph
   - Compute forward pass (=prediction) for validation/test data

General steps

General steps developing a neural network with TensorFlow:

1. Data preparation
2. Define neural network architecture with trainable variables
3. Define loss function and optimizer
4. Define graph consisting of all operations (forward and backward pass)
5. Run all operations within a TensorFlow Session
6. Train graph:
   - Compute forward pass
   - Compute loss value
   - Compute backward pass and adjust weights
7. Validate/test graph:
   - Freeze all values in graph
   - Compute forward pass (=prediction) for validation/test data
8. Monitor training process using TensorBoard

Step 0: TensorFlow preparations and terms

- Add tensorflow package:
  - Command: **addpackage tensorflow**
  - follow the instructions

Step 0: TensorFlow preparations and terms

- Add tensorflow package:
  - Command: **addpackage tensorflow**
  - follow the instructions
- Import TensorFlow:

```
import tensorflow as tf
```

Step 0: TensorFlow preparations and terms

- Add tensorflow package:
  - Command: **addpackage tensorflow**
  - follow the instructions
- Import TensorFlow:

```
import tensorflow as tf
```

- Define interacting operations with so-called symbolic variables:
  - Placeholder for input tensors (=training or test batches of data)

```
images_placeholder =
tf.placeholder(tf.float32, shape=(batch_size, height_size,
    width_size, channels_size))
```

Step 0: TensorFlow preparations and terms

- Variables (=modifiable during training)

```
# Variable with shape [3, 3, 3, 16]
example_variable = tf.get_variable('example_var', shape=[3, 3,
    3, 16], initializer=tf.contrib.layers.xavier_initializer())
```

Step 0: TensorFlow preparations and terms

- Variables (=modifiable during training)

```
# Variable with shape [3, 3, 3, 16]
example_variable = tf.get_variable('example_var', shape=[3, 3,
    3, 16], initializer=tf.contrib.layers.xavier_initializer())
```

Step 0: TensorFlow preparations and terms

- Variables (=modifiable during training)

```
# Variable with shape [3, 3, 3, 16]
example_variable = tf.get_variable('example_var', shape=[3, 3,
    3, 16], initializer=tf.contrib.layers.xavier_initializer())
```

- Variables are fed with values and evaluated during a so-called session:

```
# Create a session
with tf.Session() as sess:
        # Initialize the variables
        sess.run(tf.global_variables_initializer())
        # Run this session and evaluate e.g. values which are
        # contained at the moment in the variable:
        sess.run(example_variable)
```

Step 0: TensorFlow preparations and terms

- Variables (=modifiable during training)

```
# Variable with shape [3, 3, 3, 16]
example_variable = tf.get_variable('example_var', shape=[3, 3,
    3, 16], initializer=tf.contrib.layers.xavier_initializer())
```

- Variables are fed with values and evaluated during a so-called session:

```
# Create a session
with tf.Session() as sess:
        # Initialize the variables
        sess.run(tf.global_variables_initializer())
        # Run this session and evaluate e.g. values which are
        # contained at the moment in the variable:
        sess.run(example_variable)
```

Step 0: TensorFlow preparations and terms

- Variables (=modifiable during training)

```
# Variable with shape [3, 3, 3, 16]
example_variable = tf.get_variable('example_var', shape=[3, 3,
    3, 16], initializer=tf.contrib.layers.xavier_initializer())
```

- Variables are fed with values and evaluated during a so-called session:

```
# Create a session
with tf.Session() as sess:
        # Initialize the variables
        sess.run(tf.global_variables_initializer())
        # Run this session and evaluate e.g. values which are
        # contained at the moment in the variable:
        sess.run(example_variable)
```

Step 0: TensorFlow preparations and terms

- Variables (=modifiable during training)

```
# Variable with shape [3, 3, 3, 16]
example_variable = tf.get_variable('example_var', shape=[3, 3,
    3, 16], initializer=tf.contrib.layers.xavier_initializer())
```

- Variables are fed with values and evaluated during a so-called session:

```
# Create a session
with tf.Session() as sess:
        # Initialize the variables
        sess.run(tf.global_variables_initializer())
        # Run this session and evaluate e.g. values which are
        # contained at the moment in the variable:
        sess.run(example_variable)
```

Step 1: Data preparation

- Load data from disk and provide batches for training
- Normalization & data augmentation
- Handling of unbalanced data etc.

➔ In this exercise, this part will already be provided in `dataset.py` in the code skeleton.

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Convolution:

```
# Definition of a convolution layer
conv1_layer = tf.layers.conv2d(input_layer, filters=16, strides
    =(CONV_STRIDE_SIZE, CONV_STRIDE_SIZE), padding=PADDING,
    use_bias=True)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Convolution:

```
# Definition of a convolution layer
conv1_layer = tf.layers.conv2d(input_layer, filters=16, strides
    =(CONV_STRIDE_SIZE, CONV_STRIDE_SIZE), padding=PADDING,
    use_bias=True)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Convolution:

```
# Definition of a convolution layer
conv1_layer = tf.layers.conv2d(input_layer, filters=16, strides
    =(CONV_STRIDE_SIZE, CONV_STRIDE_SIZE), padding=PADDING,
    use_bias=True)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Fully connected:

```
# If e.g. the previous layer was a convolution and thus the
# input tensor is a 4-D one, we need to reshape our input
# tensor first
input_layer = tf.layer.flatten(input_layer)
# Then we can multiply the input_layer with the weights matrix
# and add a bias in a fully connected layer (tf: dense layer)
fc = tf.layers.dense(input_layer, units=n_weights_fc1,
    activation=None, use_bias=True)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Fully connected:

```
# If e.g. the previous layer was a convolution and thus the
# input tensor is a 4-D one, we need to reshape our input
# tensor first
input_layer = tf.layer.flatten(input_layer)
# Then we can multiply the input_layer with the weights matrix
# and add a bias in a fully connected layer (tf: dense layer)
fc = tf.layers.dense(input_layer, units=n_weights_fc1,
    activation=None, use_bias=True)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Fully connected:

```
# If e.g. the previous layer was a convolution and thus the
# input tensor is a 4-D one, we need to reshape our input
# tensor first
input_layer = tf.layer.flatten(input_layer)
# Then we can multiply the input_layer with the weights matrix
# and add a bias in a fully connected layer (tf: dense layer)
fc = tf.layers.dense(input_layer, units=n_weights_fc1,
    activation=None, use_bias=True)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Activation:

```
layer_output = tf.nn.relu(pre_activation)
```

Step 2: Build architecture

Architecture of a network can consist of different layers:

- Pooling:

```
pooled_output = tf.layers.max_pooling2d(input_layer, pool_size
    =(POOL_KERNEL_SIZE, POOL_KERNEL_SIZE), strides=(
    POOL_STRIDE_SIZE, POOL_STRIDE_SIZE), padding=PADDING)
```

Step 3: Build loss function and optimizer

- Loss: Compare current predictions with ground-truth labels
  - For our classification task: Cross entropy

```
norm_prediction = tf.nn.softmax(net_output)
cross_entropy = tf.losses.softmax_cross_entropy(labels=
    gt_labels, logits=net_output)
```

Step 3: Build loss function and optimizer

- Loss: Compare current predictions with ground-truth labels
  - For our classification task: Cross entropy

```
norm_prediction = tf.nn.softmax(net_output)
cross_entropy = tf.losses.softmax_cross_entropy(labels=
    gt_labels, logits=net_output)
```

- Optimizer
  - E.g. Stochastic gradient descent

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# We use default graph
with tf.Graph().as_default():
# Get placeholder for data
images_placeholder = tf.placeholder(tf.float32, [None,
    height_size, width_size, channels_size])
labels_placeholder = tf.placeholder(tf.float32, [None,
    NUM_CLASSES])
# Some preprocessing... e.g. mean subtraction
mean_normalized_images = tf.subtract(images_placeholder, mean)

# Define the forward pass of the model, e.g., for implemented
    alexnet
output = model.alexnet(mean_normalized_images)
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# We use default graph
with tf.Graph().as_default():
# Get placeholder for data
images_placeholder = tf.placeholder(tf.float32, [None, height_size, width_size, channels_size])
labels_placeholder = tf.placeholder(tf.float32, [None, NUM_CLASSES])
# Some preprocessing... e.g. mean subtraction
mean_normalized_images = tf.subtract(images_placeholder, mean)

# Define the forward pass of the model, e.g., for implemented alexnet
output = model.alexnet(mean_normalized_images)
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# We use default graph
with tf.Graph().as_default():
# Get placeholder for data
images_placeholder = tf.placeholder(tf.float32, [None,
    height_size, width_size, channels_size])
labels_placeholder = tf.placeholder(tf.float32, [None,
    NUM_CLASSES])
# Some preprocessing... e.g. mean subtraction
mean_normalized_images = tf.subtract(images_placeholder, mean)

# Define the forward pass of the model, e.g., for implemented
    alexnet
output = model.alexnet(mean_normalized_images)
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# We use default graph
with tf.Graph().as_default():
# Get placeholder for data
images_placeholder = tf.placeholder(tf.float32, [None, height_size, width_size, channels_size])
labels_placeholder = tf.placeholder(tf.float32, [None, NUM_CLASSES])
# Some preprocessing... e.g. mean subtraction
mean_normalized_images = tf.subtract(images_placeholder, mean)

# Define the forward pass of the model, e.g., for implemented alexnet
output = model.alexnet(mean_normalized_images)
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# We use default graph
with tf.Graph().as_default():
# Get placeholder for data
images_placeholder = tf.placeholder(tf.float32, [None, height_size, width_size, channels_size])
labels_placeholder = tf.placeholder(tf.float32, [None, NUM_CLASSES])
# Some preprocessing... e.g. mean subtraction
mean_normalized_images = tf.subtract(images_placeholder, mean)

# Define the forward pass of the model, e.g., for implemented alexnet
output = model.alexnet(mean_normalized_images)
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# Loss and training (= backward pass) for exclusive labels
loss = tf.losses.softmax_cross_entropy(labels_placeholder,
    output)
optimizer =  tf.train.AdamOptimizer(LEARNING_RATE)
train_op = optimizer.minimize(loss)

# Create summary object for TensorBoard
summary = tf.summary.merge_all()

# Initialize all variables in graph, e.g. weights in layers
init = tf.global_variables_initializer()

# Save the current model state
saver = tf.train.Saver()
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# Loss and training (= backward pass) for exclusive labels
loss = tf.losses.softmax_cross_entropy(labels_placeholder,
    output)
optimizer =  tf.train.AdamOptimizer(LEARNING_RATE)
train_op = optimizer.minimize(loss)

# Create summary object for TensorBoard
summary = tf.summary.merge_all()

# Initialize all variables in graph, e.g. weights in layers
init = tf.global_variables_initializer()

# Save the current model state
saver = tf.train.Saver()
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# Loss and training (= backward pass) for exclusive labels
loss = tf.losses.softmax_cross_entropy(labels_placeholder,
    output)
optimizer =  tf.train.AdamOptimizer(LEARNING_RATE)
train_op = optimizer.minimize(loss)

# Create summary object for TensorBoard
summary = tf.summary.merge_all()

# Initialize all variables in graph, e.g. weights in layers
init = tf.global_variables_initializer()

# Save the current model state
saver = tf.train.Saver()
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# Loss and training (= backward pass) for exclusive labels
loss = tf.losses.softmax_cross_entropy(labels_placeholder,
    output)
optimizer =  tf.train.AdamOptimizer(LEARNING_RATE)
train_op = optimizer.minimize(loss)

# Create summary object for TensorBoard
summary = tf.summary.merge_all()

# Initialize all variables in graph, e.g. weights in layers
init = tf.global_variables_initializer()

# Save the current model state
saver = tf.train.Saver()
```

Step 4: Build graph

- Compose all operations in a TensorFlow graph:

```
# Loss and training (= backward pass) for exclusive labels
loss = tf.losses.softmax_cross_entropy(labels_placeholder,
    output)
optimizer =  tf.train.AdamOptimizer(LEARNING_RATE)
train_op = optimizer.minimize(loss)

# Create summary object for TensorBoard
summary = tf.summary.merge_all()

# Initialize all variables in graph, e.g. weights in layers
init = tf.global_variables_initializer()

# Save the current model state
saver = tf.train.Saver()
```

Step 5: Create session and initialize graph

- Create and run session within the graph:

```
# Create a session
with tf.Session() as sess:
        # Run the initialization operation
        sess.run(init)
```

Step 6: Train graph

- Running a session with an operation from the graph will evaluate all operations needed for the given one:

```
# We run the session and evaluate all data in the graph
# up to the loss operation
# loss_value is the current result for the evaluated loss
# feed_dict is the current batch of data fed into the graph
_, loss_value = sess.run([train_op, loss], feed_dict=feed_dict)
```

Step 7: Validate/test graph

- Same as before: Running a session with an operation from the graph will evaluate all operations needed for the given one:

```
# For evaluation we only run the prediction operation and/or
# the loss operation
# prediction returns the prediction for the current batch
# loss_value is the current result for the evaluated loss
# feed_dict is the current batch of data fed into the graph
prediction, loss_value = sess.run([norm_prediction, loss],
    feed_dict=feed_dict)
```

Step 8: Monitoring with TensorBoard

- First serialize data during training, e.g. loss value

```
# Add a scalar summary for the snapshot loss
tf.summary.scalar('loss', loss)
```

Step 8: Monitoring with TensorBoard

- First serialize data during training, e.g. loss value

```
# Add a scalar summary for the snapshot loss
tf.summary.scalar('loss', loss)
```

- After training: launch TensorBoard
  - Command: **`tensorboard - -logdir=path/to/log-directory`**
  - Navigate browser to **localhost:6006**

The code skeleton

- In contrast to previous exercises, there is a code skeleton

➔ First task: Make yourself familiar with the existing code

The code skeleton

- In contrast to previous exercises, there is a code skeleton

→ First task: Make yourself familiar with the existing code

- You will have to adapt four classes for the mandatory part:
  - `train.py`: Adapt hyper parameters if necessary, add loss function
  - `train/trainer.py`: Implement actual training, validation and stopping criteria
  - `model/alexnet.py` and `model/resnet.py`: Implement architectures

The code skeleton

- In contrast to previous exercises, there is a code skeleton

→ First task: Make yourself familiar with the existing code

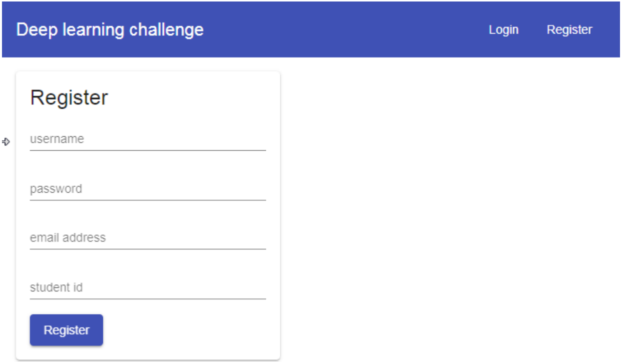
- You will have to adapt four classes for the mandatory part:
  - `train.py`: Adapt hyper parameters if necessary, add loss function
  - `train/trainer.py`: Implement actual training, validation and stopping criteria
  - `model/alexnet.py` and `model/resnet.py`: Implement architectures
- Additional important files:
  - `data/dataset.py` and associated files: Responsible for data loading and augmentation
  - `evaluation/evaluation.py` and `evaluation/measures.py`: Allows monitoring training

Submission to online tool

- After training run, model is automatically saved
- Online submission tool will be made available **by the end of the week**
- Website: https://lme156.informatik.uni-erlangen.de/dl-challenge
- **Only available from within the university network**
- Same teams (max. 2) as before allowed

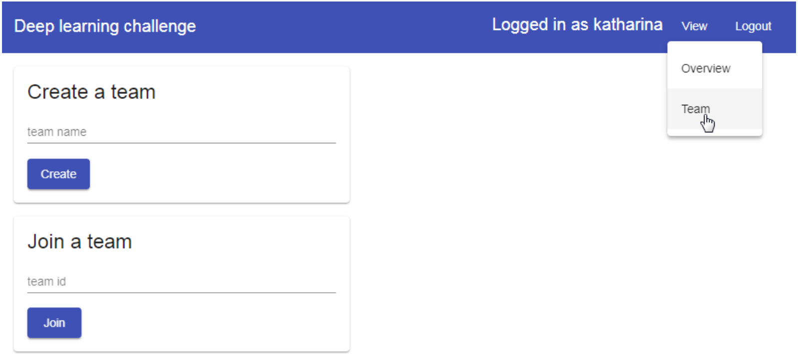
Submission to online tool: Registration

Register with your email and student id.



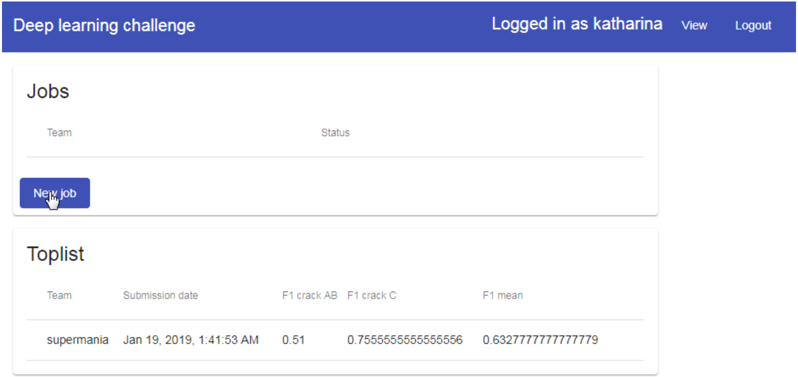
Submission to online tool: Team

If you work in a team: One of you has to create a new team, the other has to join.



Submission to online tool: Submit model

Submit trained models (zip-file generated by train.py) by uploading them. You may submit multiple models.



THE CHALLENGE

Improve on the baseline by AlexNet/ResNet:

- Adapt architectures/try out new architectures
- Pretraining ?
- Regularization ?
- Data augmentation ?
- Use your creativity!
- Best model from each team will be tested on independent data after March 01
- **Best participants will receive a winner's certificate and a prize!**

THE CHALLENGE

Improve on the baseline by AlexNet/ResNet:

- Adapt architectures/try out new architectures
- Pretraining ?
- Regularization ?
- Data augmentation ?
- Use your creativity!
- Best model from each team will be tested on independent data after March 01
- **Best participants will receive a winner's certificate and a prize!**
- **May the best machine learners win!**