TDT4195: Visual Computing Fundamentals

Image Processing - NE + Project

Håkon Hukkelås
haakohu@stud.ntnu.no
Department of Computer Science
Norwegian University of Science and Technology (NTNU)

October 24, 2018

- **Delivery deadline: November 21, 2018 by 2000.**
- **This project count towards 15% of your final grade.**
- The grade distribution is the following:
  - **Numerical Example:** 6%
  - **Project:** 9%
- **You will hold a presentation of your work on November 22 or November 23, 2018**. Timeslots will be allocated closer to the deadline.
- You can work on your own or in groups of up to 3 people.
- Deliver your solution on *Blackboard* before the deadline.
- Upload your code as a single ZIP file. All code (including numerical example and the mini-project) should be in a **SINGLE** ZIP file.
- Upload your presentation as a PDF file before the deadline.
- You can choose what language and frameworks you use. We recommend Python with Keras/OpenCV. Matlab with relevant toolboxes is also a possibility.
- The delivered code is taken into account with the evaluation. Ensure your code is well documented and as readable as possible.

1 Introduction

This is the only assignment for the Image Processing part in TDT4195, and it will consist of two parts, a numerical example(NE), and a mini-project. It is encouraged to work in groups for both parts. **The same group has to be used for both the numerical example, and the mini-project.**

The first part consists of one or more numerical example(s), where we expect you to give a practical example utilizing the theory presented in class (Image Processing Part). It is expected that students working alone make at least one numerical example, and people working in a group to make at least two numerical examples.

The second part is a mini-project, where you can choose between two project options. **The first option** is based on neural networks and your task is to build a neural network to classify objects. **The second option** is based on more traditional image processing techniques, and here you are supposed to build a pipeline that is able to spot lane lines on the road.

In these tasks you are free to explore several techniques related to image processing. You are encouraged to explore techniques not mentioned in this document, the example code provided or the theory lectures. It is possible to use techniques not covered in this course, but we expect you to understand the underlying theory behind the methods, and we might ask about them during your presentation.

2 Python Setup Instructions

In the assignment folder, and the course website we provide a short tutorial on how to set up your python environment. It is called "Python Setup Instruction". We recommend you to use Anaconda to setup your python environment, which simplifies installing dependencies required for this project.

3 Numerical Example

For this part you are free to select a topic covered in lecture. The requirement for this part is that you create a practical numerical example of the theory. Examples of numerical examples have been given in lectures. Examples of this can be:

- Numerical example of forward/backward propagation
- Visualization of weights in a fully-trained convolutional neural network
- Filtering in the frequency domain
- Morphological Image Processing
- Edge detection

Again, this are examples of possible numerical examples. We encourage in trying something else than this, and apply the theory learned in class to practical examples.

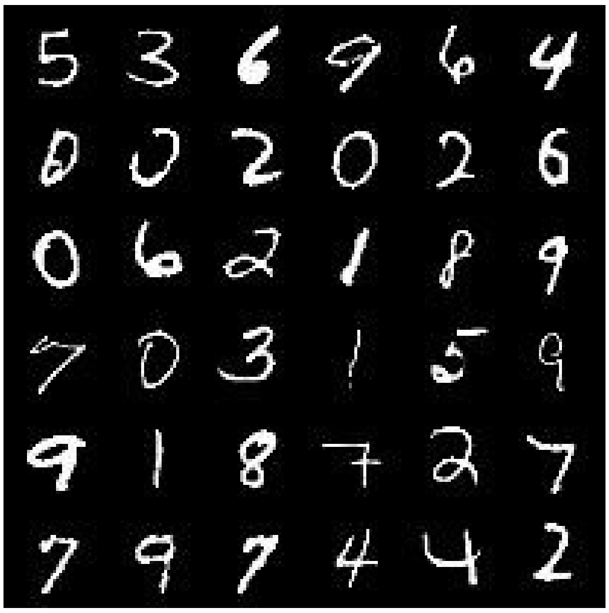
Requirements:

- (Single Student): Create one numerical example.
- (Groups): Create two numerical examples.
- Deliver your numerical example in a PDF file. If you did your numerical example in a jupyter notebook, export this as a PDF file with executed output cells and deliver this (preferred way of delivery).
- If you implement any code for your numerical example(s), include this in your delivery as a ZIP file.

4 Project option 1: Neural networks and deep learning

4.1 Datasets

In this project you are going to start classifying digits in the well-known dataset MNIST. The MNIST dataset consists of 70, 000 handwritten digits, split into 10 object classes (the numbers 0-9). The images are 28x28 grayscale images, and every image is perfectly labeled. The images are split into two datasets, a training set consisting of 60, 000 images, and a testing set consisting of 10, 000 images. State-of-the-art methods are extremely successful on this dataset, reaching test accuracy above 99.75%. Examples of images in this dataset can be seen in Figure 1a


(a) Examples of digits in the MNIST dataset.

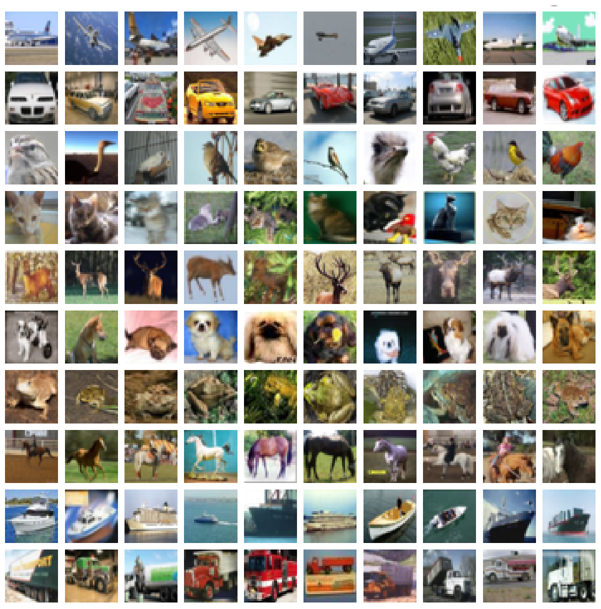

(b) Examples of objects in the CIFAR10 dataset.

Figure 1: Datasets to be used in this assignment

The second dataset you are going to explore is the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Examples of images in this dataset is seen in Figure 1b

4.2 Recommended language and framework: Python with Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It was developed with a focus on enabling fast experimentation. With this assignment we include an introduction to Keras on the MNIST dataset which you can use to get started with your project. If you want to further read up on Keras, the documentation provided with the framework is a good tool: keras.io

Example code: With the assignment files we provide a jupyter notebook file to get you started with Keras. Here we give an example on very simple data exploration, model building, network training pipeline, and model analysis on the MNIST dataset.

4.3 MNIST

In this project you will start exploring Keras on the MNIST dataset.

4.3.1 Initial Data analysis

The first step before building your model is to get to know your dataset, and perform initial analysis of your dataset. The example code shows how to do some of this analysis. Some questions to explore is:

- How many examples do I have in my dataset?
- What is the shape of my input images?
- How many classes do I have?
- What is the class distribution in my dataset?
- Is the testing set similar to the training set?

We encourage you here to find other potential statistics to further explore the dataset here.

4.3.2 Data pre-processing

The next step is usually **data pre-processing**. This includes splitting our training set into a training and a validation set. We want to use a validation set to validate our model, and keep the test set for testing purposes only. The testing set is only for final evaluation, while we use the validation set to tune different hyperparameters in our model. A *hyperparameter* is a configuration of our model that we are unable to learn, and we are required to define this value before we train our model. Examples of this is number of hidden units in a layer. This quora post goes into more detail about what a hyperparameter is.

Further data pre-processing includes normalization of our data. Normalization of our data is a crucial step for neural networks, which helps the convergence rate of the training process, restricts the input values to a certain range, and zero-centers our input data. We recommend you to use pixel-wise normalization (this is shown in the example code provided with the assignment).

4.3.3 Model construction

This is the final step before we start training and evaluating our model. Here we usually choose our network architecture, the loss function we want to optimize for, and the optimization algorithm.

Network architecture: For the MNIST dataset we expect you to build a fully-connected neural network. The example code provides a simple fully connected network which achieves about 91% accuracy on the test set. Further build upon this to get the network architecture shown in Table 1. This network introduces a single hidden layer with 128 hidden units. The ReLU activation function is the rectified linear unit. You should expect to achieve an accuracy of about 96%

Layer Type	Number of Hidden Units	Activation Function
Dense	128	ReLU
Dense	10	Softmax

Table 1: A simple multi-layer fully connected neural network

Loss function: The loss function is usually dependent on the task, and not the network architecture. The standard(and almost only used) for multi-class object classification is the Cross entropy loss function (categorical_crossentropy in Keras).

Optimization algorithm: The choice of which optimization algorithm to use is currently a hugely researched area in deep learning. The standard stochastic gradient descent(SGD) is very popular, but algorithms such as Adam and RMSProp have been heavily used the last years. We recommend you to explore different optimization algorithms for your models, but Adam is often a good default choice.

4.3.4 Training and evaluation

The final step is to train your model and evaluate it on both the validation set and the testing set.

Tracking validation loss: You should track the loss and accuracy on your validation set during training. This can help you get an idea on how good your model is generalizing, and see if you are overfitting to your training set. In the example code we have shown how you can keep track of this, and also see the history of the training when it is finished.

Analysis of performance: Here you want to analyze your model, and visualization of accuracy, loss, etc is some useful tools. In the example code we present a simple method to analyze how our model is generalizing, and we notice a significant overfitting on the test data. Is there any way you can improve this? Other metrics you might want to track is F1-score, recall, precision. Other useful metrics to visualize is often confusion matrix, and precision-recall curve. We encourage you to explore more metrics than what is in the example code.

Hyperparameter tuning: This is where you might want to tune your hyperaparameters, and train the model again. Some hyperparameters to tune can be batch size, learning rate, network architecture.

4.3.5 MNIST Tasks:

These tasks are what we expect you to complete in your project as a **minimum**. We recommend you to further explore and test out different network architecture, analysis methods, training algorithms, network layer types, hyperaparameters, additional data augmentation, etc.

- Perform an initial analysis on the dataset.
- Implement the network in Table 1.
- Find the simplest fully-connected network that achieves more than 90% on the MNIST test-set. (In terms of number of parameters).
- Find a model that reaches the highest test accuracy on the MNIST test set.
- Perform a final analysis and evaluation of your best model.

4.4 CIFAR-10

On the CIFAR-10 dataset you are expected to perform a lot of the same tasks as on the MNIST dataset. The main difference in this task is that you will implement a Convolution Neural Network(CNN). CNN's are a powerful tool for image processing due to its properties such as translational invariance, scale invariance and sharing of weights across the image.

4.4.1 Model Construction

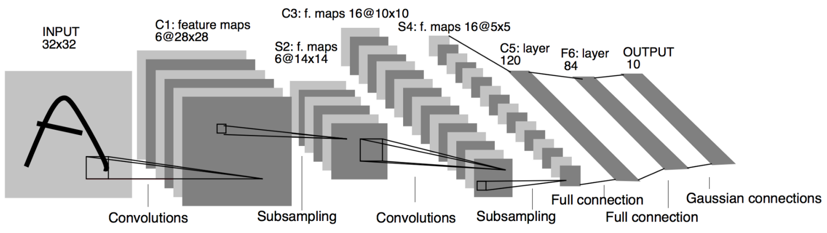


Figure 2: A version of LeNet presented by Yann LeCunn in 1998. Previous state-of-the-art method for handwritten digit recognition.

The model we recommend you to start with is the model presented in Table 2. This is very similar to the sucessful network from Yann LeCunn that was released in 1998 for handwritten digit recognition. One version of LeNet can be seen in Figure 2. This network is expected to achieve about 70% on the CIFAR-10 test set.

Layer Type	Number of Hidden Units / Number of filters	Activation Function
Conv2D	32	ReLU
Conv2D	64	ReLU
Dense	64	ReLU
Dense	10	Softmax

Table 2: A simple multi-layer convolutional neural network

4.4.2 Implementing a known model

Keras provides several models that are finished training on several datasets. Using these weights we can load the model, and finetune the model for our purposes. To finetune our model we freeze all weights in the model, except a few layers. This is known as *transfer learning*, and often is beneficial to improve our accuracy on CIFAR-10.

```
vgg = keras.applications.vgg16.VGG16(include_top=False,
                                     weights='imagenet',
                                     input_shape=X_train.shape[1:])
for layer in vgg.layers[:-2]:
    layer.trainable = False
model = Sequential()
model.add(vgg)
model.add(Flatten())
model.add(Dense(10, activation="softmax"))
```

Listing 1: Loading the VGG16 model to keras, pretrained on ImageNet.

Transfer learning in Keras is very simple to implement, and is shown in Listing 1. Here we load the VGG16 network, which is trained on ImageNet. ImageNet is a dataset consisting of millions of images, and usually split into 1000 classes. We want to load the VGG16 model, and train it on the CIFAR10 dataset. To implement this we add a Flatten layer, and a output layer with ten neurons. In the example here we train only the last two layers (plus our added layer), and keep the other weights constant. The include_top = False removes the last layer in the network. **NOTE:** If you are going to try this, it might be needed to tune down the batch size.

4.4.3 CIFAR-10 Tasks

These tasks are what we expect you to complete in your project as a **minimum**. We recommend you to further explore and test out different network architecture, analysis methods, training algorithms, network layer types, hyperaparameters, additional data augmentation, etc.

- Perform an initial analysis on the dataset.
- Train and evaluate your best fully connected neural network model from MNIST on the CIFAR10 dataset.
- Implement the network in Table 2.
- Find the simplest CNN that achieves more than 70% on the CIFAR10 test set. (In term of number of parameters).
- Find a model that reaches the highest test accuracy on the CIFAR10 test set.
- *(For groups)* Implement a known model on CIFAR10 (eg: VGG16, resnet50, etc) and train and evaluate your model on CIFAR10. This can be done with transfer learning.

4.5 Delivery

- Deliver your code in a ZIP file.
- If you implemented your solution in a jupyter notebook, include this in the ZIP file, but also export it as a PDF file and deliver this as well.
- Deliver your final presentation as a PDF file.
- (*For Groups*) A short summary of what each group member contributed to the project.

4.6 Final Presentation Content

This is only a suggestion in what to include in your presentation. Remember, when we are evaluating you, we want to interpret your understanding of the content presented in this class. Eg: Using a lot of time on results gives little insight to us, but analysis of your solution is more valuable for the evaluation.

Present a short summary of your initial analysis on the CIFAR-10 dataset. Present your simplest and best model for the MNIST and CIFAR-10 datasets, and quickly summarize the results. For your best model on CIFAR-10 discuss the reason behind your network architecture, hyperparameters, and show analysis of your models performance. Is there any weaknesses with your solution? Was there anything you tried that didn't work?

(*For Groups*) Shortly present your implementation of a known model on CIFAR-10, and discuss the results. How did it perform on the CIFAR-10 dataset compared to your best model? Did you have any problems in the implementation? Is there any weaknesses with this solution?

5 Project option 2: Lane Line Detection

This project will focus on more traditional image processing techniques to detect lane lines in videos. Your goal is to piece together a pipeline to detect the line segments in the image, then average/extrapolate them and draw them onto the image for display (as below). Once you have a working pipeline, try it out on a video stream. We expect the final output of your pipeline to be similar to Figure 3.

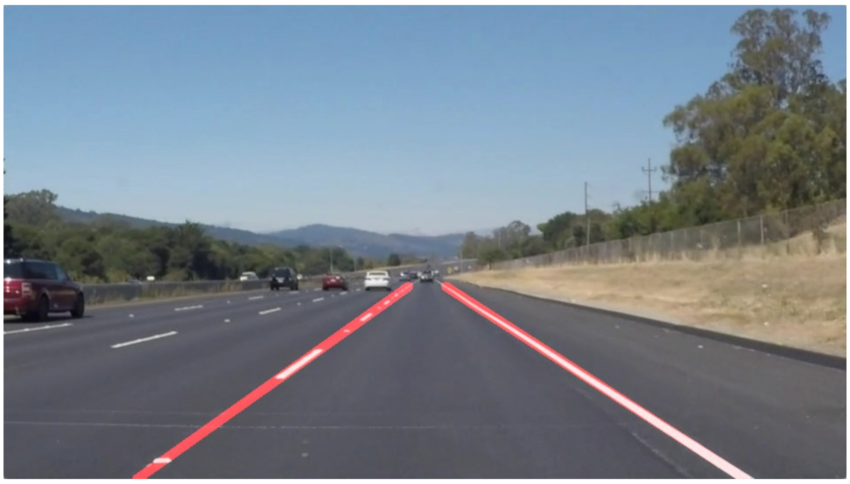


Figure 3: Expected output of the finished pipeline

In this project you are completely free to choose what tools you want to use, but we will come with some recommendation to what a potential working pipeline will look like.

This project has six images, and three videos to test your pipeline. We expect your pipeline to work well on all the images, and two of the videos. The final video "challenge.mp4" is here to present a challenge for those who want to further improve their pipeline.

5.1 Recommended framework: Python with OpenCV

OpenCV (Open Source Computer Vision Library) is a library consisting of most of the image processing techniques presented in class, and gives you a highly efficient and simple library to implement the different techniques. We highly recommend you to use this library.

Example code: We include some starter code to get you familiar with OpenCV, numpy, matplotlib and other libraries in python. This is written in a jupyter notebook, and is named LLDetection_Example.ipynb.

5.2 Recommended pipeline layout

This is a suggested pipeline layout that works for thresholding lane lines, but you are encouraged to be creative in modifying the different segments. This task has a lot of solutions, and everything that works well on the given images and videos will be accepted.

5.2.1 Threshold and extract lane lines

In this part you want to threshold and separate the lane lines from the rest of the image. There exists several techniques that works well on this problem. In the provided code we present a method that finds edges in the Y-direction of the image, and this method is able to extract it well on the test images. We encourage you to explore different methods to extract the lane lines, as this is a crucial part of your pipeline.

5.2.2 Region of Interest masking

As you see in Figure 3, there is large parts of the image that is irrelevant to detect the lane lines. Such as the sky, or the trees on the side. This can be removed by using region of interest masking. Example of how to do this is shown in the provided example code.

5.2.3 Hough Transform

The hough transform is a powerful tool to find straight lines in an image. In the provided code there is a function to find and draw lines based on the hough transform.

5.2.4 Line average/extrapolation

When you have implemented the hough transform you might notice you get several lines for the left and right lane. To generate the expected output, it is required to average or extrapolate each of the lane lines. Can you detect which lines that belongs to the left and right lane? (hint: analyze the derivative).

5.2.5 Final notes

The described pipeline will give you a good start on your project. We encourage to further improve upon this pipeline by introducing several of the concepts taught in the class (or finding other techniques that works well!).

5.3 Project tasks

These tasks are what we expect you to complete in your project as a **minimum**. We recommend you to further explore and test out your pipeline (maybe on some other data collected by you?).

- A robust pipeline that satisfies the requirements below.
- Analysis of the pipeline. What is the strengths of your pipeline? Weakness? Where might it fail?
- Analyze possible improvements to your pipeline.
- *(For groups)*: Try your pipeline on challenge.mp4 video, and evaluate how your pipeline is working on this. What improvements can you implement to make it work on this? (It is not required that your pipeline manages this task flawlessly, but it is a bonus)

Your pipeline will be evaluated on test_videos/solidWhiteRight.mp4, and test_videos/solidYellowLeft.mp4.
What we expect of your output:

- The output video is an annotated version of the input video.
- In a rough sense, the left and right lane lines are accurately annotated throughout almost all of the video. Annotations can be segmented or solid lines.
- Visually, the left and right lane lines are accurately annotated by solid lines throughout most of the video.

5.4 Delivery

Required files:

- Your code in a single .zip file together with the video files.
- Your pipeline result on the videos test_videos/solidWhiteRight.mp4, and test_videos/solidYellowLeft.mp4. Should be saved in the folder result_videos, in .mp4 format.
- (For groups) Your pipeline result on the challenging video, test_videos/challenge.mp4. Should be saved in the folder result_videos, in .mp4 format.

- Deliver your final presentation as a PDF file.
- (*For Groups*) A short summary of what each group member contributed to the project.

5.5 Final Presentation Content

This is only a suggestion in what to include in your presentation. Remember, when we are evaluating you, we want to interpret your understanding of the content presented in this class. Eg: Using a lot of time on results gives little insight to us, but analysis of your solution is more valuable for the evaluation.

In your final presentation you should clearly explain the stages of your pipeline, and show the results of the pipeline. Explain the approach you took, and the decisions you took during the project. You should discuss and analyze your pipeline; strengths and weaknesses of your pipeline (eg: where might it fail?), techniques you tried out but failed, etc.

(*For groups*) Present your result on the challenge.mp4 video, and discuss how your pipeline is working on this. Discuss potential improvements, flaws in the pipeline, etc.

6 Presentation

The final presentation will be a significant part of your final grade for both the numerical example and the mini-project. In the presentation you are expected to present **ONE** numerical example, and your mini-project. Students working alone will have 9 minutes to present, groups of two will have 10 minutes to present, and groups of three or more will have 11 minutes to present. **We will stop any group using more then the maximum presentation time.** For general guidelines for presenting a project, Professor Charles Elkan has some good advice. Date and timeslots will be decided closer to the deadline day.