Cultural Data Science

Visual Analytics Exam 2021

Johan Kresten Horsmans
AU ID: au618771

School of Communication and Culture,
Aarhus University
Jens Chr. Skous Vej 2, 8000 Aarhus, Denmark

May 20th, 2021

Regarding code:

This portfolio submission contains the written part of my exam for Visual Analytics. For the accompanying code, please refer to the main repository at: https://github.com/JohanHorsmans/cds-visual-exam-2021. Throughout the assignment, the repository structure is described as it appears before downloading the additional data specified in the bash-script below.

To run the assignments, you need to go through the following steps in your bash-terminal to download the needed data and configure a virtual environment with the needed prerequisites for all assignments.

NOTE: The code and scripts below will only work for Linux, OS X and Worker02:

```
cd {directory where you want the assignment saved}
git clone https://github.com/JohanHorsmans/cds-visual-exam-2021.git
cd cds-visual-exam-2021
bash create_visual_venv.sh
source visual_venv/bin/activate
bash data_download.sh
```

Contents

1 Assignment 3

1.1 Github link

My code for assignment 3 can be found here: https://github.com/JohanHorsmans/cds-visual-exam-2021/tree/main/assignment_3

1.2 Official description from instructor

ASSIGNMENT 3: Finding text using edge detection

The purpose of this assignment is to use computer vision to extract specific features from images. In particular, we're going to see if we can find text. We are not interested in finding whole words right now; we'll look at how to find whole words in a coming class. For now, we only want to find language-like objects, such as letters and punctuation.

Download and save the image at the link below:

https://upload.wikimedia.org/wikipedia/commons/f/f4/%22We_Hold_These_Truths%22_at_Jefferson_Memorial_IMG_4729.JPG

Using the skills you have learned up to now, do the following tasks:

- Draw a green rectangular box to show a region of interest (ROI) around the main body of text in the middle of the image. Save this as **image with ROI.jpg**.
- Crop the original image to create a new image containing only the ROI in the rectangle. Save this as **image_cropped.jpg**.
- Using this cropped image, use Canny edge detection to 'find' every letter in the image
- Draw a green contour around each letter in the cropped image. Save this as **image_letters.jpg**

Tips

- Remember all of the skills you've learned so far and think about how they might be useful
- This means: colour models; cropping; masking; simple and adaptive thresholds; binerization; mean, median, and Gaussian blur.
- Experiment with different approaches until you are able to find as many of the letters and punctuation as possible with the least amount of noise. You might not be able to remove all artifacts - that's okay!

Bonus challenges

- If you want to push yourself, try to write a script which runs from the command line and which takes any similar input (an image containing text) and produce a similar output (a new image with contours drawn around every letter).

General instructions

- For this exercise, you can upload either a standalone script OR a Jupyter Notebook
- Save your script as edge_detection.py OR edge_detection.ipynb
- If you have external dependencies, you must include a requirements.txt
- You can either upload the script here or push to GitHub and include a link - or both!
- Your code should be clearly documented in a way that allows others to easily follow along
- Similarly, remember to use descriptive variable names! A name like cropped is more readable than crp.
- The filenames of the saved images should clearly relate to the original image

Purpose
This assignment is designed to test that you have a understanding of:

- how to use a variety of image processing steps;
- how to perform edge detection;
- how to combine these skills in order to find specific features in an image

1.3 Methods

The problem in this assignment relates to preprocessing images and then extracting specific features/edges in the image. To solve the task, I started by preprocessing the image with various cv2-functions. To find the edges, I used the *findContours*-function from cv2. As an added bonus, I also used pytesseract to convert the text in the image to a string and print the text in the terminal. To solve the bonus assignment, I used argparse to enable the user to specify arguments from the terminal. With argparse, I made it possible for the user to specify their own image path with an argument called --image_path. In the data-folder in the repository, I included an image titled "Pure_text.png" which the user can use to test the --image_path argument.

1.4 Usage

1.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section is this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 3:

```
cd {root directory (i.e. cds-visual-exam-2021}
cd assignment_3
python3 edge_detection.py
```

Type: `python3 edge_detection.py -h` for a detailed guide on how to specify script-parameters.

1.4.2 Repository structure and contents

The repository contains the following folder:

Folder	Description
`data/`	Folder containing data for assignment 3

Furthermore, it holds the following files:

File	Description
`edge_detection.py`	The python script for the assignment
`README.md`	A README-file containing the same information as you are currently reading

1.4.3 Data

The data-folder contains the following files:

File	Description
WHTT.jpg	An image of a wall with the declaration of independence inscribed. Default image for the script.
Pure_text.png	An image with text saying "Pure Text". Can be used to test the --image_path argument in the script.

1.5 Discussion of results

The contours drawn on the image were quite good (see *figure 1*). Nonetheless, we can see that it has not succeeded in exclusively capturing the letters since it also has drawn contours on the brick-lines in the wall where the text is inscribed. When testing the --image_path argument, I experimented with drawing contours on a lot of different images where it achieved similar levels of performance. As such, I believe that my solution is quite robust. Furthermore, my script is fairly successful in converting the image to a string as seen by the following output where there are only a few errors:

"we hold these truths to be self evident that all men are created equal that they are endowed by their creator with certain inalienable rights among these are life liberty and the pursuit of happiness that to securethese rights governments are instituted among men we solemnly publish and declare that these colonies are and of right ought to be free and independent states and for the support of this declaration with a firm reliance on the protection of divine providence we mutually pledge our lives our fortunes and our sacred honour we".

Figure 1: The final contours drawn on the image.

2 Assignment 4

2.1 Github link

My code for assignment 4 can be found here: https://github.com/JohanHorsmans/cds-visual-exam-2021/tree/main/assignment_4

2.2 Official description from instructor

ASSIGNMENT 4: Classifier benchmarks using Logistic Regression and a Neural Network

This assignment builds on the work we did in class and from session 6.

You'll use your new knowledge and skills to create two command-line tools which can be used to perform a simple classification task on the MNIST data and print the output to the terminal. These scripts can then be used to provide easy-to-understand benchmark scores for evaluating these models.

You should create two Python scripts. One takes the full MNIST data set, trains a Logistic Regression Classifier, and prints the evaluation metrics to the terminal. The other should take the full MNIST dataset, train a neural network classifier, and print the evaluation metrics to the terminal.

Tips

- I suggest using scikit-learn for the Logistic Regression Classifier
- In class, we only looked at a small sample of MNIST data. I suggest using fetch_openml() to get the full dataset, like we did in session 6
- You can use the NeuralNetwork() class that I introduced you to during the code along session
- I recommend saving your .py scripts in a folder called src; and have your NeuralNetwork class in a folder called utils, like we have on worker02
- You may need to do some data manipulation to get the MNIST data into a usable format for your models
- If you have trouble doing this on your own machine, use worker02!

Bonus challenges

- Have the scripts save the classifier reports in a folder called out, as well as printing them to screen. Add the user should be able to define the file name as a command line argument (easier)
- Allow users to define the number and size of the hidden layers using command line arguments (intermediate)
- Allow the user to define Logistic Regression parameters using command line arguments (intermediate)
- Add an additional step where you import some unseen image, process it, and use the trained model to predict it's value - like we did in session 6 (intermediate)
- Add a new method to the Neural Network class which will allow you to save your trained model for future use (advanced)

General instructions

- Save your script as lr-mnist.py and nn-mnist.py
- If you have external dependencies, you must include a requirements.txt
- You can either upload the script here or push to GitHub and include a link - or both!
- Your code should be clearly documented in a way that allows others to easily follow along
- Similarly, remember to use descriptive variable names! A name like X_train is (just) more readable than x1.
- The filenames of the saved images should clearly relate to the original image

Purpose This assignment is designed to test that you have a understanding of:

- how to train classification models using machine learning and neural networks;
- how to create simple models that can be used as statistical benchmarks;
- how to do this using scripts which can be executed from the command line

2.3 Methods

This problem of this assignment relates to using predictive models to classify grayscale images of hand drawn numbers and evaluating the model classifications. For the logistic regression classifier, I used the *LogisticRegression*-function from *sklearn.linear_model*. For the neural network, I created a dense network where one can specify the structure and size of the layers between the input and the output as an argument called --layers. The default structure of the neural network is [224, 8, 16, 10]. Note that my script uses the same data for validation and testing. Furthermore, I solved all of the bonus assignments. For the tasks that required specifying arguments from the terminal, I used argparse. For model evaluation, I have chosen to use macro F1-score as my metric:

The F1-score is a metric devised to fuse the relation between model precision and recall into a unified score. The metric is defined as taking the harmonic mean of precision and recall. The reason for using the harmonic mean, rather than the arithmetic mean, is that the harmonic mean of a recall-score of 0 and a precision-score of 100 would result in an F1-score of 0, rather than 50. This is advantageous, since it means that a model cannot achieve a high F1-score by having a high recall or precision by itself. The macro-averaging procedure of the macro F1-score involves calculating the arithmetic mean of the F1-score for each class.

2.4 Usage

2.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section is this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 4:

```
cd {root directory (i.e. cds-visual-exam-2021}
cd assignment_4
python3 lr-mnist.py
python3 nn-mnist.py
```

Type: `python3 lr-mnist.py -h` and `python3 nn-mnist.py -h` for a detailed guide on how to specify script-parameters.

2.4.2 Repository structure and contents

This repository contains the following files:

File	Description
`lr-mnist.py`	The python script for the logistic regression assignment.
`nn-mnist.py`	The python script for the neural network assignment.
`README.md`	A README-file containing the same information as you are currently reading.

2.5 Discussion of results

For the neural network model (with default parameters), I achieved a macro F1-score of 0.89 after training for 50 epochs. For the logistic regression model, I achieved a macro F1-score of 0.91 using a tolerance of 0.1 and no penalty. When testing the models with custom image-files the performance dropped drastically indicating that the models are overfitting to the specifics of the MNIST-dataset, leading to low generalizability.

3 Assignment 5

3.1 Github link

My code for assignment 5 can be found here: https://github.com/JohanHorsmans/cds-visual-exam-2021/tree/main/assignment_5

3.2 Official description from instructor

Multi-class classification of impressionist painters

So far in class, we've been working with 'toy' datasets - handwriting, cats, dogs, and so on. However, this course is on the application of computer vision and deep learning to cultural data. This week, your assignment is to use what you've learned so far to build a classifier which can predict artists from paintings.

You can find the data for the assignment here: https://www.kaggle.com/delayedkarma/impressionist-classifier-data

Using this data, you should build a deep learning model using convolutional neural networks which classify paintings by their respective artists. Why might we want to do this? Well, consider the scenario where we have found a new, never-before-seen painting which is claimed to be the artist Renoir. An accurate predictive model could be useful here for art historians and archivists!

For this assignment, you can use the CNN code we looked at in class, such as the ShallowNet architecture or LeNet. You are also welcome to build your own model, if you dare - I recommend against doing this.

Perhaps the most challenging aspect of this assignment will be to get all of the images into format that can be fed into the CNN model. All of the images are of different shapes and sizes, so the first task will be to resize the images to have them be a uniform (smaller) shape.

You'll also need to think about how to get the images into an array for the model and how to extract 'labels' from filenames for use in the classification report

Tips

- You should save visualizations showing loss/accuracy of the model during training; you should also a save the output from the classification report
- I suggest working in groups for this assignment. The data is quite large and will take some time to move over to worker02. Similarly training the models will be time consuming, so it is preferably to have fewer models on the go.

- You might want to consider a division of labour in your group. One person working on resizing images, one working on extracting labels, one developing the model, etc.
- For reshaping images, I suggest checking out cv.resize() with the cv2.INTER_AREA method
- If you have trouble doing this on your own machines, use worker02
- Don't worry if the results aren't great! This is a complex dataset we're working with

General instructions

- Save your script as cnn-artists.py
- If you have external dependencies, you must include a requirements.txt
- You can either upload the script here or push to GitHub and include a link - or both!
- Your code should be clearly documented in a way that allows others to easily follow along
- Similarly, remember to use descriptive variable names! A name like X_train is (just) more readable than x1

Purpose

This assignment is designed to test that you have a understanding of:

- how to build and train deep convolutional neural networks;
- how to preprocess and prepare image data for use in these models;
- how to work with complex, cultural image data, rather than toy datasets

3.3 Methods

The problem of the assignment relates to classifying complex image data that even most humans would have a hard time classifying correctly. To address the problem, I finetuned an advanced pretrained CNN in the form of the *MobileNetV2*-model developed by Google AI. Besides finetuning parameters, I also expanded the MobileNetV2-model by adding three layers on top; an average pooling layer, a dropout layer (with 40% of the layer inputs set to 0) and, finally a classification layer. A large challenge in finetuning the model consists of preprocessing the images into a compatible format. To do this, I have taken inspiration from my self-assigned project, where I define a function that can easily convert any image into a format compatible with a CNN-architecture. I converted all the paintings to 224x224 images, since this is what the model was originally trained on. I used argparse to enable the user to specify the number of training epochs from the terminal. Note that my script uses the same data for validation and testing.

3.4 Usage

3.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section is this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run assignment 5:

```
cd {root directory (i.e. cds-visual-exam-2021}
cd assignment_5
python3 cnn-artists.py
```

Type: `python3 cnn-artists.py -h` for a detailed guide on how to specify script-parameters.

3.4.2 Repository structure and contents

This repository contains the following files:

File	Description
`cnn-artists.py`	The python script for the assignment
`README.md`	A README-file containing the same information as you are currently reading

3.5 Discussion of results

Using the CNN, I achieve a macro F1-score of 0.65 after training for 20 epochs. For the reasoning behind using macro F1-score as my evaluation metric, see ***section 2.3:*** *Assignment 4 - Methods*. The training curves show that the accuracy and loss for the validation data (i.e. training data) flattens after approximately 10 epochs (see *figure 2*). As such, I argue that more epochs would most likely not result in a better macro F1-score. Even though the results might not seem impressive, I argue that it is a very convincing performance for such a complex task.

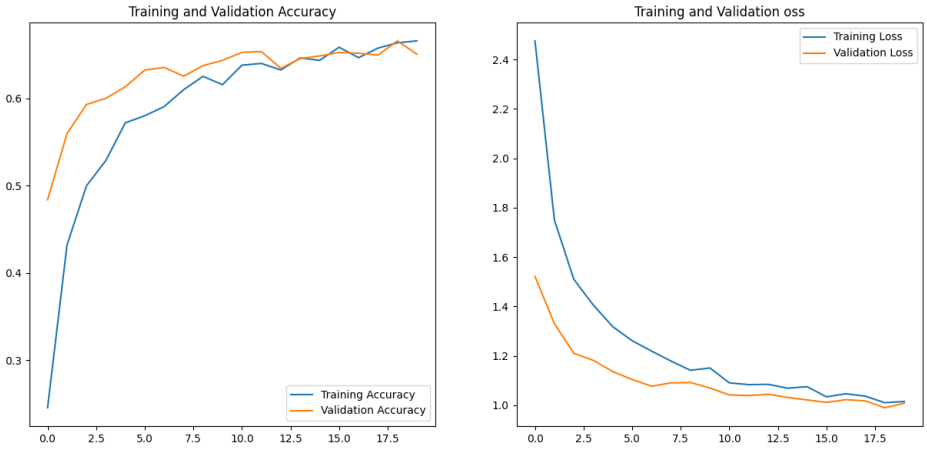

Figure 2: Training curves for the MobileNetV2-model in assignment 5.

4 Self assigned

4.1 Github link

My code for my self assigned project can be found here: https://github.com/JohanHorsmans/cds-visual-exam-2021/tree/main/self_assigned

4.2 Project description

Creating fake Monet-images with style transfer and classifying them with CNN's:

For my self-assigned project, I wish to see if I can use style-transferring to create fake Monet-paintings out of landscape pictures. Furthermore, I want to test if these images are believable enough to trick a CNN-image classifier into classifying them as real Monet-paintings. I have decided to turn in my project as a Jupyter Notebook, since there is a great emphasis on visualizations throughout. Furthermore, the code is not intended to be used for other purposes. Therefore, I deemed it to be the most beneficial format for my project. As such, it is a deliberate choice that I have not converted it to as a .py-script and not a case of lacking abilities.

The notebook consists of the following steps (in rough terms):

1. Creating fake Monet-images out of landscape images using style transfer.
   - The dataset used for the Monet-images (i.e. style-images) is the following: *Impressionist Classifier Data*. I have manually removed the image titled *"9223372032559844173.jpg"*, since it was corrupt.
   - The dataset used for landscape-images (i.e. content-images) is the following: *Landscape Pictures*
2. Saving the fake images.
3. Splitting the data into a testing- and training dataset (containing real- and fake images).
4. Preprocessing the images into a consistent format compatible with a CNN classifier.
5. Building and training a CNN-classifier from scratch.
6. Classifying testing data with the CNN-classifier.
7. Fine-tuning a pretrained *MobileNetV2* model.
8. Using the fine-tuned model to classify testing data.

4.3 Methods

The task is a two-part problem. First, one needs to create a model for carrying out the style transfer and second, one needs to build a classifier capable of distinguishing between the real- and "fake" images. To address the first problem, I initially experimented with using a *VGG19*-model, but it took approximately 1.5 hour per style-transferred image, rendering the task practically undoable. Instead, I used the *magenta/arbitrary-image-stylization-v1-256*-model from tensorflow hub. This model was a lot more efficient and produced mostly convincing results that I deemed to be only moderately worse compared to the VGG19-model (see comparison in *figure 3*). Both models took only a single content- and style image per style transfer.

For the classification task, I first developed a CNN from scratch. This generated mediocre results so I wanted to see if I could find a better and more efficient classifier. To do so I utilized the pretrained *MobileNetV2*-model. Both CNN-models and their predictions are included in the notebook. Note that my notebook uses the same data for validation and testing.

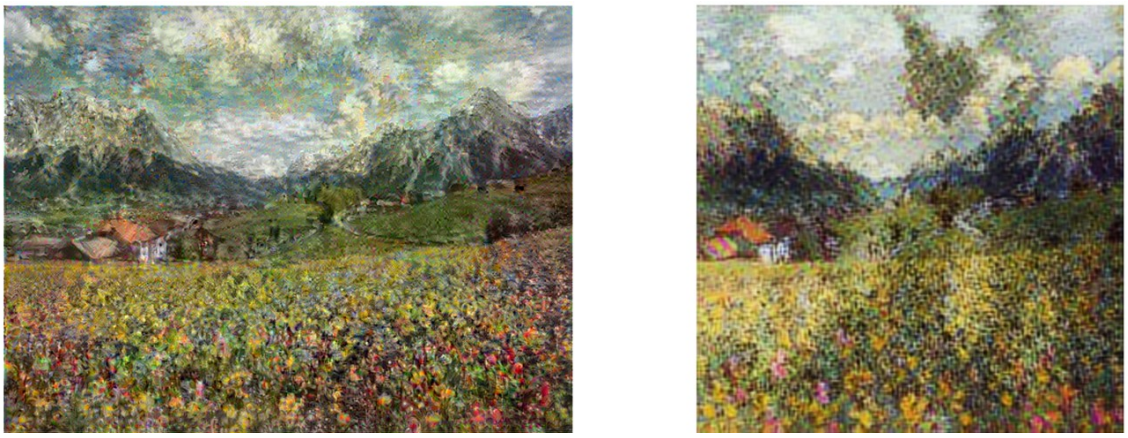

Figure 3: Comparison of the style-transferred output produced by the VGG19-model (left) and the arbitrary-image-stylization-v1-256-model (right).

4.4 Usage

4.4.1 How to run

NOTICE: To run the assignment, you need to have configured and activated your virtual environment. See the section is this PDF titled *"Regarding code"* or the main `README` for a guide on how to do this.

Go through the following steps to run the assignment:

1. Navigate to the "self_assigned"-folder.
2. Open the "self_assigned.ipynb"-file.
3. Make sure the kernel is set to "visual_venv".
   - You can do this by pressing "kernel" → "change kernel" → "visual_venv".

4.4.2 Repository structure and contents

The repository contains the following folder:

File	Description
`raw_data/`	A folder with all the 398 Monet images and 25 handpicked content-images depicting French landscapes.

Furthermore, it holds the following files:

File	Description
`self-assigned-project.ipynb`	The jupyter notebook for the assignment.
`README.md`	A README-file containing the same information as you are currently reading

4.5 Discussion of results

The "homemade" CNN-model (henceforth referred to as model 1) yielded a macro F1-score of 0.64 after training for 50 epochs. By comparison the pretrained MobileNetV2-model achieved a macro F1-score of 0.99 after only 10 epochs of training. As such, it greatly outperforms the initial model on both performance and efficiency. The training- loss and accuracy for model 1 (see *figure 4*), does not seem to flatten, suggesting that more training epochs could potentially improve results. For both models the accuracy for the validation data (i.e. testing data) is higher than the accuracy for the training data. This is true for nearly all epochs (see *figure 4* and *5*). I argue that this is most likely due to the `dropout` layer at the end of both models since, when training, 60% of the dropout input nodes are set to 0 whereas, in testing, all nodes are active and utilized. It seems plausible that this would be the reason behind the more robust classification of the testing data. I tried both removing the dropout-layer and making it less conservative. For both models, this resulted in a performance decrease.

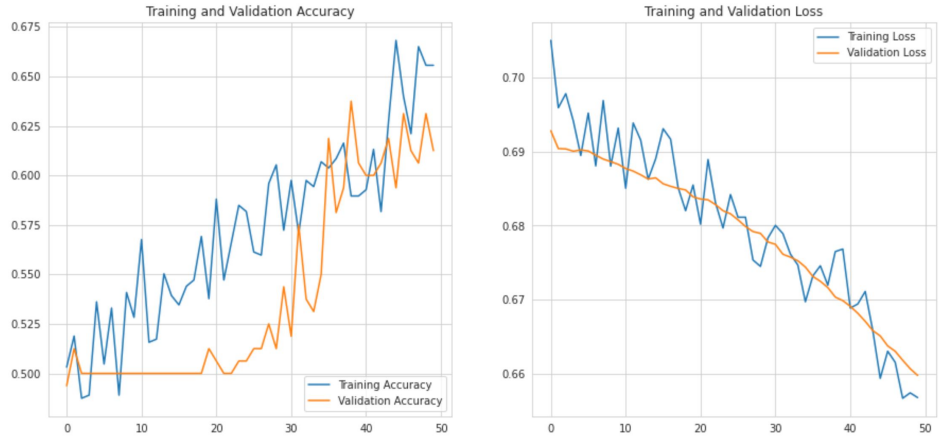

Figure 4: Training curves for model1 in the self-assigned project.

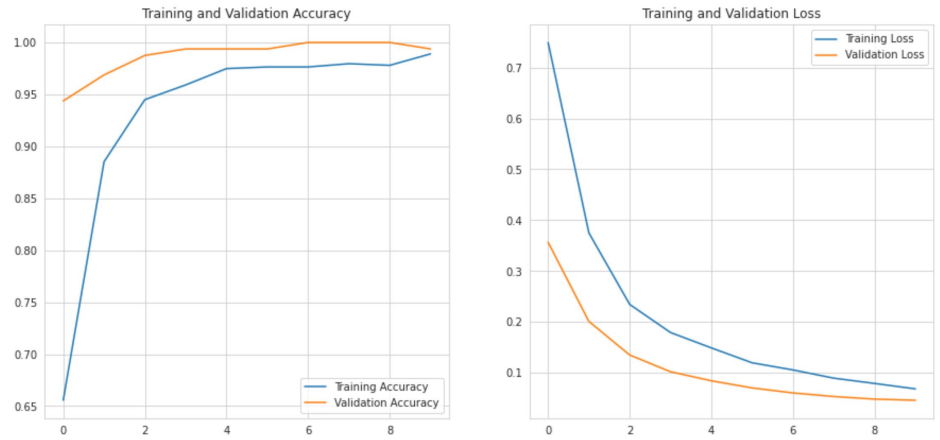

Figure 5: Training curves for the MobileNetV2-model in the self-assigned project.