# Report
# Pattern Recognition

Shekhar “Shakes” Chandra

Version 1.57

Creating and maintaining open source algorithms either for pattern recognition and analysis or in general in the current landscape of big data and artificial intelligent, is important for anyone working with data, whether it be an open source developer, a researcher, an engineer or a data/computer scientist.

In this report, you will choose a recognition problem from a list provided and solve it with the concepts and materials taught in this course. The problems (and associated data) provided are very close to real research problems that are still being solved! It is hoped that you get an opportunity to have a go at real world data crunching and pattern analysis by creating your own solutions to these problems. This will give you an opportunity to practice all the necessary skills required to develop and maintain your own open source project(s) and writing a mini report in the form of a read me file and a GitHub pull-request. These are skills that are highly desirable in many employment opportunities involving machine learning, big data and artificial intelligence.

You will also have an opportunity to learn how to collaborate on an open source project by forking a pre-exiting repository and adding your project to it via a pull-request. You will then be required to write the documentation and instructions for that algorithm. You will be required to complete the necessary software engineering tasks to complete the report with tasks such as design, commenting, pull requests etc. Your algorithm will directly contribute to the open source PatternAnalysis library on GitHub. Upon submission, the teaching team will provide a professional code review of your submission, so that you get an idea of what to expect when doing such a submission at a work place or for an open source project.

## 1 The Report

### 1.1 Purpose of the Assessment

It is now common place for start-ups, industry and research to use open source (and particularly source versioning) workflows to develop and solve problems that require any form of pattern recognition and artificial intelligence. This is primarily because such solutions require algorithms, but also because it is dominated by deep learning, which is almost entirely based on open source software and its principles.

This assessment attempts to simulate a real world scenario when joining a team to do research and development into algorithms. Once completed it will provide you with the minimal professional development required to work on a small team using Git and deep learning. The teaching team are experts in the models and projects below and will provide valuable feedback to enable to grow into the field.

### 1.2 Objective

Your objective is to solve a recognition problem for your choice (see below), document your solution and code, then submit it for code review and acceptance into the course repository. The problems are divided into three difficulty levels where:

- [Easy Difficulty] is capped at 10/20 marks for the implementation portion of the marking criteria
- [Normal Difficulty] is capped at 15/20 marks for the implementation portion marking criteria
- [Hard Difficulty] is not capped for the implementation portion marking criteria

## 1.3 Projects Available

The following problems (and associated requirements) are available with the difficulties indicated noting that those marked for updating may change the dataset of model required in the initial week of release:

**The tasks/projects below are finalised. Please check Blackboard for latest version of this file.**

1. Segment the HipMRI Study on Prostate Cancer (see Appendix for link) using the processed 2D slices (2D images) available here with the 2D UNet [1] with all labels having a minimum Dice similarity coefficient of 0.75 on the test set on the prostate label. You will need to load Nifti file format and sample code is provided in Appendix B. [Easy Difficulty]

2. **[Maybe be Updated]** Create a suitable multi-layer graph neural network (GNN) model to carry out a semi supervised multi-class node classification using Facebook Large Page-Page Network dataset [2] with reasonable accuracy. (You may use this partially processed dataset where the features are in the form of 128 dim vectors.) You should also include a TSNE or UMAP embeddings plot with ground truth in colors and provide a brief interpretation/discussion. [Normal Difficulty]

3. Detect lesions within the ISIC 2017/8 data set (see Appendix for link) with the or a detection network such as the YOLOv7 or newer (pre-trained modela are allowed) with all detections having a minimum Intersection Over Union of 0.8 on the test set and a suitable accuracy for classification. [Normal Difficulty]

4. Segment the (downsampled) Prostate 3D data set (see Appendix for link) with the 3D Improved UNet3D [3] with all labels having a minimum Dice similarity coefficient of 0.7 on the test set. See also CAN3D [4] for more details and use the data augmentation library here for TF or use the appropriate transforms in PyTorch. You may begin with the original 3D UNet [5]. You will need to load Nifti file format and sample code is provided in Appendix B. [Normal Difficulty - 3D UNet] [Hard Difficulty - 3D Improved UNet]

5. Classify Alzheimer's disease (normal and AD) of the ADNI brain data (see Appendix for link) using one of the latest vision transformers such as the GFNet [6] set having a minimum accuracy of 0.8 on the test set. [Hard Difficulty]

6. Create a classifier based on Siamese network [7] to classify the ISIC 2020 Kaggle Challenge data set (normal and melanoma) having an accuracy of around 0.8 on the test set (see Appendix for links). [Hard Difficulty]

7. Create a generative model of the HipMRI Study on Prostate Cancer using the processed 2D slices (2D images) available here with the using a VQVAE [8] or VQVAE2 [9] that has a "reasonably clear image" and a Structured Similarity (SSIM) of over 0.6. [Hard Difficulty]

8. Create a generative model of one of the ADNI brain data set (see Appendix for links) using either a variant of StyleGAN [10]/StyleGAN2 [11] or Stable Diffusion [12] that has a "reasonably clear image". You should also include a TSNE or UMAP embeddings plot with ground truth in colors and provide a brief interpretation/discussion. See recent UQ work for hints of how to incorporate labels in style [13]. [Hard Difficulty]

A listing and links to the various data sets is provided in Appendix A. Additional recognition problems may become available and will be added to this list. You are also able to propose your own network for the above problems or an entirely different problem to the course coordinator for approval. The teaching staff will decide what constitutes "reasonably clear image" and the SSIM is not strict, but both will be negotiable with the staff depending the difficulty of the problem.

## 2 Submission

The submission of the final report will be done via a GitHub pull-request and a Turn-it-in submission. The pull request should contain

1. the code of your algorithm and test script(s) written to run the algorithm (20 Marks),
2. commit log to provide evidence of originality (5 Marks, Pass Hurdle)
3. documentation via a README.md file (10 Marks)
4. pull-request, with clear description, completed properly (5 Marks, Pass Hurdle)

The Turn-it-in submission should be a PDF version of the README.md file. **You are required to make the pull request to receive any marks for the assessment. You must complete a pull-request (with a working version of the algorithm committed) at least once before the assessment deadline.** The following sections describe the individual tasks to be completed in detail.

## 3 Checklist

The following steps should be followed to complete this assessment task:

1. fork the PatternAnalysis library on GitHub
2. checkout the topic-recognition branch within your fork
3. choose a problem via section 1 and create a folder for your problem in the 'recognition' folder of the PatternAnalysis trunk with a meaningful name. We suggest using a name that incorporates the name of your model you are implementing appended with your name or ID
4. create a README.md file that describes your report.
5. complete the solution to the recognition problem by building your model, making sure to commit any progress to the topic-recognition branch of your fork
6. Do not commit datasets or model files, your fork and commits should only have code/scripts plus any additional content needed for your README.md file
7. complete the test driver script that calls and runs your algorithm
8. complete the documentation of your algorithm including a summary of your results and model in your README.md file
9. lodge a pull request
10. complete a Turn-it-in submission of the PDF of the README.md file.

## 4 Recognition Problem (20 Marks)

Your implementation must include the following files

1. **"modules.py"** containing the source code of the components of your model. Each component must be implementated as a class or a function
2. **"dataset.py"** containing the data loader for loading and preprocessing your data
3. **"train.py"** containing the source code for **training**, **validating**, **testing** and **saving** your model. The model should be imported from "modules.py" and the data loader should be imported from "dataset.py". Make sure to plot the losses and metrics during training
4. **"predict.py"** showing example usage of your **trained** model. Print out any results and / or provide visualisations where applicable
5. **"README.MD"** to sufficiently document your project (see Section 6).

Note: You may create other helper files such as "utils.py" to better organise your project. You may reuse some of your own elements from a previous demo, but the final model must be substantially different from your previous work.

### Marking Criteria

1. Algorithm solves the respective pattern recognition problem appropriately (5 Marks)
2. Algorithm implemented in TF/PyTorch demonstrated and functions as intended to solve the respective problem (3 Marks)
3. Good design implemented in TF/PyTorch that solves the problem (1 Marks)
4. Commenting (1 Mark)
5. Algorithm is normal or above difficulty while implemented in TF/PyTorch while solving the respective problem (5 Marks)
6. Algorithm is hard difficulty while implemented in TF/PyTorch while solving the respective problem (5 Marks)

## 5 Commit Log (5 Marks, Pass Hurdle)

### Requirements

1. **Ensure that when you are implementing your algorithm to solve the recognition problem, you are committing regularly and keeping a detailed commit log.** You should for example commit whenever you reach any small milestone, such as creating a module or a working test script, completed the algorithm and completed the documentation etc.
2. **You must have a reasonable commit log showing progress and development of your algorithm. You should not commit your entire submission as one commit.** You will not receive any marks if your algorithm only contains a single commit. You may not pass the assessment if your commit log does not provide sufficient evidence of individual work and originality.
3. **Do not commit datasets or model files.** Your fork and commits should only have code/scripts plus any additional content needed for your README.md file.

### Marking Criteria

1. Meaningful commit messages with evidence of individual work and originality (2 Marks)
2. Progressive commits and logical log structure providing evidence of individual work and originality (3 Marks)

## 6 Documentation (10 Marks)

Once your algorithm has been implemented, you will need to provide sufficient comments in your code and documentation of your algorithm in a README.md file.

### Requirements

1. **The readme file should contain a title, a description of the algorithm and the problem that it solves (approximately a paragraph), how it works in a paragraph and a figure/visualisation.**
2. It should also list any dependencies required, including versions and address reproduciblility of results, if applicable.
3. provide example inputs, outputs and plots of your algorithm
4. The read me file should be properly formatted using GitHub markdown
5. Describe any specific pre-processing you have used with references if any. Justify your training, validation and testing splits of the data.

### Marking Criteria

1. description and explanation of the working principles of the algorithm implemented and the problem it solves (5 Marks)
2. description of usage and comments throughout scripts (3 Marks)
3. proper formatting using GitHub markdown (2 Mark)

## 7 Pull Request (5 Marks)

Once your algorithm module, test script and read me file are completed, lodge a pull-request to the Pattern-Analysis repository via the topic-recognition branch. **A pull-request must be made to receive any marks for this assessment. This pull request must have a working version of the algorithm committed.**

You can see a guide on creating pull requests via GitHub here and a guide about pull-requests via GitHub here. See also this tutorial by Atlassian that explains the concept quite well.

### Marking Criteria

1. Creating a pull-request into correct branch with a working and demonstrable version of the algorithm (2 Marks)
2. Incorporating feedback into the pull request (2 Marks)
3. Description of and comments within the pull request (1 Mark)

## 8 Tips and Advice

It is best advised to try an Easy problem first especially if you are not so confident with TF/PyTorch and problems provided. Once working, expand up to the Normal problem. For generative problems, try to get your code working on a simpler data set such as the MNIST first and then attempt it with the data for your problem. You should see the previous years sucessful pull requests and the type of expected content and hints to the problems. **Hard problems should only be attempted if you are extremely confident with your deep learning skills!**

## 9 Academic Integrity and Misconduct

Academic Integrity is a core value of the UQ community and as such high academic integrity expectations apply to assessments. Using online resources for self-learning is fine, however **simply re-using entire blocks of code for them is not acceptable and may result in a violation of the Student Charter as a UQ student and an investigation into Academic Misconduct**. You must follow the guidelines for Academic Integrity found here, which also applies to code. Where possible, you should write your own code and if certain elements are required from online resources should be referenced in your pull-request or code. For more information see the UQ policy here and a tutorial on understanding this issue further go to the official UQ tutorial.

## 10 Conclusion

By completing this report, you will obtain experience in developing and maintaining your own open source project, how to contribute to other open source projects and good practices in algorithmic design and implementation. All the while, you will become even more knowledgeable in Tensorflow/PyTorch and pattern recognition!

## Appendix

## A Datasets

**The following datasets are all available on the Rangpur HPC in the *\/home/groups/comp3710* directory.** The information and details for Rangpur can be found at the link here. You can also find these datasets at the sources below:

1. ISIC 2017/8 dermoscopic imaging challenge data for skin cancer - This is part of the ISIC 2017/8 challenge and comes with segmentation labels and lesion class labels. You can find a processed version of the data uploaded here.

```
Rangpur Path: /home/groups/comp3710/ISIC2018
```

2. ISIC 2020 Kaggle Challenge data set with dermoscopic imaging having image-level class labels (normal or melanoma). See also the Kaggle Challenge page here for more details.

3. The ADNI dataset for Alzheimer's disease. Look at the DOWNLOAD > Image Collections > Advanced Search area to download the data. The preprocessed version of this data set can be found here and has two classes: Alzheimer's disease (AD) and Cognitive Normal (CN). It can be found on Rangpur at:

```
Rangpur Path: /home/groups/comp3710/ADNI
```

4. The HipMRI Study for prostate cancer radiotherapy consisting of Prostate 2D MRI slide data for 2D tasks and Prostate 3D MRI data for 3D tasks of 38 patients with 211 3D MRI volumes. The body outline, bone, bladder, rectum and prostate regions are segmented in 3D by a MR physicist with more than 10 years of experience.

   ```
   Rangpur Path: /home/groups/comp3710/HipMRI_Study_open
   2D Data: /home/groups/comp3710/HipMRI_Study_open/keras_slices_data
   ```

## B Reading Nifti Files

The following code blocks are examples of loading Nifti image files using the Nilearn and Nibabel libraries, which are the standard file format in medical imaging. Listing 1 shows how to load 2D Nifti files and Listing 2 shows how to load 3D Nifti files as numpy arrays.

```python
import numpy as np
import nibabel as nib
from tqdm import tqdm

def to_channels(arr: np.ndarray, dtype=np.uint8) -> np.ndarray:
    channels = np.unique(arr)
    res = np.zeros(arr.shape + ( len(channels),), dtype=dtype)
    for c in channels:
        c =  int(c)
        res[..., c:c+1][arr == c] = 1

    return res

#load medical image functions
def load_data_2D(imageNames, normImage=False, categorical=False, dtype=np.float32, getAffines=False, early_stop=False):
    '''
    Load medical image data from names, cases list provided into a list for each.

    This function pre-allocates 4D arrays for conv2d to avoid excessive memory usage.

    normImage: bool (normalise the image 0.0-1.0)
    early_stop: Stop loading pre-maturely, leaves arrays mostly empty, for quick loading and testing scripts.
    '''
    affines = []

    #get fixed size
    num =  len(imageNames)
    first_case = nib.load(imageNames[0]).get_fdata(caching='unchanged')
    if  len(first_case.shape) == 3:
```

```
        first_case = first_case[:,:,0] #sometimes extra dims, remove
    if categorical:
        first_case = to_channels(first_case, dtype=dtype)
        rows, cols, channels = first_case.shape
        images = np.zeros((num, rows, cols, channels), dtype=dtype)
    else:
        rows, cols = first_case.shape
        images = np.zeros((num, rows, cols), dtype=dtype)

    for i, inName in  enumerate(tqdm(imageNames)):
        niftiImage = nib.load(inName)
        inImage = niftiImage.get_fdata(caching='unchanged') #read disk only
        affine = niftiImage.affine
        if  len(inImage.shape) == 3:
            inImage = inImage[:,:,0] #sometimes extra dims in HipMRI_study data
        inImage = inImage.astype(dtype)
        if normImage:
            #~ inImage = inImage / np.linalg.norm(inImage)
            #~ inImage = 255. * inImage / inImage.max()
            inImage = (inImage - inImage.mean()) / inImage.std()
        if categorical:
            inImage = utils.to_channels(inImage, dtype=dtype)
            images[i,:,:,:] = inImage
        else:
            images[i,:,:] = inImage

        affines.append(affine)
        if i > 20 and early_stop:
            break

    if getAffines:
        return images, affines
    else:
        return images
```

```
def load_data_3D(imageNames, normImage=False, categorical=False, dtype=np.float32, ↘
   getAffines=False, orient=False, early_stop=False):
    '''
    Load medical image data from names, cases list provided into a list for each.

    This function pre-allocates 5D arrays for conv3d to avoid excessive memory ↘
        usage.

    normImage: bool (normalise the image 0.0-1.0)
    orient: Apply orientation and resample image? Good for images with large slice ↘
        thickness or anisotropic resolution
    dtype: Type of the data. If dtype=np.uint8, it is assumed that the data is ↘
        labels
    early_stop: Stop loading pre-maturely? Leaves arrays mostly empty, for quick ↘
        loading and testing scripts.
    '''
    affines = []

    #~ interp = 'continuous'
    interp = 'linear'
    if dtype == np.uint8: #assume labels
        interp = 'nearest'

    #get fixed size
    num =  len(imageNames)
    niftiImage = nib.load(imageNames[0])
    if orient:
        niftiImage = im.applyOrientation(niftiImage, interpolation=interp, scale=1)
        #~ testResultName = "oriented.nii.gz"
        #~ niftiImage.to_filename(testResultName)
    first_case = niftiImage.get_fdata(caching='unchanged')
    if  len(first_case.shape) == 4:
        first_case = first_case[:,:,:,0] #sometimes extra dims, remove
    if categorical:
        first_case = to_channels(first_case, dtype=dtype)
        rows, cols, depth, channels = first_case.shape
        images = np.zeros((num, rows, cols, depth, channels), dtype=dtype)
    else:
        rows, cols, depth = first_case.shape
        images = np.zeros((num, rows, cols, depth), dtype=dtype)

    for i, inName in  enumerate(tqdm(imageNames)):
        niftiImage = nib.load(inName)
        if orient:
            niftiImage = im.applyOrientation(niftiImage, interpolation=interp, scale=1)
        inImage = niftiImage.get_fdata(caching='unchanged') #read disk only
        affine = niftiImage.affine
        if  len(inImage.shape) == 4:
            inImage = inImage[:,:,:,0] #sometimes extra dims in HipMRI_study data
        inImage = inImage[:,:,:depth] #clip slices
        inImage = inImage.astype(dtype)
        if normImage:
            #~ inImage = inImage / np.linalg.norm(inImage)
            #~ inImage = 255. * inImage / inImage.max()
            inImage = (inImage - inImage.mean()) / inImage.std()
        if categorical:
            inImage = utils.to_channels(inImage, dtype=dtype)
            #~ images[i,:,:,:,:] = inImage
            images[i,:inImage.shape[0],:inImage.shape[1],:inImage.shape[2],:inImage.shape[3]] = inImage #with pad
        else:
            #~ images[i,:,:,:] = inImage
            images[i,:inImage.shape[0],:inImage.shape[1],:inImage.shape[2]] = inImage #with pad

        affines.append(affine)
        if i > 20 and early_stop:
            break

    if getAffines:
        return images, affines
    else:
        return images
```

## References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, ser. Lecture Notes in Computer Science, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[2] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale Attributed Node Embedding," *arXiv:1909.13021 [cs, stat]*, Mar. 2021, arXiv: 1909.13021. [Online]. Available: http://arxiv.org/abs/1909.13021

[3] F. Isensee, P. Kickingereder, W. Wick, M. Bendszus, and K. H. Maier-Hein, "Brain Tumor Segmentation and Radiomics Survival Prediction: Contribution to the BRATS 2017 Challenge," Feb. 2018. [Online]. Available: https://arxiv.org/abs/1802.10508v1

[4] W. Dai, B. Woo, S. Liu, M. Marques, C. B. Engstrom, P. B. Greer, S. Crozier, J. A. Dowling, and S. S. Chandras, "CAN3D: Fast 3D Medical Image Segmentation via Compact Context Aggregation," *arXiv:2109.05443 [cs, eess]*, Sep. 2021, arXiv: 2109.05443. [Online]. Available: http://arxiv.org/abs/2109.05443

[5] O. Cicek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*, ser. Lecture Notes in Computer Science, S. Ourselin, L. Joskowicz, M. R. Sabuncu, G. Unal, and W. Wells, Eds. Cham: Springer International Publishing, 2016, pp. 424–432.

[6] Y. Rao, W. Zhao, Z. Zhu, J. Zhou, and J. Lu, "GFNet: Global Filter Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 10 960–10 973, Sep. 2023, conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. [Online]. Available: https://ieeexplore.ieee.org/document/10091201?denied=

[7] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, "Siamese neural networks for one-shot image recognition," in *ICML deep learning workshop*, vol. 2. Lille, 2015, p. 0.

[8] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu, "Neural Discrete Representation Learning," *arXiv:1711.00937 [cs]*, May 2018, arXiv: 1711.00937. [Online]. Available: http://arxiv.org/abs/1711.00937

[9] A. Razavi, A. v. d. Oord, and O. Vinyals, "Generating Diverse High-Fidelity Images with VQ-VAE-2," *arXiv:1906.00446 [cs, stat]*, Jun. 2019, arXiv: 1906.00446. [Online]. Available: http://arxiv.org/abs/1906.00446

[10] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," *arXiv:1812.04948 [cs, stat]*, Mar. 2019, arXiv: 1812.04948. [Online]. Available: http://arxiv.org/abs/1812.04948

[11] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," *arXiv:1912.04958 [cs, eess, stat]*, Mar. 2020, arXiv: 1912.04958. [Online]. Available: http://arxiv.org/abs/1912.04958

[12] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," Apr. 2022, arXiv:2112.10752 [cs]. [Online]. Available: http://arxiv.org/abs/2112.10752

[13] S. Liu, L. Liu, C. Engstrom, X. V. To, Z. Ge, S. Crozier, F. Nasrallah, and S. S. Chandra, "Style-Based Manifold forÂ Weakly-Supervised Disease Characteristic Discovery," in *Medical Image Computing and Computer Assisted Intervention â€" MICCAI 2023*, ser. Lecture Notes in Computer Science, H. Greenspan, A. Madabhushi, P. Mousavi, S. Salcudean, J. Duncan, T. Syeda-Mahmood, and R. Taylor, Eds. Cham: Springer Nature Switzerland, 2023, pp. 368–378.