NTNU
Norwegian University of
Science and Technology

Deep Learning Applied to Automatic Anomaly Detection in Capsule Video Endoscopy

Johnny Offerdal

Applied Computer Science
Submission date: June 2018
Supervisor: Rune Hjelsvold, IDI
Co-supervisor: Ahmed Mohammed, IDI

Norwegian University of Science and Technology
Department of Computer Science

Preface

This is a Master's thesis in Applied Computer Science at NTNU Gjøvik, carried out during the spring semester in 2018. The idea to the project came up during an discussion with one of the project's supervisors, Mohammed Ahmed Kedir. Readers of this thesis should be familiar with computer science but most concepts discussed are explained in the thesis.

01-06-2018

Acknowledgment

I would like to thank the following persons for their great help during Mohammed Ahmed Kedir and Rune Hjelsvold for supervision and interesting discussion throughout this project. I would also like to Øistein Hovde for his contributions and explainations of diseases in the gastrointestinal tract.

J.O.

Abstract

Introduction: Regularly screening of the gastrointestinal tract for polyps is the an important measure for preventing colorectal cancer. Screening large population's gastrointestinal tract is with todays common methods too time consuming and expensive to accomplish. In this thesis explore the possibilities of using capsule video endoscopy (CVE) in combination with state of the art convolutional neural network (CNN) to create a computer aided diagnosis-system for automatic classification of diseases in the gastrointestinal tract.

Methods: First we create a dataset using images extracted from real CVE examinations of 19 patients. We use tensorflow framework to train and evaluate different state of the art CNNs compare with each other. We also propose a new CNN consisting of two state of the art CNNs in a parallel architecture. We also develop a Graphical User Interface (GUI) aimed at medical doctors for classifying VCE data.

Results: The dataset created contains 3267 labeled images with highlighted lesion annotation. We achieve F_{Macro}-score, precision, recall and accuracy at 0.547, 0.553, 0.548 and 0.682 respectively on the best performing CNN. The GUI prototype is a simple application which classifies images based on predictions done by the trained network.

Discussion: The dataset have some balance issues as the largest class, normal images contains more than thousand samples, and five of the diseases contains less than hundred. The CNNs show promising preliminary results suggesting that it's a feasible approach. The GUI prototype suffers from severe performance issues and is only able classify about two images per second.

Conclusion: In this thesis we have shown that using CNN in combination with CVE is a a feasible approach. The results are promising but more research is required. The big challenge in image classification continues to be having a large and reliable dataset.

Contents

Preface . . . . . **i**
Acknowledgment . . . . . **iii**
Abstract . . . . . **v**
Contents . . . . . **vii**
List of Figures . . . . . **ix**
List of Tables . . . . . **xi**
1 Introduction **1**
1.1 Background & motivation 1
1.2 Problem statement 2
1.3 Main contributions 2
1.4 Outline 2
2 Background **5**
2.1 Traditional screening methods 5
2.1.1 Capsule Video Endoscopy 6
2.1.2 Computer Aided Diagnosis (CAD) 6
2.2 Machine Learning 6
2.3 Convolutional Neural Networks 7
2.3.1 Well known CNN architectures 7
2.4 Related Work 8
3 Creating the dataset **9**
3.1 Methods 9
3.2 Results 10
3.3 Discussion 11
4 Experiments **13**
4.1 Methods 13
4.1.1 Setup 13
4.1.2 Experiments 1-4 13
4.1.3 Experiment 5 - Proposed network 14
4.1.4 Experiment 6 14
4.1.5 Evaluation 16
4.2 Results 16
4.2.1 Experiment 1 16
4.2.2 Experiment 2 16

4.2.3 Experiment 3 . . . 17
4.2.4 Experiment 4 . . . 18
4.2.5 Experiment 5 - Proposed network . . . 18
4.2.6 Experiment 6 . . . 20
4.3 Discussion . . . 20
5 GUI Prototype . . . 23
5.1 Methods . . . 23
5.1.1 Tools & libraries . . . 23
5.2 Implementation . . . 25
5.3 Results . . . 25
5.4 Discussion . . . 25
6 Conclusion . . . 27
6.1 Future Work . . . 27
Bibliography . . . 29
A Evaluation scripts . . . 31
B Training . . . 43
B.1 train_image_classifier.py . . . 43
B.2 Training script for proposed architecture . . . 55
C Proposed network . . . 59
D Main code for CveClassifier . . . 61
E Helper scripts . . . 65

List of Figures

1 The GI tract with labels 1
2 A Colonoscope 5
3 A Capsule Endoscope 6
4 Example of an .annotation file. 10
5 imgAnnotation tool in use. 11
6 Data distribution in the dataset. 12
7 Confusion matrix for experiment 1 17
8 Confusion matrix for experiment 2 17
9 Confusion matrix for experiment 3 18
10 Correctly predicted images. 18
11 Incorrect predictions done by the network. 19
12 Confusion matrix for experiment 4 19
13 Confusion matrix for experiment 5 19
14 Confusion matrix for experiment 6 20
15 Use case diagram for the prototype 24
16 Sequence diagram for the prototype. 24
17 Screen dump of the application in use. 26

List of Tables

1 Showing training parameters for experiment 1-4 15
2 Summary of the results from the experiments 21

1 Introduction

1.1 Background & motivation

There are several types of diseases that can affect the human gastrointestinal (GI) tract(see figure 1). This includes three types of cancer, colorectal cancer, stomach cancer and esophagus cancer. Other diseases located in the GI tract include inflammatory bowel dieases (IBD) like chron's disease and ulcerative colitis[1] among others.

Colorectal cancer is the fourth most common cancer diagnosed in the USA, and is expected to cause over 50.000 deaths in USA alone, in 2018 according American Cancer Society[2]. These cancers often develop in lesions called polyps, growths inside the large bowel (colon) and the small bowl. Early detection of polyps in the gastrointestinal (GI) tract is crucial in preventing the disease to develop and spread, as well as provide more options regarding treatments[3].

Todays screening methods like endoscopy and colonoscopy are not compatible with population wide screening regimes because of the time consumption and the increased cost for health care. Therefore new technology has to be applied for time efficiency and cost reduction.

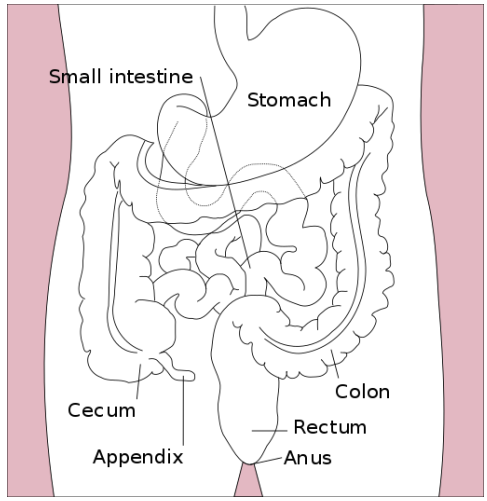


Figure 1: The GI tract with labels[1]

[1]Image gathered from: https://commons.wikimedia.org/wiki/File:Stomach_colon_rectum_

1.2 Problem statement

Capsule video endoscopy (CVE) is still rarely used, and one of the reason for that is the expenses in terms of the system itself and the time consumption related to analyzing the data. A full video from the capsule may be as long as 8 hours, which a medical doctor will have to examine. This is not just a tiresome task for the medical doctor but also many hours the doctor could have spent consulting other patients. Having a system that can greatly reduce the time spent on analyzing CVE examination might be a step in the right direction for population wide gastrointestinal screening.

The main goals of this thesis is therefore:

- Create a dataset containing
- Explore the possibility for a new architecture for convolutional neural networks.
- Create Graphical User Interface (GUI) for disease classification.
- Compare state of the art convolutional neural networks against each other for sake of lesion classification.

1.3 Main contributions

In this thesis, we shown that convolutional neural networks can be used in the case of automatic lesion classification in capsule endoscopy videos. The preliminary results are promising and suggests that there is large potential in this technology for automatic lesion classification. The system shows low error-rate for normal vs lesion which is important factor within the medical area.

The main contribution of this thesis is the comparisons of different state of the art convolutional neural networks on the task of lesion classification, and the graphical user interface aimed at medical doctors. Hopefully this will trigger more interest in the area so that we in the future patients will have better and quicker screening options available as well as reduced health care costs.

1.4 Outline

The rest this thesis is structures as follows:

- **Chapter 2 - Background:** The background covers today's screening methods, brief history of machine learning techniques, introduction to convolutional neural networks and related works.
- **Chapter 3 - Creating the dataset:** In this chapter we present the methods used for creating the dataset, we present the results and show some examples from the dataset. We then discuss the process and the findings.

`diagram-en.svg`

- **Chapter 4 - Experiments:** In this chapter we present the methods we used in the experiments, we present the results from training and evaluation. We then discuss the findings.
- **Chapter 5 - GUI Prototype:** In this chapter we present the methods we used to develop the prototype, we present a use case and provide a sequence diagram explaining the how the application work. Further we show the design and results of the development and at last we discuss the development process and results.
- **Chapter 6 - Conclusion:** In this chapter we summarize our findings in the thesis and present potential future works.

2 Background

2.1 Traditional screening methods

The most common method used to examine the digestive tract is the use of a flexible wired endoscope, where a small camera is inserted either trough the mouth, called endoscopy or through the rectum, called colonoscopy (shown in figure 2). These are very common procedures, and according to Gastrointestinal Endoscopy Associates (GIEA), the procedure only lasts about 30 minutes [4]. However these procedures are limited, where the endoscopy are able to examine the esophagus, stomach, duodenum and a colonoscopy only examines the colon and terminal illeum, the small intestines (illeum) are not examined. These procedures may also be uncomfortable for the patients and some will require drugs, to be sedated or to numb the back of the throat. Particularly patients having problems with gagging will feel pain and uncomfortable during an endoscopy.

Figure 2: An example of a colonoscope[1]

A less invaisive method for examining the colon is CT colonography also called virtual colonoscopy. This method uses x-ray images to examine for polyps in the colon. The problem with CT colonography is that it's only effective in detecting larger polyps with size greater than 1 cm. This procedure also exposes the patient for radiation, which induces the patient for a slight risk of cancer.

[1]Image gathered from: https://commons.wikimedia.org/wiki/File:Colonoscope.jpg

2.1.1 Capsule Video Endoscopy

Capsule video endoscopy (CVE) or wireless capsule endoscopy (WCE) is a tiny camera shaped as a vitamin pill (shown in figure 3). It contains one or two cameras, LED flash, a battery and a wireless transmitter that transfers video to a receiver, which often is placed on the patient's body. The patients swallow this capsule, and it will record video at a rate of 2-6 frames per second depending on the velocity of capsule, through the whole digestive tract. This of course also includes the small intestines which is missed by the traditional screening method.

Figure 3: An example of a capsule endoscope[2]

2.1.2 Computer Aided Diagnosis (CAD)

Computer aided diagnosis (CAD) are software that can assist medical professionals in diagnosing the patient by interpreting medical images. There are multiple examples of CAD systems in use at hospitals, including mammography (breast cancer screening)[5], colon cancer[6], and for coronary artery disease[7] among others. The goal of CADs is to assist the medical doctor to diagnose the patient more quickly than manually examining the medical images, to reduce cost and securing the correct diagnosis.

2.2 Machine Learning

Machine learning was defined back in 1959 by Arthur Samuel [8], where he created a program which learned checkers by playing against itself. Over time the game learned patterns which would lead to wins and patterns which would lead to losses and after only hours of training it would actually play checkers better than Arthur Samuel himself. A more formal definition was made by Tom Mitchell in

[2]Image gathered from: https://commons.wikimedia.org/wiki/File:CapsuleEndoscope.jpg

1998 [**?**]: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

There are two main categories of machine learning, supervised learning and unsupervised learning. In supervised learning, we have a input x and an output Y and use an algorithm to map the function from input to output. This method of learning requires a large set of labeled data the algorithm can use to learn.

The goal is to generalize such that when the function receives new data it will output the correct prediction. Supervised learning can further be broken down to classification problems and regression problems.

In unsupervised learning, there is just a input X with no corresponding correct output, the network itself then have to learn the underlying structure in the data.

Deep learning is a machine learning approach that in recent times grown great popularity in the artificial intelligence community even though deep learning was introduced to the machine learning community by Rina Dechter back in 1986 [9]. The introduction of powerful hardware and especially the usage of multi-core Graphical Processor Units (GPUs) really kicked off the community. Deep learning include architectures like deep neural networks, deep belief networks among others, have in recent years applied to a wide range tasks like computer vision and natural language processing to mention a few. The most common neural network for computer vision and image classification tasks are convolutional neural networks, which will be our focus in this thesis.

2.3 Convolutional Neural Networks

Convolutional neural networks (CNN) or deep CNNs are artificial neural networks primarily used in image recognition and classification tasks. The architecture is inspired by how the neurons in the human brain works. Each neuron in the CNN will eiter activate or deactivate when looking at some object. Each layer will focus on different features, e.g will the first layers focus on lines and edges, while we progress towards the last layers the focus will change to e.g. colours. Putting all the neurons together and the network is able to recognize even small details in an image.

2.3.1 Well known CNN architectures

Inception and Inception ResNets [10] is one of the most famous CNNs available, and is developed by the Google brain team working on artificial intelligence. The first version of inception was called GoogLeNet, later years they have released updated models every now and then. The latest model released Inception v4 and Inception ResNet v2. These are currently among the top performers in the ImageNet

challenge.

AlexNet [11] was presented by Alex Krizhevsky et al. in 2012 with results on spectactular results on the ImageNet[3] dataset.

2.4 Related Work

There is a much research going on in this field but most of the studies seem to be investigating polyp detection only. Yuan et al. [12] did a study on capsule video endoscopy published 2017 on using deep learning for polyp, bubble and turbid detection. They used a deep learning technique called stacked sparse autoencoder with image manifold contraint (SSAEIM). Their results are state of the art with an overall recognition at 98%. The dataset they use is unfortunatly not available to the public because of privacy issues.

In 2017 Li et al. [13] conducted a study using capsule video endoscopy classifying between normal and haemorrhage. Their dataset consisted of 40.000 normal images and about 1.300 haemorrhage images. They implemented rotation and luminance change, blurring and poisson noise in their image augmentation technique. They achieved an F-score of 98.87%.

[3]ImageNet is a dataset containing 1000 classes, and is used in computer vision challenges

3 Creating the dataset

In deep learning is having a good dataset essential to get good results when training a network. This means having large amount of images so that the network is able to learn the repeating patterns. In this chapter we present the method, the results and discuss the process and the outcome of the creation of the dataset used in this thesis.

3.1 Methods

Sykehuset Innlandet provided 36 anonymised PillCam reports in Rapid Reader [14] format, with several lesions highlighted. These reports were reviewed, and interesting parts were extracted into video sequences a few seconds before and after the highlighted lesion. These video sequences were then split into single PNG image files, for each lesion per patient.

The images were further annotated with a open source tool called imgAnnotation available at github[1]. In discussions with my supervisor Ahmed Kedir and his supervisor at Sykehuset Innlandet Øistein Hovde we came up with what information that would benefit this thesis, and further work in the field. When annotating images we were looking for the following lesions:

- Polyps
- Pseudopolyps
- Granularity
- Erosions
- Ulceration
- Diverticulosis
- Erythema
- Haemmorhoids
- Oedema
- Haemorrhage

All lesions observed where then labeled with bounding boxes around the region of interest. Images with no observed lesions was labeled as 'normal'. Additionally we stored the following relevant information about the images:

- Bile: Binary 1/0

[1] imgAnnotation tool available at https://github.com/alexklaeser/imgAnnotation

- Bubble: Binary 1/0
- Cleanse level: Graded 1-3 (1 is best cleanse)
- Informative: Binary 1/0
- Debris: Binary 1/0

This information was saved as a .annotation (see example in figure 4) file created by the imgAnnotation application. We used this information to further filter our data. Images with the worst cleanse (3) and didn't contain any lesions, are not included in the final dataset as it is not useful to see only debris or bubbles for the medical doctor.

```
########## NEW FILE ##########
file: /home/johnny/imgAnnotation-master/imgAnnotation-master/....
debris: 1

object: 2
bbox: 70,393,321,142
erosions: 1
erythema: 1

object: 3
bbox: 97,32,302,87
erosions: 1
erythema: 1

########## NEW FILE ##########
file: /home/johnny/imgAnnotation-master/imgAnnotation-master/....
debris: 1
good_cleanse: 0
informative: 0
```

Figure 4: Example of an .annotation file.

A python script was created to automate the process of extracting the relevant images based on these criteria, the script can be reviewed in appendix E.The images were then converted into TFRecords, with 90% / 10% split in training and validation respectively, using the scripts in appendix E.

3.2 Results

We labeled images from 19 of 36 available patients with different diseases, gathering in total 3267 images. The distribution of images in the dataset can be seen in figure 6. The resulting dataset is quite skewed, where the normal class contains

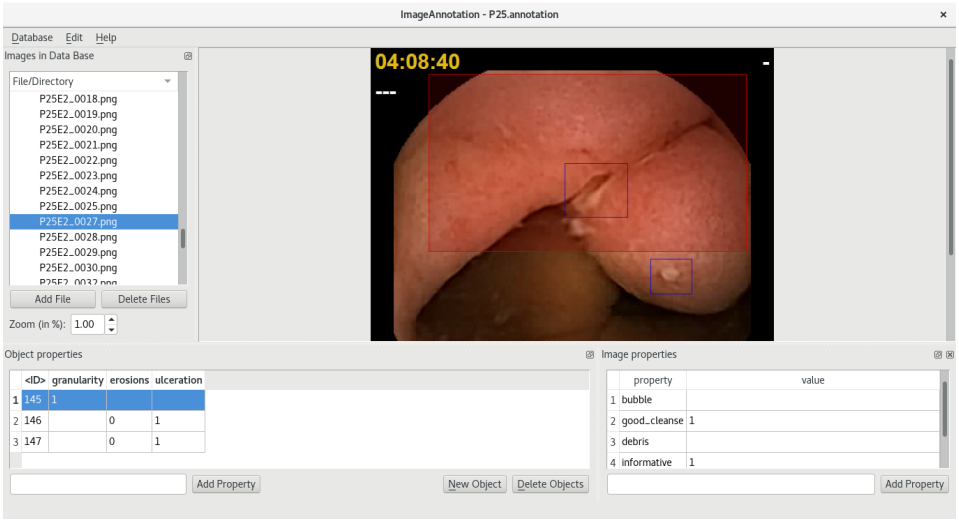


Figure 5: imgAnnotation tool in use

more than thousand samples and four of the lesion classes contains less than hundred samples.

In figure 6 it's hard to see but the haemmorhoids class contains only 11 samples.

3.3 Discussion

The task of creating the dataset was very time consuming, especially since we dont have any expertise nor in-depth knowledge of what to look for in these images. We were very dependant on using the rapid reader reports created by specialists from Sykehuset Innlandet and follow the marked lesions forward and backwards in time when annotating images.

The outcome of the dataset is not great, there are large imbalances between classes and the total size is small, but because of time constraint concerning the thesis we had to say stop at some point, to finish in time.

It's important to note that only a small fraction of the dataset had been validated by a specialist. This was done with a specialist from Sykehuset Innlandet, in collaboration with us. The dataset may contain human errors and should be treated as that. Hopefully this dataset can be extended and validated and in the future be released to the public so that more researcher can work on solving the problems at hand.

The dataset is not released to the public because of privacy issues but might be released in the future.

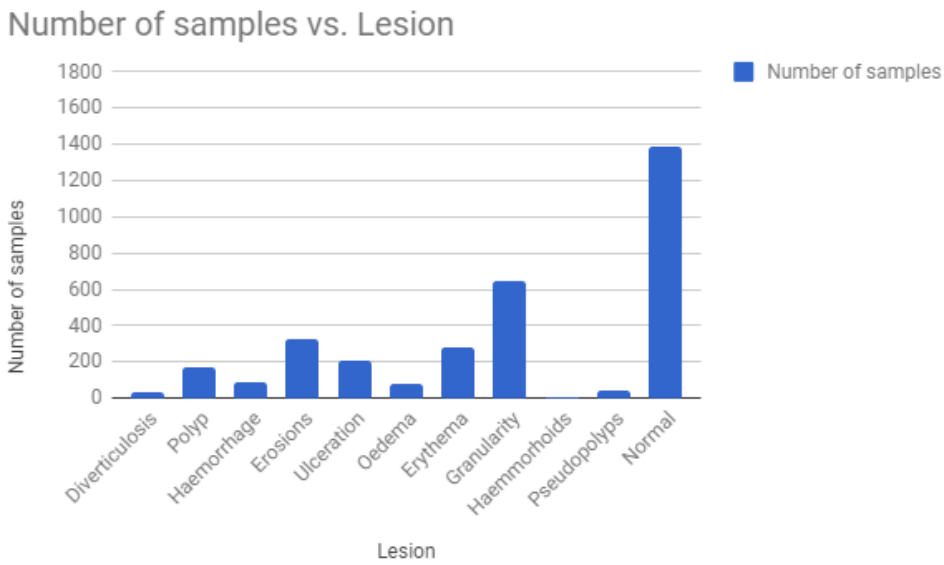


Figure 6: Data distribution in the dataset.

4 Experiments

In this chapter we present the methods used in the experiments and the results from each experiment and in the end we discuss the process and results. All files used are publicly except dataset available at https://github.com/johnnyof/cve-classifica

4.1 Methods

In this section we present the hardware and software configurations used to conduct the experiments as well as individual differences between the experiments. The experiments 1-4 was conducted in a very similar manner but the proposed architecture have some design changes relative to the others.

4.1.1 Setup

All experiments were conducted on a computer provided by NTNU Gjøvik. The hardware configuration was as follows:

- Processor: Intel Xeon CPU E5-1620 v4 @3.5GHz x 8
- RAM: 32GB
- GPU: nVidia TITAN x (Pascal)

The system was running ubuntu 16.04 LTS, Python version 3.5.2 and Tensorflow-GPU version 1.6 (released February 28th) installed natively using pip3. In addition are jupyter (version 1.0.0 tested), numpy (version 1.14.2 tested), scikit-learn (version 0.19.1 tested) and protobuf (version 3.5.2 tested) required for running training and evaluation code. The TensorFlow slim framework is the basis of all the work done in this part, some code is used out-of-the-box, while some code is modified to fit the use for this thesis.

All scripts used in the following experiments must be placed in the root folder of tensorflow slim unless otherwise is specified.

4.1.2 Experiments 1-4

Training

These experiments follow the same procedure for training using out-of-the-box script for training the networks (see appendix B train_image_classifier.py). We train the networks using the dataset discussed in chapter 3. A complete list of parameters for each of the experiments are shown in table 1. All experiments use the same data augmentation technique provided by tf slim's inception preprocessing script. This include random rotations, color distortions, scaling and cropping.

Experiment 1 was fine-tuning the *inception_resnet_v2* [10]. We trained the network for 100k steps, with batch size of 32 using the adam optimizer. In this experiment only the *Logits* and *AuxLogits* layers were trained. The weights were initialized with weigths from a ImageNet checkpoint.

In **experiment 2** we trained *alexnet_v2* [11] from scratch using batch size of 32 for 100k steps with the adam optimizer. The initial weights were created using the xavier_initializer function in tf.

For **experiment 3** we trained the *inception_resnet_v2* from scratch using the same parameters as in experiment 1 except for increased training steps with 50k to 150k and the initial weights now was done with xavier_initializer.

Experiment 4 replicated experiment 3 in every way except that we instead trained *inception_v4* [10].

4.1.3 Experiment 5 - Proposed network

In this experiment we create a new architecture, a combination of two state of the art CNNs in a parallel structure. We used Inception v3 [15] in combination with Inception ResNet V2, section C shows an overview of how the architecture is constructed. In the code below the construction of the network is shown. We first feed the images to cnn1(Inception ResNet v2) and cnn2(Inception v3) then we extract the features from the layers *Mixed_7a* and *Mixed_7c* from cnn1 and cnn2 respectively. We then concatenate the output tensors at axis 3. Further we flatten the output before we add a dropout layer and fully connected layer. We trained the network from scratch for 100k steps, with batch size 16 with the adam optimizer with an initial learning rate of 0.001. The training was done using the jupyter notebook shown in appendix B.2

```
def jonet(images):
    net1, end_points1 = cnn1(images)
    net1 = end_points1['Mixed_7a']
    net2, end_points2 = cnn2(images)
    net2 = end_points2['Mixed_7c']
    net = tf.concat((net1, net2), 3)
    net = slim.flatten(net)
    net = slim.dropout(net, 0.8, is_training=True)
    net = slim.fully_connected(net, 11, activation_fn=None)
    return net
```

4.1.4 Experiment 6

To have more data to evaluate our proposed method we did a experiment using most of the same code but we removed one of the networks but keep the last layers, corresponding to flatten, dropout and the fully connected layer as demonstrated in the code below. Other than that the experiment was conducted in the same manner

Parameters / Experiment	Experiment 1	Experiment 2	Experiment 3	Experiment 4
model_name	inception_resnet_v2	alexnet_v2	inception_resnet_v2	inception_v4
trained from scratch	no	yes	yes	yes
batch_size	32	32	32	32
preprocessing_name	inception	inception	inception	inception
max_number_of_steps	100.000	100.000	150.000	150.000
Optimizer	adam	adam	adam	adam
Initial learning rate	0.01	0.01	0.01	0.01
End learning rate	0.0001	0.0001	0.0001	0.0001
train_image_size	299	224	299	299

Table 1: Showing training parameters for experiment 1-4

as experiment 5.

```
def jonet(images):
    net, end_points1 = cnn1(images)
    net = end_points1['Mixed_7a']
    #net2, end_points2 = cnn2(images)
    #net2 = end_points2['Mixed_7c']
    #net = tf.concat((net1, net2), 3)
    net = slim.flatten(net)
    net = slim.dropout(net, 0.8, is_training=True)
    net = slim.fully_connected(net, 11, activation_fn=None)
    return net
```

4.1.5 Evaluation

The evaluation was done using the evaluation part of the dataset, it's important to note that those images was never included in the training of the networks. The evaluation dataset contains as discussed 400 images. To evaluate our models we use the metrics precision, recall, $F1_{Macro}$ and accuracy [1]. In addition we provide a confusion matrix. The metrics computed by the scikit-learn library *sklearn.metrics.precision_recall_fscore_support*, except for accuracy which is calculated by the tensorflow module *tf.metrics.accuracy*. The evaluation was done using the evaluation scripts provided in appendix A, because the general evaluation script provided in tf slim did not provide sufficient evaluation metrics.

4.2 Results

In this section we present the results from each of the experiments, described in the methods section and present the confusion matrix for them.

4.2.1 Experiment 1

The first experiment on Inception ResNet v2 using transfer learning and fine-tuning the last layers achieved $F1_{Macro}$, precision, recall and accuracy scores at 0.19, 0.22, 0.20 and 0.59 respectively. The results was suprisingly considering transfer learning are often used in cases when the dataset is not containg large amount of data. Figure 7 shows the confusion matrix for evaluation of the experiment.

4.2.2 Experiment 2

In the second experiment we trained Alexnet v2 from scratch, performed worst than the first experiment of fine tuning. With $F1_{Macro}$-score of 0.157, it's the worst performing network in our case and on our dataset. The confusion matrix is shown in figure 8. The other evaluation metrics was precision at 0.14, recall at 0.19 and

[1] Metrics derived from sklearn class http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

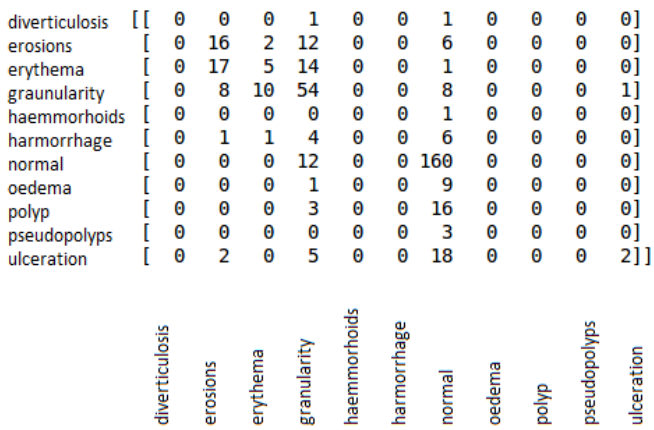


Figure 7: Confusion matrix for experiment 1

accuracy at 0.57

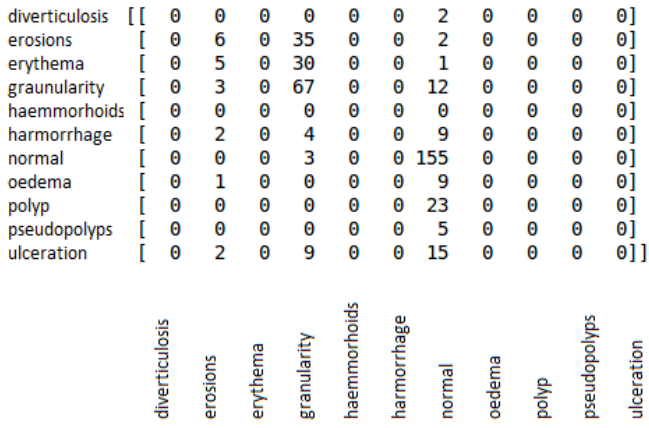


Figure 8: Confusion matrix for experiment 2

4.2.3 Experiment 3

The third experiment provided the best results on when considering the most reliable metric, the $F1_{Macro}$-score. The $F1_{Macro}$-score was at 0.548, the precision at 0.553, the recall at 0.548 and accuracy at 0.6825. The confusion matrix is provided in 9.

We also extract some of the predictions done by the network on the validation set, in figure 10 we show correctly predicted images and in figure 11 we show

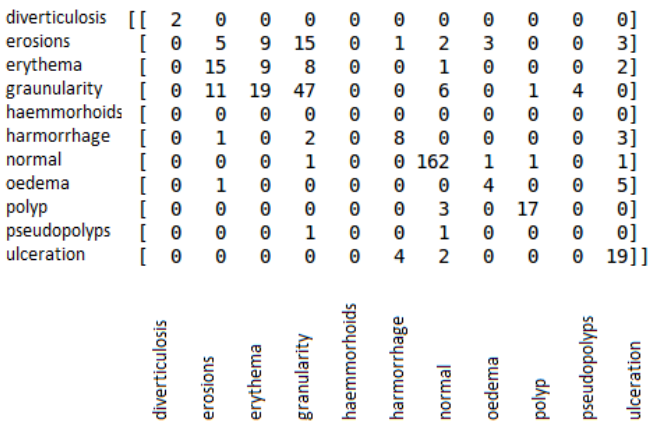


Figure 9: Confusion matrix for experiment 3

incorrect predictions.

Figure 10: Correctly predicted images

4.2.4 Experiment 4

The Inception v4 model provided the best accuracy of the tested networks at 0.697. In the other metrics it performed significantly worse than Inception ResNet v2 in experiment 3. The $F1_{Macro}$, precision and recall was 0.43, 0.49, and 0.40 respectively. Confusion matrix for this experiment is shown in figure 14.

4.2.5 Experiment 5 - Proposed network

Our proposed network using a dual parallel architecture did not provide better results than Inception ResNet v2 alone, in experiment 3. How ever the $F1_{Macro}$, precision and recalll is quite close at 0.528, 0.531 and 0.5399 respectively. The accuracy in this experiment came in at 0.617. The confusion matrix is shown in figure 13

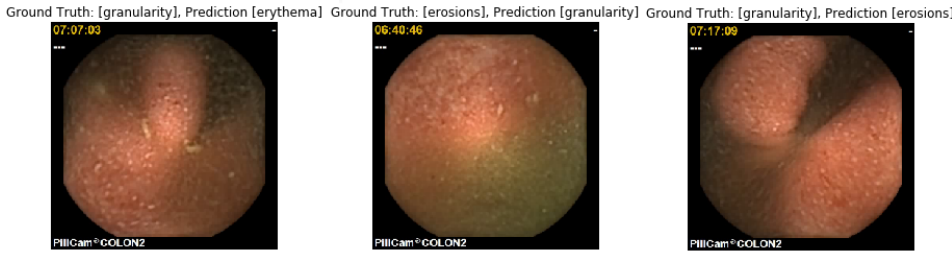


Figure 11: Incorrect predictions done by the network.

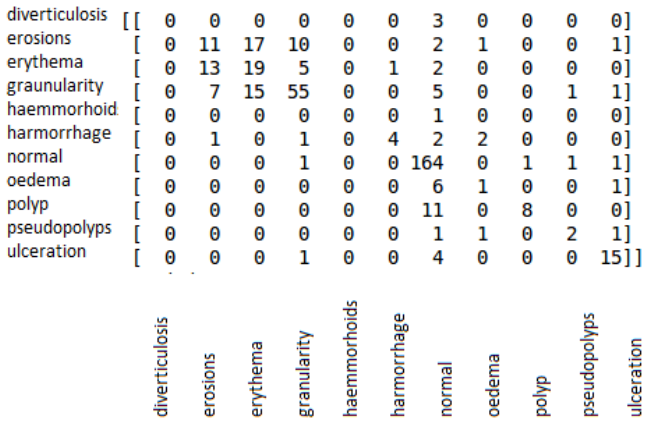


Figure 12: Confusion matrix for experiment 4

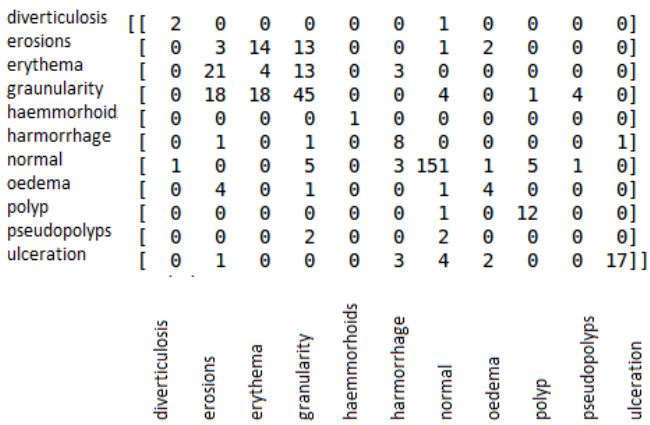


Figure 13: Confusion matrix for experiment 5

4.2.6 Experiment 6

This experiment was conducted to verify potential flaws or mistakes in the proposed network. This network performance slightly better than the proposed netowork but slightly worse than experiment 3. $F1_{Macro}$, precision, recall and accuracy is calculated at 0.53, 0.51, 0.56. and 0.635 respectively.

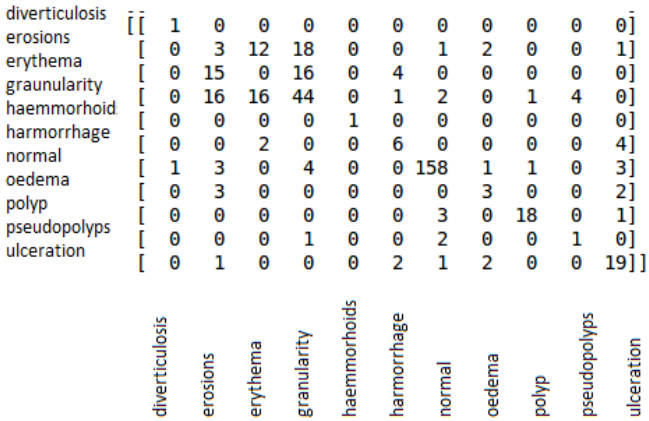


Figure 14: Confusion matrix for experiment 6

Summary

All the results are summarised in table 2.

4.3 Discussion

Some of the results are quite surprising, experiments 1 and 2 performed a lot worse than expected. On one hand transfer learning and fine tuning is said to be a efficient way of training networks when the dataset is small [] and the other is that when the dataset is small using deeper networks does not improve performance significantly []. Our results suggests that deeper networks do improve performance significantly and that training ResNets from scratch do perform better than other models even if the dataset is small. This is also backed up by Kornblith et al. [16].

Investigating the confusion matrix from the experiments we see the there are in most cases very few predictions that images with lesions is predicted to be normal. In our best performing model, the there are 15 wrong predictions, where a lesion is predicted as normal. This calculates to an error rate equal to $1 - (15/400) = 0.0375$. This is the most important variable in our case, and it's very important in the field of medical imaging that the risk of classifying a disease condition as normal condition is minimal. Another important note on the confusion matrix' is

Metric / Experiment	Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5	Experiment 6
$F1_{Macro}$	0.1900	0.1568	0.5479	0.4324	0.5278	0.5300
Precision	0.2287	0.1433	0.5531	0.4931	0.5312	0.5108
Recall	0.2045	0.1937	0.5480	0.4058	0.5399	0.5665
Accuracy	0.5925	0.57	0.6825	0.6975	0.6175	0.635

Table 2: Summary of the results from the experiments

that generally most of the prediction errors occur on the same lesions, erosions, erythema and granularity. One of the reasons for this might be that on the images in the dataset there is possibly multiple diseases in one image, but that specific image is only mapped to one specific lesion. Another case is that some of the lesions look very similar, even trained specialists have problems with certain lesions, and different specialists may label diseases differently. Studying figure 11, there is hard to differ one from each the other and it's not easy to make a prediction ourselves.

It's also important to note that in a real scenario, when analyzing roughly 60.000 it is very likly that the same lesion will appear on multiple images, further increasing the chance for detecting it. This will positively affect the system.

5 GUI Prototype

In this chapter we present the methods, results and discussion from the development of the GUI prototype. The prototype is publicly available at https://github.com/johnnyof/cve-gui. The main code can be seen in appendix D.

5.1 Methods

5.1.1 Tools & libraries

The Graphical User Interface (GUI) prototype was written in Python 3[1] and designed with Qt Designer 4.8, a free "what you see is what you get (WYSIWYG)" tool for PyQt 4 [2]. The following libraries are required for the application to function properly:

- system
- os
- time
- tensorflow 1.4
- shutil
- numpy
- PyQt4

We include in the simplest for a use case diagram (seen in figure 15) where a medical doctor interacts with the application. The medical doctor has in total five button to interact with. Three of them open the file explorer dialog, this include the functions for finding the relevant images (e.g. a folder with relevant images), selecting a folder to store the classfication results (e.g. destination where images are copied to while classifying), selecting the model/graph to use for prediction/inference.

The process of the how the prototype works is explained in the sequence diagram in figure 16. The actor, the medical doctor selects the input images, output destination and the model to use for predictions, and clicks the classify button. The application then loads the selected graph into memory, and start looping through the images and for each image gives a prediction of what it sees. The top result are chosen, and the file is copied to the corresponding folder in the output destination folder.

[1] Python 3 available at https://www.python.org/download/releases/3.0/
[2] Qt available at https://www.qt.io/

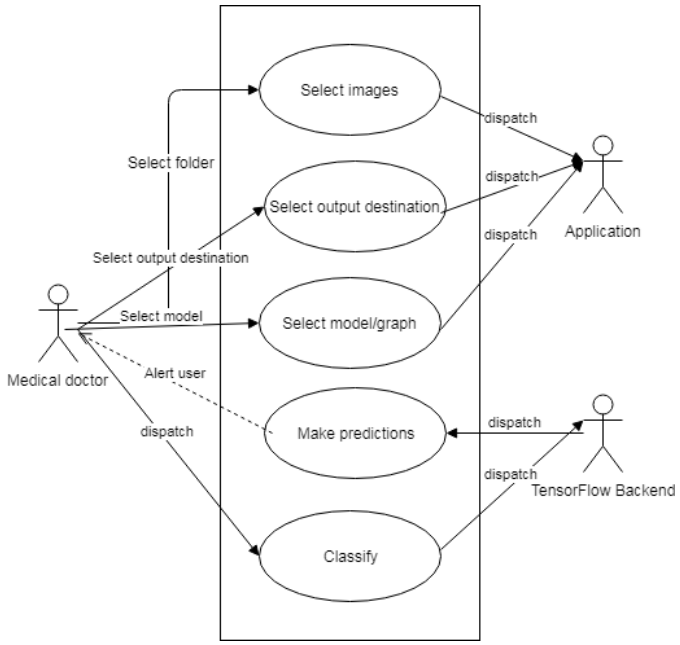


Figure 15: Use case diagram for the prototype

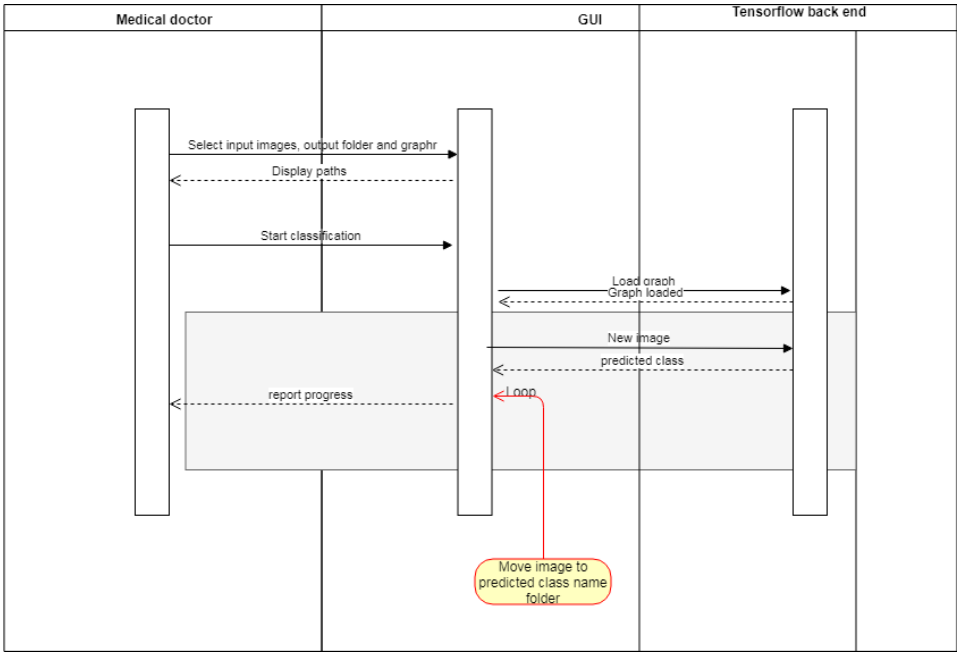


Figure 16: Sequence diagram for the prototype.

5.2 Implementation

The model we selected for implementation was the Inception ResNet v2 because thats the one which performed best in our previous experiments. To be able to use the model in external application we had to freeze the graph weights in protobuf file (.pb). To do this we used the *freeze_graph.py* script provided in tf slim. The model can then be imported to external applications by loading the graph in a tf.session inside the application.

The model file is not hardcoded into the application so that it's easy to update change the model with a newer and better model subsequently. How ever the label file, containing the name of the lesions is hardcoded. This means that if new lesions or the order or the lesions change in the model, this has to be changed as well. Either by replacing the labels.txt file in the graph folder, or changing the path to the new labels file in the source code. This is only one line of code with the a file path.

5.3 Results

The outcome of the prototype development, is a very simple GUI with that should be very easy to use, with just the intructions provided when launching the application. In figure 17 the application is shown in progress of classifying images.

In the right top corner there is a text windows that constantly reports the progress back to the user. In the bottom there is a progressbar. The top left corner the functions explained in the implementation part, for selecting folders and the model.

The application is only able to process about two images per second in the current state, on the software/hardware configuration explained in detail in the experiments section.

5.4 Discussion

The prototype is a very simple proof of concept how it *could* be working. The performance of the application is quite bad with only about two frames per second. The reason for this is not clear, it might be the implementation of the tensorflow code. One idea to fix this is to use batches here as well. Currently it's just looping through the source folder picking one image at a time. Another reason can be that it's reading every image from disk one by one, which is slow, but it doesn't explain the terrible speed of course. One idea to fix this problem is to automate the process of converting images to tf records, which is the preferable file format by tensorflow.

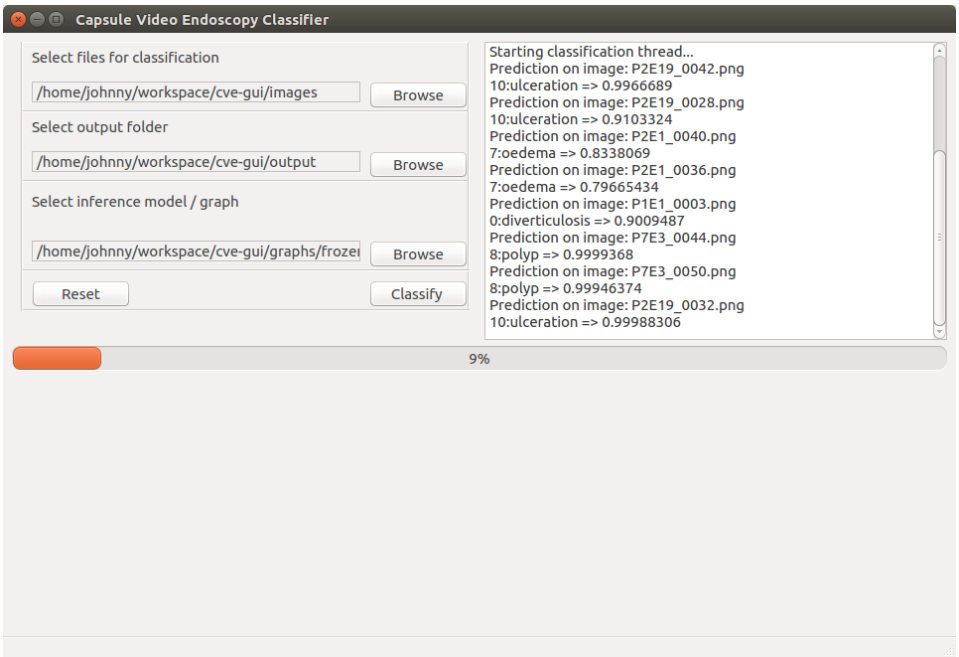


Figure 17: Screen dump of the application in use.

6 Conclusion

In this thesis demonstrated one possible approach for creating datasets for image classification tasks, we have then used this dataset to train and evaluate multiple CNNs. The dataset also contains information about disease location and could essentially also be used for object detection tasks in the future. The annotation of medical images proved to be hard for an untrained non-expert and this task should ideally be done by experts. This will ensure that the labeled data is correct and be more credible.

The results from the multiple experiments conducted in this thesis suggests that Inception ResNet v2 trained from scratch performs significantly better than transfer learning and AlexNet v2. Inception ResNet v2 also performs slightly better than Inception v4 and our proposed method of dual parallel CNNs, on **our dataset**. The overall results are promising and there is reason to believe that the use of CNN is a very viable path towards lesion detection and classification in GI tract. There is reason to believe that a larger and more reliable dataset is required to further increase performance of the networks.

A GUI prototype aimed for medical doctor was also developed by implementing the trained model for image classification. The prototype is merely a proof of concept and is suffering from severe performance issues, only able to compute about two predictions per second.

The biggest challenge continues to be a large database of labeled capsule video endoscopy images and/or videos to make further progress in this field.

6.1 Future Work

For future work there would be interesting to further optimize the network architecture proposed in this thesis. There would be interesting trying with other networks than those proposed, extract the features in a different layer, optimize the layers coming after concatenation or having one network use weights from e.g. imagenet.

For the GUI there is likely to be multiple ways of improving the performance with regards to prediction speed. One way is to use batches and not single images, another could be to automate the conversion to tf records which is the prefered file format for tensorflow.

The most important work for the future is still labeling more images, ideally done by experts in the field.

Bibliography

[1] Fakhoury, M., Negrulj, R., Mooranian, A., & Al-Salami, H. Jun 2014. Inflammatory bowel disease: clinical aspects and treatments. *J Inflamm Res*, 7, 113–120. jir-7-113[PII]. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4106026/, doi:10.2147/JIR.S65979.

[2] Society, A. C. Key statistics for colorectal cancer. Accessed 5. May 2018. URL: https://www.cancer.org/cancer/colon-rectal-cancer/about/key-statistics.html.

[3] Society, A. C. Key statistics for colorectal cancer. Accessed 5. May 2018. URL: https://www.cancer.org/cancer/colon-rectal-cancer/detection-diagnosis-staging.html.

[4] Associates, G. E. Colonoscopy. 21. May 2018. URL: https://www.giendo.net/our-procedures/colonoscopy.html.

[5] Jalalian, A., Mashohor, S. B. T., Mahmud, H. R., Saripan, M. I. B., Ramli, A. R. B., & Karasfi, B. May 2013. Computer-aided detection/diagnosis of breast cancer in mammography and ultrasound: a review. *Clinical Imaging*, 37(3), 420–426. URL: http://dx.doi.org/10.1016/j.clinimag.2012.09.024, doi:10.1016/j.clinimag.2012.09.024.

[6] Perumpillichira, J. J., Yoshida, H., & Sahani, D. V. Aug 2005. Computer-aided detection for virtual colonoscopy. *Cancer Imaging*, 5(1), 11–16. 16154812[pmid]. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1665218/, doi:10.1102/1470-7330.2005.0016.

[7] Faust, O., Acharya, U. R., Sudarshan, V. K., Tan, R. S., Yeong, C. H., Molinari, F., & Ng, K. H. Jan 2017. Computer aided diagnosis of coronary artery disease, myocardial infarction and carotid atherosclerosis using ultrasound images: A review. *Physica Medica: European Journal of Medical Physics*, 33, 1–15. URL: http://dx.doi.org/10.1016/j.ejmp.2016.12.005, doi:10.1016/j.ejmp.2016.12.005.

[8] Samuel, A. L. *Some Studies in Machine Learning Using the Game of Checkers. I*, 335–365. Springer New York, New York, NY, 1988. URL: https://doi.org/10.1007/978-1-4613-8716-9_14, doi:10.1007/978-1-4613-8716-9_14.

[9] Dechter, R. 01 1986. Learning while searching in constraint-satisfaction-problems. 178–185.

[10] Szegedy, C., Ioffe, S., & Vanhoucke, V. 2016. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261. URL: http://arxiv.org/abs/1602.07261, arXiv:1602.07261.

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, Pereira, F., Burges, C. J. C., Bottou, L., & Weinberger, K. Q., eds, 1097–1105. Curran Associates, Inc. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[12] Yixuan, Y. & Q.-H., M. M. Deep learning for polyp recognition in wireless capsule endoscopy images. *Medical Physics*, 44(4), 1379–1389. URL: https://aapm.onlinelibrary.wiley.com/doi/abs/10.1002/mp.12147, arXiv:https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1002/mp.12147, doi:10.1002/mp.12147.

[13] Li, P., Li, Z., Gao, F., Wan, L., & Yu, J. July 2017. Convolutional neural networks for intestinal hemorrhage detection in wireless capsule endoscopy images. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, 1518–1523. doi:10.1109/ICME.2017.8019415.

[14] Medtronic. Rapid reader v8.3 software update | medtronic. Accessed 16. January 2018. URL: http://www.medtronic.com/covidien/en-us/support/gastrointestinal-product-software-upgrades/rapid-reader-v8-3-software.html.

[15] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. 2015. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567. URL: http://arxiv.org/abs/1512.00567, arXiv:1512.00567.

[16] Kornblith, S., Shlens, J., & Le, Q. V. May 2018. Do Better ImageNet Models Transfer Better? *ArXiv e-prints*. arXiv:1805.08974.

A Evaluation scripts

eval_jonet

May 31, 2018

In [ ]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import math
import numpy as np
import tensorflow as tf
import time
import re
from sklearn.metrics import precision_recall_fscore_support as score
from datasets import dataset_utils
from datasets import cve_diseases
from nets import inception_resnet_v2
from nets import alexnet
from nets import nets_factory
from preprocessing import inception_preprocessing
# Main slim library
from tensorflow.contrib import slim
```

In [ ]:
```
model1 = "inception_resnet_v2"
model2 = "inception_v3"

dataset_name = "cve_diseases"
dataset_split_name = "validation"
dataset_dir = "tmp/"
batch_size = 100
max_number_of_steps = 100000
train_dir = "./tmp/cve_diseases-models/model1-ds2"

cnn1 = nets_factory.get_network_fn(
    model1,
    num_classes=None,
    weight_decay=0.00004,
    is_training=False)
```

```
        cnn2 = nets_factory.get_network_fn(
            model2,
            num_classes=None,
            weight_decay=0.00004,
            is_training=False)
```

In [ ]:
```
def jonet(images):

    net1, end_points1 = cnn1(images)
    net1 = end_points1['Mixed_7a']

    net2, end_points2 = cnn2(images)
    net2 = end_points2['Mixed_7c']
    net = tf.concat((net1, net2), 3)
    net = slim.flatten(net)
    net = slim.dropout(net, 0.8, is_training=False)
    net = slim.fully_connected(net, 11, activation_fn=None)
    return net
```

In [ ]:
```
def load_batch(dataset, batch_size=8, height=299, width=299, is_training=False):

    data_provider = slim.dataset_data_provider.DatasetDataProvider(dataset)

    image_raw, label = data_provider.get(['image', 'label'])

    # Preprocess image for usage by Inception.
    image = inception_preprocessing.preprocess_image(image_raw, height, width, is_traini

    # Preprocess the image for display purposes.
    image_raw = tf.expand_dims(image_raw, 0)
    image_raw = tf.image.resize_images(image_raw, [height, width])
    image_raw = tf.squeeze(image_raw)

    # Batch it up.
    images, images_raw, labels = tf.train.batch(
          [image, image_raw, label],
          batch_size=batch_size,
          num_threads=1,
          capacity=2 * batch_size)

    return images, images_raw, labels
```

In [ ]:
```
image_size = 299
batch_size = 400
imgs = 10

with tf.Graph().as_default():
```

```
tf.logging.set_verbosity(tf.logging.INFO)

dataset = cve_diseases.get_split('validation', dataset_dir)
images, images_raw, labels = load_batch(dataset, batch_size)

# Create the model, use the default arg scope to configure the batch norm parameters

logits = jonet(images)

probabilities = tf.nn.softmax(logits)
con_mat = tf.confusion_matrix(
    labels=labels,
    predictions=tf.argmax(probabilities, 1))


predictions = tf.argmax(probabilities, 1)
accuracy = tf.metrics.accuracy(labels, predictions)


checkpoint_path = tf.train.latest_checkpoint(train_dir)


init_fn = slim.assign_from_checkpoint_fn(
  checkpoint_path,
  slim.get_variables_to_restore())
print(tf.argmax(labels))
with tf.Session() as sess:
    with slim.queues.QueueRunners(sess):
        sess.run(tf.initialize_local_variables())
        init_fn(sess)
        accuracy, np_predictions, np_probabilities, np_images_raw, np_labels, con_ma


        plt.imshow(con_mat, interpolation='nearest', cmap=plt.cm.viridis);
        plt.title("Confusion Matrix")
        plt.colorbar()
        plt.clim(0,30)
        plt.xticks(rotation=0)
        plt.yticks(rotation=0)
        plt.tight_layout()
        plt.ylabel("True label")
        plt.xlabel("Predicted label")
        plt.show()
        print(con_mat)

        precision, recall, f1, _ = score(np_labels, np_predictions, average='macro')

        print('precision: {}'.format(precision))
```

```
print('recall: {}'.format(recall))
print('fscore: {}'.format(f1))
print('accuracy: {}'.format(accuracy))

for i in range(imgs):
    image = np_images_raw[i, :, :, :]
    true_label = np_labels[i]
    predicted_label = np.argmax(np_probabilities[i, :])
    predicted_name = dataset.labels_to_names[predicted_label]
    true_name = dataset.labels_to_names[true_label]


    plt.figure()
    plt.imshow(image.astype(np.uint8))
    plt.title('Ground Truth: [%s], Prediction [%s]' % (true_name, predicted_
    plt.axis('off')
    plt.show()
```

eval_jonet_single

May 31, 2018

In [ ]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import math
import numpy as np
import tensorflow as tf
import time
import re
from sklearn.metrics import precision_recall_fscore_support as score
from datasets import dataset_utils
from datasets import cve_diseases
from nets import inception_resnet_v2
from nets import alexnet
from nets import nets_factory
from preprocessing import inception_preprocessing
# Main slim library
from tensorflow.contrib import slim
```

In [ ]:
```
model1 = "inception_resnet_v2"
model2 = "inception_v3"

dataset_name = "cve_diseases"
dataset_split_name = "validation"
dataset_dir = "tmp/"
batch_size = 100
max_number_of_steps = 100000
train_dir = "./tmp/cve_diseases-models/model2-ds2"

cnn1 = nets_factory.get_network_fn(
    model1,
    num_classes=None,
    weight_decay=0.00004,
    is_training=False)
```

```
        cnn2 = nets_factory.get_network_fn(
            model2,
            num_classes=None,
            weight_decay=0.00004,
            is_training=False)
```

In [ ]:
```
def jonet(images):

    net, end_points1 = cnn1(images)
    #net1 = end_points1['Mixed_7a']

    #net2, end_points2 = cnn2(images)
    #net2 = end_points2['Mixed_7c']
    #net = tf.concat((net1, net2), 3)
    net = slim.flatten(net)
    net = slim.dropout(net, 0.8, is_training=False)
    net = slim.fully_connected(net, 11, activation_fn=None)
    return net
```

In [ ]:
```
def load_batch(dataset, batch_size=8, height=299, width=299, is_training=False):

    data_provider = slim.dataset_data_provider.DatasetDataProvider(dataset)

    image_raw, label = data_provider.get(['image', 'label'])

    # Preprocess image for usage by Inception.
    image = inception_preprocessing.preprocess_image(image_raw, height, width, is_traini

    # Preprocess the image for display purposes.
    image_raw = tf.expand_dims(image_raw, 0)
    image_raw = tf.image.resize_images(image_raw, [height, width])
    image_raw = tf.squeeze(image_raw)

    # Batch it up.
    images, images_raw, labels = tf.train.batch(
          [image, image_raw, label],
          batch_size=batch_size,
          num_threads=1,
          capacity=2 * batch_size)

    return images, images_raw, labels
```

In [ ]:
```
image_size = 299
batch_size = 400
imgs = 10

with tf.Graph().as_default():
```

```
tf.logging.set_verbosity(tf.logging.INFO)

dataset = cve_diseases.get_split('validation', dataset_dir)
images, images_raw, labels = load_batch(dataset, batch_size)

# Create the model, use the default arg scope to configure the batch norm parameters

logits = jonet(images)

probabilities = tf.nn.softmax(logits)
con_mat = tf.confusion_matrix(
    labels=labels,
    predictions=tf.argmax(probabilities, 1))


predictions = tf.argmax(probabilities, 1)
accuracy = tf.metrics.accuracy(labels, predictions)


checkpoint_path = tf.train.latest_checkpoint(train_dir)


init_fn = slim.assign_from_checkpoint_fn(
  checkpoint_path,
  slim.get_variables_to_restore())
print(tf.argmax(labels))
with tf.Session() as sess:
    with slim.queues.QueueRunners(sess):
        sess.run(tf.initialize_local_variables())
        init_fn(sess)
        accuracy, np_predictions, np_probabilities, np_images_raw, np_labels, con_ma


        plt.imshow(con_mat, interpolation='nearest', cmap=plt.cm.viridis);
        plt.title("Confusion Matrix")
        plt.colorbar()
        plt.clim(0,30)
        plt.xticks(rotation=0)
        plt.yticks(rotation=0)
        plt.tight_layout()
        plt.ylabel("True label")
        plt.xlabel("Predicted label")
        plt.show()
        print(con_mat)

        precision, recall, f1, _ = score(np_labels, np_predictions, average='macro')

        print('precision: {}'.format(precision))
```

```
print('recall: {}'.format(recall))
print('fscore: {}'.format(f1))
print('accuracy: {}'.format(accuracy))

for i in range(imgs):
    image = np_images_raw[i, :, :, :]
    true_label = np_labels[i]
    predicted_label = np.argmax(np_probabilities[i, :])
    predicted_name = dataset.labels_to_names[predicted_label]
    true_name = dataset.labels_to_names[true_label]


    plt.figure()
    plt.imshow(image.astype(np.uint8))
    plt.title('Ground Truth: [%s], Prediction [%s]' % (true_name, predicted_
    plt.axis('off')
    plt.show()
```

eval_inception_resnet

May 31, 2018

In [ ]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
import math
import numpy as np
import tensorflow as tf
import time
import re
from sklearn.metrics import precision_recall_fscore_support as score

from datasets import dataset_utils
from datasets import cve_diseases
from nets import inception_resnet_v2
from nets import inception_v4

from nets import nets_factory
from preprocessing import inception_preprocessing
# Main slim library
from tensorflow.contrib import slim
```

In [ ]:
```
model1="inception_resnet_v2"
dataset_name = "cve_diseases"
dataset_split_name = "validation"
dataset_dir = "tmp/"

train_dir = "./tmp/cve_diseases-models/model3_ds2-finetune"

cnn1 = nets_factory.get_network_fn(
    model1,
    num_classes=11,
    weight_decay=0.00004,
    is_training=False)
```

In [ ]:
```
```

```
In [ ]: def load_batch(dataset, batch_size=8, height=299, width=299, is_training=False):

            data_provider = slim.dataset_data_provider.DatasetDataProvider(dataset)

            image_raw, label = data_provider.get(['image', 'label'])

            # Preprocess image for usage by Inception.
            image = inception_preprocessing.preprocess_image(image_raw, height, width, is_traini

            # Preprocess the image for display purposes.
            image_raw = tf.expand_dims(image_raw, 0)
            image_raw = tf.image.resize_images(image_raw, [height, width])
            image_raw = tf.squeeze(image_raw)

            # Batch it up.
            images, images_raw, labels = tf.train.batch(
                  [image, image_raw, label],
                  batch_size=batch_size,
                  num_threads=1,
                  capacity=2 * batch_size)

            return images, images_raw, labels
```

```
In [ ]:
        image_size = 299
        batch_size = 400
        imgs = 10

        with tf.Graph().as_default():
            tf.logging.set_verbosity(tf.logging.INFO)

            dataset = cve_diseases.get_split('validation', dataset_dir)
            images, images_raw, labels = load_batch(dataset, batch_size)

            # Create the model, use the default arg scope to configure the batch norm parameters

            logits, _ = cnn1(images)

            probabilities = tf.nn.softmax(logits)
            con_mat = tf.confusion_matrix(
                labels=labels,
                predictions=tf.argmax(probabilities, 1))

            predictions = tf.argmax(probabilities, 1)
            accuracy = tf.metrics.accuracy(labels, predictions)

            checkpoint_path = tf.train.latest_checkpoint(train_dir)
            init_fn = slim.assign_from_checkpoint_fn(
```

```
  checkpoint_path,
  slim.get_variables_to_restore())

with tf.Session() as sess:
    with slim.queues.QueueRunners(sess):
        sess.run(tf.initialize_local_variables())
        init_fn(sess)
        accuracy, np_predictions, np_probabilities, np_images_raw, np_labels, con_ma


        plt.imshow(con_mat, interpolation='nearest', cmap=plt.cm.viridis);
        plt.title("Confusion Matrix")
        plt.colorbar()
        plt.clim(0,30)
        plt.xticks(rotation=0)
        plt.yticks(rotation=0)
        plt.tight_layout()
        plt.ylabel("True label")
        plt.xlabel("Predicted label")
        plt.show()
        print(con_mat)
        precision, recall, f1, _ = score(np_labels, np_predictions, average='macro')

        print('precision: {}'.format(precision))
        print('recall: {}'.format(recall))
        print('fscore: {}'.format(f1))
        print('accuracy: {}'.format(accuracy))
        for i in range(imgs):
            image = np_images_raw[i, :, :, :]
            true_label = np_labels[i]
            predicted_label = np.argmax(np_probabilities[i, :])
            predicted_name = dataset.labels_to_names[predicted_label]
            true_name = dataset.labels_to_names[true_label]


            plt.figure()
            plt.imshow(image.astype(np.uint8))
            plt.title('Ground Truth: [%s], Prediction [%s]' % (true_name, predicted_
            plt.axis('off')
            plt.show()
```

B Training

B.1 train_image_classifier.py

```
# Copyright 2016 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, soft
# distributed under the License is distributed on an "AS IS" BASI
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# See the License for the specific language governing permissions
# limitations under the License.
# ==============================================================
"""Generic training script that trains a model using a given data

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf

from datasets import dataset_factory
from deployment import model_deploy
from nets import nets_factory
from preprocessing import preprocessing_factory

slim = tf.contrib.slim

tf.app.flags.DEFINE_string(
    'master', '', 'The address of the TensorFlow master to use.')

tf.app.flags.DEFINE_string(
    'train_dir', '/tmp/tfmodel/',
    'Directory where checkpoints and event logs are written to.')
```

```
tf.app.flags.DEFINE_integer('num_clones', 1,
                            'Number␣of␣model␣clones␣to␣deploy.')

tf.app.flags.DEFINE_boolean('clone_on_cpu', False,
                            'Use␣CPUs␣to␣deploy␣clones.')

tf.app.flags.DEFINE_integer('worker_replicas', 1, 'Number␣of␣worker␣repl

tf.app.flags.DEFINE_integer(
    'num_ps_tasks', 0,
    'The␣number␣of␣parameter␣servers.␣If␣the␣value␣is␣0,␣then␣the␣parame
    'are␣handled␣locally␣by␣the␣worker.')

tf.app.flags.DEFINE_integer(
    'num_readers', 4,
    'The␣number␣of␣parallel␣readers␣that␣read␣data␣from␣the␣dataset.')

tf.app.flags.DEFINE_integer(
    'num_preprocessing_threads', 4,
    'The␣number␣of␣threads␣used␣to␣create␣the␣batches.')

tf.app.flags.DEFINE_integer(
    'log_every_n_steps', 2,
    'The␣frequency␣with␣which␣logs␣are␣print.')

tf.app.flags.DEFINE_integer(
    'save_summaries_secs', 600,
    'The␣frequency␣with␣which␣summaries␣are␣saved,␣in␣seconds.')

tf.app.flags.DEFINE_integer(
    'save_interval_secs', 600,
    'The␣frequency␣with␣which␣the␣model␣is␣saved,␣in␣seconds.')

tf.app.flags.DEFINE_integer(
    'task', 0, 'Task␣id␣of␣the␣replica␣running␣the␣training.')

######################
# Optimization Flags #
######################

tf.app.flags.DEFINE_float(
    'weight_decay', 0.00004, 'The␣weight␣decay␣on␣the␣model␣weights.')

tf.app.flags.DEFINE_string(
    'optimizer', 'rmsprop',
```

```
    'The␣name␣of␣the␣optimizer,␣one␣of␣"adadelta",␣"adagrad",␣"ad
    '"ftrl",␣"momentum",␣"sgd"␣or␣"rmsprop".')

tf.app.flags.DEFINE_float(
    'adadelta_rho', 0.95,
    'The␣decay␣rate␣for␣adadelta.')

tf.app.flags.DEFINE_float(
    'adagrad_initial_accumulator_value', 0.1,
    'Starting␣value␣for␣the␣AdaGrad␣accumulators.')

tf.app.flags.DEFINE_float(
    'adam_beta1', 0.9,
    'The␣exponential␣decay␣rate␣for␣the␣1st␣moment␣estimates.')

tf.app.flags.DEFINE_float(
    'adam_beta2', 0.999,
    'The␣exponential␣decay␣rate␣for␣the␣2nd␣moment␣estimates.')

tf.app.flags.DEFINE_float('opt_epsilon', 1.0, 'Epsilon␣term␣for␣t

tf.app.flags.DEFINE_float('ftrl_learning_rate_power', -0.5,
                          'The␣learning␣rate␣power.')

tf.app.flags.DEFINE_float(
    'ftrl_initial_accumulator_value', 0.1,
    'Starting␣value␣for␣the␣FTRL␣accumulators.')

tf.app.flags.DEFINE_float(
    'ftrl_l1', 0.0, 'The␣FTRL␣l1␣regularization␣strength.')

tf.app.flags.DEFINE_float(
    'ftrl_l2', 0.0, 'The␣FTRL␣l2␣regularization␣strength.')

tf.app.flags.DEFINE_float(
    'momentum', 0.9,
    'The␣momentum␣for␣the␣MomentumOptimizer␣and␣RMSPropOptimizer.

tf.app.flags.DEFINE_float('rmsprop_momentum', 0.9, 'Momentum.')

tf.app.flags.DEFINE_float('rmsprop_decay', 0.9, 'Decay␣term␣for␣R

#######################
# Learning Rate Flags #
#######################
```

```
tf.app.flags.DEFINE_string(
    'learning_rate_decay_type',
    'exponential',
    'Specifies how the learning rate is decayed. One of "fixed", "expone
    ' or "polynomial"')

tf.app.flags.DEFINE_float('learning_rate', 0.01, 'Initial learning rate.

tf.app.flags.DEFINE_float(
    'end_learning_rate', 0.0001,
    'The minimal end learning rate used by a polynomial decay learning r

tf.app.flags.DEFINE_float(
    'label_smoothing', 0.0, 'The amount of label smoothing.')

tf.app.flags.DEFINE_float(
    'learning_rate_decay_factor', 0.94, 'Learning rate decay factor.')

tf.app.flags.DEFINE_float(
    'num_epochs_per_decay', 2.0,
    'Number of epochs after which learning rate decays.')

tf.app.flags.DEFINE_bool(
    'sync_replicas', False,
    'Whether or not to synchronize the replicas during training.')

tf.app.flags.DEFINE_integer(
    'replicas_to_aggregate', 1,
    'The Number of gradients to collect before updating params.')

tf.app.flags.DEFINE_float(
    'moving_average_decay', None,
    'The decay to use for the moving average.'
    'If left as None, then moving averages are not used.')

#######################
# Dataset Flags #
#######################

tf.app.flags.DEFINE_string(
    'dataset_name', 'imagenet', 'The name of the dataset to load.')

tf.app.flags.DEFINE_string(
    'dataset_split_name', 'train', 'The name of the train/test split.')
```

```
tf.app.flags.DEFINE_string(
    'dataset_dir', None, 'The␣directory␣where␣the␣dataset␣files␣a

tf.app.flags.DEFINE_integer(
    'labels_offset', 0,
    'An␣offset␣for␣the␣labels␣in␣the␣dataset.␣This␣flag␣is␣primar
    'evaluate␣the␣VGG␣and␣ResNet␣architectures␣which␣do␣not␣use␣a
    'class␣for␣the␣ImageNet␣dataset.')

tf.app.flags.DEFINE_string(
    'model_name', 'inception_v3', 'The␣name␣of␣the␣architecture␣t

tf.app.flags.DEFINE_string(
    'preprocessing_name', None, 'The␣name␣of␣the␣preprocessing␣to
    'as␣`None`,␣then␣the␣model_name␣flag␣is␣used.')

tf.app.flags.DEFINE_integer(
    'batch_size', 32, 'The␣number␣of␣samples␣in␣each␣batch.')

tf.app.flags.DEFINE_integer(
    'train_image_size', None, 'Train␣image␣size')

tf.app.flags.DEFINE_integer('max_number_of_steps', None,
                            'The␣maximum␣number␣of␣training␣steps

#####################
# Fine-Tuning Flags #
#####################

tf.app.flags.DEFINE_string(
    'checkpoint_path', None,
    'The␣path␣to␣a␣checkpoint␣from␣which␣to␣fine-tune.')

tf.app.flags.DEFINE_string(
    'checkpoint_exclude_scopes', None,
    'Comma-separated␣list␣of␣scopes␣of␣variables␣to␣exclude␣when␣
    'from␣a␣checkpoint.')

tf.app.flags.DEFINE_string(
    'trainable_scopes', None,
    'Comma-separated␣list␣of␣scopes␣to␣filter␣the␣set␣of␣variable
    'By␣default,␣None␣would␣train␣all␣the␣variables.')

tf.app.flags.DEFINE_boolean(
```

```
    'ignore_missing_vars', False,
    'When␣restoring␣a␣checkpoint␣would␣ignore␣missing␣variables.')

FLAGS = tf.app.flags.FLAGS


def _configure_learning_rate(num_samples_per_epoch, global_step):
  """Configures the learning rate.

  Args:
    num_samples_per_epoch: The number of samples in each epoch of traini
    global_step: The global_step tensor.

  Returns:
    A `Tensor` representing the learning rate.

  Raises:
    ValueError: if
  """
  decay_steps = int(num_samples_per_epoch / FLAGS.batch_size *
                    FLAGS.num_epochs_per_decay)
  if FLAGS.sync_replicas:
    decay_steps /= FLAGS.replicas_to_aggregate

  if FLAGS.learning_rate_decay_type == 'exponential':
    return tf.train.exponential_decay(FLAGS.learning_rate,
                                      global_step,
                                      decay_steps,
                                      FLAGS.learning_rate_decay_factor,
                                      staircase=True,
                                      name='exponential_decay_learning_r
  elif FLAGS.learning_rate_decay_type == 'fixed':
    return tf.constant(FLAGS.learning_rate, name='fixed_learning_rate')
  elif FLAGS.learning_rate_decay_type == 'polynomial':
    return tf.train.polynomial_decay(FLAGS.learning_rate,
                                     global_step,
                                     decay_steps,
                                     FLAGS.end_learning_rate,
                                     power=1.0,
                                     cycle=False,
                                     name='polynomial_decay_learning_rat
  else:
    raise ValueError('learning_rate_decay_type␣[%s]␣was␣not␣recognized',
                     FLAGS.learning_rate_decay_type)
```

```
def _configure_optimizer(learning_rate):
  """Configures the optimizer used for training.

  Args:
    learning_rate: A scalar or `Tensor` learning rate.

  Returns:
    An instance of an optimizer.

  Raises:
    ValueError: if FLAGS.optimizer is not recognized.
  """
  if FLAGS.optimizer == 'adadelta':
    optimizer = tf.train.AdadeltaOptimizer(
        learning_rate,
        rho=FLAGS.adadelta_rho,
        epsilon=FLAGS.opt_epsilon)
  elif FLAGS.optimizer == 'adagrad':
    optimizer = tf.train.AdagradOptimizer(
        learning_rate,
        initial_accumulator_value=FLAGS.adagrad_initial_accumulat
  elif FLAGS.optimizer == 'adam':
    optimizer = tf.train.AdamOptimizer(
        learning_rate,
        beta1=FLAGS.adam_beta1,
        beta2=FLAGS.adam_beta2,
        epsilon=FLAGS.opt_epsilon)
  elif FLAGS.optimizer == 'ftrl':
    optimizer = tf.train.FtrlOptimizer(
        learning_rate,
        learning_rate_power=FLAGS.ftrl_learning_rate_power,
        initial_accumulator_value=FLAGS.ftrl_initial_accumulator_
        l1_regularization_strength=FLAGS.ftrl_l1,
        l2_regularization_strength=FLAGS.ftrl_l2)
  elif FLAGS.optimizer == 'momentum':
    optimizer = tf.train.MomentumOptimizer(
        learning_rate,
        momentum=FLAGS.momentum,
        name='Momentum')
  elif FLAGS.optimizer == 'rmsprop':
    optimizer = tf.train.RMSPropOptimizer(
        learning_rate,
        decay=FLAGS.rmsprop_decay,
        momentum=FLAGS.rmsprop_momentum,
```

```
        epsilon=FLAGS.opt_epsilon)
  elif FLAGS.optimizer == 'sgd':
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
  else:
    raise ValueError('Optimizer [%s] was not recognized', FLAGS.optimize
  return optimizer


def _get_init_fn():
  """Returns a function run by the chief worker to warm-start the traini

  Note that the init_fn is only run when initializing the model during t
  first global step.

  Returns:
    An init function run by the supervisor.
  """
  if FLAGS.checkpoint_path is None:
    return None

  # Warn the user if a checkpoint exists in the train_dir. Then we'll be
  # ignoring the checkpoint anyway.
  if tf.train.latest_checkpoint(FLAGS.train_dir):
    tf.logging.info(
        'Ignoring --checkpoint_path because a checkpoint already exists
        % FLAGS.train_dir)
    return None

  exclusions = []
  if FLAGS.checkpoint_exclude_scopes:
    exclusions = [scope.strip()
                  for scope in FLAGS.checkpoint_exclude_scopes.split(',

  # TODO(sguada) variables.filter_variables()
  variables_to_restore = []
  for var in slim.get_model_variables():
    excluded = False
    for exclusion in exclusions:
      if var.op.name.startswith(exclusion):
        excluded = True
        break
    if not excluded:
      variables_to_restore.append(var)

  if tf.gfile.IsDirectory(FLAGS.checkpoint_path):
```

```
    checkpoint_path = tf.train.latest_checkpoint(FLAGS.checkpoint
  else:
    checkpoint_path = FLAGS.checkpoint_path

  tf.logging.info('Fine-tuning␣from␣%s' % checkpoint_path)

  return slim.assign_from_checkpoint_fn(
      checkpoint_path,
      variables_to_restore,
      ignore_missing_vars=FLAGS.ignore_missing_vars)


def _get_variables_to_train():
  """Returns a list of variables to train.

  Returns:
    A list of variables to train by the optimizer.
  """
  if FLAGS.trainable_scopes is None:
    return tf.trainable_variables()
  else:
    scopes = [scope.strip() for scope in FLAGS.trainable_scopes.s

  variables_to_train = []
  for scope in scopes:
    variables = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES
    variables_to_train.extend(variables)
  return variables_to_train


def main(_):
  if not FLAGS.dataset_dir:
    raise ValueError('You␣must␣supply␣the␣dataset␣directory␣with␣

  tf.logging.set_verbosity(tf.logging.INFO)
  with tf.Graph().as_default():
    ######################
    # Config model_deploy #
    ######################
    deploy_config = model_deploy.DeploymentConfig(
        num_clones=FLAGS.num_clones,
        clone_on_cpu=FLAGS.clone_on_cpu,
        replica_id=FLAGS.task,
        num_replicas=FLAGS.worker_replicas,
        num_ps_tasks=FLAGS.num_ps_tasks)
```

```
# Create global_step
with tf.device(deploy_config.variables_device()):
  global_step = slim.create_global_step()

######################
# Select the dataset #
######################
dataset = dataset_factory.get_dataset(
    FLAGS.dataset_name, FLAGS.dataset_split_name, FLAGS.dataset_dir)

######################
# Select the network #
######################
network_fn = nets_factory.get_network_fn(
    FLAGS.model_name,
    num_classes=(dataset.num_classes - FLAGS.labels_offset),
    weight_decay=FLAGS.weight_decay,
    is_training=True)

#####################################
# Select the preprocessing function #
#####################################
preprocessing_name = FLAGS.preprocessing_name or FLAGS.model_name
image_preprocessing_fn = preprocessing_factory.get_preprocessing(
    preprocessing_name,
    is_training=True)

##############################################################
# Create a dataset provider that loads data from the dataset #
##############################################################
with tf.device(deploy_config.inputs_device()):
  provider = slim.dataset_data_provider.DatasetDataProvider(
      dataset,
      num_readers=FLAGS.num_readers,
      common_queue_capacity=20 * FLAGS.batch_size,
      common_queue_min=10 * FLAGS.batch_size)
  [image, label] = provider.get(['image', 'label'])
  label -= FLAGS.labels_offset

  train_image_size = FLAGS.train_image_size or network_fn.default_im

  image = image_preprocessing_fn(image, train_image_size, train_imag

  images, labels = tf.train.batch(
```

```
        [image, label],
        batch_size=FLAGS.batch_size,
        num_threads=FLAGS.num_preprocessing_threads,
        capacity=5 * FLAGS.batch_size)
    labels = slim.one_hot_encoding(
        labels, dataset.num_classes - FLAGS.labels_offset)
    batch_queue = slim.prefetch_queue.prefetch_queue(
        [images, labels], capacity=2 * deploy_config.num_clones

####################
# Define the model #
####################
def clone_fn(batch_queue):
  """Allows data parallelism by creating multiple clones of n
  images, labels = batch_queue.dequeue()
  logits, end_points = network_fn(images)

  #############################
  # Specify the loss function #
  #############################
  if 'AuxLogits' in end_points:
    slim.losses.softmax_cross_entropy(
        end_points['AuxLogits'], labels,
        label_smoothing=FLAGS.label_smoothing, weights=0.4,
        scope='aux_loss')
  slim.losses.softmax_cross_entropy(
      logits, labels, label_smoothing=FLAGS.label_smoothing,
  return end_points

# Gather initial summaries.
summaries = set(tf.get_collection(tf.GraphKeys.SUMMARIES))

clones = model_deploy.create_clones(deploy_config, clone_fn,
first_clone_scope = deploy_config.clone_scope(0)
# Gather update_ops from the first clone. These contain, for
# the updates for the batch_norm variables created by network
update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS, first

# Add summaries for end_points.
end_points = clones[0].outputs
for end_point in end_points:
  x = end_points[end_point]
  summaries.add(tf.summary.histogram('activations/' + end_poi
  summaries.add(tf.summary.scalar('sparsity/' + end_point,
                                  tf.nn.zero_fraction(x)))
```

```
# Add summaries for losses.
for loss in tf.get_collection(tf.GraphKeys.LOSSES, first_clone_scope
  summaries.add(tf.summary.scalar('losses/%s' % loss.op.name, loss))

# Add summaries for variables.
for variable in slim.get_model_variables():
  summaries.add(tf.summary.histogram(variable.op.name, variable))

#################################
# Configure the moving averages #
#################################
if FLAGS.moving_average_decay:
  moving_average_variables = slim.get_model_variables()
  variable_averages = tf.train.ExponentialMovingAverage(
      FLAGS.moving_average_decay, global_step)
else:
  moving_average_variables, variable_averages = None, None

########################################
# Configure the optimization procedure. #
########################################
with tf.device(deploy_config.optimizer_device()):
  learning_rate = _configure_learning_rate(dataset.num_samples, glob
  optimizer = _configure_optimizer(learning_rate)
  summaries.add(tf.summary.scalar('learning_rate', learning_rate))

if FLAGS.sync_replicas:
  # If sync_replicas is enabled, the averaging will be done in the c
  # queue runner.
  optimizer = tf.train.SyncReplicasOptimizer(
      opt=optimizer,
      replicas_to_aggregate=FLAGS.replicas_to_aggregate,
      total_num_replicas=FLAGS.worker_replicas,
      variable_averages=variable_averages,
      variables_to_average=moving_average_variables)
elif FLAGS.moving_average_decay:
  # Update ops executed locally by trainer.
  update_ops.append(variable_averages.apply(moving_average_variables

# Variables to train.
variables_to_train = _get_variables_to_train()

#  and returns a train_tensor and summary_op
total_loss, clones_gradients = model_deploy.optimize_clones(
```

```
        clones,
        optimizer,
        var_list=variables_to_train)
    # Add total_loss to summary.
    summaries.add(tf.summary.scalar('total_loss', total_loss))

    # Create gradient updates.
    grad_updates = optimizer.apply_gradients(clones_gradients,
                                             global_step=global_s
    update_ops.append(grad_updates)

    update_op = tf.group(*update_ops)
    with tf.control_dependencies([update_op]):
      train_tensor = tf.identity(total_loss, name='train_op')

    # Add the summaries from the first clone. These contain the s
    # created by model_fn and either optimize_clones() or _gather
    summaries |= set(tf.get_collection(tf.GraphKeys.SUMMARIES,
                                       first_clone_scope))

    # Merge all summaries together.
    summary_op = tf.summary.merge(list(summaries), name='summary_


    ############################
    # Kicks off the training. #
    ############################
    slim.learning.train(
        train_tensor,
        logdir=FLAGS.train_dir,
        master=FLAGS.master,
        is_chief=(FLAGS.task == 0),
        init_fn=_get_init_fn(),
        summary_op=summary_op,
        number_of_steps=FLAGS.max_number_of_steps,
        log_every_n_steps=FLAGS.log_every_n_steps,
        save_summaries_secs=FLAGS.save_summaries_secs,
        save_interval_secs=FLAGS.save_interval_secs,
        sync_optimizer=optimizer if FLAGS.sync_replicas else None


if __name__ == '__main__':
  tf.app.run()
```

B.2 Training script for proposed architecture

train

May 31, 2018

IMPORTS

In [ ]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf
from datasets import cve_diseases
from nets import inception_resnet_v2
from nets import alexnet
from nets import nets_factory
from preprocessing import inception_preprocessing

slim = tf.contrib.slim
```

CONFIGURE VARIABLES

In [ ]:
```
model1 = "inception_resnet_v2"
model2 = "inception_v3"

dataset_name = "cve_diseases"
dataset_split_name = "train"
dataset_dir = "tmp/"
batch_size = 16
max_number_of_steps = 100000
train_dir = "./tmp/cve_diseases-models/model2-ds2/"

cnn1 = nets_factory.get_network_fn(
    model1,
    num_classes=None,
    weight_decay=0.00004,
    is_training=True)
cnn2 = nets_factory.get_network_fn(
    model2,
    num_classes=None,
    weight_decay=0.00004,
    is_training=True)
```

DEFINE FUNCTIONS

```
In [ ]: def jonet(images):

            net1, end_points1 = cnn1(images)
            net1 = end_points1['Mixed_7a']

            net2, end_points2 = cnn2(images)
            net2 = end_points2['Mixed_7c']
            net = tf.concat((net1, net2), 3)
            net = slim.flatten(net)
            net = slim.dropout(net, 0.8, is_training=True)
            net = slim.fully_connected(net, 11, activation_fn=None)
            return net
```

```
In [ ]: def load_batch(dataset, batch_size=8, height=299, width=299, is_training=False):
            data_provider = slim.dataset_data_provider.DatasetDataProvider(dataset)

            image, label = data_provider.get(['image', 'label'])

            image = inception_preprocessing.preprocess_image(
                image,
                height,
                width,
                is_training)
            one_hot_labels = slim.one_hot_encoding(label, dataset.num_classes)


            images, labels = tf.train.batch(
                [image, one_hot_labels],
                batch_size=batch_size,
                allow_smaller_final_batch=True)

            return images, labels
```

MAIN

```
In [ ]: #Select dataset
        dataset = cve_diseases.get_split('train', dataset_dir)

        #Load batch
        images, labels = load_batch(
            dataset,
            batch_size,
            is_training=True)

        # run the image through the model

        logits = jonet(images)
```

```
# get the cross-entropy loss

loss = slim.losses.softmax_cross_entropy(logits, labels)
total_loss = slim.losses.get_total_loss()

#Optimizer
optimizer = tf.train.AdamOptimizer(0.00001)



predictions = tf.argmax(logits, 1)
targets = tf.argmax(labels, 1)

correct_prediction = tf.equal(predictions, targets)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

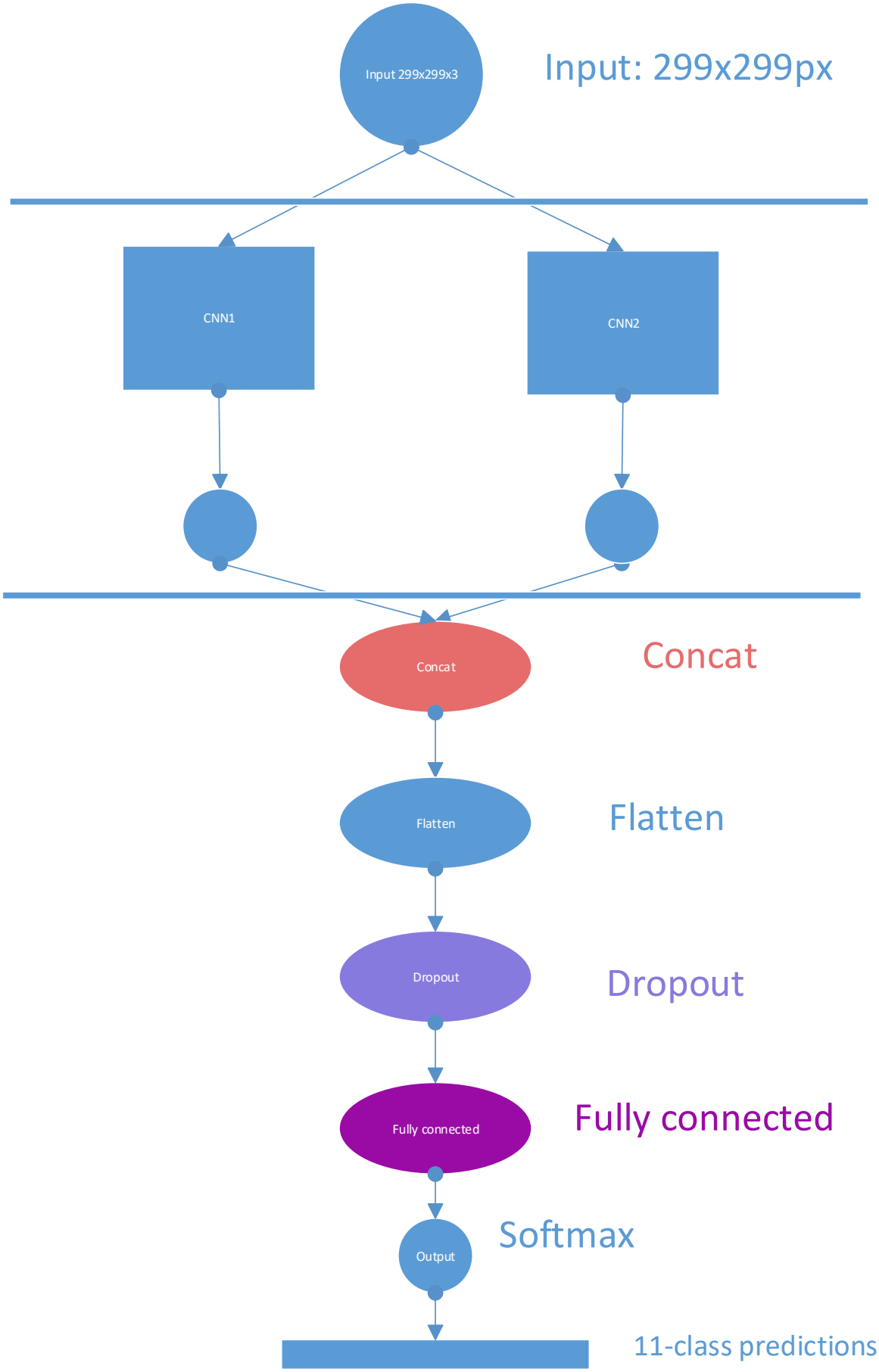

#Write summary
tf.summary.scalar('losses/total_loss', total_loss)
tf.summary.scalar('Accuracy', accuracy)
summary_op = tf.summary.merge_all()


# create train op
train_op = slim.learning.create_train_op(
    total_loss,
    optimizer,
    summarize_gradients=True)

# run training
slim.learning.train(
    train_op,
    logdir=train_dir,
    number_of_steps=max_number_of_steps,
    summary_op=summary_op,
    save_summaries_secs=30,
    save_interval_secs=30,
    log_every_n_steps=100)
```

Helper funcs for endpoint in end_points1: print("Endpoint: %s - Shape: %s" % (endpoint, end_points1[endpoint].get_shape())) print("################################") for endpoint in end_points2: print("Endpoint: %s - Shape: %s" % (endpoint , end_points2[endpoint].get_shape()))

C Proposed network


Input 299x299x3
Input: 299x299px
CNN1
CNN2
Concat
Concat
Flatten
Flatten
Dropout
Dropout
Fully connected
Fully connected
Output
Softmax
11-class predictions

D Main code for CveClassifier

```
import sys, os
import time
import tensorflow as tf
import funcs
import shutil
import numpy as np
from PyQt4 import QtCore, QtGui


from cveGui import Ui_MainWindow


class Window(QtGui.QMainWindow, Ui_MainWindow):

    def __init__(self, parent=None):
        super(Window, self).__init__(parent)
        self.setupUi(self)
        self.progressBar.setValue(0)
        self.plainTextEdit.appendPlainText("##########INSTRUCTION
        self.plainTextEdit.appendPlainText("1. Select the folder 
        self.plainTextEdit.appendPlainText("2. Select the folder 
        self.plainTextEdit.appendPlainText("3. Select the inferen
        self.plainTextEdit.appendPlainText("4. Click \"Classify\"

        #Define button click actions
        self.pushButton_3.clicked.connect(self.browseImg)
        self.pushButton_4.clicked.connect(self.selectOutputFolder
        self.pushButton_5.clicked.connect(self.selectGraph)
        self.pushButton.clicked.connect(self.classifyImages)
        self.pushButton_2.clicked.connect(self.reset)



    #Methods related to button clicks
    def classifyImages(self):
        imgFolder = self.label_3.text()
        outFolder = self.label_5.text()
        graph = self.label_6.text()
```

```
        if not imgFolder:
            self.plainTextEdit.appendPlainText("Image␣folder␣must␣be␣sel
        if not outFolder:
            self.plainTextEdit.appendPlainText("Output␣folder␣must␣be␣se
        if imgFolder and outFolder and graph:
            self.plainTextEdit.appendPlainText("Starting␣classification␣

            self.progressBar.setMaximum(len(os.listdir(imgFolder)))
            self.progressBar.setValue(0)
            self.cThread = classifyThread(
                self.label_3.text(),
                self.label_5.text(),
                self.label_6.text())

            self.connect(self.cThread, QtCore.SIGNAL("finished()"), self
            self.connect(self.cThread, QtCore.SIGNAL("classified(QString
            self.connect(self.cThread, QtCore.SIGNAL("progress()"), self

            self.cThread.start()

    def selectOutputFolder(self):
         name = str(QtGui.QFileDialog.getExistingDirectory(self, 'Select
         self.label_5.setText(name)
         text = "Output␣folder␣set␣to:␣" + self.label_5.text()
         self.plainTextEdit.appendPlainText(text)
    def browseImg(self):
        name = str(QtGui.QFileDialog.getExistingDirectory(self, 'Open␣fi
    #     file = open(name, 'r')
        self.label_3.setText(name)
        text = "Image␣folder␣set␣to:␣" + self.label_3.text()
        self.plainTextEdit.appendPlainText(text)
    def selectGraph(self):
        name = QtGui.QFileDialog.getOpenFileName(self, 'Open␣file')
        file = open(name, 'r')
        self.label_6.setText(file.name)
        text = "Inference␣graph␣selected:␣" + self.label_6.text()
        self.plainTextEdit.appendPlainText(text)
    def reset(self):
        self.label_3.clear()
        self.label_5.clear()
        self.label_6.clear()
    def log(self, text):
        self.plainTextEdit.appendPlainText(text)
    def add(self):
        self.progressBar.setValue(self.progressBar.value()+1)
```

```
    #Signals
    def done(self):
        QtGui.QMessageBox.information(self, "Complete!", "Done␣cl
        self.plainTextEdit.appendPlainText('Classification␣comple
#Threading
class classifyThread(QtCore.QThread):
    def __init__(self, imgf, outf, g):
        QtCore.QThread.__init__(self)
        self.imgf = imgf
        self.outf = outf
        self.g = g


    def __del__(self):
        self.wait()

    def run(self):
        graph = funcs.load_graph(self.g)
        input_name = "prefix/input"
        output_name = "prefix/InceptionResnetV2/Logits/Prediction
        input_op = graph.get_operation_by_name(input_name)
        output_op = graph.get_operation_by_name(output_name)
        for file in os.listdir(self.imgf):
            file_name = self.imgf + "/" + file
            t = funcs.read_image_file(
                file_name,
                input_height=299,
                input_width=299,
                input_mean=0,
                input_std=255)
            with tf.Session(graph=graph) as sess:
                results = sess.run(output_op.outputs[0], {
                    input_op.outputs[0]: t
                    })
            results = np.squeeze(results)

            top_k = results.argsort()[-1:][::-1]
            label_file = "./graphs/labels.txt"
            labels = funcs.load_labels(label_file)

            text = "Prediction␣on␣image:␣" + file
            self.emit(QtCore.SIGNAL('classified(QString)'), text)
```

```
            for i in top_k:
                disease = labels[i]
                _, disease = disease.split(":")
                if not os.path.exists(self.outf + "/" + disease):
                    os.makedirs(self.outf + "/" + disease)
                shutil.copy2(file_name, self.outf + "/" + disease)
                text = labels[i] + "␣=>␣" + str(results[i])
                self.emit(QtCore.SIGNAL('classified(QString)'), text)

            self.emit(QtCore.SIGNAL('progress()'))



app = QtGui.QApplication(sys.argv)
w = Window()
w.show()
sys.exit(app.exec_())
```

E Helper scripts

Copydata

```
#Copies labeled data from imgAnnotation tool to
# 'destination' folder based on disease names in the
# 'diseases' list
# Configure source and destination for your needs.
#
import sys
import os
import shutil
################### Configure parameters#############################
#Source for annotation databases(Absoulute path):
source = "/home/johnny/imgAnnotation-master/imgAnnotation-master/

#Destination for output files:#####################################
destination = "/home/johnny/Database/"
#Check for these diseases:##########################################
diseases = ["diverticulosis", "polyp", "uc", "chrons", "haemorrha
"erosions", "angioectasia", "ulceration", "oedema", "erythema", "
"aphta", "granularity", "haemmorhoids", "pseudopolyps"]

#Include following patients:#######################################
patients = ["P1", "P2", "P3", "P4", "P5", "P7", "P9", "P11", "P13
 "P17", "P19", "P21", "P23", "P25"]

 #Folder for non-disease image default: normal#####################
normal = 'normal'
###################################################################
#### Helper vars ###
path = False
cleanseLevel = False
informative = False
fil = False
copied = False
####################

#Loop through patient files
for patient in patients:
    with open(source + patient + '.annotation') as f:   #Open cor
```

```
        for line in f:                                          #Read line by li
            if line.startswith("file:"):
                fil, path = line.split()
            if line.startswith("good_cleanse:"):
                if len(line.split()) > 1:
                    fil, cleanseLevel = line.split()
            if line.startswith("informative:"):
                if len(line.split()) > 1:
                    fil, informative = line.split()
            if any(line.startswith(disease) for disease in diseases):
                disease, fil = line.split(':')
                if not os.path.exists(destination + disease):
                    os.makedirs(destination + disease)
                if os.path.exists(path):
                    shutil.copy2(path, destination + disease)
                    copied = True
            if line.startswith("##########"):
                if not copied:
                    if path and int(cleanseLevel) < 3 and int(cleanseLev
                        if not os.path.exists(destination + normal):
                            os.makedirs(destination + normal)
                        if os.path.exists(path):
                            shutil.copy2(path, destination + normal)

                path = False
                cleanseLevel = False
                informative = False
                copied = False
```

Convert data

```
# Copyright 2016 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implie
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
r"""Converts␣a␣particular␣dataset.
```

```
Usage:
```shell
$ python convert_data.py \
    --dataset_name=cve_diseases \
    --dataset_dir=tmp/cve_diseases
```
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import tensorflow as tf

from datasets import convert_cve_diseases

FLAGS = tf.app.flags.FLAGS

tf.app.flags.DEFINE_string(
    'dataset_name',
    None,
    'The name of the dataset to convert, one of "cifar10", "flowe

tf.app.flags.DEFINE_string(
    'dataset_dir',
    None,
    'The directory where the output TFRecords and temporary files

def main(_):
  if not FLAGS.dataset_name:
    raise ValueError('You must supply the dataset name with --dat
  if not FLAGS.dataset_dir:
    raise ValueError('You must supply the dataset directory with

  if FLAGS.dataset_name == 'cve_diseases':
    convert_cve_diseases.run(FLAGS.dataset_dir)
  else:
    raise ValueError(
        'dataset_name [%s] was not recognized.' % FLAGS.dataset_n

if __name__ == '__main__':
  tf.app.run()
```

```
  # Copyright 2016 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implie
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
"""Provides data for the flowers dataset.

The dataset scripts used to create the dataset can be found at:
tensorflow/models/research/slim/datasets/download_and_convert_flowers.py
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import tensorflow as tf

from datasets import dataset_utils

slim = tf.contrib.slim

_FILE_PATTERN = 'cve_diseases_%s_*.tfrecord'

SPLITS_TO_SIZES = {'train': 2940, 'validation': 327}

_NUM_CLASSES = 11

_ITEMS_TO_DESCRIPTIONS = {
    'image': 'A color image of varying size.',
    'label': 'A single integer between 0 and 4',
}
```

```
def get_split(split_name, dataset_dir, file_pattern=None, reader=
  """Gets a dataset tuple with instructions for reading flowers.

  Args:
    split_name: A train/validation split name.
    dataset_dir: The base directory of the dataset sources.
    file_pattern: The file pattern to use when matching the datas
      It is assumed that the pattern contains a '%s' string so th
      name can be inserted.
    reader: The TensorFlow reader type.

  Returns:
    A `Dataset` namedtuple.

  Raises:
    ValueError: if `split_name` is not a valid train/validation s
  """
  if split_name not in SPLITS_TO_SIZES:
    raise ValueError('split name %s was not recognized.' % split_

  if not file_pattern:
    file_pattern = _FILE_PATTERN
  file_pattern = os.path.join(dataset_dir, file_pattern % split_n

  # Allowing None in the signature so that dataset_factory can us
  if reader is None:
    reader = tf.TFRecordReader

  keys_to_features = {
      'image/encoded': tf.FixedLenFeature((), tf.string, default_
      'image/format': tf.FixedLenFeature((), tf.string, default_v
      'image/class/label': tf.FixedLenFeature(
          [], tf.int64, default_value=tf.zeros([], dtype=tf.int64
  }

  items_to_handlers = {
      'image': slim.tfexample_decoder.Image(),
      'label': slim.tfexample_decoder.Tensor('image/class/label')
  }

  decoder = slim.tfexample_decoder.TFExampleDecoder(
      keys_to_features, items_to_handlers)

  labels_to_names = None
```

```
  if dataset_utils.has_labels(dataset_dir):
    labels_to_names = dataset_utils.read_label_file(dataset_dir)

  return slim.dataset.Dataset(
      data_sources=file_pattern,
      reader=reader,
      decoder=decoder,
      num_samples=SPLITS_TO_SIZES[split_name],
      items_to_descriptions=_ITEMS_TO_DESCRIPTIONS,
      num_classes=_NUM_CLASSES,
      labels_to_names=labels_to_names)
```

```
  # Copyright 2016 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implie
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
r"""Converts␣labeled␣CVE␣images␣to␣TFRecords␣of␣TF-Example␣protos.


"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import math
import os
import random
import sys

import tensorflow as tf
```

```
from datasets import dataset_utils


# The number of images in the validation set.
_NUM_VALIDATION = 1224

# Seed for repeatability.
_RANDOM_SEED = 0

# The number of shards per dataset split.
_NUM_SHARDS = 2


class ImageReader(object):
  """Helper class that provides TensorFlow image coding utilities

  def __init__(self):
    # Initializes function that decodes RGB JPEG data.
    self._decode_jpeg_data = tf.placeholder(dtype=tf.string)
    self._decode_jpeg = tf.image.decode_jpeg(self._decode_jpeg_da

  def read_image_dims(self, sess, image_data):
    image = self.decode_jpeg(sess, image_data)
    return image.shape[0], image.shape[1]

  def decode_jpeg(self, sess, image_data):
    image = sess.run(self._decode_jpeg,
                     feed_dict={self._decode_jpeg_data: image_dat
    assert len(image.shape) == 3
    assert image.shape[2] == 3
    return image


def _get_filenames_and_classes(dataset_dir):
  """Returns a list of filenames and inferred class names.

  Args:
    dataset_dir: A directory containing a set of subdirectories r
      class names. Each subdirectory should contain PNG or JPG en

  Returns:
    A list of image file paths, relative to `dataset_dir` and the
    subdirectories, representing class names.
  """
  rootdir = os.path.join(dataset_dir, 'cve_diseases')
```

```
  directories = []
  class_names = []
  for filename in os.listdir(rootdir):
    path = os.path.join(rootdir, filename)
    if os.path.isdir(path):
      directories.append(path)
      class_names.append(filename)

  photo_filenames = []
  for directory in directories:
    for filename in os.listdir(directory):
      path = os.path.join(directory, filename)
      photo_filenames.append(path)

  return photo_filenames, sorted(class_names)


def _get_dataset_filename(dataset_dir, split_name, shard_id):
  output_filename = 'cve_diseases_%s_%05d-of-%05d.tfrecord' % (
      split_name, shard_id, _NUM_SHARDS)
  return os.path.join(dataset_dir, output_filename)


def _convert_dataset(split_name, filenames, class_names_to_ids, dataset_
  """Converts the given filenames to a TFRecord dataset.

  Args:
    split_name: The name of the dataset, either 'train' or 'validation'.
    filenames: A list of absolute paths to png or jpg images.
    class_names_to_ids: A dictionary from class names (strings) to ids
      (integers).
    dataset_dir: The directory where the converted datasets are stored.
  """
  assert split_name in ['train', 'validation']

  num_per_shard = int(math.ceil(len(filenames) / float(_NUM_SHARDS)))

  with tf.Graph().as_default():
    image_reader = ImageReader()

    with tf.Session('') as sess:

      for shard_id in range(_NUM_SHARDS):
        output_filename = _get_dataset_filename(
            dataset_dir, split_name, shard_id)
```

```
        with tf.python_io.TFRecordWriter(output_filename) as tfre
          start_ndx = shard_id * num_per_shard
          end_ndx = min((shard_id+1) * num_per_shard, len(filenam
          for i in range(start_ndx, end_ndx):
            sys.stdout.write('\r>> Converting image %d/%d shard %
                i+1, len(filenames), shard_id))
            sys.stdout.flush()

            # Read the filename:
            image_data = tf.gfile.FastGFile(filenames[i], 'rb').r
            height, width = image_reader.read_image_dims(sess, im

            class_name = os.path.basename(os.path.dirname(filenam
            class_id = class_names_to_ids[class_name]

            example = dataset_utils.image_to_tfexample(
                image_data, b'jpg', height, width, class_id)
            tfrecord_writer.write(example.SerializeToString())

  sys.stdout.write('\n')
  sys.stdout.flush()


def _clean_up_temporary_files(dataset_dir):
  """Removes temporary files used to create the dataset.

  Args:
    dataset_dir: The directory where the temporary files are stor
  """
  filename = _DATA_URL.split('/')[-1]
  filepath = os.path.join(dataset_dir, filename)
  tf.gfile.Remove(filepath)

  tmp_dir = os.path.join(dataset_dir, 'cve_diseases')
  tf.gfile.DeleteRecursively(tmp_dir)


def _dataset_exists(dataset_dir):
  for split_name in ['train', 'validation']:
    for shard_id in range(_NUM_SHARDS):
      output_filename = _get_dataset_filename(
          dataset_dir, split_name, shard_id)
      if not tf.gfile.Exists(output_filename):
        return False
```

```
  return True


def run(dataset_dir):
  """Runs the download and conversion operation.

  Args:
    dataset_dir: The dataset directory where the dataset is stored.
  """
  if not tf.gfile.Exists(dataset_dir):
    tf.gfile.MakeDirs(dataset_dir)

  if _dataset_exists(dataset_dir):
    print('Dataset files already exist. Exiting without re-creating ther
    return

#  dataset_utils.download_and_uncompress_tarball(_DATA_URL, dataset_dir)
  photo_filenames, class_names = _get_filenames_and_classes(dataset_dir)
  class_names_to_ids = dict(zip(class_names, range(len(class_names))))

  # Divide into train and test:
  random.seed(_RANDOM_SEED)
  random.shuffle(photo_filenames)
  training_filenames = photo_filenames[_NUM_VALIDATION:]
  validation_filenames = photo_filenames[:_NUM_VALIDATION]

  # First, convert the training and validation sets.
  _convert_dataset('train', training_filenames, class_names_to_ids,
                   dataset_dir)
  _convert_dataset('validation', validation_filenames, class_names_to_id
                   dataset_dir)

  # Finally, write the labels file:
  labels_to_class_names = dict(zip(range(len(class_names)), class_names)
  dataset_utils.write_label_file(labels_to_class_names, dataset_dir)

 # _clean_up_temporary_files(dataset_dir)
  print('\nFinished converting the cve_diseases dataset!' )
```