Svetlana Rovinska

GI Tract Medical Images Classification

Table of Contents

1. Introduction and Problem Description

For the course project the topic of Gastrointestinal (GI) tract medical images classification was selected. It is an interesting topic because in some ways it differs from general image classification problem, whereby medical image classification problem faces additional challenges. First, labeling large medical images dataset poses a challenge, as it requires medical doctors to label each image, which is time consuming and takes their time away from attending to patients. Due to that, large labeled medical images datasets are still quite scarce. Another common challenge is related to training and generalizing Machine Learning or Deep Learning models. Transfer Learning approaches have been popular and successful in many image classifications tasks. However, commonly available Transfer Learning architectures were pre-trained on images of general objects found in the world and therefore learned features that are quite different from the features of medical images. To make the model generalizable, large number of images are required, preferably collected in different hospitals that use different equipment, which can be difficult to obtain. Additionally, the features that model has to learn from medical images in order to detect anomaly can be subtle and may not be as pronounced as those in basic objects of the world, which may require higher image resolution and subsequently more computational resources.

For this project I worked with GI tract images dataset that was collected during endoscopy procedures and labeled by medical doctors. Endoscopy is considered the gold standard in modern medicine for the detection and diagnosis of GI tract diseases. In this report a number of approaches for classifying GI tract anomalies using endoscopic images will be evaluated based on the conducted research and experiments: Logistic Regression classifier, 6-layer CNN built from scratch, three pre-trained and fine-tuned CNN models (VGG-16, InseptionV3, DenseNet-169), and the combination of two pre-trained and fine-tuned models (VGG-16 and DenseNet-169) for feature extraction.

2. Data Sources

https://datasets.simula.no/kvasir/
Download link: Kvasir Dataset v2

The KVASIR dataset contains images from the inside of gastrointestinal tract captured during endoscopy examinations and labelled by medical experts from Vestre Viken Health Trust and the Cancer Registry of Norway.

The dataset is balanced, consisting of 8,000 images classified into 8 classes with 1,000 images in each class. The pathological findings classes are esophagitis, polyps, and ulcerative colitis. “Normal” images are split into three classes according to the anatomical landmark: Z-line, pylorus, and cecum. In addition, two classes of images related to polyps’ removal are included: “dyed and lifted polyp” and “dyed resection margins”. The images are organized in separate folders according to their classification label and vary in resolution from 720 x 576 up to 1920 x 1072 pixels.

3. Literature Review

Several studies have been done on classification and disease detection using images from KVASIR dataset. I started my work on the project by reviewing papers that discuss research and accomplishments in classifying GI tract images using this particular dataset. (2),(4),(5),(16) Afterwards, I reviewed papers focused on other areas of medical images' classification, for example, lungs X-Rays and MRI. (7), (8), (9) After seeing recurring themes of utilizing Deep Learning architectures and Transfer Learning in those studies, I reviewed a few articles discussing those architectures and image classification approaches more generally.(3),(12),(13),(14),(15) Additionally, in my work on the project I found the chapter on computer vision of "Deep Learning with Python" book by Francois Chollet especially useful and applied the techniques presented there in my implementation (1). The approach of processing image files using keras utilities, as well as some of the preprocessing and augmentation techniques, were adopted from Chapter 5 of the book.

The paper "KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection" introduces the dataset that I worked with in detail, providing information on how the data was collected, explains the meaning of each of the eight class labels (which is particularly useful for people who do not have experience in medical field), and presents initial models that were built and can be used as a baseline. The aim of this paper was to introduce the KVASIR dataset and present the initial multi-class detection experiment as a baseline for future experiments. Three different approaches were presented: global features (GF), Deep CNN, and transfer learning (TFL). Best performing approach in their experiment was 6 GF with LMT classifier. However, it states that that CNN parameters were not optimized, and the models were trained over a small number of epochs (2).

In earlier studies, handcrafted features, such as color, texture, and shape, were extracted from endoscopic images and used in classification models to detect anomalies. Handcrafted features extraction had to be performed by domain experts. The main drawbacks of such approaches are that it requires a lot of experience to generate handcrafted features, cannot generalize well between similar problems and cannot produce high performance in real-time. In more recent studies most approaches that resulted state-of-the-art performance rely on Deep CNNs architectures. CNNs are used to generate features from images automatically, without requiring human intervention. In the Esophageal Anomaly Detection study (2019), features extracted using DenseNet architecture were concatenated with features extracted using Gabor filter, and combined features were fed to the classifier. Adding Gabor filter responses to the feature map improved detection from 87% to 90.2% when using DenseNet as CNN backbone (based on recall metric). Additionally, the study found that using DenseNet as CNN backbone resulted better performance compared to using VGG-16 and AlexNet as CNN backbone, which was attributed to the improved flow of information with DenseNet as parameters efficiency was improved by reusing learned features from the previous layers (5).

Öztürk and Özkay (2021) proposed Residual LSTM Layered CNN architecture for classification of GI tract medical images and used KVASIR dataset in their study. They experimented using three different CNN architectures for features extraction (AlexNet, GoogLeNet and ResNet-50). Features from first, middle and last pooling layers of CNN were transmitted separately to three different LSTM layers and then classification was made by combining all LSTM layers. The hypothesis was that high level semantic information in the last layer and the basic image information in the early layers would all be utilized in the classification, which would lead to better performance. The architecture they proposed achieved state-of-the-art accuracy of 98% using ResNet-50 as CNN backbone, and outperformed architectures proposed in previous GI tract image

classification research (4). Previously, 97% accuracy was achieved in another study based on KVASIR dataset, which combined three pretrained architectures as the feature extractors (DenseNet-201, ResNet-18, and VGG-16) (16).

In multiple medical image classification studies Densely Connected Convolutional Networks (DenseNet) were used as backbone CNN for feature extractions or combined with another CNN for combining feature maps before feeding them to classifier (5),(6),(7),(16). The advantage of DenseNet is that each layer in the block receives features from all preceding layers, which means features extracted by early layers are directly used by deep layers throughout the same dense block. As the result of this design, DenseNet scale to layers depth well, allow parameter and computational efficiency, and are less prone to vanishing gradients and overfitting than other Deep Neural Networks architectures (12),(13). In the study by Huang, Gao, et al., (2017) DenseNet performance was benchmarked on four large image datasets and compared to best-in-class ResNet performance. For the experiment hyperparameters settings were optimized for ResNet, but DenseNet still achieved comparable performance while using significantly fewer parameters and less computations (3).

Medical images come in various resolution sizes and selecting input image size should be considered as one of the optimization criteria. A reduced number of inputs or features is desirable as a means of lowering the number of parameters that must be optimized, which in turn diminishes the risk of model overfitting. However, extensive lowering of image resolution eliminates information that is useful for classification (8). Therefore, we must achieve the balance between reducing the image size enough to lower the number of model parameters, and keeping high enough resolution to preserve the features that are important for the classification. In the study "The Effect of Image Resolution on Deep Learning in Radiography" best performance was achieved at image resolutions between 256 × 256 and 448 × 448 pixels. However, it also found that optimal image resolution depends on the diagnosis in question. Certain types of diagnosis with subtler pathological findings benefit from CNN training at relatively higher image resolutions.

Another paper I reviewed focuses on challenges of generalizing GI tract anomaly classification models and empirically shows that good performance measures with a single dataset may not imply good real-world performance (6). It suggests that using the same data source for training and testing split does not generalize well in real world. For example, a model that was optimized using one dataset may not be reliable across different hospitals that use different equipment. In this study 5 models were created – two ML models using Global Features, and three CNN based models; each model was evaluated on 5 different GI tract image sources. While CNN based models performed best on most of the datasets, GF based ML model performed better on one of the datasets. Cross-dataset evaluation metrics are more reliable for building a generalizable model than performance evaluation metrics based on a train/test split from the same data source, however, obtaining such diverse labelled medical image data can be a challenge.

Recently, Deep Learning methods have been almost exclusively implemented in either TensorFlow, Keras, or Pytorch. Although DL methods can be extremely useful for classifying complex datasets, they come with some significant challenges. In addition to requiring large amount of data to optimize and train well generalizable model, they are often not easily interpretable, resulting a difficulty in understanding what is it exactly that makes one model perform better than the other (9).

4. Technologies and Related Issues

I decided to use keras libraries for implementing my Deep Learning models and to utilize GPUs on Google Colab for training the models.

I loaded the image files (2.36 GB) to Google Drive and mounted the drive from Google colab. This setup resulted in severe time delays brining the files into colab environment every time after re-connecting to colab runtime (1 – 1.5 hours). Unfortunately, I could not train DL models on my laptop locally, as it does not have GPU and only has 8 GB RAM. On top of that, I experienced colab dropping connection either due to idle session or running out of RAM. RAM issue was resolved by processing data in batches.

5. Preprocessing

Data loading, normalization and augmentation were performed using keras utilities:

```
keras.preprocessing.image_dataset_from_directory
```

```
keras.layers.experimental.preprocessing
```

Loading data from Google Drive

KVASIR dataset images came organized into 8 folders according to their label, 1000 files in each folder. I had to re-organize them into train/test subfolders while keeping the rest of the folder structure. This was done in Python script using os and shutil libraries. I decided to move 100 files from each class folder into test folder and 900 files to train folder to be used for training and validation split. Within train and test folder, the subfolder structure based on class label was preserved. This step was done once to prepare the file structure for processing by keras utility (keras.preprocessing.image_dataset_from_directory).

For loading image data into colab runtime environment, google drive where the files were stored was mounted and keras image_dataset_from_directory utility was used to process the images in batches (normalize and augment data, and then train and test models). In my experience, I felt there were both pros and cons in processing data in this manner. The advantages were that it automatically picks up labels from folder names, enables automatic batch processing, and allows to specify training/validation split when creating the dataset iterator. However, I felt this setup limited flexibility in manipulating the data and controlling splits. It was difficult to find clear documentation on how to manipulate data points while processing data through this batch iterator. Additionally, I was unable to run Optuna using this datatype (tf.data.Dataset) in model training and had to resort setting up my own semi-manual hyper-parameter tunning process.

Input size

Images in the dataset came with the different resolution from 720 x 576 up to 1920 x 1072 pixels. For several reasons I decided to downsample the images into 224 x 224 pixels. First, pre-trained models that I was using were trained on 224 x 224 size images from ImageNet. Also, in one of the studies on KVASIR dataset (16) 224 x 224 resolution was used with a remarkable performance results. Additionally, conclusions from the study "The Effect of Image Resolution on Deep Learning in Radiography" were considered (8).

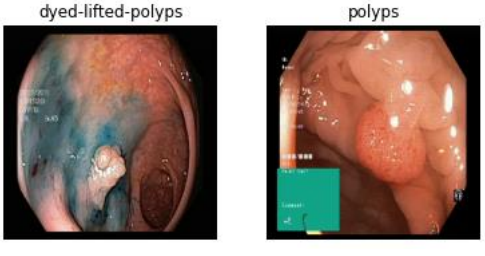
Normalization and Augmentation

Pixel values of the images were normalized (divided by maximum pixel value of 255).

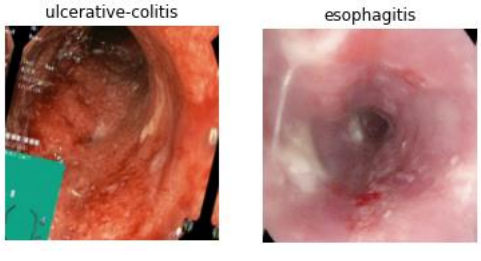
The following augmentation techniques were applied: CenterCrop, RandomRotation and RandomZoom.

I decided to center crop the images slightly to remove the black boarder, as it did not provide any useful features. Other ideas for augmentation were picked up from "An Extensive Study on Cross-Dataset Bias and Evaluation Metrics Interpretation for Machine Learning Applied to Gastrointestinal Tract Abnormality Classification" (6).

Examples of original images:



Examples of augmented images:



Experiments in (6) showed that removing the whole green box (bottom left corner on two of the images above) by applying a larger crop size is not advisable, because for some images too much content of the finding is lost with a larger crop size.

6. Analysis

Data Analysis

To illustrate the complexity of KVASIR dataset, data was visualized using TSNE plot and compared to the visualization of images of handwritten digits in MNIST dataset (Figure 1). It shows that it is much harder to find differences between images in KVASIR dataset than between images in MNIST dataset. Based on this, the intuition was that simple ML model will not be able to accurately classify images in KVASIR dataset. To validate the intuition, KVASIR dataset images were flattened and fed to Logistic Regression model. Using 224 x 224 image size directly in Logistic Regression model was not possible, because the algorithm would not converge after a large number of iterations. The dimensions were reduced to 28 x 28. This experiment resulted 50.5% accuracy on test data. The conclusion was that Deep Learning models may do a better job in extracting subtle features required to detect anomalies in GI tract images.

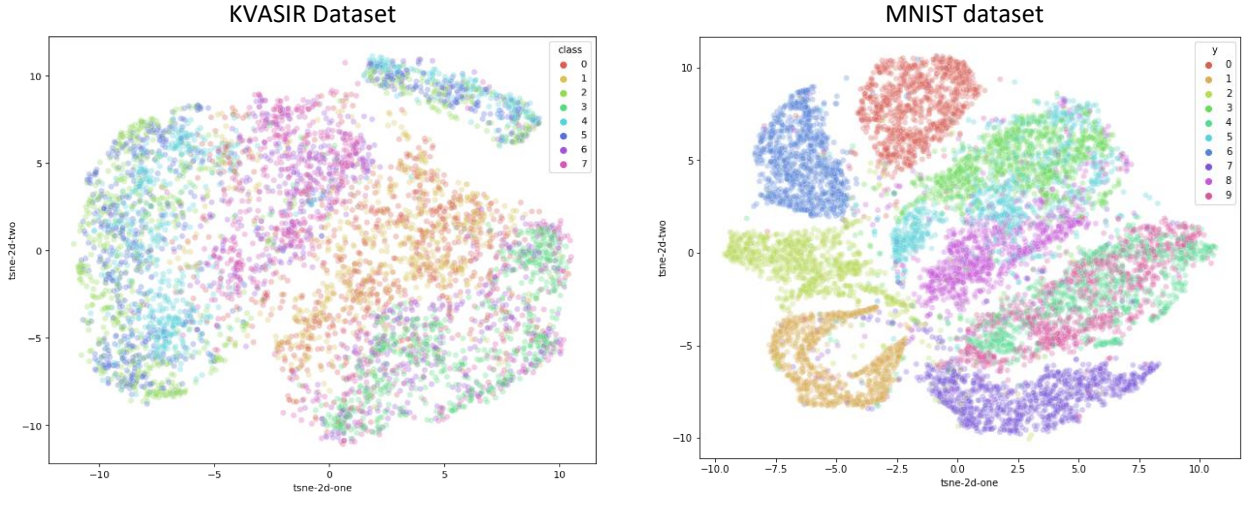


Figure 1

Deep Learning Models

First, I experimented with 3 pre-trained models: VGG-16, InceptionV3 and DenseNet-169. In initial experiment pre-trained models were used to extract features without fine-tuning. Since all the pre-trained models were trained on ImageNet, which contains images quite different from GI tract images, the models did not extract features applicable to GI images which did not result a good performance. Fine tuning all layers of pre-trained models resulted better classification performance.

In the first four models listed in Table 1, two fully connected layers were used for classification. First fully connected layer was setup with 256 hidden units and ReLU activation function, and second fully connected layer was setup with 8 hidden units and softmax activation function. All hyperparameters were tuned for the first model (CNN from scratch). After analysing performance of the first four models, it was decided to combine two best performing models for feature extraction, which were DenseNet-169 and VGG-16. Combining DenseNet-169 and VGG-16 features further improved performance, and 93% accuracy on test data was achieved using Logistic Regression classifier for combined features.

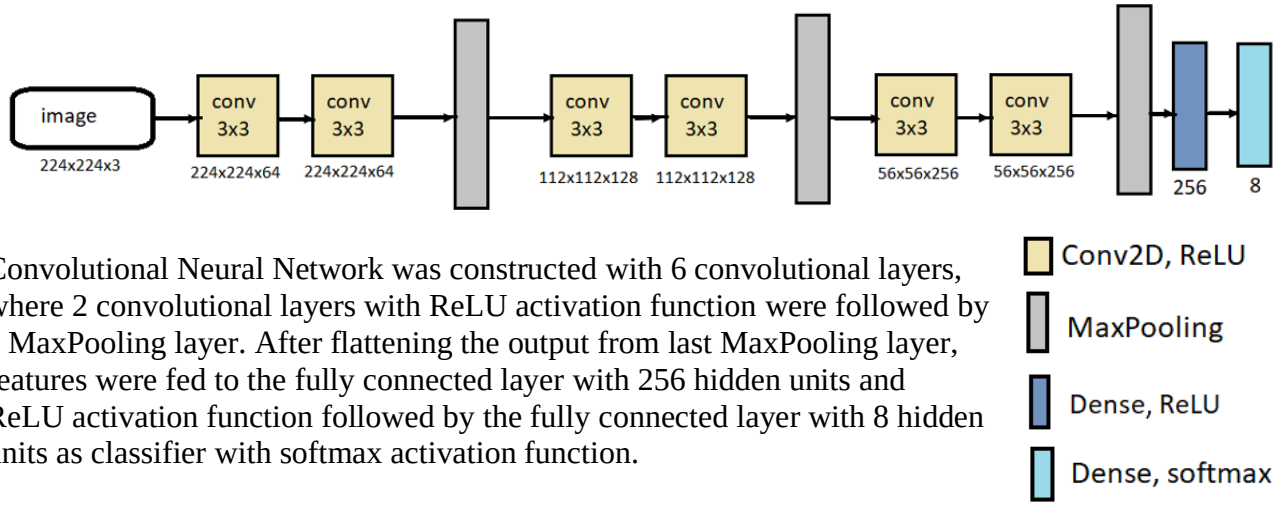
Table 1: Models performance on KVASIR test data

Model	Accuracy	Precision	Recall
CNN from scratch (6 conv layers)	74.5	75	74.5
VGG-16	89.88	90.5	89.88
DenseNet-169	90.63	90.75	90.63
InceptionV3	85.88	85.75	85.38
VGG-16 and DenseNet-169 combined features, LR classifier	93	93	92.88

Recall = TP / (TP + FN)
Precision = TP / (TP + FP)
Although the dataset is balanced and all three metrics (accuracy, precision, and recall) are almost the same, recall is slightly lower than precision with all five models.
In medical Computer Aided Detection (CAD) applications a False Negative (FN) may be considered a worse error than a False Positive (FP), since they are supposed to catch and flag the anomalies that may have been overlooked by human error. Therefore, we want to maximize Recall metric by minimizing False Negatives. The smallest gap between Precision and Recall metrics was observed with DenseNet-169 architecture and with combined VGG-16 and DenseNet-16 architecture (0.12% difference in both cases). It can be concluded that those architectures provide balanced accuracy results and do not result significantly higher False Negatives than False Positives. However, for this dataset confusion matrices discussed in the later section of this report can provide better insights into misclassification errors than precision and recall metrics. Confusion matrices revealed that most "False Negatives" are the false negatives on the "Normal" true class instance, meaning models label true normal image as an anomaly. This misclassification will be counted as FN, even though in medical diagnosis sense it would be considered false positive (falsely classified normal image as a disease).

CNN model from scratch

Figure 2



Convolutional Neural Network was constructed with 6 convolutional layers, where 2 convolutional layers with ReLU activation function were followed by a MaxPooling layer. After flattening the output from last MaxPooling layer, features were fed to the fully connected layer with 256 hidden units and ReLU activation function followed by the fully connected layer with 8 hidden units as classifier with softmax activation function.

Table 2 shows hyperparameters that were experimented with and tuned for the CNN model. All hyperparameters tuning was done for CNN model depicted in Figure 2, and the same hyperparameters that performed best for CNN model were used to train VGG-16, DenseNet-169 and InceptionV3 models on KVASIR dataset. All models were compiled using categorical_crossentropy loss function and trained over 20 epochs.

Some of the hyperparameters chosen for optimization were based on the values mentioned in the GI tract images classification literature. For example, some of the papers mentioned which optimizer, learning rate and dropout generated optimal results in their study. The optimizer and learning rate that worked best for my model was used in VGG-16 transfer learning example in Chollet book (Chapter 5) (1).

Table 2: Hyperparameters tuning.

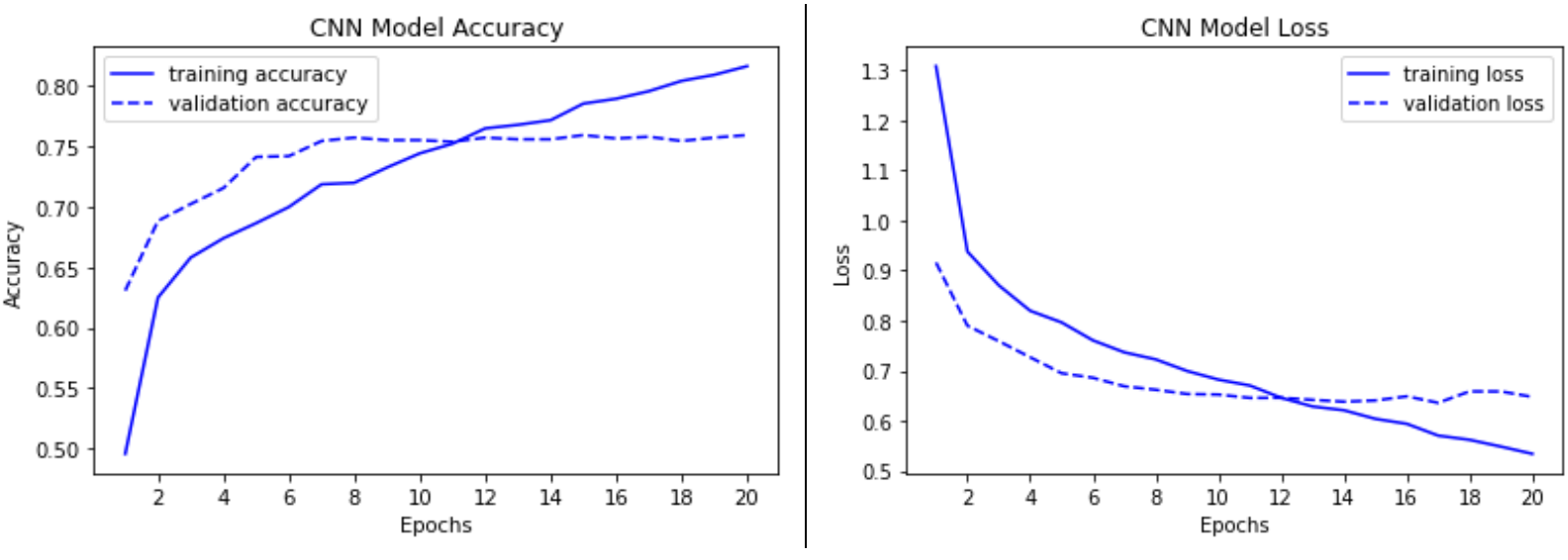
Hyperparameters	Options Evaluated	Best performing
Optimizer/ Learning Rates/ Momentum	RMSprop(lr=2e-5) RMSprop(lr=2e-4) SGD(lr=0.001, momentum=0.8) SGD(lr=0.001, momentum=0.9) SGD(lr=0.01, momentum=0.8) SGD(lr=0.01, momentum=0.9) SGD(lr=0.0001, momentum=0.8) SGD(lr=0.0001, momentum=0.9) Adam()	RMSprop(lr=2e-5)
Regularizer	l2(1e-4) l1(1e-5) l2(1e-5) l2(1e-3)	l2(1e-4)
Batch size	16, 32, 64	32
Number of Hidden Units in first Dense Layer	128, 256, 512, 300, 400	256
Dropout	0.2, 0.35, 0.4, 0.5	0.4

Hyperparameters tuning was done using 80/20 training/validation split of 7,200 images (900 images from each class).

After choosing hyperparameters, I ran 2-fold cross validation using the same 7,200 images from training dataset. Validation accuracy was slightly higher with 2-fold cross validation, but the final accuracy on test dataset did not change.

Figure 3 below shows training and validation accuracy and loss curves when the model was trained using “best performing” hyperparameters from Table 2.

Figure 3



VGG-16, DenseNet-169, and InceptionV3 Models

Figure 4 below shows a snippet of code for creating VGG-16 model (imported from `keras.applications.vgg16)`. InceptionV3 and DenseNet-169 models were created in the same manner.

Figure 4: VGG-16 model setup

```
# load model without classifier layers
model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
# add new classifier layers
flat1 = Flatten()(model.layers[-1].output)
drop1 = layers.Dropout(0.4)(flat1)
class1 = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(1e-4))(drop1)
drop2 = layers.Dropout(0.4)(class1)
output = Dense(8, activation='softmax', kernel_regularizer=regularizers.l2(1e-4))(drop2)
# define new model
model = Model(inputs=model.inputs, outputs=output)
```

Figure 5 Figure 5: VGG-16, InceptionV3, and DenseNet-169 models with Dense Layer classifiersvisualizes three architectures.

Figure 6 shows training and validation accuracy and loss curves for these three architectures when the models were trained using 80/20 training/validation split and “best performing” hyperparameters from Table 2.

Table 3 shows best validation accuracies achieved with each architecture, as well as the number of trainable parameters. Performance metrics on test data were provided earlier in Table 1.

Figure 5: VGG-16, InceptionV3, and DenseNet-169 models with Dense Layer classifiers

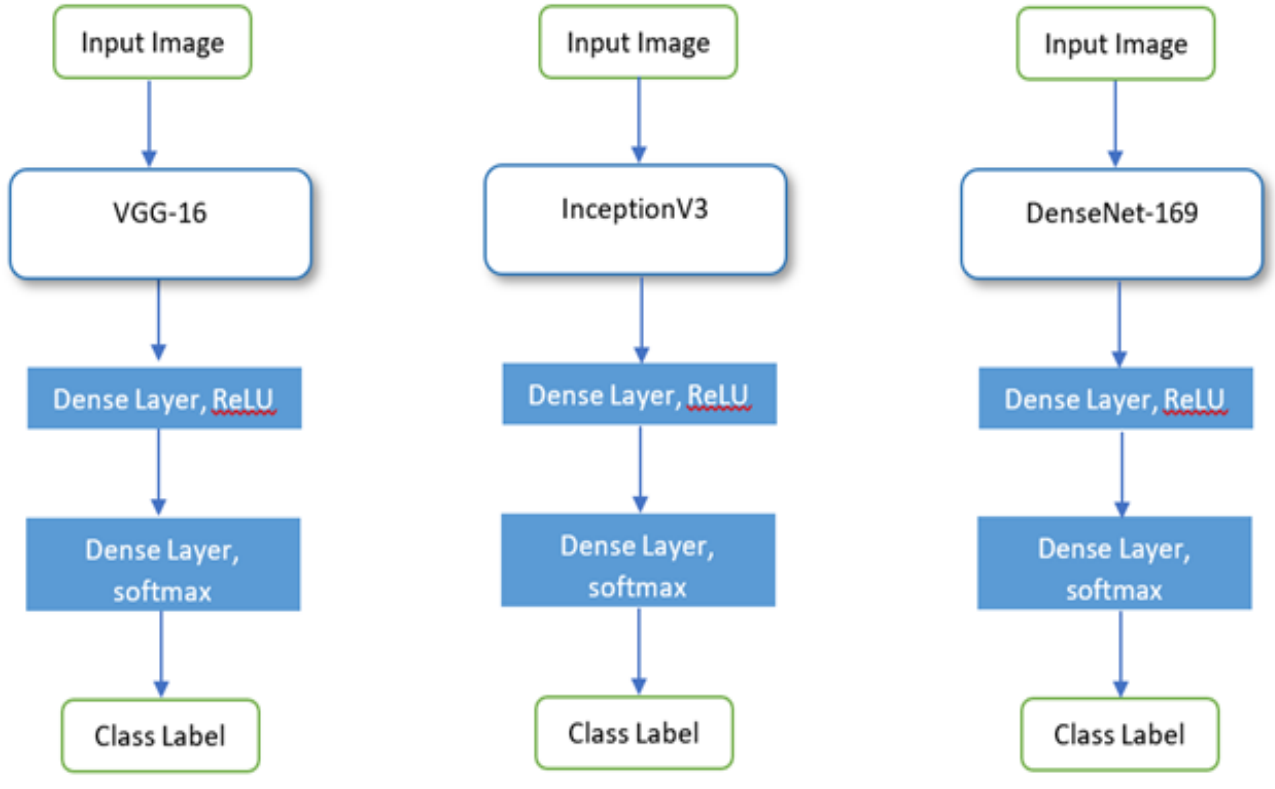
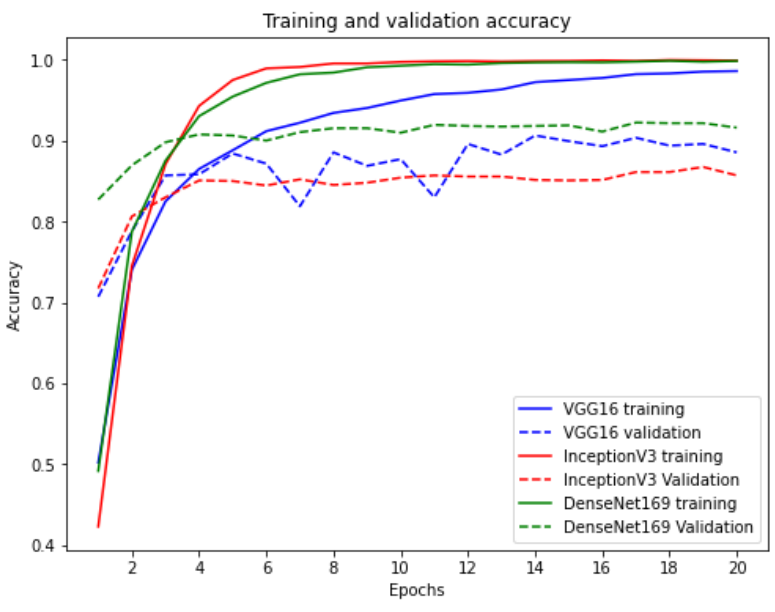
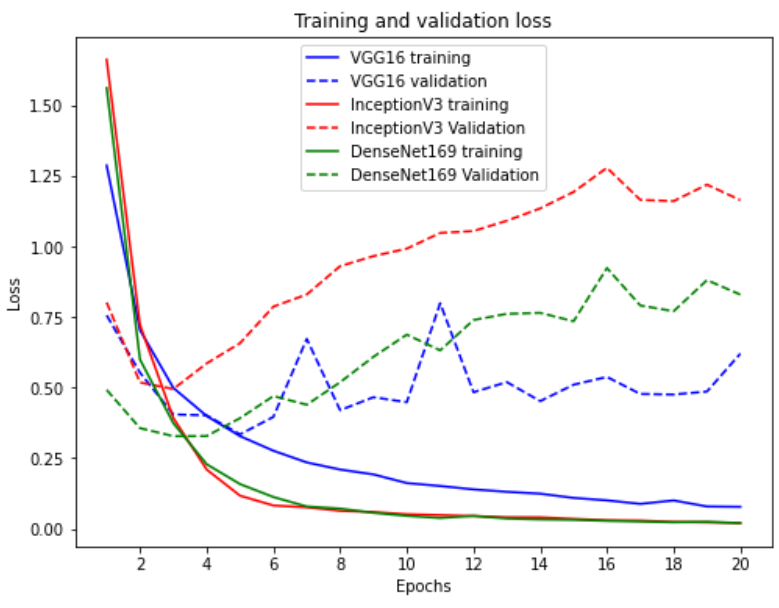


Table 3

Model	Best Validation Accuracy %	Trainable Parameters
VGG-16	91.39	21,139,528
DenseNet-169	92.15	33,360,008
InceptionV3	88.06	34,877,864

Figure 6





Combined VGG-16 and DenseNet-169 architecture

After analysing performance of three pretrained models, it was decided to use two best performing models to extract image features and combine them for classification.

Steps to create and test combined VGG-16 and DenseNet-169 architecture:

1. Using VGG-16 model described in the previous section that was trained on KVASIR dataset images, create new model without the last classifier layer:

```
vgg_features_model = Model (model_vgg.layers[0].input,  model_vgg.layers[21].output)
```

Last layer of vgg_features_model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 256)               6422784
```

Full summary of vgg_features_model is included in the Appendix.

2. Extract VGG-16 features:

```
vgg_features = vgg_features_model.predict(train_ds_norm)
vgg_features.shape: (5760, 256)
```

3. Using DenseNet-169 model described in the previous section that was trained on KVASIR dataset images, create new model without the last classifier layer:

```
dn_features_model = Model (model_dn.layers[0].input,  model_dn.layers[597].output)
```

Last layer of dn_features_model and number of parameters:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 256)               20873472  
=================================================================
Total params: 33,516,352
Trainable params: 33,357,952
Non-trainable params: 158,400
_________________________________________________________________
```

Full summary of dn_features_model is omitted from the report, because it is exceedingly long and spans several pages.

4. Extract DenseNet-169 features:

```
dn_features = dn_features_model.predict(train_ds_norm)
dn_features.shape: (5760, 256)
```

5. Combine features:

```
combined_features = np.concatenate ((dn_features, vgg_features), axis=1)
combined_features.shape: (5760, 512)
```

Combined features were plotted using TSNE algorithm (Figure 7), and it showed that instances can be nicely separated into clusters according to classes. Seeing this plot gave reassurance that a simple classifier such as Logistic Regression may be able to handle these features well.

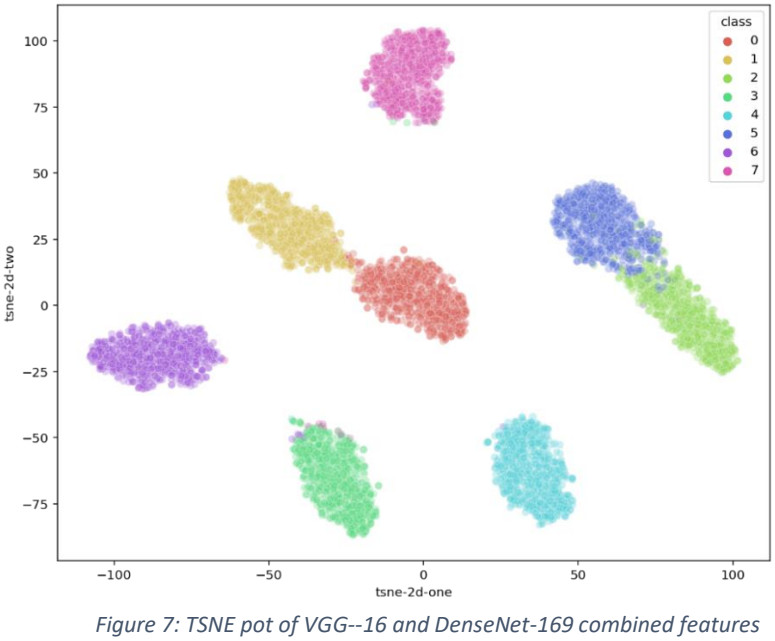


Figure 7: TSNE pot of VGG--16 and DenseNet-169 combined features

6. Create and train Logistic Regression classifier. Other classifiers that I experimented with were KNN and a single hidden layer ANN classifier. KNN did not perform well. ANN classifier was close in accuracy to Logistic Regression, however Logistic Regression was chosen because it is a simpler model and resulted slightly better classification performance on test data.

```
lr = LogisticRegression(max_iter=10000)
lr.fit(combined_features, y_train)
```

7. Extract and combine VGG-16 and DenseNet-169 features using images from test dataset (repeat steps 2, 4, and 5 using test data).

8. Classify test dataset features (obtained in step 7) using logistic regression model trained in step 6.

The process is summarized in the figure below:

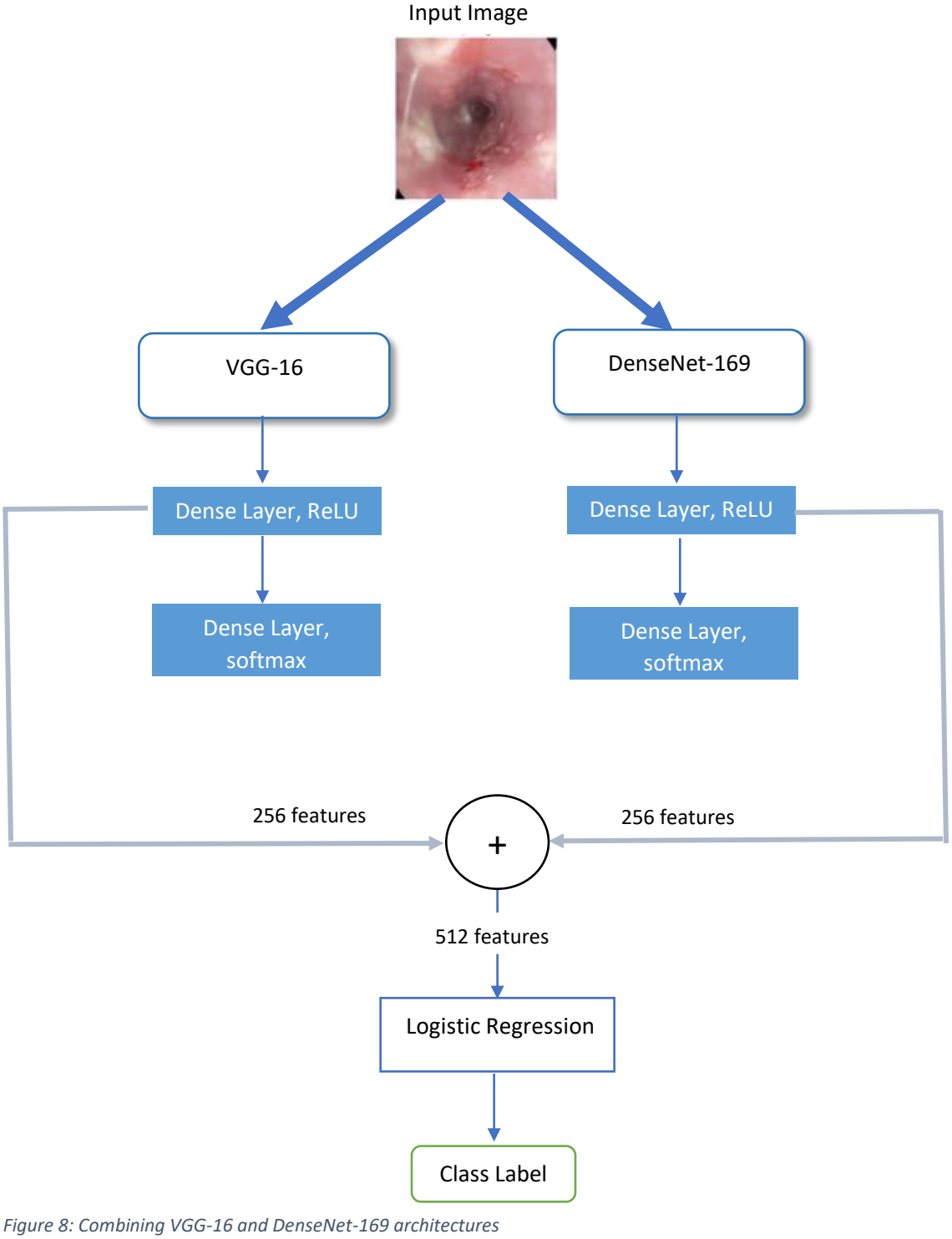


Figure 8: Combining VGG-16 and DenseNet-169 architectures

Confusion Matrices comparison and analysis

Confusion matrixes in Figure 9 were generated using sklearn Python library and visualized using seaborn library based on KVASIR test dataset containing 100 images of each class. In confusion matrices illustrated below, i-th row and j-th column entry indicates the number of samples with true label being i-th class and predicted label being j-th class (17).

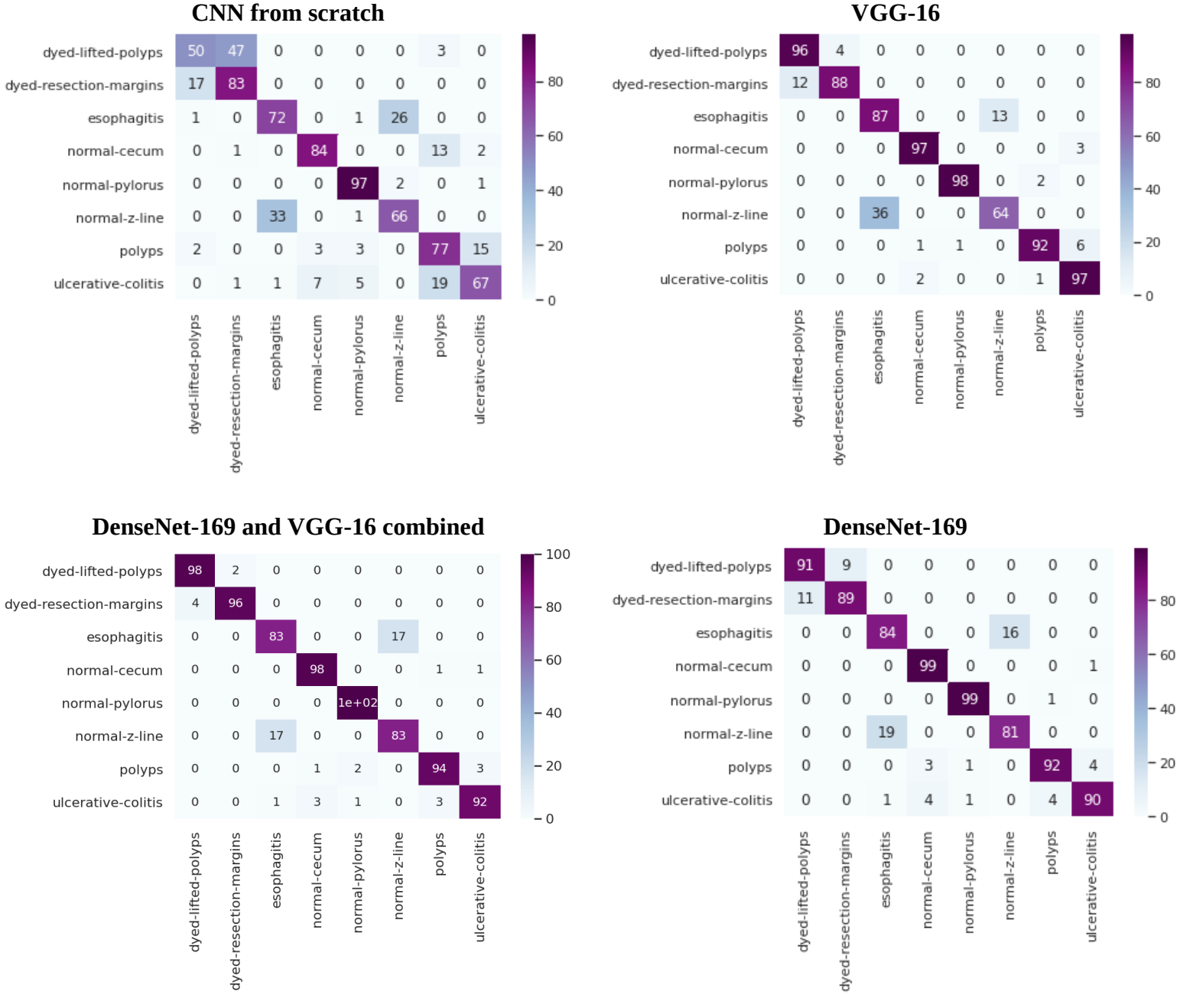


Figure 9: Confusion Matrices for Test data classification results

By examining confusion matrices, it can be observed that across all four models most errors occurred due to difficulty distinguishing between two classes: esophagitis and normal-z-line. Normal-z-line class is a normal image captured at the anatomical landmark of GI tract where esophagitis markers can be present. This may indicate that esophagitis disease images have subtle features that most models had difficulty distinguishing from features of normal anatomical landmark where the anomaly may be present. To further improve classification accuracy, we could focus on images from these two classes. Some further augmentation and filtering techniques can be considered, for example, combining features extracted using CNN architectures with Gabor filter features (5).

Additionally, it should be mentioned that all the models, except for the combined VGG-16 and DenseNet-169 architectures, misclassified "normal-z-line" class as "esophagitis" more frequently than the other way around. This misclassification is reported as False Negative using Python sklearn library functions. However, if we consider the semantics of the classification, labeling true "normal-z-line" image as "esophagitis" is False Positive in a medical diagnosis sense.

It can also be observed that CNN model built from scratch made many misclassification errors related to two polyp removal classes (dyed-lifted-polyps and dyed-resection-margins). However, it maybe more worthwhile to focus on improving stronger models and discard the weakest models.

Combined VGG-16 and DenseNet-169 model shows good performance. If the errors around esophagitis and normal-z-line classes were reduced, it would be quite reliable. In computer aided diagnosis it may be better to reduce multi-class classification problem to a set of binary classification problems. For example, have three binary classification problems focusing on detecting three diseases separately (esophagitis, polyps, ulcerative colitis). This reasoning occurred after seeing that in a few cases our strongest model misclassified one disease as another disease. For example, in three instances polyps were wrongly labeled as ulcerative-colitis, and in three other instances ulcerative-colitis was labeled as polyps.

Comparison with existing work

We can compare approaches and results to some of the existing work in medical images classification. Common approach is to rely on one of the existing architectures such as (ResNet, DenseNet, VGG-16 or other) and either combine multiple existing architectures in some way or use them for feature extraction that would feed into other models. Some studies on GI tract images classification were able to achieve impressive performance of over 98% accuracy. For example, Öztürk and Özkaya (4) achieved 98.05% accuracy classifying KVASIR images using ResNet50 as CNN backbone and adding Residual LSTM model. However, LSTM models can be computationally expensive. Before developing my own models, I attempted to reproduce the Residual LSTM model presented in the study, but was faced with resources limitations that impeded successful implementation.

C. Gamage, et al., (16) achieved 97.38% accuracy on KVASIR dataset by combining features from ResNet-18, VGG-16 and DenseNet-201. They reduced images size to 224 x 224 pixels, the same size as I chose in this project. DenseNet-201 extracted 1920 features, ResNet-18 extracted 512 features, and VGG-16 extracted 512 features, for the total of 2944 features for each image. Single hidden layer ANN was used for classification. Similarly, in this project I combined features but only from two of

these models by extracting 256 features from DenseNet-169 and 256 features from VGG-16. I found that single hidden layer ANN classifier did not result better performance than Logistic Regression classifier. Other experiment that I could have done to see if performance of our combined model can be improved, would be to increase number of features extracted from each model to 512 and/or add features from a third model. However, due to time constraint this could not be done, as even just loading images dataset from google drive to colab takes excessive amount of time, besides the time required to train and tune the model.

In another study, Thambawita, et al., (6) used a different approach for combining existing architectures. Instead of concatenating features from multiple CNN architectures, they fed two probability vectors output by the two separate networks (ResNet-152 and DenseNet-161) into Multi-Layer Perceptron (MLP). Probabilities vectors output by the two base networks are classification results in the form of the values between 0 and 1 indicating how likely each class is. First, they re-trained ResNet-152 and DenseNet-161 networks on Medico dataset (which is another GI tract images dataset). Then, only MLP was trained with the pre-trained ResNet-152 and DenseNet-161 for the Medico dataset to decide the final prediction based on the probabilities that come from two networks. In Figure 10 (copied from their paper) it is suggested how to expand their architectures to more than two base networks.

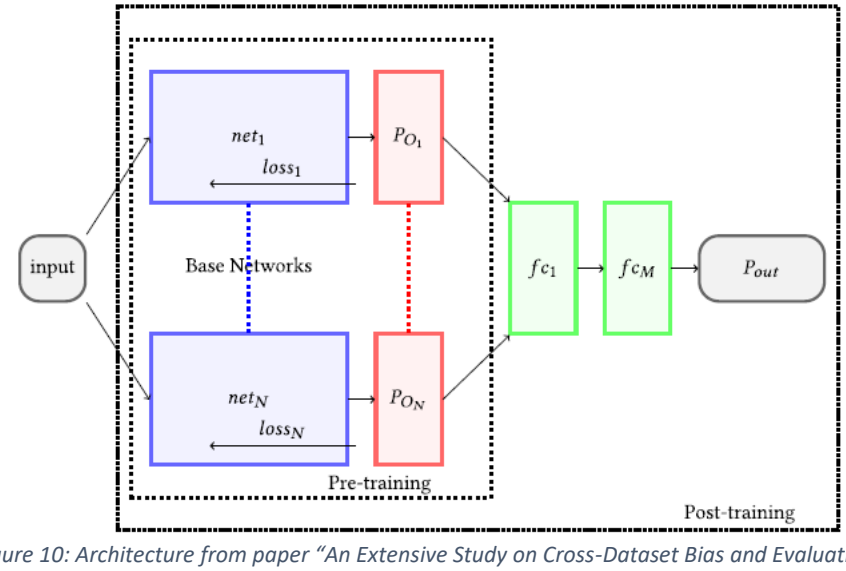


Figure 10: Architecture from paper "An Extensive Study on Cross-Dataset Bias and Evaluation Metrics Interpretation for Machine Learning Applied to Gastrointestinal Tract Abnormality Classification."

Medico dataset used in the study (6) is a multi-class GI tract images dataset with 16 classes, but is unbalanced with different number of images belonging to each class. Therefore, instead of accuracy we can compare precision and recall metrics and consider confusion matrix comparison. Both precision and recall in study (6) on the model described above were 94.58%. This approach (labelled as "method 5" in the paper) won second place in 2018 Medico Classification Task challenge. They also evaluated four other models on Medico dataset, which resulted lower performance, ranging from 85% to 94% in precision and recall metrics. For comparison, combined DenseNet-169 and VGG-16 architecture implemented in this course project achieved 93% accuracy, 93% precision, and 92.88% recall on a similar GI tract images classification task.

Certain similarities can be observed between confusion matrices (CM) resulted from combined DenseNet-169/VGG-16 architecture implemented in this course project and Thambawita, et al., (6) method 5. From Thambawita, et al., paper: "According to the CM, we can identify two main bottlenecks to improve the performance of method 5. The first one is **misclassification between esophagitis and normal-z-line**, and the second one is misclassification between dyed-lifted-polyps and dyed-resection-margins." [These were exactly the week points of the models evaluated in this course project.] "Therefore, images from these classes were manually examined to identify the reasons for the misclassifications. For the conflict between esophagitis and normal-z-line, the reason is the very close locations of these two landmarks in the GI tract. However, the confusion between dyed-lifted-polyps and dyed-restrictions is caused because of the same color patterns and the same texture structures of both types of images. With these limitations, method 5 showed the best performance with an MCC of 0.9421, which was the important measurement to win the 2018 Medico Task. Based on the MCC value, we won second place in the 2018 Medico Task."

Table 5. CM of Method 5 (Our Best Model) Based on the Medico Test Dataset

		Actual Class															
		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Predicted Class	Ulcerative-colitis (A)	500	—	—	—	–	—	—	—	39	—	3	—	1	1	—	7
	Esophagitis (B)	3	432	48	—	—	—	—	—	—	—	—	—	—	—	—	—
	Normal-z-line (C)	1	121	513	—	—	—	—	—	—	—	—	—	1	—	—	—
	Dyed-lifted-polyps (D)	1	—	—	522	31	—	—	—	—	—	2	—	—	—	—	34
	Dyed-resection-margins (E)	—	—	—	33	532	—	—	—	—	—	1	—	—	—	—	17
	Out-of-patient (F)	—	—	—	—	1	5	—	—	—	—	—	—	—	—	—	—
	Normal-pylorus (G)	3	3	2	—	—	—	559	—	—	—	2	—	—	—	—	—
	Stool-inclusions (H)	—	—	—	—	—	—	—	501	7	—	—	—	—	—	—	—
	Stool-plenty (I)	1	—	—	—	—	—	—	—	1,918	—	—	—	—	—	—	1
	Blurry-nothing (J)	1	—	—	—	—	—	—	—	1	37	—	—	—	—	—	—
	Polyps (K)	10	—	—	1	—	—	1	—	—	—	358	6	—	1	—	46
	Normal-cecum (L)	18	—	—	—	—	—	—	—	—	—	6	578	—	—	—	2
	Colon-clear (M)	1	—	—	—	—	—	—	5	—	—	—	—	1,063	—	1	—
	Retroflex-rectum (N)	3	—	—	—	—	—	—	—	—	—	2	—	—	188	1	—
	Retroflex-stomach (O)	—	—	—	—	—	—	1	—	—	—	—	—	—	2	395	1
	Instruments (P)	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	165

The diagonal value represents true predictions (number of images) of the model. A through P are the classes corresponding to the class names in the first column. The most confusion can be observed between classes B and C, and classes D and E. Looking at the images, we can see that they are quite similar in their visual features (colors, texture, etc.).

Figure 11: Confusion Matrix from "An Extensive Study on Cross-Dataset Bias and Evaluation Metrics Interpretation for Machine Learning Applied to Gastrointestinal Tract Abnormality Classification."

This is consistent with the "bottlenecks" in our models as can be observed from confusion matrices presented in earlier section. The same trend was observed in confusion matrix from the architecture that won first place in 2018 Medico classification task (Hoang et al., 19), where normal-z-line and esophagitis were the top most confusing classes. To reduce the probability of misclassifying esophagitis as normal-z-line (and missing a disease diagnosis), they proposed adding a condition for the two classes. Whenever the normal-z-line class appeared to have the highest probability, they added a small bias to the probability of the esophagitis, making the model prefer esophagitis to normal-z-line when it was confused between these classes.

7. Lessons Learned

Four approaches to classifying images in the KVASIR dataset were experimented with in the scope of the project:

1. Using image features directly with Logistic Regression Model (50.5% accuracy).

2. Classifying images using CNN model built from scratch (74.5% accuracy).

3. Training one of the existing models on KVASIR dataset (86-91% accuracy on VGG-16, DenseNet-169, and InceptionV3 models used separately).

4. Combining features extracted from VGG-16 and DenseNet-169 for classification using Logistic Regression classifier (93% accuracy).

Fourth approach resulted significantly higher accuracy (as well as precision and recall) than other approaches that were considered. Furthermore, upon evaluation of confusion matrices it can be concluded that fourth approach leads to more reliable results than the other approaches that were experimented with in the scope of the project. The classification results presented in the confusion matrix are consistent with those presented in multiple papers on GI tract images classification problem.

Based on experimental results and review of previous work on GI tract images classification we can make the following conclusions:

> For complex image classification tasks existing available CNN models are useful (e.g. VGG, DenseNet, ResNet, Inception, etc.). In literature, models that achieved state-of-the-art performance for medical images classification used existing DL architectures for features extraction and built on top.
>
> It can be very challenging to achieve good classification results with a CNN model built from scratch. There are almost infinite number of combinations of the layers we can use in the model, before even considering other hyperparameter tuning. It would require deep domain knowledge to construct effective model from scratch. Throwing a bunch of layers together may not result a good performance even with thorough tuning. It seems we need to figure out the science and math of it to build a well performing CNN model from scratch. However, the nature of Deep Learning makes it hard to understand what it is exactly that makes one model perform better than the other, which complicates the task even more.
>
> We cannot achieve good classification performance by feeding complex image features directly into a simple Machine Learning model. It would require a lot of effort from experts to manually enhance image features in order to achieve performance comparable to what can be achieved using existing CNN architectures without requiring hand-crafted features. Existing CNN architectures can effectively extract most important features directly from images without applying manual filtering.

In GI tract images classification (and presumably in medical images classification in general) some diseases can be more difficult to classify than others. More subtle features separate them from normal class images, which makes them more prone to misdiagnosis. For example, this was observed in our confusion matrix as well as in the literature in the case of the esophagitis, where it was more frequently confused with one of the normal anatomical landmarks than other classes. Choosing higher image resolution or adding manual filters in those cases may improve the detection accuracy. Also, as found in one study, manually adding a small bias to the probability value of the disease class may reduce false negatives.

Finally, I would like to mention a lesson learned from working on this project related to technical setup. The setup I used in this project was storing the dataset on google drive, and then mounting google drive from colab to load the images. This setup was extremely inefficient and time consuming, as every time I connected to colab runtime it took 1 – 1.5 hours to bring the image data into colab environment. That would not be that bad if I could stay connected all day, but colab could drop connection sporadically for various reasons (idle session, out of memory, etc.), at which point all data would be gone from colab runtime environment and had to be loaded from google drive again. Based on my experience, I would not recommend using colab runtime for working with large high dimensional datasets. In retrospect, I should have saved resized images (224 x 224 size that was used to train the models) back to google drive and bring them in every time instead of original images, but the idea occurred to me too late, and I am not sure if it would have helped.

8. References

(1) Francois Chollet, Deep Learning with Python. Manning Publications, 2017.

(2) Pogorelov, Konstantin & Randel, Kristin & Griwodz, Carsten & de Lange, Thomas & Eskeland, Sigrun & Johansen, Dag & Spampinato, Concetto & Dang Nguyen, Duc Tien & Lux, Mathias & Schmidt, Peter & Riegler, Michael & Halvorsen, Pål. (2017). KVASIR: A Multi-Class Image Dataset for Computer Aided Gastrointestinal Disease Detection. 10.1145/3083187.3083212.

(3) Huang, Gao, et al. "Densely Connected Convolutional Networks." *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 2261–2269, doi:10.1109/CVPR.2017.243.

(4) Öztürk, Şaban, and Umut Özkaya. "Residual LSTM Layered CNN for Classification of Gastrointestinal Tract Diseases." *Journal of Biomedical Informatics*, vol. 113, no. Complete, Jan. 2021, p., doi:10.1016/j.jbi.2020.103638.

(5) Ghatwary, Noha, et al. "Esophageal Abnormality Detection Using DenseNet Based Faster R-CNN With Gabor Features." *IEEE Access*, vol. 7, July 2019, pp. 84374–84385, doi:10.1109/ACCESS.2019.2925585.

(6) *Vajira Thambawita, Debesh Jha, Hugo Lewi Hammer, Håvard D. Johansen, Dag Johansen, Pål Halvorsen, and Michael A. Riegler. 2020. An Extensive Study on Cross-Dataset Bias and Evaluation Metrics Interpretation for Machine Learning Applied to Gastrointestinal Tract Abnormality Classification. ACM Trans. Comput. Healthcare 1, 3, Article 17 (July 2020), 29 pages. DOI:https://doi.org/10.1145/3386295*

(7) Hashmi, Mohammad Farukh et al. "Efficient Pneumonia Detection in Chest Xray Images Using Deep Transfer Learning." *Diagnostics (Basel, Switzerland)* vol. 10,6 417. 19 Jun. 2020, doi:10.3390/diagnostics10060417 **https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7345724/**

(8) Sabottke, Carl F., et al. “The Effect of Image Resolution on Deep Learning in Radiography.” *Radiology: Artificial Intelligence*, 22 Jan. 2020, pubs.rsna.org/doi/10.1148/ryai.2019190015. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7345724/

(9) An overview of deep learning in medical imaging focusing on MRI: https://www.sciencedirect.com/science/article/pii/S0939388918301181

(10) Through The Eyes of Gabor Filter: https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

(11) Faster RCNN Object detection: https://towardsdatascience.com/faster-rcnn-object-detection-f865e5ed7fc4

(12) Review: DenseNet – Dense Convolutional Network (Image Classificaiton): https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

(13) Exploring DenseNets from paper to keras: https://towardsdatascience.com/exploring-densenets-from-paper-to-keras-dcc01725488b

(14) Understanding high dimensional spaces in machine learning: https://towardsdatascience.com/understanding-high-dimensional-spaces-in-machine-learning-4c5c38930b6a

(15) VGG16 – Convolutional Network for Classification and Detection: https://neurohive.io/en/popular-networks/vgg16/

(16) C. Gamage, I. Wijesinghe, C. Chitraranjan and I. Perera, "GI-Net: Anomalies Classification in Gastrointestinal Tract through Endoscopic Imagery with Deep Learning," 2019 Moratuwa Engineering Research Conference (MERCon), Moratuwa, Sri Lanka, 2019, pp. 66-71, doi: 10.1109/MERCon.2019.8818929.

(17) https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

(18) A comprehensive analysis of classification methods in gastrointestinal endoscopy imaging: https://www.sciencedirect.com/science/article/pii/S1361841521000530

(19) T.-H. Hoang, H.-D. Nguyen, T.-A. Nguyen, “An application of residual network and faster - RCNN for medico: multimedia task at mediaeval 2018”, Proc. CEUR Worksh. Multim. Bench. Worksh. (MediaEval) (2018) http://ceur-ws.org/Vol-2283/MediaEval_18_paper_11.pdf

Appendix

```
vgg_features_model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout_11 (Dropout)         (None, 25088)             0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               6422784
=================================================================
Total params: 21,137,472
Trainable params: 21,137,472
Non-trainable params: 0
```