
Classification of Anomalies in Gastrointestinal Tract through Endoscopic Imagery with Deep Learning



Introduction

According to research conducted by the World Health Organization (WHO), each year 1.8 million deaths have been reported due to diseases related to gastrointestinal tract. As usual as many other diseases, main reason for such a large number of deaths is Insufficient knowledge about disease among the public as well as insufficient diagnosis methods in the medical field. In this paper we have discussed how classification of Image through the endoscope help to solve this problem.

Due to its powerful feature extraction and classification functions, convolution neural networks (CNN) have been widely used in various image processing applications including wireless capsule endoscopy (WCE). CNN is the most popular and powerful type of deep neural network . It is especially useful when looking for patterns in images to identify objects, faces and scenes. They have the ability to learn directly from image data sets, so that they can detect patterns used to classify images, thereby eliminating the need to manually extract features.

Gastrointestinal (GI) endoscopy is more commonly used for the diagnosis and treatment of gastrointestinal diseases. Endoscope A flexible tube with a camera passes through the patient's mouth Or rectum and Navigate with the patient's digestive system. In spite of This is happening, the gastroenteritis can check View pictures of the gastrointestinal tract on the screen, and then manually detect and Classify anomalies in real-time. Endoscopic evaluation Disease classification decisions and decisions are highly dependent on Gastroenteritis. In other words, the discovery may be different From one clinician to another.

The existing disease assessment system needs more information, Improvements to improve the quality of diagnosis and treatment of gastrointestinal diseases. Therefore, autonomous detection And classification systems may help to optimize existing procedures, evaluation of symptoms and treatment.

The startup should be consistent and more effective. All normal And abnormal findings must be recorded in writing The report is for further investigation by the patient. Because The aforementioned fact that an autonomous system will provide By minimizing the workload of clinicians and Paperwork while increasing the consistency of the disease Detection, efficiency, and patient time consumption concern. In addition, early diagnosis of the disease may help Control or prevent the spread of infection. In this research work, we proposed an autonomous system to Classify gastrointestinal diseases through deep integration models.

Literature Review

Abstract

The human gastrointestinal (GI) tract may be contaminated by different diseases. In case, these diseases are not diagnosed at first stages, there are possibilities to advance into gastrointestinal cancer, which is a common sort of malignancies with yearly global cases surpassing one million. Endoscopy is the often-used technique to diagnosis anomalies. Various methods using artificial intelligence (AI) have made a major contribution to the field of medical imaging and video-based diagnostics, including the classification of gastrointestinal (GI) diseases, such as radiology, pathology, and endoscopy. In the recent past, many research groups have proposed multiple GI tract disease classification methods based on image processing and machine learning techniques. Still this debate is going on. In this work we propose a deep learning model based deep convolution neural network. Our model aims to automatically detect diseases from endoscopic images. The newly designed architecture is confirmed on the publicly available dataset KVASIR, which has 8000 images.

Keywords in Article

Gastrointestinal (GI), Convolutional Neural Network (CNN), Transfer Learning, Classification, Inspection, Pattern recognition, Artificial intelligence, Tensorflow, keras.

Deep Learning in Medical Field

Modern computer science and engineering theories and methodologies have invaded a huge area in manually processed tasks. As a result of that, most of the medical diagnosis and disease prediction are done using computer science programmes. Among those computer technologies deep learning is one of the most prominent fields.

Pharmaceutical industry is investing a lot on this deep learning field for medical usages. The reason for that is they have massive amounts of data collected from researches and patient outcomes. The companies try to use deep learning on these data sets and get predictions on new drugs. BenevolentAI is a good example for this.

This company has developed custom algorithms to analyze current and past medical researches and find clues about new drugs.[1]

Another problem found in the medical field is, the problem of detecting infected and non infected cells using MRI and CT scans. However medical research companies are changing this paradigm through deep learning. They collect millions of past images with their results and use deep learning to train those data sets. After that they put new images and get the most accurate results. [1]

As discussed so far the main research involved in this research is “How to apply deep learning to the medical field to improve the outcome”.

Historical Background

When we draw back to 1970, in 1970 medical images were analyzed to get efficient outcomes. But those image processing tasks were based on simple algorithms. Due to this the operations were limited to small areas. By 1990, supervised learning technique, which is used to train a data set to get predictions, became popular. This technique was applied to medical image analyzing. As a result of this there was rapid improvement in the medical sector. Still today, we can see the effects of this application. In recent, Conventional Neural Networks became so popular. Today we can see a huge range of medical areas are using these CNN techniques. Moreover, researchers are still doing research to improve these applications.

Specific Research Area Chosen for Research

As we discussed above, we can apply deep learning into a wide range of applications in medical fields. However, to observe in depth, we narrow this research into specific area. Through this research, it is intended to observe how we can use deep neural networks to classify anomalies in gastrointestinal tract through endoscopic images.

Nature of Research Area

Digestive system of the human is often affected by several diseases. Some of those diseases may be not serious and some of them will cause death. Due to this reason, it is important to detect these diseases as soon as possible. Computer automation can be used to detect these diseases within a short period of time.

These diseases can be divided into three groups,

1. Anatomical Landmarks : These types of diseases can be easily detected using endoscopic images. This anatomical landmarks can be occurred at [2],

1. Z-Line
2. Pylorus
3. Cecum

2. Pathological Finding : In this type, it can be observed abnormal features within the gastrointestinal tract. It can be seen as damage or change in the normal situation. These anomalies are [2],

1. Esophagitis
2. Polyps
3. Ulcerative Colitis

3. Polyp Removal : Polyps in the large bowel may be precursors to some diseases. Staining dye is added to facilitate accurate identification of the polyp margins. Here are some classification on dyed polyp [2],

1. Dyed and Lifted Polyps
2. Dyed Resection Margins

To make this detection task automated it is necessary to get an inside image of the patient's stomach, esophagus, first part of small bowel and large bowel. For this task the endoscope (A tube with light and camera which is sent to the stomach through mouth and throat) is used. However this operation needs expensive technical equipment [3].

A trained Convolutional Neural Network can be used to detect the disease category in the newly supplied image, which is captured from the endoscope.

Methodology of Research

Past research papers, previous proposals, books related to Conventional Neural Networks, Article related to endoscopic images and some websites were used to research on this topic. Critical reviews were done on those research papers and proposals. Then important facts and data were gathered with their references. Those references are added as the reference of this article. Readers can view them in the reference section.

Overview of Reviews

In recent studies on Classification on Anomalies In Gastrointestinal Tract through Endoscopic Imagery with Deep Learning studies have used KAVISAR dataset.

Chathurika Gamage et al. [4] propose a system to predict eight-class anomalies of the digestive tract diseases. This proposed system use an ensemble of deep features as a single feature vector by combining pre trained DenseNet-201, ResNet-18, and VGG-16 CNN models as the feature extractors followed by a global average pooling (GAP) layer to predict eight-class anomalies of the digestive tract diseases. Dataset was splitted before feeding to the models as follows: 80% of the data as a training set and 20% as test[4]. Five-fold cross-validation was used on the training set for tuning the hyper-parameters and selecting the best model, whereas the test set is used to give an unbiased estimation to the final model selected from the cross-validation[4]. According to the author, the system has achieved an accuracy of over 97%. This is a great achievement.

Samira Lafraxo et al. [5] proposes a system to automatically detect diseases from endoscopic images. Author proposes a novel deep learning model based deep convolutional neural network model for this research. According to the author the system was successfully achieved 96.89% in terms of accuracy.

Proposed Methodology

According to [6], even though various research groups came up with various conclusions, an effective and comprehensive solution is not available to classify Anomalies in Gastrointestinal Tract through Endoscopic Imagery with Deep Learning in the literature so far.

Our proposed methodology aims to classify Anomalies in Gastrointestinal Tract through Endoscopic Imagery with Deep Learning. This is hope to implement as follows,

- Build Conventional Neural Network using image dataset,
 - Flatten the input image dimensions to 1D (width pixels x height pixels)
 - Normalize the image pixel values (divide by 255)
 - One-Hot Encode the categorical column
 - Build a model architecture (Sequential) with Dense layers
 - Train the model and make predictions
- Images of the patient are collected according to the procedure mentioned in section 'Nature of Research Area'.
- Input the patient's images to CNN to get prediction.

Many technologies and tools are used to build CNN. Some of those tools are,

- [Keras](#): Deep Learning Framework comes with a Tensor flow backend.
- [NumPy](#) & [Scikit-learn](#): Used to work with image pixel matrices and scientific calculations.
- [Matplotlib](#): Used to plot data for analyzing purposes.

Dataset will be divided into Train and Test. The processing is done again and again until accuracy becomes high. Following parameters will be changed to improve accuracy,

- 1.Number of Layer in CNN
- 2.Number of Kernels
- 3.Size of Kernel

Dataset Used

For this research it is used **Kvasir dataset** consists of 8,000 annotated GI tract images in 8 different anomalies (as mentioned above) where 1000 images belong to each type.

This dataset is verified by medical doctors. These data sets consist of images with resolution from 720x576. Dataset is 1.15 GB in size. This dataset can be downloaded from the link below.

Link to Dataset : <https://datasets.simula.no/kvasir/data/kvasir-dataset.zip>

These images are divided into anomalies categories as shown below,

- 1.z-lines
- 2.pylorus
- 3.cecum
- 4.esophagitis
- 5.polyps
- 6.ulcerative colitis
- 7.dyed-lifted-polyps
- 8.dyed-resection-margins.

There are 8 folders in the dataset which contain 1000 images for each category in each folder.

Discussion

In this paper we discussed how deep learning became a trend in the medical field. For this task we use the Conventional Neural Network. The proposed solution will be trained with existing datasets verified by doctors. However, after training data sets we can predict diseases within a few milliseconds. It will save time and the cost.

Before deep learning involves, medical imagery was observed by doctors to predict the situation of the patient. This is one of the most critical and difficult tasks done by a doctor. However, modern world computer scientists were able to automate these tasks by using deep learning. The most important thing is, although it was automated, the accuracy is still high. The researchers can use the outcome of these research to solve the problems occurring in other medical fields.

References

- [1] Ekins, S. (2016). The next era: deep learning in pharmaceutical research. *Pharmaceutical research*, 33(11), 2594-2603.
- [2] Pogorelov, K., Randel, K. R., Griwodz, C., Eskeland, S. L., de Lange, T., Johansen, D., ... & Riegler, M. (2017, June). Kvasir: A multi-class image dataset for computer aided gastrointestinal disease detection. In *Proceedings of the 8th ACM on Multimedia Systems Conference* (pp. 164-169).
- [3] *Digestive Diseases and Endoscopy*. (2004, July 9). WebMD. <https://www.webmd.com/digestive-disorders/digestive-diseases-endoscopy#1>
- [4] Gamage, Chathurika & Wijesinghe, Isuru & Chithraranjan, charith & Perera, Indika. (2019). GI-Net: Anomalies Classification in Gastrointestinal Tract through Endoscopic Imagery with Deep Learning. 10.1109/MERCon.2019.8818929.
- [5] S. Lafraxo and M. El Ansari, "GastroNet: Abnormalities Recognition in Gastrointestinal Tract through Endoscopic Imagery using Deep Learning Techniques," *2020 8th International Conference on Wireless Networks and Mobile Communications (WINCOM)*, Reims, 2020, pp. 1-5, doi: 10.1109/WINCOM50532.2020.9272456.
- [6] 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.

METHODOLOGY

In this work we are going to train simple Convolution Neural Network using Keras with Tensorflow back end. For this task we use python as our programming language. We used Kvasir data set of images as training data. These data will be split into training and testing sets.

As the first step let's import all required libraries,

```
import os
import tensorflow
import numpy as np
from PIL import Image
from keras import backend as K
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.utils import np_utils
from keras.preprocessing.image import img_to_array
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
```

After loading required libraries into notebook, we prepare images for training. For that first we create two arrays, one to store image and other to store categories of images. Kvasir data set contains different categories of images in different folders. Hence we read all those directories one by one. For each image, we convert it into RGB mode, resize it, build pixel matrix, get transpose, reshape matrix and finally we add that image pixel array into one of above array and its category into other array. Likewise we prepare all images for processing.

```
file='kvasir-dataset'
m=n=96

classes=os.listdir("./"+file)
x=[]
y=[]
for classname in classes:
    print(classname)
    imagefiles=os.listdir("./"+file+"/"+classname+'/')
    for img in imagefiles:
        im=Image.open(r"./"+file+"/"+classname+'/'+img);
        im=im.convert(mode='RGB')
        imrs=im.resize((m,n))
        imrs=img_to_array(imrs)/255;
        imrs=imrs.transpose(2,0,1);
        imrs=imrs.reshape(3,m,n);
        x.append(imrs)
        y.append(classname)
```

Then we define some required parameters.

```
nb_classes=len(classes)
nb_filters=48
nb_pool=2
nb_conv=3
```

Now we divide processed images into training and testing data sets with ratio 80:20.

```
x=np.array(x)
y=np.array(y)
x=np.array(tensorflow.transpose(x,[0,2,3,1]))
x_train, x_test,y_train, y_test= train_test_split(x,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=42)

uniques, id_train=np.unique(y_train,return_inverse=True)
Y_train=np_utils.to_categorical(id_train,nb_classes)
uniques, id_test=np.unique(y_test,return_inverse=True)
Y_test=np_utils.to_categorical(id_test,nb_classes)
```

After that we define a checkpoint which will use to save the values for the epoch which has highest val_accuracy.

```
checkpointer = ModelCheckpoint(filepath="high_accuracy.hdf5",
                               monitor = 'val_accuracy',
                               verbose=1,
                               save_best_only=True)
```

Let's now proceed with conventional neural network construction. First we initiate model.

```
model=Sequential()
```

Then we configure Convolution layers and use MaxPooling to downsample and Dropout to prevent overfitting. We use Flatten to convert matrix into flat array. We also use Dense layer with relu activation and softmax activation.

```
model.add(Convolution2D(nb_filters,
                        (nb_conv,nb_conv),
                        activation="relu",
                        data_format='channels_last',
                        input_shape=(m,n,3)));
model.add(Convolution2D(nb_filters,
                        (nb_conv,nb_conv),
                        activation="relu"));
```


Here is the output of training result.

```
Epoch 1/11
100/100 [=====] - 82s 809ms/step - loss: 1.25
33 - accuracy: 0.4962 - val_loss: 0.8778 - val_accuracy: 0.5512

Epoch 00001: val_accuracy improved from -inf to 0.55125, saving model
to high_accuracy.hdf5
Epoch 2/11
100/100 [=====] - 76s 759ms/step - loss: 0.69
12 - accuracy: 0.6974 - val_loss: 0.7027 - val_accuracy: 0.6712

Epoch 00002: val_accuracy improved from 0.55125 to 0.67125, saving model
to high_accuracy.hdf5
Epoch 3/11
100/100 [=====] - 76s 762ms/step - loss: 0.56
67 - accuracy: 0.7708 - val_loss: 0.7544 - val_accuracy: 0.6712

Epoch 00003: val_accuracy did not improve from 0.67125
Epoch 4/11
100/100 [=====] - 76s 759ms/step - loss: 0.47
73 - accuracy: 0.7984 - val_loss: 0.7007 - val_accuracy: 0.7025

Epoch 00004: val_accuracy improved from 0.67125 to 0.70250, saving model
to high_accuracy.hdf5
Epoch 5/11
100/100 [=====] - 76s 760ms/step - loss: 0.43
90 - accuracy: 0.8121 - val_loss: 0.8095 - val_accuracy: 0.6875

Epoch 00005: val_accuracy did not improve from 0.70250
Epoch 6/11
100/100 [=====] - 76s 762ms/step - loss: 0.36
99 - accuracy: 0.8522 - val_loss: 0.7017 - val_accuracy: 0.7212

Epoch 00006: val_accuracy improved from 0.70250 to 0.72125, saving model
to high_accuracy.hdf5
Epoch 7/11
100/100 [=====] - 76s 763ms/step - loss: 0.22
43 - accuracy: 0.9233 - val_loss: 0.8479 - val_accuracy: 0.6913

Epoch 00007: val_accuracy did not improve from 0.72125
Epoch 8/11
100/100 [=====] - 76s 762ms/step - loss: 0.19
23 - accuracy: 0.9324 - val_loss: 0.9975 - val_accuracy: 0.6837
```


```
Epoch 00008: val_accuracy did not improve from 0.72125
Epoch 9/11
100/100 [=====] - 77s 769ms/step - loss: 0.16
08 - accuracy: 0.9390 - val_loss: 0.9047 - val_accuracy: 0.7163

Epoch 00009: val_accuracy did not improve from 0.72125
Epoch 10/11
100/100 [=====] - 78s 782ms/step - loss: 0.09
07 - accuracy: 0.9660 - val_loss: 1.1283 - val_accuracy: 0.6963

Epoch 00010: val_accuracy did not improve from 0.72125
Epoch 11/11
100/100 [=====] - 82s 821ms/step - loss: 0.06
09 - accuracy: 0.9796 - val_loss: 1.0712 - val_accuracy: 0.7125

Epoch 00011: val_accuracy did not improve from 0.72125
```


Now We can see that values for epoch with high val_accuracy 0.72125 is saved. We can use it for later.

 high_accuracy.hdf5

Now let's save final model as FinalModel.h5 which has val_accuracy of 0.7125

```
model.save('Final_Model.h5')
```

Now we can see Final Model is saved as h5.

 Final_Model.h5

Step by step we were able to train a CNN using Kvasir dataset, which has 71%+ accuracy.

Classification Experiments and Discussion of Results - Insights

In Methodology part we explained how to build a conventional neural network step by step. In this part let's observe some insights of our trained CNN.

Overall Accuracy of Test Data

After training of the neural network let's check the accuracy of CNN predictions for testing data.

Overall Accuracy of Test Data

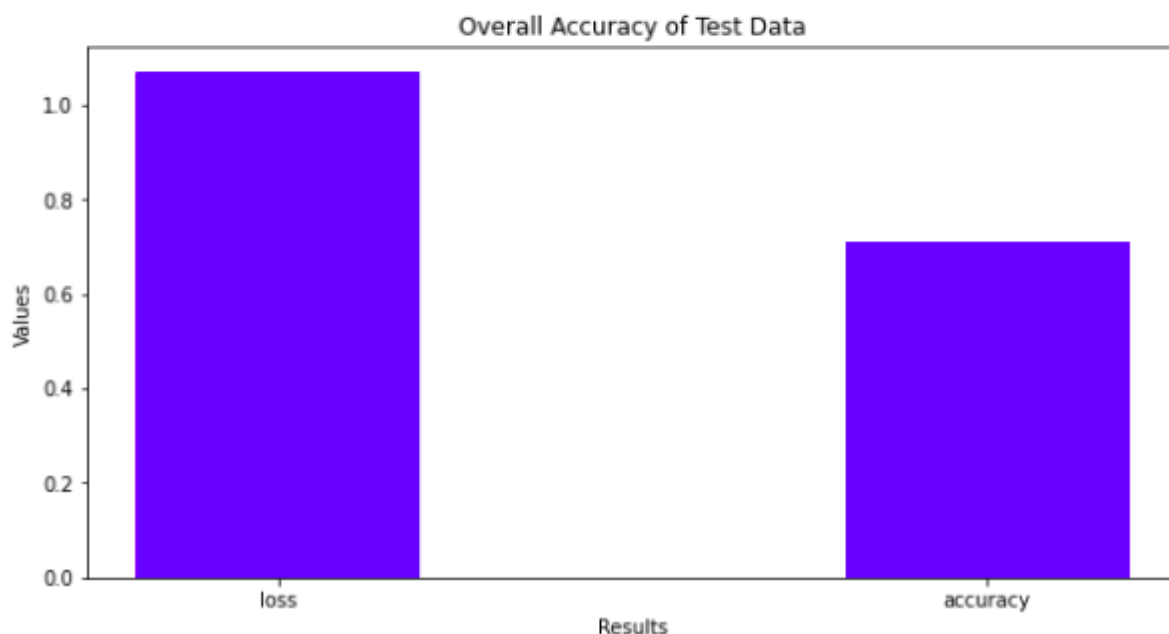
```
import numpy as np
import matplotlib.pyplot as plt

courses = ('loss', 'accuracy')
values = model.evaluate(x_test, Y_test)

fig = plt.figure(figsize = (10, 5))
plt.bar(courses, values, color = 'blue',
        width = 0.4)

plt.xlabel("Results")
plt.ylabel("Values")
plt.title("Overall Accuracy of Test Data")
plt.show()
```

25/25 [=====] - 4s 147ms/step - loss: 1.0712
- accuracy: 0.7125

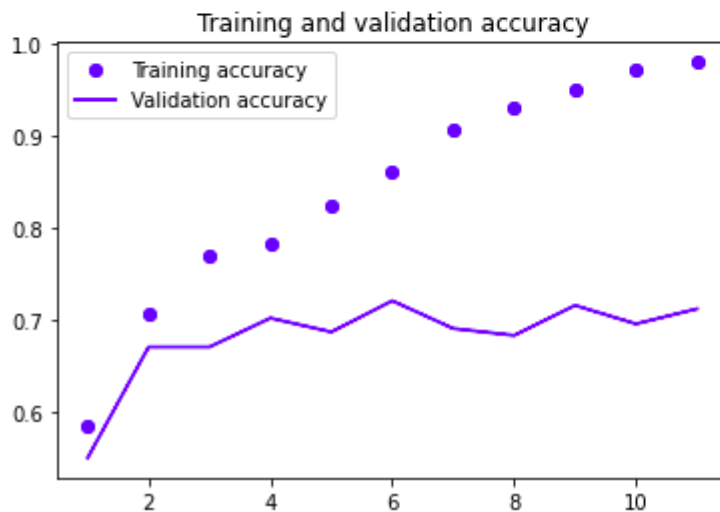


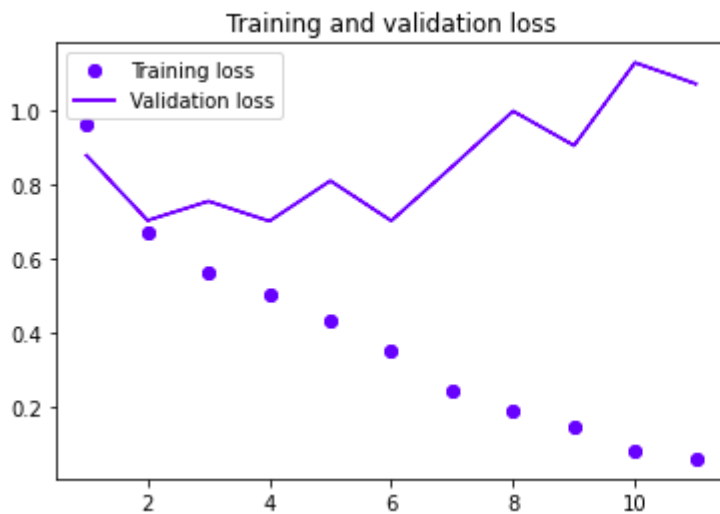
Hence we can see this has 1.0712 of loss and 0.7125 of accuracy.

Loss and Accuracy During Training

Loss and Accuracy During Training

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Intermediate Layers

Intermediate Layers

Function to view intermediate layers

```
import tensorflow as tf

def visualize_conv_layer(layer_name):
    layer_output=model.get_layer(layer_name).output
    intermediate_model=tf.keras.models.Model(inputs=model.input,outputs=layer_output)
    intermediate_prediction=intermediate_model.predict(x_train[2].reshape(1,96,96,3))

    row_size=4
    col_size=8

    img_index=0
    print(np.shape(intermediate_prediction))
    fig,ax=plt.subplots(row_size,col_size,figsize=(10,8))

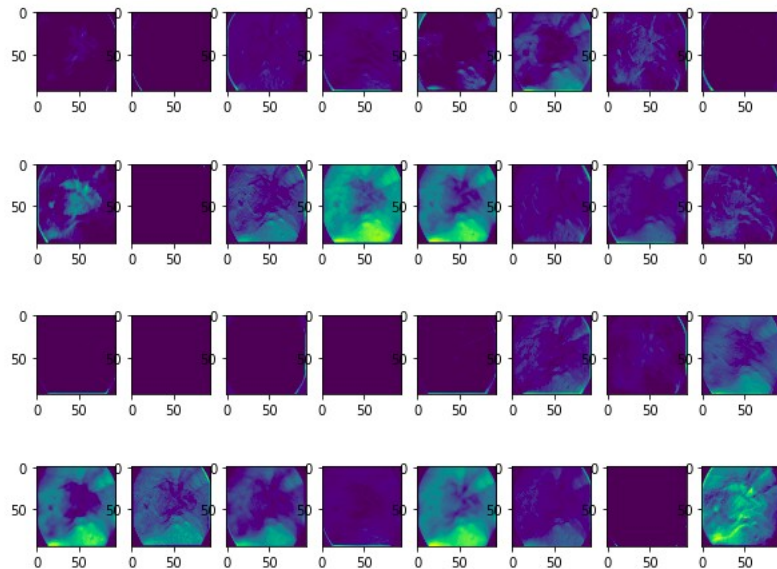
    for row in range(0,row_size):
        for col in range(0,col_size):
            ax[row][col].imshow(intermediate_prediction[0, :, :, img_index])
            img_index=img_index+1
```

Layer 1 Output : conv2d

```
visualize_conv_layer('conv2d')
```

WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7f76b2e93310> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

(1, 94, 94, 48)

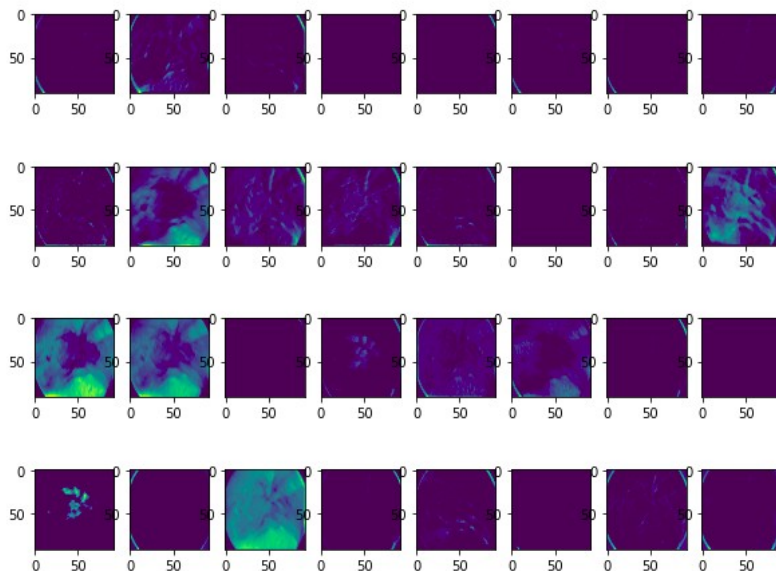


Layer 2 Output : conv2d_1

```
visualize_conv_layer('conv2d_1')
```

WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7f76b2ddc790> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

(1, 92, 92, 48)

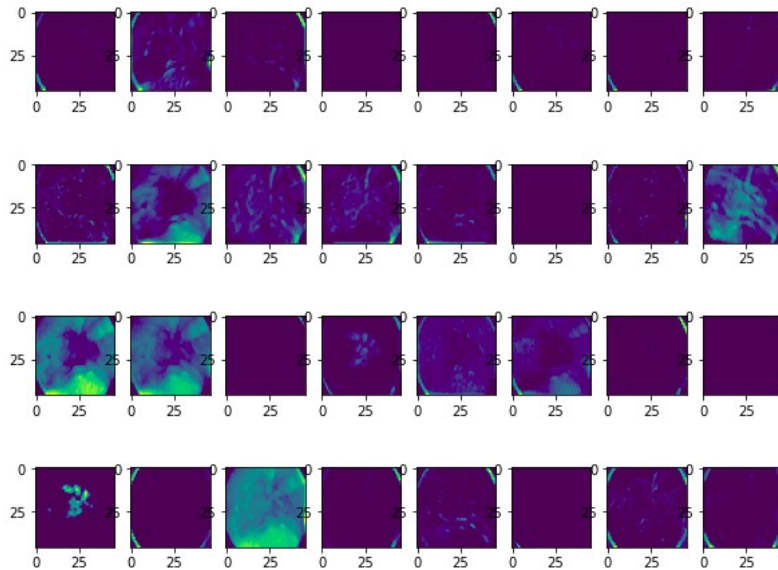


Layer 3 Output : max_pooling2d

```
visualize_conv_layer('max_pooling2d')
```

WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7f76b0c8c160> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

(1, 46, 46, 48)

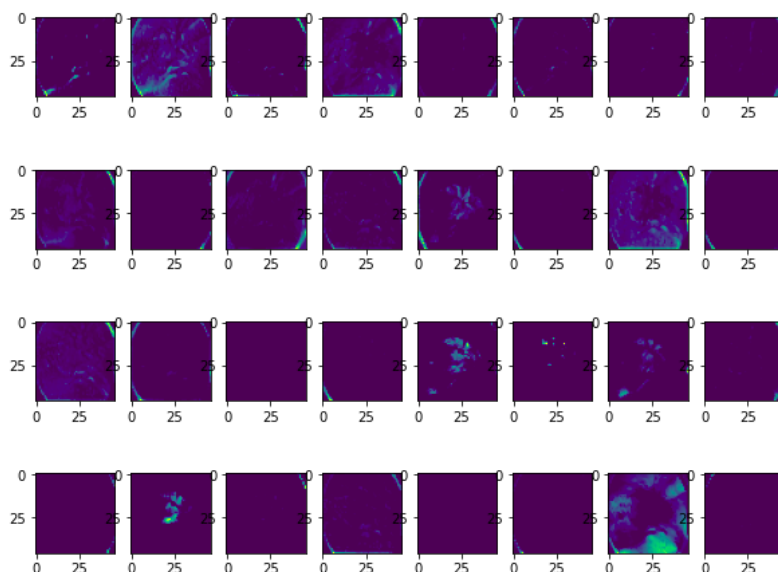


Layer 4 Output : dense

```
visualize_conv_layer('dense')
```

WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7f76b085f4c0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

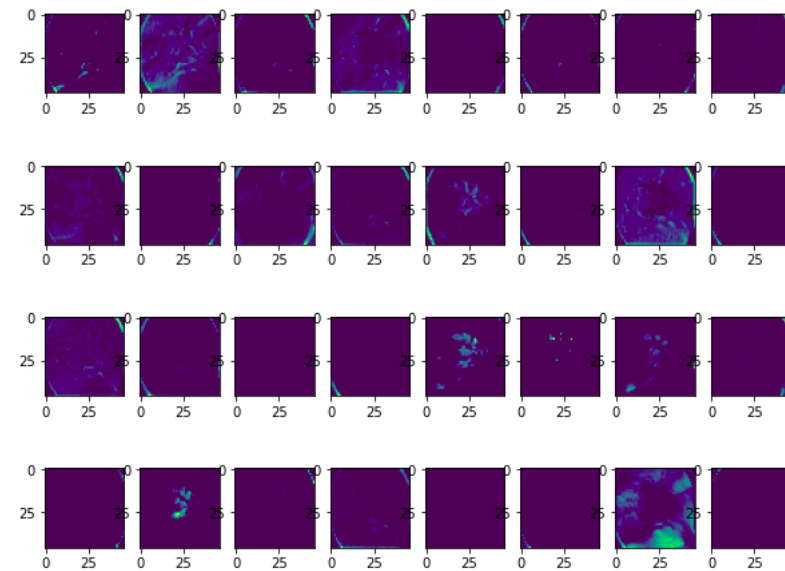
(1, 46, 46, 64)



Layer 5 Output : dropout

```
: visualize_conv_layer('dropout')
```

WARNING:tensorflow:11 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7f76b35755e0> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has experimental_relax_shapes=True option that relaxes argument shapes that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.



Accuracy by Category

```
import random
list_file=os.listdir(file)

accuracy={}

for folder in list_file:
    images=os.listdir(file+'/'+folder)
    total_images=len(images)
    correct_predicts=0
    for image in images:
        im = Image.open("./"+file+"/"+folder+'/'+image)
        imrs = im.resize((m,n))
        imrs=img_to_array(imrs)/255;
        imrs=imrs.transpose(2,0,1);
        imrs=imrs.reshape(3,m,n);
        x=[]
        x.append(imrs)
        x=np.array(x);
        x=np.array(tensorflow.transpose(x,[0,2,3,1]))
        predictions = model.predict(x)
        max_val=max(predictions[0])
        category=None
        i=0
        for val in predictions[0]:
            if val== max_val:
                category=list_file[i]
                i+=1
        #print(category)
        if category==folder:
            correct_predicts+=1
    acc=(correct_predicts*100.0)/total_images
    print(acc)
    accuracy[folder]=acc

print(accuracy)
```

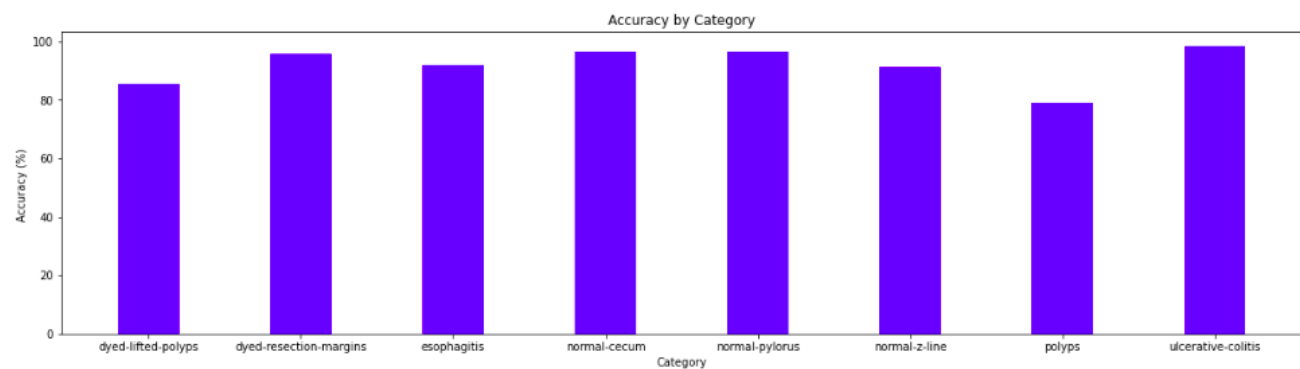
85.4
95.8
91.8
96.4
96.4
91.0
79.0
98.4

```
import numpy as np
import matplotlib.pyplot as plt

courses = accuracy.keys()
values = accuracy.values()

fig = plt.figure(figsize = (20, 5))
plt.bar(courses, values, color = 'blue',
        width = 0.4)

plt.xlabel("Category")
plt.ylabel("Accuracy (%) ")
plt.title("Accuracy by Category")
plt.show()
```



Conclusion

In this paper we discussed how deep learning used to classify medical images. For this task we trained a Conventional Neural Network. The CNN was trained using Kvasir dataset to detect image category. Finally we were able to get 71+% accuracy. However, after training data sets we could able to predict diseases within a few milliseconds. It saved time and the cost.

Before deep learning involves, medical imagery was observed by doctors to predict the situation of the patient. This is one of the most critical and difficult tasks done by a doctor. However, modern world computer scientists were able to automate these tasks by using deep learning. The most important thing is, although it was automated, the accuracy is still high. The researchers can use the outcome of these research to solve the problems occurring in other medical fields.

This CNN can be improve by doing different researches with different values. Our final accuracy is about 71%. But it is not good enough when this model is used practically in the medical field. So, we have to improve the accuracy and reduce the loss by continuing the training process further using different strategies. As an example, we can use pre-trained CNN models such as DenseNet-201, ResNet-18. And different data augmentation techniques can be used to improve the model accuracy.

*** END ***