Student Name: Job Thomas Thekkekara

Student No: R00195427

For the module Data8001 as part of the

Master of Science in Data Science and Analytics, Department of Mathematics, 2021/22

Declaration of Authorship

I, Job Thomas Thekkekara, declare that the work submitted is my own.

- I declare that I have not obtained unfair assistance via use of the internet or a third party in the completion of this examination.
- I acknowledge that the Academic Department reserves the right to request me to present for oral examination as part of the assessment regime for this module.
- I confirm that I have read and understood the policy and procedures concerning academic honesty, plagiarism and infringements.
- I understand that where breaches of this declaration are detected, these will be reviewed under MTU (Cork) policy and procedures concerning academic honesty, plagiarism and infringements, as well as any other University regulations and policies which may apply to the case. I also understand that any breach of academic honesty is a serious issue and may incur penalties.
- EXAMINATION/ASSESSMENT MATERIAL MAY, AT THE DISCRETION OF THE INTERNAL EXAMINER, BE SUBMITTED TO THE UNIVERSITY'S PLAGIARISM DETECTION SOLUTION
- Where I have consulted the published work of others, this is always clearly attributed
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this work is entirely my own work
- I have acknowledged all main sources of help

Signed: Job Thomas Thekkekara

Date: 09/05/2022

Introduction

Convolution neural networks (CNN) or Convnets is a type of neural network that is largely used for computer vision and image classification problems. There are also other applications for CNN for sequential data such as time series, audio and NLP. Usually, the convnets are stacked first and the output is given to dense (fully connected) neural network. A convnet takes input as tensors of shape (image_height, image_width, image_channels) to the first layer. First two are the spatial axis and the third one is the channel axis. For a colour image, number of image channels will be 3 (red, green, blue) and hence colour images are 3D images. For a black and white image, the number of channels will be one and hence it will be a 2D image. The main difference between densely connected layer and convnet layer is that convolution layers learn local patterns first whereas dense layers figure out global patterns. In case of images, patterns in small 2D windows are learned by convnets first. If a convnet learns a certain pattern in a part of the picture, it can recognize it anywhere, making it more data efficient when processing images. Also, the first convolution layer will learn small local patterns such as edges, a subsequent convolution layer will learn larger patterns made of the features of the first layers, and so on. Thus, more complexities in visualizations are learned efficiently.

In a CNN, convolution operation between a kernel/filter and the input data is the main operation. Traditionally convolution is an operation performed on two functions to create a third function. In image processing convolution is used to transform images by applying a kernel/filter over each pixel of input image(Albawi et al., 2017). Kernel is essentially a matrix whose size and values determines the transformation effect of convolution process. The steps of a convolution process are - First, a kernel is placed on top of an image position (say top right corner) such that the kernel values are on the top of pixel values of the image. Then, each value of the Kernel is multiplied with the corresponding pixel it is over. Finally, sum of the resulting multiplied values becomes the first value (centre pixel value) of the output/feature map (e.g.: at top right corner). This process is repeated across the entire image. The way in which a kernel moves across the image can be define by stride. The distance between two successive windows of a kernel is a parameter of the convolution, called its stride(O'Shea and Nash, 2015). Default is 1, but it is possible to have other stride convolutions. Using stride 3 means the width and height of the feature map are down sampled by a factor of 3. These down samplings can sometimes come in handy. An important point is that it is not one kernel that moves across the image. There are several ones. Each one will pick a different pattern about the image. As more convolution layers are stacked the number filters/kernels in each layer is increased usually. Thus, depth of the feature map gets increased as the layers increases while height and width decreases.

Training of a neural network happens in two main sections- Forward propagation through the network and backward propagation to update weights and minimise losses. In forward propagation, each neuron in a layer has its inputs (X) multiplied by weights (W) and then a bias (B) is added to it. ((W * X) + B).Output of this is given to an activation function which creates non linearity. This nonlinearity helps neural networks to detect complex patterns. Finally, the output of the model is compared with the actual output to calculate loss. Once the losses are found, backward propagation starts. Gradient descend is used to update weights in each of the neurons to minimise losses. Chain rule of differentiation is used in the weight

update by gradient descend. This process is continued till the losses are minimised to the optimum level. Training a neural network with all the training data for one cycle (forward and backward propagation) is called an epoch. Batch size is the number of samples that will be passed through to the network at one time. Batch size is specified so that only a small portion of data passes through network at a time. An epoch is completed when all the batches are finished. Several epochs of training are done on a neural network. In this classification problem, a batch size of 32 is given for training and the shape of image is (32, 200, 200, 3).

2D input image with a 1D convolution operation

A 2D input image will have image width and height but only one input channel. 2D images are grey scale images with no colour channels. Each pixel value of the image will have values between 0-255. In 1D CNN the kernel moves only in 1 direction. The input and output data of 1D CNN is two dimensional(Kiranyaz et al., 2021).

Below is an image of an example worked out to perform 1D CNN on a 2D image. An image of size (4*4) was selected as feature vector/input. Convolution operation is performed by a filter of size (4*1). The weight vector strides across feature vector where each value of weight is multiplied with the corresponding pixel value of image. This generates a feature map or output of size (4*1). The number of parameters is calculated as

- Weights in one filter of size (4,1) = 4*1 =4
- Bias =1. One bias will be added to each filter. Since only one kernel is used, bias =1
- Total number of parameters with one kernel of size (4,1) = 4+1 =5

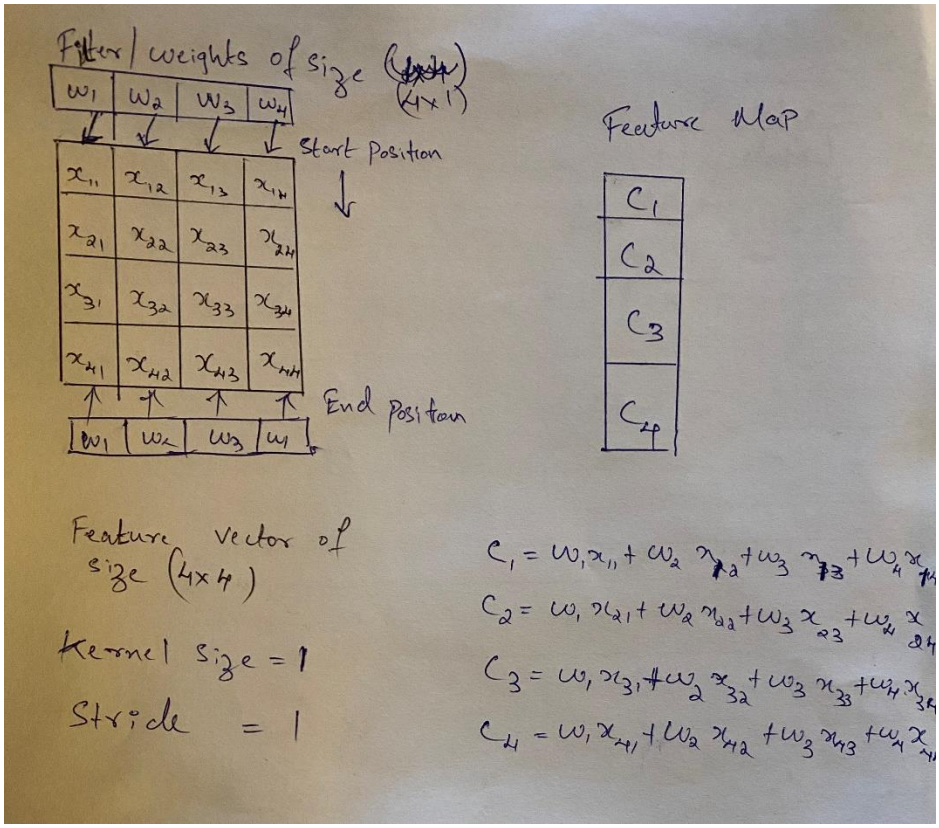

Fig1: Working of 1D convolution on a 2D image

In Fig1 the kernel width was chosen as one and the stride value was also chosen as one. In 1D CNN, the width of the image and length of the kernel must be the same. Then only movement of kernel over image and multiplication of corresponding=g values will happen properly. But it is possible to have kernels of larger width.

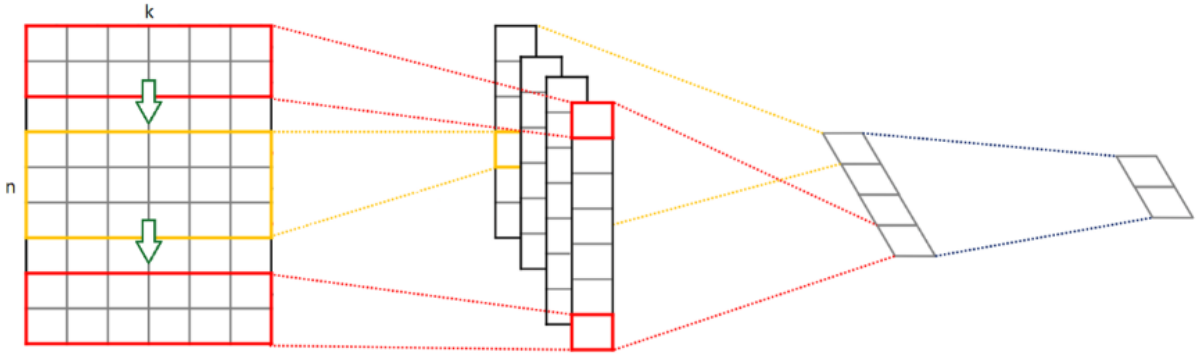

Fig2: Working of kernels of different sizes on an image. The output is a feature map which is flattened later and given to a fully connected neural network.("Social Network for Programmers and Developers," n.d.)

In figure 2. Feature vector is of size (9*6) is defined. Kernel in red is of size (6*2). It has a width of two. Hence first two rows of input image will be convoluted with corresponding weights to produce first result of feature map. Then 2^{nd} and 3^{rd} rows of input image will be convoluted with corresponding weights to produce second result of feature map and so on. If the width of kernel was one, the feature map would have a size of (9*2). But here the feature map has a size of (7*2).

Also, if the value of stride was set as 2 then initially first two rows of input image will be convoluted with corresponding weights to produce first result of feature map. Then the kernel will move two steps in one direction instead of one step. Hence 3^{rd} and 4^{th} rows of input image will be convoluted with corresponding weights to produce second result of feature map and so on.

Thus, changing the size of kernel and number of strides changes the size of feature map. Formula for this is given as:

Feature map shape= (W-F+2P)/S + 1

W → Size of input

F → Size of filter

P → Length of Padding

S → Length of Stride

In Fig 1: Feature map shape= (4-1 +(2*0))/1 +1 = (4-1) +1 = 4

Also, it is to be remembered that there are multiple kernels operating on an input to learn different type of features. There are certain kernels designed to learn edges, sides etc.

3D input image with a 2D convolution operation

A 3D input image will have image width, height and three input channels. Theses input channels are colour channels- usually red, green and blue. For each cell the pixel value is the combination of values in these colour channels. In 2D CNN the kernel moves in 2 directions. This is the most widely used technique for image processing. Tensorflow and Keras provide CONV2D() that enables this.

So, a 3D RGB input image has pixel intensities in height, width and a depth of 3. A kernel (matrix of weights) here can be thought of as a set of filters each of which is applied to one channel of image. Each filter slides over each of 2D input feature matrix data and performs element wise multiplication with the part of the input it is currently on and finally sums up the result into a single output pixel. This process is repeated by the filter for every location it slides over and thus it converts each 2D input matrix into a 2D feature map. The output produced from each of the channel is summed up to form one channel which is the final 2D feature map. Each of the output filter also has one bias term. The bias term gets added to the output channel finally.

Even though input is 3D, output shape is not 3D, but a 2D matrix. Also, the filter depth must match with the number of input channels. Although the kernel size mentioned in CONV2D() is only (3,3) the true dimensions of the kernel when initialised will be 3*3*3.

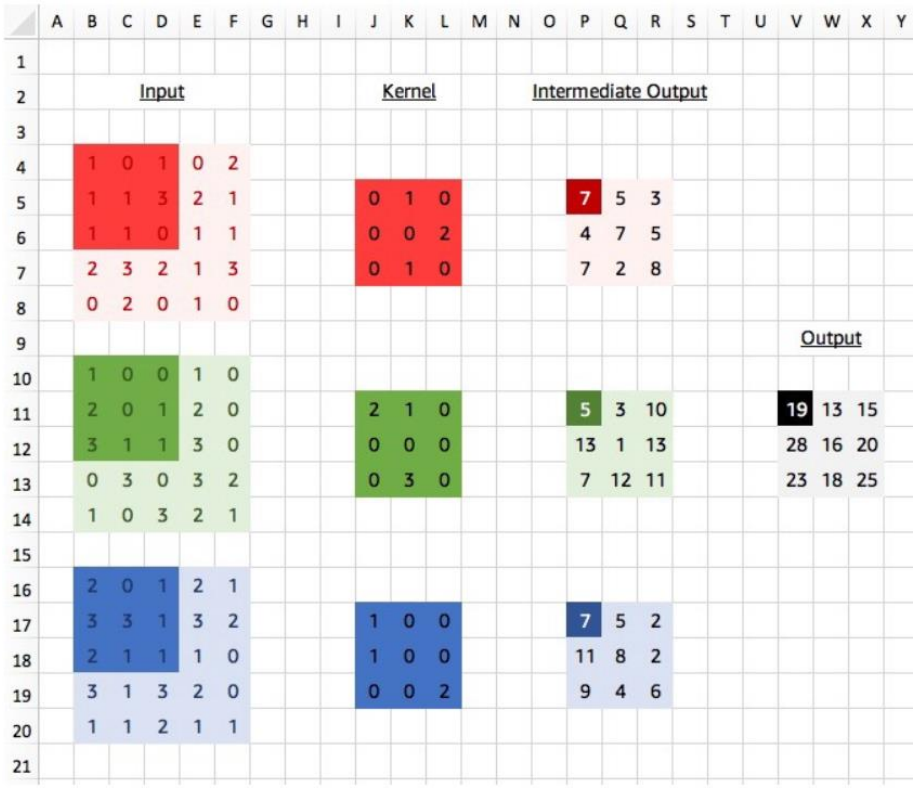

Fig3: 2D convolution on 3D image(“Multi-Channel Convolutions explained| Medium,” n.d.)

Figure 3 shows an experiment in which a 5*5 RGB input feature is convoluted with a 3*3 kernel to produce a feature map of 3*3. Stride here is 1.

Feature map shape= (W-F+2P)/S + 1

W=5, F=3, P=0, S=1

Feature map shape= (5-3+ (2*0))/1 +1 = (5-3) +1 = 3

Calculation of number of parameters:

- Parameters in one filter of size (3,3) =3 *3 =9
- The filter will convolve over 3 channels, RGB of input image. So number of parameters in one filter will be 3*3*3 =27
- Bias =1. One bias will be added to each filter
- Total parameters for one kernel of size (3,3) with depth 3 = (3*3*3) +1 =28
- The total number of filters =3
- Total number of parameters for 3 kernel of size (3,3), input image channel =3 will be 28*3 = 84

Often **max pooling** is done after a CNN layer to give more summarized or abstracted form of result which further prevents the model from overfitting by reducing variance. Also, computational cost is reduced as the overall size of the output matrix reduces and the number of trainable parameters decreases. It also helps in extracting smooth and sharp features. max pooling aggressively down sample feature maps(Al-Saffar et al., 2017). It is done by first placing a window at the output of a convolution layer/feature map. Then maximum value occurring in this window is extracted as output. This window is shifted across the feature map to get a down sampled one.

Padding: When a 5 * 5 feature map (25 tiles) is convoluted with a 3 * 3 kernel, there are only 9 tiles that can be properly centred with this (3,3) window. Hence the output feature map shrinks a little. In order to get an output feature map with the same spatial dimensions as the input, padding is used. Padding is done by adding an appropriate number of rows and columns on each side of the input feature map so that kernel windows can be placed centred around every input tile. Padding can be done. padded rows and column can be set with values such as zeroes or ones etc. Padding to an image processed by a CNN allows for more accurate analysis of images.

3D input image with a 3D convolution operation

In 3D CNN the kernel moves in 3 directions. The output is a 3Dimensional volume space such as cube or cuboid. Mostly used in medical image processing etc. 2D and 3D convolution doesn't have major difference. Essentially in 2D convolution, input image will be mostly 3D, but the filter depth will be the same as that of input layer depth. Hence the movement of filter is restricted to 2 dimensions. A 3D convolution is a generalization of 2D convolution in which the filter depth is smaller than the input layer depth – kernel size < input channel size. Thus, 3D filter can move in all 3 directions (height, width, channel). As the filter slides, at each position element wise operations (multiplication and addition) occurs which results in a final number. Since filter slides through a 3D space the output features can be arranged in a 3D space as well.

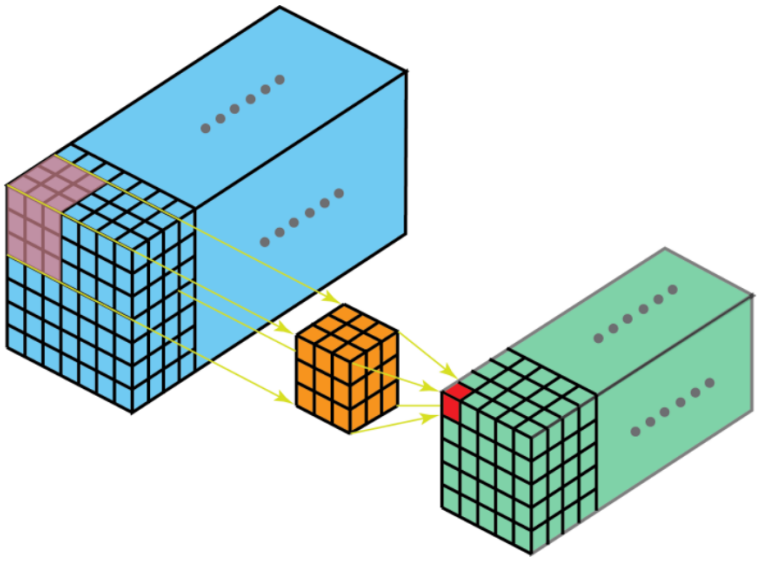

Fig4: 3D convolution on 3D images.(nbro, 2020)

If there is no stack of images, a 3D convolution produces similar parameters as of 2D convolution. 2D convolution is mostly used for images as it is computationally less intensive also. But for video processing which has stack of images, 3D convolution is used.

Consider a (3,3,3) filter doing convolution doing 3D convolution with 32 kernels

Calculation of number of parameters:

Number of weights = kernel width * kernel height * kernel depth (because 3d) x number of channels in input image * number of kernels = 3*3*3*3*32 = 2592

Number of biases = number of kernels = 32

Number of parameters = 2592+32 = 2624

Deep Learning

Kvasir dataset consists of 2.3GB of high quality medical annotated images that can be used for computer aided gastrointestinal disease detection. It consists of 8 classes of images. The image dataset was unzipped and was split into train test and validation datasets in 70:15:15b ratio.

Initial setup of Keras and tensorflow had caused several challenges. Google collab provided easy running of code without any set up requirements. But as the number and size of images were high, the GPU provided by google in free tier was not sufficient enough. The setup of tensorflow in personal pc caused many compatibility issues with other packages resulting in errors. Finally, a separate environment with tensorflow installation was done to train the model.

Python version: 3.8.8, Tensorflow version 2.0

Base Model

First a base model is built to process the images. models.sequential() in Keras allows to build the neural network architecture by stacking different layers. A 2D convolution over a 3D image is being performed here. Layers are added by specifying conv2D(). Input shape of the image is given as (200, 200, 3). The number of filters specified is 32. That is 32 filters are computed over the input. This becomes the number of channels of the output feature map of conv layer. The size of kernel is specified as (3,3). These are the typical specifications given in conv2D(Gulli and Pal, 2017). The activation function specified is 'relu', relu(x) if x>0 is x while relu(x) if x<=0 is 0. The output of convolution is given to activation function to create nonlinearity. That is convolution values which are less than one will be switched off (zero) by relu, creating a nonlinearity. The output of first layer is given to a max pooling function and then given to a second convolution layer with 64 filters. The output from convolution layer is flattened to a form a 1D array of pixel values. This is given as the input to a dense fully connected neural network. Output of neural network is given to a SoftMax function to produce the result

In the base model defined, padding is given as 'same 'so that input is padded to get the same sized output. Default parameter is 'valid' which has no padding. Due to padding output shape of first conv2D layer obtained is (None, 200, 200, 32) which us same as input except with larger depth. The number of parameters to be tuned was found to be 896 for the first layer. Number of parameters = Number of weights + Number of biases. A bias is added for each kernel. In first layer 32 kernels were provided.

Number of parameters = no: output channels * (no: input channels * kernel height * kernel width + Bias) = 32(3 * 3 * 3 + 1) = 896.

This output is given to a max pooling window of size (2,2) which decreases the size of its output by two-fold. Hence output becomes (None, 100, 100, 32). Width and height are reduced by half.

For second layer with 64 kernels, Number of parameters = no: output channels * (no: input channels * kernel height * kernel width + Bias) = 64*(32* 3 * 3 + 1) = 18496. Again, the output is given to max pool function reducing the size to (None, 50,50,64). Image height and width got halved while depth doubled.

The output of convnet is flattened to 1D array by using Flatten(). The shape becomes (None, 160000) which is (None, 50 * 50* 64). This is given as input to a dense layer of 128 neurons each with an activation function of 'relu'. Here the functionality (W*X + B) happens in each neuron. W is the weights; X is the input and B is the bias. X = 160000 which is the input from flattened 1D array. W = 128, each of the 128 neurons will have a weight associated with it. 128 * 160000= 20,480,000 parameters. Then 128 bias (B) parameters are added to it, one for each neuron. Thus, the total number of parameters become 20,480,128.

At the final layer there should be only one neuron which outputs the probability of the image belonging to respective classes. Usually, the output of a neural network will have a sigmoid or SoftMax activation function inside the neuron outputting probabilities. Sigmoid Function is mostly in the range of values between 0 and 1 or -1 and 1 and used for binary classification

problems. SoftMax Function takes numbers as inputs and normalizes them into a probability distribution proportional to the exponentials of the input numbers. Here since it is a multiclass classification problem SoftMax is used as activation function. The number of parameters at the output becomes (W * X + B) = (8 * 128 + 8) = 1032. All the parameters obtained are tuned to minimize losses at each epoch.

Comparison of Base model with Github Model

The defined base model was trained and validated for 49 epochs. 60 epochs were assigned for it. But since a callback was defined based on validation loss, only 49 epochs were completed. The test accuracy was found to be around 75.16%

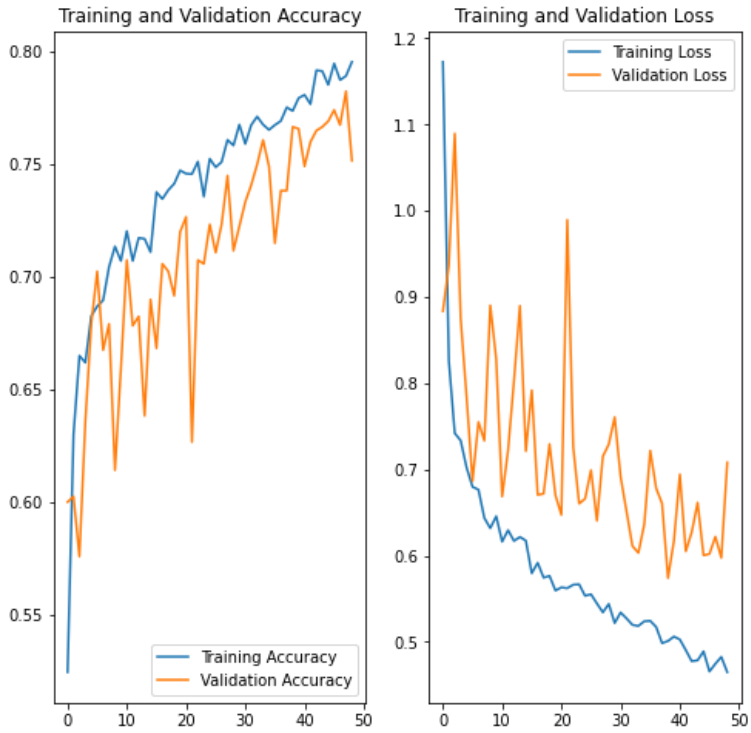

Fig5: Accuracy and loss plots for base model

Fig 5 shows Cross entropy loss and accuracy for both training and validation data after each epoch. The history of the model created contains loss and accuracy values at each epoch. This is useful in understanding model performance. Ideally losses should decrease, and accuracies has to increase as the number of epochs increases.

It was observed that both training and validation accuracy improved as the epoch increase although there were more fluctuations in validation accuracy. On observing losses, it is seen that training loss minimised as epochs increases but validation loss does not decrease that much. This indicates overfitting. Due to this consistent increase in validation loss the model was forced to stop training at 49 epochs. The base model has very a smaller number of layers and running too many epochs on such a small model might have caused this issue. The model might have learned too much from training data and hence can perform poorly with validation data.

On the other hand, the model given in GitHub is a far advanced model with 6 Convolutional layers and 2 neural network layers. Also, to minimize overfitting regularization techniques

such as BatchNormalization() is applied after certain layers in the model. It applies a transformation to maintain the mean output close to 0 and standard deviation of 1. Additionally, a technique called Dropout is used in dense neural network. By using Dropout, certain neurons are switched off so that the model does not learn too much from parameters. Dropout (0.2) means 20% of neurons are switched off. As the number of layers is high, running large number of epochs with large number of images does not lead to overfitting. The model has done 100 epochs and has produced a test accuracy of 85%

Advanced Model

A more advanced model was created to further improve the prediction accuracy. 7 layers of CNN and 3 layers of dense neural networks was used in this model. Additionally, both Batch Normalization and dropout technique was employed to prevent over fitting. The model was run for 100 epochs.

A user defined function was created to train the model. First, model.compile() is used to compile the model created. Inside that a loss function is defined to minimize error. There are several loss functions present according to regression or classification problems. Cross-entropy is the default loss function for classification problems. Mathematically it is the most preferred one. Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for output class. Since the problem definition here is multiclass classification, categorical cross entropy is used as loss function. Next, right optimizer has to be specified which calculates the weight update using gradient descend algorithm. Optimizers does the tuning of parameters of the model. Optimizers shape the model into the most accurate form by changing weights. Tensorflow & Keras provides several optimizer classes each implementing different algorithm. Gradient descend is used in most popular optimizer algorithms like Adagrad, RMSprop, and Adam. Here Adam is used as the optimizer. Accuracy metric is set as preferred on to compile the model as the accuracy of prediction is of interest here.

Fit() is used to fit data from the ImageDatagenerator. This function gets a python generator as input containing input features and targets. Because the data is generated endlessly the model needs to know how many samples to be drawn from generator before declaring than an epoch is over. This is specified in steps_per_epoch parameter(Arora et al., 2021). It is given as length(training data)// Batch size. This will use all of the data points, one batch size at a time. Batch size here is specified as 32(not too small or large) so that there occurs less memory problems and to not get stuck in local minima. There are 5600 training data points and hence steps_per_epoch will be around 5600/32 = 175. Hence as a single epoch gets completed about 175 data points will be fitted. 100 epochs are defined for the model. Also in Fit() validation data is provided and the number of data points to be used for validation at a time is specified by validation_steps parameter. Fit() method outputs training and validation accuracy and loss for each epoch according to the weights adjusted. Finally the number of epochs to be done is also specified in Fit(). Evaluate() evaluates the fitted function with validation data and outputs validation accuracy with the optimum weights being kept constant for the entire data.

A call-back parameter is specified to save the best model while training. The entire best model is saved with ModelCheckpoint(). Also, Early stopping is specified with a patience of

20 according to the performance metric validation loss. I.e. validation loss after each epoch will be considered and if it does not decrease for 20 epochs continuously then the model training is stopped, and the best model will be saved.

Model Performance

The model performance has improved from the baseline model. Both training and validation accuracy was found to be higher than baseline model. Model performance improved as the number of layers in CNN, dense layer increased and also with the higher number of epochs. As the layers got deeper the model was able to pick up more complex patterns from the images. Also, sufficient number of epochs were defined so that the model was able to learn from optimum number of images in each batch. Further nonlinearity was introduced by dropout techniques. Validation loss for the advanced model came down to be below 0.5 which has not happened for the baseline model. **Validation accuracy** was found to be **83.3%.** The **test accuracy** was found to be around **88%** which has improved a lot. The test data loss has also come down to 0.3

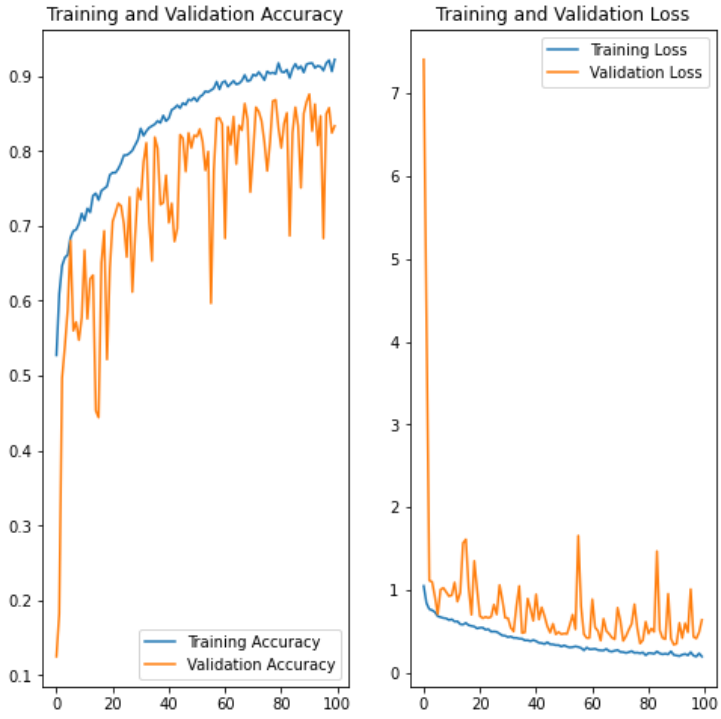

Fig6: Accuracy and loss plots for advanced model

From Fig6 it is observed that as the epochs increased both training and validation accuracy increased while both losses have decreased. The model shows no signs of over fitting even after 100 epochs. This might be due to larger number of layers combined with batch normalization and dropout techniques. Also, sufficient number of images were present for the model to train and validate. Giving the padding parameter as ‘same’ has also improved the accuracy of the model. More accurate image analysis was done because the filter was able to cover the entire image without losing the edges.

	Model Name	Test Data Loss	Test Data Accuracy
0	Base Model	0.531987	0.774167
1	Advanced Model	0.310508	0.882500

Fig7: Comparison between base model and the advanced model

Image Augmentation

Keras and Tensorflow is the most commonly used in python to implement neural network models. Keras is a deep learning API written in python and it runs on top of a machine learning platform called tensorflow.To load data to train and validate a neural network architecture, Keras provides ImageDataGenerator. ImageDataGenerator class gives easy way to augment images. Image augmentation is a way of applying different transformations to original images such as shifting, rotation, flipping etc which results in multiple transformed copies of the same image. After each transformation, a copy of the original image is generated. Main advantage is that it generates augmented images while the model is training thus making it more memory efficient. It only returns transformed images and it make sure that the model receives new variations of images at each epoch. Neural networks require huge amount of data to generate great results. It is because the model learns patterns by itself from data without much manual intervention and augmentation helps a lot in getting better results.

Cutmix is a recently developed image data augmentation technique. Before that dropout techniques were used to enhance the performance of CNN. But dropout removes informative pixels with a patch of either black pixels or random noise. This leads to information loss. In cutmix augmentation random patches are cut and pasted in training images. But instead of replacing them with noise, patches from other images are used. The ground truth labels are also mixed proportionately to the number of pixels of combined images.(Yun et al., 2019, p. 4)

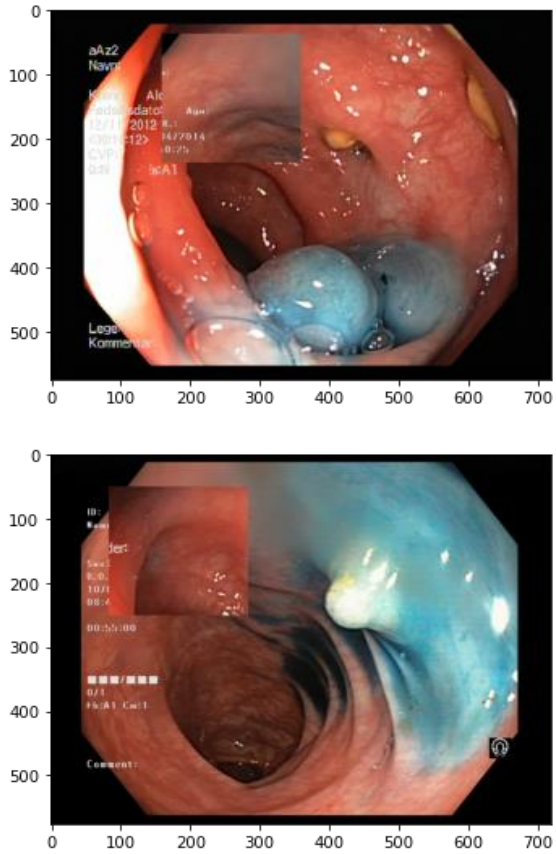

Fig8: Cutmix operation performed on sample images

As illustrated in fig7 The CutMix function takes two images and label pairs to perform the augmentation. Two images are randomly drawn from a batch of dataset. Then a label was randomly assigned to each of them. The second image is cropped, and this portion is padded in the first image at same location. The same is applied conversely too. A parameter lambda determines the size of the bounding box, and this parameter is stochastically sampled from a beta distribution. Beta distribution is a family of continuous probability distribution defined in an interval [0,1]. The label of each resultant augmented image is estimated as a weighted sum of the original label and the label of the image from which the patch is borrowed, where weights are lambda and (1-lambda) respectively.

Cutout is another image augmentation and regularization technique in which random square portions of training images are masked out. This technique helps in improving the overall performance and robustness of the CNN models. Often in image classification, there are problems such as occlusion in which certain parts of the image gets hidden by random objects. If the model is not able to learn such images from training data due to the lack of such examples the model fails in recognizing it during testing phase with new images. Cutout process helps to create occluded examples for training and hence prepare the model better to encounter occlusions in real world.(DeVries and Taylor, 2017)

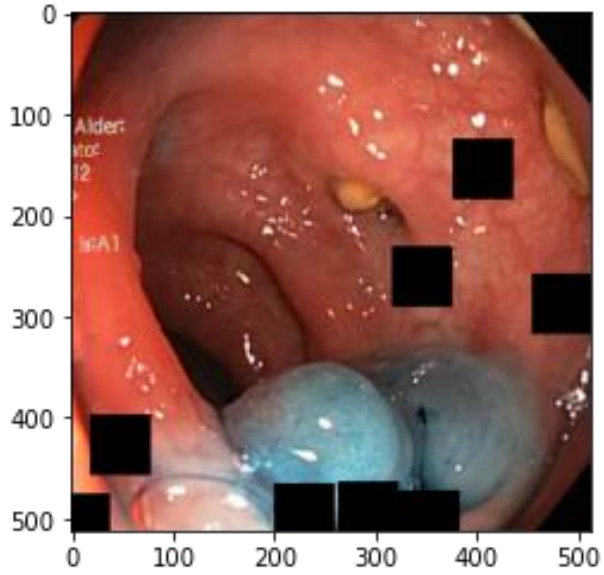

Fig9: Cutout operation performed on a sample image from dataset

In fig8 certain number of square patches are cut from the image. This is what happens in cutout process. On defining the hole size and number of patches required. The hole size can be subtracted from image width and height randomly as many times as the number of patches defined.

Mixup is another data augmentation technique that creates new images from weighted combinations of a random pair of images. In this way new corrupted images and labels are created to train the neural network model. The formula for mixup is as below.

newImage = alpha * image1 + (1-alpha) * image2

newTarget = alpha * target1 + (1-alpha) * target2

Here alpha parameter is a value within the range of [0,1] which is sampled from beta distribution. Literally mixing up of images and their corresponding labels happens here.

Implementation

Instead of a regular Imagedatagenerator for creating training samples, a user defined class is created that generated mixed up images(“Mixup | DLology,” n.d.). Alpha value of 0.2 was given initially. Later on, it takes random values from the beta distribution. Two separate generators are defined in the class that takes random images from the training set. Then the new mixed-up labels and images are returned by the class. This process gets repeated.

This new training data was given to the advanced model defined earlier. Due to high fluctuations in validation loss and accuracy, the model ran for only 44 epochs and produced a test accuracy of 85% which is less than the accuracy produced by the original image augmented advanced model. Perhaps more hyper parameter tuning and changing the optimizer from ‘adam’ to stochastic gradient descend might yield in better results.

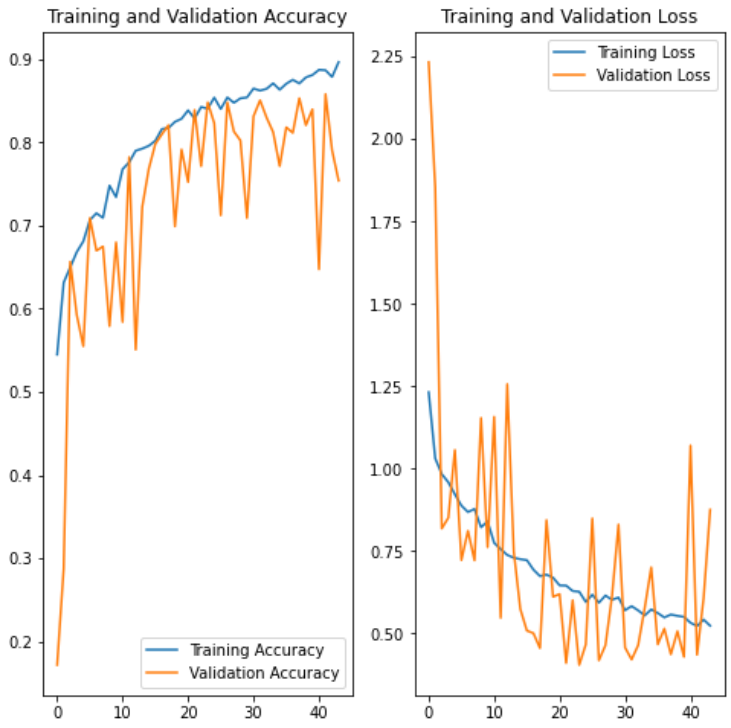

Fig10: Accuracy and loss plots for Mixup model

Variability of results

The performance and results produced by the model differs when different portions of training dataset are used. The same dataset when modelled with different architecture and tuned with different hyper parameters can yield in different results. Thus, reproducibility of the results is always a challenge.

Variability in the results might be due to the following factors(Renard et al., 2020)

- Variability present in the dataset
- The choice of model architecture
- Variability due to optimization and regularization
- Variability due to the choice of hyper parameters
- Variability due to infrastructure

To avoid variability in the dataset, random shuffling of images is first done before splitting the data into train and test. Also, a separate validation dataset is created to validate the model before testing with unseen data. Cross validation techniques can be performed which divides the dataset into several folds and assigning them into train validation and test sets. This also prevents the model from overfitting to training data.

In a neural network the number of parameters to be estimated is often greater than the number of images available. Another technique to control variability is to have more random training data points. To do these different kinds of image augmentation techniques with image generators are used so that the model gets to learn from different type of wide variety of

images. Both validation loss and accuracy can be monitored and early stopping of model training can be done to get the best fit model.

Several types of architectures can be created in neural network which can produce variability in results. Estimating the minimum number of layers and the base architecture required to get a certain level of accuracy with a given dataset can help in avoiding the variability of results due to model architecture to an extent.

Several trials with the same parameters and same dataset can be conducted to minimize variability due to optimization. Optimization techniques can be conducted several times along with cross validation. Later these results can be grouped. A one-way ANOVA test can be performed on different groups to test if there is a significant different between the groups or among them.

A plethora of hyper parameter tuning techniques is possible in machine learning modelling, and each can yield in different results. Instead of manually picking hyper parameters randomized search grid search can be used to find optimum hyper parameter values that makes the results more robust.

With infrastructure, the number of available processing units and other features like RAM of the system has to be considered. Distributed systems can be considered to produce results in reasonable time. Also, executions in compartments such as docker containers have to be encouraged to avoid external interactions

Images with Greenish box

Some of the images have a green picture in picture illustrating the position and configuration of the endoscope inside the bowel. Although this might aid in interpreting the images for medical professionals, it can adversely affect the performance of a machine learning model. The model learns from features of the images which in turn is the pixel intensity values. If there were more images from ‘class A’ with green boxes in training data, the model learns these patterns and finally if an image with green box comes from ‘class B’ in testing set, the model might classify it wrongly. This could lead to poorer performance of the model in a clinical setting.

Image augmentation techniques can be used here to avoid this. Mixup can be used here. The weighted combination of two images from different classes can be combined to form new random images. These can create good amount of generalization in images for the model.

Using Cutmix augmentation is another good technique for this. Portion of the image containing the green box can be cut from the image and can be replaced with a similar size image segment from another image. Thus, enough samples can be generated where the original images with green boxes not having it after augmentation and the ones without green boxes having it. Hence the model would not learn green box as a property of any specific class.

Machine Learning

Construction of the dataset

Kvasir dataset features contained 8000 files belonging to 8 different classes. Each class had 1000 files. Each file contained 6 lines of information. A feature name and comma separated feature values. The 6 feature names where TCD, Tamura, ColorLayout, EdgeHistogram, AutoColorCorrelogram, PHOG. Feature names were separated from the feature values by the symbol colon. All files had feature names and values in same order.

To construct the dataset for modelling purpose, first 6 separate data frames were created, one for each feature(Ruwaa, 2020). Files were read from each class one by one. In each file, line by line reading of data was done and feature values in each line was converted to a pandas series object and the name of the series was given as the class value. This series was appended to its associated data frame. The separation of feature name and value was done using split() with colon as parameter.

Example- If files from 'dyed-lifted-polyps' class was read first, then in each file the first line contained JCD feature name and its values. This was separated and the values were given to JCD dataframe with the index name as dyed-lifted-polyps. Then the second line contained Tamura feature name and its values. Feature value was appended to Tamura data frame with the index name as dyed-lifted-polyps. This was repeated for all files in all classes. This would result in the creation of 6 separate dataframes each having 8000 rows of feature values with class name as index.

```
feature_JCD_df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	158	159	160	161	162	163	164	165	166	167
dyed-lifted-polyps	0.5	2.0	4.5	0.0	0.0	0.0	3.5	3.5	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dyed-lifted-polyps	0.0	1.5	4.5	0.0	0.0	0.0	2.5	1.5	3.5	0.0	...	0.0	1.0	1.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0
dyed-lifted-polyps	1.0	2.0	6.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	...	0.0	2.0	1.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0
dyed-lifted-polyps	1.5	2.5	5.5	0.5	0.5	0.5	2.5	3.0	3.5	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
dyed-lifted-polyps	3.0	3.5	6.0	0.0	0.0	0.0	1.0	0.0	0.5	0.5	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 168 columns

Fig11: JCD feature dataframe containing JCD feature values from all 8 classes

Finally, all the 6 dataframes were concatenated column wise to form one final data frame of 8000 rows and 1185 columns. Now this dataframe contains information of all the 6 features for 8 different classes. The class names present in the index was converted to a column. This became the Y variable and the rest of the columns containing feature values became the X variable. This was given to the model as input data.

(8000, 1185)

	class	0	1	2	3	4	5	6	7	8	...	620	621	622	623	624	625	626	627	628	629
0	dyed-lifted-polyps	0.5	2.0	4.5	0.0	0.0	0.0	3.5	3.5	5.0	...	7.0	3.0	3.0	6.0	6.0	4.0	2.0	2.0	1.0	3.0
1	dyed-lifted-polyps	0.0	1.5	4.5	0.0	0.0	0.0	2.5	1.5	3.5	...	0.0	6.0	0.0	0.0	0.0	0.0	2.0	3.0	0.0	0.0
2	dyed-lifted-polyps	1.0	2.0	6.0	0.0	0.0	0.0	0.0	0.0	3.0	...	0.0	14.0	3.0	5.0	14.0	0.0	0.0	4.0	3.0	0.0
3	dyed-lifted-polyps	1.5	2.5	5.5	0.5	0.5	0.5	2.5	3.0	3.5	...	0.0	0.0	0.0	0.0	4.0	5.0	0.0	0.0	0.0	6.0
4	dyed-lifted-polyps	3.0	3.5	6.0	0.0	0.0	0.0	1.0	0.0	0.5	...	8.0	4.0	1.0	3.0	3.0	1.0	2.0	14.0	2.0	0.0

5 rows × 1186 columns

Fig12: Final dataframe for all the classes

6GF Random Forest

Random forest consists of large number of individual decision trees and works as an ensemble technique, that is, as a group of trees. Random forest can be used for both classification and regression techniques. In a classification setting, each individual tree in a random forest outputs a response class prediction and the class with the majority vote out of all the trees becomes the model's output. Each tree in a random forest classifier will be good in evaluating some aspect of the data set. When all the trees combine, a better result would be obtained. The reason why a random forest works really well because, all the trees inside the random forest is not too correlated with each other. If there was correlation, then error rate would increase, and it will decrease the performance of the model. This non correlation is achieved through the process of bagging.

In the paper of Kvasir :A Multiclass Image-Dataset for Computer Aided Gastrointestinal Disease Detection, to implement classical machine learning algorithm all the 6 features extracted from images were combined together forming a feature vector of 1186. A random forest model developed using this dataset is called a 6GF random forest. The model produced as accuracy of 93.3%

To implement this the combined data frame all 6 features for the 8 classes was splitted into x and y. 1185 features contained in x was the features to train the model and y variable contained the class names. The dataset was split into train and test set. 80% od the data was used to train the model and the rest 20% was used to test the model.

First a base random forest model was defined, and then hyper parameter tuning was performed using RandomizedSearchCV on it to improve the performance of the model(Koehrsen, 2018). RandomizedSearchCV was used instead of GridSearchCV as it uses a fixed number of parameter settings from all the parameter distributions Given. On the other hand, GridSearchCV uses all the parameters to find the optimum ones. Since the training time was very high, RandomizedSearchCV was preferred. A cross validated search is done on all the parameter settings. The parameters specified to tune were:

- n_estimators : Number of tress in random forest
- max_features : Number of features to be considered at every split
- max_depth : maximum number of levels in a tree

- min_samples_split : minimum number of samples required to split a node in a tree
- min_samples_leaf : minimum number of samples required at each leaf node
- bootstrap: Sampling with replacement required or not

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 3
0, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True,
False]}
```

Fig13: Parameters used to train the model to find the best fit

The search was done with 3-fold cross validation and with 100 iterations.

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
{'n_estimators': 1800, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 80, 'bootstrap': Fal
se}
```

Fig 14: Best parameters found after RandomizedSearchCV

Results

Random forest model produced an accuracy of 76.12% initially. After hyper parameter tuning the model produced a test accuracy of 77.5%. The tuning increased accuracy by 1.5%. The Kvasir paper produced a 93.3% accuracy with 6GF random forest model. Compared to that the accuracy of the current model is very low. Accuracy metric depicts how many were correctly predicted by the model out of the total predictions.

Accuracy of the model is given by $TP+TN$ / ($TP+TN+FP+FN$)

But another observation is that the precision and recall of the Kvasir 6GF random forest model was 73.2%, the current model has a precision of 77% and a recall of 78% which is an improvement. Recall tells us how many actual positive classes were there in the entire testing data and out of that how many were predicted correctly. Precision tells how much good the model is in predictions. Precision talks only about predicted classes.

Recall/Sensitivity= TP/ ($TP+F$N)

Precision= TP / ($TP+FP$)

This points to the fact that the number of True Negatives is very less in the current model as compared to the Kvasir model, since Kvasir model has very high specificity.

	precision	recall	f1-score	support
dyed-lifted-polyps	0.68	0.72	0.70	200
dyed-resection-margins	0.72	0.69	0.70	200
esophagitis	0.77	0.68	0.72	200
normal-cecum	0.84	0.92	0.88	200
normal-pylorus	0.91	0.98	0.94	200
normal-z-line	0.71	0.80	0.75	200
polyps	0.78	0.68	0.72	200
ulcerative-colitis	0.78	0.74	0.76	200
accuracy			0.78	1600
macro avg	0.77	0.78	0.77	1600
weighted avg	0.77	0.78	0.77	1600

Fig 15: Classification Report of the model

Classification report give in fig 15 shows that there were a total of 1600 images available in testing dataset. 200 images belonging to each class. It is observed that the accuracy, precision, recall and f1-score of the model in classifying 'normal-pylorus' and 'normal-cecum' is very high as compared to the rest of the classes. Overall, the precision of the model has improved from Kvasir 6GF model which is very important in medical image classification.

Bibliography

Albawi, S., Mohammed, T.A., Al-Zawi, S., 2017. Understanding of a convolutional neural network, in: 2017 International Conference on Engineering and Technology (ICET). Presented at the 2017 International Conference on Engineering and Technology (ICET), IEEE, Antalya, pp. 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

Al-Saffar, A.A.M., Tao, H., Talab, M.A., 2017. Review of deep convolution neural network in image classification, in: 2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET). Presented at the 2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET), IEEE, Jakarta, pp. 26–31. https://doi.org/10.1109/ICRAMET.2017.8253139

Arora, P., Kapse, V.M., Sinha, S., Gera, S., 2021. An analytical study for Pneumonia Detection towards building an intelligent system using Image Data Generator, in: 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). Presented at the 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 1–4. https://doi.org/10.1109/ICRITO51393.2021.9596434

DeVries, T., Taylor, G.W., 2017. Improved Regularization of Convolutional Neural Networks with Cutout. arXiv:1708.04552 [cs].

Gulli, A., Pal, S., 2017. Deep Learning with Keras. Packt Publishing Ltd.

Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., Inman, D.J., 2021. 1D convolutional neural networks and applications: A survey. Mechanical Systems and Signal Processing 151, 107398. https://doi.org/10.1016/j.ymssp.2020.107398

Koehrsen, W., 2018. Hyperparameter Tuning the Random Forest in Python [WWW Document]. Medium. URL https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74 (accessed 5.9.22).

Mixup | DLology [WWW Document], n.d. URL https://www.dlology.com/blog/how-to-do-mixup-training-from-image-files-in-keras/ (accessed 5.9.22).

Multi-Channel Convolutions explained| Medium [WWW Document], n.d. URL https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108 (accessed 5.9.22).

nbro, 2020. 3D Convolution. Artificial Intelligence Stack Exchange.

O'Shea, K., Nash, R., 2015. An Introduction to Convolutional Neural Networks. arXiv:1511.08458 [cs].

Renard, F., Guedria, S., Palma, N.D., Vuillerme, N., 2020. Variability and reproducibility in deep learning for medical image segmentation. Sci Rep 10, 13724. https://doi.org/10.1038/s41598-020-69920-0

Ruwaa, 2020. ML Classification Kvasir.

Social Network for Programmers and Developers [WWW Document], n.d. URL https://morioh.com (accessed 5.9.22).

Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y., 2019. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. Presented at the Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 6023–6032.