National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Paras Jain
Student ID: 18182119

School of Computing
National College of Ireland

Supervisor: Mr. Hicham Rifai

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Paras Jain		
Student ID:	18182119		
Programme:	MSc DATA ANALYTICS	**Year:**	2019-2020
Module:	MSc RESEARCH PROJECT		
Lecturer:	Mr. Hicham Rifai		
Submission Due Date:	17/08/2020		
Project Title:	Liver Disease Detection from CT scan images using Deep Learning and Transfer Learning		
Word Count:	**697**	**Page Count:**	**14**

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Paras Jain
18182119

1 Introduction

This configuration manual details the system setup, software and hardware specifications and the step taken to carry out the Research Project: Liver Disease Detection from CT scan images using Deep Learning and Transfer Learning.

In this manual, Section 2 contains information on software and hardware specifications. Section 3 discusses the environment setup, data collection and preparation, importing of libraries and mounting of Google Drive. Section4 explains the various steps involved in image processing. Section 5 describes the design of the models and show their implementation.

2 System Configuration

This section of the configuration manual provides information about the system configuration utilised to implement the project.

2.1 Hardware Requirements

Operating System	Windows 10
RAM	12.72 GB (Google Colaboratory)
Disk Space	107.77 GB (Google Colaboratory)
Storage Space	15 GB (Google Drive)

Table 1 Hardware Configuration

2.2 Software Requirements

Programming Language Tools	Google Colaboratory (Cloud-based Jupyter notebook environment), Python 3.6.9
Web Browser	Google Chrome or Mozilla Firefox
Email Account	Gmail account for Google Drive and Colaboraory
Other software	Microsoft Word and Microsoft Excel

Table 2 Software Configuration

3 Project Development

This section of the configuration manual provides information about the setup of environment and collection and usage of data.

3.1 Google Colaboratory Environment Setup

Google Colaboratory[1] environment is used for the research implementation. It provides Python version 3.6.9 along with powerful RAM and GPU support and is hosted on Google Cloud. Gmail account is essential to use Google Colab as it requires authentication.

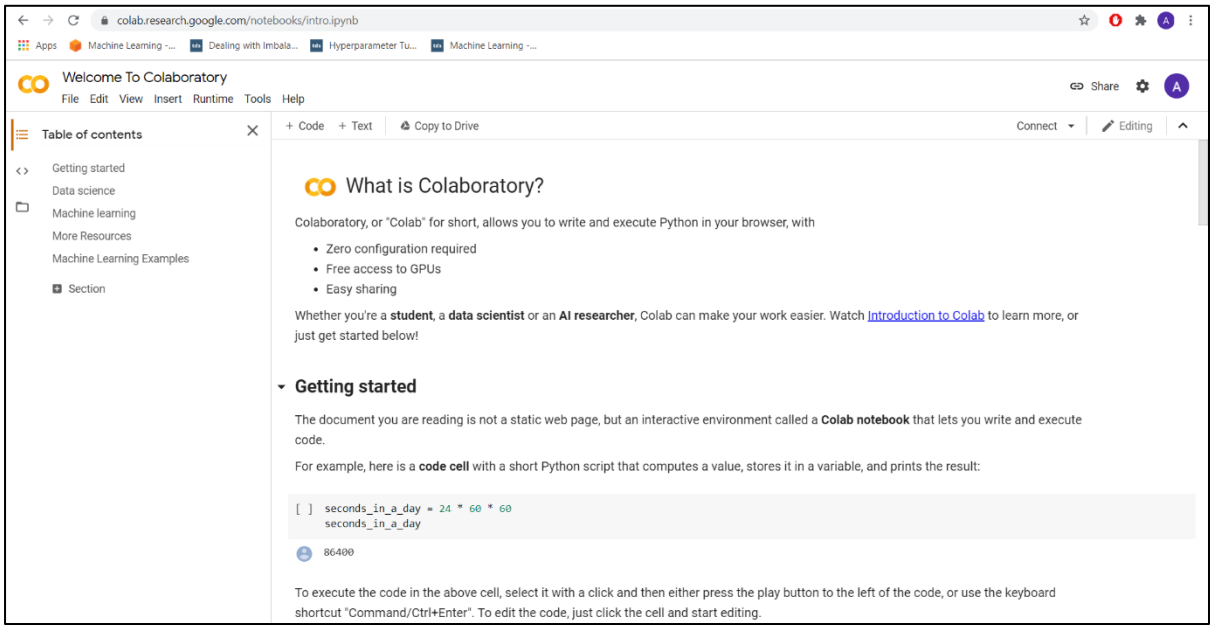

Figure 1 Google Colaboratory Homepage

3.2 Data Collection

The data is collected from Medpix[2] which provides an online collection of medical images for research purposes. It is publicly available and an ethical data source. Image data is gathered in the form of '.jpg' files for the two categories.

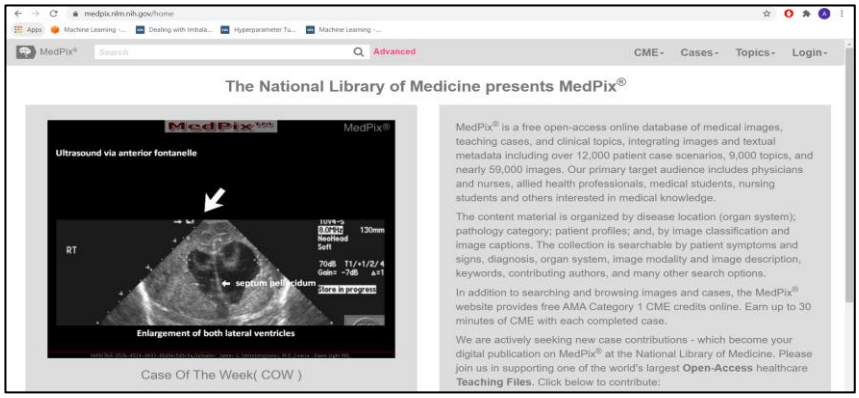

Figure 2 Data Source - Medpix

3.3 Data Preparation

This section describes the approach used for preparing data. The data is divided into 3 folders on Google Drive namely, Training, Validation, and Testing.

[1] https://colab.research.google.com/notebooks/
[2] https://medpix.nlm.nih.gov/

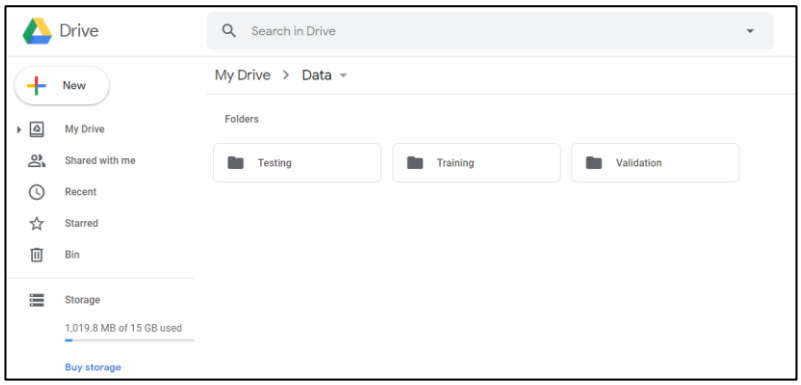

Figure 3 Google Drive Data

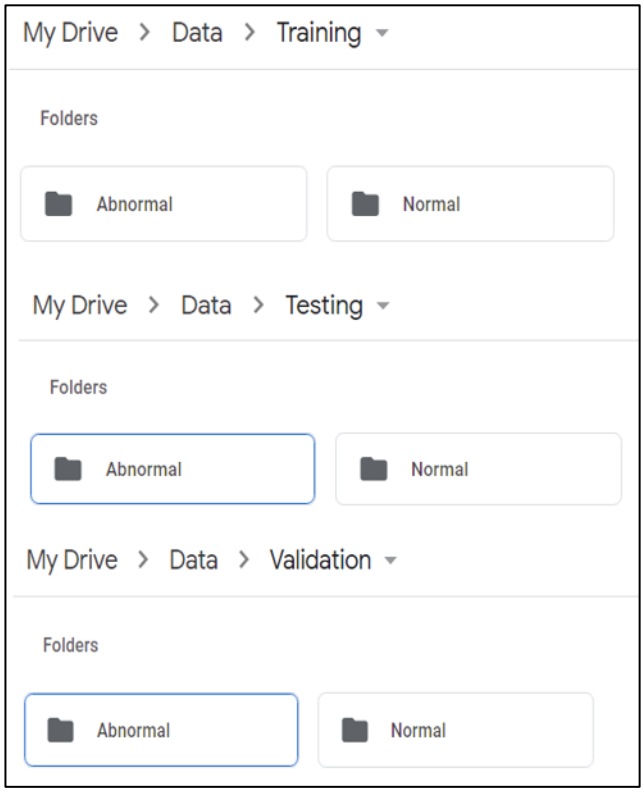

Figure 4 Train, Valid and Test folders

3.4 Importing libraries

This section shows the libraries that are needed for the implementation of the research project. The libraries available in Colab are imported and unavailable libraries are installed using pip command. OpenCV[3], Tensorflow[4] and Keras[5] are the primarily used libraries in the research.

[3] https://pypi.org/project/opencv-python/
[4] https://www.tensorflow.org/
[5] https://keras.io/

```
+ Code  + Text

[ ]  import os
     import sys
     import cv2
     import shutil
     import random
     import warnings
     import numpy as np
     import pandas as pd
     import seaborn as sns
     import multiprocessing as mp
     import matplotlib.pyplot as plt

     from keras.activations import elu

     from sklearn.utils import class_weight
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import confusion_matrix, cohen_kappa_score
     from keras import backend as K
     from keras.models import Model
     from keras.utils import to_categorical
     from keras import optimizers, applications
     from keras.layers import Dense, Dropout, GlobalAveragePooling2D, Input
     from keras.callbacks import EarlyStopping, ReduceLROnPlateau, Callback, LearningRateScheduler
     from keras.models import Sequential

     import tensorflow as tf
     from tensorflow import keras
     from keras.models import Sequential
     from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
     from tensorflow.keras import layers
     from keras.utils import to_categorical
     import numpy as np
     import matplotlib.pyplot as plt
     plt.style.use('fivethirtyeight')


     from tensorflow.keras.callbacks import EarlyStopping

     from sklearn.metrics import classification_report
```

Figure 5 Importing Libraries

3.5 Mounting Google Drive

Google Drive needs to be mounted in Google Colab to use the data. It requires authentication through the Gmail account used for Colab.

```
[ ]  from google.colab import drive
     drive.mount('/content/drive/')

     Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%2

     Enter your authorization code:
     ··········
     Mounted at /content/drive/
```

Figure 6 Mounting Google Drive on Colab Notebook

4 Image Processing

This section provides knowledge about different steps involved in the processing of images.

4.1 Image Pre-processing

There are various image processing techniques used in the research such as Resize, Histogram Equalisation, Noise removal, etc.

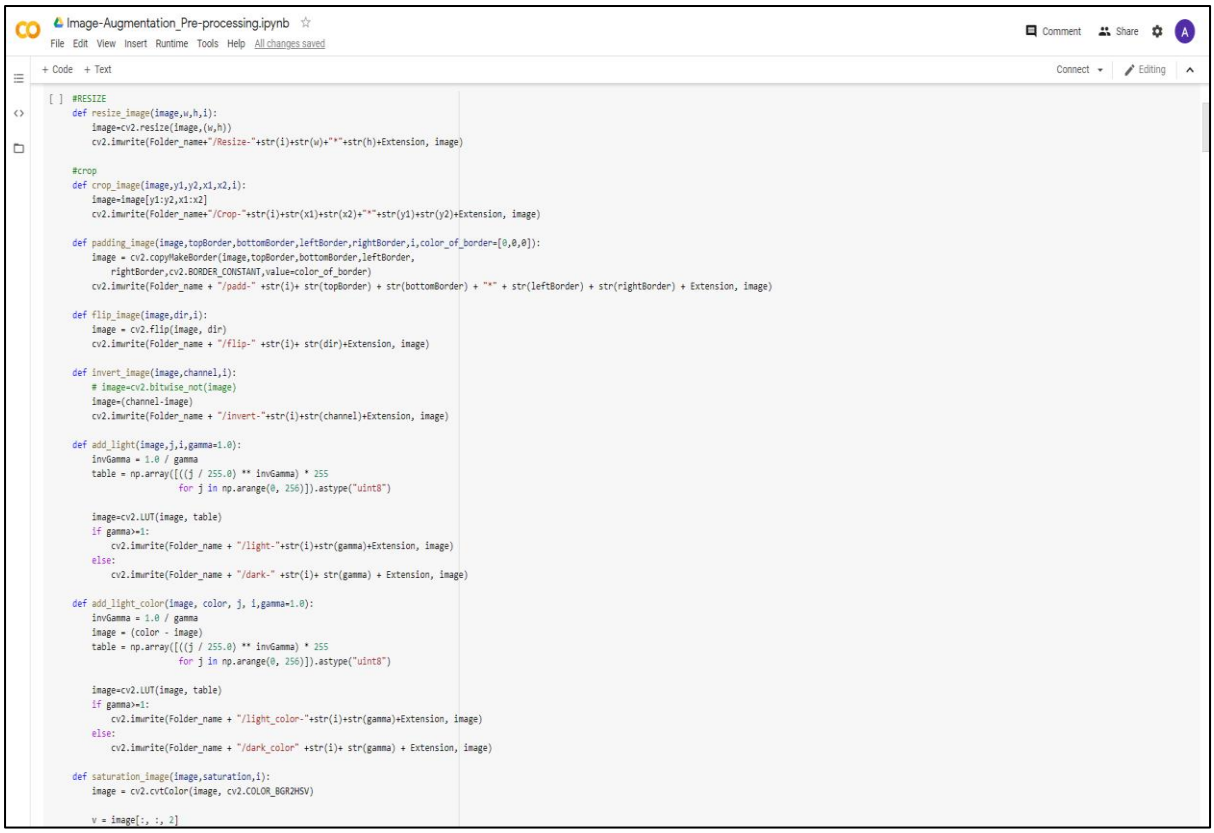

Figure 7 Image Pre-processing

```
[ ]  #histogram equalisation

     img = cv2.imread(image,0)

     hist,bins = np.histogram(img.flatten(),256,[0,256])

     cdf = hist.cumsum()
     cdf_normalized = cdf * hist.max()/ cdf.max()

     plt.plot(cdf_normalized, color = 'b')
     plt.hist(img.flatten(),256,[0,256], color = 'r')
     plt.xlim([0,256])
     plt.legend(('cdf','histogram'), loc = 'upper left')
     plt.show()
```

```
[ ]  cdf_m = np.ma.masked_equal(cdf,0)
     cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
     cdf = np.ma.filled(cdf_m,0).astype('uint8')
```

```
[ ]  img2 = cdf[img]
```

```
[ ]  #histogram equalisation
     img = cv2.imread(image,0)
     equ = cv2.equalizeHist(img)
     res = np.hstack((img,equ)) #stacking images side-by-side
```

```
     img = cv2.imread(image,0)

     # create a CLAHE object (Arguments are optional).
     clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
     cl1 = clahe.apply(img)
```

Figure 8 Image Pre-processing (contd.)

4.2 Image Augmentation

Image augmentation is done to increase the size of the dataset. Various techniques such as flip, rotate, gaussian blur, morphological gradient, etc are used.

```
for i,image in enumerate(abnormal_images):
  flip_image(image,-1,i=i)#both
  hue_image(image,50,i=i)
  gausian_blur(image,0.25,i=i)
  morphological_gradient_image(image,5,i=i)
  adaptive_gaussian_noise(image,i=i)
  transformation_image(image,i=i)
```

Figure 9 Image Augmentation

```
[ ] print("Abnormal Images after Augmentation",len(abnormal_images))
    print("Normal Images after Augmentation",len(normal_images))

    Abnormal Images after Augmentation 1520
    Normal Images after Augmentation 356
```

Figure 10 Augmentation Results

4.3 Image Segmentation

Watershed Image segmentation is implemented to segment the required parts from the entire image.

```
#thresholding
img = cv2.imread(image)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

```
[ ]  # noise removal
     kernel = np.ones((3,3),np.uint8)
     opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

     # sure background area
     sure_bg = cv2.dilate(opening,kernel,iterations=3)

     # Finding sure foreground area
     dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
     ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)

     # Finding unknown region
     sure_fg = np.uint8(sure_fg)
     unknown = cv2.subtract(sure_bg,sure_fg)
```

```
[ ]  # Marker labelling
     ret, markers = cv2.connectedComponents(sure_fg)

     # Add one to all labels so that sure background is not 0, but 1
     markers = markers+1

     # Now, mark the region of unknown with zero
     markers[unknown==255] = 0
```

```
[ ]  markers = cv2.watershed(img,markers) #Segmentation
     img[markers == -1] = [255,0,0]
```

Figure 11 Watershed Transform for Image Segmentation

5 Modelling

```
[ ]  base_dir = '/content/drive/My Drive/Data'

     #Train, Validation & Testing Directory
     train_dir = os.path.join(base_dir, 'Training')
     validation_dir = os.path.join(base_dir, 'Validation')
     test_dir = os.path.join(base_dir, 'Testing')

     #Train Data
     train_abnormal_dir = os.path.join(train_dir, 'Abnormal')
     train_normal_dir = os.path.join(train_dir, 'Normal')

     #Validation Data
     validation_abnormal_dir = os.path.join(validation_dir, 'Abnormal')
     validation_normal_dir = os.path.join(validation_dir, 'Normal')

     #Test Data
     test_abnormal_dir = os.path.join(test_dir, 'Abnormal')
     test_normal_dir = os.path.join(test_dir, 'Normal')
```

```
[ ]  print("Total training abnormal images:", len(os.listdir(train_abnormal_dir)))
     print("Total training normal images:", len(os.listdir(train_normal_dir)))

     print("Total validation abnormal images:", len(os.listdir(validation_abnormal_dir)))
     print("Total validation normal images:", len(os.listdir(validation_normal_dir)))

     print("Total testing abnormal images:", len(os.listdir(test_abnormal_dir)))
     print("Total testing normal images:", len(os.listdir(test_normal_dir)))
```

```
Total training abnormal images: 970
Total training normal images: 230
Total validation abnormal images: 246
Total validation normal images: 54
Total testing abnormal images: 304
Total testing normal images: 72
```

Figure 12 Data loaded into Directories

```
[ ]  train_generator = datagen.flow_from_directory(train_dir, batch_size=32, class_mode='categorical', target_size=(299,299))

     valid_generator = datagen.flow_from_directory(validation_dir, batch_size=32, class_mode='categorical', target_size=(299,299))

     test_generator = datagen.flow_from_directory(test_dir, batch_size=1, class_mode=None, target_size=(299,299))
```

```
Found 1200 images belonging to 2 classes.
Found 300 images belonging to 2 classes.
Found 376 images belonging to 2 classes.
```

Figure 13 Image Data Generators

5.1 Support Vector Machine (SVM)

The implementation of SVM with Grid Search Cross Validation is illustrated.

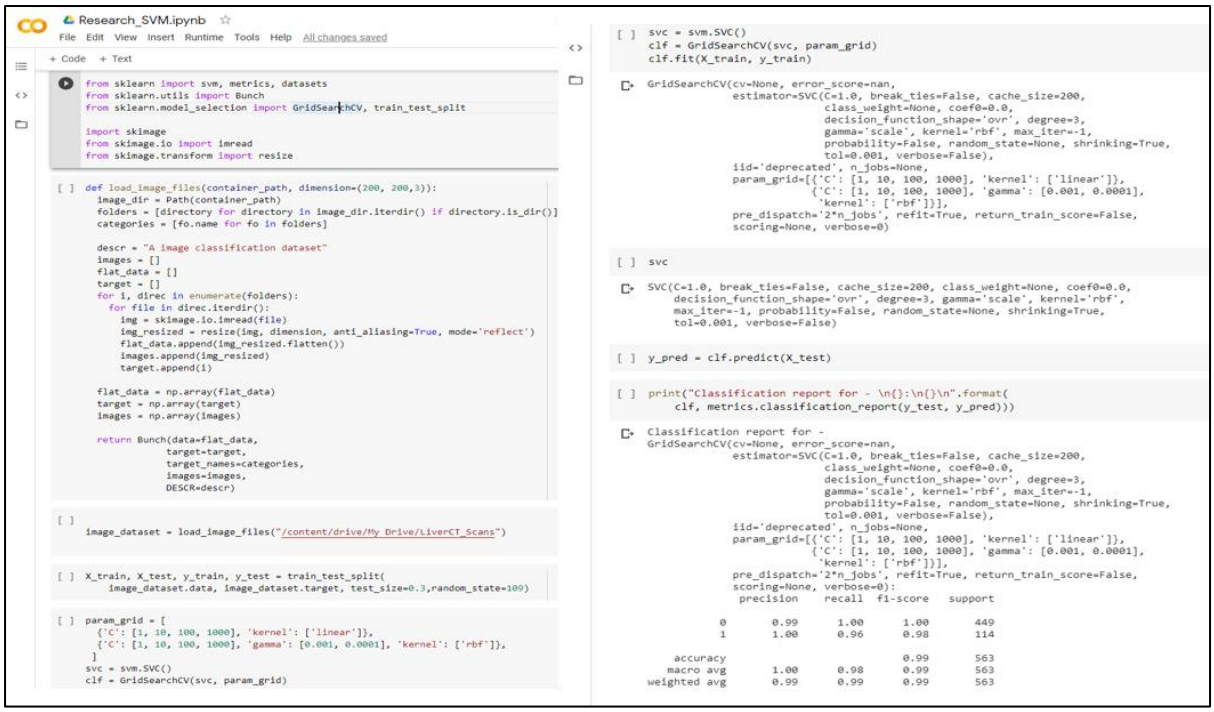

Figure 14 Implementation of SVM

5.2 Convolution Neural Network (CNN)

The implementation of CNN built from scratch is illustrated. Early Stopping and Reduce Learning Rate on Plateau features are used as well.

```
[ ]  model = Sequential()

     model.add(Conv2D(16, (3,3), activation='relu', input_shape=(299, 299, 3)))
     model.add(MaxPooling2D(2,2))
     model.add(Conv2D(32, (3,3), activation='relu'))
     model.add(MaxPooling2D(2,2))
     model.add(Conv2D(64, (3,3), activation='relu'))
     model.add(MaxPooling2D(2,2))
     model.add(Flatten())
     model.add(Dense(512, activation='relu',kernel_regularizer=regularizers.l2(0.01)))
     model.add(Dense(2, activation='sigmoid'))
```

Figure 15 Building CNN Model

```
#EPOCHS = 40
WARMUP_EPOCHS = 2
LEARNING_RATE = 1e-4
WARMUP_LEARNING_RATE = 1e-3
ES_PATIENCE = 2
RLROP_PATIENCE = 3
DECAY_DROP = 0.5
```

```
es = EarlyStopping(monitor='val_accuracy', patience=ES_PATIENCE, restore_best_weights=True, verbose=1)
rlrop = ReduceLROnPlateau(monitor='val_loss', mode='min', patience=RLROP_PATIENCE, factor=DECAY_DROP, min_lr=1e-6, verbose=1)

callback_list = [es, rlrop]


callback_list = [es, rlrop]
optimizer = optimizers.Adam(lr=LEARNING_RATE)
model.compile(optimizer=optimizer, loss="binary_crossentropy",  metrics=['accuracy'])
model.summary()

model.summary()
```

Figure 16 Compiling CNN Model

```
[ ] STEP_SIZE_TRAIN = train_generator.n//train_generator.batch_size
    STEP_SIZE_VALID = valid_generator.n//valid_generator.batch_size
    print(STEP_SIZE_TRAIN,STEP_SIZE_VALID)

    37 9

[ ] history_finetunning = model.fit(train_generator,
                                    steps_per_epoch=STEP_SIZE_TRAIN,
                                    validation_data=valid_generator,
                                    validation_steps=STEP_SIZE_VALID,
                                    epochs=10,
                                    callbacks=callback_list,
                                    verbose=1)

    Epoch 1/10
    37/37 [==============================] - 328s 9s/step - loss: 6.6127 - accuracy: 0.8780 - val_loss: 3.4166 - val_accuracy: 0.9340
    Epoch 2/10
    37/37 [==============================] - 125s 3s/step - loss: 2.0570 - accuracy: 0.9563 - val_loss: 1.1467 - val_accuracy: 0.9478
    Epoch 3/10
    37/37 [==============================] - 126s 3s/step - loss: 0.7865 - accuracy: 0.9700 - val_loss: 0.5006 - val_accuracy: 0.9813
    Epoch 4/10
    37/37 [==============================] - 128s 3s/step - loss: 0.4331 - accuracy: 0.9812 - val_loss: 0.4572 - val_accuracy: 0.9739
    Epoch 5/10
    37/37 [==============================] - 127s 3s/step - loss: 0.3304 - accuracy: 0.9696 - val_loss: 0.2661 - val_accuracy: 0.9869
    Epoch 6/10
    37/37 [==============================] - 123s 3s/step - loss: 0.2444 - accuracy: 0.9876 - val_loss: 0.2101 - val_accuracy: 0.9832
    Epoch 7/10
    37/37 [==============================] - 125s 3s/step - loss: 0.1924 - accuracy: 0.9882 - val_loss: 0.1488 - val_accuracy: 0.9888
    Epoch 8/10
    37/37 [==============================] - 124s 3s/step - loss: 0.1827 - accuracy: 0.9805 - val_loss: 0.1503 - val_accuracy: 0.9832
    Epoch 9/10
    37/37 [==============================] - 127s 3s/step - loss: 0.1540 - accuracy: 0.9872 - val_loss: 0.1250 - val_accuracy: 0.9944
    Epoch 10/10
    37/37 [==============================] - 124s 3s/step - loss: 0.1335 - accuracy: 0.9880 - val_loss: 0.1643 - val_accuracy: 0.9795
```

Figure 17 CNN Model Fitting

5.3 Inception-v4

The implementation of Inception-v4 based on transfer learning is illustrated. The pre-trained weights used are “inception-v4_weights_tf_dim_ordering_tf_kernels_notop.h5” [6].

[6] https://github.com/kentsommer/keras-inceptionV4/releases

```
WEIGHTS_PATH = 'https://github.com/kentsommer/keras-inceptionV4/releases/download/2.1/inception-v4_weights_tf_dim_ordering_tf_kernels.h5'
WEIGHTS_PATH_NO_TOP = 'https://github.com/kentsommer/keras-inceptionV4/releases/download/2.1/inception-v4_weights_tf_dim_ordering_tf_kernels_notop.h5'
```

```
[ ] def preprocess_input(x):
        x = np.divide(x, 255.0)
        x = np.subtract(x, 0.5)
        x = np.multiply(x, 2.0)
        return x


    def conv2d_bn(x, nb_filter, num_row, num_col,
                  padding='same', strides=(1, 1), use_bias=False):
        """
        Utility function to apply conv + BN.
        (Slightly modified from https://github.com/fchollet/keras/blob/master/keras/applications/inception_v3.py)
        """
        if K.image_data_format() == 'channels_first':
            channel_axis = 1
        else:
            channel_axis = -1
        x = Convolution2D(nb_filter, (num_row, num_col),
                          strides=strides,
                          padding=padding,
                          use_bias=use_bias,
                          kernel_regularizer=regularizers.l2(0.00004),
                          kernel_initializer=initializers.VarianceScaling(scale=2.0, mode='fan_in', distribution='normal', seed=None))(x)
        x = BatchNormalization(axis=channel_axis, momentum=0.9997, scale=False)(x)
        x = Activation('relu')(x)
        return x


    def block_inception_a(input):
        if K.image_data_format() == 'channels_first':
            channel_axis = 1
        else:
            channel_axis = -1

        branch_0 = conv2d_bn(input, 96, 1, 1)

        branch_1 = conv2d_bn(input, 64, 1, 1)
        branch_1 = conv2d_bn(branch_1, 96, 3, 3)

        branch_2 = conv2d_bn(input, 64, 1, 1)
        branch_2 = conv2d_bn(branch_2, 96, 3, 3)
        branch_2 = conv2d_bn(branch_2, 96, 3, 3)
```

Figure 18 Building Inception-v4 model

```
[ ] def create_model(num_classes=1001, dropout_prob=0.2, weights=None, include_top=True):
        return inception_v4(num_classes, dropout_prob, weights, include_top)
```

```
[ ] incept_model = create_model(num_classes=1001, dropout_prob=0.2, weights=None, include_top=False)
    incept_model.load_weights('/content/drive/My Drive/inception-v4_weights_tf_dim_ordering_tf_kernels_notop.h5')
```

```
[ ] for l in incept_model.layers:
        if l is not None: l.trainable = True

    x = incept_model.output
    x = GlobalAveragePooling2D(data_format='channels_last')(x)
    x = BatchNormalization()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(2048, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(2, activation='softmax')(x)

    model = Model(inputs=incept_model.input, outputs=predictions)
    model.summary()
```

Figure 19 Compiling Inception-v4 model

```
[ ] history_finetunning = model.fit(train_generator,
                                    steps_per_epoch=STEP_SIZE_TRAIN,
                                    validation_data=valid_generator,
                                    validation_steps=STEP_SIZE_VALID,
                                    epochs=10,
                                    callbacks=callback_list,
                                    verbose=1)

Epoch 1/10
37/37 [==============================] - 2153s 58s/step - loss: 0.9451 - accuracy: 0.9187 - val_loss: 1.4155 - val_accuracy: 0.8160
Epoch 2/10
37/37 [==============================] - 2148s 58s/step - loss: 0.7703 - accuracy: 0.9923 - val_loss: 1.3201 - val_accuracy: 0.8160
Epoch 3/10
37/37 [==============================] - ETA: 0s - loss: 0.7540 - accuracy: 0.9983 Restoring model weights from the end of the best epoch.
37/37 [==============================] - 2148s 58s/step - loss: 0.7540 - accuracy: 0.9983 - val_loss: 1.6690 - val_accuracy: 0.8160
Epoch 00003: early stopping
```

Figure 20 Inception-v4 Model Fitting

5.4 DenseNet-169

The implementation of DenseNet-169 based on transfer learning is illustrated. The pre-trained weights used are "DenseNet-BC-169-32-no-top.h5" [7].

```
from keras.applications import DenseNet169
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, Flatten, BatchNormalization, GlobalAveragePooling2D
from keras.optimizers import Adam

[ ] densenet = DenseNet169(include_top=False,weights='/content/drive/My Drive/80814_187731_compressed_DenseNet-BC-169-32-no-top.h5/DenseNet-BC-169-32-no-top.h5',input_shape=(299,299,3))

[ ] def build_model():
        model = Sequential()
        model.add(densenet)
        model.add(GlobalAveragePooling2D())
        model.add(Dropout(0.5))
        model.add(Dense(2, activation='sigmoid'))

        model.compile(
            loss='binary_crossentropy',
            optimizer=Adam(lr=0.000005),
            metrics=['accuracy']
        )
```

Figure 21 Building DenseNet-169 Model

```
[ ] history_finetunning = model.fit(train_generator,
                                    steps_per_epoch=STEP_SIZE_TRAIN,
                                    validation_data=valid_generator,
                                    validation_steps=STEP_SIZE_VALID,
                                    epochs=10,
                                    callbacks=callback_list,
                                    verbose=1)

Epoch 1/10
37/37 [==============================] - 1762s 48s/step - loss: 0.5539 - accuracy: 0.8031 - val_loss: 0.4837 - val_accuracy: 0.9306
Epoch 2/10
37/37 [==============================] - 1682s 45s/step - loss: 0.3627 - accuracy: 0.9426 - val_loss: 0.3376 - val_accuracy: 0.9757
Epoch 3/10
37/37 [==============================] - 1682s 45s/step - loss: 0.2750 - accuracy: 0.9666 - val_loss: 0.2415 - val_accuracy: 0.9792
Epoch 4/10
37/37 [==============================] - 1680s 45s/step - loss: 0.1919 - accuracy: 0.9846 - val_loss: 0.1880 - val_accuracy: 0.9826
Epoch 5/10
37/37 [==============================] - 1686s 46s/step - loss: 0.1421 - accuracy: 0.9863 - val_loss: 0.1444 - val_accuracy: 0.9826
Epoch 6/10
37/37 [==============================] - 1670s 45s/step - loss: 0.1185 - accuracy: 0.9897 - val_loss: 0.1131 - val_accuracy: 0.9896
Epoch 7/10
37/37 [==============================] - 1672s 45s/step - loss: 0.0952 - accuracy: 0.9940 - val_loss: 0.0961 - val_accuracy: 0.9896
Epoch 8/10
37/37 [==============================] - ETA: 0s - loss: 0.0821 - accuracy: 0.9966 Restoring model weights from the end of the best epoch.
37/37 [==============================] - 1674s 45s/step - loss: 0.0821 - accuracy: 0.9966 - val_loss: 0.0751 - val_accuracy: 0.9896
Epoch 00008: early stopping
```

Figure 22 DenseNet-169 Model Fitting

5.5 Cross-Validation

[7] https://www.kaggle.com/xhlulu/densenet-keras?select=DenseNet-BC-169-32-no-top.h5

The implementation of K-Fold Cross-Validation is illustrated. It is used to check the variance of the models and avoid the overfitting.

```
#K-fold Cross Validation

VALIDATION_ACCURACY = []
VALIDAITON_LOSS = []

save_dir = '/saved_models/'
fold_var = 1

for train_index, val_index in kf.split(np.zeros(n),Y):
  training_data = train_data.iloc[train_index]
  validation_data = train_data.iloc[val_index]

  # CREATE NEW MODEL
  model = create_new_model()
  # COMPILE NEW MODEL
  model.compile(loss='categorical_crossentropy',
          optimizer=opt,
          metrics=['accuracy'])

  # CREATE CALLBACKS
  checkpoint = tf.keras.callbacks.ModelCheckpoint(save_dir+get_model_name(fold_var),
              monitor='val_accuracy', verbose=1,
              save_best_only=True, mode='max')
  callbacks_list = [checkpoint]
  # There can be other callbacks, but just showing one because it involves the model name
  # This saves the best model
  # FIT THE MODEL
  history = model.fit(train_data_generator,
          epochs=num_epochs,
          callbacks=callbacks_list,
          validation_data=valid_data_generator)

  # LOAD BEST MODEL to evaluate the performance of the model
  model.load_weights("/saved_models/model_"+str(fold_var)+".h5")

  results = model.evaluate(valid_data_generator)
  results = dict(zip(model.metrics_names,results))

  VALIDATION_ACCURACY.append(results['accuracy'])
  VALIDATION_LOSS.append(results['loss'])

  tf.keras.backend.clear_session()

  fold_var += 1
```

Figure 23 K-Fold Cross Validation