B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,

Bangalore-560 019

Department of INFORMATION Science & Engineering

Course – Deep Learning

Course Code – 20IS6PEDLG

AY 2020-21

Initial Report on Seminar Project

DIABETES PREDICTION

Submitted to

RASHMI R (Assistant Professor)

Submitted by

PRAMILA DALAVAI	NIKHIL S K	Samarth M Hathwar
1BM18IS068	1BM18IS058	1BM18IS088

Contents

Chapter 1

Introduction

The rise of artificial intelligence in recent years is grounded in the success of deep learning. Three major drivers caused the breakthrough of (deep) neural networks: the availability of huge amounts of training data, powerful computational infrastructure, and advances in academia. Thereby deep learning systems start to outperform not only classical methods, but also human benchmarks in various tasks like image classification or face recognition. This creates the potential for many disruptive new businesses leveraging deep learning to solve real-world problems.

Diabetes if untreated may turn into fatal and directly or indirectly invites lot of other diseases such as heart attack, heart failure, brain stroke and many more. Therefore, early detection of diabetes is very significant so that timely action can be taken and the progression of the disease may be prevented to avoid further complications. Healthcare organizations accumulate huge amount of data including Electronic health records, images, omics data, and text but gaining knowledge and insight into the data remains a key challenge. The latest advances in Deep learning technologies can be applied for obtaining hidden patterns, which may diagnose diabetes at an early phase.

The main intent of this research work is to propose the development of a prognostic tool for early diabetes prediction and detection with improved accuracy. There have been an extensive amount of data and datasets available on the internet or external sources and the PIMA dataset which has been used in this work is one of the most widely used dataset in many researches and it is collected by the National Institute of Diabetes and Digestive and Kidney Diseases (NIDDK). This project presents a methodology for diabetes prediction using a diverse machine learning algorithm using the PIMA dataset.

Chapter 2

Data Collection And Analysis

There are several studies found in the literature that have used various techniques on the Pima Indian Diabetes dataset to train and test data. This paper focuses only on the data mining techniques used for classification on the same dataset. The National Institute of Diabetes and Digestive and Kidney Diseases of the NIH originally owned the Pima Indian Diabetes Database (PIDD).

The database has around 30000+ patients each with 9 numeric variables. The data refers to females of ages from 21 to 81. Out of the nine condition attributes, six attributes describe the result of physical examination, rest of the attributes are of chemical examinations. The independent or target variable is the class variable (diabetes = 1 (yes), diabetes =0 (no)), represented by the 9 th variable.

Dataset description

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective is to predict based on diagnostic measurements whether a patient has diabetes.

Fields description follow:

preg = Number of times pregnant

plas = Plasma glucose concentration a 2 hours in an oral glucose tolerance test

pres = Diastolic blood pressure (mm Hg)

skin = Triceps skin fold thickness (mm)

test = 2-Hour serum insulin (mu U/ml)

mass = Body mass index (weight in kg/(height in m)^2)

pedi = Diabetes pedigree function

age = Age (years)

class = Class variable (1:tested positive for diabetes, 0: tested negative for diabetes)

Chapter 3

Data Preprocessing

• In this Keras tutorial, we are going to use the Pima Indians onset of diabetes dataset. This is a standard machine learning dataset from the UCI Machine Learning repository. It describes patient medical record data for Pima Indians and whether they had an onset of diabetes within five years.

• Once the CSV file is loaded into memory, we can split the columns of data into input and output variables.

• The data will be stored in a 2D array where the first dimension is rows and the second dimension is columns, e.g. [rows, columns].

• We can split the array into two arrays by selecting subsets of columns using the standard NumPy slice operator or ":" We can select the first 8 columns from index 0 to index 7 via the slice 0:8. We can then select the output column (the 9th variable) via index 8.

Data Preprocessing

```
[5] import numpy as np # linear algebra
    import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
    from sklearn.preprocessing import StandardScaler

    import os

    data = pd.read_csv("diabetes (1).csv")
    array = data.values
    X = array[:,0:8]
    Y = array[:,8]
    print(X)
```

```
[[  6.    148.     72.    ...  33.6     0.627  50.    ]
 [  1.     85.     66.    ...  26.6     0.351  31.    ]
 [  8.    183.     64.    ...  23.3     0.672  32.    ]
 ...
 [  5.    121.     72.    ...  26.2     0.245  30.    ]
 [  1.    126.     60.    ...  30.1     0.349  47.    ]
 [  1.     93.     70.    ...  30.4     0.315  23.    ]]
```

```
[6] #StandardScaler
    transforms = StandardScaler()
    standardScale_X = transforms.fit_transform(X)
    print(standardScale_X)
```

```
[[ 0.63994726  0.84832379  0.14964075 ...  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 ... -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 ... -1.10325546  0.60439732
  -0.10558415]
 ...
 [ 0.3429808   0.00330087  0.14964075 ... -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 ... -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 ... -0.20212881 -0.47378505
  -0.87137393]]
```

Chapter 4

API'S used with flow diagram of model with description

1. Sequential Model

• We create a Sequential model and add layers one at a time until we are happy with our network architecture.

• The first thing to get right is to ensure the input layer has the right number of input features. This can be specified when creating the first layer with the **input_dim** argument and setting it to 8 for the 8 input variables.

• we will use a fully-connected network structure with three layers.

• Fully connected layers are defined using the Dense class. We can specify the number of neurons or nodes in the layer as the first argument, and specify the activation function using the **activation** argument.

• We will use the rectified linear unit activation function referred to as ReLU on the first two layers and the Sigmoid function in the output layer.

• Better performance is achieved using the ReLU activation function. We use a sigmoid on the output layer to ensure our network output is between 0 and 1 and easy to map to either a probability of class 1 or snap to a hard classification of either class with a default threshold of 0.5.

• We can piece it all together by adding each layer:

· The model expects rows of data with 8 variables (the input_dim=8 argument)

· The first hidden layer has 12 nodes and uses the relu activation function.

· The second hidden layer has 8 nodes and uses the relu activation function.

· The output layer has one node and uses the sigmoid activation function.

• Now that the model is defined, *we can compile it*.

• We must specify the loss function to use to evaluate a set of weights, the optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training.

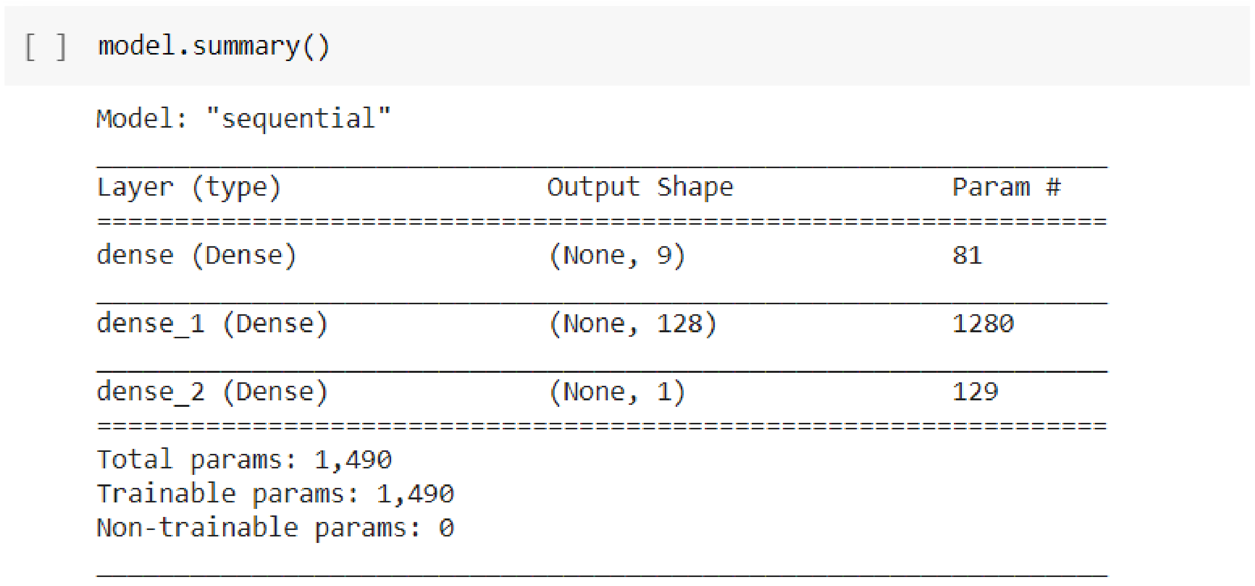
• In this case, we will use cross entropy as the **loss** argument. This loss is for a binary classification problems and is defined in Keras as "**binary_crossentropy**".

• We will define the **optimizer** as the efficient stochastic gradient descent algorithm "**adam**". This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems.

• Finally, because it is a classification problem, we will collect and report the classification accuracy, defined via the **metrics** argument.

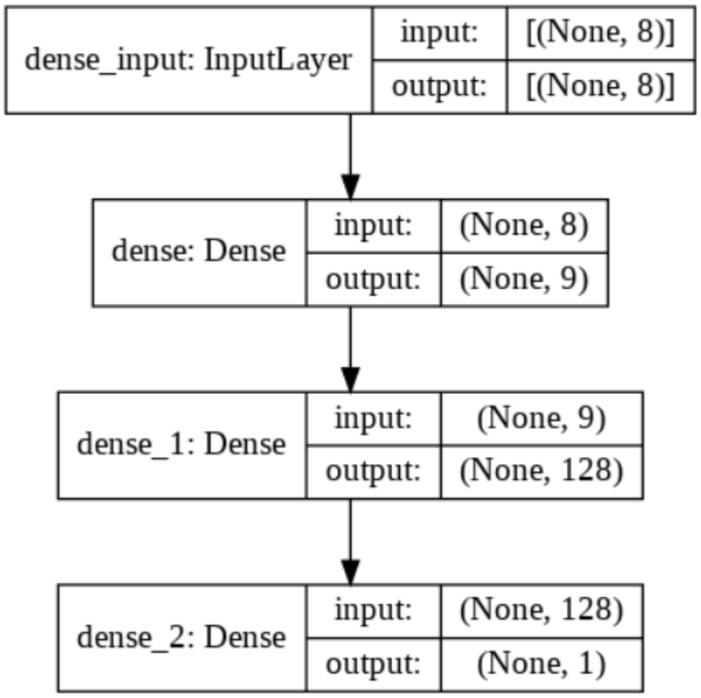
• We have trained our neural network on the entire dataset and we can evaluate the • performance of the network on the same dataset.

• You can evaluate your model on your training dataset using the **evaluate()** function on your model and pass it the same input and output used to train the model.



```
[ ]  model.summary()

     Model: "sequential"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     dense (Dense)                (None, 9)                 81
     _________________________________________________________________
     dense_1 (Dense)              (None, 128)               1280
     _________________________________________________________________
     dense_2 (Dense)              (None, 1)                 129
     =================================================================
     Total params: 1,490
     Trainable params: 1,490
     Non-trainable params: 0
     _________________________________________________________________
```

```
[ ]  from keras.utils.vis_utils import plot_model

     plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```



2. Functional Model

• Unlike the Sequential model, you must create and define a standalone Input layer that specifies the shape of input data.

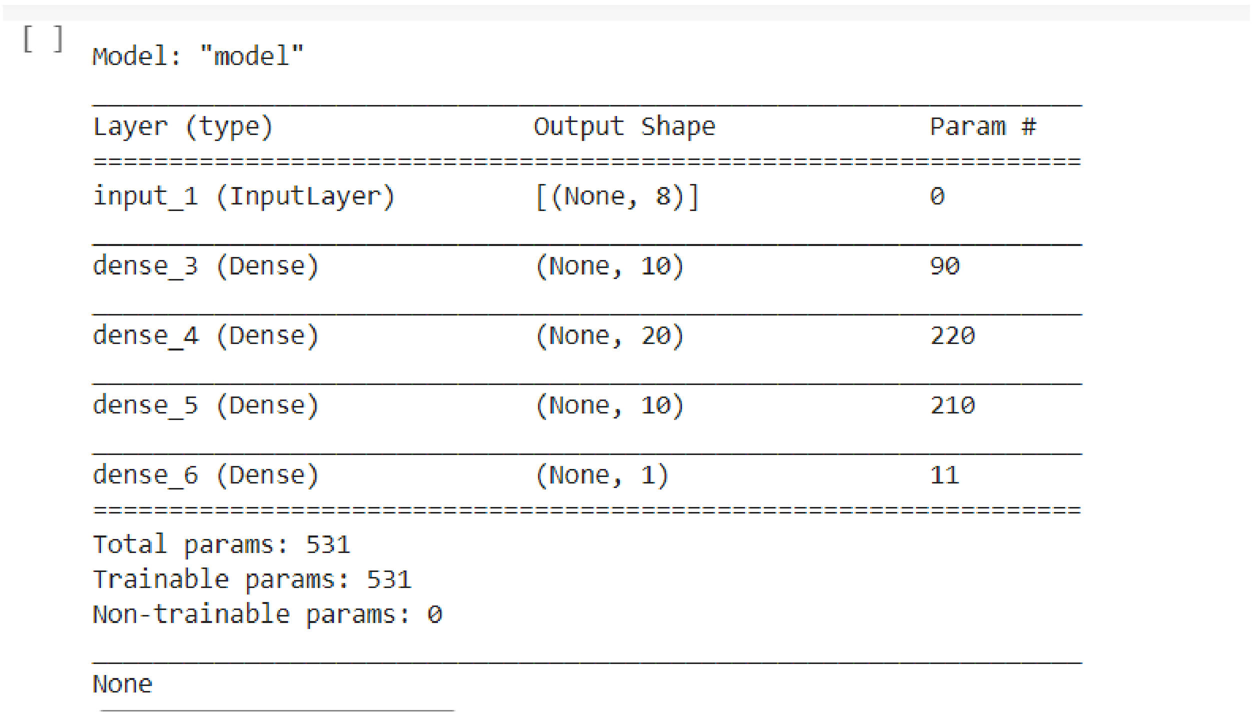
• The input layer takes a shape argument that is a tuple that indicates the dimensionality of the input data.

• When input data is one-dimensional, such as for a multilayer Perceptron, the shape must explicitly leave room for the shape of the mini-batch size used when splitting the data when training the network.

• we define a multilayer Perceptron model for binary classification.

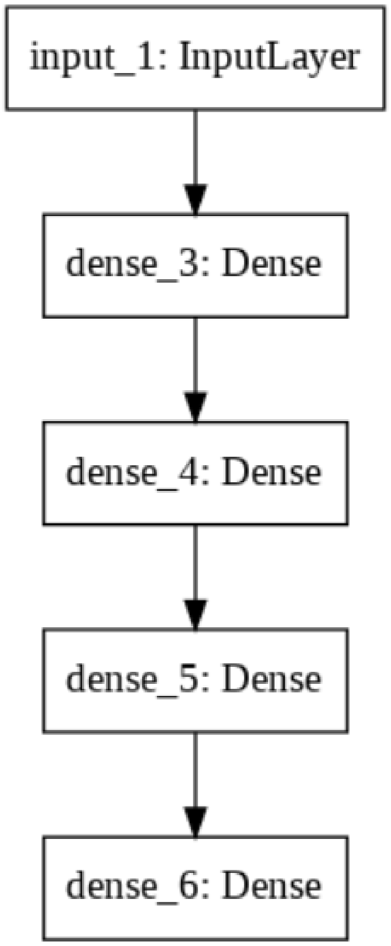
• The model has 10 inputs, 3 hidden layers with 10, 20, and 10 neurons, and an output layer with 1 output. Rectified linear activation functions are used in each hidden layer and a sigmoid activation function is used in the output layer, for binary classification.

• Next we compile the kearas model



```
[ ]  Model: "model"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     input_1 (InputLayer)         [(None, 8)]               0
     _________________________________________________________________
     dense_3 (Dense)              (None, 10)                90
     _________________________________________________________________
     dense_4 (Dense)              (None, 20)                220
     _________________________________________________________________
     dense_5 (Dense)              (None, 10)                210
     _________________________________________________________________
     dense_6 (Dense)              (None, 1)                 11
     =================================================================
     Total params: 531
     Trainable params: 531
     Non-trainable params: 0
     _________________________________________________________________
     None
```

```
________________________________________________________________
None
```



3. Custom Model

In machine learning, a model is a function with learnable parameters that maps an input to an output. The optimal parameters are obtained by training the model on data. A well-trained model will provide an accurate mapping from the input to the desired output.

In Custom Model , we can customise the model by creating our own layers with suitable hyperparameters.

We can create a custom by subclassing the keras.Model.The constructor of MyModel class contains the layers description.

num_classes=8 , denoting the 8 columns in the dataset which is used to determine whether the patient has diabetes or not.

We have created 2 layers with 64 neurons and 8 neurons.

The call() method is used to initialise the weights in the constructor.

Here we have used the RELU activation function. The rectified linear activation function (called ReLU) has been shown to lead to very high-performance networks.This function takes a single number as an input, returning 0 if the input is negative, and the input if the input is positive.

The function nn. relu() provides support for the ReLU in Tensorflow

- Next,we have used the SparseCategoricalCrossentropy() as the loss function.
- sparse_categorical_crossentropy is defined as categorical crossentropy with integer targets:

Arguments: target: An integer tensor. output: A tensor resulting from a softmax

The from_logits=True attribute inform the loss function that the output values generated by the model are not normalized, a.k.a. logits. In other words, the softmax function has not been applied on them to produce a probability distribution.

We have used Adam algorithm for optimization.

Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models

Chapter 5

Implementation of the models

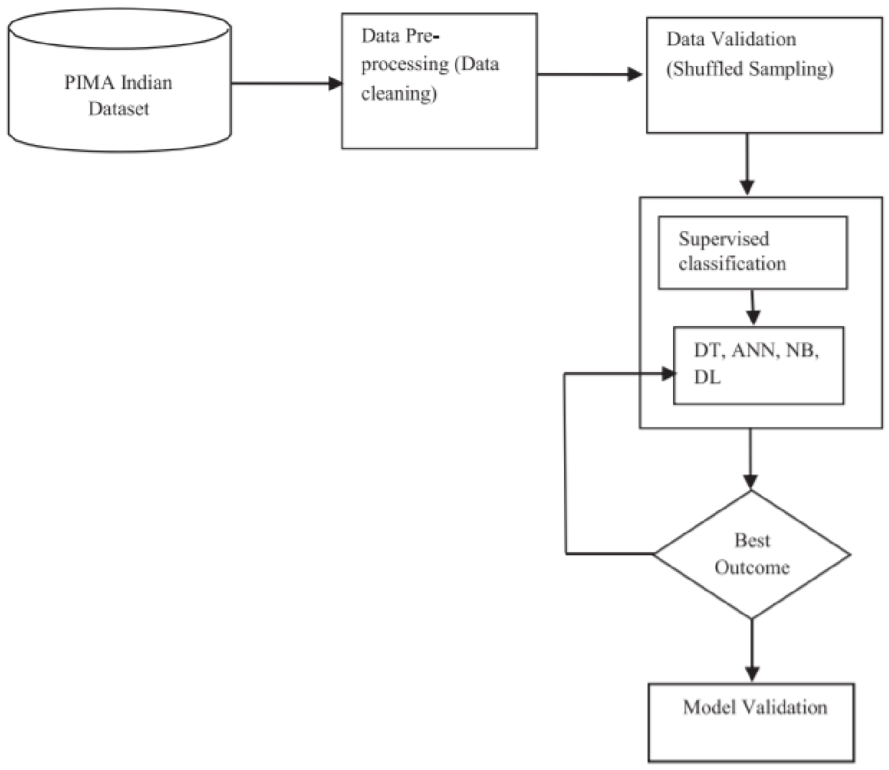

PIMA Indian Dataset
Data Pre-processing (Data cleaning)
Data Validation (Shuffled Sampling)
Supervised classification
DT, ANN, NB, DL
Best Outcome
Model Validation

Fig. 4 Flow chart of the proposed model for diabetes Prediction

Sequential Model

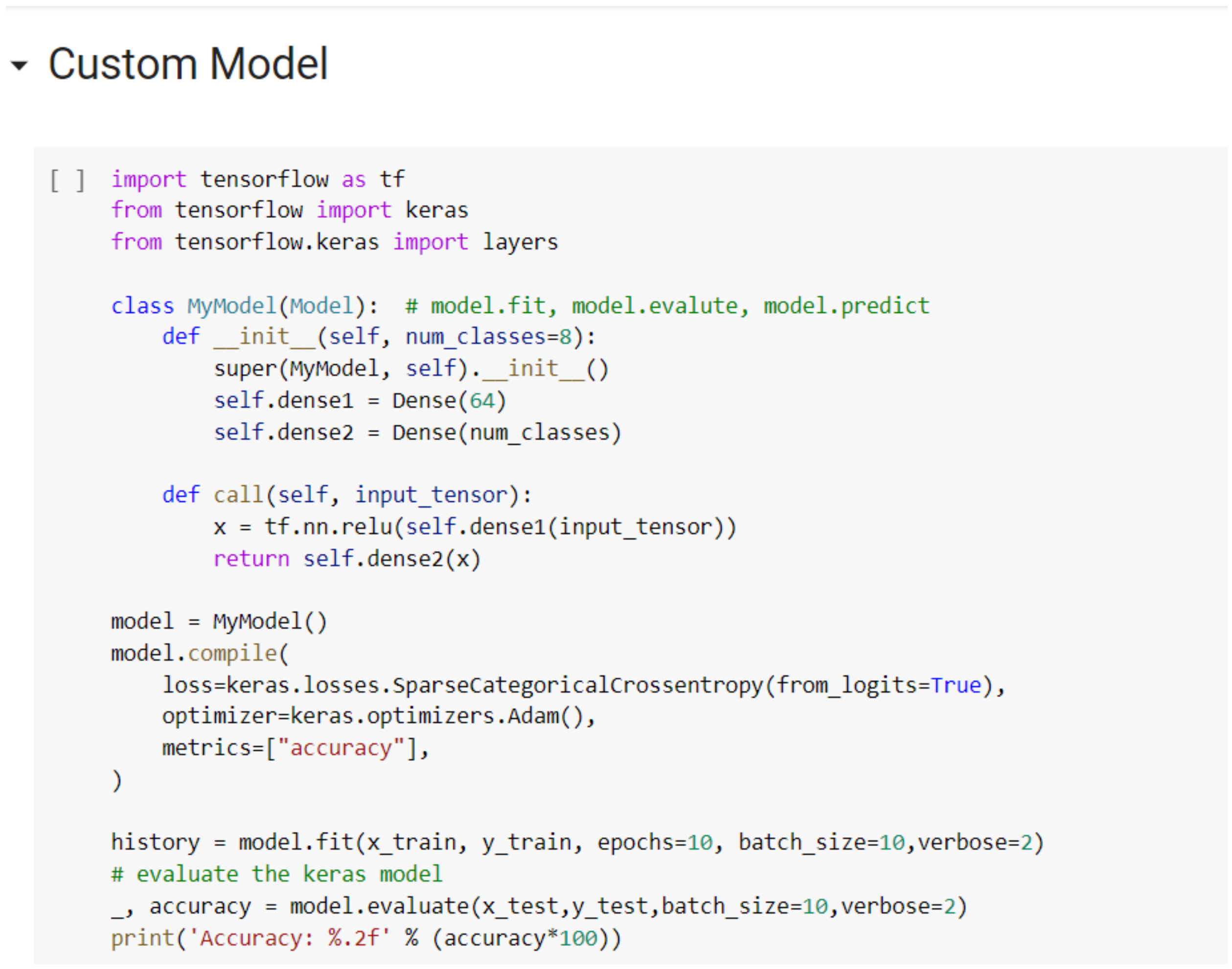
[ ]

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
# load the dataset
dataset = loadtxt('dataset.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:8]
y = dataset[:,8]
# print("X",X)
# print("Y",y)

# define the keras model
model = Sequential()
model.add(Dense(9, input_dim=8, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Functional API

[ ]

```
from keras.utils.vis_utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
input = Input(shape=(8,))
hidden1 = Dense(10, activation='relu')(input)
hidden2 = Dense(20, activation='relu')(hidden1)
hidden3 = Dense(10, activation='relu')(hidden2)
output = Dense(1, activation='sigmoid')(hidden3)
model = Model(inputs=input, outputs=output)
# summarize layers
print(model.summary())
# plot graph
```



Custom Model

```
[ ]  import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras import layers

     class MyModel(Model):  # model.fit, model.evalute, model.predict
         def __init__(self, num_classes=8):
             super(MyModel, self).__init__()
             self.dense1 = Dense(64)
             self.dense2 = Dense(num_classes)

         def call(self, input_tensor):
             x = tf.nn.relu(self.dense1(input_tensor))
             return self.dense2(x)

     model = MyModel()
     model.compile(
         loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
         optimizer=keras.optimizers.Adam(),
         metrics=["accuracy"],
     )

     history = model.fit(x_train, y_train, epochs=10, batch_size=10,verbose=2)
     # evaluate the keras model
     _, accuracy = model.evaluate(x_test,y_test,batch_size=10,verbose=2)
     print('Accuracy: %.2f' % (accuracy*100))
```

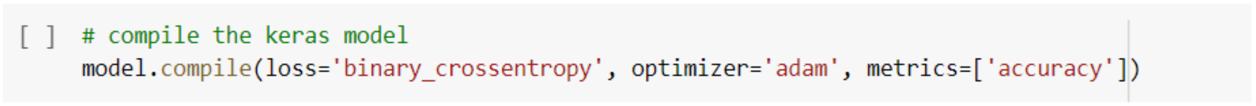
Chapter 6

Performance Measures

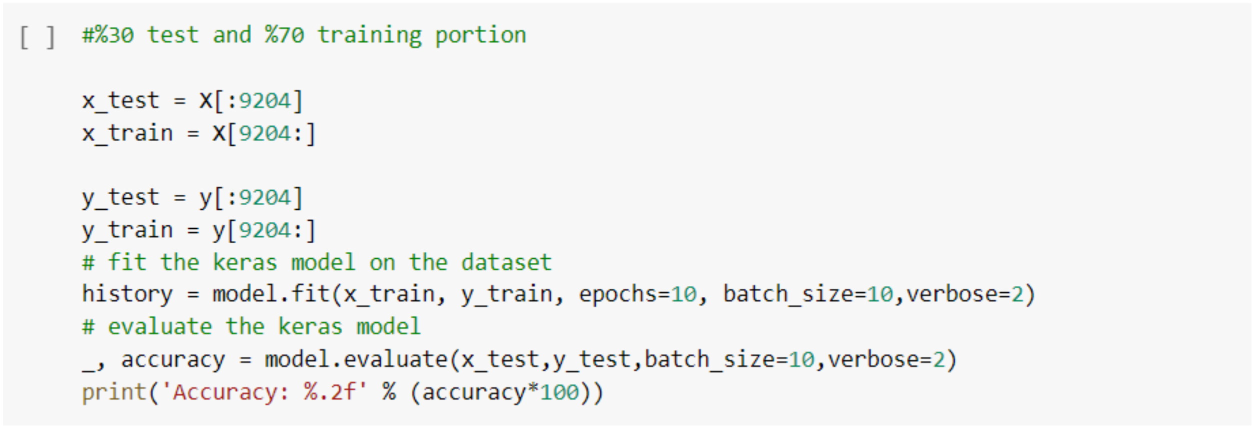
Sequential Model

• We have trained our neural network on the entire dataset and we can evaluate the • performance of the network on the same dataset.

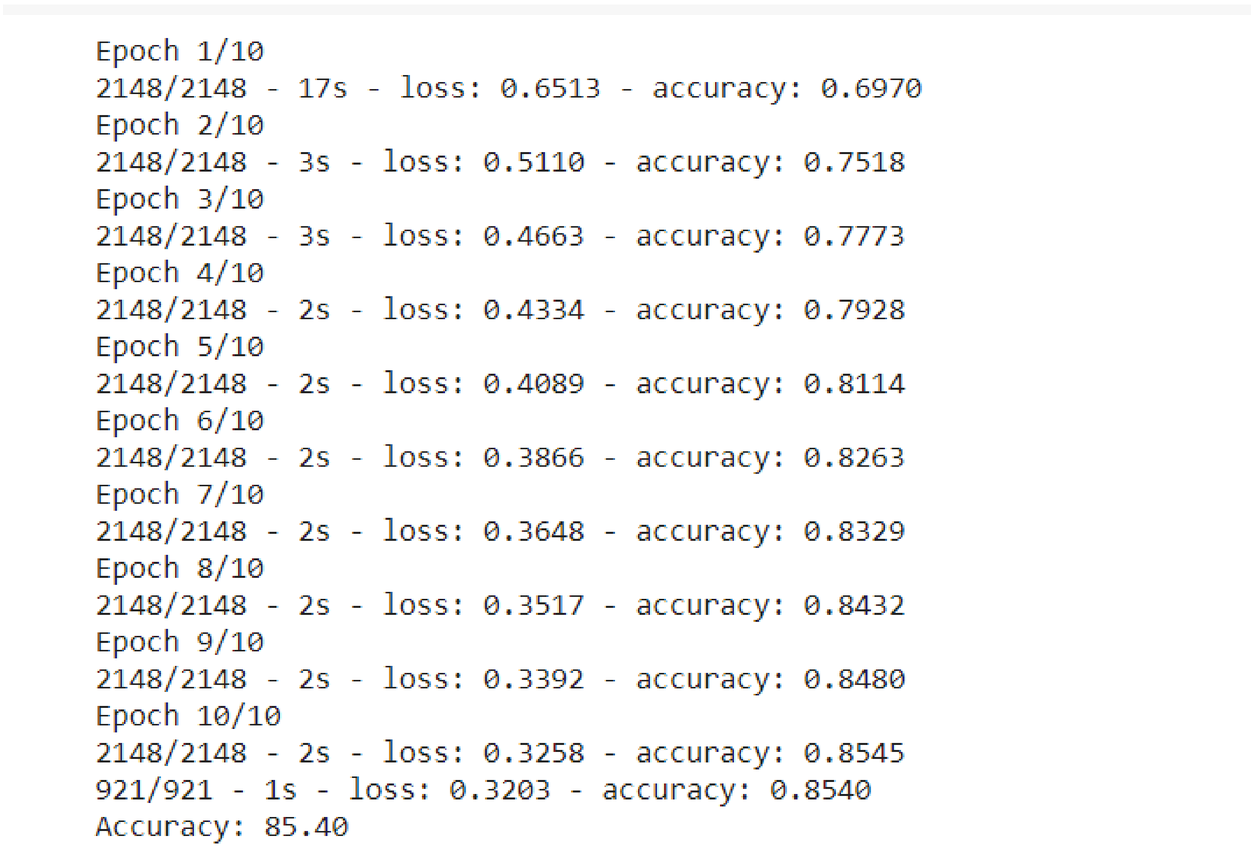
• You can evaluate your model on your training dataset using the **evaluate()** function on your model and pass it the same input and output used to train the model.



```
[ ] # compile the keras model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```



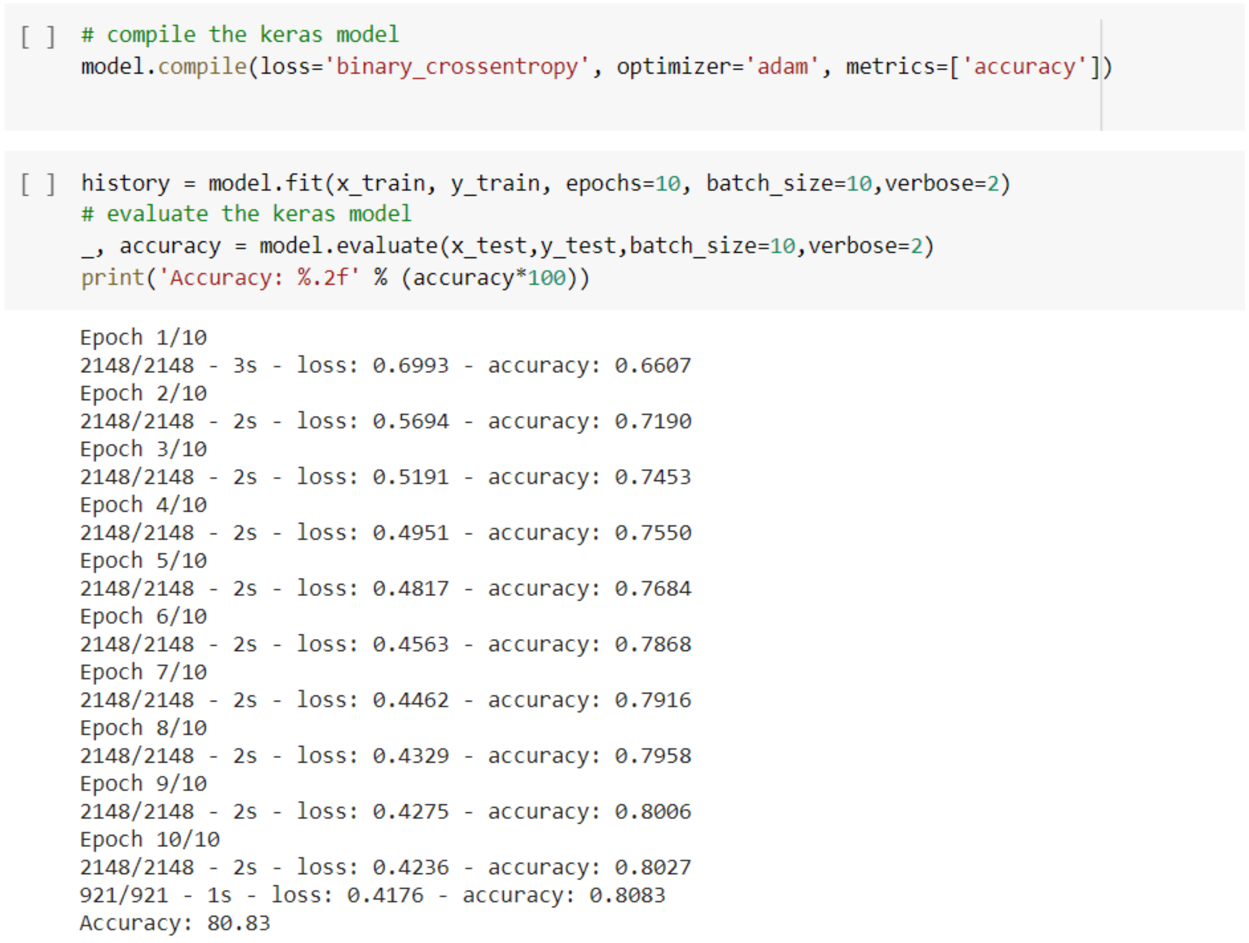
```
[ ] #%30 test and %70 training portion

    x_test = X[:9204]
    x_train = X[9204:]

    y_test = y[:9204]
    y_train = y[9204:]
    # fit the keras model on the dataset
    history = model.fit(x_train, y_train, epochs=10, batch_size=10,verbose=2)
    # evaluate the keras model
    _, accuracy = model.evaluate(x_test,y_test,batch_size=10,verbose=2)
    print('Accuracy: %.2f' % (accuracy*100))
```



```
Epoch 1/10
2148/2148 - 17s - loss: 0.6513 - accuracy: 0.6970
Epoch 2/10
2148/2148 - 3s - loss: 0.5110 - accuracy: 0.7518
Epoch 3/10
2148/2148 - 3s - loss: 0.4663 - accuracy: 0.7773
Epoch 4/10
2148/2148 - 2s - loss: 0.4334 - accuracy: 0.7928
Epoch 5/10
2148/2148 - 2s - loss: 0.4089 - accuracy: 0.8114
Epoch 6/10
2148/2148 - 2s - loss: 0.3866 - accuracy: 0.8263
Epoch 7/10
2148/2148 - 2s - loss: 0.3648 - accuracy: 0.8329
Epoch 8/10
2148/2148 - 2s - loss: 0.3517 - accuracy: 0.8432
Epoch 9/10
2148/2148 - 2s - loss: 0.3392 - accuracy: 0.8480
Epoch 10/10
2148/2148 - 2s - loss: 0.3258 - accuracy: 0.8545
921/921 - 1s - loss: 0.3203 - accuracy: 0.8540
Accuracy: 85.40
```

Functional Model

• Now it is time to execute the model on some data.



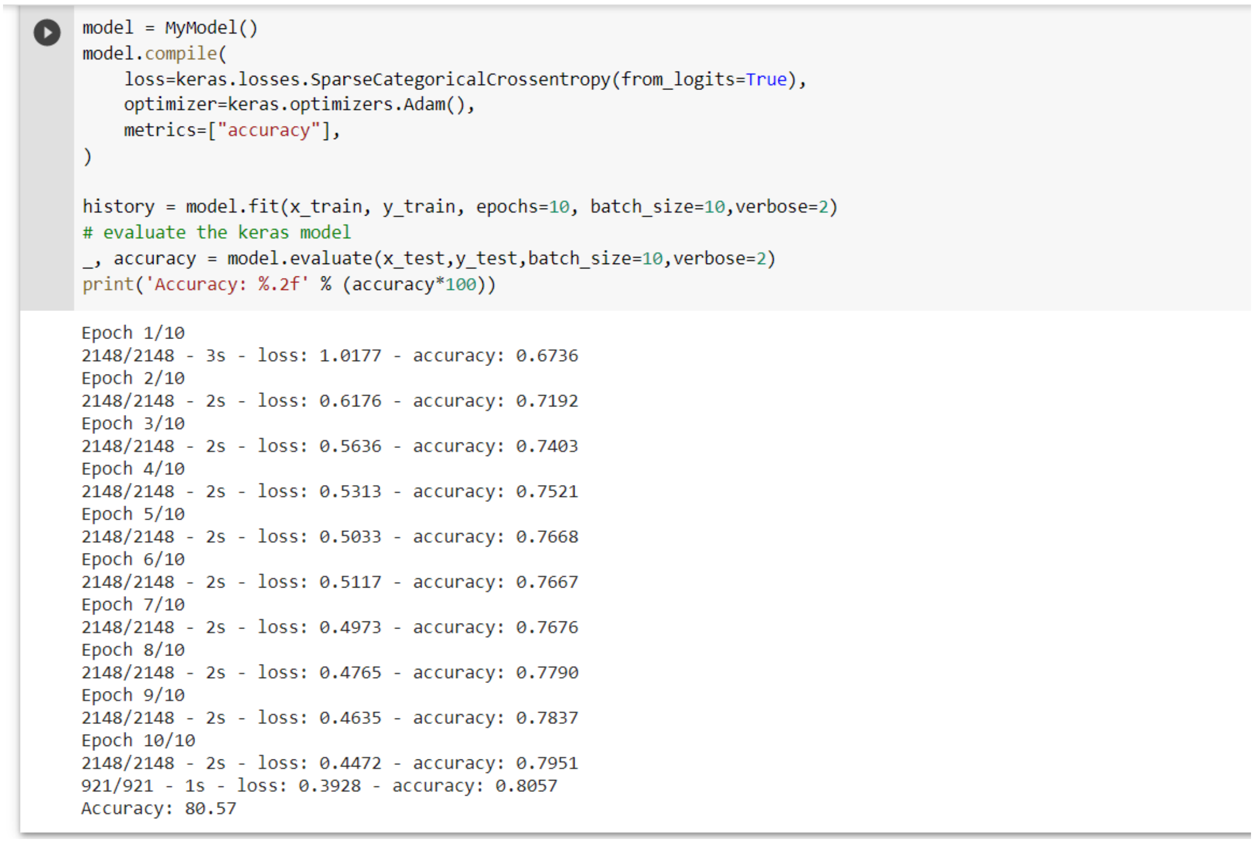
```
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
history = model.fit(x_train, y_train, epochs=10, batch_size=10,verbose=2)
# evaluate the keras model
_, accuracy = model.evaluate(x_test,y_test,batch_size=10,verbose=2)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/10
2148/2148 - 3s - loss: 0.6993 - accuracy: 0.6607
Epoch 2/10
2148/2148 - 2s - loss: 0.5694 - accuracy: 0.7190
Epoch 3/10
2148/2148 - 2s - loss: 0.5191 - accuracy: 0.7453
Epoch 4/10
2148/2148 - 2s - loss: 0.4951 - accuracy: 0.7550
Epoch 5/10
2148/2148 - 2s - loss: 0.4817 - accuracy: 0.7684
Epoch 6/10
2148/2148 - 2s - loss: 0.4563 - accuracy: 0.7868
Epoch 7/10
2148/2148 - 2s - loss: 0.4462 - accuracy: 0.7916
Epoch 8/10
2148/2148 - 2s - loss: 0.4329 - accuracy: 0.7958
Epoch 9/10
2148/2148 - 2s - loss: 0.4275 - accuracy: 0.8006
Epoch 10/10
2148/2148 - 2s - loss: 0.4236 - accuracy: 0.8027
921/921 - 1s - loss: 0.4176 - accuracy: 0.8083
Accuracy: 80.83
```

Custom Model

- Finally, we have fit the model with a batch size of 10 and with epochs 10 and thus calculated the accuracy



```
model = MyModel()
model.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=keras.optimizers.Adam(),
    metrics=["accuracy"],
)

history = model.fit(x_train, y_train, epochs=10, batch_size=10,verbose=2)
# evaluate the keras model
_, accuracy = model.evaluate(x_test,y_test,batch_size=10,verbose=2)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/10
2148/2148 - 3s - loss: 1.0177 - accuracy: 0.6736
Epoch 2/10
2148/2148 - 2s - loss: 0.6176 - accuracy: 0.7192
Epoch 3/10
2148/2148 - 2s - loss: 0.5636 - accuracy: 0.7403
Epoch 4/10
2148/2148 - 2s - loss: 0.5313 - accuracy: 0.7521
Epoch 5/10
2148/2148 - 2s - loss: 0.5033 - accuracy: 0.7668
Epoch 6/10
2148/2148 - 2s - loss: 0.5117 - accuracy: 0.7667
Epoch 7/10
2148/2148 - 2s - loss: 0.4973 - accuracy: 0.7676
Epoch 8/10
2148/2148 - 2s - loss: 0.4765 - accuracy: 0.7790
Epoch 9/10
2148/2148 - 2s - loss: 0.4635 - accuracy: 0.7837
Epoch 10/10
2148/2148 - 2s - loss: 0.4472 - accuracy: 0.7951
921/921 - 1s - loss: 0.3928 - accuracy: 0.8057
Accuracy: 80.57
```

Chapter 7

Conclusion

1. This paper aimed to implement a prediction model for the risk measurement of diabetes. As discussed earlier, a large part of the human population is in the hold of diabetes disease. If remains untreated, then it will create a huge risk for the world.

2. Therefore In our proposed research, we have put into practice diverse classifiers on the PIMA dataset and proved that data mining and machine learning algorithm can reduce the risk factors and improve the outcome in terms of efficiency and accuracy.

3. The outcome achieved on the PIMA Indian dataset is higher than other proposed methodologies on the same dataset using deep learning models.

Chapter 8

References

[1] Zehra, A., Asmawaty, T. and Aznan, M., 2014. *A comparative study on the pre-processing and mining of Pima Indian Diabetes Dataset*. Technical report. 80, 98, 99, 102, 106, 138, 141, 142.

[2] Naz, H. and Ahuja, S., 2020. Deep learning approach for diabetes prediction using PIMA Indian dataset. *Journal of Diabetes & Metabolic Disorders*, *19*(1), pp.391-403.