HOCHSCHULE SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

A Project Report on

"Deep Learning"

Submitted in partial fulfilment of the requirements for the Course:

Project

(Winter Semester 2021-22)

Master of Science

in

Applied Computer Science

By,

SEEMA GANPAT MORE

(310906)

Under the Guidance

of

Constantin Pohl (M.sc)

Abstract

Machine Learning is a method of data analysis that automates analytical model building. Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look. It is used in variety of applications, such as fraud detection, prediction of equipment failures, Email spam filtering, etc. Being a part of broader family of machine learning methods deep learning simply refers to neural networks with more than one hidden layer. Deep learning performs big role in collecting data, analysing data, and making predictions on large amount of data. It makes these processes faster and easier.

Programming Language

- Python

Libraries used

- Numpy
- Pandas
- Pytorch
- Keras
- Tensorflow
- Seaborn
- Matplotlib

Tool Used

- Jupyter Notebook

Basic Terminologies

Data Visualization

Data visualization is graphical or pictorial representation of dataset. That helps to make decisions on certain data and understand it more clearly.

Data Pre-processing

Sometimes our raw dataset is not in a perfect format which we want to create model so data pre-processing is one of the important processes in machine learning. It coverts our datasets in a format of model implementation

Classification Report

Classification report returns precision, recall, f1-score of each class in dataset

- ***Precision*** - The precision returns the proportion of true positives among all the values predicted as positive.
- ***Recall*** - The recall returns the proportion of positive values correctly predicted.
- ***F1 Score*** - The f1-score is the harmonic mean of precision and recall. It is often used to compare classifiers

Prediction:

Prediction refers to the output of an algorithm after it has been trained on a dataset and applied to new data when forecasting the likelihood of a particular outcome, such as whether or not a patient has heart disease or not.

Confusion Matrix:

A confusion matrix is a quick way to compare the labels a model predicts and the actual labels it was supposed to predict. It gives idea of where the model is getting is getting confused. To create the confusion matrix, we can use sklearn confusion_matrix (), which takes the real values(y_test) and the predicted values(y_predict).

Accuracy:

Machine learning model accuracy is the measurement used to determine which model is best at identifying relationships and patterns between variables in a dataset based on the input, or training, data

Introduction of Datasets

For this project we used two types of datasets:

1. **Tabular dataset** – Heart Disease Dataset

2. **Image dataset** – Cloth Datasets

Heart-Disease Dataset

World Health Organization has estimated 12 million deaths occur worldwide; every year due to heart diseases. So, it becoming a need to find different features that becomes a cause of heart related problems. The dataset https://www.kaggle.com/faressayah/predicting-heart-disease-using-machine-learning/data?select=heart.csv has different clinical parameters such as age, sex, maximum heart rate achieved and other health parameters, we implement the karas and pytorch model that predicts whether or not patient has heart disease based on different features.

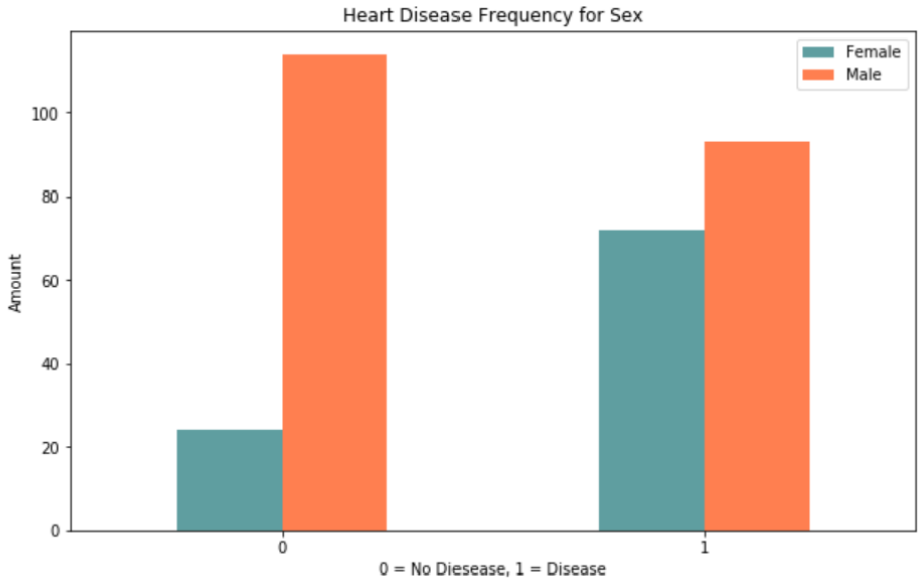
Cloth Dataset

Cloth Dataset is a dataset available on Kaggle https://www.kaggle.com/rizzsum/basic-pattern-women-clothing-dataset has different cloth with various patterns on the dress. It consists of training and test data set of 1000+ images with animal, heart and leaf patterns on the dress. Each example is associated with a label of 0, 1 and 2 classes. The goal of machine learning algorithm is to implement a model that can find a pattern on dress (whether it's a heart pattern, animal pattern or leaf pattern) and gives its performs on basis of accuracy.

Tabular Dataset

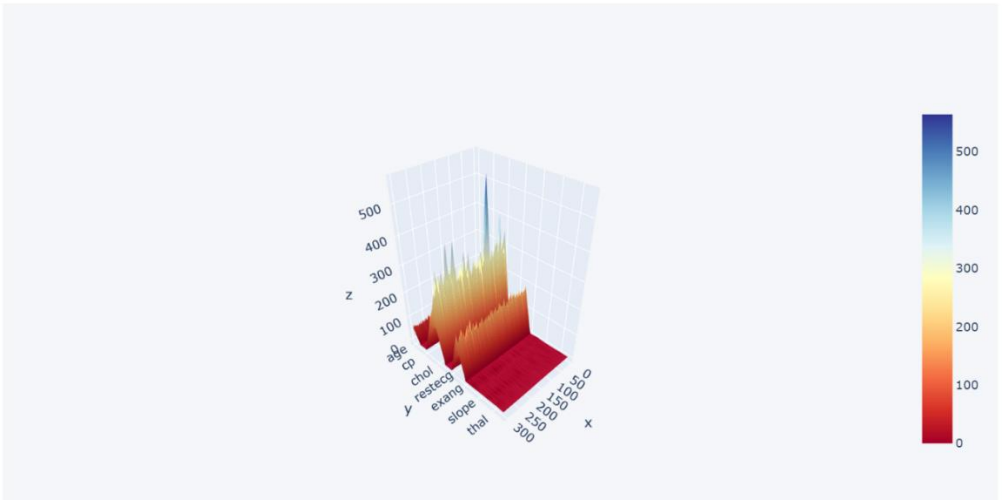
- **Data Visualization of Heart Disease**

Using Bar plot of pandas library, the below graph shows the heart disease between male and female.

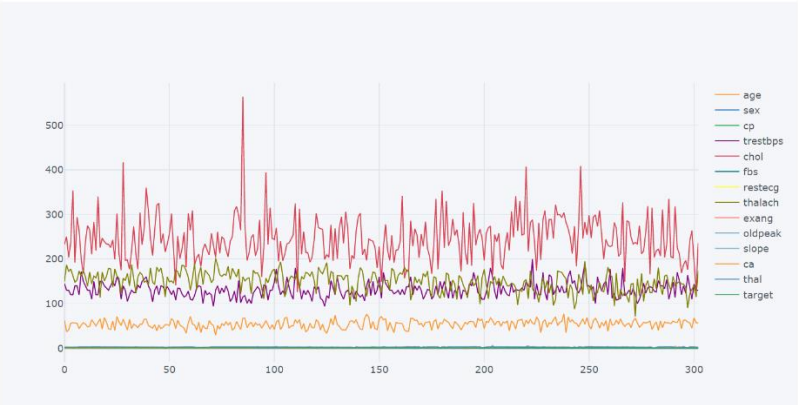


- More visualization of our dataset in using iplot function.

```
In [24]: df.iplot(kind = 'surface', colorscale = 'rdylbu')
```



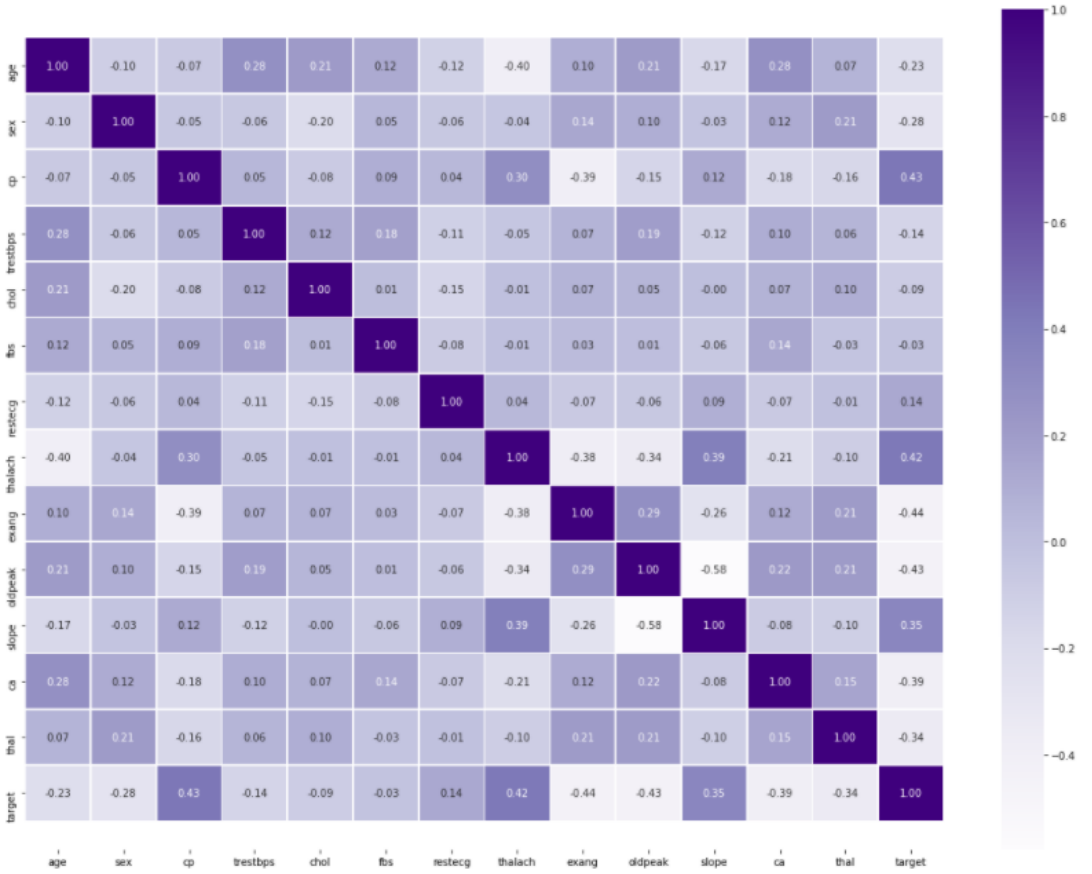
```
In [27]: df.iplot()
```



- The Corelation matrix displaying the corelation between all values in a dataset.

```
In [30]: # Let's make our correlation matrix a little prettier
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(20, 15))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.7,
                 fmt=".2f",
                 cmap="Purples");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

```
Out[30]: (14.5, -0.5)
```



Data Pre-processing

Step 1:

To predict a target value we drop our target column from existing dataset. In Our Dataset Target is a column on which we are making a prediction So we drop it. To implement a model, we need to check the different features of our dataset.

Step 2:

1. Get Dummies

There are some columns in our dataset named as **'sex', 'cp', 'fbs','restecg','exang','slope','ca','thal'** these columns have one category per observation so by using get dummies we produce a new column for each unique categorical value.

```
In [33]: dataset = pd.get_dummies(data=df, columns = ['sex', 'cp', 'fbs', 'restecg','exang','slope','ca','thal'])
```

```
In [34]: dataset
```

Out[34]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	thal_2	thal_3
0	63	145	233	150	2.3	1	0	1	0	0	...	0	1	0	0	0	0	0	1	0	0
1	37	130	250	187	3.5	1	0	1	0	0	...	0	1	0	0	0	0	0	0	1	0
2	41	130	204	172	1.4	1	1	0	0	1	...	1	1	0	0	0	0	0	0	1	0
3	56	120	236	178	0.8	1	0	1	0	1	...	1	1	0	0	0	0	0	0	1	0
4	57	120	354	163	0.6	1	1	0	1	0	...	1	1	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	140	241	123	0.2	0	1	0	1	0	...	0	1	0	0	0	0	0	0	0	1
299	45	110	264	132	1.2	0	0	1	0	0	...	0	1	0	0	0	0	0	0	0	1
300	68	144	193	141	3.4	0	0	1	1	0	...	0	0	0	1	0	0	0	0	0	1
301	57	130	131	115	1.2	0	0	1	1	0	...	0	0	1	0	0	0	0	0	0	1
302	57	130	236	174	0.0	0	1	0	0	1	...	0	0	1	0	0	0	0	0	1	0

303 rows × 31 columns

2. Standard Scalar

For other columns in our dataset such as **'age', 'trestbps', 'chol', 'thalach', 'oldpeak'** we use standard scalar to remove mean and get unit variance.

Step 3:

Depend on our dataset we create train and test labels of dataset.

❖ Keras Sequential Model Creation

Model 1

- We create sequential model with input shape of 13 because train dataset is 13. We use activation function relu and output function sigmoid.
- Crossentropy = Binary Crossentropy as we are dealing with binary dataset.
- Epochs = 20
- Optimizer = adam

Implementing Sequential Model

```
In [45]: #from tensorflow.keras.layers import Input, Dense
         import tensorflow.keras as keras
         Model_1 = keras.Sequential()
         Model_1.add(keras.Input(shape=(13,)))
         Model_1.add(keras.layers.Dense(13, activation='relu'))
         Model_1.add(keras.layers.Dense(1, activation='sigmoid'))

         Model_1.compile(optimizer='adam',
                         loss='binary_crossentropy',
                         metrics=['accuracy'])

         Hist_1 = Model_1.fit(Train, TrainLabel, validation_split=0.2, epochs=20)
```

Input shape same as train dataset shape

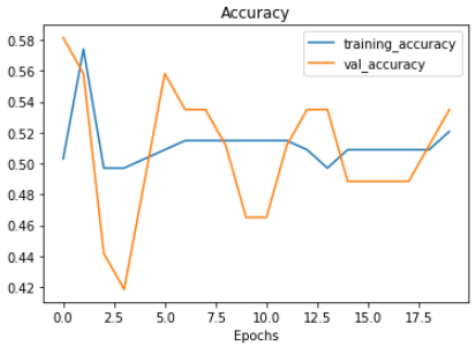
1 Activation function Relu

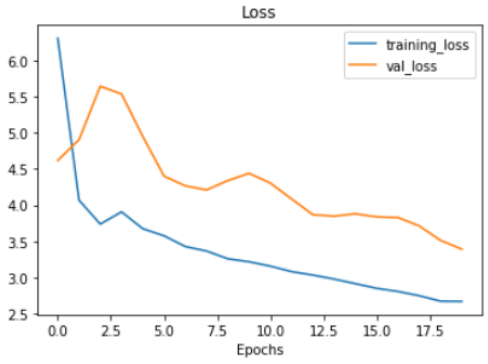
Output function sigmoid as we are dealing with binary classification

Epoch - 20

we get 41% Accuracy

- **Loss and Accuracy Curves**





- Model 2,3,4,5 has improved with adding 2 more dense layers and more epoch steps but still our accuracy doesn't improve.

Improving Our Model

Model 6

In Model 6. I have added more input layers and epochs of 150 So its performs quite well.

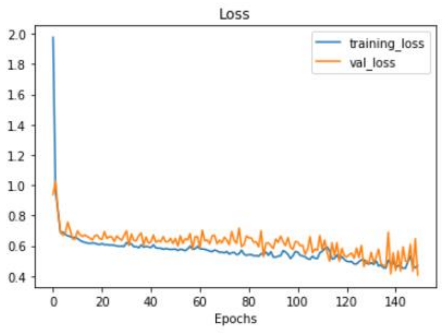
```
In [61]: # Set random seed
         tf.random.set_seed(42)

         #from tensorflow.keras.layers import Input, Dense
         import tensorflow.keras as keras
         Model_6 = keras.Sequential()     # add 1 more dense layers and epochs to 150
         Model_6.add(keras.Input(shape=(13,)))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(13, activation='relu'))
         Model_6.add(keras.layers.Dense(1, activation='sigmoid'))

         Model_6.compile(optimizer='adam',
                         loss='binary_crossentropy',
                         metrics=['accuracy'])

         Hist_6 = Model_6.fit(Train, TrainLabel, validation_split=0.2, epochs=150)
```

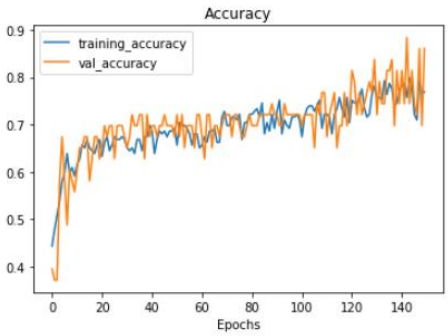
Add 6 more activation layers

Epochs 150

Our Model 6 gets accuracy of 80 % which is satisfying.

- **Loss and Accuracy Curves**





We try to improve Accuracy more. So We add 2 more Dense Layers and epochs to 200.

Model 7

```
In [62]: # Set random seed
tf.random.set_seed(42)

#from tensorflow.keras.layers import Input, Dense
import tensorflow.keras as keras
Model_7 = keras.Sequential()      # add 2 more dense layers and epochs to 200
Model_7.add(keras.Input(shape=(13,)))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(13, activation='relu'))
Model_7.add(keras.layers.Dense(1, activation='sigmoid'))

Model_7.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

Hist_7 = Model_7.fit(Train, TrainLabel, validation_split=0.2, epochs=200)
```

Add 2 more activation than model 6

Epochs are same as model 6

Though model 7 has less accuracy than model 6 but based on classification report Model 7 has performed well.

Prediction on model 7

```
In [83]: print(PredsLabelsA[:10])
         print(TestLabel[:10])
```

```
[1 1 1 1 1 0 1 1 1 1]
[1 1 1 1 1 1 1 1 1 1]
```

Classification Report

Classification Report

```
In [84]: print(classification_report(TestLabel, PredsLabelsA))
```

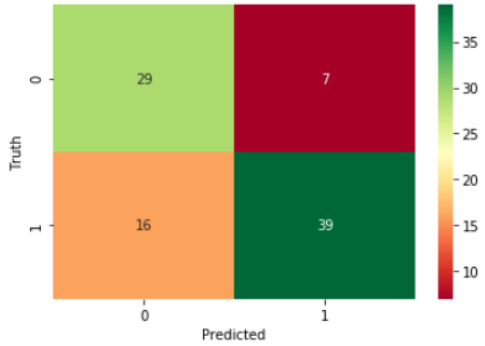
	precision	recall	f1-score	support
0	0.64	0.81	0.72	36
1	0.85	0.71	0.77	55
accuracy			0.75	91
macro avg	0.75	0.76	0.74	91
weighted avg	0.77	0.75	0.75	91

Confusion Matrix

Confusion Matrix

In [85]:
```
import tensorflow as tf
import seaborn as sn
n_conf_mat = tf.math.confusion_matrix(labels=TestLabel, predictions=PredsLabelsA)

sn.heatmap(n_conf_mat, annot=True, cmap='RdYlGn', fmt='d') #YlOrBr
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[85]: Text(33.0, 0.5, 'Truth')



Model Summary

In [65]:
```
Model_7.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_32 (Dense)            (None, 13)                182

 dense_33 (Dense)            (None, 13)                182

 dense_34 (Dense)            (None, 13)                182

 dense_35 (Dense)            (None, 13)                182

 dense_36 (Dense)            (None, 13)                182

 dense_37 (Dense)            (None, 13)                182

 dense_38 (Dense)            (None, 13)                182

 dense_39 (Dense)            (None, 13)                182

 dense_40 (Dense)            (None, 13)                182

 dense_41 (Dense)            (None, 1)                 14

=================================================================
Total params: 1,652
Trainable params: 1,652
Non-trainable params: 0
_________________________________________________________________
```

❖ *Pytorch Sequential Model of heart disease dataset*

Pre-processing

As we have already split labels from dataset we use same pre-processed dataset for pytorch model.

Then we split dataset into train and validation dataset

```
In [102]: # split train and validation data
          p = int((len(cTrain)*0.8))

          cTrain_p = cTrain[:p]
          cVal = cTrain[p:]

          cTrainLabels_p = cTrainLabels[:p]
          cValLabels = cTrainLabels[p:]

          print(len(cTrain_p), " ", len(cVal))
```

```
169   43
```

Build sequential nn model

Based on our model 7 we get some more ideas about dataset and its working so in pytorch model

We have used 3 linear layers with and Relu as activation function and sigmoid as output function

Build Sequential nn Model

```
In [154]: class CNeuralNetwork(nn.Module):
              def __init__(self):
                  super(CNeuralNetwork,self).__init__()
                  self.flatten = nn.Flatten()
                  self.model = nn.Sequential(
                      nn.Linear(13, 26),
                      nn.ReLU(),
                      nn.Linear(26, 13),
                      nn.Sigmoid(),
                      nn.Linear(13,1),
                  )

              def forward(self, x):
                  x = self.flatten(x)
                  prediction = self.model(x)
                  return prediction
```

- Define Train and Test function

In [156]:
```
def cTraining(dataloader, model, loss_fn, optimizer):
    full_loss = 0
    correct = 0

    for (X, y) in dataloader:

        pred = model(X)
        yf = y.type(torch.FloatTensor)
        loss = loss_fn(pred, yf.unsqueeze(1)) # [32] -> [32, 1]
        full_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        correct += (torch.flatten((pred > 0).long()) == y).type(torch.float).sum().item()

    full_loss /= len(dataloader)
    correct /= len(dataloader.dataset)
    print("Training Loss ", full_loss)
    print("Training Accuracy ", correct)
    return full_loss, correct
```

In [157]:
```
def cTesting(dataloader, model, loss_fn):
    test_loss = 0
    correct = 0
    predLabels = []

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            yf = y.type(torch.FloatTensor)
            test_loss += loss_fn(pred, yf.unsqueeze(1)).item()
            correct += (torch.flatten((pred > 0).long()) == y).type(torch.float).sum().item()
            predLabels.append(torch.flatten((pred > 0).long()).tolist())

    test_loss /= len(dataloader)
    correct /= len(dataloader.dataset)
    print("Testing loss ", test_loss)
    print("Testing Accuracy ", correct)
    predLabels = list(np.concatenate(predLabels).flat)
    return test_loss, correct, predLabels
```

- **<u>Evaluate our model with 100 epochs</u>**

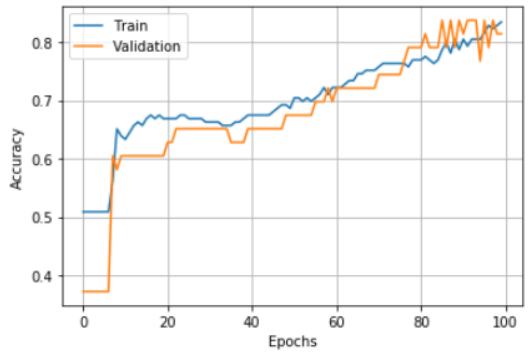
In [158]:
```
cModel = CNeuralNetwork()
cLoss_fn = nn.BCEWithLogitsLoss() #nn.BCELoss
cOptimizer = torch.optim.Adam(cModel.parameters())

cTrainingAcc = []
cTrainingLoss = []
cValidationAcc = []
cValidationLoss = []

for e in range(100):
    print("Epoch ", e+1, "\n---------------------------------")
    cTLoss, cTAcc = cTraining(cTrain_dataloader, cModel, cLoss_fn, cOptimizer)
    cTrainingAcc.append(cTAcc)
    cTrainingLoss.append(cTLoss)
    cVLoss, cVAcc, x = cTesting(cVal_dataloader, cModel, cLoss_fn)
    cValidationAcc.append(cVAcc)
    cValidationLoss.append(cVLoss)
    print()
x, y, cPredLabels = cTesting(cTest_dataloader, cModel, cLoss_fn)
```

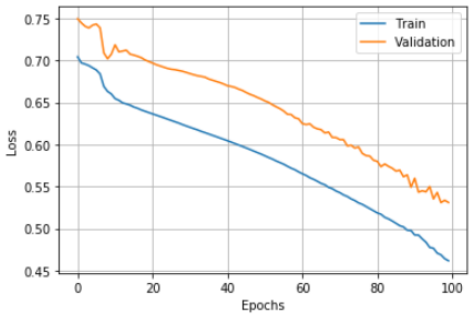
Epochs

```
Epoch  1
---------------------------------
Training Loss  0.7043562233448029
Training Accuracy  0.5088757396449705
Testing loss  0.7496601939201355
Testing Accuracy  0.37209302325581395

Epoch  2
---------------------------------
Training Loss  0.6971265574296316
Training Accuracy  0.5088757396449705
Testing loss  0.7447157502174377
Testing Accuracy  0.37209302325581395

Epoch  3
---------------------------------
Training Loss  0.6956970989704132
Training Accuracy  0.5088757396449705
Testing loss  0.7407433092594147
```

Our Model got 81% Accuracy

- **Accuracy and loss curves**





- **Classification Report**

Classification Report

```
In [161]: print(classification_report(cTestLabels, cPredLabels))
```

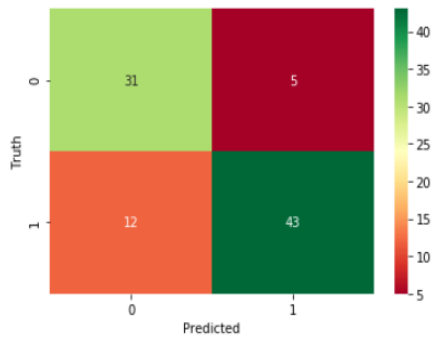
	precision	recall	f1-score	support
0	0.72	0.86	0.78	36
1	0.90	0.78	0.83	55
accuracy			0.81	91
macro avg	0.81	0.82	0.81	91
weighted avg	0.83	0.81	0.82	91

- **Confusion Matrix**

Confusion Matrix

```
In [162]: cConf_mat = confusion_matrix(cTestLabels, cPredLabels)

          sn.heatmap(cConf_mat, annot=True, cmap='RdYlGn', fmt='d')
          plt.xlabel('Predicted')
          plt.ylabel('Truth')
```

```
Out[162]: Text(33.0, 0.5, 'Truth')
```



➢ Comparison Keras and Pytorch Model

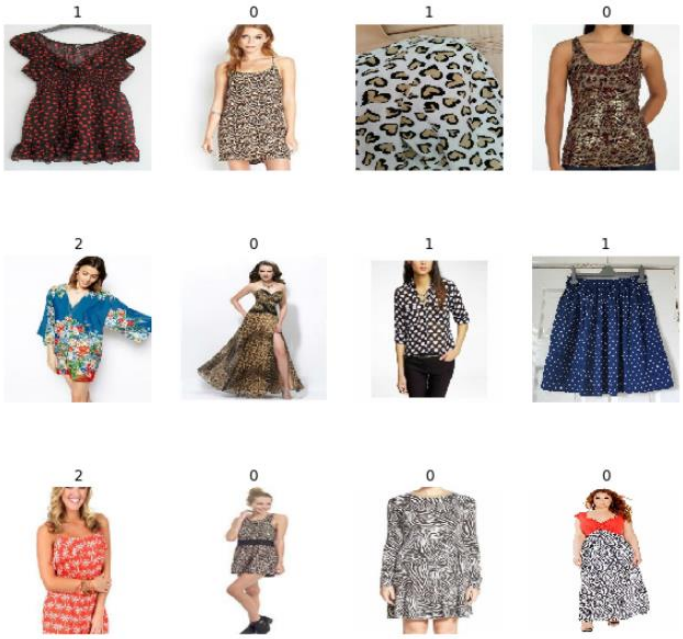
		Precision	Recall	F1 score	Epochs	Accuracy
Keras	0	0.64	0.81	0.72	200	0.75
	1	0.85	0.71	0.77		
Pytorch	0	0.72	0.86	0.78	100	0.81
	1	0.90	0.78	0.83		

Based on Our Keras and pytorch models our Pytorch model performs very well based on accuracy, F1 Score, recall, precision score.

Image Dataset

- **Data Visualization of cloth dataset**

```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in train_dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



- **<u>Data Preprocessing</u>**

Same like tabular dataset we split our train and test labels from train and test data

In [8]:
```
img_train = []
label_train = []
img_path = 'Train Dataset'

for directory in os.listdir(img_path):
    for file in os.listdir(os.path.join(img_path, directory)):

        image_path = os.path.join(img_path, directory, file)
        image = np.array(Image.open(image_path))
        image = cv2.imread(image_path)
        image = np.array(cv2.resize(image,(45,45)))
        image = image / 255
        image = image.astype('float32')
        img_train.append(image)
        dir_int = int(directory)
        label_train.append(dir_int)

img_arr_train = np.array(img_train)
label_arr_train = np.array(label_train)
```

In [15]:
```
img_test = []
label_test = []
img_path = 'Test Dataset'

for directory in os.listdir(img_path):
    for file in os.listdir(os.path.join(img_path, directory)):

        image_path = os.path.join(img_path, directory, file)
        image = np.array(Image.open(image_path))
        image = cv2.imread(image_path)
        image = np.array(cv2.resize(image,(45,45)))
        image = image / 255
        image = image.astype('float32')
        img_test.append(image)
        dir_int = int(directory)
        label_test.append(dir_int)

img_arr_test = np.array(img_test)
label_arr_test = np.array(label_test)
```

❖ Keras Model Implementation

Model 1

- For our image dataset we have input of **6075 neurons** similar as train dataset shape.
- We have used **sparse categorical Crossentropy** as we are dealing with categorical classification problem.
- **Softmax** as output function used for categorical classification problem.

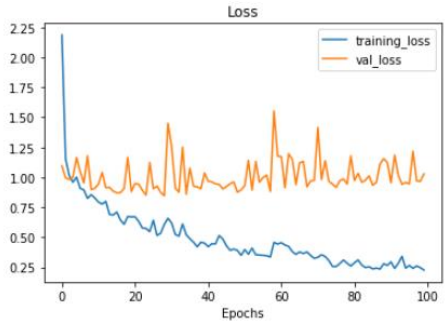
In [73]:
```
Model_1 = keras.Sequential()
Model_1.add(keras.Input(shape=(6075,)))
Model_1.add(keras.layers.Dense(3, activation='softmax'))

Model_1.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

History_1 = Model_1.fit(Train, TrainLabels, validation_split=0.2, epochs=100)
```

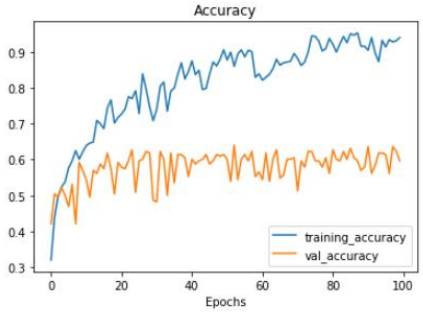
Input layer (of 6075 nuerons)

Output function

Epochs

- **Loss and accuracy curves**





Model 1 get **88 % Accuracy** with test data. We need to improve model more.

Improvement of Our Model

Model 2

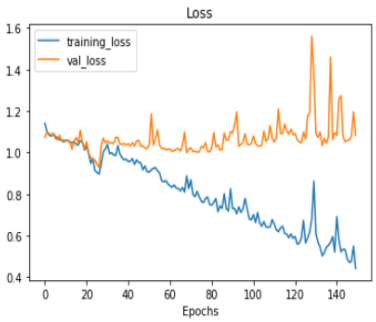
```
In [76]: Model_2 = keras.Sequential()
         Model_2.add(keras.Input(shape=(6075,))),
         Model_2.add(keras.layers.Dense(10, activation='relu')),
         Model_2.add(keras.layers.Dense(10, activation='relu')),
         Model_2.add(keras.layers.Dense(3, activation='softmax'))

         Model_2.compile(optimizer='adam',
                         loss='sparse_categorical_crossentropy',
                         metrics=['accuracy'])

         History_2 = Model_2.fit(Train, TrainLabels, validation_split=0.2, epochs=150)
```

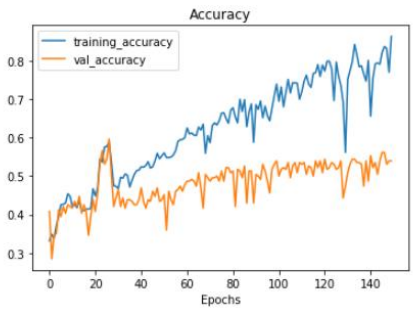
Added 2 more input layers

Added 50 more epochs

Accuracy – 80%

- **Loss and Accuracy Curves**





By including more input layers and epochs of 150. but still our model is not performing well. So, will need to improve our model more.

In [40]:
```
Model_3 = keras.Sequential()
Model_3.add(keras.Input(shape=(6075,))),
Model_3.add(keras.layers.Dense(64, activation='relu')),
Model_3.add(keras.layers.Dense(128, activation='relu')),
Model_3.add(keras.layers.Dense(256, activation='relu')),
Model_3.add(keras.layers.Dense(3, activation='softmax'))

Model_3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])

History_3 = Model_3.fit(Train, TrainLabels, validation_split=0.2, epochs=15)
```

Added 1 more input layer

But, reduced epochs to 15

Model 3 got less accuracy than model 2 So, will do prediction on model 2.

- ## Predictions on model 2

In [79]:
```
#Do some predictions with our model 2
Preds = Model_2.predict(Test)
PredLabels = []
for arr in Preds:
    PredLabels.append(np.argmax(arr))
Final_Pred_Label = np.array(PredLabels)
```

In [80]:
```
print(Final_Pred_Label[:15])
print(TestLabels[:15])
```

```
[2 1 0 2 1 1 2 2 0 1 2 0 1 0 0]
[2 1 0 2 2 1 2 2 0 1 2 0 1 2 0]
```

- ## Classification Report

Classification Report

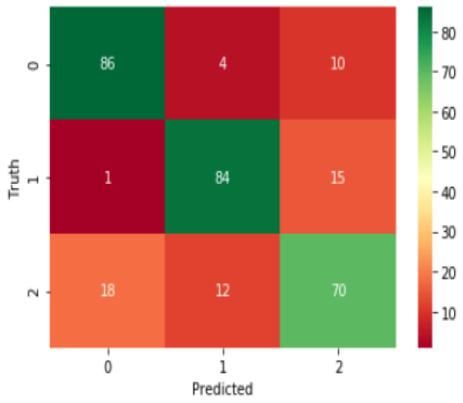
In [81]:
```
print(classification_report(TestLabels, Final_Pred_Label))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       100
           1       0.84      0.84      0.84       100
           2       0.74      0.70      0.72       100

    accuracy                           0.80       300
   macro avg       0.80      0.80      0.80       300
weighted avg       0.80      0.80      0.80       300
```

- **Confusion Matrix**

```
In [82]: conf_mat = tf.math.confusion_matrix(labels=TestLabels, predictions=Final_Pred_Label)

         sn.heatmap(conf_mat, annot=True, cmap='RdYlGn', fmt='d')
         plt.xlabel('Predicted')
         plt.ylabel('Truth')
```

```
Out[82]: Text(33.0, 0.5, 'Truth')
```



Model Summary

```
In [87]: # Summary of model_2
         Model_2.summary()
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_57 (Dense)            (None, 10)                60760

 dense_58 (Dense)            (None, 10)                110

 dense_59 (Dense)            (None, 3)                 33

=================================================================
Total params: 60,903
Trainable params: 60,903
Non-trainable params: 0
_________________________________________________________________
```

❖ Pytorch Model for image dataset

- We used already pre-processed dataset for pytroch model also.
- Then we split it into train and validation set

In [178]:
```
# split train and validation data
p = int((len(Train)*0.8))

Train_p = Train[:p]
Val = Train[p:]

TrainLabels_p = TrainLabels[:p]
ValLabels = TrainLabels[p:]

print(len(Train_p), " ", len(Val))
```

```
912   228
```

- Then we initialize dataset and define data loader with batch size

In [179]:
```
# Initialize Dataset
class CustomDataset(Dataset):
    def __init__(self, features, labels):
        self.features = torch.from_numpy(features).float()
        self.labels = torch.from_numpy(labels).long()

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        feature_vec = self.features[idx]
        label = self.labels[idx]
        return feature_vec, label
```

In [180]:
```
# Define dataloader with batch size
iTrain_dataloader = DataLoader(CustomDataset(Train_p, TrainLabels_p), batch_size=32)
iVal_dataloader = DataLoader(CustomDataset(Val, ValLabels), batch_size=32)
iTest_dataloader = DataLoader(CustomDataset(Test, TestLabels), batch_size=32)
```

Build Sequential nn model

Build Sequential nn Model

In [9]:
```
class INeuralNetwork(nn.Module):
    def __init__(self):
        super(INeuralNetwork,self).__init__()
        self.flatten = nn.Flatten()
        self.model = nn.Sequential(
            nn.Linear(6075, 3),
            nn.Dropout(0.2),
            nn.Linear(3, 10),
            #nn.ReLU(),
            nn.LogSoftmax(dim=1),
        )

    def forward(self, x):
        x = self.flatten(x)
        prediction = self.model(x)
        return prediction
```

Added 2 input layers

Drop some layers

- **Define Training and testing datasets for modelling**

In [190]:
```
def iTraining(dataloader, model, loss_fn, optimizer):
    full_loss = 0
    correct = 0

    for (X, y) in dataloader:

        pred = model(X)
        loss = loss_fn(pred, y)
        full_loss += loss.item()

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    full_loss /= len(dataloader)
    correct /= len(dataloader.dataset)
    print("Training Loss ", full_loss)
    print("Training Accuracy ", correct)
    return full_loss, correct
```

In [191]:
```
def iTesting(dataloader, model, loss_fn):
    test_loss = 0
    correct = 0
    predLabels = []

    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()
            predLabels.append(pred.argmax(1).tolist())

    test_loss /= len(dataloader)
    correct /= len(dataloader.dataset)
    print("Testing loss ", test_loss)
    print("Testing Accuracy ", correct)
    predLabels = list(np.concatenate(predLabels).flat)
    return test_loss, correct, predLabels
```

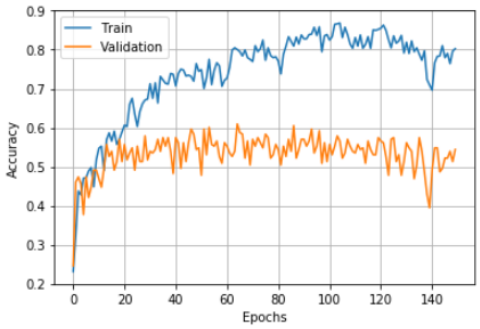
- **Evaluate the Model**

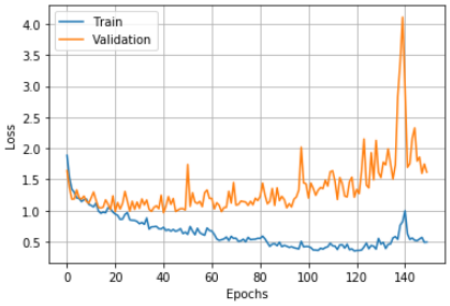
```
In [232]: iModel = INeuralNetwork()
          iLoss_fn = nn.CrossEntropyLoss()
          iOptimizer = torch.optim.Adam(iModel.parameters(), lr = 0.001)

          iTrainingAcc = []
          iTrainingLoss = []
          iValidationAcc = []
          iValidationLoss = []

          for e in range(150):
              print("Epoch ", e+1, "\n-----------------------------------")
              iTLoss, iTAcc = iTraining(iTrain_dataloader, iModel, iLoss_fn, iOptimizer)
              iTrainingAcc.append(iTAcc)
              iTrainingLoss.append(iTLoss)
              iVLoss, iVAcc, x = iTesting(iVal_dataloader, iModel, iLoss_fn)
              iValidationAcc.append(iVAcc)
              iValidationLoss.append(iVLoss)
              print()
          x, y, iPredLabels = iTesting(iTest_dataloader, iModel, iLoss_fn)
```

150 Epochs

Our Model gives 76 % Accuracy which is very well

- **Loss and Accuracy curves**





- **Classification Report**

Classification Report

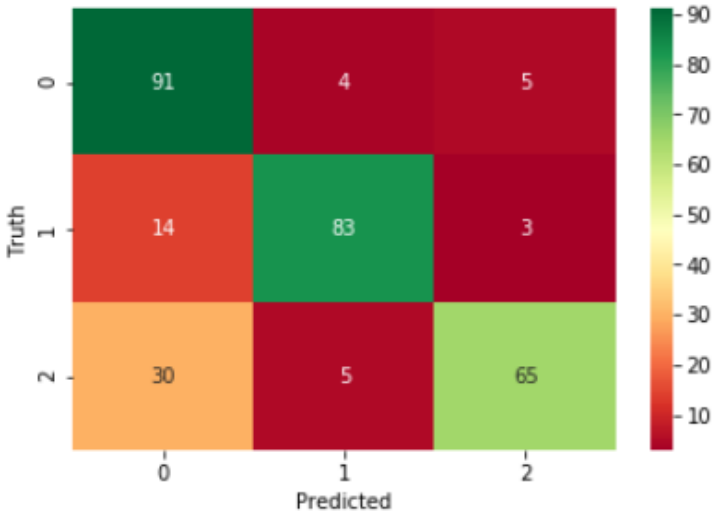
```
In [16]: print(classification_report(TestLabels, iPredLabels))
```

	precision	recall	f1-score	support
0	0.67	0.91	0.77	100
1	0.90	0.83	0.86	100
2	0.89	0.65	0.75	100
accuracy			0.80	300
macro avg	0.82	0.80	0.80	300
weighted avg	0.82	0.80	0.80	300

Confusion Matrix

```
: iConf_mat = confusion_matrix(TestLabels, iPredLabels)

  sn.heatmap(iConf_mat, annot=True, cmap='RdYlGn', fmt='d')
  plt.xlabel('Predicted')
  plt.ylabel('Truth')
```

```
: Text(33.0, 0.5, 'Truth')
```



So, Our keras model has more Accuracy than Pytorch Model. Keras model is best suitable for image dataset.

- **Comparison of Keras and Pytorch Model**

		Precision	Recall	F1 score	Epochs	Accuracy
Keras	0	0.82	0.86	0.84	100	0.80
	1	0.84	0.84	0.84		
	2	0.74	0.70	0.72		
Pytorch	0	0.67	0.91	0.77	150	0.80
	1	0.90	0.83	0.86		
	2	0.89	0.65	0.75		

Our Keras and pytorch models has same accuracy of 80% but based on other parameters of F1 Score, recall, precision pytorch model works better than keras model.

Conclusion:

Both Keras and Pytroch model gives average accuracy on predictions of our datasets. But, on implementations, Keras model is easy to use and readable. Also, with different variations in models like epochs values, input layers we get variations of models. So, we can see the performance of model and try to make our model works excellently.

Pytroch model performs well when we have huge amount of dataset. But it needs more additional programming efforts in training and testing phase.

Though, for our dataset Pytorch model works excellent. But, on basis of implementation I found keras model much easy.