Aaron P. Mills / Z-23547104
Dr. Ghoraani
Intro to Deep Learning / CAP 4613
03 April 2022
Assignment 5
https://colab.research.google.com/drive/1j1So0FdoJ1Nwb5m1p_lShFbtKKTvNlhu?usp=sharing
NOTE: Please note that the values in my tables do not precisely match the code because I had to rerun the program for graphs to display in the PDF.

```
# Aaron P. Mills /                Z-23547104 /
# Dr. Ghoraani
# Intro to Deep Learning  /       CAP 4613 /
# Assignment 5             /      03 April 2022
# Discription: build neural networks with keras
# https://colab.research.google.com/drive/1j1So0FdoJ1Nwb5m1p_lShFbtKKTvNlh
u?usp=sharing
##########################################################################
####################
#header block   - includes heading, imports, functions, classes, etc.
#notes: batch size-> train using samples/batch_size
#imports for Problem 1)
import math as mth
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
#imports for Problem 2)
from keras.datasets import mnist
from random import randint
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score
from keras.utils.np_utils import to_categorical
#imports for Problem 3)
import pandas as pd
from sklearn import preprocessing
from google.colab import drive
from keras.callbacks import ModelCheckpoint
from keras.models import load_model
drive.mount('/content/drive')
#functions for Problem 1)
def plot_fun(features,labels,classes):
  plt.plot(features[labels[:]==classes[0],0], features[labels[:]==classes[
0],1], 'rs',
```

```
         features[labels[:]==classes[1],0], features[labels[:]==classes[1]
,1], 'g^',markersize=15)
  #plt.axis([-2,2,-2,2])
  plt.xlabel('x: feature 1')
  plt.ylabel('y: feature 2')
  plt.legend(['Class'+str(classes[0]), 'Class'+str(classes[1])])
  plt.show()

def plot_fun_thr(features,labels,thre_parms,classes):
  #ploting the data points
  plt.plot(features[labels[:]==classes[0],0], features[labels[:]==classes[
0],1], 'rs',
           features[labels[:]==classes[1],0], features[labels[:]==classes[
1],1], 'g^',
           markersize=15)
  #plt.axis([-1,2,-1,2])
  #ploting the seperating line
  x1 = np.linspace(-2,2,50)
  x2 = -(thre_parms[0]*x1+thre_parms[2])/thre_parms[1]
                     #a X1 + b X2 + c=0 --> x2 = -(a X1 + c)/b
  plt.plot(x1, x2, '-r')
  plt.xlabel('x: feature 1')
  plt.ylabel('y: feature 2')
  plt.legend(['Class'+str(classes[0]), 'Class'+str(classes[1])])
  #plt.pause(0.5)
  #plt.show()

def plot_curve(accuracy_train, loss_train):
  epochs=np.arange(loss_train.shape[0])
  plt.subplot(1,2,1)
  plt.plot(epochs,accuracy_train)
  #plt.axis([-1,2,-1,2])
  plt.xlabel('Epoch#')
  plt.ylabel('Accuracy')
  plt.title('Training Accuracy')

  plt.subplot(1,2,2)
  plt.plot(epochs,loss_train)
  plt.xlabel('Epoch#')
  plt.ylabel('Binary crossentropy loss')
  plt.title('Training loss')
  plt.show()

#functions for Problem 2)
def img_plt(images, labels):
```

```
  plt.figure()
  for i in range(1,11):
    plt.subplot(2,5,i)
    plt.imshow(images[i-1,:,:],cmap='gray')
    plt.title('Label: ' + str(labels[i-1]))
  plt.show()

def feat_extract(images):
  width=images.shape[1]     #width of img
  height=images.shape[2]    #height of img
  features=np.zeros((images.shape[0],4))    #empty array with rows=# of im
gs, column = 4 for 4 columns
  #to get each quadrant of img, we must sum up all pixels in quadrant (axi
s=2:)<- across the 3 dimensions
  features_temp=np.sum(images[:,0:int(width/2),0:int(height/2)],axis=2) #q
uadrant 0
  features[:,0]=np.sum(features_temp,axis=1)/(width*height/4)   #avrg
  features_temp=np.sum(images[:,0:int(width/2),int(height/2):],axis=2) #qu
adrant 1
  features[:,1]=np.sum(features_temp,axis=1)/(width*height/4)   #avrg
  features_temp=np.sum(images[:,int(width/2):,0:int(height/2)],axis=2) #qu
adrant 2
  features[:,2]=np.sum(features_temp,axis=1)/(width*height/4)   #avrg
  features_temp=np.sum(images[:,int(width/2):,int(height/2):],axis=2)  #qu
adrant 3
  features[:,3]=np.sum(features_temp,axis=1)/(width*height/4)   #avrg
  return features

def feat_plot(features,labels,classes):
  for class_i in classes:
    plt.plot(features[labels[:]==classes[class_i],0],
             features[labels[:]==classes[class_i],1],'o', markersize=15)
  #plt.axis([-2,2,-2,2])
  plt.xlabel('x: feature 1')
  plt.ylabel('y: feature 2')
  plt.legend(['Class'+str(classes[class_i]) for class_i in classes])
  plt.show()

def acc_fun(labels_actual, labels_pred):
  acc=np.sum(labels_actual==labels_pred)/len(labels_actual)*100
  return acc
```

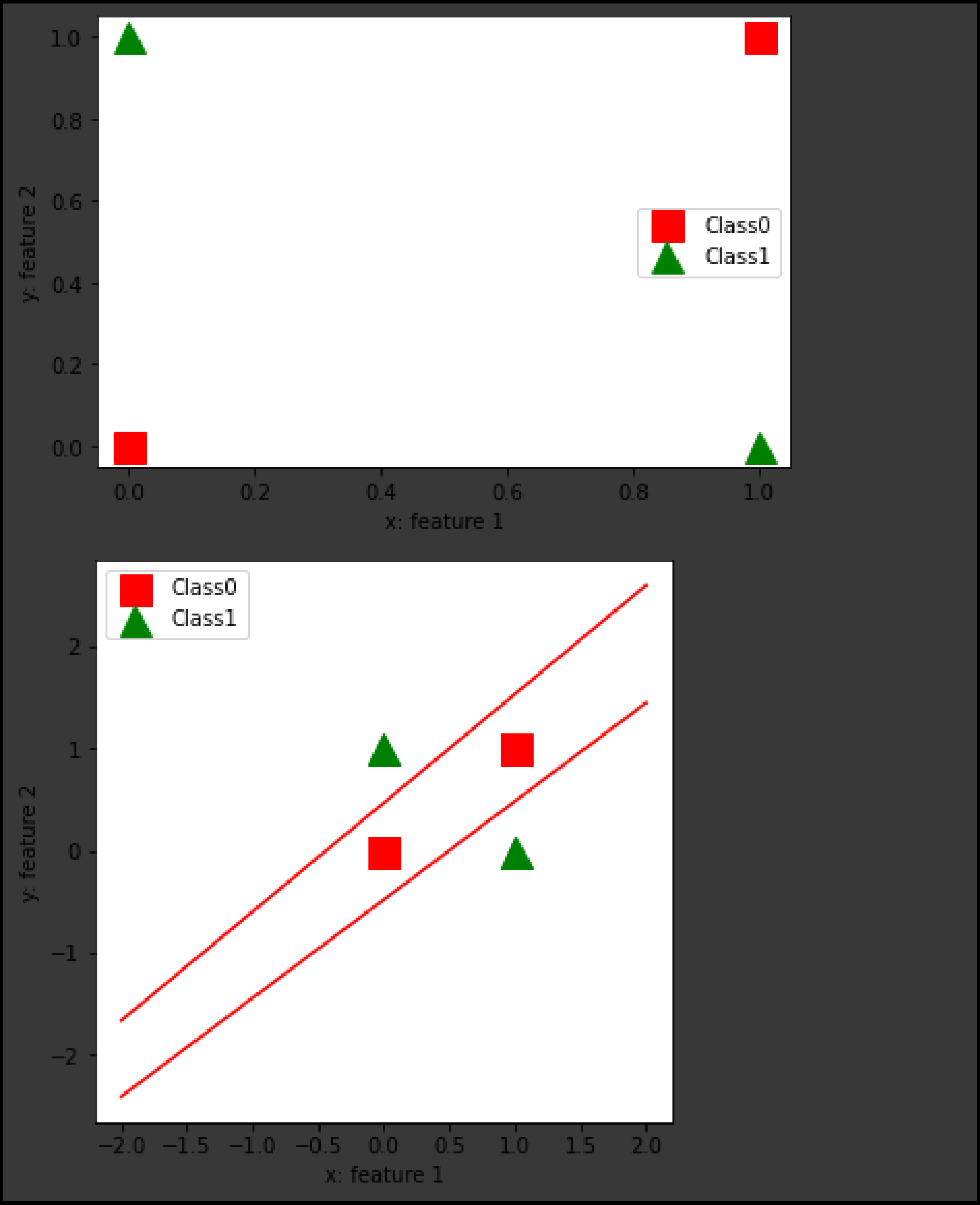
```
Drive already mounted at /content/drive; to attempt to forcibly remount,
call drive.mount("/content/drive", force_remount=True).
```

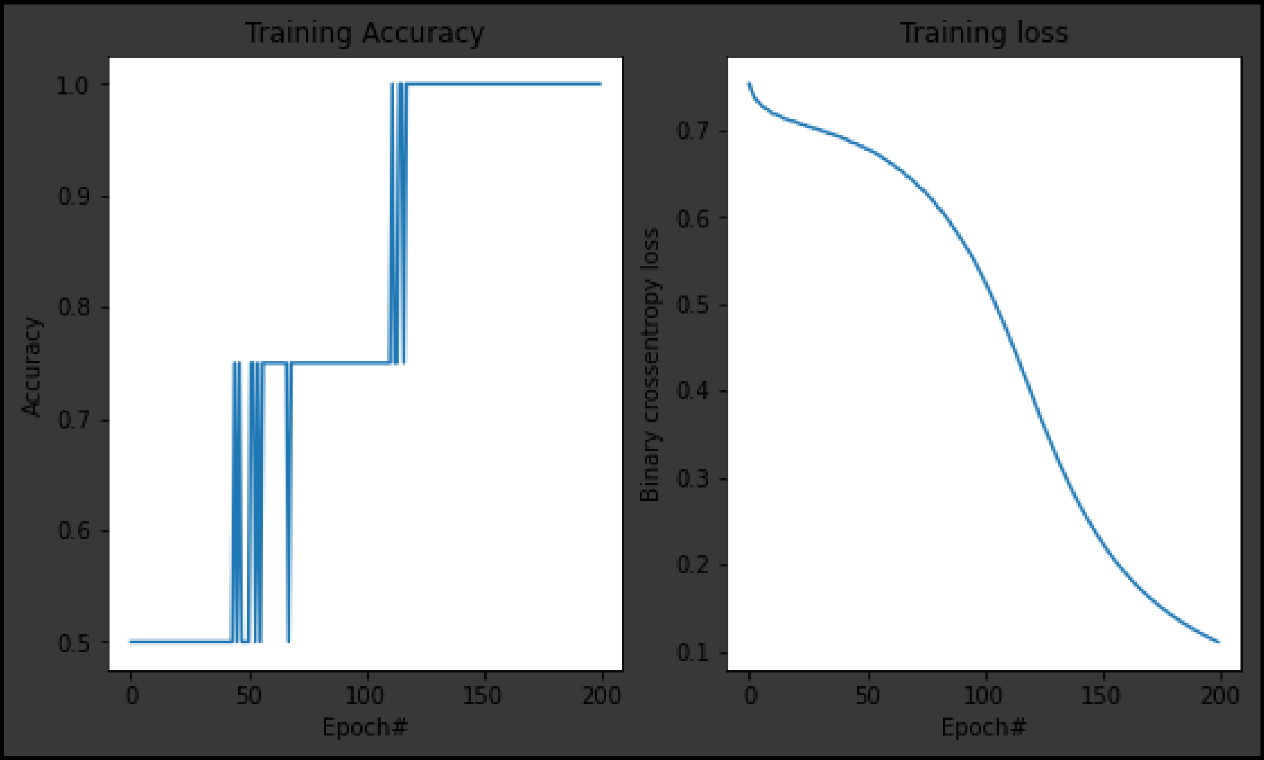
Problem 1) Application of Keras to build, compile, and train a neural network to perform XOR operation.

```
#d) defining/building the model
model_a=Sequential()                                        #sequential: w
hen we have layers next to each other in order side by side ()
model_a.add(Dense(input_dim=2, units=2, activation='tanh')) #.add: add lay
ers to model;dense=defualt layer?;(inputs,units=size of layer, activation
function)
model_a.add(Dense(units=1, activation='sigmoid'))           #^second layer

model_a.summary()                                           #displays (#)l
ayers,output shape, #params in output

#e) compiling the model
opt = tf.keras.optimizers.SGD(learning_rate=0.1)    #defining the optimize
r
model_a.compile(loss='binary_crossentropy',         #loss function used to
 determine how loss is calculated; crossentropy for multiclass problems
              optimizer=opt,                        #the way the network l
earns (ex: gradient descent learning)
              metrics=['accuracy'])                 #model shall train to
maximize metrix (accuracy in this case) by minimizing loss function

#loading the data and normalization, keras computes with bias automaticall
y
features=np.array([ [0,0],[0,1],[1,0],[1,1] ])  #1.a) input array
labels=np.array([0,1,1,0], dtype=np.uint8)      #1.a) labels array
classes=[0,1]

# data normalization is essential for multilayer networks
# we can experiment with and without data normalization and observe the mo
del behavior
# features = (features-
np.mean(features,axis=0))/np.std(features,axis=0) #normalization; {messes
up the graph}
plot_fun(features,labels,classes)               #1.b) graph data
               #plot the data & trained model

#f) train netwrok for 200 epochs with batch size=1
history=model_a.fit(features, labels,   #a.fit: train the model, it return
s values which will be stored in history for this case
```

```
          batch_size=1,                  #keras shall take the first batch_
size data samples together, feed into network, & adjust weights according
to that
          epochs=200,                   ## of epochs
          verbose=0)                     #0: during training, progress will
 NOT be displayed, 1: displays progress

#g) plot the final classifire line
#plot the model / classification li
weights=model_a.layers[0].get_weights()
plt.figure(figsize=[5,5])
for node_i in range(weights[0].shape[1]):
  thre_parms=np.array(weights[0][:,node_i])               #this first item i
s the weights for the inputs
  thre_parms=np.append(thre_parms, weights[1][node_i])  #second item the w
eights for the bias
  plot_fun_thr(features,labels,thre_parms,classes)
plt.show()

#h) plot the learning curve
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

```
Model: "sequential_253"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_804 (Dense)           (None, 2)                 6

 dense_805 (Dense)           (None, 1)                 3

=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```


Class0
Class1
y: feature 2
x: feature 1
Class0
Class1
y: feature 2
x: feature 1



c) Based on the plot from part b), the minimum number of layers is 2 and weights is 9. 2 layers: 1 layer for the separation lines + 1 output layer. 9 parameters = 3x2 + 3x1 = 9. 3 inputs (including bias) have 2 nodes for Layer 1;3 inputs from Layer 1 (including bias) have 1 input from Layer 2; add products to get total nodes. Layer 1 has 2 parameters, because the data shall be separated by 2 lines. Layer 2 has 1 parameter because that 1 parameter shall be used to determine whether output is class 0 or 1; there's also only 1 region

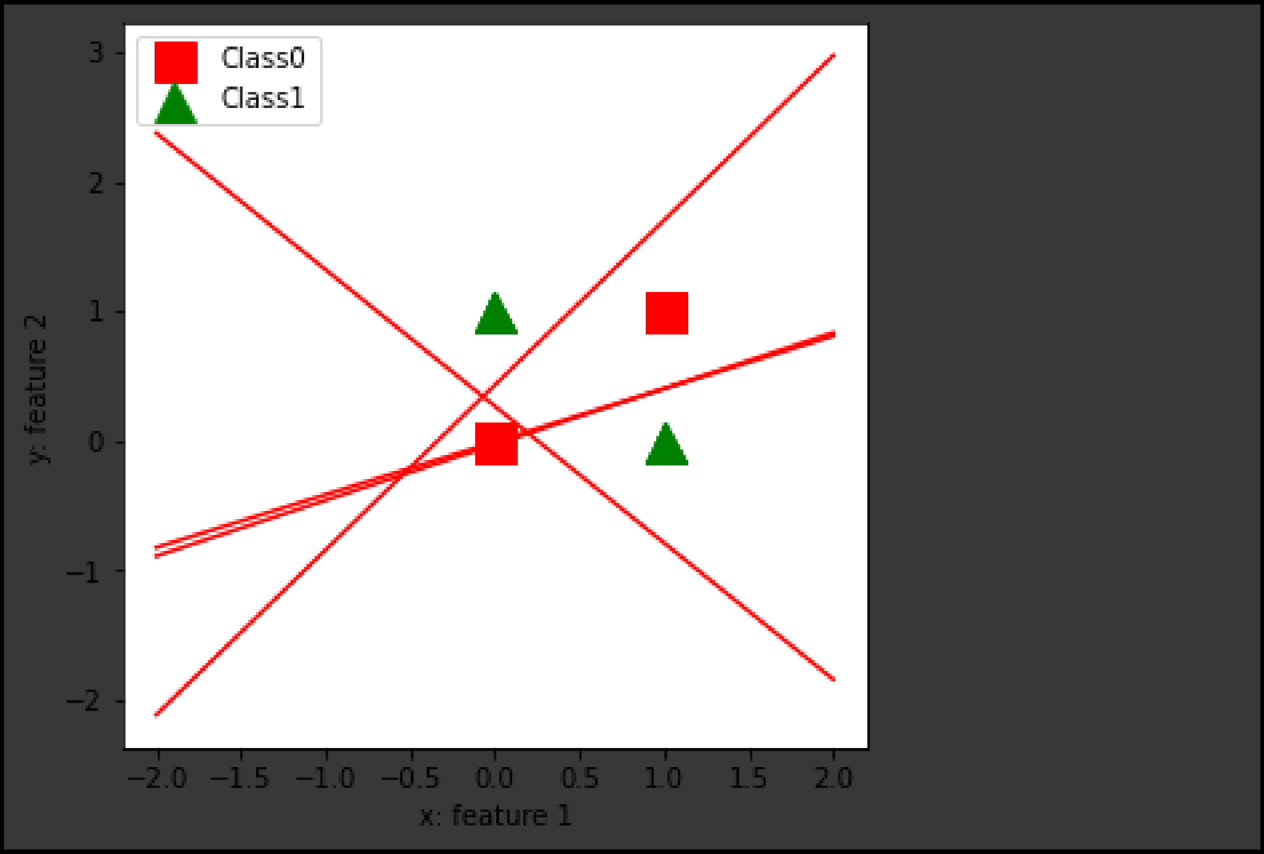
---

e) The 'binary_crossentropy' loss function was selected due to its relationship & good performance with multiclass classification problems.

```
#i) Repeat steps (d) to (g) after adding 2 more nodes to the first layer a
nd training for 400 epochs

#d2) defining/building the model; increase nodes in FIRST layer
model_a=Sequential()                                        #sequential: w
hen we have layers next to each other in order side by side ()
model_a.add(Dense(input_dim=2, units=4, activation='tanh')) #.add: add lay
ers to model;dense=defualt layer?;(inputs,units=size of layer, activation
function)
model_a.add(Dense(units=1, activation='sigmoid'))           #^second layer

model_a.summary()                                           #displays (#)l
ayers,output shape, #params in output
```

```
#compiling the model
opt = tf.keras.optimizers.SGD(learning_rate=0.1)     #defining the optimize
r
model_a.compile(loss='binary_crossentropy',          #loss function used to
 determine how loss is calculated; crossentropy for multiclass problems
              optimizer=opt,                         #the way the network l
earns (ex: gradient descent learning)
              metrics=['accuracy'])                  #model shall train to
maximize metrix (accuracy in this case) by minimizing loss function

#train netwrok for 400 epochs with batch size=1
history=model_a.fit(features, labels,    #a.fit: train the model, it return
s values which will be stored in history for this case
          batch_size=1,                  #keras shall take the first batch_
size data samples together, feed into network, & adjust weights according
to that
          epochs=400,                   ## of epochs
          verbose=0)                     #0: during training, progress will
 NOT be displayed, 1: displays progress

#g) plot the final classifire line
#plot the model / classification li
weights=model_a.layers[0].get_weights()
plt.figure(figsize=[5,5])
for node_i in range(weights[0].shape[1]):
  thre_parms=np.array(weights[0][:,node_i])            #this first item i
s the weights for the inputs
  thre_parms=np.append(thre_parms, weights[1][node_i])  #second item the w
eights for the bias
  plot_fun_thr(features,labels,thre_parms,classes)
plt.show()
```

```
Model: "sequential_254"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_806 (Dense)           (None, 4)                 12

 dense_807 (Dense)           (None, 1)                 5

=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
```



j) It is observed that the model performs awkwardly when an additional node is added to Layer 1. For this reason, it is suitable to use 9 nodes for this exercise, distributed the same way they were part c).

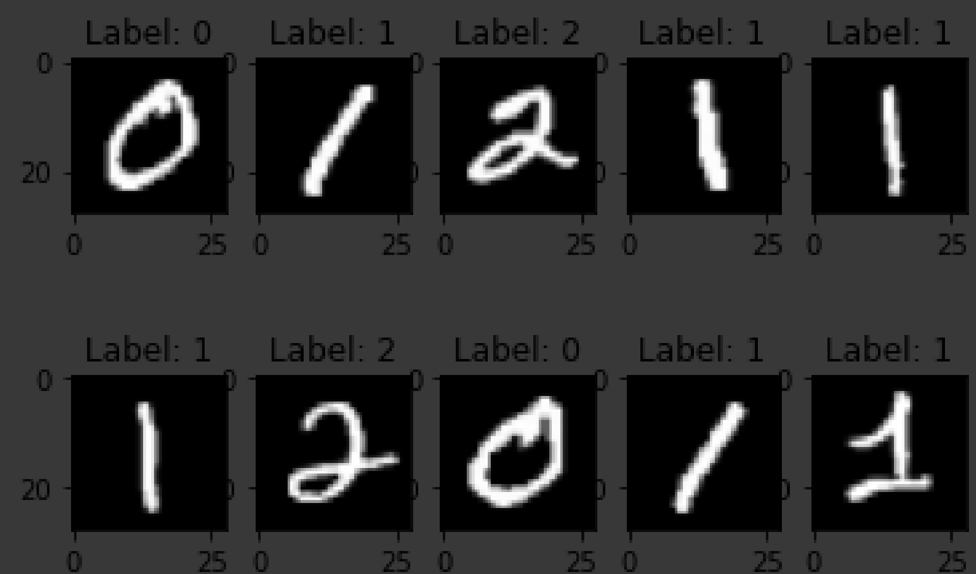
Problem 2) Application of Keras to build, compile, and train a neural network as a three-class classifier for MNIST dataset (0 vs. 1 vs. 2).

```
#a) use mnist function in keras.datasets to load MNIST dataset & split it
into training and testing sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()          #60k t
raining imgs, 10k testing imgs, x=data,y=label

#selecting only 0,1, & 2 digits from the training and testing sets
classes=[0,1,2]
              #desired numbers
x_train_012=x_train[np.logical_or.reduce((y_train==0,y_train==1,y_train==2
)),0:28,0:28] #=img data [where label is 0,1,or 2] with size (0:28,0:28) p
ixels
```

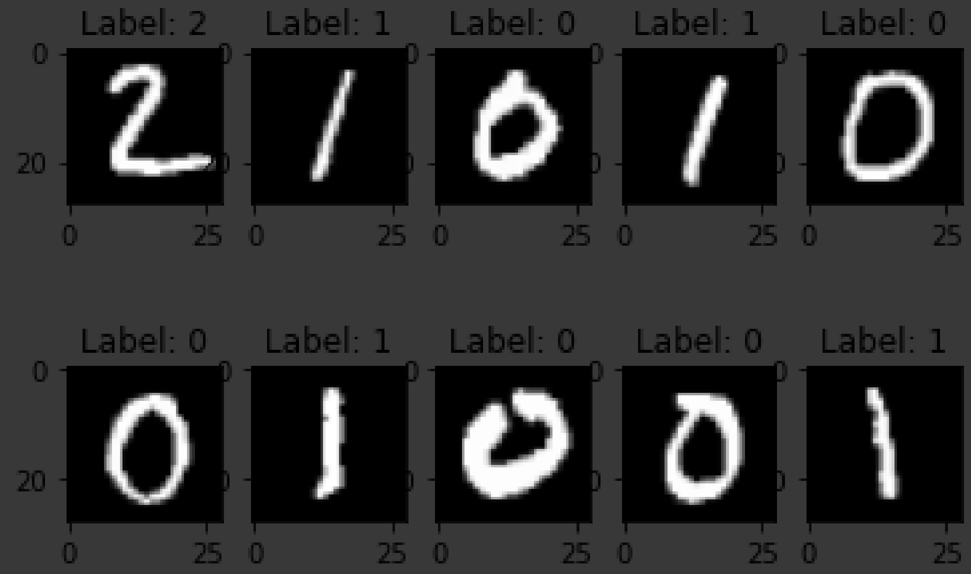
```
y_train_012=y_train[np.logical_or.reduce((y_train==0,y_train==1,y_train==2
))]            #^ corresponding labels
print('Samples of the training images')

img_plt(x_train_012[0:10,:,:],y_train_012[0:10])
              #plot img with labels

x_test_012=x_test[np.logical_or.reduce((y_test==0,y_test==1,y_test==2)),0:
28,0:28]      #=img data [where label is 0,1,or 2] with size (0:28,0:28) p
ixels
y_test_012=y_test[np.logical_or.reduce((y_test==0,y_test==1,y_test==2))]
              #^ corresponding labels
print('Samples of the testing images')
img_plt(x_test_012[0:10,:,:],y_test_012[0:10])
              #plot img with labels

#selecting number images from training data as the validation set
#shuffling trainig data
num_train_img=x_train_012.shape[0]          #number of training imgs
train_ind=np.arange(0,num_train_img)        #indx = 1 dimensional array of
 indices from 0 to number of data
train_ind_s=np.random.permutation(train_ind)#shuffle the indexes of the da
ta
x_train_012=x_train_012[train_ind_s,:,:]    #new data = data[shuffled indx
] = shuffled data
y_train_012=y_train_012[train_ind_s]        #corresponding labels
#selecting number of random images for validation
#a) randomly select 20% of the training images along with their correspond
ing labels to be the validation data.
indx = round(0.20*x_train_012.shape[0])     #indx upper bound of 20% of tr
aining imgs
x_val_012=x_train_012[0:indx,:,:]           #validation = {indx} RANDOM (s
huffled) imgs of train set
y_val_012=y_train_012[0:indx]               #^ coressponding labels
#the rest of the training set
x_train_012=x_train_012[indx:,:,:]          #remove validation from traini
ng set
y_train_012=y_train_012[indx:]              #corresponding labels
print('Samples of the validation images')
img_plt(x_val_012[0:10,:,:],y_val_012[0:10])    #plot validation imgs with
 labels

#b) feature extraction: avrg pixel values in quadrants of each img to make
 a feature vector of 4 values for each img
```

```
#calculating the training, validation and testing feature (average of the
four quadrants grid)
feature_train=feat_extract(x_train_012) #get avrg for all 4 quadrants for
train set
feature_val=feat_extract(x_val_012)      #^ for validation set
feature_test=feat_extract(x_test_012)    #^ for testing set

#c) convert the label vectors for all sets to binary class matrices using
to_categorical() Keras function
#convert class labels vectors to binary class matrices
y_train_012_c = to_categorical(y_train_012, len(classes))    #EX: Labels =
[2,0,1]->conversion=[[0,0,1],[1,0,0],[0,1,0]]
y_val_012_c = to_categorical(y_val_012, len(classes))        #^cols=# of cl
asses, row=labels[indx=row]
y_test_012_c = to_categorical(y_test_012, len(classes))      #^put a 1 in m
atrix where labels[i]=col
#plotting features
print('Plotting the features of 500 training images: ')
feat_plot(feature_train[1:500, 0:2],y_train_012[1:500], classes)     #avrg
of quadrants 0 with respect to quadrant 1 of training set
feat_plot(feature_train[1:500, 2:4],y_train_012[1:500], classes)     #quadr
ants 2 & 3 ^
#the combination between the features could be changed
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Samples of the training images
```

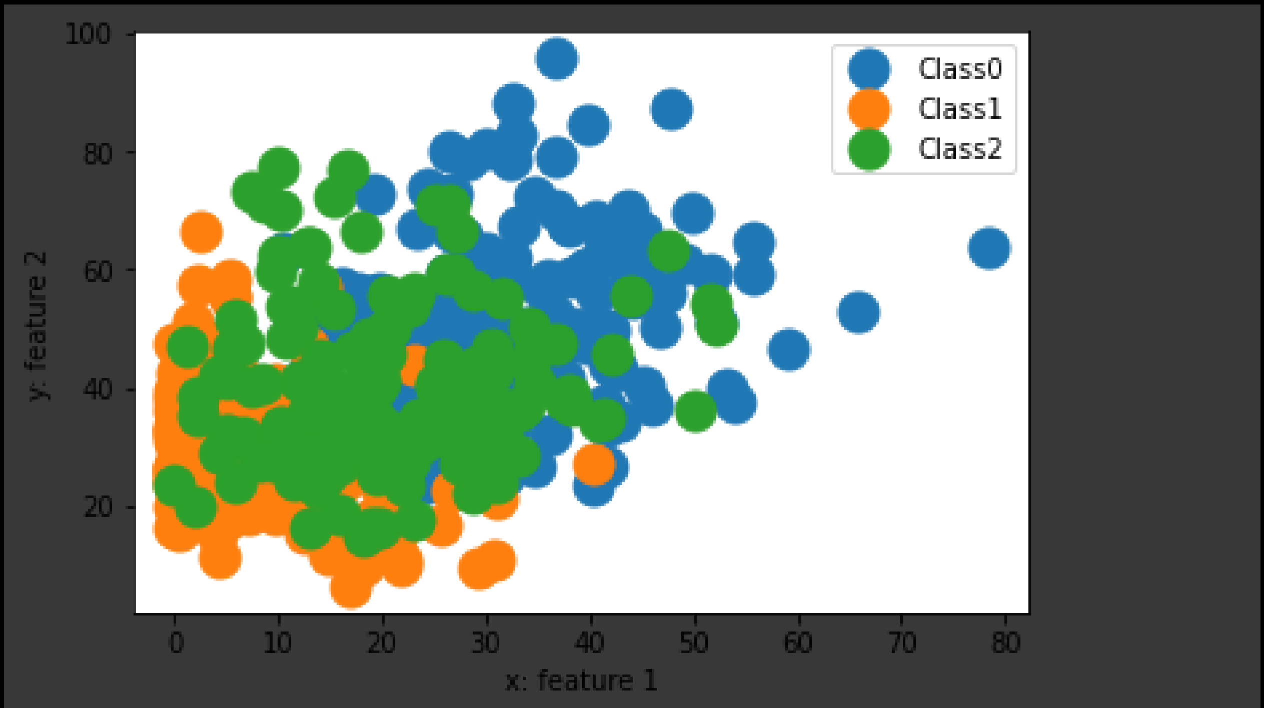


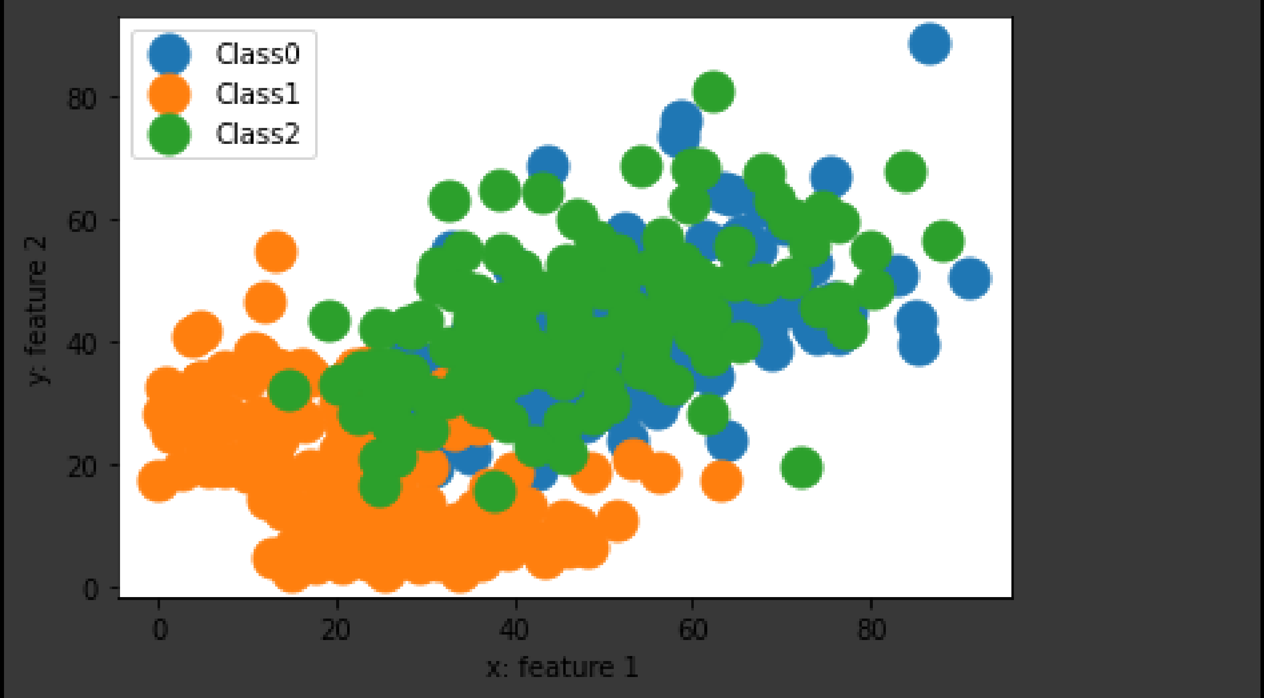

```
Samples of the testing images
```



Samples of the validation images

Plotting the features of 500 training images:



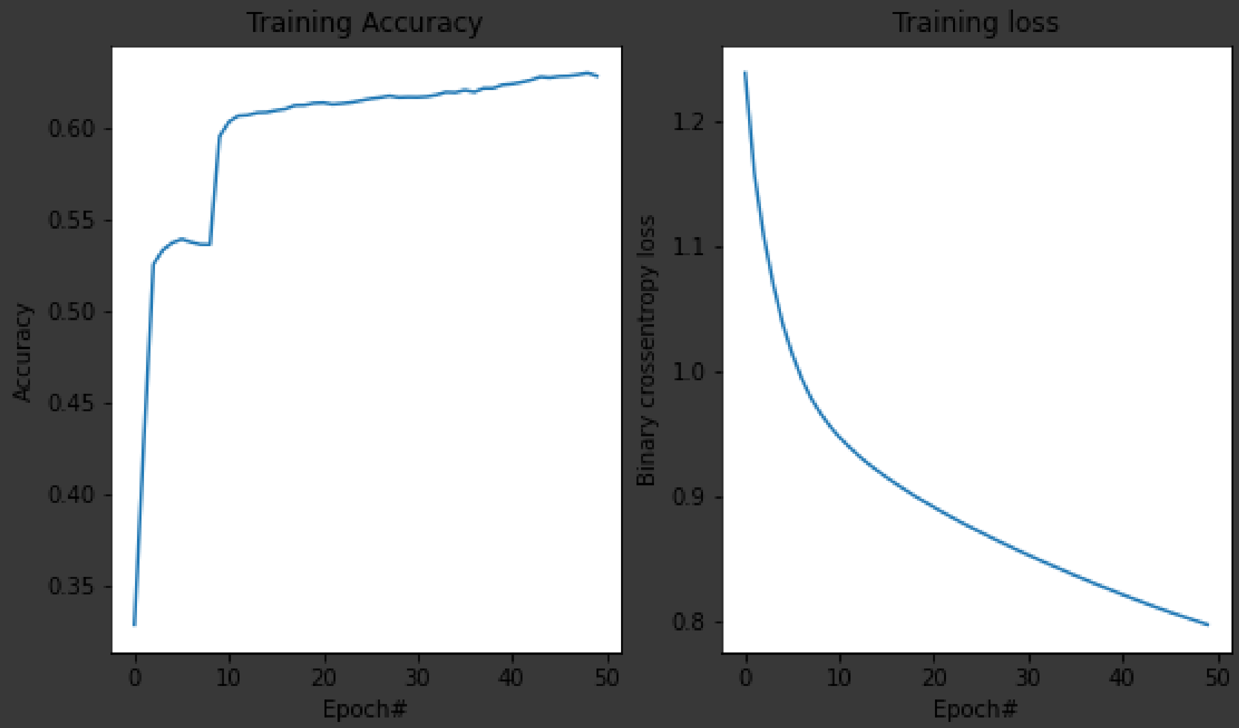



```
#d) build, compile, train, and then evaluate
#defining the model
model_a=Sequential()     #sequential: when we have layers next to each othe
r in order side by side ()
model_a.add(Dense(input_dim=4, units=10, activation='tanh'))     #d.i) 1 la
yer: 10 nodes, 4 inputs bcuz 4 features
```

```
# model_a.add(Dense(units=10, activation='tanh'))                 #2nd lay
er: 10 nodes
model_a.add(Dense(units=len(classes), activation='softmax'))    #output la
yer: len(class) neurons
model_a.summary()                                               #displays
(#)layers,output shape, #params in output

#d.ii) compile network: select a correct loss function for this problem; u
se (SGD,learning rate of 0.0001).
#compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.0001) #defining the optimize
r using stochastic gradient descent (SGD)
model_a.compile(loss='categorical_crossentropy',    #loss function used to
 determine how loss is calculated; crossentropy for multiclass problems

              optimizer=opt,                        #the way the network l
earns (ex: gradient descent learning)
              metrics=['accuracy'])                 #model shall train to
maximize metrix (accuracy in this case) by minimizing loss function

#d.iii) train the network for 50 epochs & batch size of 16.
#train the model
history=model_a.fit(feature_train, y_train_012_c,   #a.fit: train the mode
l, it returns values which will be stored in history for this case
          batch_size=16,                            #keras shall take the
first {batch_size} data samples together, feed into network, & adjust weig
hts according to that
          epochs=50,                                #number of epochs
          verbose=0)                                #0: during training, p
rogress will NOT be displayed, 1: displays progress

#d)v. use the evaluate() Keras function to find training & validation loss
 & accuracy
#evaluating the model on the training samples
score=model_a.evaluate(feature_train,y_train_012_c) #evaluate model agains
t training data set
print('Total loss on training set: ', score[0])
print('Accuracy of training set: ', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(feature_val,y_val_012_c)     #evaluate model agains
t validation data set
print('Total loss on validation set: ', score[0])
print('Accuracy of validation set: ', score[1])
```

```
#d.iv) plot the training loss (i.e., the learning curve) for all the epoch
s
#plot final results
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

```
Model: "sequential_255"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_808 (Dense)           (None, 10)                50

 dense_809 (Dense)           (None, 3)                 33

=================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
_________________________________________________________________
466/466 [==============================] - 1s 1ms/step - loss: 0.7965 -
accuracy: 0.6308
Total loss on training set:  0.7964867949485779
Accuracy of training set:  0.6308229565620422
117/117 [==============================] - 0s 1ms/step - loss: 0.7852 -
accuracy: 0.6365
Total loss on validation set:  0.7852145433425903
Accuracy of validation set:  0.6365100741386414
```



d.ii) The 'binary_crossentropy' loss function was selected due to its relationship & good performance with multiclass classification problems.

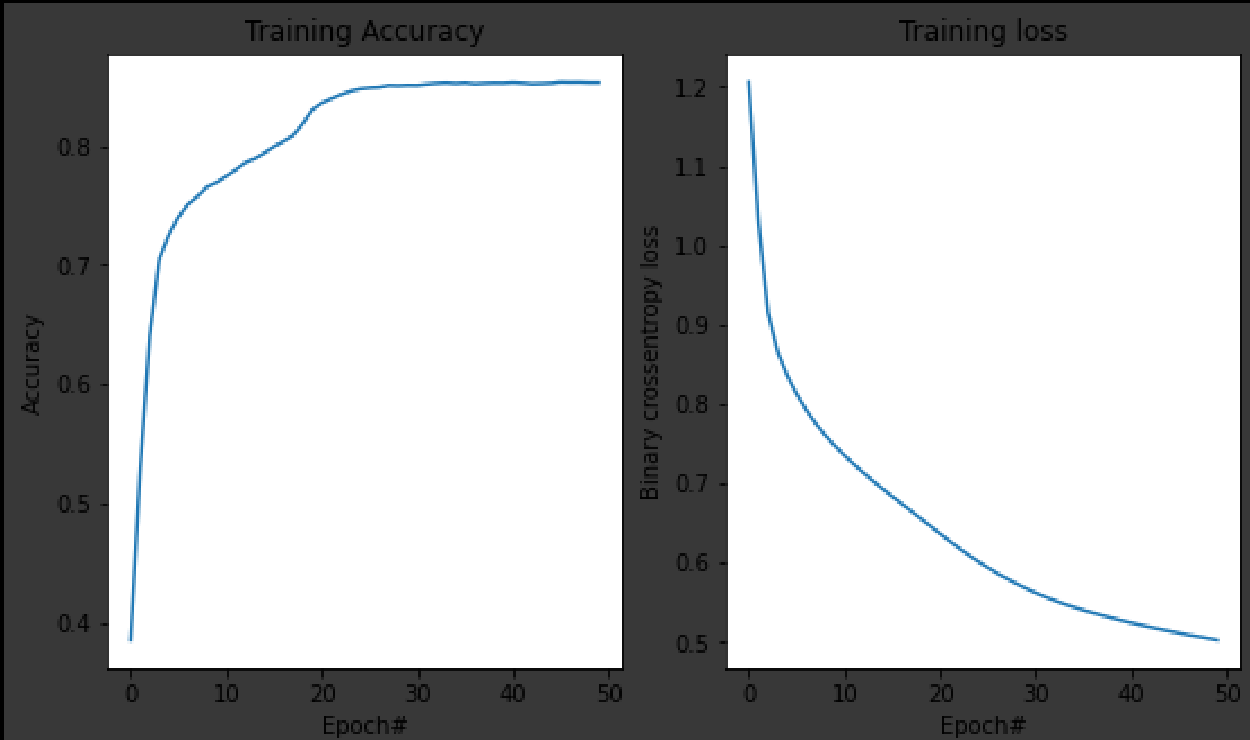
```
#e) repeat step d) for each of the given networks

print("1 layer\t 10 nodes *************************************************
*************************")
#defining the model
model_a=Sequential()                                                #new seque
ntial model
model_a.add(Dense(input_dim=4, units=10, activation='tanh'))    #layer 1
model_a.add(Dense(units=len(classes), activation='softmax'))    #output la
yer: len(class) neurons
# model_a.summary()
#compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
model_a.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['ac
curacy'])
#train the model
history=model_a.fit(feature_train, y_train_012_c, batch_size=16,epochs=50,
verbose=0)
#evaluating the model on the training samples
score=model_a.evaluate(feature_train,y_train_012_c)
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(feature_val,y_val_012_c)
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
#plot final results
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
#e) repeat step d) for each of the given networks
print("1 layer\t 50 nodes *************************************************
*************************")
#defining the model
model_a=Sequential()                                                #new seque
ntial model
model_a.add(Dense(input_dim=4, units=50, activation='tanh'))    #layer 1
model_a.add(Dense(units=len(classes), activation='softmax'))    #output la
yer: len(class) neurons
# model_a.summary()
#compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
```

```
model_a.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['ac
curacy'])
#train the model
history=model_a.fit(feature_train, y_train_012_c, batch_size=16,epochs=50,
verbose=0)
#evaluating the model on the training samples
score=model_a.evaluate(feature_train,y_train_012_c)
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(feature_val,y_val_012_c)
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
#plot final results
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
#e) repeat step d) for each of the given networks
print("1 layer\t 100 nodes ***********************************************
****************************")
#defining the model
model_a=Sequential()                                              #new seque
ntial model
model_a.add(Dense(input_dim=4, units=100, activation='tanh'))    #layer 1
model_a.add(Dense(units=len(classes), activation='softmax'))    #output la
yer: len(class) neurons
# model_a.summary()
#compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
model_a.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['ac
curacy'])
#train the model
history=model_a.fit(feature_train, y_train_012_c, batch_size=16,epochs=50,
verbose=0)
#evaluating the model on the training samples
score=model_a.evaluate(feature_train,y_train_012_c)
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(feature_val,y_val_012_c)
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
#plot final results
plt.figure(figsize=[9,5])
```

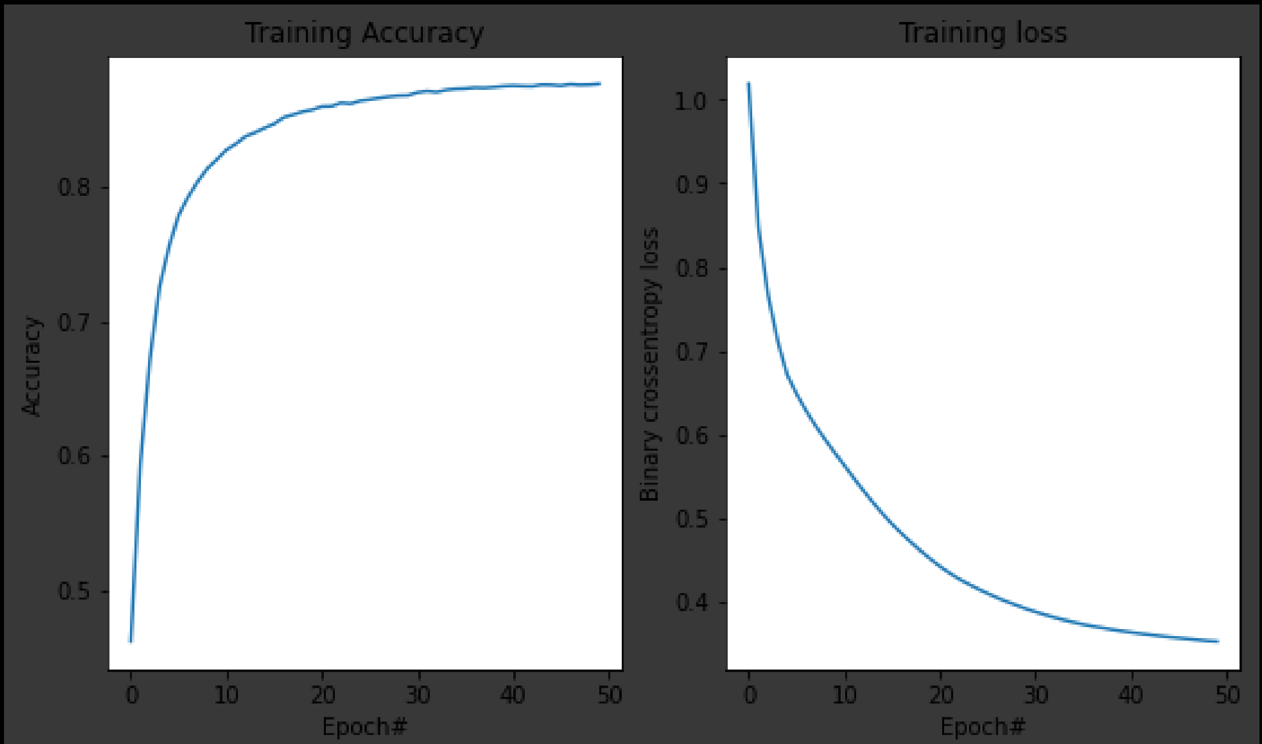
```
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
#e) repeat step d) for each of the given networks
print("2 layers\t 100 nodes\t 10 nodes ***********************************
****************************************")
#defining the model
model_a=Sequential()                                                #new seque
ntial model
model_a.add(Dense(input_dim=4, units=100, activation='tanh'))      #layer 1
model_a.add(Dense(units=10, activation='tanh'))                     #layer 2
model_a.add(Dense(units=len(classes), activation='softmax'))       #output la
yer: len(class) neurons
# model_a.summary()
#compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
model_a.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['ac
curacy'])
#train the model
history=model_a.fit(feature_train, y_train_012_c, batch_size=16,epochs=50,
verbose=0)
#evaluating the model on the training samples
score=model_a.evaluate(feature_train,y_train_012_c)
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(feature_val,y_val_012_c)
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
#plot final results
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
#e) repeat step d) for each of the given networks
print("2 layers\t 100 nodes\t 50 nodes ***********************************
****************************************")
#defining the model
model_a=Sequential()                                                #new seque
ntial model
model_a.add(Dense(input_dim=4, units=100, activation='tanh'))      #layer 1
model_a.add(Dense(units=50, activation='tanh'))                     #layer 2
model_a.add(Dense(units=len(classes), activation='softmax'))       #output la
yer: len(class) neurons
# model_a.summary()
```

```
#compile the model
opt = tf.keras.optimizers.SGD(learning_rate=0.0001)
model_a.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['ac
curacy'])
#train the model
history=model_a.fit(feature_train, y_train_012_c, batch_size=16,epochs=50,
verbose=0)
#evaluating the model on the training samples
score=model_a.evaluate(feature_train,y_train_012_c)
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(feature_val,y_val_012_c)
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
#plot final results
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

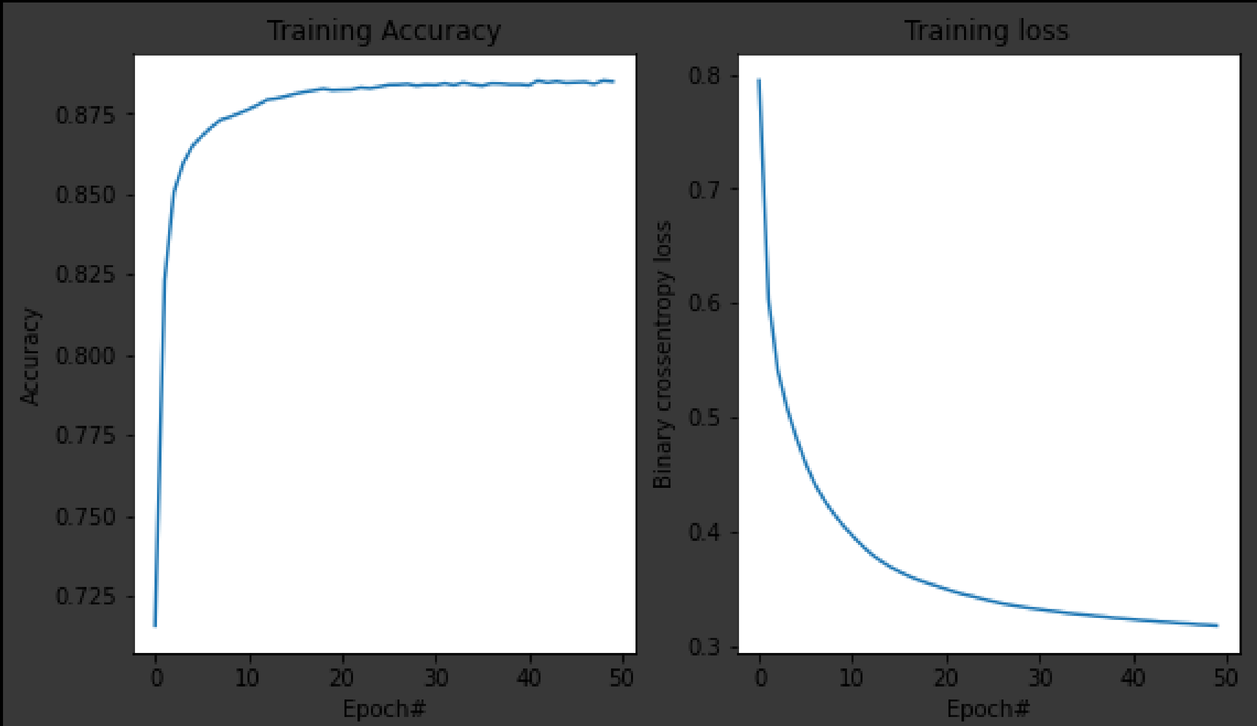
```
1 layer  10 nodes
*************************************************************************
466/466 [==============================] - 1s 1ms/step - loss: 0.5013 -
accuracy: 0.8541
Total loss on training set:     0.501322865486145
Accuracy of training set:       0.8540743589401245
117/117 [==============================] - 0s 1ms/step - loss: 0.4799 -
accuracy: 0.8690
Total loss on validation set:   0.4798869490623474
Accuracy of validation set:     0.8689932823181152
```



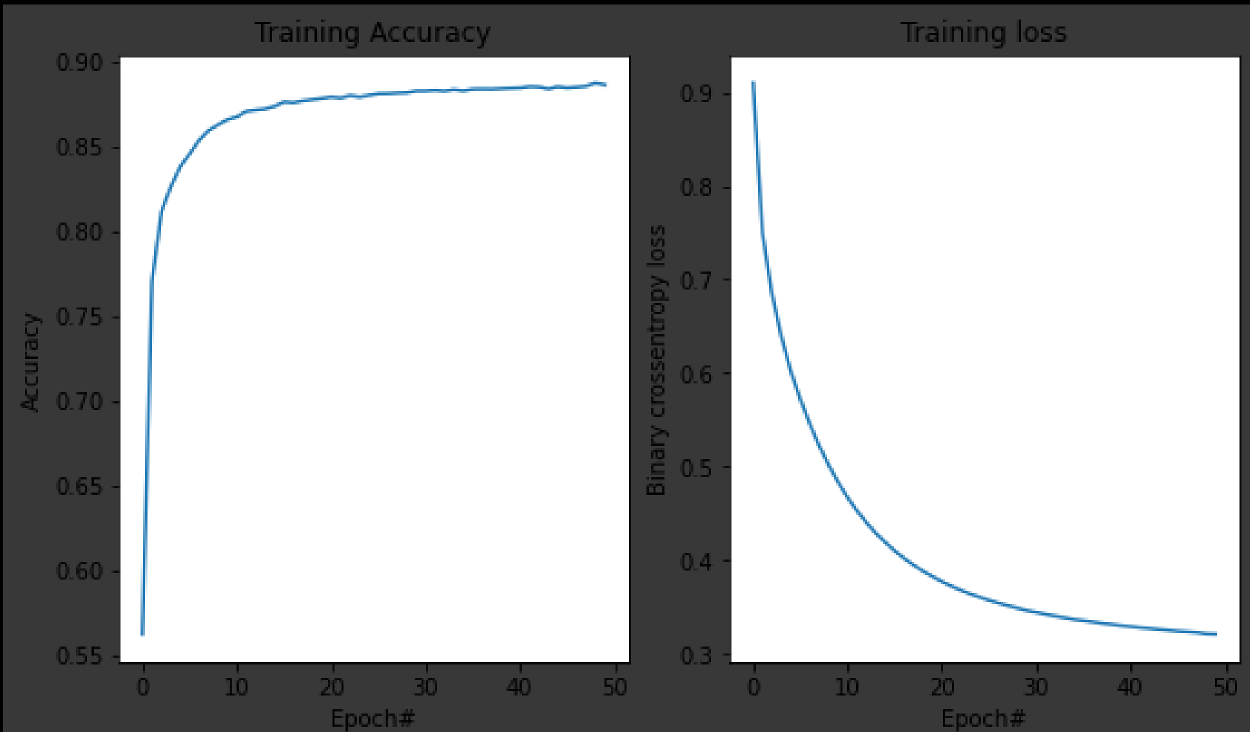

```
1 layer  50 nodes
***************************************************************************
466/466 [==============================] - 1s 2ms/step - loss: 0.3534 -
accuracy: 0.8758
Total loss on training set:     0.353436678647995
Accuracy of training set:       0.8757551312446594
117/117 [==============================] - 0s 2ms/step - loss: 0.3270 -
accuracy: 0.8843
Total loss on validation set:   0.32699376344680786
Accuracy of validation set:     0.8842952847480774
```



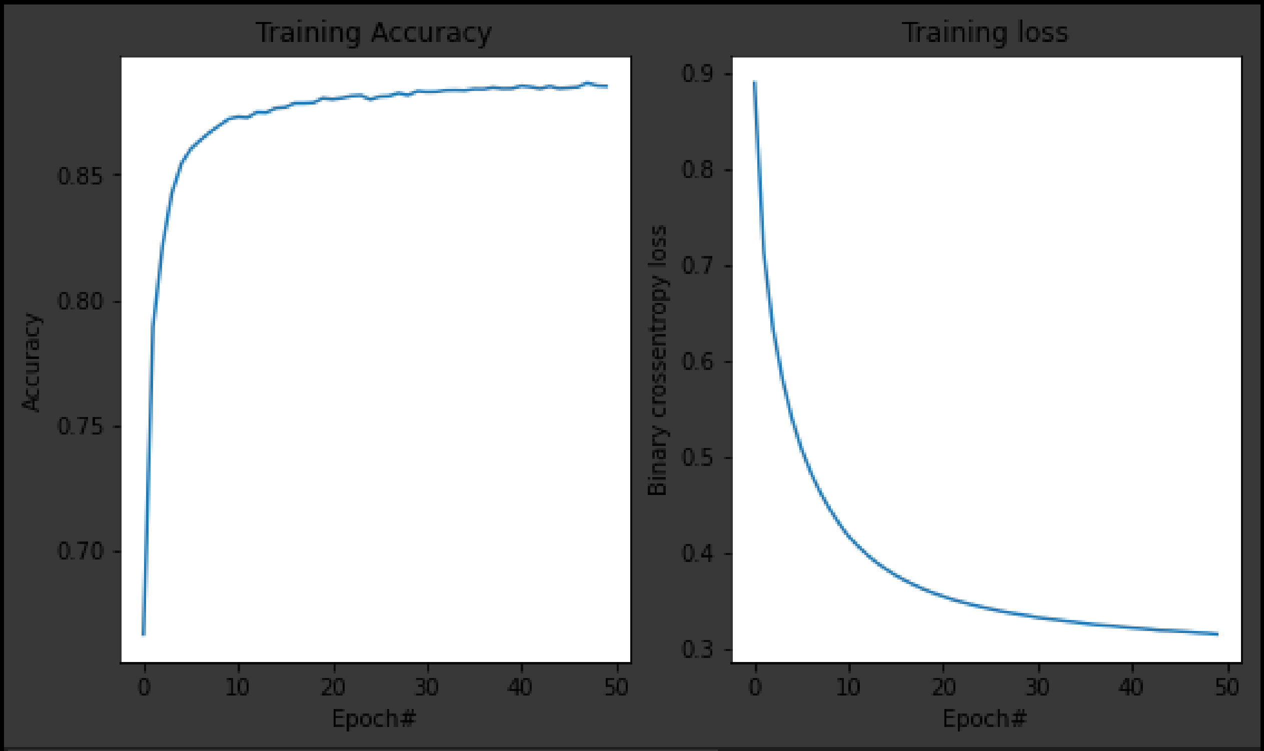

```
1 layer  100 nodes
*****************************************************************************
466/466 [==============================] - 1s 1ms/step - loss: 0.3169 -
accuracy: 0.8856
Total loss on training set:     0.3169096112251282
Accuracy of training set:       0.88555508852005
117/117 [==============================] - 0s 1ms/step - loss: 0.2914 -
accuracy: 0.8983
Total loss on validation set:   0.29139086604118347
Accuracy of validation set:     0.8982550501823425
```




```
2 layers        100 nodes      10 nodes
***************************************************************************
466/466 [==============================] - 1s 2ms/step - loss: 0.3201 -
accuracy: 0.8871
Total loss on training set:     0.32006141543388367
Accuracy of training set:       0.8870989680290222
117/117 [==============================] - 0s 2ms/step - loss: 0.2943 -
accuracy: 0.8980
Total loss on validation set:   0.29428717494010925
Accuracy of validation set:     0.8979865908622742
```




```
2 layers        100 nodes      50 nodes
***************************************************************************
466/466 [==============================] - 1s 2ms/step - loss: 0.3141 -
accuracy: 0.8855
Total loss on training set:     0.31406116485595703
Accuracy of training set:       0.885487973690033
117/117 [==============================] - 0s 2ms/step - loss: 0.2879 -
accuracy: 0.8974
Total loss on validation set:   0.28791508078575134
Accuracy of validation set:     0.8974496722221375
```



e) Please see table: https://drive.google.com/file/d/1j-vPAEAEnlDbB1sVX_zFxj7kod4ttktj/view?usp=sharing
Note that the percentages are rounded.

#	Layers	Nodes	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1	1	10	59.84	85.3	60.31	85.18
2	1	50	34.65	88.13	34.37	88.03
3	1	100	31.38	88.76	31.03	88.56
4	2	100,10	33.03	88.74	32.76	89.02
5	2	100,50	30.19	88.92	29.69	89.05

f) I observe that as the layers & nodes increase, both training & validation losses decrease, while the corresponding accuracies increase. Thus, model 5 is most suitable for this problem, as the validation loss is lowest and validation accuracy is highest. This implies the model has least errors and best peformance of the 5 models.

```
#g) evaluating the model on the held-out samples

score=model_a.evaluate(feature_test,y_test_012_c)    #evaluate model against testing data set
print('Total loss on testing set:\t', score[0])
print('Accuracy of testing set:\t', score[1])

#predicting the class of the held-out samples - NOT required
# test_class1_prob=model_a.predict(feature_test)      #predict the testing set
# test_lab=np.argmax(test_class1_prob,axis=1)
```

```
# print('The accuracy using the testing set: ',        #display accuracy of
 the test prediction
#        accuracy_score(test_lab,y_test_012))
# conf_mat=confusion_matrix(test_lab,y_test_012)       #confusion matrix
# print('The confusion matrix using testing set: \n',
#        conf_mat)
```

```
99/99 [==============================] - 0s 2ms/step - loss: 0.3307 -
accuracy: 0.8783
Total loss on testing set:    0.33066874742507935
Accuracy of testing set:      0.8782967925071716
```

Problem 3) Application of Keras to build, compile, and train a neural network to classify songs from Spotify dataset.

```
#a) import spotify_preprocessed.csv into code
#import data
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/spotify_preproc
essed.csv')
data = np.array(data)   #convert to matrix format
classes =[0,1]
# print(f"Here's the spotify data:↓ Size:{data.shape}\n{data}")

#b) shuffle data, split into sets: 90% training, 10% testing, (80%training
)=training set,(20%training)=validation
#shuffle all data
num_sng = data.shape[0]                          #total number of music tracks
sng_ind = np.arange(0,num_sng)                 #indx variable = 1 dimensional arr
ay of indices from 0 to number of data
sng_ind_s=np.random.permutation(sng_ind)#shuffle the indexes of the data
data=data[sng_ind_s,:]                         #new data = data[shuffled indx] =
shuffled data
# print(f"Here's the spotify data:↓ Size:{data.shape}\n{data}")
#separate features from labels
x_sng = data[:,0:-1]                   #features of every sample
y_sng = data[:,-1]                     #^corresponding labels
classes = [0,1]
# print(f"{x_sng}\n{y_sng}")
#split data
indx = round(0.90*num_sng)             #90% samples are for training
x_train_sng = x_sng[0:indx,:]          #{indx}% samples for training
y_train_sng = y_sng[0:indx]            #^corresponding samples
#testing set
x_test_sng = x_sng[indx:,:]            #the rest of the samples
```

```
y_test_sng = y_sng[indx:]         #^corresponding labels
#validation set
num_train_sng = x_train_sng.shape[0]#number of training samples
indx = round(0.20*num_train_sng)    #20% of training samples are for valid
ation
x_val_sng = x_train_sng[0:indx,:]   #^features of validation set
y_val_sng = y_train_sng[0:indx]      #^corresponding labels
#training set
x_train_sng = x_train_sng[indx:,:]  #the rest of the training samples
y_train_sng = y_train_sng[indx:]     #^corresponding labels
# print(f"feature size:test:{x_test_sng.shape[0]};val:{x_val_sng.shape[0]}
;train:{x_train_sng.shape[0]}")
# print(f"labels size:test:{y_test_sng.shape[0]};val:{y_val_sng.shape[0]};
train:{y_train_sng.shape[0]}")
# print(num_sng)

#c) build, compile, train, and then evaluate; c.i) 2 hidden layers, 32 nod
es each, ouput layer, 1 unit
model_a=Sequential()                                        #sequential: w
hen we have layers next to each other in order side by side ()
model_a.add(Dense(input_dim=15, units=32, activation='relu'))#.add: add la
yers to model;dense=defualt layer?;(inputs=#features,units=size of layer,
activation function)
model_a.add(Dense(units=32, activation='relu'))             #2nd hidden la
yer
model_a.add(Dense(units=1, activation='sigmoid'))           #^second layer

model_a.summary()                                           #displays (#)l
ayers,output shape, #params in output

#c.ii) compiling the model with SGD,binary_crossentropy, l_rate=0.01
opt = tf.keras.optimizers.SGD(learning_rate=0.01)   #defining the optimize
r
model_a.compile(loss='binary_crossentropy',         #loss function used to
 determine how loss is calculated; crossentropy for multiclass problems
              optimizer=opt,                        #the way the network l
earns (ex: gradient descent learning)
              metrics=['accuracy'])                 #model shall train to
maximize metrix (accuracy in this case) by minimizing loss function

features_train = x_train_sng
labels_train = y_train_sng
#c.iii) train network with batc size of 16 samples for 50 epochs
history=model_a.fit(features_train, labels_train,   #a.fit: train the mode
l, it returns values which will be stored in history for this case
```

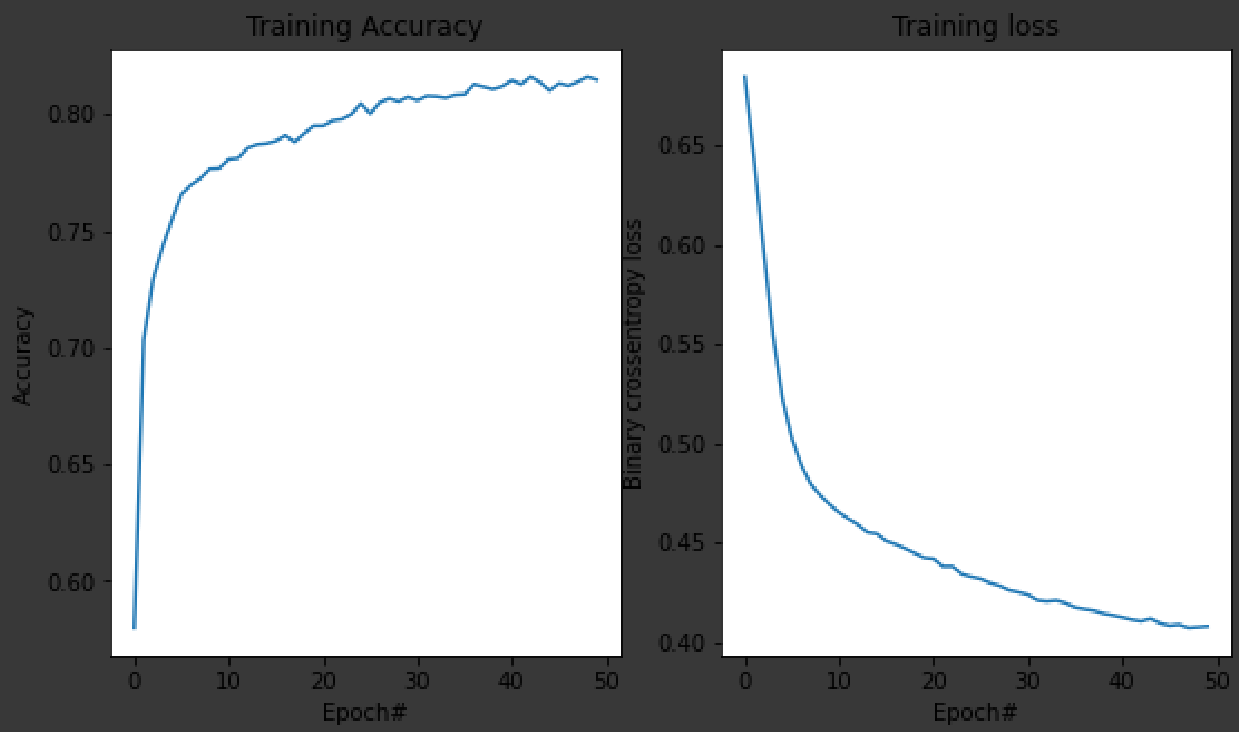
```
          batch_size=16,                                #keras shall take the 
first batch_size data samples together, feed into network, & adjust weight
s according to that
          epochs=50,                                    ## of epochs
          verbose=1)                                    #0: during training, p
rogress will NOT be displayed, 1: displays progress

#c.iv) plot training & validation loss
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

```
Model: "sequential_261"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_822 (Dense)           (None, 32)                512       
                                                                 
 dense_823 (Dense)           (None, 32)                1056      
                                                                 
 dense_824 (Dense)           (None, 1)                 33        
                                                                 
=================================================================
Total params: 1,601
Trainable params: 1,601
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
288/288 [==============================] - 1s 2ms/step - loss: 0.6848 - 
accuracy: 0.5799
Epoch 2/50
288/288 [==============================] - 0s 2ms/step - loss: 0.6426 - 
accuracy: 0.7034
Epoch 3/50
288/288 [==============================] - 0s 2ms/step - loss: 0.5974 - 
accuracy: 0.7295
Epoch 4/50
288/288 [==============================] - 0s 2ms/step - loss: 0.5543 - 
accuracy: 0.7432
Epoch 5/50
288/288 [==============================] - 0s 2ms/step - loss: 0.5225 - 
accuracy: 0.7549
Epoch 6/50
288/288 [==============================] - 0s 2ms/step - loss: 0.5025 - 
accuracy: 0.7657
Epoch 7/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4890 - 
accuracy: 0.7696
Epoch 8/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4794 - 
accuracy: 0.7725
Epoch 9/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4739 - 
accuracy: 0.7766
```

```
Epoch 10/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4693 -
accuracy: 0.7768
Epoch 11/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4652 -
accuracy: 0.7807
Epoch 12/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4620 -
accuracy: 0.7812
Epoch 13/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4591 -
accuracy: 0.7855
Epoch 14/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4552 -
accuracy: 0.7870
Epoch 15/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4545 -
accuracy: 0.7875
Epoch 16/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4509 -
accuracy: 0.7885
Epoch 17/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4492 -
accuracy: 0.7909
Epoch 18/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4471 -
accuracy: 0.7881
Epoch 19/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4448 -
accuracy: 0.7918
Epoch 20/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4424 -
accuracy: 0.7950
Epoch 21/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4419 -
accuracy: 0.7950
Epoch 22/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4380 -
accuracy: 0.7972
Epoch 23/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4380 -
accuracy: 0.7979
Epoch 24/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4342 -
accuracy: 0.8000
Epoch 25/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4329 -
accuracy: 0.8046
Epoch 26/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4318 -
accuracy: 0.8003
Epoch 27/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4298 -
accuracy: 0.8050
Epoch 28/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4284 -
accuracy: 0.8068
```

```
Epoch 29/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4261 -
accuracy: 0.8055
Epoch 30/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4251 -
accuracy: 0.8074
Epoch 31/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4238 -
accuracy: 0.8059
Epoch 32/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4211 -
accuracy: 0.8079
Epoch 33/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4204 -
accuracy: 0.8076
Epoch 34/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4210 -
accuracy: 0.8070
Epoch 35/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4196 -
accuracy: 0.8083
Epoch 36/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4174 -
accuracy: 0.8085
Epoch 37/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4165 -
accuracy: 0.8129
Epoch 38/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4158 -
accuracy: 0.8118
Epoch 39/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4143 -
accuracy: 0.8107
Epoch 40/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4134 -
accuracy: 0.8120
Epoch 41/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4124 -
accuracy: 0.8146
Epoch 42/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4112 -
accuracy: 0.8129
Epoch 43/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4104 -
accuracy: 0.8161
Epoch 44/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4118 -
accuracy: 0.8137
Epoch 45/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4094 -
accuracy: 0.8100
Epoch 46/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4082 -
accuracy: 0.8133
Epoch 47/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4087 -
accuracy: 0.8122
```

```
Epoch 48/50
288/288 [==============================] - 1s 2ms/step - loss: 0.4071 - accuracy: 0.8139
Epoch 49/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4074 - accuracy: 0.8161
Epoch 50/50
288/288 [==============================] - 0s 2ms/step - loss: 0.4078 - accuracy: 0.8148
```



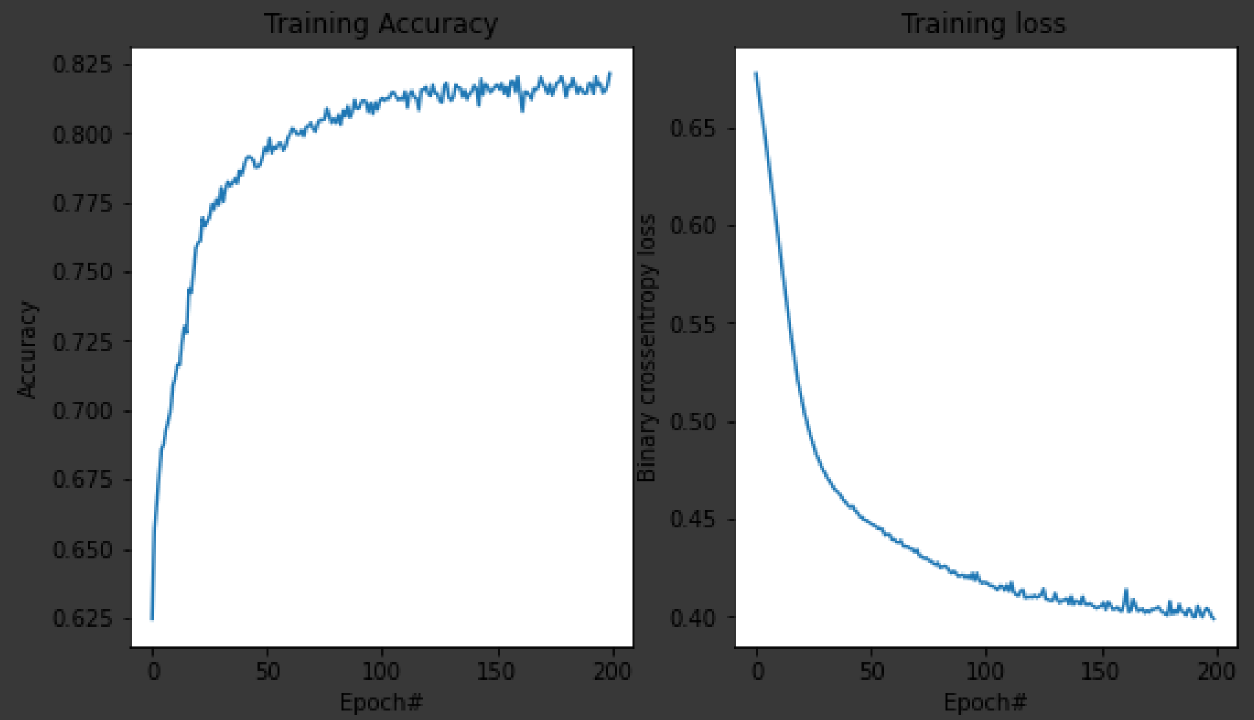

```
#c.v) use evaluate() to find training & validation loss & accuracy
score=model_a.evaluate(features_train,labels_train) #evaluate against (features+labels) of training set
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(x_val_sng,y_val_sng)        #evaluate against (features+labels) of validation set
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
```

```
144/144 [==============================] - 0s 2ms/step - loss: 0.4020 - accuracy: 0.8163
Total loss on training set:   0.4020034372806549
Accuracy of training set:     0.8163265585899353
36/36 [==============================] - 0s 2ms/step - loss: 0.3987 - accuracy: 0.8229
Total loss on validation set:      0.3987441956996918
Accuracy of validation set:   0.8229166865348816#e) find the "best" model
```

```
print("Model 01|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||||||||||||")
#build model
model_a=Sequential()                                         #sequential: w
hen we have layers next to each other in order side by side ()
model_a.add(Dense(input_dim=15, units=32, activation='relu'))#.add: add la
yers to model;dense=defualt layer?;(inputs=#features,units=size of layer,
activation function)
model_a.add(Dense(units=32, activation='relu'))              #2nd hidden la
yer
model_a.add(Dense(units=32, activation='relu'))              #3rd hidden la
yer
model_a.add(Dense(units=1, activation='sigmoid'))            #^4th: output
layer
# model_a.summary()                                            #displays (#
)layers,output shape, #params in output
#compile
opt = tf.keras.optimizers.SGD(learning_rate=0.01)    #defining the optimize
r
model_a.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accurac
y'])
#train model
features_train = x_train_sng
labels_train = y_train_sng
history=model_a.fit(features_train, labels_train,batch_size=100, epochs=20
0,verbose=0)
#evaluate the model
score=model_a.evaluate(features_train,labels_train) #evaluate against (fea
tures+labels) of training set
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(x_val_sng,y_val_sng)           #evaluate against (fe
atures+labels) of validation set
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
# plot model
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

```
Model
01|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||
144/144 [==============================] - 0s 2ms/step - loss: 0.4008 -
accuracy: 0.8183
```

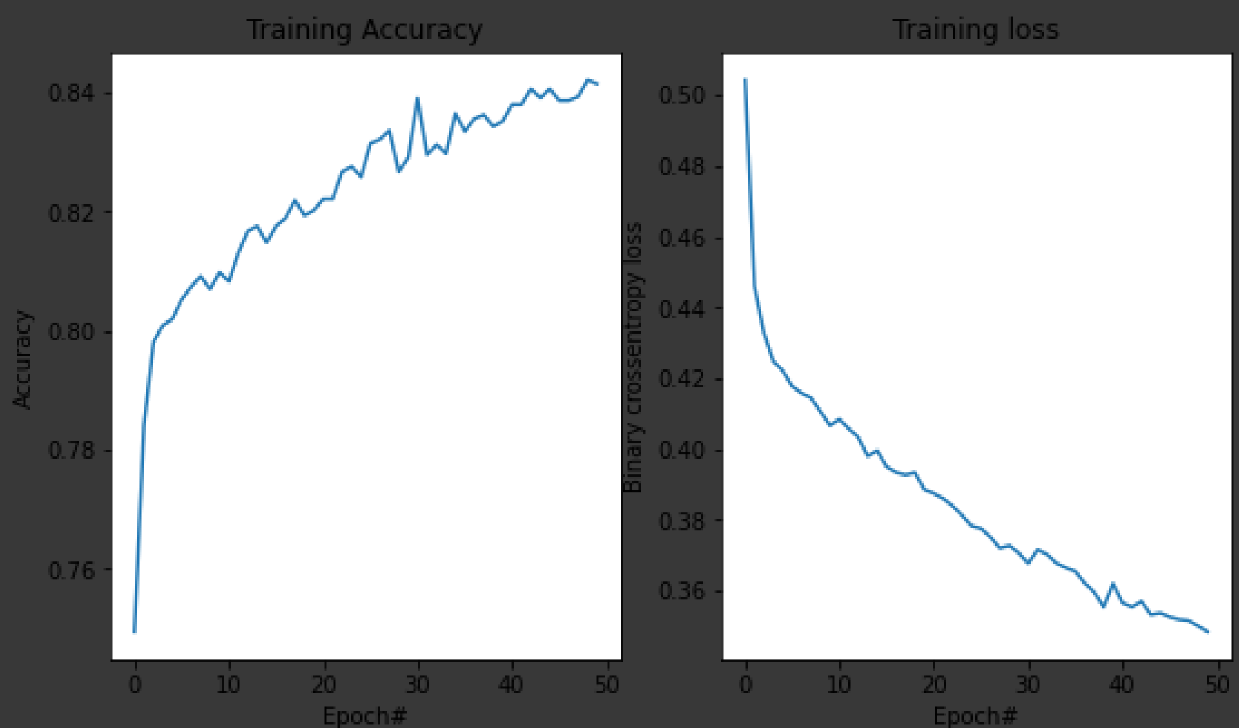
```
Total loss on training set:     0.400775611400604250
Accuracy of training set:       0.8182805180549622
36/36 [==============================] - 0s 2ms/step - loss: 0.4014 -
accuracy: 0.8168
Total loss on validation set:   0.4013625383377075
Accuracy of validation set:     0.8168402910232544
```




```
#e) find the "best" model
print("Model 02|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||||||||||||")
#build model
model_a=Sequential()
model_a.add(Dense(input_dim=15, units=32, activation='relu'))
model_a.add(Dense(units=32, activation='relu'))
model_a.add(Dense(units=1, activation='sigmoid'))
# model_a.summary()
#compile
opt = tf.keras.optimizers.Nadam(learning_rate=0.004)    #defining the optim
izer
model_a.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accurac
y'])
#train model
features_train = x_train_sng
labels_train = y_train_sng
history=model_a.fit(features_train, labels_train,batch_size=16, epochs=50,
verbose=0)
#evaluate the model
```

```
score=model_a.evaluate(features_train,labels_train) #evaluate against (fea
tures+labels) of training set
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(x_val_sng,y_val_sng)        #evaluate against (fe
atures+labels) of validation set
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
# plot model
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

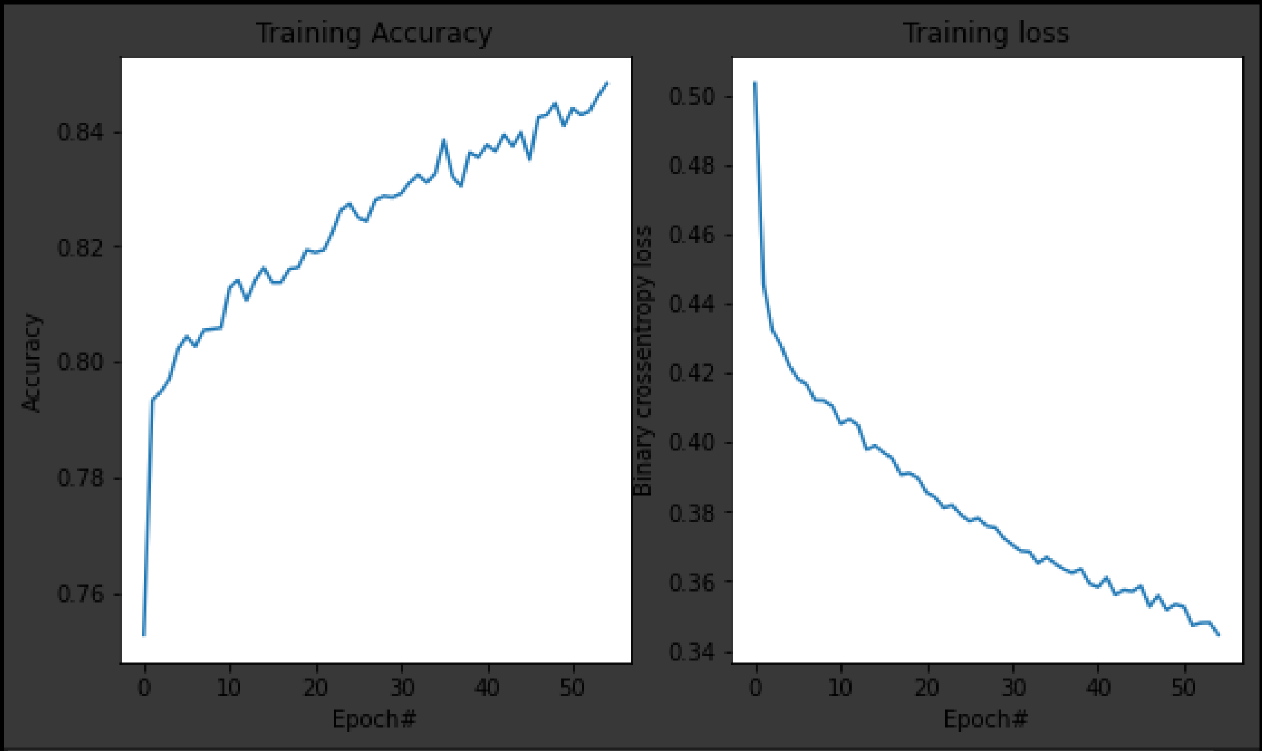
```
Model
02||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||
144/144 [==============================] - 0s 2ms/step - loss: 0.3328 -
accuracy: 0.8526
Total loss on training set:     0.3327885568141937
Accuracy of training set:       0.8525835871696472
36/36 [==============================] - 0s 2ms/step - loss: 0.3828 -
accuracy: 0.8351
Total loss on validation set:   0.3827921748161316
Accuracy of validation set:     0.8350694179534912
```




```
#e) find the "best" model
print("Model 03||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||||||||||||||||||")
```

```
#build model
model_a=Sequential()
model_a.add(Dense(input_dim=15, units=32, activation='relu'))
model_a.add(Dense(units=32, activation='relu'))
model_a.add(Dense(units=1, activation='sigmoid'))
# model_a.summary()
#compile
opt = tf.keras.optimizers.Nadam(learning_rate=0.004)   #defining the optim
izer
model_a.compile(loss='binary_crossentropy',optimizer=opt,metrics=['accurac
y'])
#train model
features_train = x_train_sng
labels_train = y_train_sng
history=model_a.fit(features_train, labels_train,batch_size=16, epochs=55,
verbose=0)
#evaluate the model
score=model_a.evaluate(features_train,labels_train) #evaluate against (fea
tures+labels) of training set
print('Total loss on training set: \t', score[0])
print('Accuracy of training set: \t', score[1])
#evaluating the model on the validation samples
score=model_a.evaluate(x_val_sng,y_val_sng)          #evaluate against (fe
atures+labels) of validation set
print('Total loss on validation set: \t', score[0])
print('Accuracy of validation set: \t', score[1])
# plot model
plt.figure(figsize=[9,5])
acc_curve=np.array(history.history['accuracy'])
loss_curve=np.array(history.history['loss'])
plot_curve(acc_curve,loss_curve)
```

```
Model
03||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
|||||||||||||||
144/144 [==============================] - 0s 2ms/step - loss: 0.3333 -
accuracy: 0.8478
Total loss on training set:     0.3333059847354889
Accuracy of training set:       0.8478072285652161
36/36 [==============================] - 0s 2ms/step - loss: 0.3878 -
accuracy: 0.8403
Total loss on validation set:   0.3878057599067688
Accuracy of validation set:     0.8402777910232544
```



d) This is the table of best performances. Please note that not all attempts were recorded, as the document and notebook would be too long.

#	Layers	Nodes	Batch Size	Epochs	Learning Rate	Optimizer	Loss()	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0	3	32 each	16	50	0.01	SGD	Binary Crossentropy	39.33	81.76	42.08	81.08
1	4	↑	100	200	↑	↑	↑	61.61	62.4	64.7	61.2
2	3	↑	16	50	0.004	Nadam	↑	34.43	84.67	42.81	81.34
3	↑	↑	↑	55	↑	↑	↑	32.19	85.67	40.54	82.55

```
print("Based off the above, Model 3 is the best.")

#f) evaluate the selected model on the test set and report the testing los
s and accuracy.
score=model_a.evaluate(x_test_sng,y_test_sng)   #evaluate model against te
sting data set
print('Total loss on testing set:\t', score[0])
print('Accuracy of testing set:\t', score[1])
```

```
Based off the above, Model 3 is the best.
20/20 [==============================] - 0s 5ms/step - loss: 0.3812 -
accuracy: 0.8391
Total loss on testing set:    0.3812185227870941
Accuracy of testing set:
0.839062511920929
```

```
# convert to good looking pdf

# !jupyter nbconvert --to html /content/Aaron_Mills_Assignment05.ipynb
```

```
[NbConvertApp] WARNING | pattern '/content/Aaron_Mills_Assignment05.ipynb'
matched no files

This application is used to convert notebook files (*.ipynb)

        to various other formats.
```

```
        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error
and include the error message in the cell output (the default behaviour is
```

```
to abort conversion). This flag is only relevant if '--execute' was specified, too.

    Equivalent to: [--ExecutePreprocessor.allow_errors=True]

--stdin

    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'

    Equivalent to: [--NbConvertApp.from_stdin=True]

--stdout

    Write notebook output to stdout instead of files.

    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]

--inplace

    Run nbconvert in place, overwriting the existing notebook (only

            relevant when converting to notebook format)

    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory=]

--clear-output

    Clear output of current file and save in place,

            overwriting the existing notebook.

    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWriter.build_directory= --ClearOutputPreprocessor.enabled=True]

--no-prompt

    Exclude input and output prompts from converted document.

    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=True]

--no-input

    Exclude input cells and output prompts from converted document.

            This mode is ideal for generating code-free reports.

    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True]

--log-level=<Enum>

    Set the log level by value or name.

    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
```

```
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'rst', 'script', 'slides']
            or a dotted object name that represents the import path for an
            `Exporter` class
    Default: 'html'
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
```

```
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook. To recover
                                  previous default behaviour (outputting to the current
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
```

```
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default format
(probably HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex',
'markdown', 'notebook', 'pdf', 'python', 'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic' and
'full'. You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic
mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex
```

```
            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a
couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file,
containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```