Task 1:

data = pd.read\_csv('/content/drive/My Drive/Boston.csv')

data = pd.DataFame(data, columns=["crim","zn","indus","chas","nox","rm","age","dis","rad","tax","ptratio","lstat","medv"

])

label\_col = 'mdv'

x\_\_trian,x\_\_tests,y\_\_trian,y\_\_tests = trian\_\_tests\_\_split(data.iloc[:,0:12], data.iloc[:,12],

tests\_\_size=0.3, random\_state=87)

np.random.sed(155)

def norm\_stats(df1, df2):

dfs = df1.append(df2)

minim = np.min(dfs)

maxim = np.max(dfs)

mu = np.mean(dfs)

sigm = np.std(dfs)

return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

m, M, mu, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c]-mu[c])/s[c]

return df2

stats = norm\_stats(x\_trian,x\_tests)

arr\_x\_trian = np.array(z\_score(x\_trian, stats))

arr\_y\_trian = np.array(y\_trian)

arr\_x\_tests = np.array(z\_score(x\_tests, stats))

arr\_y\_tests = np.array(y\_tests)

print('trianing shape:', arr\_x\_trian.shape)

print('ddd',arr\_y\_trian.shape)

print('trianing samples: ', arr\_x\_trian.shape[0])

print('Validation samples: ', arr\_x\_tests.shape[0])

#basic\_moedl\_1 created moedl with some parameters

def basic\_moedl\_1(x\_size, y\_size):

t\_moedl = Seuqential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dense(50, activation="relu"))

t\_moedl.add(Dense(y\_size))

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

return(t\_moedl)

def basic\_moedl\_2(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dropout(0.1))

t\_moedl.add(Dense(50, activation="relu"))

t\_moedl.add(Dense(20, activation="relu"))

t\_moedl.add(Dense(y\_size))

keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

tensorboard = TensorBoard(log\_dir="logs/final1",histogram\_freq=0, write\_graph=True, write\_images=True)

return(t\_moedl)

moedl = basic\_moedl\_2(arr\_x\_trian.shape[1], 1)

moedl.summary()

epochs = 10

batch\_size =64

from keras.callbacks import TensorBoard

tensorboard = TensorBoard(log\_dir="logs1",histogram\_freq=0, write\_graph=True, write\_images=True)

history = moedl.fit(arr\_x\_trian, arr\_y\_trian,

batch\_size=batch\_size,

epochs=epochs,

shuffle=True,

verbose=2, # Change it to 2, if wished to observe execution

validation\_data=(arr\_x\_tests, arr\_y\_tests),callbacks=[tensorboard])

trian\_score = moedl.evaluate(arr\_x\_trian, arr\_y\_trian, verbose=0)

valid\_score = moedl.evaluate(arr\_x\_tests, arr\_y\_tests, verbose=0)

print('trian MAE: ', round(trian\_score[1], 4), ', trian Loss: ', round(trian\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=2),

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=0),

TensorBoard(log\_dir='./moedl\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0, embeddings\_layer\_names=None, embeddings\_metadata=None),

EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

# summarize history for MAE

plt.subplot(211)

plt.plot(h['mean\_absolute\_error'])

plt.plot(h['val\_mean\_absolute\_error'])

plt.title('trianing vs Validation MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

# summarize history for loss

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

data = pd.read\_csv('/content/drive/My Drive/Boston.csv')

data = pd.DataFrame(data, columns=["crim","zn","indus","chas","nox","rm","age","dis","rad","tax","ptratio","lstat","medv"])

label\_col = 'medv'

#print(kc\_data.describe())

x\_trian,x\_tests,y\_trian,y\_tests = trianing\_testsing\_split(data.iloc[:,0:12], data.iloc[:,12],

tests\_size=0.3, random\_state=87)

np.random.seed(155)

def norm\_stats(df1, df2):

dfs = df1.append(df2)

minimum = np.min(dfs)

maximum = np.max(dfs)

mu = np.mean(dfs)

sigma = np.std(dfs)

return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

m, M, mu, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c]-mu[c])/s[c]

return df2

stats = norm\_stats(x\_trian,x\_tests)

arr\_x\_trian = np.array(z\_score(x\_trian, stats))

arr\_y\_trian = np.array(y\_trian)

arr\_x\_tests = np.array(z\_score(x\_tests, stats))

arr\_y\_tests = np.array(y\_tests)

print('trianing shape:', arr\_x\_trian.shape)

print('ddd',arr\_y\_trian.shape)

print('trianing samples: ', arr\_x\_trian.shape[0])

print('Validation samples: ', arr\_x\_tests.shape[0])

#basic\_moedl\_1 created moedl with some parameters

def basic\_moedl\_1(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dense(50, activation="relu"))

t\_moedl.add(Dense(y\_size))

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

return(t\_moedl)

#basic\_moedl\_2 is different from basic\_moedl\_1 but doing the same task with different structure

def basic\_moedl\_2(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dropout(0.1))

t\_moedl.add(Dense(50, activation="softmax"))

t\_moedl.add(Dense(20, activation="softmax"))

t\_moedl.add(Dense(y\_size))

keras.optimizers.Adam(lr=0.01, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

tensorboard = TensorBoard(log\_dir="logs/final1",histogram\_freq=0, write\_graph=True, write\_images=True)

return(t\_moedl)

moedl = basic\_moedl\_2(arr\_x\_trian.shape[1], 1)

moedl.summary()

epochs = 10

batch\_size =64

from keras.callbacks import TensorBoard

tensorboard = TensorBoard(log\_dir="logs1",histogram\_freq=0, write\_graph=True, write\_images=True)

history = moedl.fit(arr\_x\_trian, arr\_y\_trian,

batch\_size=batch\_size,

epochs=epochs,

shuffle=True,

verbose=2, # Change it to 2, if wished to observe execution

validation\_data=(arr\_x\_valid, arr\_y\_tests),callbacks=[tensorboard])

trian\_score = moedl.evaluate(arr\_x\_trian, arr\_y\_trian, verbose=0)

valid\_score = moedl.evaluate(arr\_x\_valid, arr\_y\_tests, verbose=0)

print('trian MAE: ', round(trian\_score[1], 4), ', trian Loss: ', round(trian\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=2),

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=0),

TensorBoard(log\_dir='./moedl\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0, embeddings\_layer\_names=None, embeddings\_metadata=None),

EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

# summarize history for MAE

plt.subplot(211)

plt.plot(h['mean\_absolute\_error'])

plt.plot(h['val\_mean\_absolute\_error'])

plt.title('trianing vs Validation MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

# summarize history for loss

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

data = pd.read\_csv('/content/drive/My Drive/Boston.csv')

data = pd.DataFrame(data, columns=["crim","zn","indus","chas","nox","rm","age","dis","rad","tax","ptratio","lstat","medv"])

label\_col = 'medv'

#print(kc\_data.describe())

x\_trian,x\_tests,y\_trian,y\_tests = trianing\_testsing\_split(data.iloc[:,0:12], data.iloc[:,12],

tests\_size=0.3, random\_state=87)

np.random.seed(155)

def norm\_stats(df1, df2):

dfs = df1.append(df2)

minimum = np.min(dfs)

maximum = np.max(dfs)

mu = np.mean(dfs)

sigma = np.std(dfs)

return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

m, M, mu, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c]-mu[c])/s[c]

return df2

stats = norm\_stats(x\_trian,x\_tests)

arr\_x\_trian = np.array(z\_score(x\_trian, stats))

arr\_y\_trian = np.array(y\_trian)

arr\_x\_tests = np.array(z\_score(x\_tests, stats))

arr\_y\_tests = np.array(y\_tests)

print('trianing shape:', arr\_x\_trian.shape)

print('ddd',arr\_y\_trian.shape)

print('trianing samples: ', arr\_x\_trian.shape[0])

print('Validation samples: ', arr\_x\_valid.shape[0])

#basic\_moedl\_1 created moedl with some parameters

def basic\_moedl\_1(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dense(50, activation="relu"))

t\_moedl.add(Dense(y\_size))

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

return(t\_moedl)

#basic\_moedl\_2 is different from basic\_moedl\_1 but doing the same task with different structure

def basic\_moedl\_2(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dropout(0.1))

t\_moedl.add(Dense(50, activation="softmax"))

t\_moedl.add(Dense(20, activation="softmax"))

t\_moedl.add(Dense(y\_size))

keras.optimizers.Adam(lr=0.01, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

tensorboard = TensorBoard(log\_dir="logs/final1",histogram\_freq=0, write\_graph=True, write\_images=True)

return(t\_moedl)

moedl = basic\_moedl\_2(arr\_x\_trian.shape[1], 1)

moedl.summary()

epochs = 10

batch\_size =64

from keras.callbacks import TensorBoard

tensorboard = TensorBoard(log\_dir="logs1",histogram\_freq=0, write\_graph=True, write\_images=True)

history = moedl.fit(arr\_x\_trian, arr\_y\_trian,

batch\_size=batch\_size,

epochs=epochs,

shuffle=True,

verbose=2, # Change it to 2, if wished to observe execution

validation\_data=(arr\_x\_valid, arr\_y\_tests),callbacks=[tensorboard])

trian\_score = moedl.evaluate(arr\_x\_trian, arr\_y\_trian, verbose=0)

valid\_score = moedl.evaluate(arr\_x\_valid, arr\_y\_tests, verbose=0)

print('trian MAE: ', round(trian\_score[1], 4), ', trian Loss: ', round(trian\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=2),

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=0),

TensorBoard(log\_dir='./moedl\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0, embeddings\_layer\_names=None, embeddings\_metadata=None),

EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

# summarize history for MAE

plt.subplot(211)

plt.plot(h['mean\_absolute\_error'])

plt.plot(h['val\_mean\_absolute\_error'])

plt.title('trianing vs Validation MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

# summarize history for loss

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

data = pd.read\_csv('/content/drive/My Drive/Boston.csv')

data = pd.DataFrame(data, columns=["crim","zn","indus","chas","nox","rm","age","dis","rad","tax","ptratio","lstat","medv"

])

label\_col = 'medv'

x\_trian,x\_tests,y\_trian,y\_tests = trianing\_testsing\_split(data.iloc[:,0:12], data.iloc[:,12],

tests\_size=0.3, random\_state=87)

np.random.seed(155)

def norm\_stats(df1, df2):

dfs = df1.append(df2)

minimum = np.min(dfs)

maximum = np.max(dfs)

mu = np.mean(dfs)

sigma = np.std(dfs)

return (minimum, maximum, mu, sigma)

def z\_score(col, stats):

m, M, mu, s = stats

df2 = pd.DataFrame()

for c in col.columns:

df2[c] = (col[c]-mu[c])/s[c]

return df2

stats = norm\_stats(x\_trian,x\_tests)

arr\_x\_trian = np.array(z\_score(x\_trian, stats))

arr\_y\_trian = np.array(y\_trian)

arr\_x\_tests = np.array(z\_score(x\_tests, stats))

arr\_y\_tests = np.array(y\_tests)

print('trianing shape:', arr\_x\_trian.shape)

print('ddd',arr\_y\_trian.shape)

print('trianing samples: ', arr\_x\_trian.shape[0])

print('Validation samples: ', arr\_x\_tests.shape[0])

#basic\_moedl\_1 created moedl with some parameters

def basic\_moedl\_1(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dense(50, activation="relu"))

t\_moedl.add(Dense(y\_size))

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

return(t\_moedl)

#basic\_moedl\_2 is different from basic\_moedl\_1 but doing the same task with different structure

def basic\_moedl\_2(x\_size, y\_size):

t\_moedl = Sequential()

t\_moedl.add(Dense(100, activation="tanh", input\_shape=(x\_size,)))

t\_moedl.add(Dropout(0.1))

t\_moedl.add(Dense(50, activation="relu"))

t\_moedl.add(Dense(20, activation="relu"))

t\_moedl.add(Dense(y\_size))

keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.0, amsgrad=False)

t\_moedl.compile(loss='mean\_squared\_error',

optimizer=Adam(),

metrics=[metrics.mae])

tensorboard = TensorBoard(log\_dir="logs/final1",histogram\_freq=0, write\_graph=True, write\_images=True)

return(t\_moedl)

moedl = basic\_moedl\_2(arr\_x\_trian.shape[1], 1)

moedl.summary()

epochs = 10

batch\_size =128

from keras.callbacks import TensorBoard

tensorboard = TensorBoard(log\_dir="logs1",histogram\_freq=0, write\_graph=True, write\_images=True)

history = moedl.fit(arr\_x\_trian, arr\_y\_trian,

batch\_size=batch\_size,

epochs=epochs,

shuffle=True,

verbose=2, # Change it to 2, if wished to observe execution

validation\_data=(arr\_x\_tests, arr\_y\_tests),callbacks=[tensorboard])

trian\_score = moedl.evaluate(arr\_x\_trian, arr\_y\_trian, verbose=0)

valid\_score = moedl.evaluate(arr\_x\_tests, arr\_y\_tests, verbose=0)

print('trian MAE: ', round(trian\_score[1], 4), ', trian Loss: ', round(trian\_score[0], 4))

print('Val MAE: ', round(valid\_score[1], 4), ', Val Loss: ', round(valid\_score[0], 4))

keras\_callbacks = [

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}-{val\_loss:.2f}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=2),

moedlCheckpoint('/tmp/keras\_checkpoints/moedl.{epoch:02d}.hdf5', monitor='val\_loss', save\_best\_only=True, verbose=0),

TensorBoard(log\_dir='./moedl\_3', histogram\_freq=0, write\_graph=True, write\_images=True, embeddings\_freq=0, embeddings\_layer\_names=None, embeddings\_metadata=None),

EarlyStopping(monitor='val\_mean\_absolute\_error', patience=20, verbose=0)

]

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

# summarize history for MAE

plt.subplot(211)

plt.plot(h['mean\_absolute\_error'])

plt.plot(h['val\_mean\_absolute\_error'])

plt.title('trianing vs Validation MAE')

plt.ylabel('MAE')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

# summarize history for loss

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['trian', 'Validation'], loc='upper left')

Task 2:

from \_\_future\_\_ import print\_function

import keras

from keras.moedls import Sequential

from keras.layers import Dense

import numpy as np

import pandas as pd

from sklearn.moedl\_selection import trian\_tests\_split

from keras.callbacks import TensorBoard

from time import time

dataset = pd.read\_csv('/content/drive/My Drive/heart.csv',index\_col=0)

dataset.astype(float)

# Normalize values to range [0:1]

dataset /= dataset.max()

y = dataset['target']

X = dataset.drop(['target'], axis = 1)

X\_trian, X\_tests, Y\_trian, Y\_tests = trian\_tests\_split(X, y, tests\_size = 0.33, random\_state = 0)

np.random.seed(155)

moedl = Sequential() # create moedl

moedl.add(Dense(40, input\_dim=12, activation='relu')) # hidden layer

moedl.add(Dense(20, input\_dim=40, activation='relu'))

moedl.add(Dense(1, activation='sigmoid')) # output layer

moedl.compile(loss= keras.losses.binary\_crossentropy,

optimizer=keras.optimizers.adamax(),

metrics=['accuracy'])

tensorborad = TensorBoard(log\_dir="logs/{}".format(time()))

history = moedl.fit(X\_trian, Y\_trian,batch\_size=500,epochs=5,verbose=1,

validation\_data=(X\_tests, Y\_tests), callbacks=[tensorborad])

y\_pred = moedl.predct\_classes(X\_tests)

score = moedl.evaluate(X\_tests, Y\_tests, verbose=0)

print('Loss:', score[0])

print('Accuracy:', score[1])

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Accuracy')

plt.legend(['trian', 'Validation'], loc='upper left')

from \_\_future\_\_ import print\_function

import keras

from keras.moedls import Sequential

from keras.layers import Dense

import numpy as np

import pandas as pd

from sklearn.moedl\_selection import trian\_tests\_split

from keras.callbacks import TensorBoard

from time import time

dataset = pd.read\_csv('/content/drive/My Drive/heart.csv',index\_col=0)

dataset.astype(float)

# Normalize values to range [0:1]

dataset /= dataset.max()

y = dataset['target']

X = dataset.drop(['target'], axis = 1)

X\_trian, X\_tests, Y\_trian, Y\_tests = trian\_tests\_split(X, y, tests\_size = 0.33, random\_state = 0)

np.random.seed(155)

moedl = Sequential() # create moedl

moedl.add(Dense(40, input\_dim=12, activation='tanh')) # hidden layer

moedl.add(Dense(20, input\_dim=40, activation='tanh'))

moedl.add(Dense(1, activation='softmax')) # output layer

moedl.compile(loss= keras.losses.binary\_crossentropy,

optimizer=keras.optimizers.adamax(),

metrics=['accuracy'])

tensorborad = TensorBoard(log\_dir="logs/{}".format(time()))

history = moedl.fit(X\_trian, Y\_trian,batch\_size=500,epochs=5,verbose=1,

validation\_data=(X\_tests, Y\_tests), callbacks=[tensorborad])

y\_pred = moedl.predct\_classes(X\_tests)

score = moedl.evaluate(X\_tests, Y\_tests, verbose=0)

print('Loss:', score[0])

print('Accuracy:', score[1])

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Accuracy')

plt.legend(['trian', 'Validation'], loc='upper left')

from \_\_future\_\_ import print\_function

import keras

from keras.moedls import Sequential

from keras.layers import Dense

import numpy as np

import pandas as pd

from sklearn.moedl\_selection import trian\_tests\_split

from keras.callbacks import TensorBoard

from time import time

dataset = pd.read\_csv('/content/drive/My Drive/heart.csv',index\_col=0)

dataset.astype(float)

# Normalize values to range [0:1]

dataset /= dataset.max()

y = dataset['target']

X = dataset.drop(['target'], axis = 1)

X\_trian, X\_tests, Y\_trian, Y\_tests = trian\_tests\_split(X, y, tests\_size = 0.25, random\_state = 0)

np.random.seed(155)

moedl = Sequential() # create moedl

moedl.add(Dense(40, input\_dim=12, activation='relu')) # hidden layer

moedl.add(Dense(20, input\_dim=40, activation='relu'))

moedl.add(Dense(1, activation='sigmoid')) # output layer

moedl.compile(loss= keras.losses.binary\_crossentropy,

optimizer=keras.optimizers.adamax(),

metrics=['accuracy'])

tensorborad = TensorBoard(log\_dir="logs/{}".format(time()))

history = moedl.fit(X\_trian, Y\_trian,batch\_size=500,epochs=5,verbose=1,

validation\_data=(X\_tests, Y\_tests), callbacks=[tensorborad])

y\_pred = moedl.predct\_classes(X\_tests)

score = moedl.evaluate(X\_tests, Y\_tests, verbose=0)

print('Loss:', score[0])

print('Accuracy:', score[1])

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Accuracy')

plt.legend(['trian', 'Validation'], loc='upper left')

from \_\_future\_\_ import print\_function

import keras

from keras.moedls import Sequential

from keras.layers import Dense

import numpy as np

import pandas as pd

from sklearn.moedl\_selection import trian\_tests\_split

from keras.callbacks import TensorBoard

from time import time

dataset = pd.read\_csv('/content/drive/My Drive/heart.csv',index\_col=0)

dataset.astype(float)

# Normalize values to range [0:1]

dataset /= dataset.max()

y = dataset['target']

X = dataset.drop(['target'], axis = 1)

X\_trian, X\_tests, Y\_trian, Y\_tests = trian\_tests\_split(X, y, tests\_size = 0.25, random\_state = 0)

np.random.seed(155)

moedl = Sequential() # create moedl

moedl.add(Dense(40, input\_dim=12, activation='relu')) # hidden layer

moedl.add(Dense(20, input\_dim=40, activation='relu'))

moedl.add(Dense(1, activation='sigmoid')) # output layer

moedl.compile(loss= keras.losses.binary\_crossentropy,

optimizer=keras.optimizers.SGD(),

metrics=['accuracy'])

tensorborad = TensorBoard(log\_dir="logs/{}".format(time()))

history = moedl.fit(X\_trian, Y\_trian,batch\_size=500,epochs=5,verbose=1,

validation\_data=(X\_tests, Y\_tests), callbacks=[tensorborad])

y\_pred = moedl.predct\_classes(X\_tests)

score = moedl.evaluate(X\_tests, Y\_tests, verbose=0)

print('Loss:', score[0])

print('Accuracy:', score[1])

def plot\_hist(h, xsize=6, ysize=10):

# Prepare plotting

fig\_size = plt.rcParams["figure.figsize"]

plt.rcParams["figure.figsize"] = [xsize, ysize]

fig, axes = plt.subplots(nrows=4, ncols=4, sharex=True)

plt.subplot(212)

plt.plot(h['loss'])

plt.plot(h['val\_loss'])

plt.title('trianing vs Validation Loss')

plt.ylabel('Loss')

plt.xlabel('Accuracy')

plt.legend(['trian', 'Validation'], loc='upper left')

Task 3:

import matplotlib.pylab as plt

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, Lambda, MaxPool2D, BatchNormalization, Input

from keras.moedls import Sequential

from keras.utils import to\_categorical

from keras.preprocessing.image import ImageDataGenerator

from sklearn.moedl\_selection import trian\_tests\_split

from pathlib import Path

from keras.optimizers import Adam,RMSprop,SGD

import pandas as pd

df = pd.read\_csv("/content/drive/My Drive/10-monkey-species/monkey\_labels.txt")

print(df)

height=150

width=150

channels=3

batch\_size=32

seed=1337

trian\_dir = Path('/content/drive/My Drive/10-monkey-species/trianing/trianing')

tests\_dir = Path('/content/drive/My Drive/10-monkey-species/validation/validation')

# trianing generator

trian\_datagen = ImageDataGenerator(rescale=1./255)

trian\_generator = trian\_datagen.flow\_from\_directory(trian\_dir, target\_size=(height,width),batch\_size=batch\_size,seed=seed,class\_mode='categorical')

# tests generator

tests\_datagen = ImageDataGenerator(rescale=1./255)

tests\_generator = tests\_datagen.flow\_from\_directory(tests\_dir, target\_size=(height,width),batch\_size=batch\_size,

seed=seed,class\_mode='categorical')

moedl = Sequential()

moedl.add(Conv2D(32, kernel\_size=(3, 3),activation='relu',input\_shape=(150,150,3)))

moedl.add(Flatten())

moedl.add(Dense(512,activation='relu'))

moedl.add(Dropout(0.5))

moedl.add(Dense(256,activation='relu'))

moedl.add(Dense(10, activation='softmax'))

moedl.compile(Adam(lr=0.0001),loss="categorical\_crossentropy", metrics=["accuracy"])

history = moedl.fit\_generator(trian\_generator,

steps\_per\_epoch= 1027/batch\_size,epochs=5,verbose=1,validation\_data=tests\_generator,validation\_steps= 4)

moedl.summary()

print(history.history.keys())

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.title('trianing and validation accuracy')

plt.plot(epochs, acc, 'red', label='trianing acc')

plt.plot(epochs, val\_acc, 'blue', label='Validation acc')

plt.legend()

plt.figure()

plt.title('trianing and validation loss')

plt.plot(epochs, loss, 'red', label='trianing loss')

plt.plot(epochs, val\_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()

Task 4:

import pandas as pd

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.moedls import Sequential

from keras.optimizers import SGD

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D

from keras.layers.convolutional import Conv1D, MaxPooling1D

from keras.utils.np\_utils import to\_categorical

import re

from sklearn.preprocessing import LabelEncoder

data = pd.read\_csv('/content/drive/My Drive/sentiment-analysis-on-movie-reviews/trian.tsv', sep='\t')

data1 = pd.read\_csv('/content/drive/My Drive/sentiment-analysis-on-movie-reviews/tests.tsv', sep='\t')

# Keeping only the neccessary columns

data = data[['Phrase','Sentiment']]

data['Phrase'] = data['Phrase'].apply(lambda x: x.lower())

data['Phrase'] = data['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():

row[0] = row[0].replace('rt', ' ')

max\_fatures = 2000

tokenizer = Tokenizer(num\_words=max\_fatures, split=' ')

tokenizer.fit\_on\_texts(data['Phrase'].values)

X = tokenizer.texts\_to\_sequences(data['Phrase'].values)

X = pad\_sequences(X)

data1 = data1[['Phrase']]

data1['Phrase'] = data1['Phrase'].apply(lambda x: x.lower())

data1['Phrase'] = data1['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():

row[0] = row[0].replace('rt', ' ')

tokenizer.fit\_on\_texts(data1['Phrase'].values)

Y = tokenizer.texts\_to\_sequences(data1['Phrase'].values)

Y = pad\_sequences(Y)

embed\_dim = 128

batch\_size = 32

def my\_moedl():

moedl = Sequential()

moedl.add(Embedding(max\_fatures, embed\_dim, input\_length=X.shape[1]))

moedl.add(

Conv1D(128, (5), activation='relu', kernel\_constriant=maxnorm(3)))

moedl.add(Dropout(0.2))

moedl.add(Conv1D(128, (5), activation='relu', padding='same', kernel\_constriant=maxnorm(3)))

moedl.add(MaxPooling1D(5))

moedl.add(Flatten())

moedl.add(Dense(512, activation='relu', kernel\_constriant=maxnorm(3)))

moedl.add(Dropout(0.5))

moedl.add(Dense(5, activation='softmax'))

return moedl

x = to\_categorical(integer\_encoded)

X\_trian, X\_tests, y\_trian, y\_tests = trian\_tests\_split(X,x, tests\_size=0.3, random\_state=32)

epochs = 15

lrate = 0.01

decay = lrate / epochs

moedl = my\_moedl()

sgd = SGD(lr=lrate, momentum=0.8, decay=decay, nesterov=False)

moedl.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

batch\_size = 32

history=moedl.fit(X\_trian,y\_trian, epochs=5, verbose=True, validation\_data=(X\_tests,y\_tests), batch\_size=256)

Metrics, accuracy = moedl.evaluate(X\_tests, y\_tests, verbose=2, batch\_size=batch\_size)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

print(history.history.keys())

plt.title('Accuracy-Loss')

plt.xlabel('accuracy')

plt.ylabel('loss')

plt.show()

Task 5:

import pandas as pd

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.moedls import Sequential

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D

from sklearn.moedl\_selection import trian\_tests\_split

from keras.utils.np\_utils import to\_categorical

import re

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

data = pd.read\_csv('/content/drive/My Drive/sentiment-analysis-on-movie-reviews/trian.tsv', sep='\t')

data1 = pd.read\_csv('/content/drive/My Drive/sentiment-analysis-on-movie-reviews/tests.tsv', sep='\t')

# Keeping only the neccessary columns

data = data[['Phrase','Sentiment']]

data['Phrase'] = data['Phrase'].apply(lambda x: x.lower())

data['Phrase'] = data['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():

row[0] = row[0].replace('rt', ' ')

max\_fatures = 2000

tokenizer = Tokenizer(num\_words=max\_fatures, split=' ')

tokenizer.fit\_on\_texts(data['Phrase'].values)

X = tokenizer.texts\_to\_sequences(data['Phrase'].values)

X = pad\_sequences(X)

data1 = data1[['Phrase']]

data1['Phrase'] = data1['Phrase'].apply(lambda x: x.lower())

data1['Phrase'] = data1['Phrase'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows():

row[0] = row[0].replace('rt', ' ')

tokenizer.fit\_on\_texts(data1['Phrase'].values)

Y = tokenizer.texts\_to\_sequences(data1['Phrase'].values)

Y = pad\_sequences(Y)

Y = np.delete(Y, 0, 1)

embed\_dim = 64

lstm\_out = 196

def createmoedl():

moedl = Sequential()

moedl.add(Embedding(max\_fatures, embed\_dim,input\_length = X.shape[1]))

moedl.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2))

moedl.add(Dense(5,activation='softmax'))

moedl.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy'])

return moedl

labelencoder = LabelEncoder()

integer\_encoded = labelencoder.fit\_transform(data['Sentiment'])

x = to\_categorical(integer\_encoded)

y = to\_categorical(integer\_encoded)

batch\_size = 32

X\_trian, X\_tests, y\_trian, y\_tests = trian\_tests\_split(X, x, tests\_size=0.25, random\_state=1000)

moedl = createmoedl()

history=moedl.fit(X\_trian,y\_trian, epochs=5, verbose=True, validation\_data=(X\_tests,y\_tests), batch\_size=256)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

print(history.history.keys())

plt.title('Accuracy-Loss')

plt.xlabel('accuracy')

plt.ylabel('loss')

plt.show()

Task 7:

from keras.layers import Input, Dense

from keras.moedls import moedl

import matplotlib.pyplot as plt

# this is the size of our encoded representations

encoding\_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder

input\_img = Input(shape=(784,))

# "encoded" is the encoded representation of the input

encoded = Dense(encoding\_dim, activation='relu')(input\_img)

# "decoded" is the lossy reconstruction of the input

decoded = Dense(784, activation='sigmoid')(encoded)

# this moedl maps an input to its reconstruction

autoencoder = moedl(input\_img, decoded)

# this moedl maps an input to its encoded representation

encoder = moedl(input\_img, encoded)

#let's create a seperate decoder moedl

# create a placeholder for an encoded (32-dimensional) input

encoded\_input = Input(shape=(encoding\_dim,))

# retrieve the last layer of the autoencoder moedl

decoder\_layer = autoencoder.layers[-1]

# create the decoder moedl

decoder = moedl(encoded\_input, decoder\_layer(encoded\_input))

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy',metrics=['accuracy'])

from keras.datasets import mnist, fashion\_mnist

import numpy as np

(x\_trian, y\_trian), (x\_tests, y\_tests) = fashion\_mnist.load\_data()

x\_trian = x\_trian.astype('float32') / 255.

x\_tests = x\_tests.astype('float32') / 255.

x\_trian = x\_trian.reshape((len(x\_trian), np.prod(x\_trian.shape[1:])))

x\_tests = x\_tests.reshape((len(x\_tests), np.prod(x\_tests.shape[1:])))

autoencode=autoencoder.fit(x\_trian, x\_trian,

epochs=10,

batch\_size=256,

shuffle=True,

verbose=2,

validation\_data=(x\_tests, x\_tests))

e\_imgs = encoder.predct(x\_tests)

d\_imgs = decoder.predct(encoded\_imgs)

# use Matplotlib

import matplotlib.pyplot as plt

# displaying original and reconstructed image

n = 10 # how many digits we will display

plt.figure(figsize=(20, 4))

for i in range(n):

# display original

ax = plt.subplot(2, n, i + 1)

plt.imshow(x\_tests[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# display reconstruction

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(decoded\_imgs[i].reshape(28, 28))

plt.get()

x.get\_\_

xaxis().set\_visible(False)

x.get\_\_yaxis().set\_visible(False)

plt.Show()