Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE & COMMERCE

(EMPOWERED AUTONOMOUS)

[Generative AI]

Name : Yash Sanjay Bind
Roll No: 10443
Class : MSc Data Science and Artificial Intelligence part 2 (Semester 3)

Ramniranjan Jhunjhunwala College
of Arts, Science and Commerce

Department of Data Science and Artificial Intelligence

CERTIFICATE

This is to certify Yash Sanjay Bind of Msc. Data Science and Artificial Intelligence Roll No . 10443 has successfully completed the practical of <u>Generative AI</u> during the Academic Year 2024-2025.

Date :

(Prof. Mujtaba Shaikh) **External Examiner**

Prof-In-Charge

INDEX

Sr No.	Practical Name	Date	Signature
1	Perceptron	20 Jun 2024	
2	Classification : 1 Binary 2 Multiclass	25 Jun 2024	
3	Regression	25 Jun 2024	
4	Early Stopping	5 Jul 2024	
5	Dropouts : 1 Regression 2 Classification	8 Jul 2024	
6	RNN	22 Jul 2024	
7	CNN	13 Aug 2024	
8	Autoencoders: 1 Vanilla AE 2 Denoising AE	20 Aug 2024 21 Aug 2024	
9	GANs	3 Sept 2024	

Practical 1: Perceptrons

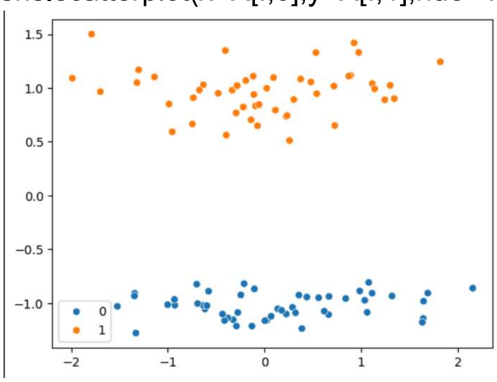
from sklearn.datasets import make_classification
X,Y
=make_classification(n_samples=100,n_features=2,n_redundant=0,n_informative=1,n_clusters_per_class=1)
Y.shape

```
(100,)
```

X.shape

```
(100, 2)
```

import matplotlib.pyplot as plt
import seaborn as sns
sns.scatterplot(x=X[:,0],y=X[:,1],hue=Y)



from sklearn.linear_model import Perceptron
p=Perceptron()
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(X,Y,test_size=0.35)
p.fit(xtrain,ytrain)

```
▾ Perceptron
Perceptron()
```

ypred=p.predict(xtest)
Ypred

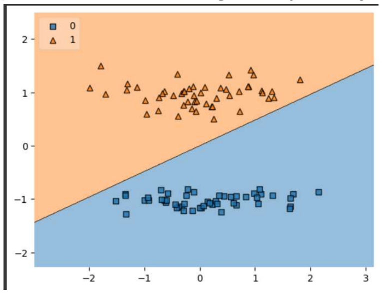
```
array([1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1])
```

from sklearn.metrics import accuracy_score
accuracy_score(ytest,ypred)

```
1.0
```

pip install mlxtend
from mlxtend.plotting import plot_decision_regions

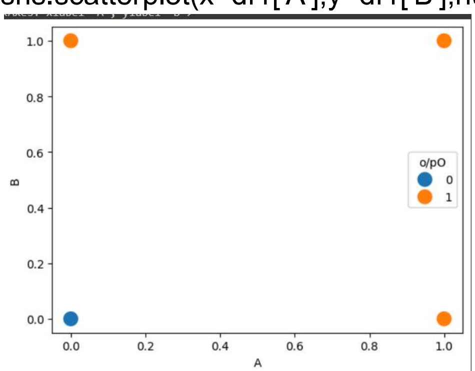
```
plot_decision_regions(X=X,y=Y,clf=p,legend=2)
```



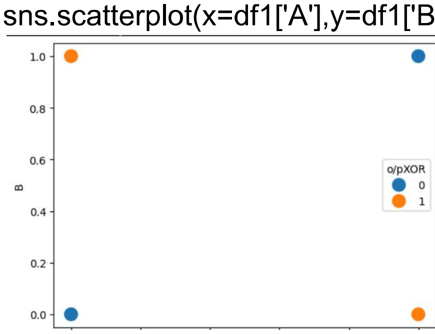

```
from mlxtend.plotting import plot_decision_regions
import pandas as pd
df1=pd.DataFrame({"A":[0,1,0,1],
                 "B":[0,0,1,1],
                 "o/pA":[0,0,0,1],
                 "o/pO":[0,1,1,1],
                 "o/pXOR":[0,1,1,0]})
```

	A	B	o/pA	o/pO	o/pXOR
0	0	0	0	0	0
1	1	0	0	1	1
2	0	1	0	1	1
3	1	1	1	1	0

```
import seaborn as sns
sns.scatterplot(x=df1['A'],y=df1['B'],hue=df1["o/pA"],s=200)
```




```
sns.scatterplot(x=df1['A'],y=df1['B'],hue=df1["o/pXOR"],s=200)
```

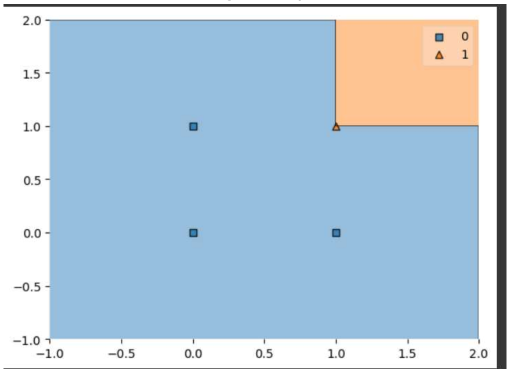



```
p1=Perceptron()
p2=Perceptron()
```

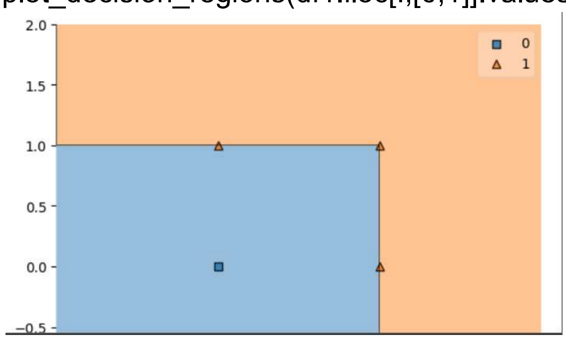
```
p3=Perceptron()
p1.fit(df1[['A','B']],df1['o/pA'])
p2.fit(df1[['A','B']],df1['o/pO'])
p3.fit(df1[['A','B']],df1['o/pXOR'])
```

```
▾ Perceptron
Perceptron()
```

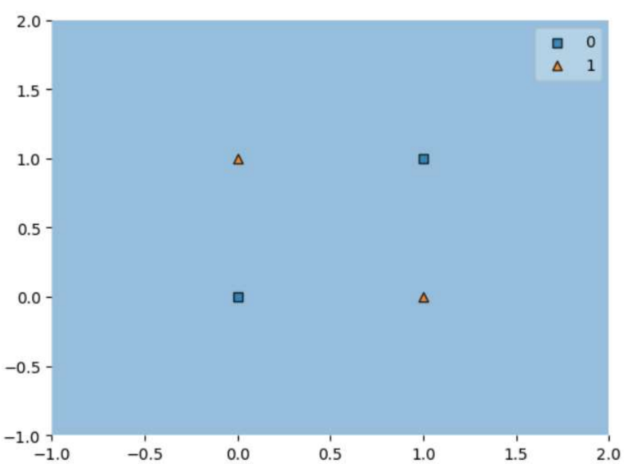
```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(df1.iloc[:,[0,1]].values,df1.iloc[:,2].values,clf=p1)
```




```
plot_decision_regions(df1.iloc[:,[0,1]].values,df1.iloc[:,3].values,clf=p2)
```




```
plot_decision_regions(df1.iloc[:,[0,1]].values,df1.iloc[:,4].values,clf=p3)
```



Practical 2:
2.1 Binary Classification

import pandas as pd
import numpy as np
import tensorflow
from tensorflow import keras
from tensorflow.keras import layers ,Sequential
from tensorflow.keras.layers import Dense
df=pd.read_csv("/content/drive/MyDrive/sem 3/generative AI/Churn_Modelling.csv")
Df

	RowNumber	CustomerId	Surname	Cr
0	1	15634602	Hargrave	
1	2	15647311	Hill	
2	3	15619304	Onio	
3	4	15701354	Boni	
4	5	15737888	Mitchell	
...	...	...	...	
9995	9996	15606229	Obijiaku	
9996	9997	15569892	Johnstone	

Df.columns

```
Index(['RowNumber', 'Custom
       'Gender', 'Age', 'Te
       'IsActiveMember', 'E
      dtype='object')
```

df.head(2)

	RowNumber	CustomerId	Surname	C
0	1	15634602	Hargrave	
1	2	15647311	Hill	

df=df.drop(columns=['RowNumber','CustomerId','Surname'])
Df

	CreditScore	Geography	Gender
0	619	France	Female
1	608	Spain	Female
2	502	France	Female
3	699	France	Female
4	850	Spain	Female
...	...	...	...

Df.dtypes

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

df.duplicated().sum()

```
0
```

df['Gender'].value_counts()

```
Gender
Male      5457
Female    4543
Name: count, dtype: int64
```

df=pd.get_dummies(df,columns=['Gender','Geography'],dtype=int,drop_first=True)
Df

	CreditScore	Age	Tenure	Balance	Nu
0	619	42	2	0.00	
1	608	41	1	83807.86	
2	502	42	8	159660.80	
3	699	39	1	0.00	
4	850	43	2	125510.82	

df.isnull().sum()

```
CreditScore          0
Age                  0
Tenure               0
Balance              0
NumOfProducts        0
HasCrCard            0
IsActiveMember       0
EstimatedSalary      0
Exited               0
Gender_Male          0
Geography_Germany    0
Geography_Spain      0
dtype: int64
```

X=df.drop(columns=['Exited'])
y=df['Exited']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
model= Sequential()
model.add(Dense(units=3,activation='sigmoid',input_dim=11))
model.add(Dense(units=2,activation='sigmoid'))
model.add(Dense(units=1,activation='sigmoid'))
model.summary()

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 3)                 36

 dense_1 (Dense)             (None, 2)                 8

 dense_2 (Dense)             (None, 1)                 3

=================================================================
```

model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
model.layers[0].get_weights()

```
array([[ 1.9409966e-01, -7.4104548e-0
       [-5.0462568e-01,  1.3782954e-0
       [-5.8791631e-01,  5.6638682e-0
       [-5.6195259e-01,  4.8079610e-0
       [-2.4322617e-01, -1.6016236e-0
       [ 1.7665690e-01,  3.4130615e-0
       [ 3.8132560e-01,  2.4976367e-0
       [ 4.8724616e-01, -2.8692663e-0
       [-5.9233463e-01, -4.9620867e-0
       [ 2.9614562e-01, -8.3914936e-0
       [ 1.1164361e-01, -4.9002475e-0
array([0., 0., 0.], dtype=float32)]
```

model.fit(X_train,y_train,batch_size=10,epochs=10)

```
Epoch 1/10
800/800 [==============================]
Epoch 2/10
800/800 [==============================]
Epoch 3/10
800/800 [==============================]
Epoch 4/10
800/800 [==============================]
Epoch 5/10
800/800 [==============================]
```

2.2 Multiclass Classification

import pandas as pd
import numpy as np
import tensorflow
from tensorflow import keras
from tensorflow.keras import layers ,Sequential
from tensorflow.keras.layers import Dense
df= pd.read_csv('/content/drive/MyDrive/sem 3/generative AI/train.csv')
df

	label	pixel0	pixel1	pixel2	pixel3	pixel4
0	1	0	0	0	0	0
1	0	0	0	0	0	0
2	1	0	0	0	0	0
3	4	0	0	0	0	0
4	0	0	0	0	0	0

Df.shape

```
(42000, 785)
```

df.isnull().sum()

```
label     0
pixel0    0
pixel1    0
pixel2    0
pixel3    0
```

df.info()

```
<class 'pandas.core.fra
RangeIndex: 42000 entr
Columns: 785 entries,
dtypes: int64(785)
memory usage: 251.5 MB
```

df.describe()

	label	pixel0
count	42000.000000	42000.0
mean	4.456643	0.0
std	2.887730	0.0
min	0.000000	0.0
25%	2.000000	0.0
50%	4.000000	0.0
75%	7.000000	0.0
max	9.000000	0.0

Sdf.duplicated().sum()

```
0
```

Df.columns

```
Index(['label', 'pixel0', 'pixel1',
       'pixel6', 'pixel7', 'pixel8',
```

X=df.drop(['label'],axis=1)
y=df['label']
Y.shape

```
(42000,)
```

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
model = Sequential()
model.add(Dense(128,activation='relu',input_dim=784))

```
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential"

 Layer (type)                Output Shape
=========================================
 dense (Dense)               (None, 128)

 dense_1 (Dense)             (None, 64)

 dense_2 (Dense)             (None, 10)
```

```
df['label'].value_counts()
```

```
label
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
```

```
df['label'].unique()
```

```
array([1, 0, 4, 7, 3, 5, 8, 9, 2, 6])
```

```
model.layers[0].get_weights()
```

```
[array([[-0.05478922,  0.05496583,  0.04529556, ...,
         -0.05539226,  0.03733491],
       [ 0.01544973,  0.02239056, -0.06521333, ...,
         -0.02017985,  0.03716185],
       [-0.01765968, -0.01163187, -0.06142766, ...,
          0.04832546, -0.02653319],
       ...,
       [ 0.0626147 , -0.0467872 , -0.06959864, ...,
          0.00088954,  0.06301472],
       [ 0.03415244,  0.04304349, -0.04351924, ...,
          0.01030172,  0.06555011],
       [-0.02193814, -0.05472783, -0.01622023, ...,
         -0.02931899,  0.00337418]], dtype=float32),
```

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(X_train,y_train,epochs=15,validation_split=0.2)
```

```
Epoch 1/15
840/840 [==============================]
Epoch 2/15
840/840 [==============================]
Epoch 3/15
840/840 [==============================]
Epoch 4/15
840/840 [==============================]
Epoch 5/15
840/840 [==============================]
```

```
from sklearn.metrics import accuracy_score
y_pred=model.predict(X_test)
y_pred=np.argmax(y_pred,axis=1)
accuracy_score(y_test,y_pred)
```

```
263/263 [==============
0.9565476190476191
```

Practical 3: Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
df=pd.read_csv('/content/drive/MyDrive/sem 3/generative AI/Admission_Predict_Ver1.1.csv')
df
```

	Serial No.	GRE Score	TOEFL Score	Unive
0	1	337	118	
1	2	324	107	
2	3	316	104	
3	4	322	110	
4	5	314	103	
...	...	...	...	
495	496	332	108	
496	497	337	117	
497	498	330	120	
498	499	312	103	
499	500	327	113	

```
df.drop(['Serial No.'],axis=1,inplace=True)
Df
```

	GRE Score	TOEFL Score	Unive
0	337	118	
1	324	107	
2	316	104	
3	322	110	
4	314	103	

```
x=df.drop(['Chance of Admit '],axis=1)
y=df['Chance of Admit ']
X
```

	GRE Score	TOEFL Score	University
0	337	118	
1	324	107	
2	316	104	
3	322	110	
4	314	103	

```
Y
```

```
0       0.92
1       0.76
2       0.72
3       0.80
4       0.65
```

```
df.isnull().sum()
```

```
GRE Score            0
TOEFL Score          0
University Rating    0
SOP                  0
LOR                  0
CGPA                 0
Research             0
Chance of Admit      0
dtype: int64
```

```
for i in x.columns:
  sns.boxplot(x[i])
  plt.show()
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(400, 7)
(100, 7)
(400,)
(100,)
```

```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
print(x_train)
print(x_test)
```

```
[ 0.72       0.64285714  0.5
  1.       ]
[ 0.6        0.32142857  0.25
  0.       ]
[ 0.34       0.46428571  0.25
  0.       ]
[ 0.2        0.25       0.
  0.       ]
[ 0.46       0.5        0.25
  0.       ]
[ 0.74       0.75       0.75
  0.       ]
[ 0.76       0.82142857  0.75
```

```
model=Sequential()
model.add(Dense(12,activation='relu',input_dim=7))
model.add(Dense(8,activation='relu'))
model.add(Dense(1,activation='linear'))
model.summary()
```

```
Model: "sequential"
_________________________________________
 Layer (type)              Output Shape
=========================================
 dense (Dense)             (None, 12)

 dense_1 (Dense)           (None, 8)

 dense_2 (Dense)           (None, 1)
```

```
import tensorflow
from tensorflow.keras import metrics
model.compile(optimizer='adam',
loss='mean_squared_error',metrics=[metrics.RootMeanSquaredError()]) # Compile the model
by specifying the optimizer and loss function.
history=model.fit(x_train,y_train,epochs=200,validation_split=0.2)
```

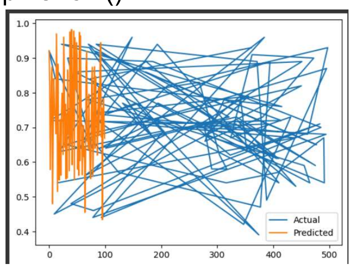
```
y_pred=model.predict(x_test)
```

```
4/4 [==============================] - 0s 3ms/step
```

```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
```

```
0.8009339495678158
```

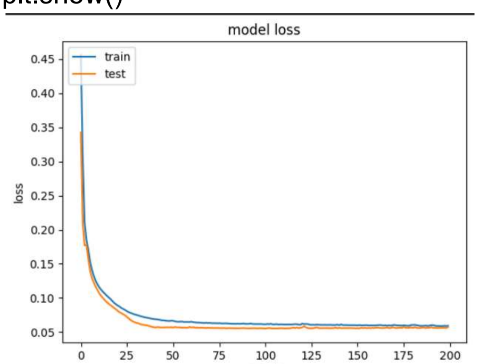
```
plt.plot(y_test,label='Actual')
plt.plot(y_pred,label='Predicted')
plt.legend()
plt.show()
```




```
history.history.keys()
```

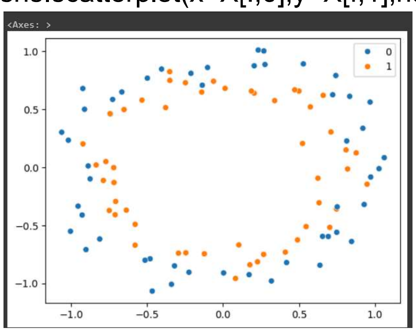
```
dict_keys(['loss', 'root_mean_squared_error', 'val_loss',
'val_root_mean_squared_error'])
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['root_mean_squared_error'])
plt.plot(history.history['val_root_mean_squared_error'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



Practical 4: Early Stopping

```
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import matplotlib.colors as ListedColormap
from pylab import rcParams
from sklearn.datasets import make_circles
from mlxtend.plotting import plot_decision_regions
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Dropout
from sklearn.model_selection import train_test_split
import seaborn as sns
X,y=make_circles(n_samples=100,noise=0.1,random_state=1)
sns.scatterplot(x=X[:,0],y=X[:,1],hue=y)
```

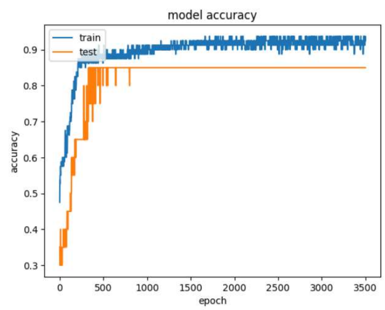



```
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
model=Sequential()
model.add(Dense(256,input_dim=2,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history=model.fit(x_train,y_train,epochs=350,validation_data=(x_test,y_test))
```

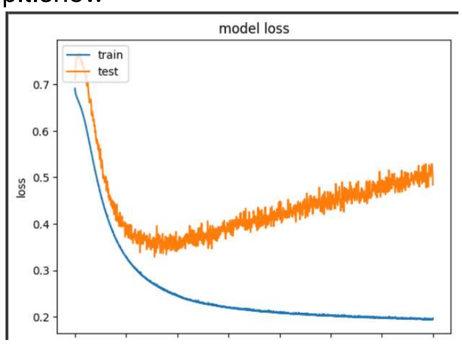
```
Epoch 1103/3500
3/3 [==============================] - 0s 33ms/step - loss: 0.2374 -
Epoch 1104/3500
3/3 [==============================] - 0s 43ms/step - loss: 0.2368 -
Epoch 1105/3500
3/3 [==============================] - 0s 43ms/step - loss: 0.2374 -
Epoch 1106/3500
3/3 [==============================] - 0s 34ms/step - loss: 0.2381 -
Epoch 1107/3500
3/3 [==============================] - 0s 37ms/step - loss: 0.2375 -
Epoch 1108/3500
3/3 [==============================] - 0s 40ms/step - loss: 0.2382 -
Epoch 1109/3500
3/3 [==============================] - 0s 34ms/step - loss: 0.2373 -
Epoch 1110/3500
3/3 [==============================] - 0s 35ms/step - loss: 0.2372 -
Epoch 1111/3500
3/3 [==============================] - 0s 35ms/step - loss: 0.2370 -
Epoch 1112/3500
3/3 [==============================] - 0s 46ms/step - loss: 0.2366 -
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
```

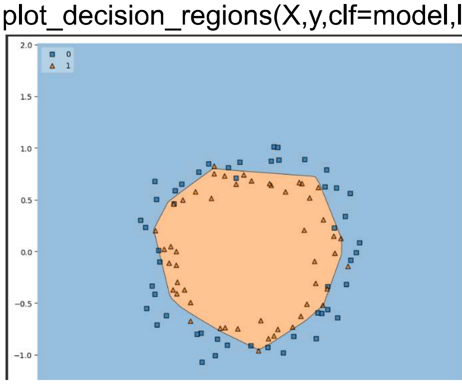
```
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
Plt.show
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show
```




```
plt.figure(figsize=(10,10))
plot_decision_regions(X,y,clf=model,legend=2)
```

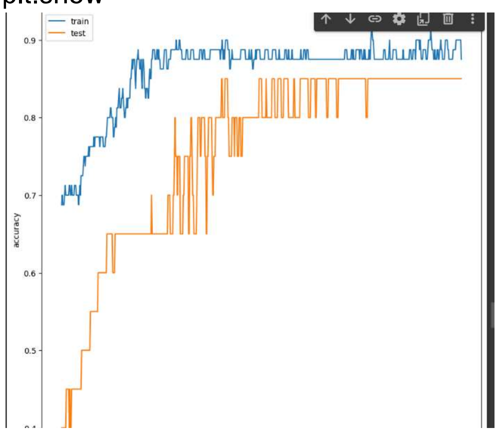


```
model=Sequential()
model.add(Dense(256,input_dim=2,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
callback=EarlyStopping(monitor='val_loss',patience=100,min_delta=0,mode='auto',baseline=N
one,restore_best_weights=False)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
history=model.fit(x_train,y_train,epochs=3500,validation_data=(x_test,y_test),callbacks=[callb
```

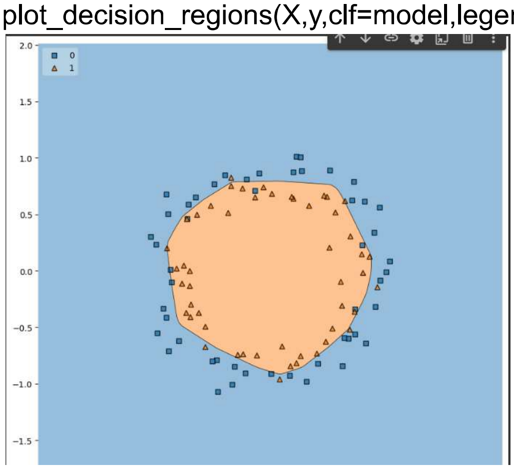
ack])

```
Epoch 1/3500
3/3 [==============================] - 0s 44ms/step
Epoch 2/3500
3/3 [==============================] - 0s 18ms/step
Epoch 3/3500
3/3 [==============================] - 0s 19ms/step
Epoch 4/3500
3/3 [==============================] - 0s 18ms/step
Epoch 5/3500
3/3 [==============================] - 0s 28ms/step
Epoch 6/3500
3/3 [==============================] - 0s 19ms/step
Epoch 7/3500
3/3 [==============================] - 0s 18ms/step
Epoch 8/3500
3/3 [==============================] - 0s 18ms/step
Epoch 9/3500
3/3 [==============================] - 0s 27ms/step
```

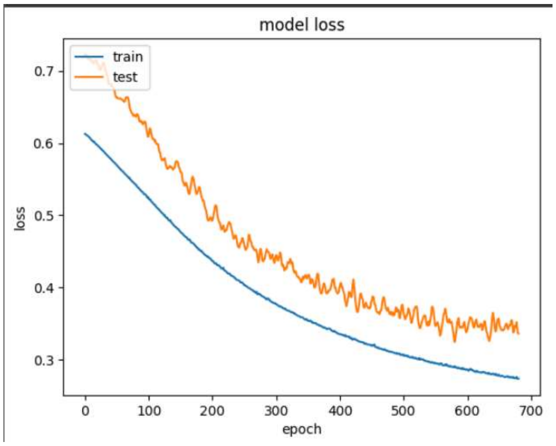
```
plt.figure(figsize=(10,10))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show
```




```
plt.figure(figsize=(10,10))
plot_decision_regions(X,y,clf=model,legend=2)
```

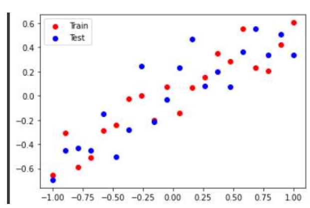


```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show
```



Practical 5: Dropout
5 .1 Regression

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train = np.linspace(-1, 1, 20)
y_train = np.array([-0.6561 , -0.3099 , -0.59035, -0.50855, -0.285  ,
                    -0.2443 , -0.02445,  0.00135, -0.2006 ,  0.07475,
                    -0.1422 ,  0.06515,  0.15265,  0.3521 ,  0.28415,
                    0.5524 ,  0.23115,  0.20835, 0.4211,  0.60485])


X_test = np.linspace(-1, 1, 20)
y_test = np.array([-0.69415, -0.451  , -0.43005, -0.4484 , -0.1475 ,
                   -0.5019 , -0.28055,  0.24595, -0.21425, -0.0286 ,
                   0.23415,  0.46575, 0.07955,  0.1973 ,  0.0719 ,
                   0.3639 ,  0.5536 ,  0.3365 , 0.50705,  0.33435])

plt.scatter(X_train, y_train, c='red', label='Train')
plt.scatter(X_test, y_test, c='blue', label='Test')
plt.legend()
plt.show()
```

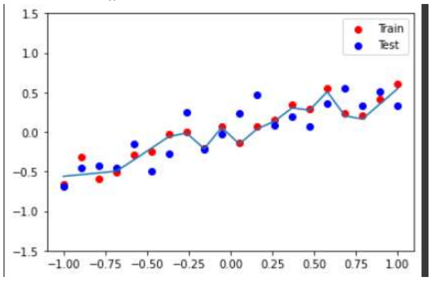



```
model_1 = Sequential()
model_1.add(Dense(128, input_dim=1, activation="relu"))
model_1.add(Dense(128, activation="relu"))
model_1.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_1.compile(loss='mse', optimizer=adam, metrics=['mse'])
history = model_1.fit(X_train, y_train, epochs=500,
                    validation_data = (X_test, y_test),
                    verbose=False)
# evaluate the model
_, train_mse = model_1.evaluate(X_train, y_train, verbose=0)
_, test_mse = model_1.evaluate(X_test, y_test, verbose=0)
print('Train: {}, Test: {}'.format(train_mse, test_mse))
```

```
Train: 0.004562994930893183, Test: 0.04608117789030075
```

```
y_pred_1 = model_1.predict(X_test)
```

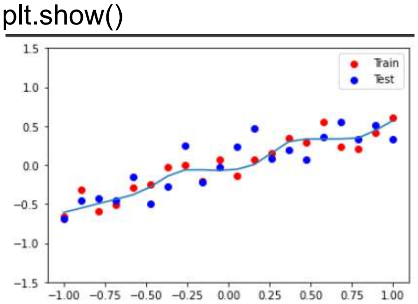
```
plt.figure()
plt.scatter(X_train, y_train, c='red', label='Train')
plt.scatter(X_test, y_test, c='blue', label='Test')
plt.plot(X_test, y_pred_1)
plt.legend()
plt.ylim((-1.5, 1.5))
plt.show()
```




```
model_2 = Sequential()
model_2.add(Dense(128, input_dim=1, activation="relu"))
model_2.add(Dropout(0.2))
model_2.add(Dense(128, activation="relu"))
model_2.add(Dropout(0.2))
model_2.add(Dense(1, activation="linear"))
adam = Adam(learning_rate=0.01)
model_2.compile(loss='mse', optimizer=adam, metrics=['mse'])

drop_out_history = model_2.fit(X_train, y_train, epochs=500,
                               validation_data = (X_test, y_test),
                               verbose=False)

# evaluate the model
_, train_mse = model_2.evaluate(X_train, y_train, verbose=0)
_, test_mse = model_2.evaluate(X_test, y_test, verbose=0)
print('Train: {}, Test: {}'.format(train_mse, test_mse))
```

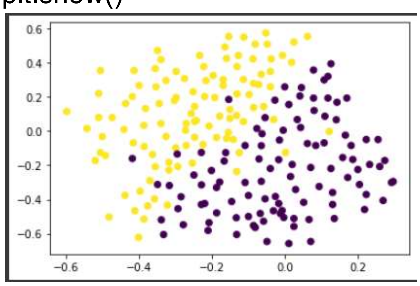
```
Train: 0.011907287873327732, Test: 0.037526600006284714
```

```
y_pred_2 = model_2.predict(X_test)
plt.figure()
plt.scatter(X_train, y_train, c='red', label='Train')
plt.scatter(X_test, y_test, c='blue', label='Test')
plt.plot(X_test, y_pred_2)
plt.legend()
plt.ylim((-1.5, 1.5))
plt.show()
```



5.2 Classification

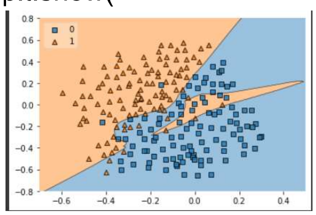
```
import numpy as np
import matplotlib.pyplot as plt
X = np.array([[-1.58986e-01,  4.23977e-01],
       [-3.47926e-01,  4.70760e-01],
       [-5.04608e-01,  3.53801e-01],
       [-5.96774e-01,  1.14035e-01],
       [-5.18433e-01, -1.72515e-01],
       [-2.92627e-01, -2.07602e-01],
       [-1.58986e-01, -4.38596e-02],
       [-5.76037e-02,  1.43275e-01],
       [-7.14286e-02,  2.71930e-01],
       [-2.97235e-01,  3.47953e-01],
       [-4.17051e-01,  2.01754e-01],
y = np.array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
import matplotlib.pyplot as plt
plt.scatter(X[:,0], X[:,1], c=y)
plt.show()
```



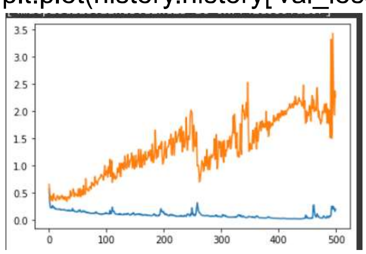

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam

model = Sequential()

model.add(Dense(128, input_dim=2, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

adam = Adam(learning_rate=0.01)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history = model.fit(X, y, epochs=500, validation_split = 0.2,verbose=1)
```

```
Epoch 1/500
6/6 [==============================] -
Epoch 2/500
6/6 [==============================] -
Epoch 3/500
6/6 [==============================] -
Epoch 4/500
6/6 [==============================] -
Epoch 5/500
```

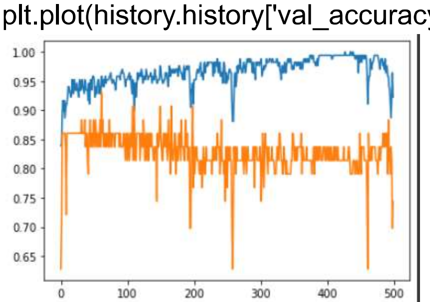
```
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X, y.astype('int'), clf=model, legend=2)
plt.xlim(-0.7,0.5)
plt.ylim(-0.8,0.8)
plt.show(
```




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




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

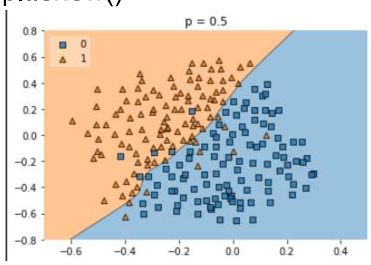



```
model = Sequential()

model.add(Dense(128, input_dim=2, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(1, activation="sigmoid"))
```

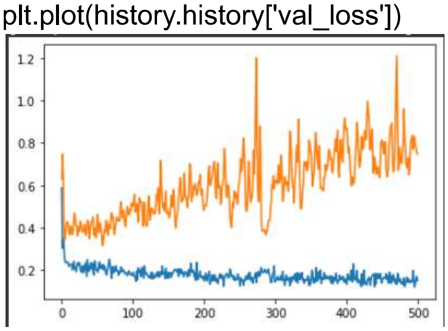
```
adam = Adam(learning_rate=0.01)
model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])

history = model.fit(X, y, epochs=500, validation_split = 0.2,verbose=1)
```

```
Epoch 1/500
6/6 [==============================] - 1s 78ms/step - loss: 0.5846 - accuracy:
Epoch 2/500
6/6 [==============================] - 0s 6ms/step - loss: 0.3438 - accuracy: 0
Epoch 3/500
6/6 [==============================] - 0s 7ms/step - loss: 0.2886 - accuracy: 0
Epoch 4/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2981 - accuracy: 0
Epoch 5/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2408 - accuracy: 0
Epoch 6/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2541 - accuracy: 0
Epoch 7/500
6/6 [==============================] - 0s 7ms/step - loss: 0.2637 - accuracy: 0
Epoch 8/500
6/6 [==============================] - 0s 8ms/step - loss: 0.2515 - accuracy: 0
Epoch 9/500
6/6 [==============================] - 0s 11ms/step - loss: 0.2342 - accuracy:
Epoch 10/500
6/6 [==============================] - 0s 6ms/step - loss: 0.2535 - accuracy: 0
```

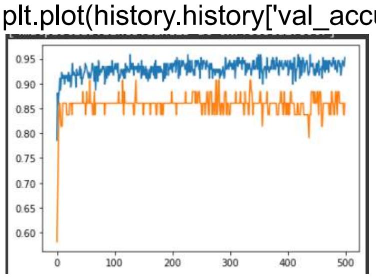
```
plot_decision_regions(X, y.astype('int'), clf=model, legend=2)
plt.xlim(-0.7,0.5)
plt.ylim(-0.8,0.8)
plt.title('p = 0.5')
plt.show()
```




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




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```



Practical 6: RNN

```
from keras.datasets import imdb
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras import Sequential
from keras.layers import Dense, Embedding, Flatten, SimpleRNN
(x_train, y_train), (x_test, y_test) = imdb.load_data()
X_train.shape
```

```
(25000,)
```

```
x_train[0]
```

```
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
```

```
x_train = pad_sequences(x_train, padding="post", maxlen=50)
x_test = pad_sequences(x_test, padding="post", maxlen=50)
model = Sequential()
model.add(SimpleRNN(32,input_shape=(50,1),return_sequences=False))
model.add(Dense(1, activation="sigmoid"))
model.summary()
```

```
Model: "sequential_2"
_________________________________________
 Layer (type)                Output Sh
=========================================
 simple_rnn_1 (SimpleRNN)    (None, 32

 dense_1 (Dense)             (None, 1)

=========================================
Total params: 1121 (4.38 KB)
Trainable params: 1121 (4.38 KB)
Non-trainable params: 0 (0.00 Byte)
```

```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])
model.fit(x_train, y_train, epochs=50, validation_data=[x_test, y_test])
```

```
Epoch 1/50
782/782 [==============================] - 12s 14ms/step - loss: 0.6941 - accura
Epoch 2/50
782/782 [==============================] - 14s 17ms/step - loss: 0.6932 - accura
Epoch 3/50
782/782 [==============================] - 11s 15ms/step - loss: 0.6927 - accura
Epoch 4/50
782/782 [==============================] - 11s 14ms/step - loss: 0.6928 - accura
Epoch 5/50
782/782 [==============================] - 13s 17ms/step - loss: 0.6927 - accura
Epoch 6/50
782/782 [==============================] - 11s 14ms/step - loss: 0.6925 - accura
Epoch 7/50
782/782 [==============================] - 11s 14ms/step - loss: 0.6925 - accura
Epoch 8/50
782/782 [==============================] - 13s 16ms/step - loss: 0.6927 - accura
```

Practical 7: CNN

```
!kaggle datasets download -d andrewmvd/pediatric-pneumonia-chest-xray
import zipfile
zip_ref = zipfile.ZipFile('/content/pediatric-pneumonia-chest-xray.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
```

```
Found 5232 files belonging to 2 classes
```

```
test_data = keras.utils.image_dataset_from_directory(
    directory = '/content/Pediatric Chest X-ray Pneumonia/test',
    labels = 'inferred',
    label_mode = 'int',
    batch_size = 32,
    image_size = (256,256)
)
```

```
Found 624 files belonging to 2 classes.
```

```
def normalize(image, label):
  image = tf.cast(image/255. , tf.float32)
  return image, label

tr_data = tr_data.map(normalize)
test_data = test_data.map(normalize)
model = Sequential()
model.add(Conv2D(32,(3,3),activation='relu', input_shape=(256,256,3))) #32 filters of 3,3
model.add(MaxPool2D((2,2), strides=2, padding='valid'))
model.add(Conv2D(64,(3,3),activation='relu', input_shape=(256,256,3)))
model.add(MaxPool2D((2,2), strides=2, padding='valid'))

model.add(Conv2D(128,(3,3),activation='relu', input_shape=(256,256,3)))
model.add(MaxPool2D((2,2), strides=2, padding='valid'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

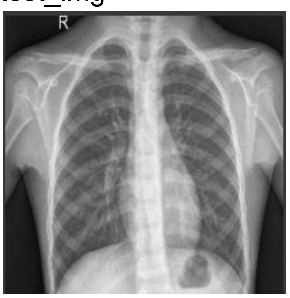
```
Model: "sequential_1"
```

Layer (type)	Output Shape
conv2d_3 (Conv2D)	(None, 254, 254, 32)
max_pooling2d_3 (MaxPooling2D)	(None, 127, 127, 32)

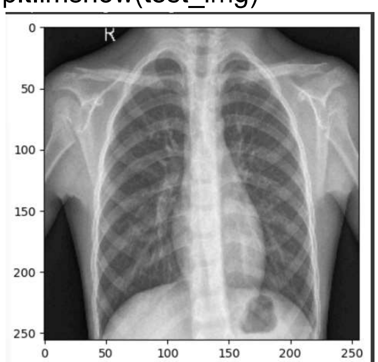
```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(tr_data, epochs=10, validation_data=test_data)
```

```
Epoch 1/10
164/164 ━━━━━━━━━━━━━━━━━━━━ 56s 281ms/step - accuracy: 0.7987 -
Epoch 2/10
164/164 ━━━━━━━━━━━━━━━━━━━━ 40s 241ms/step - accuracy: 0.9565 -
Epoch 3/10
164/164 ━━━━━━━━━━━━━━━━━━━━ 41s 241ms/step - accuracy: 0.9721 -
Epoch 4/10
164/164 ━━━━━━━━━━━━━━━━━━━━ 43s 253ms/step - accuracy: 0.9825 -
Epoch 5/10
164/164 ━━━━━━━━━━━━━━━━━━━━ 82s 254ms/step - accuracy: 0.9814 -
```

```
import cv2
test_img = keras.utils.load_img('/content/Pediatric Chest X-ray
Pneumonia/test/NORMAL/IM-0001-0001.jpeg', target_size=(256,256))
test_img
```




```
from matplotlib import pyplot as plt
plt.imshow(test_img)
```




```
import numpy as np
test_img = np.array(test_img)
test_img.shape
```

```
(256, 256, 3)
```

```
test_img = test_img.reshape((1,256,256,3))
if model.predict(test_img) == 1:
  print('Pneumonia')
else:
  print('Normal')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 924ms/step
Normal
```

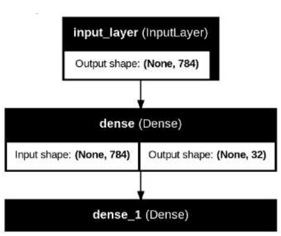
Practical 8: Autoencoders
8.1 Vanilla Autoencoder

```
import keras
from keras import layers, Input,Model
from keras.layers import Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import plot_model
import numpy as np
import matplotlib.pyplot as plt

encoding_dim=32
input_img=Input(shape=(784,))
encoded=Dense(encoding_dim,activation='relu')(input_img)
decoded=Dense(784,activation='sigmoid')(encoded)


from enum import auto
autoencoder=Model(input_img,decoded)
encoder=Model(input_img,encoded)
encoded_input=Input(shape=(encoding_dim,))
decoder_layer=autoencoder.layers[-1]
decoder=Model(encoded_input,decoder_layer(encoded_input))
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
autoencoder.summary()
plot_model(autoencoder,to_file='model_plot.png',show_shapes=True,show_layer_names=
True)
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 784)	0
dense (Dense)	(None, 32)	25,120
dense_1 (Dense)	(None, 784)	25,872

Total params: 50,992 (199.19 KB)
Trainable params: 50,992 (199.19 KB)
Non-trainable params: 0 (0.00 B)




```
(X_train,_),(X_test,_) = mnist.load_data()
X_train = X_train.astype('float32')/255.0
X_test = X_test.astype('float32')/255.0
X_train = X_train.reshape((len(X_train),np.prod(X_train.shape[1:])))
```

```
X_test = X_test.reshape((len(X_test),np.prod(X_test.shape[1:])))
print(X_train.shape)
print(X_test.shape)
```

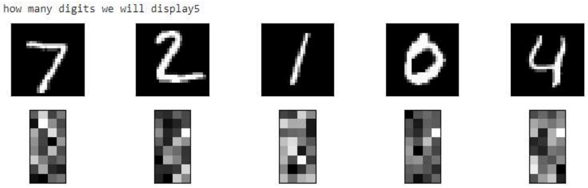
```
(60000, 784)
(10000, 784)
```

```
autoencoder.fit(X_train,X_train,epochs=10,batch_size=256,shuffle=True,validation_data=(
X_test,X_test))
encoded_imgs=encoder.predict(X_test)
decoded_imgs=decoder.predict(encoded_imgs)
```

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0962 - val_loss: 0.0943
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0953 - val_loss: 0.0937
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0948 - val_loss: 0.0932
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0945 - val_loss: 0.0930
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.0940 - val_loss: 0.0927
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0939 - val_loss: 0.0925
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0936 - val_loss: 0.0924
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0936 - val_loss: 0.0923
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0936 - val_loss: 0.0922
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.0934 - val_loss: 0.0921
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
 n= int(input('how many digits we will display'))
 plt.figure(figsize=(40,4))
 for i in range(n):

  ax=plt.subplot(3,20,i+1)
  plt.imshow(X_test[i].reshape(28,28))
  plt.gray()

  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
  ax=plt.subplot(3,20,i+1+20)
  plt.imshow(encoded_imgs[i].reshape(8,4))
  plt.gray()

  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

```
how many digits we will display5
```



8.2 Denoising Autoencoder

```
import keras
from keras.datasets import mnist
from keras import layers
from keras.callbacks import TensorBoard
from keras.models import Sequential
import numpy as np
import matplotlib.pyplot as plt

(X_train,_), (X_test,_) = mnist.load_data()
X_train = X_train.astype('float32') / 255.
X_test = X_test.astype('float32') / 255.
X_train=np.reshape(X_train,(len(X_train),28,28,1))
X_test=np.reshape(X_test,(len(X_test),28,28,1))
noise_factor=0.5
X_train_noisy=X_train+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_train.shape)
X_test_noisy=X_test+noise_factor*np.random.normal(loc=0.0,scale=1.0,size=X_test.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
X_train_noisy=np.clip(X_train_noisy,0.,1.)
X_test_noisy=np.clip(X_test_noisy,0.,1.)
print(X_train.shape)
print(X_test.shape)
```

```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
```

```
n=10
plt.figure(figsize=(20,5))
for i in range(1,n+1):
    ax=plt.subplot(1,n,i)
    plt.imshow(X_train_noisy[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
input_img=keras.Input(shape=(28,28,1))
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(input_img)
x=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
encoded=layers.MaxPooling2D((2,2),padding='same')(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(encoded)
x=layers.UpSampling2D((2,2))(x)
x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)
x=layers.UpSampling2D((2,2))(x)
decoded=layers.Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)
autoencoder=keras.Model(input_img,decoded)
autoencoder.compile(optimizer='adam',loss='binary_crossentropy')
autoencoder.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	9,248
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 32)	9,248
up_sampling2d (UpSampling2D)	(None, 14, 14, 32)	0
conv2d_3 (Conv2D)	(None, 14, 14, 32)	9,248
up_sampling2d_1 (UpSampling2D)	(None, 28, 28, 32)	0
conv2d_4 (Conv2D)	(None, 28, 28, 1)	289

```
Total params: 28,353 (110.75 KB)
Trainable params: 28,353 (110.75 KB)
Non-trainable params: 0 (0.00 B)
```

```
autoencoder.fit(X_train_noisy,X_train,epochs=3,batch_size=128,shuffle=True,validation_d
ata=(X_test_noisy,X_test),callbacks=[TensorBoard(log_dir='./tmo/tb',histogram_freq=0,writ
e_graph=False)])
predictions=autoencoder.predict(X_test_noisy)
```

```
Epoch 1/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - loss: 0.1049 - val_loss: 0.1023
Epoch 2/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.1027 - val_loss: 0.1008
Epoch 3/3
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.1012 - val_loss: 0.0998
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
m=10
plt.figure(figsize=(20,2))
for i in range(1,m+1):
    ax=plt.subplot(1,m,i)
    plt.imshow(predictions[i].reshape(28,28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Practical 9: GAN’s

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
from torchvision import transforms, datasets
import matplotlib.pyplot as plt
import numpy as np

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=32, shuffle=True)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:03<00:00, 43184268.97it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
```

```
latent_dim = 100
lr = 0.0002
beta1 = 0.5
beta2 = 0.999
num_epochs = 10

class Generator(nn.Module):
  def __init__(self):
    super(Generator, self).__init__()

    self.model = nn.Sequential(
        nn.Linear(latent_dim, 128*8*8),
        nn.ReLU(),
        nn.Unflatten(1,(128,8,8)),
        nn.Upsample(scale_factor=2),
        nn.Conv2d(128, 128, kernel_size=3, padding=1),
        nn.BatchNorm2d(128,momentum=0.78),
        nn.ReLU(),
        nn.Upsample(scale_factor=2),
        nn.Conv2d(128, 64, kernel_size=3, padding=1),
        nn.BatchNorm2d(64,momentum=0.78),
        nn.ReLU(),
        nn.Conv2d(64, 3, kernel_size=3, padding=1),
        nn.Tanh()
```

```
  )

 def forward(self, z):
  img = self.model(z)
  return img

class Generator(nn.Module):
 def __init__(self):
  super(Generator, self).__init__()

  self.model = nn.Sequential(
    nn.Linear(latent_dim, 128*8*8),
    nn.ReLU(),
    nn.Unflatten(1,(128,8,8)),
    nn.Upsample(scale_factor=2),
    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.BatchNorm2d(128,momentum=0.78),
    nn.ReLU(),
    nn.Upsample(scale_factor=2),
    nn.Conv2d(128, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64,momentum=0.78),
    nn.Relu(),
    nn.Conv2d(64, 3, kernel_size=3, padding=1),
    nn.Tanh()
  )

 def forward(self, img):
  validity = self.model(img)
  return validity

generator = Generator().to(device)
discriminator = Discriminator().to(device)

adversarial_loss = nn.BCELoss()

optimizer_G = optim.Adam(generator.parameters(), lr=lr, betas=(beta1, beta2))
optimizer_D = optim.Adam(discriminator.parameters(), lr=lr, betas=(beta1, beta2))

for epoch in range(num_epochs):
 for i, batch in enumerate(dataloader):
  real_images = batch[0].to(device)

  valid = torch.ones(real_images.size(0), 1).to(device)
  fake = torch.zeros(real_images.size(0), 1).to(device)

  real_images = real_images.to(device)

  #Train the discriminator
  optimizer_D.zero_grad()
```

```
    z = torch.randn(real_images.size(0), latent_dim, device=device)
    fake_images = generator(z)

    real_loss = adversarial_loss(discriminator(real_images), valid)
    fake_loss = adversarial_loss(discriminator(fake_images.detach()), fake)
    d_loss = (real_loss + fake_loss)/2

    d_loss.backward()
    optimizer_D.step()

    # Train the generator
    optimizer_G.zero_grad()
    gen_images = generator(z)

    g_loss = adversarial_loss(discriminator(fake_images), valid)

    g_loss.backward()
    optimizer_G.step()

    if (i+1)%100 == 0:
      print(f"Epoch [{epoch+1}/{num_epochs}], Batch [{i+1}/{len(dataloader)}], Discriminator
loss: {d_loss.item():.4f}, Generator loss: {g_loss.item():.4f}")

    if (epoch+1)%10 == 0:
      with torch.no_grad():
        z = torch.randn(16, latent_dim, device=device)
        generated = generator(z).detach().cpu()
        grid = torchvision.utils.make_grid(generated, nrow=4, normalize=True)
        plt.imshow(np.transpose(grid, (1,2,0)))
        plt.axis('off')
        plt.show()
```

```
Epoch [1/10], Batch [100/1563], Discriminator loss: 0.4990, Generator loss: 1.8431
Epoch [1/10], Batch [200/1563], Discriminator loss: 0.4044, Generator loss: 1.3375
Epoch [1/10], Batch [300/1563], Discriminator loss: 0.5269, Generator loss: 1.2825
Epoch [1/10], Batch [400/1563], Discriminator loss: 0.5489, Generator loss: 1.0057
Epoch [1/10], Batch [500/1563], Discriminator loss: 0.8894, Generator loss: 1.6310
Epoch [1/10], Batch [600/1563], Discriminator loss: 0.5873, Generator loss: 1.3102
Epoch [1/10], Batch [700/1563], Discriminator loss: 0.4926, Generator loss: 1.4184
```