1. Binary
2. Multiclass
3. Multi label

```
from sklearn.datasets import make_circles
n_samples =1000
X,Y= make_circles(n_samples,noise=0.03,random_state=42)
print(X.shape)
print(Y.shape)
```

```
(1000, 2)
(1000,)
```

```
X
```

```
array([[ 0.75424625,  0.23148074],
       [-0.75615888,  0.15325888],
       [-0.81539193,  0.17328203],
       ...,
       [-0.13690036, -0.81001183],
       [ 0.67036156, -0.76750154],
       [ 0.28105665,  0.96382443]])
```

```
Y[:10]
```

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

```
import pandas as pd
circles= pd.DataFrame({'X1':X[:,0],'X2':X[:,1],'label':Y})
circles
```

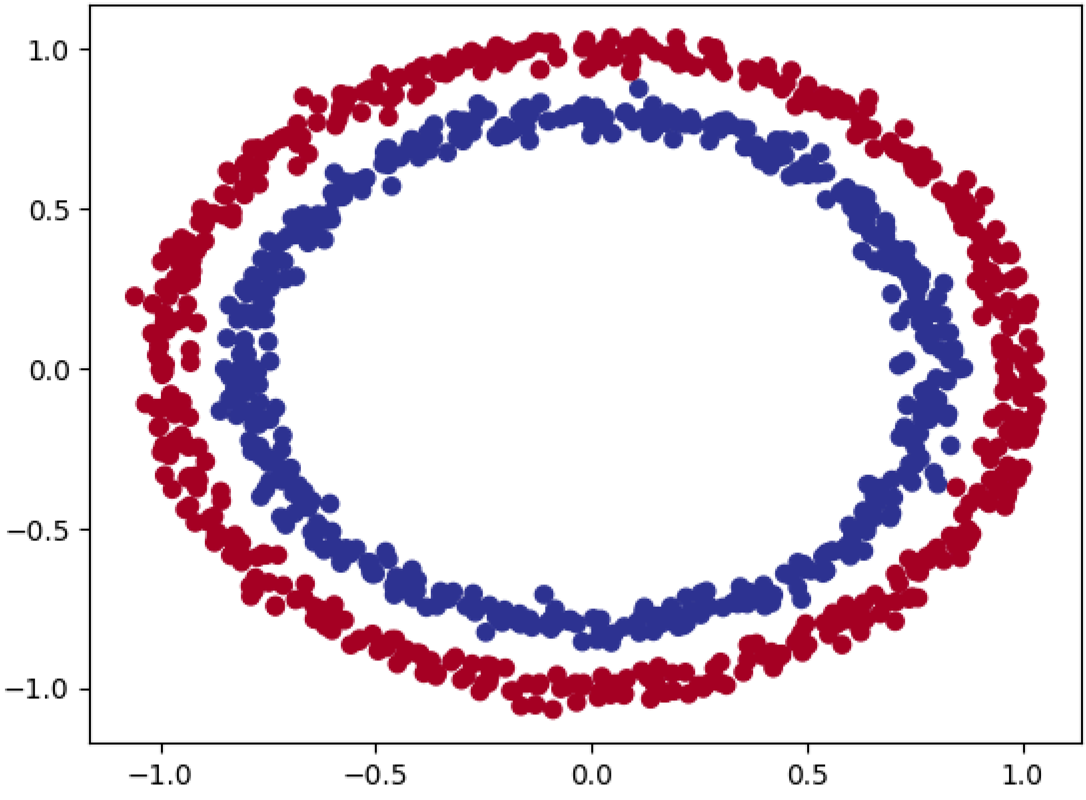
	X1	X2	label
0	0.754246	0.231481	1
1	-0.756159	0.153259	1
2	-0.815392	0.173282	1
3	-0.393731	0.692883	1
4	0.442208	-0.896723	0
...	...	...	...
995	0.244054	0.944125	0
996	-0.978655	-0.272373	0
997	-0.136900	-0.810012	1
998	0.670362	-0.767502	0
999	0.281057	0.963824	0

1000 rows × 3 columns

Next steps: Generate code with `circles` View recommended plots New interactive sheet

```
import matplotlib.pyplot as plt
plt.scatter(X[:,0],X[:,1],c=Y,cmap=plt.cm.RdYlBu)
```

```
<matplotlib.collections.PathCollection at 0x7b81e28f8af0>
```



Generate print hello world using rot13 Close

```
X.shape,Y.shape
```

```
((1000, 2), (1000,))
```

```
X[0],Y[0]
```

```
(array([0.75424625, 0.23148074]), 1)
```

```
import tensorflow as tf
tf.random.set_seed(42)
model_1=tf.keras.Sequential([
    tf.keras.layers.Dense(1)
])
model_1.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=["accuracy"])
```

```
model_1.fit(X,Y,epochs=200,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7b818d09aef0>
```

```
model_1.evaluate(X,Y)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4955 - loss: 8.1322
[8.059046745300293, 0.5]
```

```
import tensorflow as tf
tf.random.set_seed(42)
model_2=tf.keras.Sequential([
    tf.keras.layers.Dense(1),
    tf.keras.layers.Dense(1)

])
model_2.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.SGD(),
                metrics=["accuracy"])
```

```
model_2.fit(X,Y,epochs=200,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7b818d071180>
```

```
model_2.evaluate(X,Y)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4955 - loss: 0.6932
[0.6932107210159302, 0.5]
```

```
import tensorflow as tf
tf.random.set_seed(42)
```

```
model_3=tf.keras.Sequential([
    tf.keras.layers.Dense(100),
    tf.keras.layers.Dense(10),
    tf.keras.layers.Dense(1)

])
model_3.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])


model_3.fit(X,Y,epochs=100,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7b818c2ff340>
```

```
model_3.evaluate(X,Y)
```

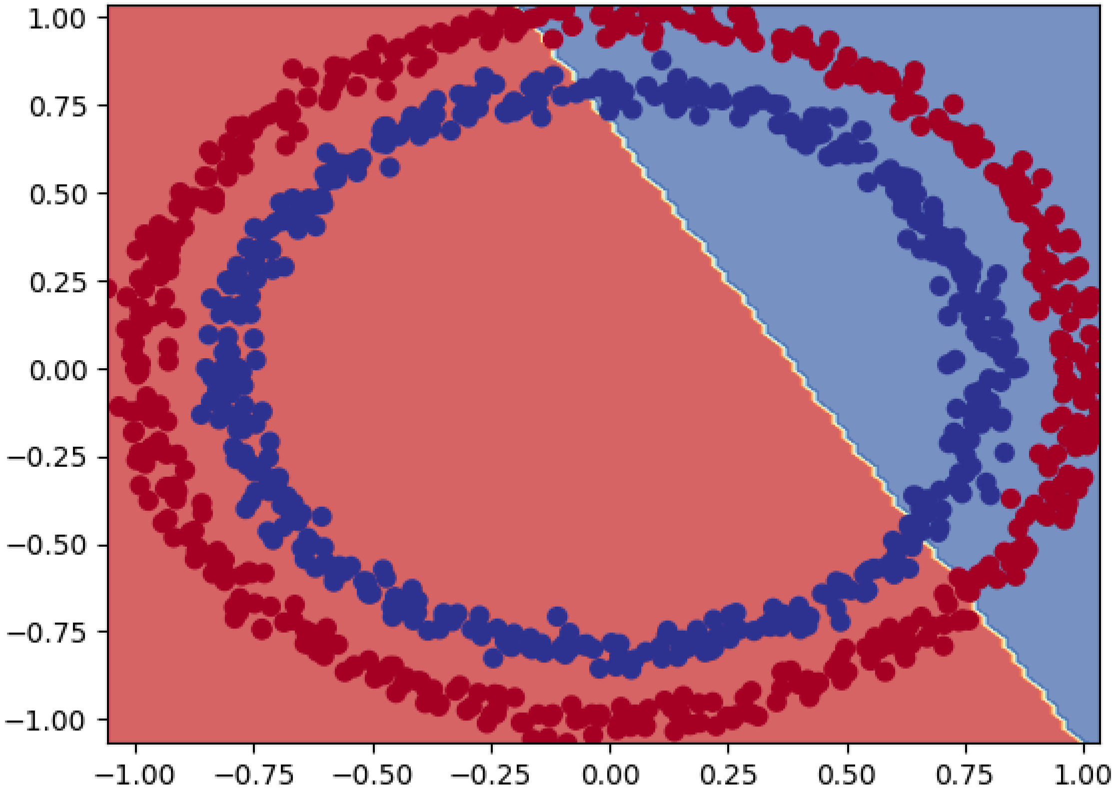
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5036 - loss: 0.6917
[0.6945011019706726, 0.48500001430511475]
```

1. create a meshgrid for different x values
2. make predictions across the meshgrid
3. plot the predictions as well as line b/w zeros

```
import numpy as np
def plot_decision_boundary(model,X,Y):
  # plt.figure(figsize=(12,8))
  # X=tf.cast(X,tf.float32)
  # Y=tf.cast(Y,tf.float32)
  x_min,x_max=tf.reduce_min(X[:,0]),tf.reduce_max(X[:,0])
  y_min,y_max=tf.reduce_min(X[:,1]),tf.reduce_max(X[:,1])
  xx,yy=np.meshgrid(np.linspace(x_min,x_max,100),np.linspace(y_min,y_max,100))
  x_in = np.c_[xx.ravel(),yy.ravel()]#stack 2d arrays togeather
  y_pred = model.predict(x_in)
  if(len(y_pred[0])) >1:
    print("doing multiclass classification")
    y_pred=np.argmax(y_pred,axis=1).reshape(xx.shape)
  else:
    print("doing binary classification")
    y_pred=np.round(y_pred).reshape(xx.shape)
  plt.contourf(xx,yy,y_pred.reshape(xx.shape),cmap=plt.cm.RdYlBu,alpha=0.7)
  plt.scatter(X[:,0],X[:,1],c=Y,s=40,cmap=plt.cm.RdYlBu)
  plt.xlim(xx.min(),xx.max())
  plt.ylim(yy.min(),yy.max())


plot_decision_boundary(model_3,X,Y)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
doing binary classification
```



Suggested code may be subject to a licence |

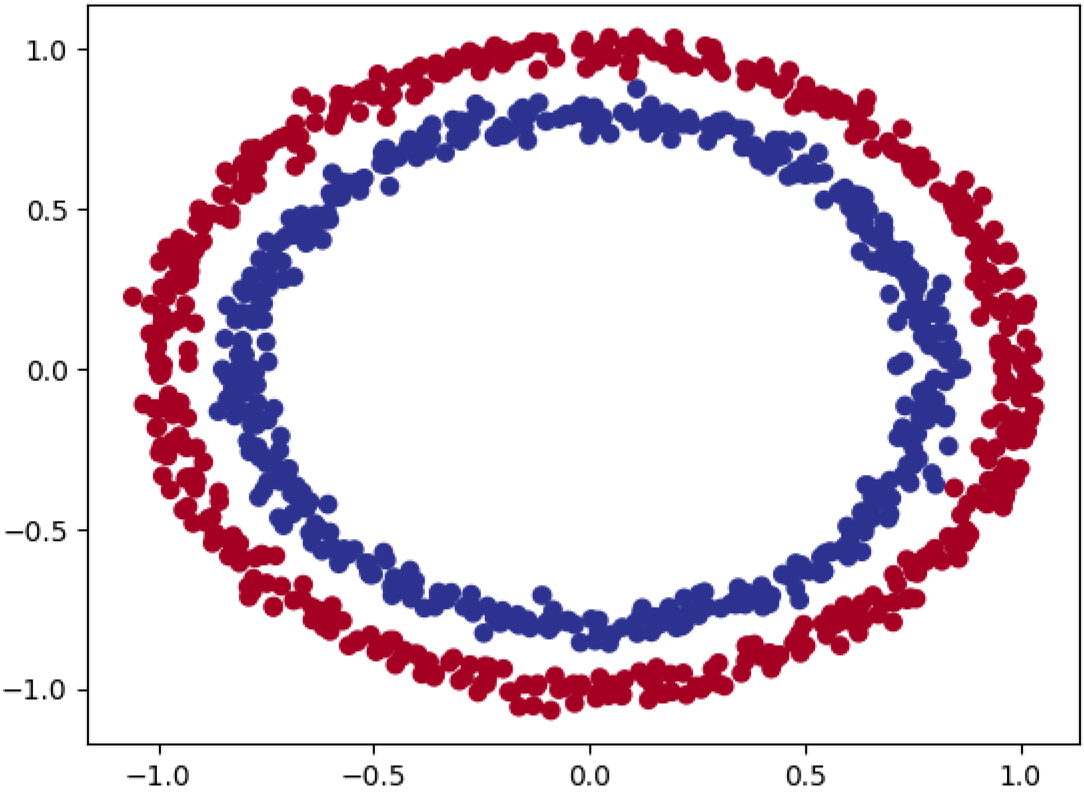
```
tf.random.set_seed(42)
model_4=tf.keras.Sequential([
    tf.keras.layers.Dense(1,activation="linear"),
])
model_4.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=["accuracy"])


history= model_4.fit(X,Y,epochs=100)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5093 - loss: 0.6923
Epoch 82/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5093 - loss: 0.6922
Epoch 83/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5074 - loss: 0.6922
Epoch 84/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5065 - loss: 0.6922
Epoch 85/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5065 - loss: 0.6922
Epoch 86/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5065 - loss: 0.6922
Epoch 87/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5065 - loss: 0.6922
Epoch 88/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5065 - loss: 0.6922
Epoch 89/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5090 - loss: 0.6922
Epoch 90/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5090 - loss: 0.6922
Epoch 91/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5090 - loss: 0.6922
Epoch 92/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5090 - loss: 0.6922
Epoch 93/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5101 - loss: 0.6923
Epoch 94/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5103 - loss: 0.6923
Epoch 95/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5095 - loss: 0.6923
Epoch 96/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5095 - loss: 0.6923
Epoch 97/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5095 - loss: 0.6923
Epoch 98/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5106 - loss: 0.6923
Epoch 99/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5106 - loss: 0.6924
Epoch 100/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5126 - loss: 0.6924
```

```
plt.scatter(X[:,0],X[:,1],c=Y,cmap=plt.cm.RdYlBu)
```

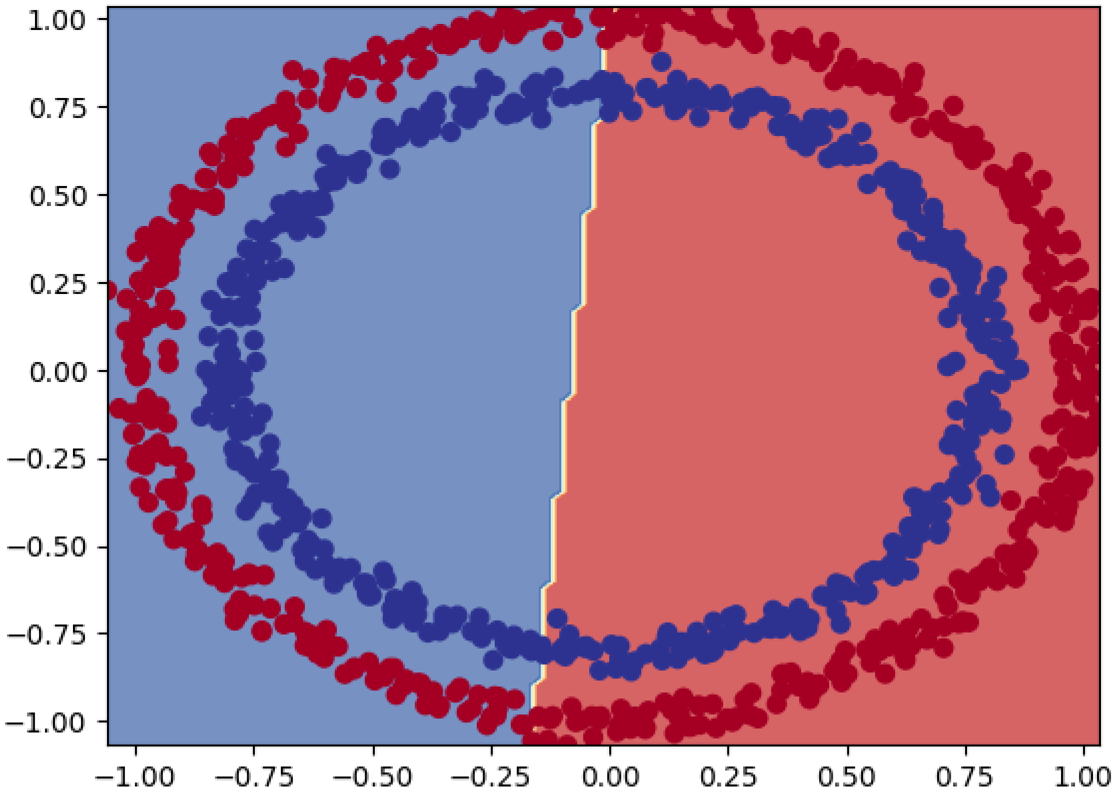
```
<matplotlib.collections.PathCollection at 0x7b817ff47700>
```



Generate | a slider using jupyter widgets | Close

```
plot_decision_boundary(model_4,X,Y)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
doing binary classification
```



```
#non linear activation function
tf.random.set_seed(42)
model_5=tf.keras.Sequential([
    tf.keras.layers.Dense(1,activation="relu"),
])
model_5.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=["accuracy"])


model_5.fit(X,Y,epochs=100)
```

```
Epoch 96/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4549 - loss: 4.7938
Epoch 97/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4549 - loss: 4.7900
Epoch 98/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4549 - loss: 4.7870
Epoch 99/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4549 - loss: 4.7844
Epoch 100/100
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4549 - loss: 4.7820
<keras.src.callbacks.history.History at 0x7b818571b6d0>
```

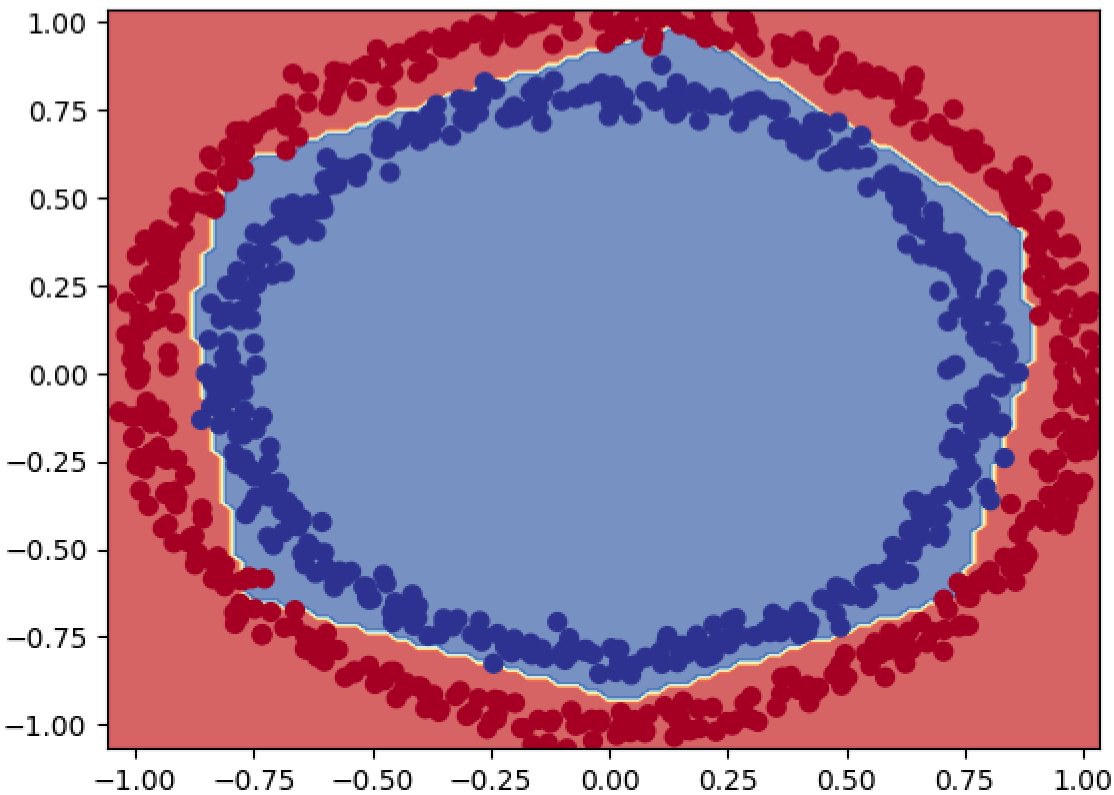
```
tf.random.set_seed(42)
model_6=tf.keras.Sequential([
    tf.keras.layers.Dense(4,activation='relu'),
    tf.keras.layers.Dense(4,activation='relu'),
    tf.keras.layers.Dense(1,activation="sigmoid"),
])
model_6.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=["accuracy"])
```

```
model_6.fit(X,Y,epochs=300,verbose=0)
```

```
<keras.src.callbacks.history.History at 0x7b8185c252d0>
```

```
plot_decision_boundary(model_6,X,Y)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
doing binary classification
```




```
model_6.evaluate(X,Y)
```

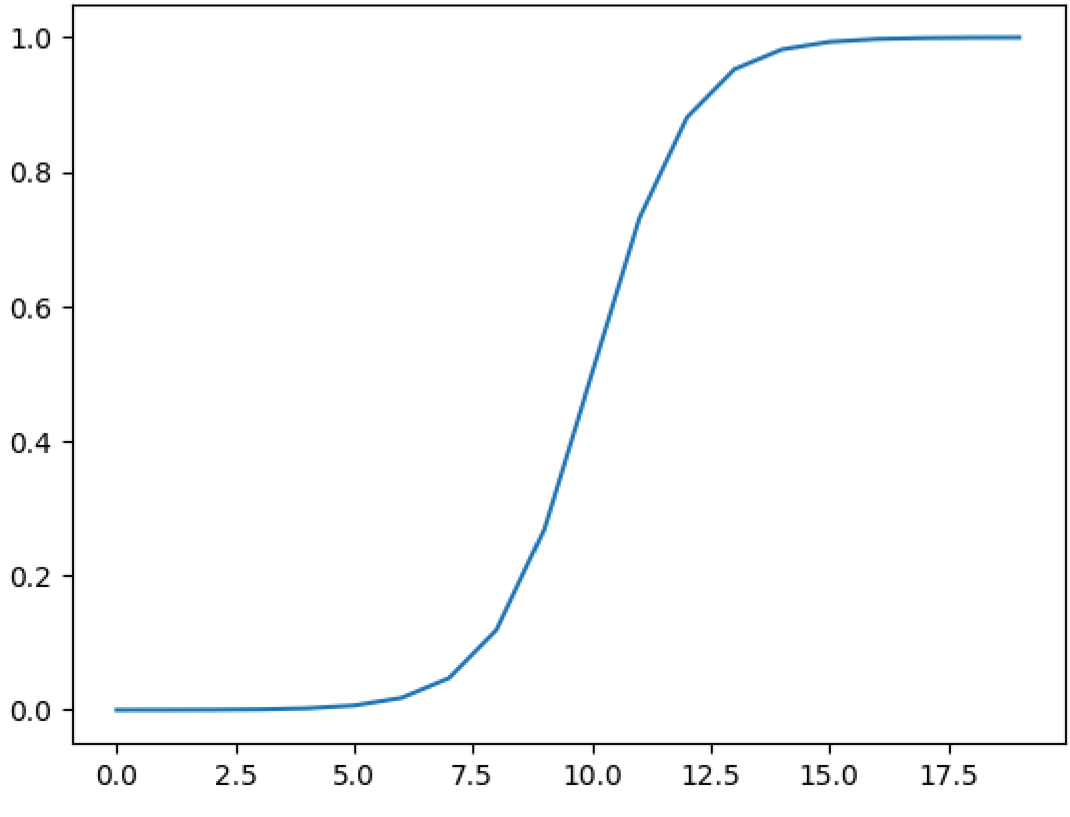
```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9848 - loss: 0.0903
[0.0886337161064148, 0.984000027179718]
```

```
#sigmoid
A= tf.cast(tf.range(-10,10),tf.float32)
def sigmoid(x):
  return 1/(1+tf.exp(-x))
sigmoid(A)
```

```
<tf.Tensor: shape=(20,), dtype=float32, numpy=
array([4.5397872e-05, 1.2339458e-04, 3.3535014e-04, 9.1105117e-04,
       2.4726233e-03, 6.6928510e-03, 1.7986210e-02, 4.7425874e-02,
       1.1920292e-01, 2.6894143e-01, 5.0000000e-01, 7.3105860e-01,
       8.8079703e-01, 9.5257413e-01, 9.8201376e-01, 9.9330717e-01,
       9.9752742e-01, 9.9908900e-01, 9.9966466e-01, 9.9987662e-01],
      dtype=float32)>
```

```
plt.plot(sigmoid(A))
```

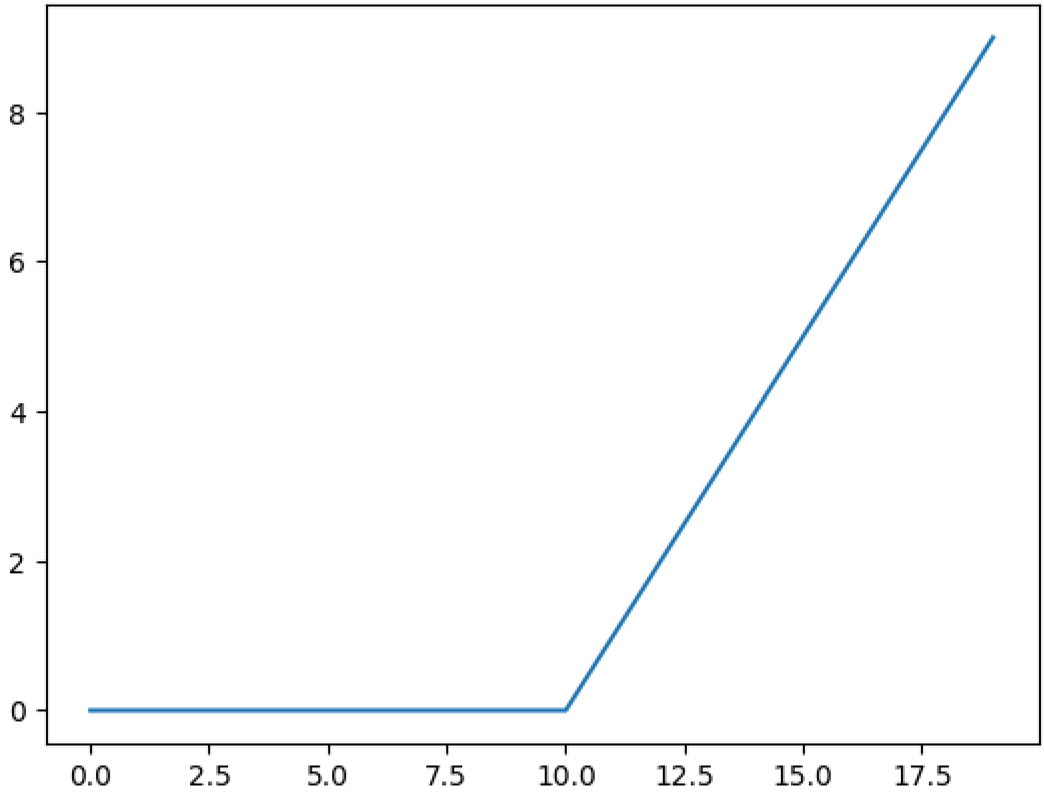
```
[<matplotlib.lines.Line2D at 0x7b8185718f40>]
```




```
#relu
def relu(x):
  return tf.maximum(0,x)
```

```
plt.plot(relu(A))
```

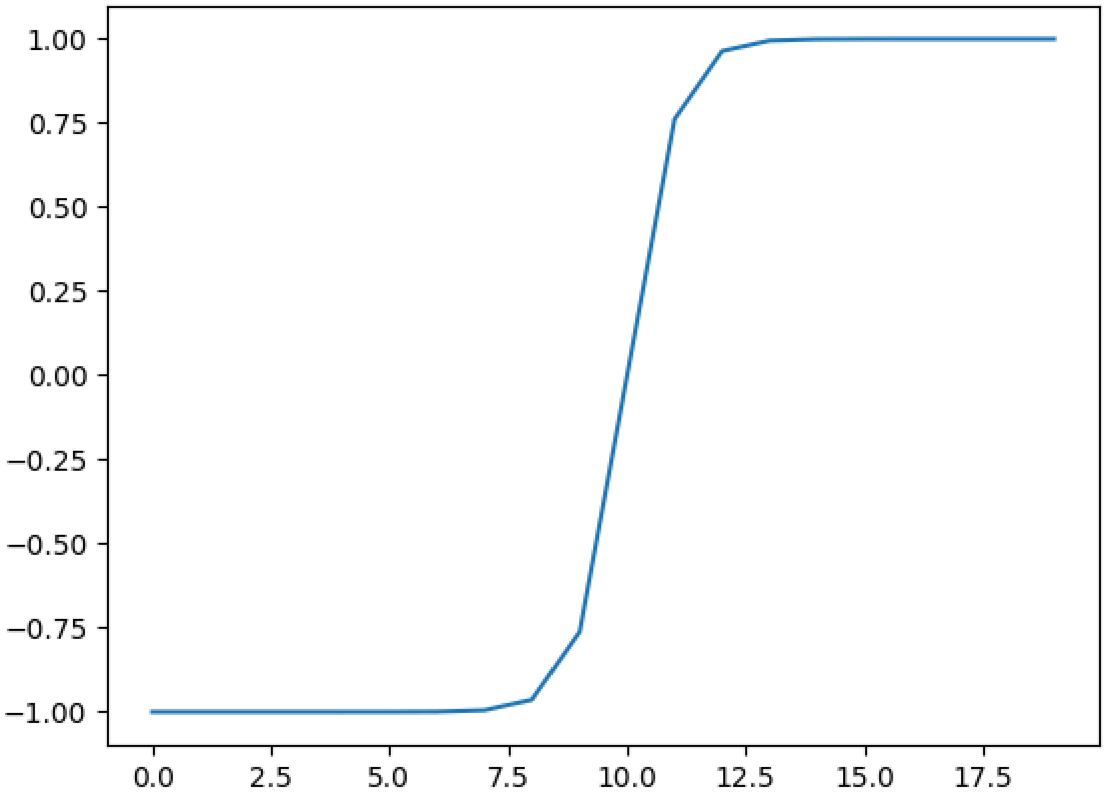
```
[<matplotlib.lines.Line2D at 0x7b8184267490>]
```




```
#tanh
def tanh(x):
  return (tf.exp(x)-tf.exp(-x))/(tf.exp(x)+tf.exp(-x))
```

```
plt.plot(tanh(A))
```

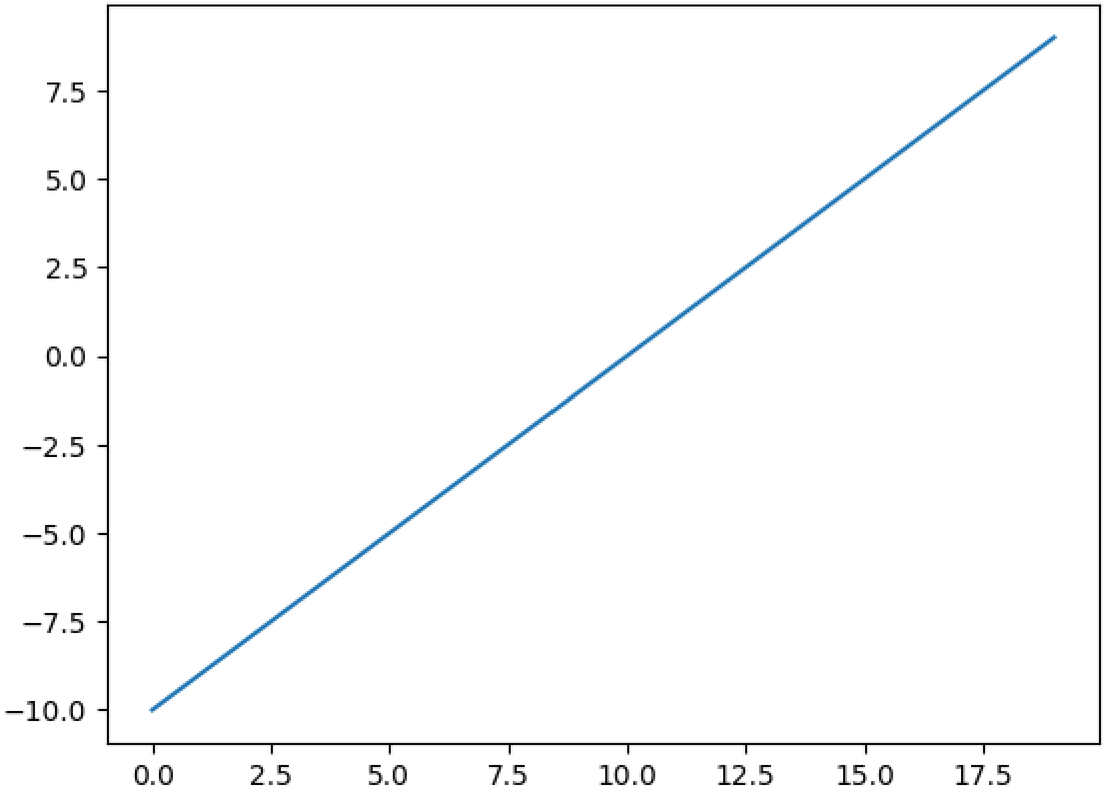
```
[<matplotlib.lines.Line2D at 0x7b818c2ff310>]
```



linear activation function returns the tensor unmodified

```
plt.plot(tf.keras.activations.linear(A))
```

```
[<matplotlib.lines.Line2D at 0x7b8185848be0>]
```



IMPROVING OUR MODEL

```
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)
```

```
tf.random.set_seed(42)
model_7=tf.keras.Sequential([
    tf.keras.layers.Dense(4,activation='relu'),
    tf.keras.layers.Dense(4,activation='relu'),
    tf.keras.layers.Dense(1,activation="sigmoid"),
])
model_7.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
                metrics=["accuracy"])
```

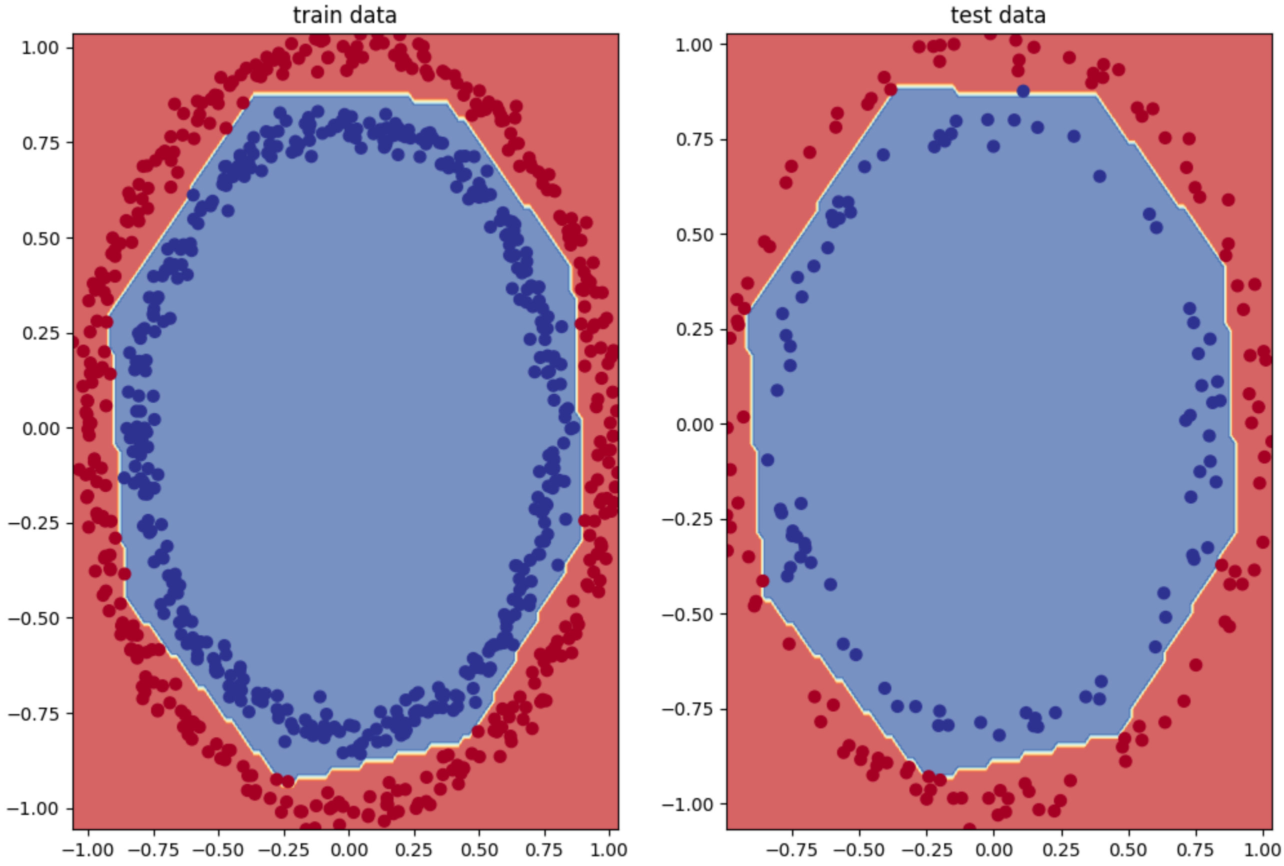
```
history=model_7.fit(X_train,Y_train,epochs=100,verbose=0)
```

```
model_7.evaluate(X_test,Y_test)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9877 - loss: 0.0257
[0.028310153633356094, 0.9850000143051147]
```

```
plt.figure(figsize=(12,8))
plt.subplot(1,2,1)
plt.title("train data")
plot_decision_boundary(model_7,X_train,Y_train)
plt.subplot(1,2,2)
plt.title("test data")
plot_decision_boundary(model_7,X_test,Y_test)
```

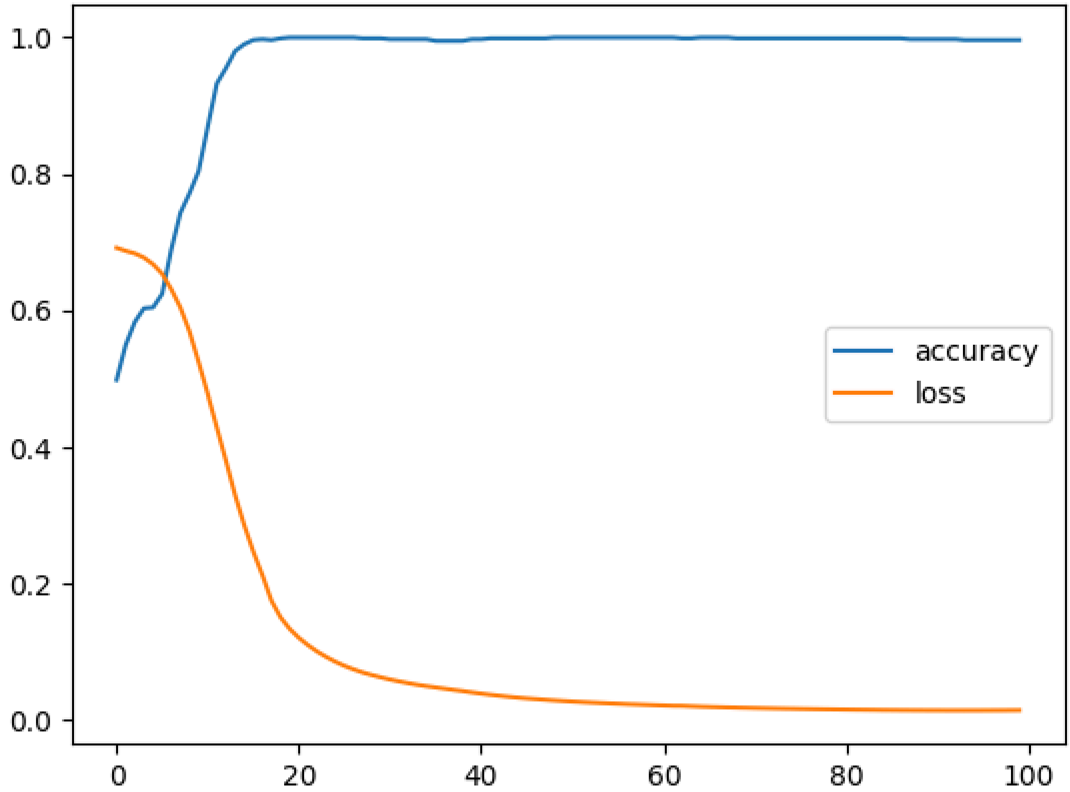
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
doing binary classification
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
doing binary classification
```



plot loss or training curves

```
pd.DataFrame(history.history).plot()
```

<Axes: >



Finding the best learning rate

use:

- a learning rate callback
- modified loss curve plot

```
tf.random.set_seed(42)
model_8=tf.keras.Sequential([
    tf.keras.layers.Dense(4,activation='relu'),
    tf.keras.layers.Dense(4,activation='relu'),
    tf.keras.layers.Dense(1,activation="sigmoid"),
])
model_8.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
                metrics=["accuracy"])


#create learning rate callback
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch:1e-3*10**(epoch/20))


history_8=model_8.fit(X_train,Y_train,epochs=100,callbacks=[lr_scheduler])
```

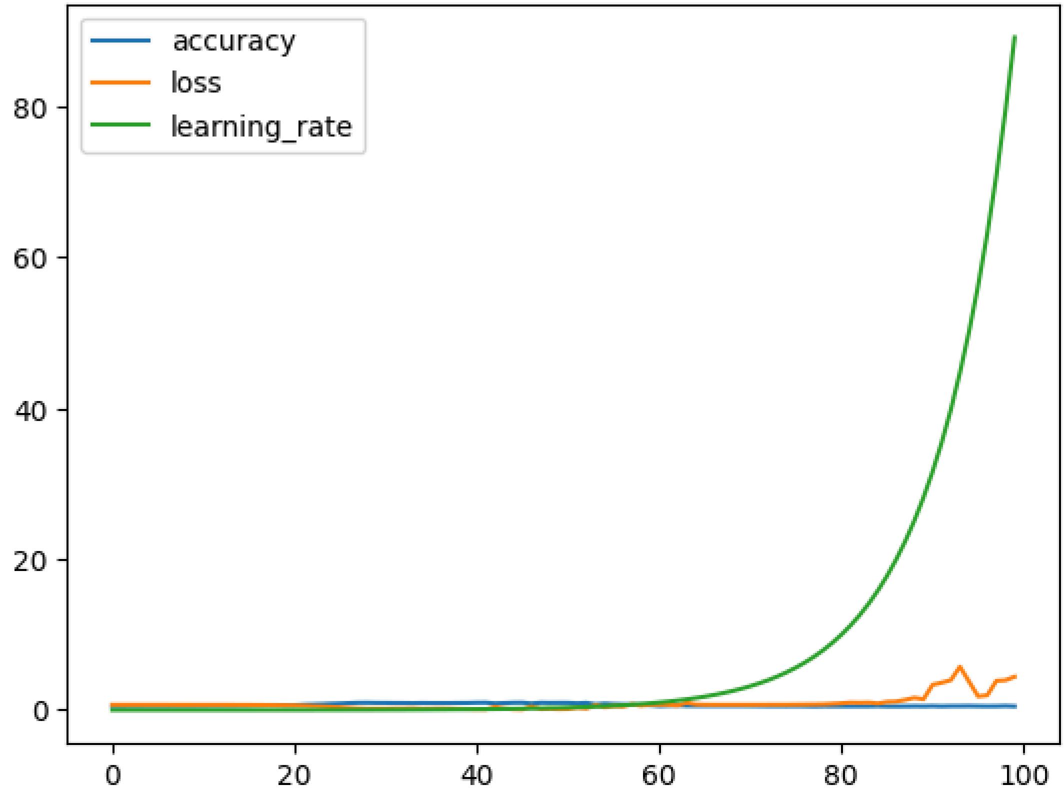
```
Epoch 75/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5129 - loss: 0.7002 - learnin
Epoch 76/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5156 - loss: 0.7090 - learnin
Epoch 77/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5120 - loss: 0.7087 - learnin
Epoch 78/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5219 - loss: 0.7270 - learnin
Epoch 79/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5054 - loss: 0.7536 - learnin
Epoch 80/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5004 - loss: 0.7745 - learnin
Epoch 81/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4986 - loss: 0.7869 - learnin
Epoch 82/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4945 - loss: 0.8646 - learnin
Epoch 83/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5233 - loss: 0.8815 - learnin
Epoch 84/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4981 - loss: 1.2398 - learnin
Epoch 85/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4916 - loss: 0.8288 - learnin
Epoch 86/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4846 - loss: 0.9371 - learnin
Epoch 87/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4915 - loss: 0.9992 - learnin
Epoch 88/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4932 - loss: 1.4662 - learnin
Epoch 89/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5066 - loss: 1.1138 - learnin
Epoch 90/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5027 - loss: 1.2253 - learnin
Epoch 91/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5163 - loss: 2.2288 - learnin
Epoch 92/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4988 - loss: 2.0046 - learnin
Epoch 93/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4986 - loss: 2.1678 - learnin
Epoch 94/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4926 - loss: 2.4326 - learnin
Epoch 95/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5040 - loss: 7.2756 - learnin
Epoch 96/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5093 - loss: 10.8878 - learni
Epoch 97/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5315 - loss: 10.6321 - learni
Epoch 98/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5020 - loss: 11.9537 - learni
Epoch 99/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4984 - loss: 3.3243 - learnin
Epoch 100/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4994 - loss: 3.4776 - learnin
```

```
model_8.evaluate(X_test,Y_test)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4112 - loss: 5.0090
[4.934600830078125, 0.41999998688697815]
```

```
pd.DataFrame(history_9.history).plot()
```

```
<Axes: >
```




```
lrs = 1e-4 * (10 ** (tf.range(len(history_9.history['loss']) / 20)))
```

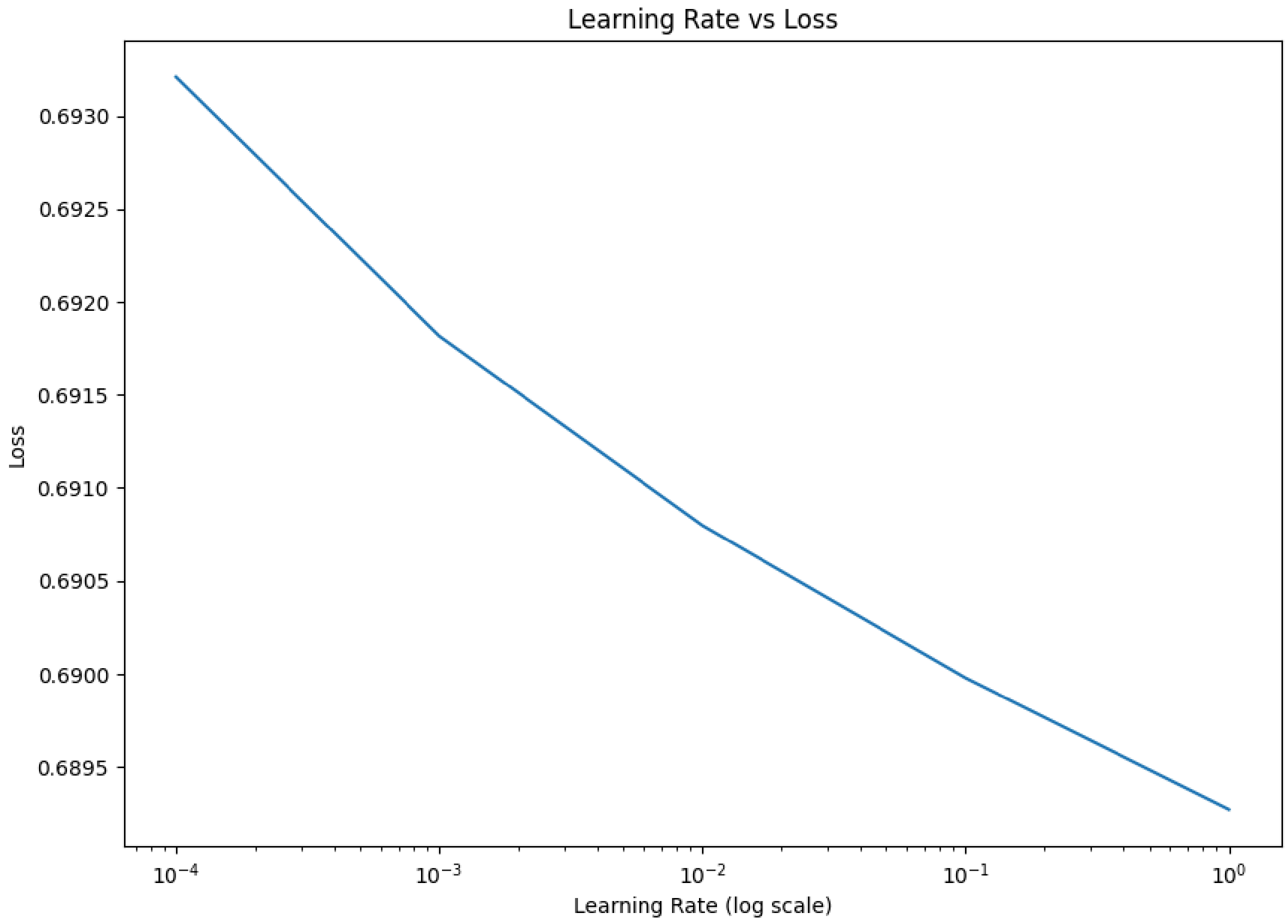
```
print("Shape of lrs:", lrs.shape)
print("Shape of loss:", len(history_9.history['loss']))
```

```
Shape of lrs: (5,)
Shape of loss: 100
```

Generate a slider using jupyter widgets Close

```
min_length = min(len(lrs), len(history_9.history['loss']))
lrs = lrs[:min_length]
loss = history_9.history['loss'][:min_length]
```

```
plt.figure(figsize=(10, 7))
plt.semilogx(lrs, loss)
plt.xlabel("Learning Rate (log scale)")
plt.ylabel("Loss")
plt.title("Learning Rate vs Loss")
plt.show()
```



- accuracy= tp+tn/tp+tn+fp+fn
- precision=tp/tp+fp (high value leads to less fp)
- recall=tp/tp+fn (high val leads to less fn)
- f1=2* precision * recall/(precision+recall)
- confusion matrix
- classification report

there is a precision recall tradeoff

```
y_pred = model_7.predict(X_test)
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

```
y_pred
```

```
       [2.14534298e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [4.61316120e-07],
       [1.15586829e-03],
       [9.96138871e-01],
       [6.82716234e-07],
       [9.51105008e-08],
       [8.12747771e-08],
       [6.33873977e-03],
       [2.73157893e-05],
       [3.55156881e-05],
       [2.41709479e-08],
       [6.77610660e-05],
       [9.96138871e-01],
       [1.07641448e-03],
       [9.96138871e-01],
       [2.25141389e-06],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.92603779e-01],
       [9.96138871e-01],
       [2.52594873e-06],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [9.60078796e-06],
       [1.28494081e-04],
       [4.35949869e-06],
       [2.25501806e-01],
       [9.96138871e-01],
       [9.96138871e-01],
       [3.55159609e-05],
       [9.96138871e-01],
       [9.96138871e-01],
       [1.01090613e-04],
       [9.96138871e-01],
       [2.74408851e-08],
       [9.86508727e-01],
       [3.17427702e-02],
       [2.02816352e-03],
       [9.96138871e-01],
       [1.36119430e-04],
       [9.96138871e-01],
       [9.96138871e-01]], dtype=float32)
```

```
Y_test
```

```
array([1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
```

```
0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0,
1, 1])
```

our predictions has come out in **prediction probability** from .... the std output from sigmoid (or softmax)activation function

```
#convert prdiction probaability to binary from
tf.round(y_pred)
```

```
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.]], dtype=float32)>
```

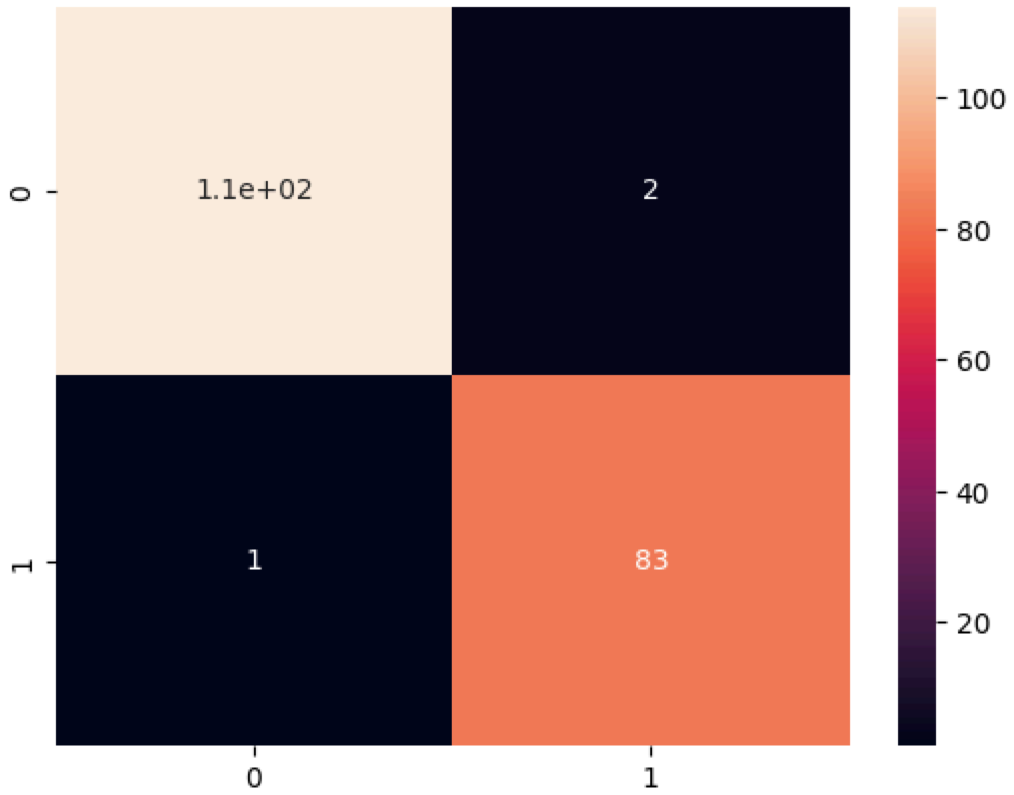
```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, prec
```

```
confusion_matrix(Y_test,tf.round(y_pred))
```

```
array([[114,   2],
       [  1,  83]])
```

```
import seaborn as sns
sns.heatmap(confusion_matrix(Y_test,tf.round(y_pred)),annot=True)
```

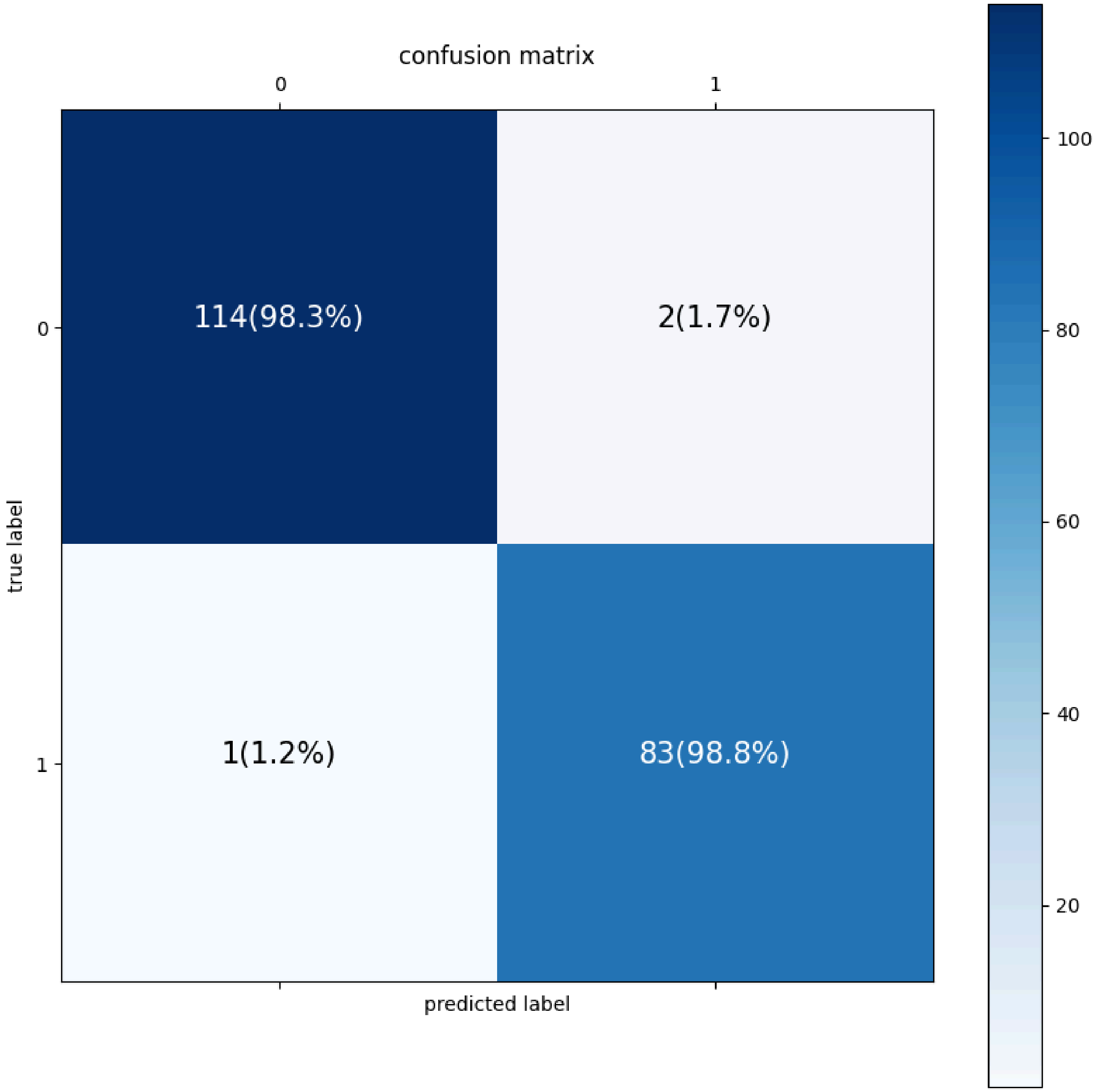
```
<Axes: >
```



Suggested code may be subject to a licence | 1mampurwad1/mnist_cnn | CrispenGari/keras-api

```
import itertools
figsize=(10,10)
cm = confusion_matrix(Y_test,tf.round(y_pred))
cm_norm= cm.astype("float")/cm.sum(axis=1)[:,np.newaxis]
n_classes=cm.shape[0]
fig,ax=plt.subplots(figsize=figsize)
cax=ax.matshow(cm,cmap=plt.cm.Blues)
fig.colorbar(cax)
classes= False
```

```
    if classes:
      labels=classes
    else:
      labels=np.arange(cm.shape[0])
    ax.set(title="confusion matrix",
          xlabel="predicted label",
          ylabel="true label",
          xticks=np.arange(n_classes),
          yticks=np.arange(n_classes),
          xticklabels=labels,
          yticklabels=labels)
    threshold =(cm.max()+cm.min())/2
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
      plt.text(j,i,f"{cm[i,j]}({cm_norm[i,j]*100:.1f}%)",
      horizontalalignment="center",
      color="white" if cm[i,j]>threshold else "black",
      size=15)
```



MULTICLASS CLASSIFICATION

- more than 2 different classes
- classify cloths is what we are gonna do

```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
```

```
(train_data,train_labels),(test_data,test_lables)=fashion_mnist.load_data()
```

```
train_data[0]
```

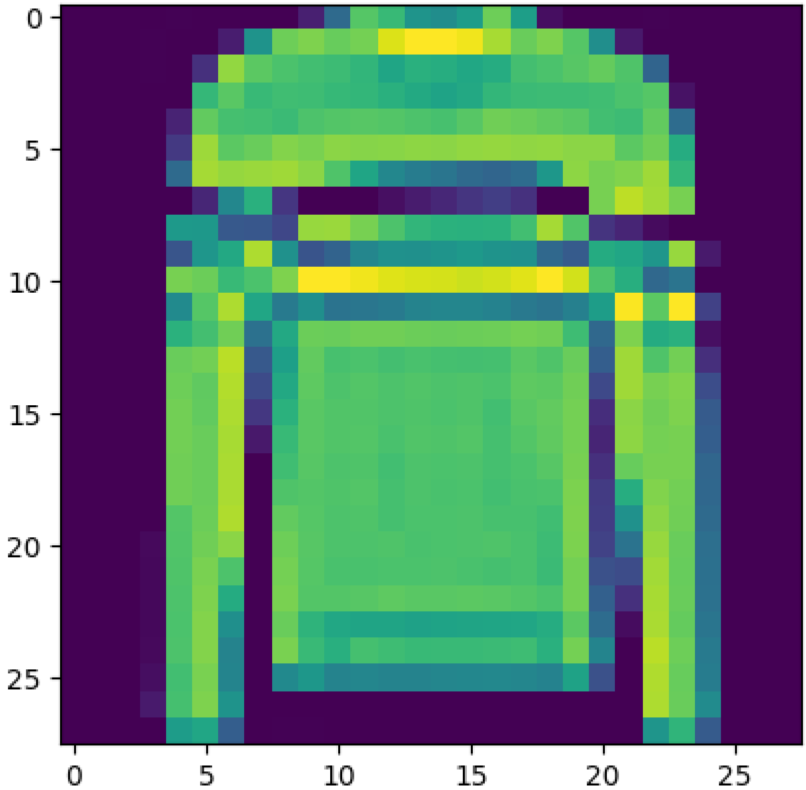
```
ndarray (28, 28) show data
```

```
train_data[0].shape,train_labels[0].shape
```

```
((28, 28), ())
```

```
import matplotlib.pyplot as plt
plt.imshow(train_data[5])
```

```
<matplotlib.image.AxesImage at 0x7b817ca70d00>
```

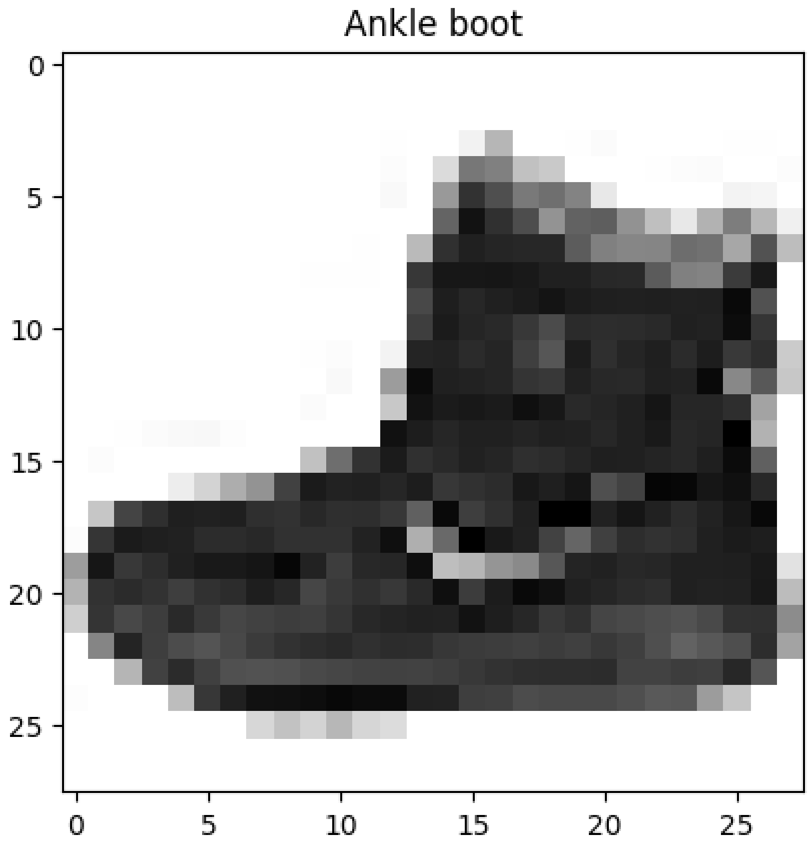


create a list so we can index onto our training labels so they are human readable

```
class_names=['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker'
```

```
plt.imshow(train_data[0],cmap=plt.cm.binary)
plt.title(class_names[train_labels[0]])
```

```
Text(0.5, 1.0, 'Ankle boot')
```




```
plt.figure(figsize=(7,7))
for i in range(4):
  ax=plt.subplot(2,2,i+1)

  plt.imshow(train_data[i],cmap=plt.cm.binary)
  plt.title(class_names[train_labels[i]])
  plt.axis("off")
```

multiclass classification model

- input shape=28x28
- o/p shape=10
- loss func= categoricalcrossentropy
- o/p layer activation=softmax

```
tf.random.set_seed(42)
model_10=tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(10,activation="softmax")
])
model_10.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: Use
  super().__init__(**kwargs)
```

- CategoricalCrossentropy: one hot encoded values are expected
- otherwise use sparseCategoricalCrossEntropy(int type)

```
non_norm_history=model_10.fit(train_data,train_labels,epochs=10,validation_data=(test_dat
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.1301 - loss: 3.1902 - val_ac
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1995 - loss: 1.9566 - val_ac
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.3079 - loss: 1.6457 - val_ac
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.3898 - loss: 1.3959 - val_ac
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.4501 - loss: 1.2570 - val_ac
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.4741 - loss: 1.1832 - val_ac
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.4740 - loss: 1.1565 - val_ac
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.4869 - loss: 1.1422 - val_ac
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.4910 - loss: 1.1313 - val_ac
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.4954 - loss: 1.1243 - val_ac
```

Generate | create a dataframe with 2 columns and 10 rows | Close

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaled_train_data=scaler.fit_transform(train_data.reshape(-1,28*28))
scaled_test_data=scaler.transform(test_data.reshape(-1,28*28))
```

```
tf.random.set_seed(42)
model_11=tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(10,activation="softmax")
])
model_11.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

```
/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`
```

```
train_data_norm=train_data/255.0
test_data_norm=test_data/255.0
```

Generate 10 random numbers using numpy Close

```
tf.random.set_seed(42)
model_12=tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(10,activation="softmax")
])
model_12.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
```

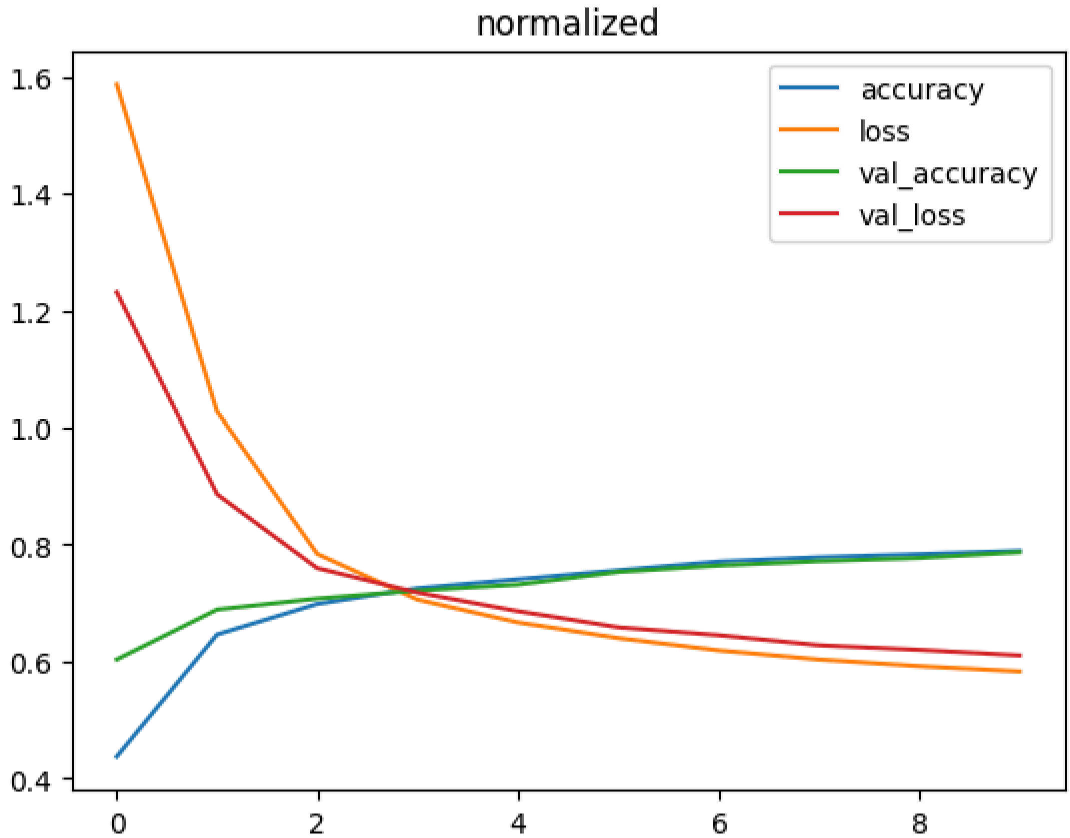
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: Use
  super().__init__(**kwargs)
```

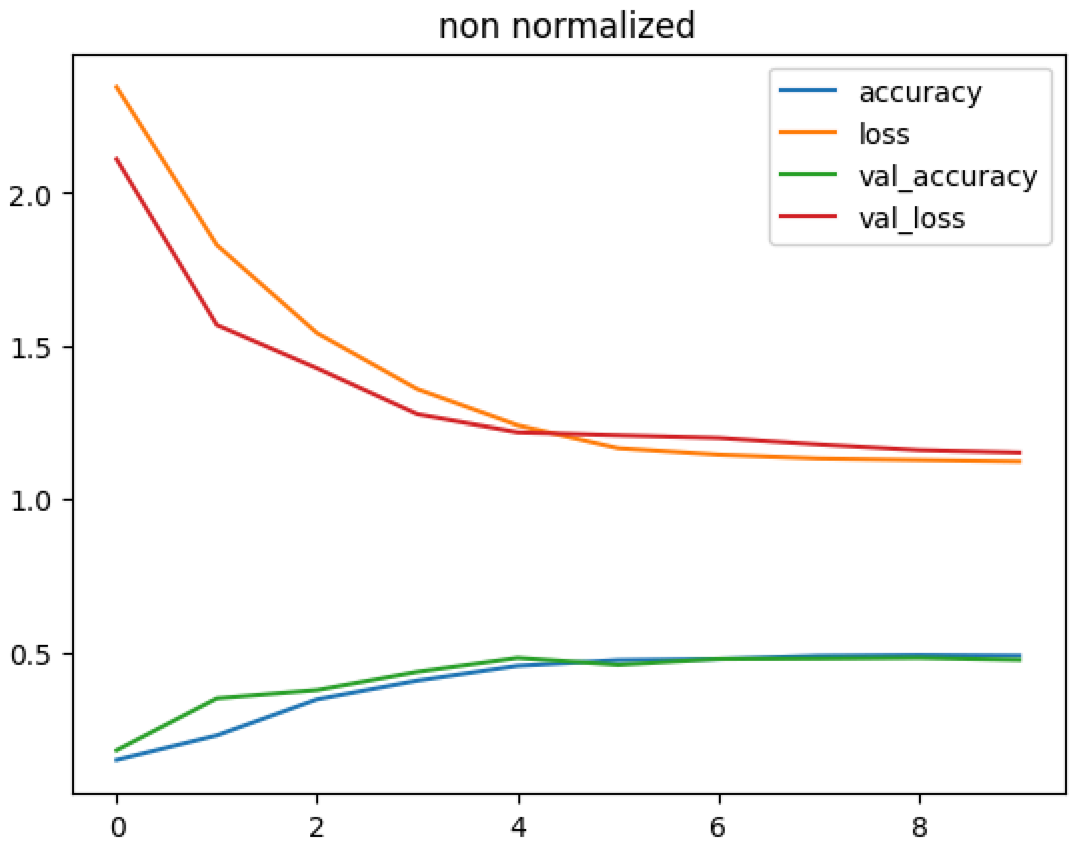
```
norm_history=model_12.fit(train_data_norm,train_labels,epochs=10,validation_data=(test_da
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.3250 - loss: 1.8457 - val_a
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.6243 - loss: 1.1099 - val_ac
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.6973 - loss: 0.8191 - val_ac
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7233 - loss: 0.7226 - val_ac
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.7391 - loss: 0.6798 - val_ac
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7525 - loss: 0.6525 - val_ac
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7696 - loss: 0.6299 - val_ac
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.7781 - loss: 0.6139 - val_ac
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.7835 - loss: 0.6024 - val_ac
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7879 - loss: 0.5938 - val_ac
```

```
import pandas as pd
pd.DataFrame(norm_history.history).plot(title="normalized")
pd.DataFrame(non_norm_history.history).plot(title="non normalized")
```

```
<Axes: title={'center': 'non normalized'}>
```





compare with same criteria since small change has dramatic effects

```
#finding ideal learning rate
tf.random.set_seed(42)
model_13=tf.keras.Sequential([
```

```
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(10,activation="softmax")
])
model_13.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
lr_scheduler=tf.keras.callbacks.LearningRateScheduler(lambda epoch:1e-3*10**(epoch/20))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: Use
  super().__init__(**kwargs)
```

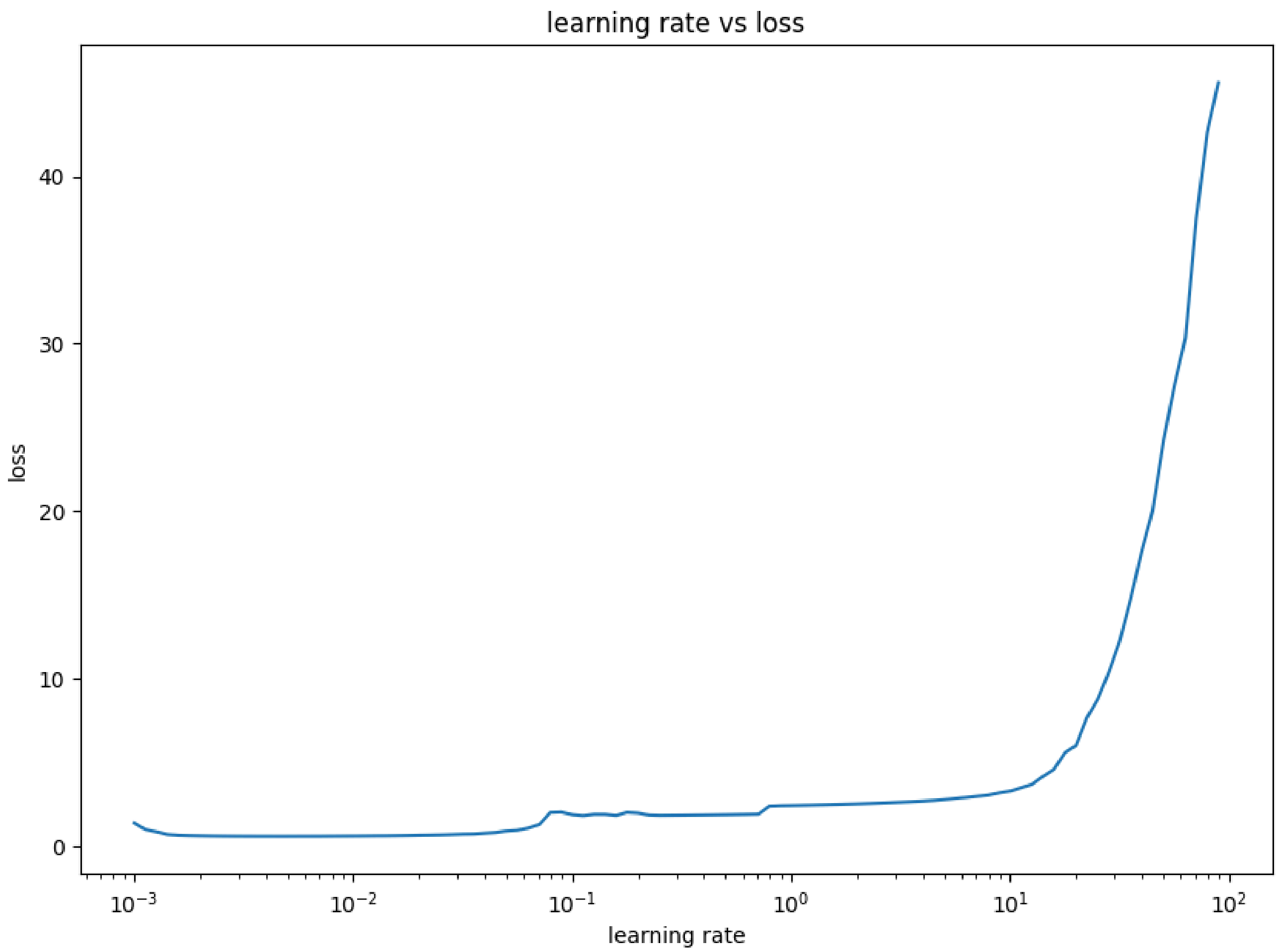
```
model_13.fit(train_data_norm,train_labels,epochs=100,callbacks=[lr_scheduler])
```

```
Epoch 94/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0999 - loss: 19.8436 - le
Epoch 95/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0982 - loss: 24.5708 - le
Epoch 96/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - accuracy: 0.0999 - loss: 27.7266 - le
Epoch 97/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1009 - loss: 31.2183 - le
Epoch 98/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0993 - loss: 36.5593 - le
Epoch 99/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.1016 - loss: 42.5347 - le
Epoch 100/100
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.1011 - loss: 44.4787 - le
<keras.src.callbacks.history.History at 0x7b8177b7af20>
```

Generate | a slider using jupyter widgets | Close

```
import numpy as np
import matplotlib.pyplot as plt
lrs=1e-3*(10**(np.arange(100)/20))
plt.figure(figsize=(10,7))
plt.semilogx(lrs,model_13.history.history["loss"])
plt.xlabel("learning rate")
plt.ylabel("loss")
plt.title("learning rate vs loss")
```

```
Text(0.5, 1.0, 'learning rate vs loss')
```



ideal learning rate is 0.001

```
tf.random.set_seed(42)
model_14=tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(4,activation="relu"),
    tf.keras.layers.Dense(10,activation="softmax")
])
model_14.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=["accuracy"])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: Use
  super().__init__(**kwargs)
```

```
history_14 = model_14.fit(train_data_norm,train_labels,epochs=10,validation_data=(test_da
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.4281 - loss: 1.5809 - val_ac
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7442 - loss: 0.7574 - val_ac
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.7778 - loss: 0.6540 - val_ac
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7902 - loss: 0.6050 - val_ac
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8004 - loss: 0.5767 - val_ac
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8066 - loss: 0.5579 - val_ac
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8115 - loss: 0.5444 - val_ac
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8147 - loss: 0.5338 - val_ac
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8168 - loss: 0.5263 - val_ac
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8192 - loss: 0.5202 - val_ac
```

EVALUATING OUR MODEL WITH OTHER METRICS

- classification metrics
- asses some of its predictions
- improve its results(change architechture)
- save and export

```
import itertools
from sklearn.metrics import confusion_matrix
```

```
def make_conf_matrix(y_true,y_pred,classes= None,figsize=(10,10)):
  cm = confusion_matrix(y_true,y_pred)
  cm_norm= cm.astype("float")/cm.sum(axis=1)[:,np.newaxis]
  n_classes=cm.shape[0]
  fig,ax=plt.subplots(figsize=figsize)
  cax=ax.matshow(cm,cmap=plt.cm.Blues)
  fig.colorbar(cax)

  if classes:
    labels=classes
  else:
    labels=np.arange(cm.shape[0])
  ax.set(title="confusion matrix",
        xlabel="predicted label",
        ylabel="true label",
        xticks=np.arange(n_classes),
        yticks=np.arange(n_classes),
        xticklabels=labels,
        yticklabels=labels)
  threshold =(cm.max()+cm.min())/2
  for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
    plt.text(j,i,f"{cm[i,j]}({cm_norm[i,j]*100:.1f}%)",
    horizontalalignment="center",
    color="white" if cm[i,j]>threshold else "black",
    size=15)
```

```
y_probs=model_14.predict(test_data_norm)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

NOTE: remember to make predictions on the same kind of data your model was trained om (eg if trained on normalized data use normalized data for predictions)

```
y_probs[0],tf.argmax(y_probs[0])
```

```
(array([1.5085043e-03, 1.1936341e-04, 1.0228357e-03, 8.6857857e-05,
        1.2778917e-05, 3.5110056e-01, 2.8528241e-04, 6.0521282e-02,
        3.2882905e-05, 5.8530962e-01], dtype=float32),
 <tf.Tensor: shape=(), dtype=int64, numpy=9>)
```

```
k=tf.argmax(y_probs[0]).numpy()
```

Generate | a slider using jupyter widgets | Close

```
class_names[k]
```

```
'Ankle boot'
```

```
#convert all prediction pobabilities to  integers
y_preds=y_probs.argmax(axis=1)
y_preds
```

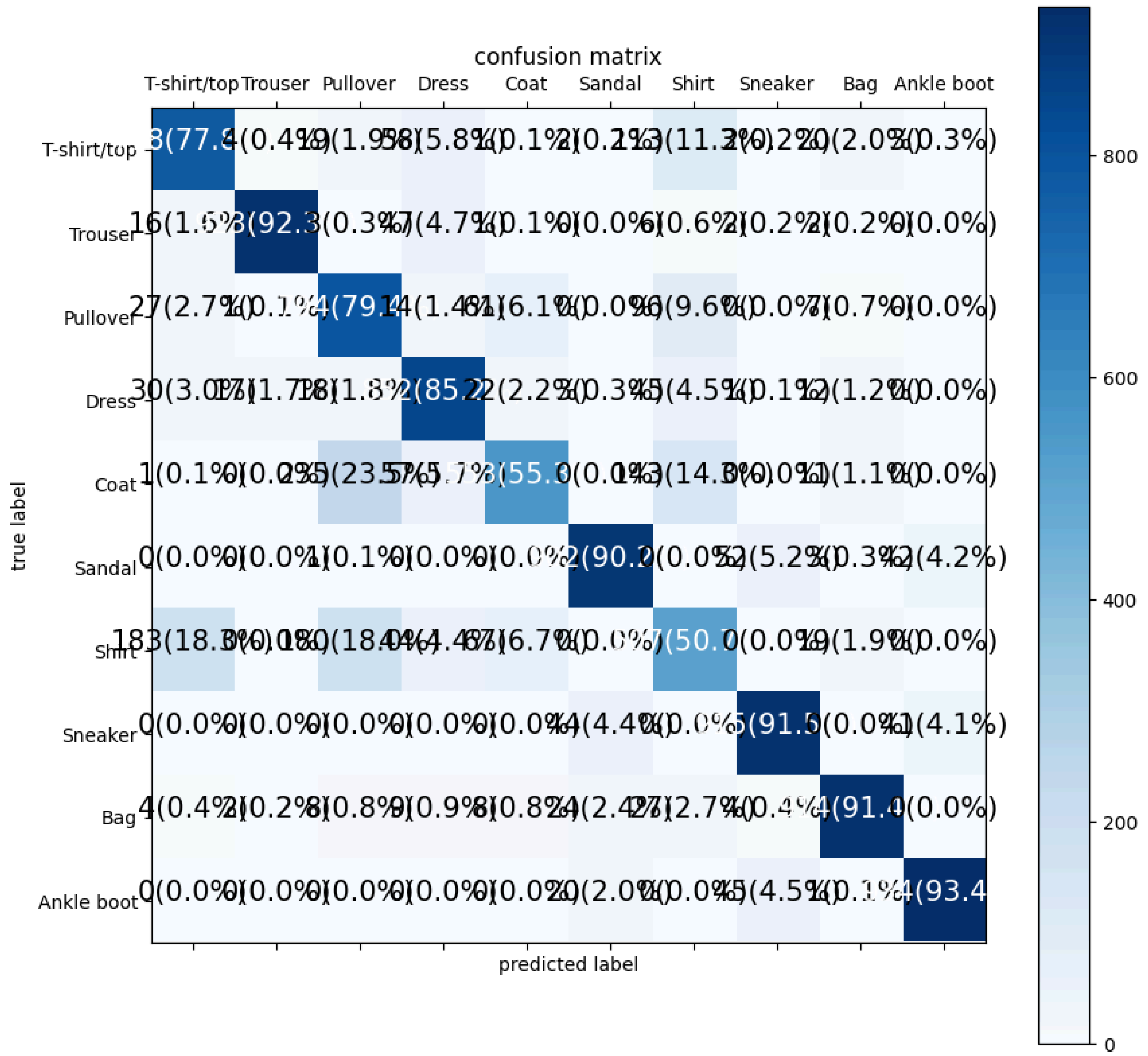
```
array([9, 2, 1, ..., 4, 1, 5])
```

Generate a slider using jupyter widgets Close

```
from sklearn.metrics import confusion_matrix
confusion_matrix(test_lables,y_preds)
```

```
array([[778,   4,  19,  58,   1,   2, 113,   2,  20,   3],
       [ 16, 923,   3,  47,   1,   0,   6,   2,   2,   0],
       [ 27,   1, 794,  14,  61,   0,  96,   0,   7,   0],
       [ 30,  17,  18, 852,  22,   3,  45,   1,  12,   0],
       [  1,   0, 235,  57, 553,   0, 143,   0,  11,   0],
       [  0,   0,   1,   0,   0, 902,   0,  52,   3,  42],
       [183,   0, 180,  44,  67,   0, 507,   0,  19,   0],
       [  0,   0,   0,   0,   0,  44,   0, 915,   0,  41],
       [  4,   2,   8,   9,   8,  24,  27,   4, 914,   0],
       [  0,   0,   0,   0,   0,  20,   0,  45,   1, 934]])
```

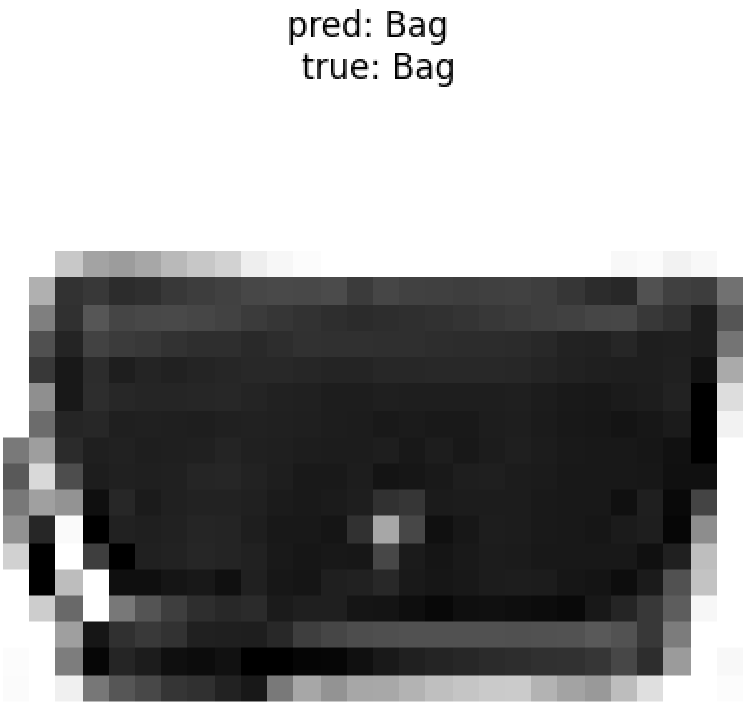
```
make_conf_matrix(test_lables,y_preds,classes=class_names)
```




```
import random
def plot_random_image(model,images,true_labels,classes):
  """
  Picks a random image, plots it and labels it with a prediction and truth label
  """
  i=random.randint(0,len(images))
  target_image=images[i]
  pred_probs=model.predict(target_image.reshape(1,28,28))
  pred_label=class_names[pred_probs.argmax()]
  true_label=class_names[true_labels[i]]
  plt.imshow(target_image,cmap=plt.cm.binary)
  if pred_label == true_label:
    color="green"
  else:
    color="red"
  plt.xlabel(f"pred: {pred_label} \n true: {true_label}",color=color)
  plt.title(f"pred: {pred_label} \n true: {true_label}")
  plt.axis("off")
```

```
plot_random_image(model_14,test_data_norm,test_lables,class_names)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
```




```
model_14.layers[1]
```

```
<Dense name=dense_53, built=True>
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.