Computational Intelligence Lab

Assignment 2

Karolina Kotlowska, lab czw. 9:30

2.1 Load Fashion MNIST dataset

The following code loads Fashion MNIST dataset. More information about the dataset

We will concatnetate training and test sets to make an own split.

In [130…
```
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X=np.concatenate((np.array(x_train),np.array(x_test)),axis=0);
y=np.concatenate((np.array(y_train),np.array(y_test)),axis=0);
# X=X.reshape(X.shape[0],X.shape[1]*X.shape[2])
X=X/255
```

Display a few dozen of images

In [131…
```
import matplotlib.pyplot as plt
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
col2 = 15
row2 = 5
fig = plt.figure(figsize=(col2, row2))
for index2 in range(0, col2*row2):
    fig.add_subplot(row2, col2, index2 + 1)

    plt.axis('off')
    plt.imshow(X[index2]) # index of the sample image
plt.show()
```

TODO 2.1.1 Print number of classes and the shape (dimensions) of the data

In [132…

```
print(f' #classes = {y.max()+1},  shape = {X.shape}')
```

```
#classes = 10,  shape = (70000, 28, 28)
```

Subset selection

We select a subset comprising only two classes, namely t-shirts/tops and trousers

In [133…

```
X2 = X[(y==0) | (y==1)]
y2=y[(y==0) | (y==1)]
```

TODO 2.1.2 Print number of classes and the shape (dimensions) of the data

In [134…

```
print(f' #classes = {y2.max()+1}, shape = {X2.shape[0]} {X2.shape[1]} {X2
```

```
#classes = 2, shape = 14000 28 28
```

TODO 2.1.3 Display images from X2

In [135…

```
col2 = 5
row2 = 3
fig = plt.figure(figsize=(col2, row2))
for index2 in range(0, col2*row2):
    fig.add_subplot(row2, col2, index2 + 1)

    plt.axis('off')
    plt.imshow(X2[index2])
plt.show()
```

Flatten images

In [136…
```
X2=X2.reshape(X2.shape[0],-1)
```

TODO 2.1.4 Print the shape of X2 after flattening

In [137…
```
print(f' #classes = {y2.max()+1}, shape = {X2.shape}')
```

```
 #classes = 2, shape = (14000, 784)
```

Train / test split

In [138…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.3
```

2.2 Binary classification

- Build a model
- Compile (use binary_crossentropy as the loss function)
- Fit the training data, set epochs=10
- Display training history

In [139…
```
import tensorflow as tf
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(X_train.shape[
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_51 (Dense)            (None, 16)                12560

 dense_52 (Dense)            (None, 8)                 136

 dense_53 (Dense)            (None, 1)                 9

=================================================================
Total params: 12,705
Trainable params: 12,705
Non-trainable params: 0
_________________________________________________________________
```

In [140…

```
model.compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['ac
```

In [141…

```
hist = model.fit(X_train,y_train,epochs=10,batch_size=128)
```

```
Epoch 1/10
74/74 [==============================] - 0s 740us/step - loss: 0.1317 -
accuracy: 0.9610
Epoch 2/10
74/74 [==============================] - 0s 618us/step - loss: 0.0546 -
accuracy: 0.9808
Epoch 3/10
74/74 [==============================] - 0s 599us/step - loss: 0.0459 -
accuracy: 0.9837
Epoch 4/10
74/74 [==============================] - 0s 729us/step - loss: 0.0410 -
accuracy: 0.9851
Epoch 5/10
74/74 [==============================] - 0s 602us/step - loss: 0.0369 -
accuracy: 0.9866
Epoch 6/10
74/74 [==============================] - 0s 593us/step - loss: 0.0348 -
accuracy: 0.9882
Epoch 7/10
74/74 [==============================] - 0s 606us/step - loss: 0.0339 -
accuracy: 0.9886
Epoch 8/10
74/74 [==============================] - 0s 576us/step - loss: 0.0308 -
accuracy: 0.9901
Epoch 9/10
74/74 [==============================] - 0s 594us/step - loss: 0.0301 -
accuracy: 0.9896
Epoch 10/10
74/74 [==============================] - 0s 571us/step - loss: 0.0264 -
accuracy: 0.9904
```

In [142…

```
for k in hist.history:
  print(k)
```

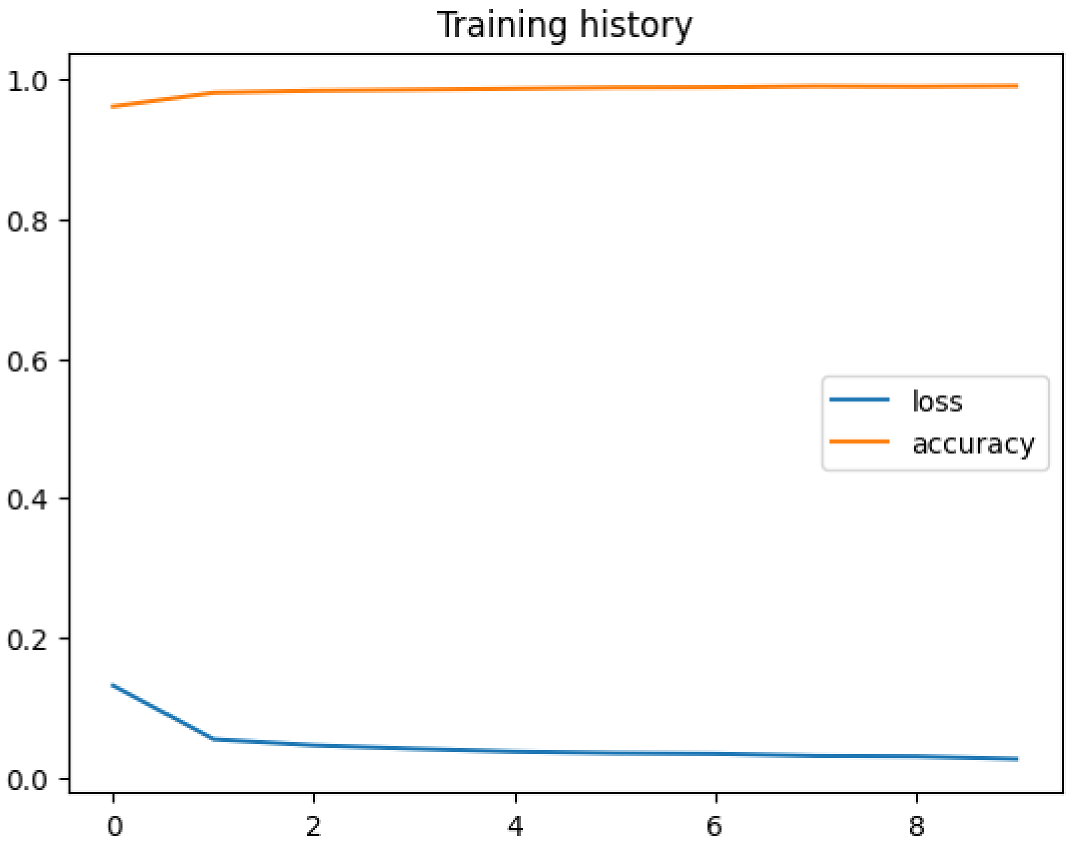
```
loss
accuracy
```

In [143…

```
plt.title('Training history')
for k in hist.history:
```

```
  plt.plot(hist.history[k],label=k)
plt.legend()
```

Out[143]: <matplotlib.legend.Legend at 0x2a726f580>



What is the performance of our classifier?

Testing on the training set

Collect the model predictions

In [144…
```
probs = model.predict(X_train)
print(probs.shape)
probs
```

```
294/294 [==============================] - 0s 347us/step
(9380, 1)
```

Out[144]:
```
array([[9.9991643e-01],
       [9.4701043e-08],
       [1.0660612e-04],
       ...,
       [9.1717315e-01],
       [2.1259647e-04],
       [2.9716314e-06]], dtype=float32)
```

We prefer a flat vector

In [145…
```
# probs = model.predict(X_train)[:,0]
# probs
# or
```

```
probs=probs.flatten()
probs
```

Out[145]:
```
array([9.9991643e-01, 9.4701043e-08, 1.0660612e-04, ..., 9.1717315e-01,
       2.1259647e-04, 2.9716314e-06], dtype=float32)
```

Compute labels

In [146…
```
y_pred = np.where(probs>.5,1,0)
y_pred
```

Out[146]:
```
array([1, 0, 0, ..., 1, 0, 0])
```

How to check results?

In [147…
```
pairs = zip(y_train,y_pred)
for i,p in enumerate(pairs):
  if i>30:break
  print(p)
```

```
(1, 1)
(0, 0)
(0, 0)
(0, 0)
(1, 1)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(1, 1)
(1, 1)
(1, 1)
(0, 0)
(1, 1)
(0, 0)
(0, 0)
(1, 1)
(0, 0)
(1, 1)
(0, 0)
(1, 1)
(1, 0)
(1, 1)
(0, 0)
(0, 0)
(1, 1)
(1, 1)
(1, 1)
```

Or load it to Pandas, then it can be converted to a fully browseable data table (use the magic wand on the right)

In [148…
```
import pandas as pd
df = pd.DataFrame(zip(y_train,y_pred))
df.head(df.size)
```

Out[148]:

	0	1
0	1	1
1	0	0
2	0	0
3	0	0
4	1	1
...	...	...
9375	1	1
9376	1	1
9377	1	1
9378	0	0
9379	0	0

9380 rows × 2 columns

However, we need a measure - a single number describing similarity of predictions and known labels

TODO 2.2.0 Compute this: $m = \frac{1}{n}\sum_{i=1}^{n} abs(y_{train}[i] - y_{pred}[i])$

In [149…
```
sum = 0
for i in range(len(y_train)):
    sum += abs(y_train[i] - y_pred[i])
m = (1/len(y_pred))*sum
print(f'm={m} 1-m={1-m}')
```

```
m=0.006929637526652515 1-m=0.9930703624733476
```

and compute accuracy. Compare results

In [150…
```
from sklearn.metrics import accuracy_score
acc=accuracy_score(y_pred,y_train)
print(f'accuracy={acc}')
```

```
accuracy=0.9930703624733476
```

Compute the **confusion matrix**

In [151…
```
from sklearn.metrics import confusion_matrix, classification_report
matrix = confusion_matrix(y_train, y_pred)
matrix
```

Out[151]:
```
array([[4667,   13],
       [  52, 4648]])
```

Useful function (small adaptation)

In [152…
```
import seaborn as sns
from sklearn import metrics

LABELS= ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
# Define the confusion matrix for the results
def show_confusion_matrix(validations, predictions, num_classes):
    matrix = metrics.confusion_matrix(validations, predictions)
    plt.figure(figsize=(num_classes, num_classes))
    hm = sns.heatmap(matrix,
                cmap='coolwarm',
                linecolor='white',
                linewidths=1,
                xticklabels=LABELS[0:num_classes],
                yticklabels=LABELS[0:num_classes],
                annot=True,
                fmt='d')
    plt.yticks(rotation = 0)  # Don't rotate (vertically) the y-axis labe
    # hm.set_ylim(0, len(matrix))
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

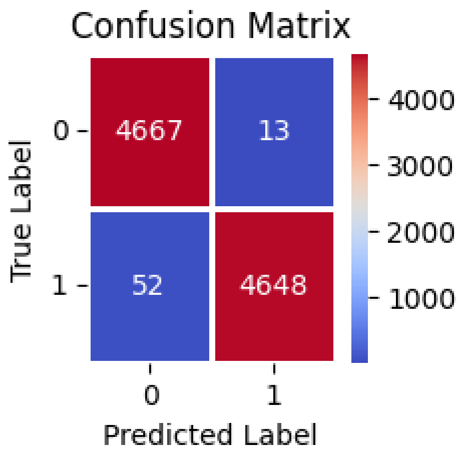
In [153…

```
y_pred = np.where(probs>.5,1,0)
y_pred
```

Out[153]: array([1, 0, 0, ..., 1, 0, 0])

In [154…

```
show_confusion_matrix(y_train, y_pred, 2)
```



In [155…

```
print(classification_report(y_train, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      4680
           1       1.00      0.99      0.99      4700

    accuracy                           0.99      9380
   macro avg       0.99      0.99      0.99      9380
weighted avg       0.99      0.99      0.99      9380
```

Exact formuls for classification scores are given here (in Polish) silde 14 and following

Validation on the test set

TODO 2.2.1 Repat the steps above on the test set. Replace X_train by X_test, etc.

In [156…

```
probs = model.predict(X_test)
probs=probs.flatten()
y_pred = np.where(probs>.5,1,0)

sum = 0
for i in range(len(y_test)):
    sum += abs(y_test[i] - y_pred[i])
m = (1/len(y_pred))*sum

print(f'm={m} 1-m={1-m}')
```

```
145/145 [==============================] - 0s 345us/step
m=0.01038961038961039 1-m=0.9896103896103896
```

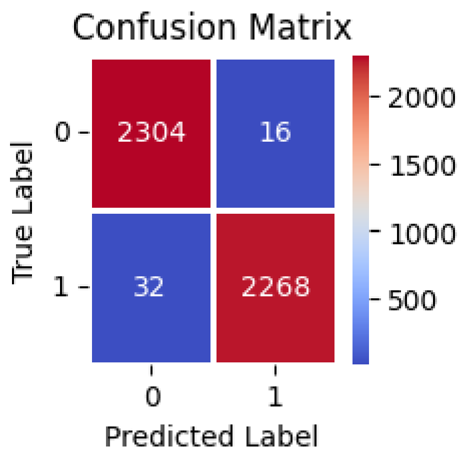
In [157…

```
acc=accuracy_score(y_pred, y_test)
print(f'accuracy={acc}')
```

```
accuracy=0.9896103896103896
```

In [158…

```
show_confusion_matrix(y_test, y_pred, 2)
```



In [159…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2320
           1       0.99      0.99      0.99      2300

    accuracy                           0.99      4620
   macro avg       0.99      0.99      0.99      4620
weighted avg       0.99      0.99      0.99      4620
```

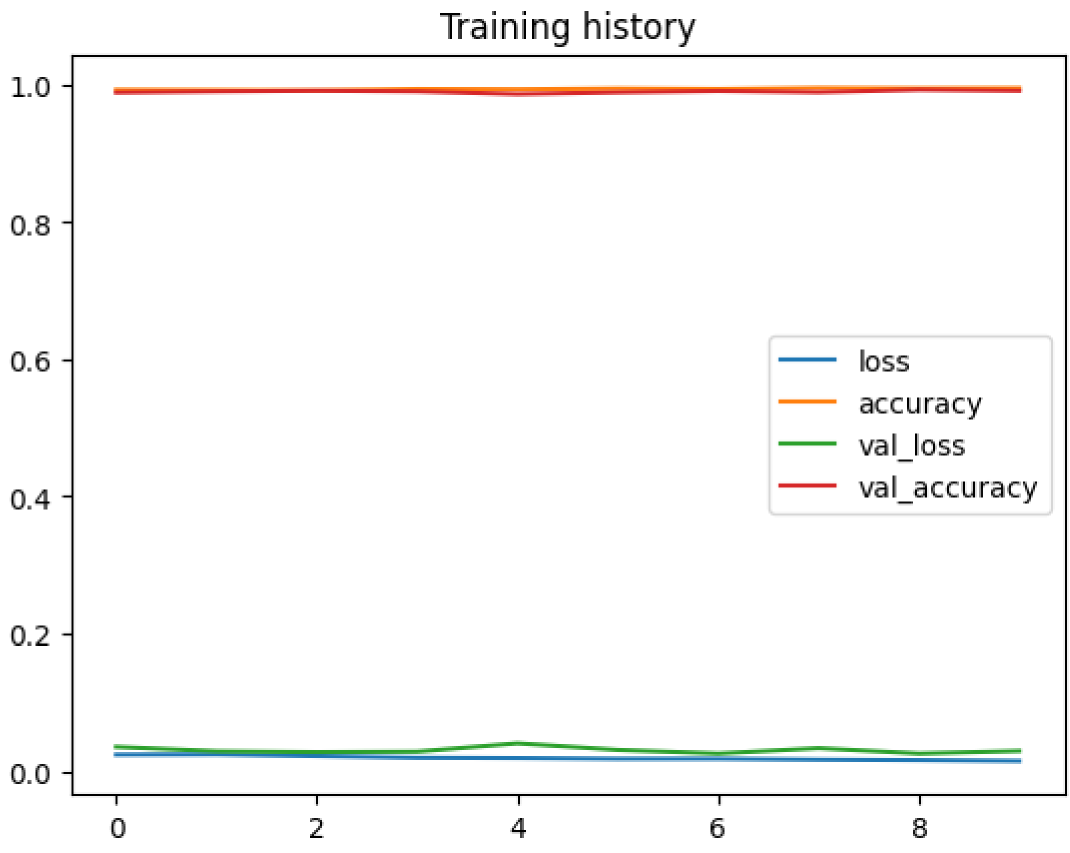
TODO 2.2.2 supply X_test, y_test as validation data, fit the model and display plots

In [160…

```
hist = model.fit(X_train,y_train,epochs=10,batch_size=128,validation_data
plt.title('Training history')
for k in hist.history:
  plt.plot(hist.history[k],label=k)
plt.legend()
```

```
Epoch 1/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0240 - ac
curacy: 0.9924 - val_loss: 0.0355 - val_accuracy: 0.9885
Epoch 2/10
74/74 [==============================] - 0s 1ms/step - loss: 0.0244 - ac
curacy: 0.9920 - val_loss: 0.0291 - val_accuracy: 0.9900
Epoch 3/10
74/74 [==============================] - 0s 959us/step - loss: 0.0221 -
accuracy: 0.9915 - val_loss: 0.0278 - val_accuracy: 0.9911
Epoch 4/10
74/74 [==============================] - 0s 951us/step - loss: 0.0197 -
accuracy: 0.9935 - val_loss: 0.0285 - val_accuracy: 0.9900
Epoch 5/10
74/74 [==============================] - 0s 954us/step - loss: 0.0191 -
accuracy: 0.9931 - val_loss: 0.0406 - val_accuracy: 0.9861
Epoch 6/10
74/74 [==============================] - 0s 924us/step - loss: 0.0180 -
accuracy: 0.9943 - val_loss: 0.0310 - val_accuracy: 0.9887
Epoch 7/10
74/74 [==============================] - 0s 932us/step - loss: 0.0180 -
accuracy: 0.9936 - val_loss: 0.0257 - val_accuracy: 0.9907
Epoch 8/10
74/74 [==============================] - 0s 928us/step - loss: 0.0169 -
accuracy: 0.9951 - val_loss: 0.0335 - val_accuracy: 0.9890
Epoch 9/10
74/74 [==============================] - 0s 930us/step - loss: 0.0161 -
accuracy: 0.9947 - val_loss: 0.0257 - val_accuracy: 0.9924
Epoch 10/10
74/74 [==============================] - 0s 940us/step - loss: 0.0149 -
accuracy: 0.9949 - val_loss: 0.0295 - val_accuracy: 0.9911
```

Out[160]: <matplotlib.legend.Legend at 0x2ad2e7d30>



2.3 Binary classification on brest cancer dataset

The information on the dataset can be found here

In [161…
```
from sklearn.datasets import load_breast_cancer

X,y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
```

TODO 2.3.1 Repeat the previous steps for to process the loaded data

- Create a model
- fit it setting epochs=100 and supplying validation data
- plot training history
- obtain predictions
- display confussion matrix
- and print the classification report

Caveat we seed everything possible to create reproducible results

In [162…
```
import os
import random
def set_seeds(seed=1):
    os.environ['PYTHONHASHSEED'] = str(seed)
    random.seed(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)

set_seeds(42)
```

In [163…
```
import tensorflow as tf
from keras import models
from keras import layers
tf.random.set_seed(1)

model = models.Sequential()
model.add(layers.Dense(8, activation='relu', input_shape=(X_train.shape[1
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

In [164…
```
# use loss='binary_crossentropy'
model.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['a
```

In [165…
```
# epochs=100, batch_size=y_train.shape[0], use validation data
hist = model.fit(X_train, y_train, batch_size=y_train.shape[0], epochs=10
```

```
Epoch 1/100
1/1 [==============================] - 0s 285ms/step - loss: 1.1419 - ac
curacy: 0.7638 - val_loss: 0.6137 - val_accuracy: 0.8138
Epoch 2/100
1/1 [==============================] - 0s 13ms/step - loss: 0.9202 - acc
uracy: 0.7612 - val_loss: 0.5920 - val_accuracy: 0.8032
Epoch 3/100
1/1 [==============================] - 0s 13ms/step - loss: 0.8497 - acc
uracy: 0.7507 - val_loss: 0.5896 - val_accuracy: 0.7766
Epoch 4/100
1/1 [==============================] - 0s 13ms/step - loss: 0.8120 - acc
uracy: 0.7244 - val_loss: 0.5941 - val_accuracy: 0.7606
Epoch 5/100
1/1 [==============================] - 0s 14ms/step - loss: 0.7843 - acc
uracy: 0.7060 - val_loss: 0.5981 - val_accuracy: 0.7181
Epoch 6/100
1/1 [==============================] - 0s 14ms/step - loss: 0.7650 - acc
uracy: 0.7034 - val_loss: 0.6022 - val_accuracy: 0.7128
Epoch 7/100
1/1 [==============================] - 0s 14ms/step - loss: 0.7524 - acc
uracy: 0.6903 - val_loss: 0.6059 - val_accuracy: 0.6968
Epoch 8/100
1/1 [==============================] - 0s 13ms/step - loss: 0.7423 - acc
uracy: 0.6850 - val_loss: 0.6089 - val_accuracy: 0.6968
Epoch 9/100
1/1 [==============================] - 0s 14ms/step - loss: 0.7337 - acc
uracy: 0.6745 - val_loss: 0.6122 - val_accuracy: 0.6862
Epoch 10/100
1/1 [==============================] - 0s 14ms/step - loss: 0.7279 - acc
uracy: 0.6745 - val_loss: 0.6145 - val_accuracy: 0.6755
Epoch 11/100
1/1 [==============================] - 0s 16ms/step - loss: 0.7246 - acc
uracy: 0.6693 - val_loss: 0.6164 - val_accuracy: 0.6809
Epoch 12/100
1/1 [==============================] - 0s 16ms/step - loss: 0.7216 - acc
uracy: 0.6667 - val_loss: 0.6181 - val_accuracy: 0.6755
Epoch 13/100
1/1 [==============================] - 0s 15ms/step - loss: 0.7189 - acc
uracy: 0.6667 - val_loss: 0.6196 - val_accuracy: 0.6702
Epoch 14/100
1/1 [==============================] - 0s 15ms/step - loss: 0.7165 - acc
uracy: 0.6614 - val_loss: 0.6200 - val_accuracy: 0.6755
Epoch 15/100
1/1 [==============================] - 0s 15ms/step - loss: 0.7143 - acc
uracy: 0.6640 - val_loss: 0.6205 - val_accuracy: 0.6755
Epoch 16/100
1/1 [==============================] - 0s 16ms/step - loss: 0.7121 - acc
uracy: 0.6667 - val_loss: 0.6206 - val_accuracy: 0.6755
Epoch 17/100
1/1 [==============================] - 0s 16ms/step - loss: 0.7100 - acc
uracy: 0.6640 - val_loss: 0.6206 - val_accuracy: 0.6436
Epoch 18/100
1/1 [==============================] - 0s 15ms/step - loss: 0.7079 - acc
uracy: 0.6194 - val_loss: 0.6202 - val_accuracy: 0.6436
Epoch 19/100
1/1 [==============================] - 0s 15ms/step - loss: 0.7058 - acc
uracy: 0.6194 - val_loss: 0.6197 - val_accuracy: 0.6436
Epoch 20/100
1/1 [==============================] - 0s 16ms/step - loss: 0.7037 - acc
uracy: 0.6194 - val_loss: 0.6192 - val_accuracy: 0.6436
```

```
Epoch 21/100
1/1 [==============================] - 0s 16ms/step - loss: 0.7016 - acc
uracy: 0.6194 - val_loss: 0.6186 - val_accuracy: 0.6436
Epoch 22/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6995 - acc
uracy: 0.6194 - val_loss: 0.6181 - val_accuracy: 0.6436
Epoch 23/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6974 - acc
uracy: 0.6194 - val_loss: 0.6176 - val_accuracy: 0.6436
Epoch 24/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6954 - acc
uracy: 0.6194 - val_loss: 0.6175 - val_accuracy: 0.6436
Epoch 25/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6933 - acc
uracy: 0.6194 - val_loss: 0.6166 - val_accuracy: 0.6436
Epoch 26/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6913 - acc
uracy: 0.6194 - val_loss: 0.6159 - val_accuracy: 0.6436
Epoch 27/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6892 - acc
uracy: 0.6194 - val_loss: 0.6145 - val_accuracy: 0.6436
Epoch 28/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6871 - acc
uracy: 0.6194 - val_loss: 0.6116 - val_accuracy: 0.6436
Epoch 29/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6846 - acc
uracy: 0.6194 - val_loss: 0.6096 - val_accuracy: 0.6436
Epoch 30/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6823 - acc
uracy: 0.6194 - val_loss: 0.6086 - val_accuracy: 0.6436
Epoch 31/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6801 - acc
uracy: 0.6194 - val_loss: 0.6075 - val_accuracy: 0.6436
Epoch 32/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6779 - acc
uracy: 0.6194 - val_loss: 0.6068 - val_accuracy: 0.6436
Epoch 33/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6758 - acc
uracy: 0.6194 - val_loss: 0.6046 - val_accuracy: 0.6436
Epoch 34/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6736 - acc
uracy: 0.6194 - val_loss: 0.6046 - val_accuracy: 0.6436
Epoch 35/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6716 - acc
uracy: 0.6194 - val_loss: 0.6019 - val_accuracy: 0.6436
Epoch 36/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6696 - acc
uracy: 0.6194 - val_loss: 0.6019 - val_accuracy: 0.6436
Epoch 37/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6677 - acc
uracy: 0.6194 - val_loss: 0.6026 - val_accuracy: 0.6436
Epoch 38/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6658 - acc
uracy: 0.6194 - val_loss: 0.6057 - val_accuracy: 0.6436
Epoch 39/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6644 - acc
uracy: 0.6194 - val_loss: 0.6035 - val_accuracy: 0.6436
Epoch 40/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6626 - acc
uracy: 0.6194 - val_loss: 0.6015 - val_accuracy: 0.6436
```

```
Epoch 41/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6609 - acc
uracy: 0.6194 - val_loss: 0.5950 - val_accuracy: 0.6436
Epoch 42/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6590 - acc
uracy: 0.6194 - val_loss: 0.5948 - val_accuracy: 0.6436
Epoch 43/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6574 - acc
uracy: 0.6194 - val_loss: 0.5959 - val_accuracy: 0.6436
Epoch 44/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6559 - acc
uracy: 0.6194 - val_loss: 0.5961 - val_accuracy: 0.6436
Epoch 45/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6544 - acc
uracy: 0.6194 - val_loss: 0.5931 - val_accuracy: 0.6436
Epoch 46/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6528 - acc
uracy: 0.6194 - val_loss: 0.5933 - val_accuracy: 0.6436
Epoch 47/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6515 - acc
uracy: 0.6194 - val_loss: 0.5930 - val_accuracy: 0.6436
Epoch 48/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6503 - acc
uracy: 0.6194 - val_loss: 0.5896 - val_accuracy: 0.6436
Epoch 49/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6490 - acc
uracy: 0.6194 - val_loss: 0.5909 - val_accuracy: 0.6436
Epoch 50/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6478 - acc
uracy: 0.6194 - val_loss: 0.5911 - val_accuracy: 0.6436
Epoch 51/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6467 - acc
uracy: 0.6194 - val_loss: 0.5863 - val_accuracy: 0.6436
Epoch 52/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6452 - acc
uracy: 0.6194 - val_loss: 0.5839 - val_accuracy: 0.6436
Epoch 53/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6441 - acc
uracy: 0.6194 - val_loss: 0.5867 - val_accuracy: 0.6436
Epoch 54/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6428 - acc
uracy: 0.6194 - val_loss: 0.5900 - val_accuracy: 0.6436
Epoch 55/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6422 - acc
uracy: 0.6194 - val_loss: 0.5898 - val_accuracy: 0.6436
Epoch 56/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6414 - acc
uracy: 0.6194 - val_loss: 0.5806 - val_accuracy: 0.6436
Epoch 57/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6400 - acc
uracy: 0.6194 - val_loss: 0.5820 - val_accuracy: 0.6436
Epoch 58/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6388 - acc
uracy: 0.6194 - val_loss: 0.5794 - val_accuracy: 0.6436
Epoch 59/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6379 - acc
uracy: 0.6194 - val_loss: 0.5818 - val_accuracy: 0.6436
Epoch 60/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6368 - acc
uracy: 0.6194 - val_loss: 0.5811 - val_accuracy: 0.6436
```

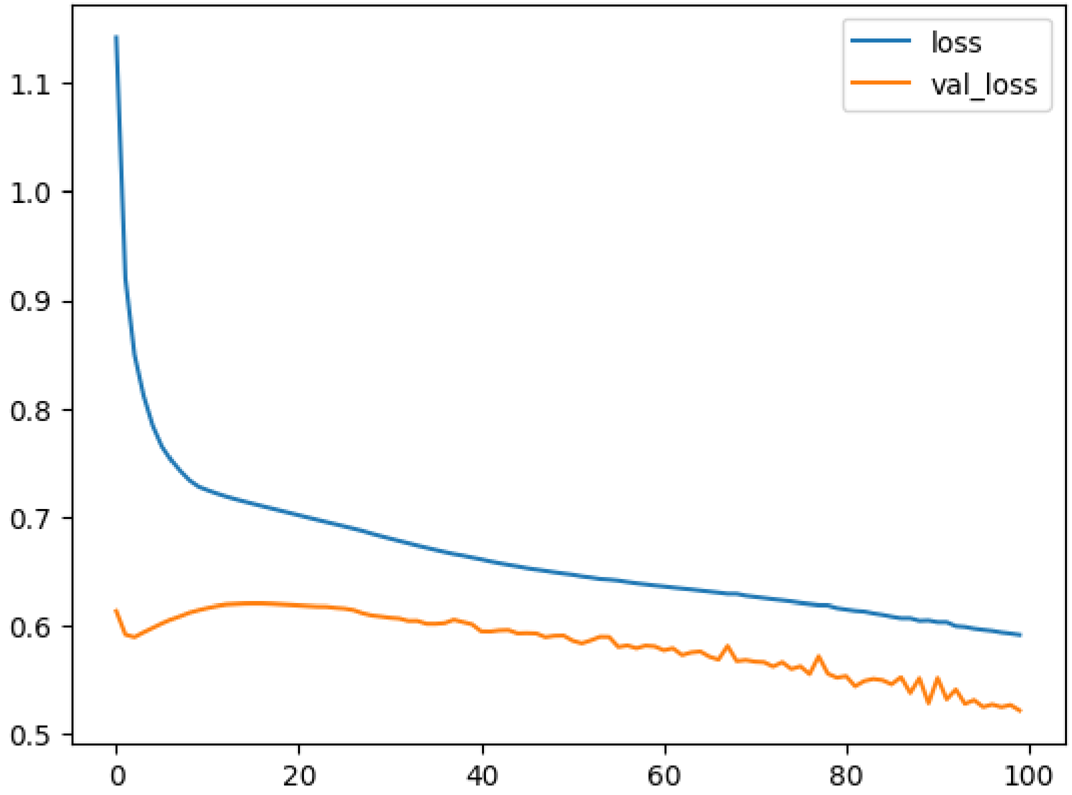
```
Epoch 61/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6359 - acc
uracy: 0.6194 - val_loss: 0.5775 - val_accuracy: 0.6436
Epoch 62/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6349 - acc
uracy: 0.6194 - val_loss: 0.5794 - val_accuracy: 0.6436
Epoch 63/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6340 - acc
uracy: 0.6194 - val_loss: 0.5731 - val_accuracy: 0.6436
Epoch 64/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6332 - acc
uracy: 0.6194 - val_loss: 0.5755 - val_accuracy: 0.6436
Epoch 65/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6322 - acc
uracy: 0.6194 - val_loss: 0.5764 - val_accuracy: 0.6436
Epoch 66/100
1/1 [==============================] - 0s 17ms/step - loss: 0.6312 - acc
uracy: 0.6194 - val_loss: 0.5716 - val_accuracy: 0.6436
Epoch 67/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6303 - acc
uracy: 0.6194 - val_loss: 0.5689 - val_accuracy: 0.6436
Epoch 68/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6292 - acc
uracy: 0.6194 - val_loss: 0.5815 - val_accuracy: 0.6436
Epoch 69/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6292 - acc
uracy: 0.6194 - val_loss: 0.5675 - val_accuracy: 0.6436
Epoch 70/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6274 - acc
uracy: 0.6194 - val_loss: 0.5685 - val_accuracy: 0.6436
Epoch 71/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6264 - acc
uracy: 0.6194 - val_loss: 0.5670 - val_accuracy: 0.6436
Epoch 72/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6253 - acc
uracy: 0.6194 - val_loss: 0.5665 - val_accuracy: 0.6436
Epoch 73/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6242 - acc
uracy: 0.6194 - val_loss: 0.5625 - val_accuracy: 0.6436
Epoch 74/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6233 - acc
uracy: 0.6194 - val_loss: 0.5665 - val_accuracy: 0.6436
Epoch 75/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6223 - acc
uracy: 0.6194 - val_loss: 0.5602 - val_accuracy: 0.6436
Epoch 76/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6210 - acc
uracy: 0.6194 - val_loss: 0.5627 - val_accuracy: 0.6436
Epoch 77/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6199 - acc
uracy: 0.6194 - val_loss: 0.5555 - val_accuracy: 0.6436
Epoch 78/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6188 - acc
uracy: 0.6194 - val_loss: 0.5720 - val_accuracy: 0.6436
Epoch 79/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6187 - acc
uracy: 0.6194 - val_loss: 0.5560 - val_accuracy: 0.6436
Epoch 80/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6163 - acc
uracy: 0.6194 - val_loss: 0.5522 - val_accuracy: 0.6436
```

```
Epoch 81/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6150 - acc
uracy: 0.6194 - val_loss: 0.5537 - val_accuracy: 0.6436
Epoch 82/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6138 - acc
uracy: 0.6194 - val_loss: 0.5443 - val_accuracy: 0.6436
Epoch 83/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6132 - acc
uracy: 0.6194 - val_loss: 0.5491 - val_accuracy: 0.6436
Epoch 84/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6115 - acc
uracy: 0.6194 - val_loss: 0.5511 - val_accuracy: 0.6436
Epoch 85/100
1/1 [==============================] - 0s 16ms/step - loss: 0.6101 - acc
uracy: 0.6194 - val_loss: 0.5499 - val_accuracy: 0.6436
Epoch 86/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6085 - acc
uracy: 0.6194 - val_loss: 0.5461 - val_accuracy: 0.6436
Epoch 87/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6070 - acc
uracy: 0.6194 - val_loss: 0.5525 - val_accuracy: 0.6436
Epoch 88/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6070 - acc
uracy: 0.6194 - val_loss: 0.5381 - val_accuracy: 0.6436
Epoch 89/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6047 - acc
uracy: 0.6194 - val_loss: 0.5518 - val_accuracy: 0.6436
Epoch 90/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6050 - acc
uracy: 0.6194 - val_loss: 0.5286 - val_accuracy: 0.6436
Epoch 91/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6035 - acc
uracy: 0.6194 - val_loss: 0.5520 - val_accuracy: 0.6436
Epoch 92/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6034 - acc
uracy: 0.6194 - val_loss: 0.5323 - val_accuracy: 0.6436
Epoch 93/100
1/1 [==============================] - 0s 15ms/step - loss: 0.5998 - acc
uracy: 0.6194 - val_loss: 0.5414 - val_accuracy: 0.8564
Epoch 94/100
1/1 [==============================] - 0s 15ms/step - loss: 0.5991 - acc
uracy: 0.8346 - val_loss: 0.5281 - val_accuracy: 0.8830
Epoch 95/100
1/1 [==============================] - 0s 14ms/step - loss: 0.5974 - acc
uracy: 0.8346 - val_loss: 0.5316 - val_accuracy: 0.8830
Epoch 96/100
1/1 [==============================] - 0s 16ms/step - loss: 0.5963 - acc
uracy: 0.8320 - val_loss: 0.5252 - val_accuracy: 0.8883
Epoch 97/100
1/1 [==============================] - 0s 17ms/step - loss: 0.5954 - acc
uracy: 0.8346 - val_loss: 0.5275 - val_accuracy: 0.8883
Epoch 98/100
1/1 [==============================] - 0s 15ms/step - loss: 0.5939 - acc
uracy: 0.8373 - val_loss: 0.5251 - val_accuracy: 0.8883
Epoch 99/100
1/1 [==============================] - 0s 15ms/step - loss: 0.5929 - acc
uracy: 0.8373 - val_loss: 0.5271 - val_accuracy: 0.8830
Epoch 100/100
1/1 [==============================] - 0s 14ms/step - loss: 0.5917 - acc
uracy: 0.8346 - val_loss: 0.5220 - val_accuracy: 0.8936
```

In [166…

```
plt.plot(hist.history['loss'],label='loss')
plt.plot(hist.history['val_loss'],label='val_loss')
plt.legend()
```

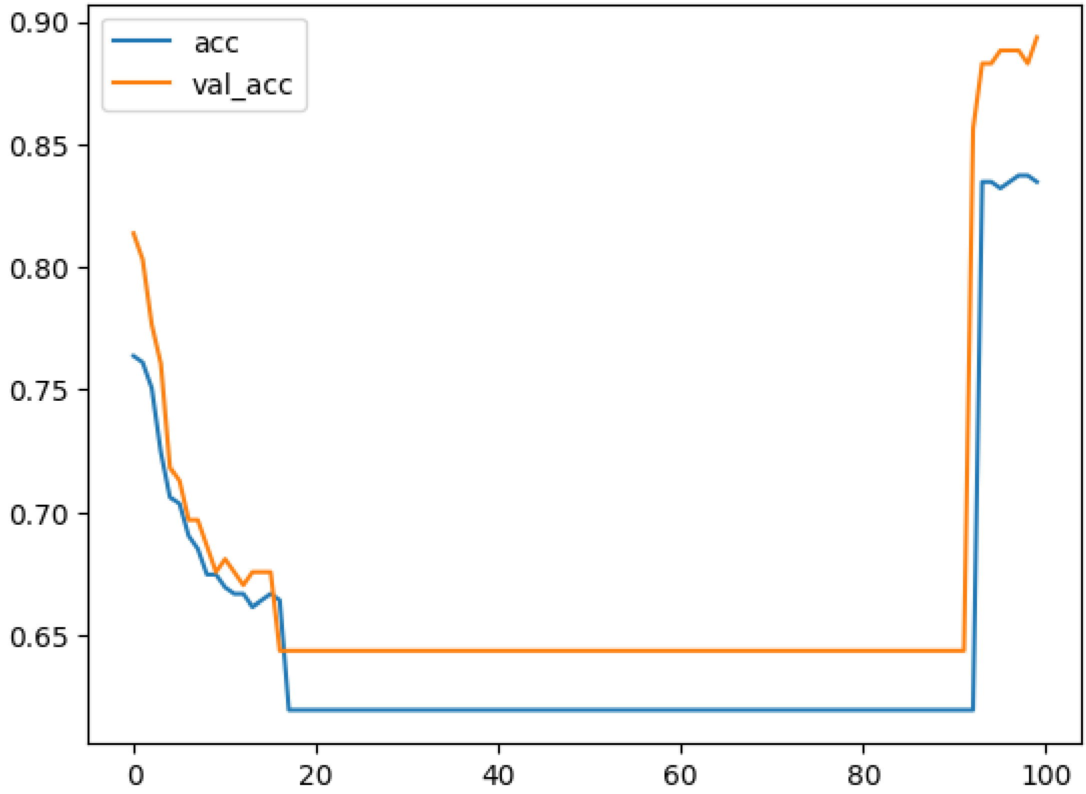
Out[166]: <matplotlib.legend.Legend at 0x2ef4195a0>



In [167…

```
plt.plot(hist.history['accuracy'],label='acc')
plt.plot(hist.history['val_accuracy'],label='val_acc')
plt.legend()
```

Out[167]: <matplotlib.legend.Legend at 0x2fdfe75b0>



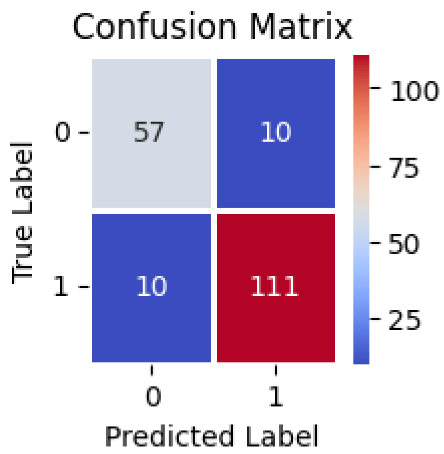
In [168…

```
probs = model.predict(X_test).flatten()
y_pred = np.where(probs>.5,1,0)
```

```
6/6 [==============================] - 0s 602us/step
```

In [169…

```
show_confusion_matrix(y_test, y_pred, 2)
```



In [170…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        67
           1       0.92      0.92      0.92       121

    accuracy                           0.89       188
   macro avg       0.88      0.88      0.88       188
weighted avg       0.89      0.89      0.89       188
```

2.4 Model #output neurons = #classes

We reload Fashion MNIST data... This may be skipped

In [171…
```
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X=np.concatenate((np.array(x_train),np.array(x_test)),axis=0);
y=np.concatenate((np.array(y_train),np.array(y_test)),axis=0);
# X=X.reshape(X.shape[0],X.shape[1]*X.shape[2])
X=X/255

X2 = X[(y==0) | (y==1)]
y2=y[(y==0) | (y==1)]
X2=X2.reshape(X2.shape[0],-1)
```

In [172…
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.3
```

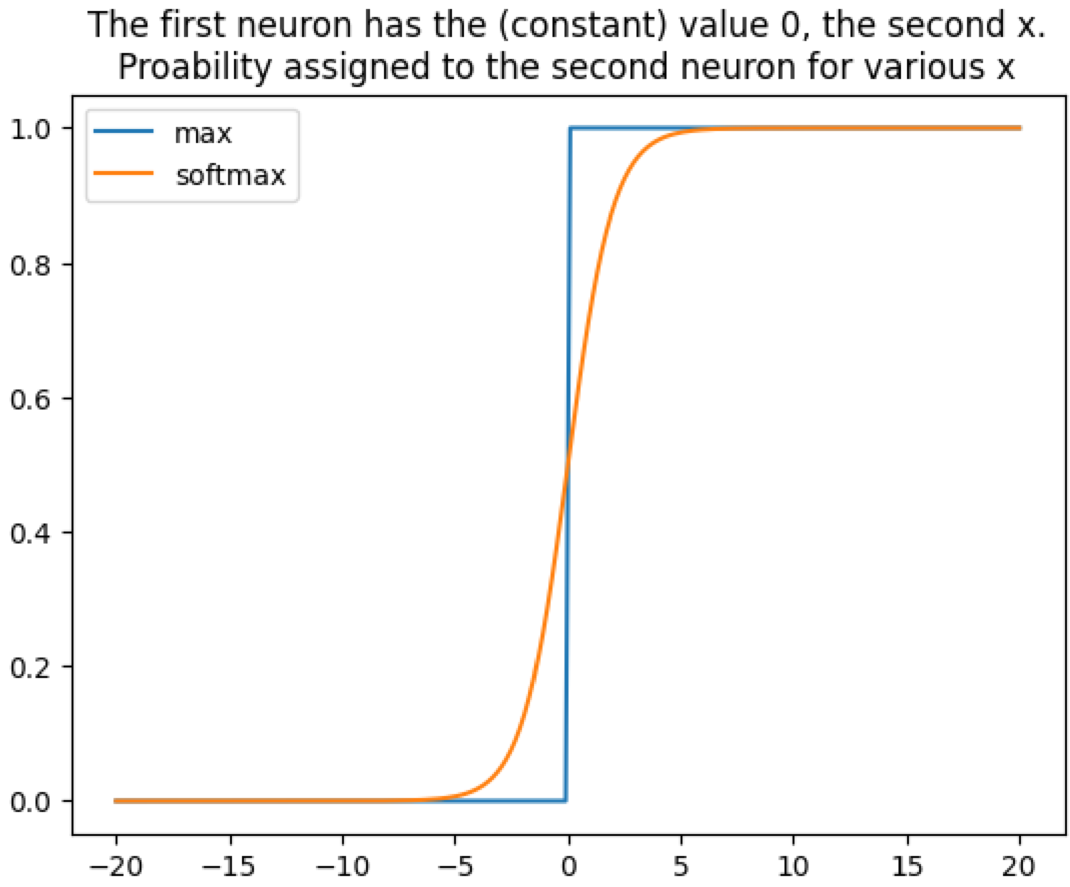
We create a neural net with two neurons in the output layer and softmax activation function. As the loss, sparse_categorical_crossentropy is used, $\sum_{i=1}^{k} y_i \cdot ln(p_i)$

In [173…
```
# softmax slects a neuron with the highest probability value
# but it is smooth and differentiable


x=np.linspace(-20,20,200)
zeros = x*0
X=np.stack((zeros,x),axis=-1)

plt.title("The first neuron has the (constant) value 0, the second x.\nPr
plt.plot(x,np.argmax(X,axis=1),label='max')
plt.plot(x,np.exp(X[:,1])/((np.exp(X[:,0])+np.exp(X[:,1]))),label='softma
plt.legend()
```

Out[173]:
```
<matplotlib.legend.Legend at 0x3232b67d0>
```



In [174…

```
from keras import models
from keras import layers


num_classes = y_train.max()+1

network = models.Sequential()
network.add(layers.Dense(256, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(40, activation='relu'))
network.add(layers.Dense(num_classes, activation='softmax'))
network.compile(optimizer='rmsprop',loss='sparse_categorical_crossentropy
```

In [175…

```
hist = network.fit(X_train,y_train,epochs=10,batch_size=128)
```

```
Epoch 1/10
74/74 [==============================] - 0s 2ms/step - loss: 0.1136 - ac
curacy: 0.9539
Epoch 2/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0517 - ac
curacy: 0.9834
Epoch 3/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0359 - ac
curacy: 0.9874
Epoch 4/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0290 - ac
curacy: 0.9900
Epoch 5/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0243 - ac
curacy: 0.9916
Epoch 6/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0241 - ac
curacy: 0.9910
Epoch 7/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0221 - ac
curacy: 0.9936
Epoch 8/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0187 - ac
curacy: 0.9933
Epoch 9/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0177 - ac
curacy: 0.9936
Epoch 10/10
74/74 [==============================] - 0s 2ms/step - loss: 0.0110 - ac
curacy: 0.9952
```

Get output probabilities

In [176…

```
probs = network.predict(X_test)
```

```
145/145 [==============================] - 0s 601us/step
```

And load them to a data frame

In [177…

```
import pandas as pd

df = pd.DataFrame(probs)
df.head(df.size)
```

Out[177]:

	0	1
0	1.000000e+00	5.036720e-10
1	1.000000e+00	2.522759e-09
2	7.518909e-09	1.000000e+00
3	1.000000e+00	1.989617e-16
4	2.011340e-11	1.000000e+00
...	...	...
4615	1.211228e-08	1.000000e+00
4616	2.363294e-10	1.000000e+00
4617	4.692118e-06	9.999954e-01
4618	1.000000e+00	1.659254e-12
4619	1.714707e-07	9.999999e-01

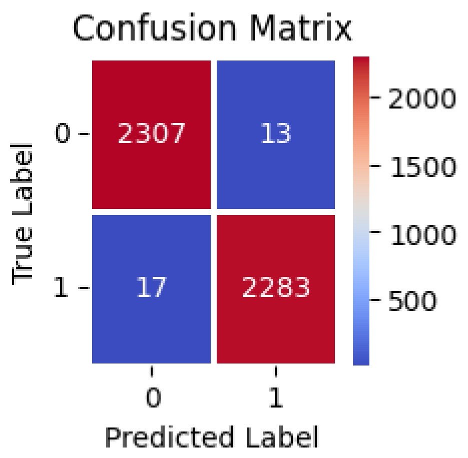
4620 rows × 2 columns

Determine predicted labels as arg_max (computed horizontally)

In [178…
```
y_pred = np.argmax(probs,axis=1)
```

Display confusion matrix and classification report

In [179…
```
show_confusion_matrix(y_test, y_pred, 2)
```



In [180…
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2320
           1       0.99      0.99      0.99      2300

    accuracy                           0.99      4620
   macro avg       0.99      0.99      0.99      4620
weighted avg       0.99      0.99      0.99      4620
```

2.5 Build a model for all ten fashion classes

In [181…

```
import numpy as np
from tensorflow.keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
X=np.concatenate((np.array(x_train),np.array(x_test)),axis=0);
y=np.concatenate((np.array(y_train),np.array(y_test)),axis=0);

X=X/255

X=X.reshape(X.shape[0],-1)
```

In [182…

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
```

TODO 2.5.1 Use the same model configuration, but adapt it to appropriate number of classes

In [183…

```
from keras import models
from keras import layers


num_classes = y_train.max()+1
print(num_classes)
# create a model add layers, compile

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
model.compile(optimizer='rmsprop',loss='sparse_categorical_crossentropy',
```

```
10
```

Fit the model

In [184…

```
hist = model.fit(X_train, y_train,epochs=10,batch_size=128)
```

```
Epoch 1/10
367/367 [==============================] - 2s 4ms/step - loss: 0.6695 -
accuracy: 0.7493
Epoch 2/10
367/367 [==============================] - 2s 4ms/step - loss: 0.4348 -
accuracy: 0.8415
Epoch 3/10
367/367 [==============================] - 2s 4ms/step - loss: 0.3845 -
accuracy: 0.8596
Epoch 4/10
367/367 [==============================] - 2s 4ms/step - loss: 0.3504 -
accuracy: 0.8713
Epoch 5/10
367/367 [==============================] - 2s 4ms/step - loss: 0.3290 -
accuracy: 0.8797
Epoch 6/10
367/367 [==============================] - 2s 4ms/step - loss: 0.3114 -
accuracy: 0.8852
Epoch 7/10
367/367 [==============================] - 2s 4ms/step - loss: 0.2941 -
accuracy: 0.8912
Epoch 8/10
367/367 [==============================] - 2s 4ms/step - loss: 0.2819 -
accuracy: 0.8956
Epoch 9/10
367/367 [==============================] - 2s 4ms/step - loss: 0.2703 -
accuracy: 0.8989
Epoch 10/10
367/367 [==============================] - 2s 4ms/step - loss: 0.2594 -
accuracy: 0.9026
```

Make predictions

In [185…

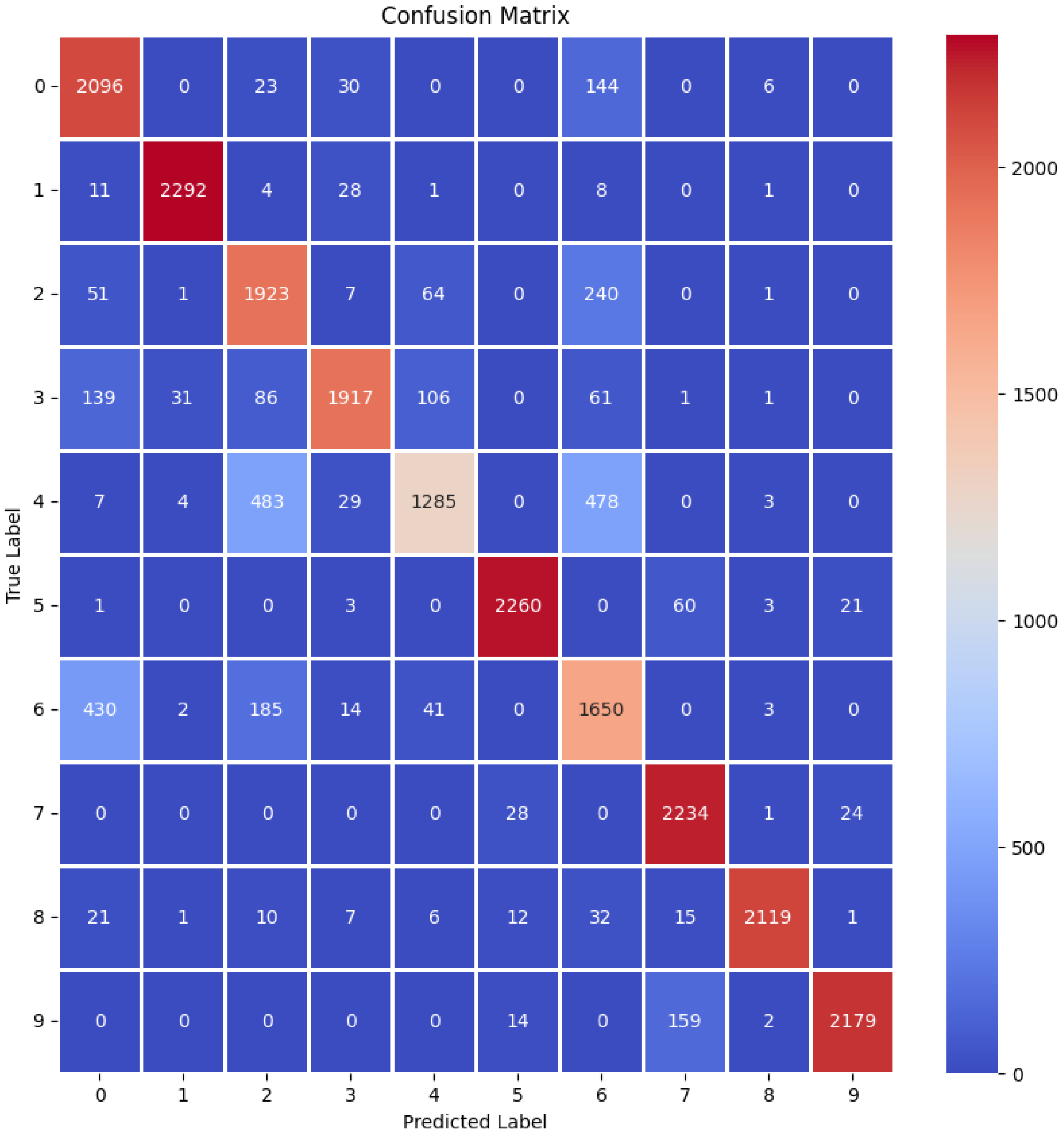
```
probs = model.predict(X_test)
print(f'Probs shape={probs.shape}')
y_pred = np.argmax(probs,axis=1)
```

```
722/722 [==============================] - 1s 897us/step
Probs shape=(23100, 10)
```

Show confusion matrix and scores (classification report)

In [186…

```
show_confusion_matrix(y_test, y_pred, 10)
```



In [187…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.91      0.83      2299
           1       0.98      0.98      0.98      2345
           2       0.71      0.84      0.77      2287
           3       0.94      0.82      0.88      2342
           4       0.85      0.56      0.68      2289
           5       0.98      0.96      0.97      2348
           6       0.63      0.71      0.67      2325
           7       0.90      0.98      0.94      2287
           8       0.99      0.95      0.97      2224
           9       0.98      0.93      0.95      2354

    accuracy                           0.86     23100
   macro avg       0.87      0.86      0.86     23100
weighted avg       0.87      0.86      0.86     23100
```

TODO 2.5.2 Analyze the results. Which fashion classes are wrongly classified. Can you explain that by similarity of forms?

2.6 Analyze the iris dataset

You can find the dataset description here

TODO 2.6.1 Implement the following steps

- First load data (code provided)
- Create a neural network comprising one hidden layer with 4 units
- Experimentaly establish the number of epochs during training.
- Provide validation data
- Display loss/validation loss and accuracies
- Predict output labels
- Display the confussion matrix and scores

In [188…

```
from sklearn.datasets import load_iris

X,y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
X.shape
```

Out[188]:

```
(150, 4)
```

In [189…

```
# Build the model

from keras import models
from keras import layers

num_classes = y_train.max()+1

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(4,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(num_classes, activation='softmax'))
model.compile(optimizer='rmsprop',loss='sparse_categorical_crossentropy',

model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_66 (Dense)            (None, 512)               2560

 dense_67 (Dense)            (None, 20)                10260

 dense_68 (Dense)            (None, 3)                 63

=================================================================
Total params: 12,883
Trainable params: 12,883
Non-trainable params: 0
_________________________________________________________________
```

In [190…

```
print(X_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
hist = model.fit(X_train,y_train, epochs=60, batch_size=y_train.shape[0],
```

```
(100, 4) (100,) (10000, 28, 28) (50,)
Epoch 1/60
1/1 [==============================] - 0s 268ms/step - loss: 1.0636 - ac
curacy: 0.3400 - val_loss: 0.9579 - val_accuracy: 0.7000
Epoch 2/60
1/1 [==============================] - 0s 13ms/step - loss: 0.9714 - acc
uracy: 0.6500 - val_loss: 0.9472 - val_accuracy: 0.3200
Epoch 3/60
1/1 [==============================] - 0s 14ms/step - loss: 0.9203 - acc
uracy: 0.3400 - val_loss: 0.8093 - val_accuracy: 0.7000
Epoch 4/60
1/1 [==============================] - 0s 14ms/step - loss: 0.8285 - acc
uracy: 0.6500 - val_loss: 0.7854 - val_accuracy: 0.7000
Epoch 5/60
1/1 [==============================] - 0s 14ms/step - loss: 0.7970 - acc
uracy: 0.6500 - val_loss: 0.7380 - val_accuracy: 0.7000
Epoch 6/60
1/1 [==============================] - 0s 14ms/step - loss: 0.7607 - acc
uracy: 0.6500 - val_loss: 0.7087 - val_accuracy: 0.7000
Epoch 7/60
1/1 [==============================] - 0s 13ms/step - loss: 0.7351 - acc
uracy: 0.6500 - val_loss: 0.6868 - val_accuracy: 0.7000
Epoch 8/60
1/1 [==============================] - 0s 14ms/step - loss: 0.7133 - acc
uracy: 0.6500 - val_loss: 0.6593 - val_accuracy: 0.7000
Epoch 9/60
1/1 [==============================] - 0s 13ms/step - loss: 0.6915 - acc
uracy: 0.6500 - val_loss: 0.6410 - val_accuracy: 0.7000
Epoch 10/60
1/1 [==============================] - 0s 14ms/step - loss: 0.6730 - acc
uracy: 0.6500 - val_loss: 0.6188 - val_accuracy: 0.7000
Epoch 11/60
1/1 [==============================] - 0s 14ms/step - loss: 0.6553 - acc
uracy: 0.6500 - val_loss: 0.6036 - val_accuracy: 0.7000
Epoch 12/60
1/1 [==============================] - 0s 13ms/step - loss: 0.6395 - acc
uracy: 0.6500 - val_loss: 0.5853 - val_accuracy: 0.7000
Epoch 13/60
1/1 [==============================] - 0s 14ms/step - loss: 0.6254 - acc
uracy: 0.6500 - val_loss: 0.5739 - val_accuracy: 0.7000
Epoch 14/60
1/1 [==============================] - 0s 14ms/step - loss: 0.6120 - acc
uracy: 0.6500 - val_loss: 0.5562 - val_accuracy: 0.7000
Epoch 15/60
1/1 [==============================] - 0s 14ms/step - loss: 0.5990 - acc
uracy: 0.6500 - val_loss: 0.5461 - val_accuracy: 0.7200
Epoch 16/60
1/1 [==============================] - 0s 14ms/step - loss: 0.5864 - acc
uracy: 0.7000 - val_loss: 0.5305 - val_accuracy: 0.7000
Epoch 17/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5750 - acc
uracy: 0.6500 - val_loss: 0.5222 - val_accuracy: 0.8200
Epoch 18/60
1/1 [==============================] - 0s 14ms/step - loss: 0.5641 - acc
uracy: 0.7900 - val_loss: 0.5081 - val_accuracy: 0.7000
Epoch 19/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5539 - acc
uracy: 0.6700 - val_loss: 0.5025 - val_accuracy: 0.8600
Epoch 20/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5458 - acc
```

```
uracy: 0.8100 - val_loss: 0.4910 - val_accuracy: 0.7000
Epoch 21/60
1/1 [==============================] - 0s 14ms/step - loss: 0.5399 - acc
uracy: 0.6500 - val_loss: 0.4837 - val_accuracy: 0.9000
Epoch 22/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5283 - acc
uracy: 0.9100 - val_loss: 0.4719 - val_accuracy: 0.7200
Epoch 23/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5190 - acc
uracy: 0.7000 - val_loss: 0.4648 - val_accuracy: 0.8800
Epoch 24/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5108 - acc
uracy: 0.8400 - val_loss: 0.4559 - val_accuracy: 0.8200
Epoch 25/60
1/1 [==============================] - 0s 13ms/step - loss: 0.5030 - acc
uracy: 0.7900 - val_loss: 0.4492 - val_accuracy: 0.8800
Epoch 26/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4955 - acc
uracy: 0.8700 - val_loss: 0.4407 - val_accuracy: 0.8200
Epoch 27/60
1/1 [==============================] - 0s 14ms/step - loss: 0.4882 - acc
uracy: 0.8000 - val_loss: 0.4349 - val_accuracy: 0.9000
Epoch 28/60
1/1 [==============================] - 0s 14ms/step - loss: 0.4813 - acc
uracy: 0.9100 - val_loss: 0.4271 - val_accuracy: 0.8000
Epoch 29/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4762 - acc
uracy: 0.7500 - val_loss: 0.4237 - val_accuracy: 1.0000
Epoch 30/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4693 - acc
uracy: 0.9700 - val_loss: 0.4141 - val_accuracy: 0.8200
Epoch 31/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4633 - acc
uracy: 0.7500 - val_loss: 0.4111 - val_accuracy: 0.9800
Epoch 32/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4568 - acc
uracy: 0.9600 - val_loss: 0.4032 - val_accuracy: 0.7800
Epoch 33/60
1/1 [==============================] - 0s 14ms/step - loss: 0.4531 - acc
uracy: 0.7300 - val_loss: 0.3997 - val_accuracy: 0.9800
Epoch 34/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4451 - acc
uracy: 0.9700 - val_loss: 0.3897 - val_accuracy: 0.8400
Epoch 35/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4380 - acc
uracy: 0.8200 - val_loss: 0.3860 - val_accuracy: 1.0000
Epoch 36/60
1/1 [==============================] - 0s 14ms/step - loss: 0.4316 - acc
uracy: 0.9700 - val_loss: 0.3780 - val_accuracy: 0.8800
Epoch 37/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4257 - acc
uracy: 0.8600 - val_loss: 0.3749 - val_accuracy: 1.0000
Epoch 38/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4203 - acc
uracy: 0.9700 - val_loss: 0.3693 - val_accuracy: 0.8200
Epoch 39/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4184 - acc
uracy: 0.8000 - val_loss: 0.3676 - val_accuracy: 0.9800
Epoch 40/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4122 - acc
```

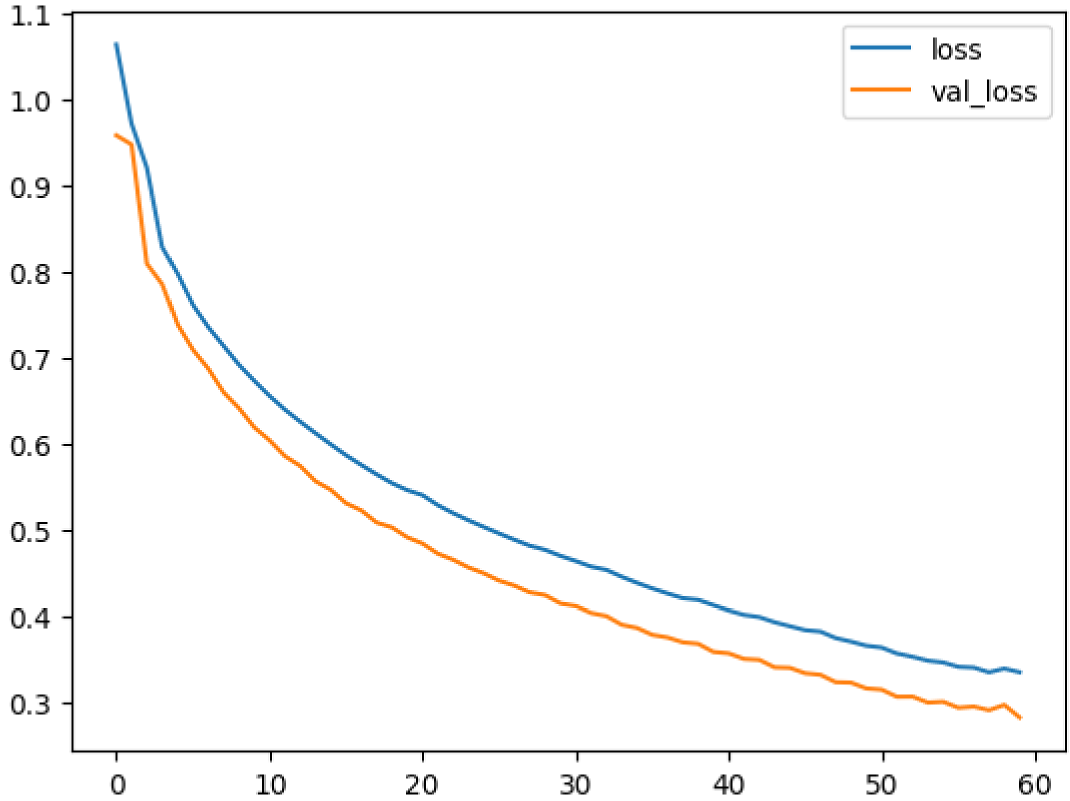
```
uracy: 0.9700 - val_loss: 0.3581 - val_accuracy: 0.8800
Epoch 41/60
1/1 [==============================] - 0s 14ms/step - loss: 0.4061 - acc
uracy: 0.8600 - val_loss: 0.3565 - val_accuracy: 0.9800
Epoch 42/60
1/1 [==============================] - 0s 13ms/step - loss: 0.4011 - acc
uracy: 0.9700 - val_loss: 0.3500 - val_accuracy: 0.8800
Epoch 43/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3985 - acc
uracy: 0.8300 - val_loss: 0.3490 - val_accuracy: 0.9600
Epoch 44/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3926 - acc
uracy: 0.9600 - val_loss: 0.3402 - val_accuracy: 0.8800
Epoch 45/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3880 - acc
uracy: 0.8600 - val_loss: 0.3397 - val_accuracy: 0.9600
Epoch 46/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3834 - acc
uracy: 0.9600 - val_loss: 0.3333 - val_accuracy: 0.8800
Epoch 47/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3818 - acc
uracy: 0.8400 - val_loss: 0.3316 - val_accuracy: 0.9600
Epoch 48/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3742 - acc
uracy: 0.9600 - val_loss: 0.3229 - val_accuracy: 0.9000
Epoch 49/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3701 - acc
uracy: 0.8900 - val_loss: 0.3225 - val_accuracy: 0.9800
Epoch 50/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3654 - acc
uracy: 0.9700 - val_loss: 0.3157 - val_accuracy: 0.9000
Epoch 51/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3633 - acc
uracy: 0.8900 - val_loss: 0.3143 - val_accuracy: 0.9800
Epoch 52/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3563 - acc
uracy: 0.9700 - val_loss: 0.3060 - val_accuracy: 0.9400
Epoch 53/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3524 - acc
uracy: 0.9200 - val_loss: 0.3061 - val_accuracy: 0.9800
Epoch 54/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3479 - acc
uracy: 0.9700 - val_loss: 0.2992 - val_accuracy: 0.9400
Epoch 55/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3459 - acc
uracy: 0.9100 - val_loss: 0.3001 - val_accuracy: 0.9600
Epoch 56/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3408 - acc
uracy: 0.9800 - val_loss: 0.2932 - val_accuracy: 0.9200
Epoch 57/60
1/1 [==============================] - 0s 13ms/step - loss: 0.3399 - acc
uracy: 0.9000 - val_loss: 0.2946 - val_accuracy: 0.9600
Epoch 58/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3344 - acc
uracy: 0.9700 - val_loss: 0.2904 - val_accuracy: 0.8800
Epoch 59/60
1/1 [==============================] - 0s 14ms/step - loss: 0.3387 - acc
uracy: 0.8700 - val_loss: 0.2965 - val_accuracy: 0.9200
Epoch 60/60
```

```
1/1 [==============================] - 0s 13ms/step - loss: 0.3345 - acc
uracy: 0.9500 - val_loss: 0.2819 - val_accuracy: 0.9200
```

In [191…

```
#plot loss
plt.plot(hist.history['loss'],label='loss')
plt.plot(hist.history['val_loss'],label='val_loss')
plt.legend()
```

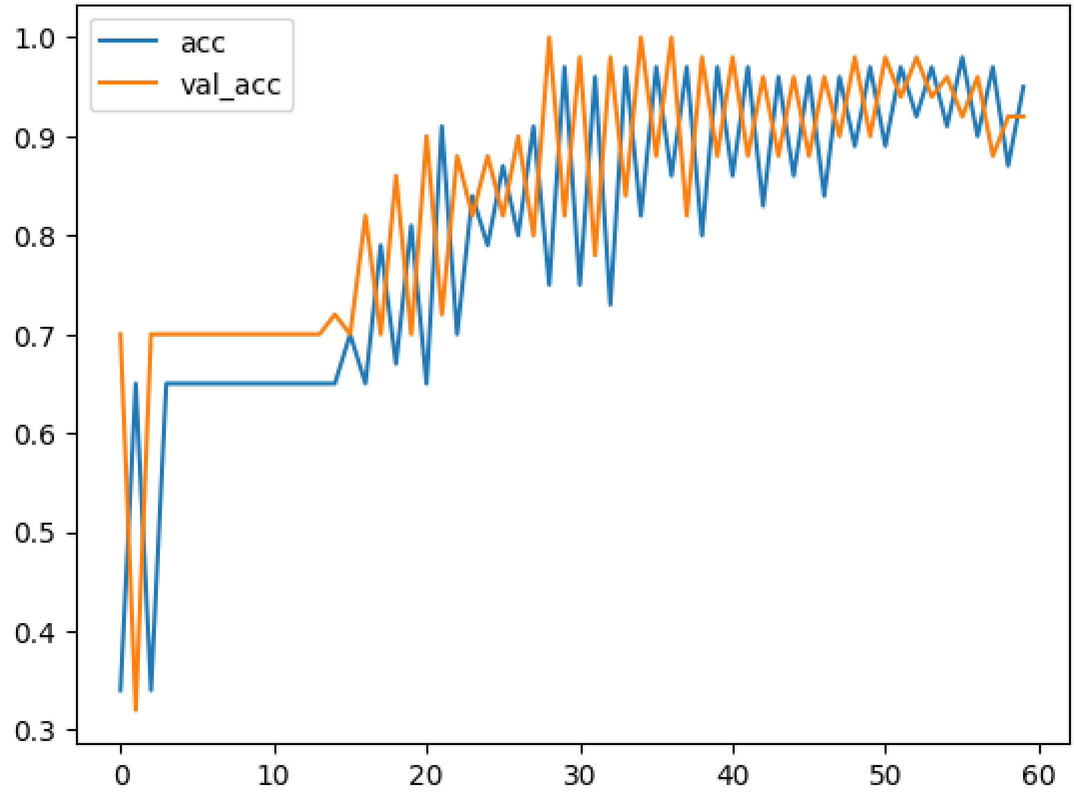
Out[191]: <matplotlib.legend.Legend at 0x2a97734c0>



In [192…

```
#Plot accuracy
plt.plot(hist.history['accuracy'],label='acc')
plt.plot(hist.history['val_accuracy'],label='val_acc')
plt.legend()
```

Out[192]: <matplotlib.legend.Legend at 0x2a9826f50>



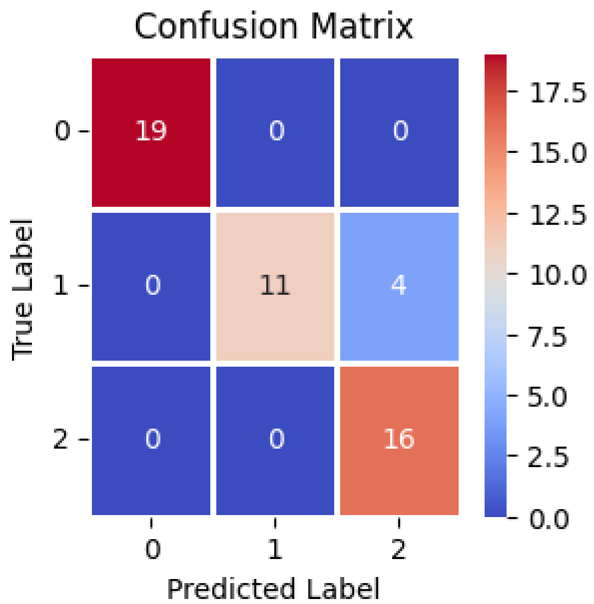
In [193…

```
# Make predictions
probs = model.predict(X_test)
print(f'Probs shape={probs.shape}')
y_pred = np.argmax(probs,axis=1)
```

```
2/2 [==============================] - 0s 1ms/step
Probs shape=(50, 3)
```

In [194…

```
# Confussion matrix
show_confusion_matrix(y_test, y_pred, 3)
```



In [195…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.73      0.85        15
           2       0.80      1.00      0.89        16

    accuracy                           0.92        50
   macro avg       0.93      0.91      0.91        50
weighted avg       0.94      0.92      0.92        50
```