In [62]:

```
import glob
import os
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

In [63]:

```
images_up = glob.glob("./veriseti/up/*.png")
images_down = glob.glob("./veriseti/down/*.png")
images_right = glob.glob("./veriseti/right/*.png")
```

In [64]:

```
width = 125
height = 70

X = []   #resimler
Y = []   #label yani etiket
```

In [65]:

```
for img in images_up:
    im = np.array(Image.open(img).convert("L").resize((width,height)))
    im = im / 255    # 0-1 arası ölçeklendirme
    X.append(im)
    Y.append("up")

for img in images_down:
    im = np.array(Image.open(img).convert("L").resize((width,height)))
    im = im / 255
    X.append(im)
    Y.append("down")

for img in images_right:
    im = np.array(Image.open(img).convert("L").resize((width,height)))
    im = im / 255
    X.append(im)
    Y.append("right")
```

In [66]:

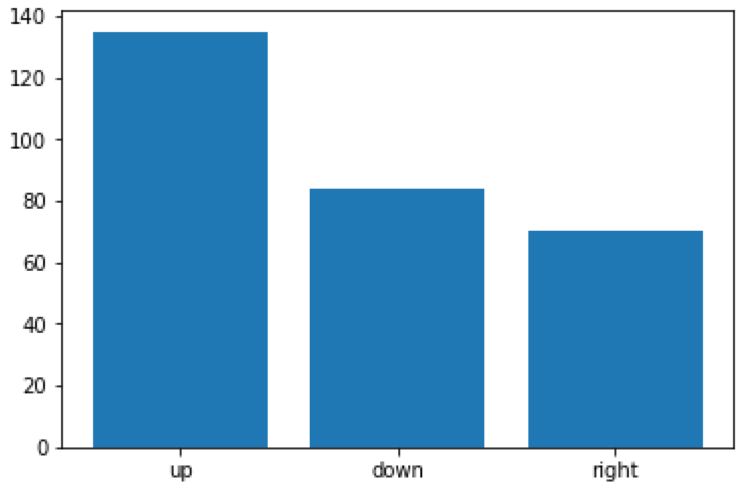
```
X = np.array(X)    # listeyi array'e çevirme
X = X.reshape(X.shape[0], width, height, 1)     # shape --> (289, 125, 70, 1)
#shape[0] toplam resim sayısı --- 1 parametresi siyah beyaz kanala çevirir
```

In [67]:

```
A = ["up","down","right"]
B = [Y.count("up"),Y.count("down"),Y.count("right")]
plt.bar(A,B)    #veri setinin dağılımı
```

Out[67]:

```
<BarContainer object of 3 artists>
```



In [68]:

```
# up,down,right label'larını 0-1-2 şeklinde kodlama
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
int_encoded = encoder.fit_transform(Y)
int_encoded # up=2 / down=0 / right=1
```

Out[68]:

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 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, 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, 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, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1], dtype=int64)
```

In [69]:

```
int_encoded = int_encoded.reshape(len(int_encoded),1)

from sklearn.preprocessing import OneHotEncoder
OneEncoder = OneHotEncoder(sparse=False)
OneEncoder = OneEncoder.fit_transform(int_encoded)
Y_encoded = OneEncoder
Y_encoded
# 001 --> 2 up
# 100 --> 0 down
# 010 --> 1 right
```

Out[69]:

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

In [70]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,Y_encoded,test_size=0.25)
```

In [71]:

```
import tensorflow as tf
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
girdi = (width,height,1) #başlangıç katmanında belirtmek yeterli

#Evrişim Katmanları
model.add(layers.Conv2D(32,kernel_size=(3,3),input_shape=girdi,activation="relu"))
model.add(layers.Conv2D(64,kernel_size=(3,3),activation="relu"))
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.Dropout(0.2))
model.add(layers.Flatten())

#Sınıflandırma Katmanları
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(3, activation="softmax"))  #3 label çıktısı

#optimizasyon (metrics --> değerlendirmeyi doğruluğa göre ölç)
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"])
```

In [72]:

```
# Eğitme
# Eğitim yaparken CPU %92'lerde işlem yaptı Batch oranını düşürebilirsin
model.fit(x_train,y_train,epochs=30,batch_size=64)
```

```
Epoch 1/30
4/4 [==============================] - 3s 674ms/step - loss: 8.6785 - accuracy: 0.32
87
Epoch 2/30
4/4 [==============================] - 3s 661ms/step - loss: 2.3344 - accuracy: 0.42
13
Epoch 3/30
4/4 [==============================] - 3s 683ms/step - loss: 0.8036 - accuracy: 0.63
43
Epoch 4/30
4/4 [==============================] - 3s 693ms/step - loss: 0.6332 - accuracy: 0.74
54
Epoch 5/30
4/4 [==============================] - 3s 693ms/step - loss: 0.4675 - accuracy: 0.90
28
Epoch 6/30
4/4 [==============================] - 3s 706ms/step - loss: 0.3594 - accuracy: 0.93
06
Epoch 7/30
4/4 [==============================] - 3s 816ms/step - loss: 0.2763 - accuracy: 0.93
98
Epoch 8/30
4/4 [==============================] - 3s 746ms/step - loss: 0.1958 - accuracy: 0.95
83
Epoch 9/30
4/4 [==============================] - 3s 726ms/step - loss: 0.1496 - accuracy: 0.96
30
Epoch 10/30
4/4 [==============================] - 3s 703ms/step - loss: 0.1401 - accuracy: 0.96
76
Epoch 11/30
4/4 [==============================] - 3s 688ms/step - loss: 0.1272 - accuracy: 0.97
22
Epoch 12/30
4/4 [==============================] - 3s 680ms/step - loss: 0.0788 - accuracy: 0.98
15
Epoch 13/30
```

```
4/4 [==============================] - 3s 697ms/step - loss: 0.0895 - accuracy: 0.97
22
Epoch 14/30
4/4 [==============================] - 3s 697ms/step - loss: 0.0884 - accuracy: 0.98
15
Epoch 15/30
4/4 [==============================] - 3s 702ms/step - loss: 0.0887 - accuracy: 0.98
61
Epoch 16/30
4/4 [==============================] - 3s 695ms/step - loss: 0.1012 - accuracy: 0.97
69
Epoch 17/30
4/4 [==============================] - 3s 686ms/step - loss: 0.0761 - accuracy: 0.97
22
Epoch 18/30
4/4 [==============================] - 3s 683ms/step - loss: 0.0645 - accuracy: 0.98
61
Epoch 19/30
4/4 [==============================] - 3s 673ms/step - loss: 0.0628 - accuracy: 0.98
15
Epoch 20/30
4/4 [==============================] - 3s 677ms/step - loss: 0.0484 - accuracy: 0.99
07
Epoch 21/30
4/4 [==============================] - 3s 680ms/step - loss: 0.0539 - accuracy: 0.98
61
Epoch 22/30
4/4 [==============================] - 3s 680ms/step - loss: 0.0473 - accuracy: 0.98
61
Epoch 23/30
4/4 [==============================] - 3s 691ms/step - loss: 0.0491 - accuracy: 0.99
07
Epoch 24/30
4/4 [==============================] - 3s 693ms/step - loss: 0.0407 - accuracy: 0.99
07
Epoch 25/30
4/4 [==============================] - 3s 691ms/step - loss: 0.0399 - accuracy: 0.99
54
Epoch 26/30
4/4 [==============================] - 3s 686ms/step - loss: 0.0349 - accuracy: 0.99
54
Epoch 27/30
4/4 [==============================] - 3s 677ms/step - loss: 0.0273 - accuracy: 0.99
54
Epoch 28/30
4/4 [==============================] - 3s 678ms/step - loss: 0.0293 - accuracy: 0.99
54
Epoch 29/30
4/4 [==============================] - 3s 677ms/step - loss: 0.0299 - accuracy: 0.99
07
Epoch 30/30
4/4 [==============================] - 3s 689ms/step - loss: 0.0272 - accuracy: 0.99
07
```

Out[72]:

```
<tensorflow.python.keras.callbacks.History at 0x16916169dc0>
```

In [73]:

```
score_train = model.evaluate(x_train,y_train)
print("Eğitim Doğruluğu: %",score_train[1]*100)

score_test = model.evaluate(x_test,y_test)
print("Test Doğruluğu: %",score_test[1]*100)
```

```
7/7 [==============================] - 1s 94ms/step - loss: 0.0156 - accuracy: 0.995
4
Eğitim Doğruluğu: % 99.53703880310059
3/3 [==============================] - 0s 53ms/step - loss: 0.0231 - accuracy: 0.986
3
Test Doğruluğu: % 98.63013625144958
```

In [74]:

```
#Modeli Kaydetme
open("TREX_model.json","w").write(model.to_json())
model.save("TREX_model.h5")
model.save_weights("TREX_weights.h5")
```

In [ ]:

```

```