In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageOps
import time, os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [37]:
```
my_location="D:\CSE504\Deep\IngilizAnahtari"
my_location2="D:\CSE504\Deep\Tornavida"
veri=np.array([])
cikis=np.array([])
```

In [38]:
```
for file in os.listdir(my_location2):
    print(my_location+'/'+file)
```

```
D:\CSE504\Deep\IngilizAnahtari/1.jpg
D:\CSE504\Deep\IngilizAnahtari/10.jpg
D:\CSE504\Deep\IngilizAnahtari/12.jpg
D:\CSE504\Deep\IngilizAnahtari/13.jpg
D:\CSE504\Deep\IngilizAnahtari/14.jpg
D:\CSE504\Deep\IngilizAnahtari/15.jpg
D:\CSE504\Deep\IngilizAnahtari/2.jpg
D:\CSE504\Deep\IngilizAnahtari/3.jpg
D:\CSE504\Deep\IngilizAnahtari/4.png
D:\CSE504\Deep\IngilizAnahtari/5.png
D:\CSE504\Deep\IngilizAnahtari/6.jpg
D:\CSE504\Deep\IngilizAnahtari/7.jpg
D:\CSE504\Deep\IngilizAnahtari/7.png
D:\CSE504\Deep\IngilizAnahtari/8.jpg
D:\CSE504\Deep\IngilizAnahtari/9.jpg
```

In [39]:
```
for file in os.listdir(my_location):
    img = cv2.imread(my_location + '/' + file)
    imgn=cv2.resize(img,(224,224))
    veri=np.append(veri,imgn)
    cikis=np.append(cikis,[1,0])
```

In [40]:
```
for file in os.listdir(my_location2):
    img = cv2.imread(my_location2 + '/' + file)
    imgn=cv2.resize(img,(224,224))
    veri=np.append(veri,imgn)
    cikis=np.append(cikis,[0,1])
```

In [41]:
```
veri.shape
```

Out[41]:
```
(4515840,)
```

In [42]:
```
veri_n=np.reshape(veri,(-1,224,224,3))
veri_n.shape
```

Out[42]:
```
(30, 224, 224, 3)
```

In [45]:
```
cikis_n=np.reshape(cikis,(-1,2))
```

In [46]:
```
cikis_n
```

Out[46]:
```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [1., 0.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [0., 1.]])
```

In [47]:
```
AlexNet = keras.Sequential()

AlexNet.add(layers.Conv2D(filters=96, input_shape=(224,224,3), kernel_size=(11,11), str
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding='same
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'
AlexNet.add(layers.BatchNormalization())

#4th Convolutional Layer
AlexNet.add(layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'
AlexNet.add(layers.BatchNormalization())

#5th Convolutional Layer
AlexNet.add(layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(layers.Flatten())
# 1st Fully Connected Layer
AlexNet.add(layers.Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.Activation('relu'))
# Add Dropout to prevent overfitting
```

```
AlexNet.add(layers.Dropout(0.4))

#2nd Fully Connected Layer
AlexNet.add(layers.Dense(4096))
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.Activation('relu'))
#Add Dropout
AlexNet.add(layers.Dropout(0.4))

#3rd Fully Connected Layer
AlexNet.add(layers.Dense(1000))
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.Activation('relu'))
#Add Dropout
AlexNet.add(layers.Dropout(0.4))

#Output Layer
AlexNet.add(layers.Dense(2))
AlexNet.add(layers.BatchNormalization())
AlexNet.add(layers.Activation('softmax'))
```

In [48]:
```
AlexNet.compile(loss = keras.losses.categorical_crossentropy, optimizer= 'adam', metric
```

In [49]:
```
AlexNet.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 56, 56, 96)        34944

 batch_normalization (BatchN  (None, 56, 56, 96)       384
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 28, 28, 96)       0
 )

 conv2d_1 (Conv2D)           (None, 28, 28, 256)       614656

 batch_normalization_1 (Batc  (None, 28, 28, 256)      1024
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 256)      0
 2D)

 conv2d_2 (Conv2D)           (None, 14, 14, 384)       885120

 batch_normalization_2 (Batc  (None, 14, 14, 384)      1536
 hNormalization)

 conv2d_3 (Conv2D)           (None, 14, 14, 384)       1327488

 batch_normalization_3 (Batc  (None, 14, 14, 384)      1536
 hNormalization)

 conv2d_4 (Conv2D)           (None, 14, 14, 256)       884992

 batch_normalization_4 (Batc  (None, 14, 14, 256)      1024
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 7, 7, 256)        0
 2D)
```

```
 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 4096)              51384320

 batch_normalization_5 (Batc  (None, 4096)             16384
 hNormalization)

 activation (Activation)     (None, 4096)              0

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 batch_normalization_6 (Batc  (None, 4096)             16384
 hNormalization)

 activation_1 (Activation)   (None, 4096)              0

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 1000)              4097000

 batch_normalization_7 (Batc  (None, 1000)             4000
 hNormalization)

 activation_2 (Activation)   (None, 1000)              0

 dropout_2 (Dropout)         (None, 1000)              0

 dense_3 (Dense)             (None, 2)                 2002

 batch_normalization_8 (Batc  (None, 2)                8
 hNormalization)

 activation_3 (Activation)   (None, 2)                 0

=================================================================
Total params: 76,054,114
Trainable params: 76,032,974
Non-trainable params: 21,140
_________________________________________________________________
```

In [50]:
```
test_veri=np.array([])
test_veri=veri_n[1:3]
test_veri=np.append(test_veri,veri_n[20:23])
test_verin=np.reshape(test_veri,(-1,224,224,3))

test_cikis=np.array([])
test_cikis=cikis_n[1:3]
test_cikis=np.append(test_cikis,cikis_n[20:23])
test_cikisn=np.reshape(test_cikis,(-1,2))
```

In [51]:
```
AlexNet.fit(veri_n/255.0, cikis_n, epochs=20)
```

```
Epoch 1/20
1/1 [==============================] - 3s 3s/step - loss: 1.0742 - accuracy: 0.4000
Epoch 2/20
1/1 [==============================] - 2s 2s/step - loss: 0.3199 - accuracy: 1.0000
Epoch 3/20
1/1 [==============================] - 1s 1s/step - loss: 0.2478 - accuracy: 0.9333
Epoch 4/20
1/1 [==============================] - 1s 1s/step - loss: 0.1653 - accuracy: 1.0000
Epoch 5/20
```

```
1/1 [==============================] - 1s 1s/step - loss: 0.1583 - accuracy: 1.0000
Epoch 6/20
1/1 [==============================] - 1s 1s/step - loss: 0.1417 - accuracy: 1.0000
Epoch 7/20
1/1 [==============================] - 1s 1s/step - loss: 0.1393 - accuracy: 1.0000
Epoch 8/20
1/1 [==============================] - 1s 1s/step - loss: 0.1433 - accuracy: 1.0000
Epoch 9/20
1/1 [==============================] - 1s 1s/step - loss: 0.1356 - accuracy: 1.0000
Epoch 10/20
1/1 [==============================] - 1s 1s/step - loss: 0.1451 - accuracy: 1.0000
Epoch 11/20
1/1 [==============================] - 1s 1s/step - loss: 0.1405 - accuracy: 1.0000
Epoch 12/20
1/1 [==============================] - 1s 1s/step - loss: 0.1400 - accuracy: 1.0000
Epoch 13/20
1/1 [==============================] - 1s 1s/step - loss: 0.1355 - accuracy: 1.0000
Epoch 14/20
1/1 [==============================] - 1s 1s/step - loss: 0.1357 - accuracy: 1.0000
Epoch 15/20
1/1 [==============================] - 1s 1s/step - loss: 0.1379 - accuracy: 1.0000
Epoch 16/20
1/1 [==============================] - 2s 2s/step - loss: 0.1357 - accuracy: 1.0000
Epoch 17/20
1/1 [==============================] - 2s 2s/step - loss: 0.1359 - accuracy: 1.0000
Epoch 18/20
1/1 [==============================] - 1s 1s/step - loss: 0.1351 - accuracy: 1.0000
Epoch 19/20
1/1 [==============================] - 2s 2s/step - loss: 0.1393 - accuracy: 1.0000
Epoch 20/20
1/1 [==============================] - 2s 2s/step - loss: 0.1316 - accuracy: 1.0000
```

Out[51]:
```
<keras.callbacks.History at 0x1920eba6790>
```

In [52]:
```
test_loss, test_acc = AlexNet.evaluate(test_verin/255.0,  test_cikisn, verbose=2)
```

```
1/1 - 0s - loss: 75.2748 - accuracy: 0.4000 - 279ms/epoch - 279ms/step
```

In [53]:
```
probability_model = tf.keras.Sequential([AlexNet,
                                         tf.keras.layers.Softmax()])
```

In [54]:
```
predictions = probability_model.predict(veri_n/255.0)
```

In [55]:
```
predictions
```

Out[55]:
```
array([[0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
```

```
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143],
       [0.7310586 , 0.26894143]], dtype=float32)
```

In [56]:

```
model = tf.keras.Sequential()
model.add(layers.Conv2D(32,3,padding="same", activation="relu", input_shape=(224,224,3)
model.add(layers.MaxPool2D())

model.add(layers.Conv2D(32, 3, padding="same", activation="relu"))
model.add(layers.MaxPool2D())

model.add(layers.Conv2D(64, 3, padding="same", activation="relu"))
model.add(layers.MaxPool2D())
model.add(layers.Dropout(0.4))

model.add(layers.Flatten())
model.add(layers.Dense(128,activation="relu"))
model.add(layers.Dense(2, activation="softmax"))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)            (None, 224, 224, 32)      896

 max_pooling2d_3 (MaxPooling  (None, 112, 112, 32)     0
 2D)

 conv2d_6 (Conv2D)            (None, 112, 112, 32)      9248

 max_pooling2d_4 (MaxPooling  (None, 56, 56, 32)       0
 2D)

 conv2d_7 (Conv2D)            (None, 56, 56, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 28, 28, 64)       0
 2D)

 dropout_3 (Dropout)          (None, 28, 28, 64)        0

 flatten_1 (Flatten)          (None, 50176)             0

 dense_4 (Dense)              (None, 128)               6422656

 dense_5 (Dense)              (None, 2)                 258

=================================================================
Total params: 6,451,554
Trainable params: 6,451,554
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss = keras.losses.categorical_crossentropy, optimizer= 'adam', metrics=
```

In [57]:

In [58]:
```
model.fit(veri_n/255, cikis_n, epochs=30)
```

```
Epoch 1/30
1/1 [==============================] - 1s 1s/step - loss: 0.6991 - accuracy: 0.4667
Epoch 2/30
1/1 [==============================] - 1s 640ms/step - loss: 8.8094 - accuracy: 0.5000
Epoch 3/30
1/1 [==============================] - 1s 666ms/step - loss: 1.6941 - accuracy: 0.5000
Epoch 4/30
1/1 [==============================] - 1s 623ms/step - loss: 1.0632 - accuracy: 0.5000
Epoch 5/30
1/1 [==============================] - 1s 656ms/step - loss: 0.6303 - accuracy: 0.5333
Epoch 6/30
1/1 [==============================] - 1s 715ms/step - loss: 0.6792 - accuracy: 0.5333
Epoch 7/30
1/1 [==============================] - 1s 715ms/step - loss: 0.5300 - accuracy: 0.9333
Epoch 8/30
1/1 [==============================] - 1s 618ms/step - loss: 0.5661 - accuracy: 0.6000
Epoch 9/30
1/1 [==============================] - 1s 734ms/step - loss: 0.4536 - accuracy: 0.9333
Epoch 10/30
1/1 [==============================] - 1s 690ms/step - loss: 0.4252 - accuracy: 0.8333
Epoch 11/30
1/1 [==============================] - 1s 689ms/step - loss: 0.3656 - accuracy: 0.9000
Epoch 12/30
1/1 [==============================] - 1s 719ms/step - loss: 0.3335 - accuracy: 0.9333
Epoch 13/30
1/1 [==============================] - 1s 583ms/step - loss: 0.2739 - accuracy: 1.0000
Epoch 14/30
1/1 [==============================] - 1s 592ms/step - loss: 0.2575 - accuracy: 0.9333
Epoch 15/30
1/1 [==============================] - 1s 630ms/step - loss: 0.2013 - accuracy: 1.0000
Epoch 16/30
1/1 [==============================] - 1s 725ms/step - loss: 0.1824 - accuracy: 0.9667
Epoch 17/30
1/1 [==============================] - 1s 668ms/step - loss: 0.1420 - accuracy: 0.9667
Epoch 18/30
1/1 [==============================] - 1s 703ms/step - loss: 0.1223 - accuracy: 0.9667
Epoch 19/30
1/1 [==============================] - 1s 676ms/step - loss: 0.0902 - accuracy: 1.0000
Epoch 20/30
1/1 [==============================] - 1s 635ms/step - loss: 0.0728 - accuracy: 1.0000
Epoch 21/30
1/1 [==============================] - 1s 727ms/step - loss: 0.0581 - accuracy: 1.0000
Epoch 22/30
1/1 [==============================] - 1s 632ms/step - loss: 0.0487 - accuracy: 1.0000
Epoch 23/30
1/1 [==============================] - 1s 616ms/step - loss: 0.0303 - accuracy: 1.0000
Epoch 24/30
1/1 [==============================] - 1s 607ms/step - loss: 0.0244 - accuracy: 1.0000
Epoch 25/30
1/1 [==============================] - 1s 733ms/step - loss: 0.0210 - accuracy: 1.0000
Epoch 26/30
1/1 [==============================] - 1s 676ms/step - loss: 0.0099 - accuracy: 1.0000
Epoch 27/30
1/1 [==============================] - 1s 624ms/step - loss: 0.0112 - accuracy: 1.0000
Epoch 28/30
1/1 [==============================] - 1s 567ms/step - loss: 0.0104 - accuracy: 1.0000
Epoch 29/30
1/1 [==============================] - 1s 648ms/step - loss: 0.0100 - accuracy: 1.0000
Epoch 30/30
1/1 [==============================] - 1s 598ms/step - loss: 0.0062 - accuracy: 1.0000
```

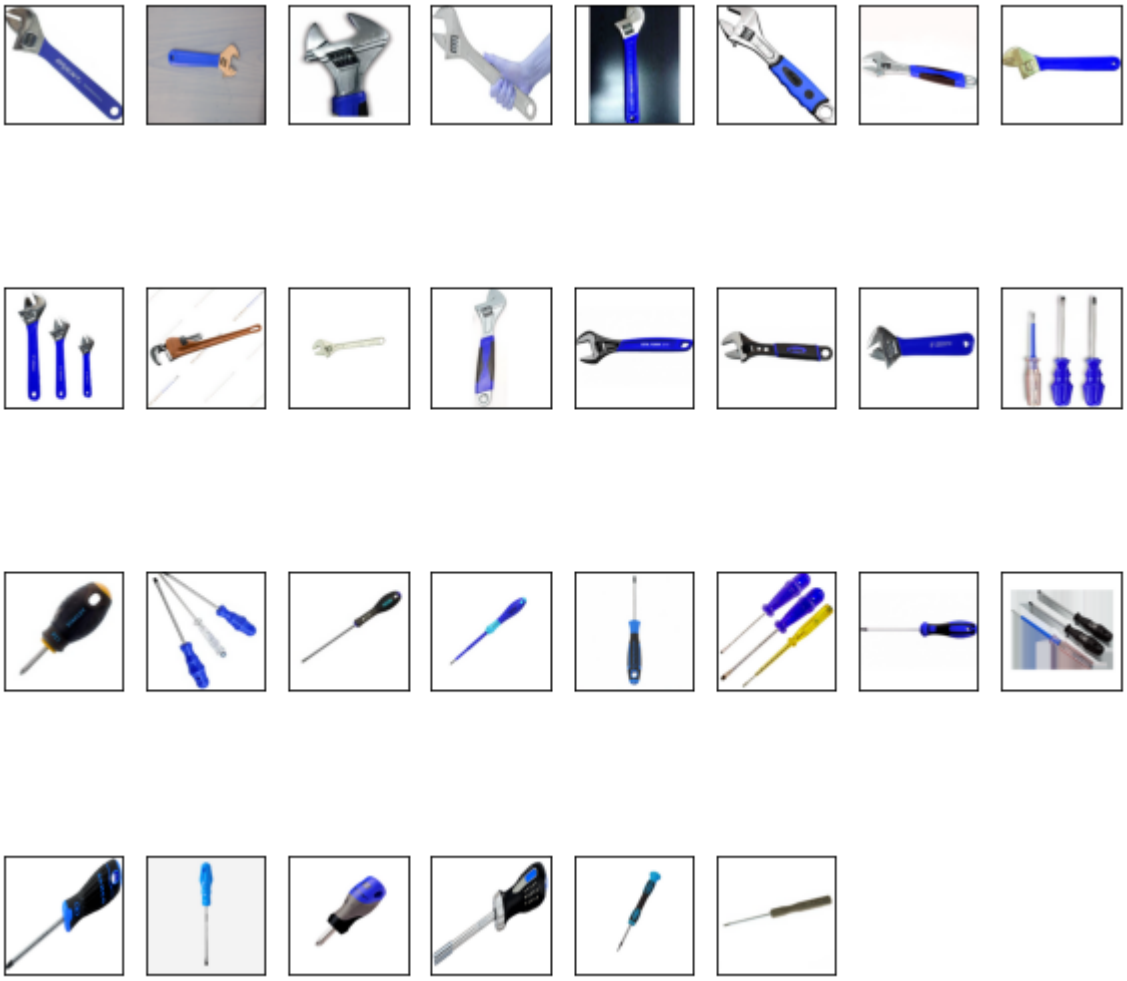
Out[58]: <keras.callbacks.History at 0x19211a1a580>

In [59]:
```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(veri_n/255)
```

In [60]:
```
predictions
```

Out[60]:
```
array([[0.73105854, 0.2689415 ],
       [0.73105264, 0.26894742],
       [0.7310376 , 0.26896235],
       [0.7308432 , 0.2691568 ],
       [0.73104846, 0.26895154],
       [0.7310586 , 0.26894143],
       [0.7310397 , 0.2689603 ],
       [0.7309653 , 0.26903462],
       [0.7310586 , 0.26894143],
       [0.7310547 , 0.26894525],
       [0.7148691 , 0.28513086],
       [0.7286222 , 0.2713778 ],
       [0.73105854, 0.26894146],
       [0.73105836, 0.26894167],
       [0.7310227 , 0.26897728],
       [0.2689415 , 0.73105854],
       [0.2689415 , 0.73105854],
       [0.26894197, 0.7310581 ],
       [0.26894143, 0.7310586 ],
       [0.26894143, 0.7310586 ],
       [0.26954427, 0.73045576],
       [0.26894143, 0.7310586 ],
       [0.26894173, 0.73105824],
       [0.26896012, 0.7310399 ],
       [0.26894143, 0.7310586 ],
       [0.27035284, 0.72964716],
       [0.26894143, 0.7310586 ],
       [0.26894143, 0.7310586 ],
       [0.26894173, 0.73105824],
       [0.28317386, 0.7168261 ]], dtype=float32)
```

In [61]:
```
imm=veri_n.astype('int16')
plt.figure(figsize=(10,10))
for i in range(30):
    plt.subplot(4,8,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(imm[i,:,:], cmap=plt.cm.binary)
plt.show()
```



In [ ]: