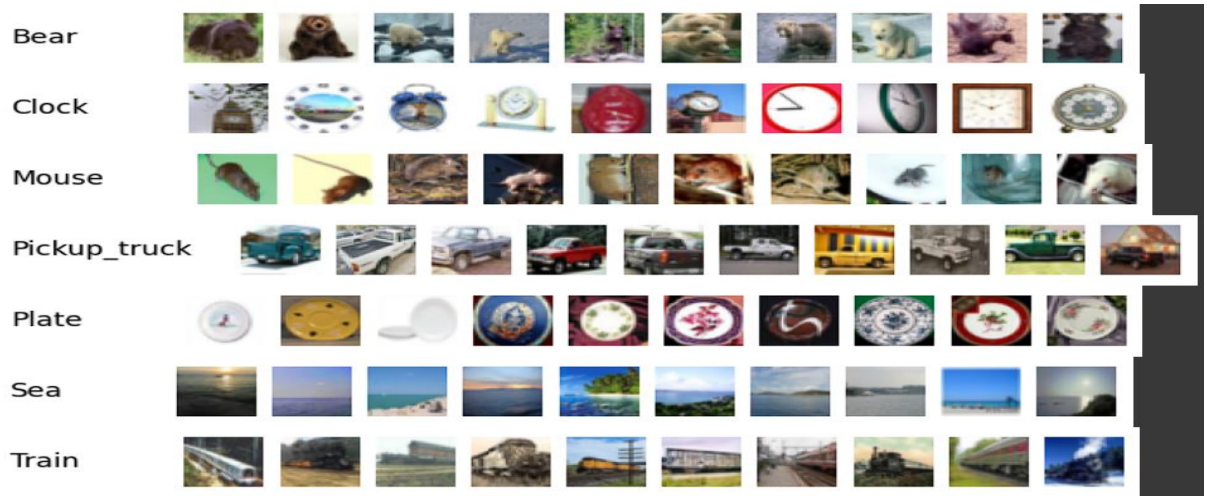
Derin Öğrenme Ve Evrişimli Sinir Ağları
Proje Ödevi

Öğrenci Adı: Enes Buğra Turğut

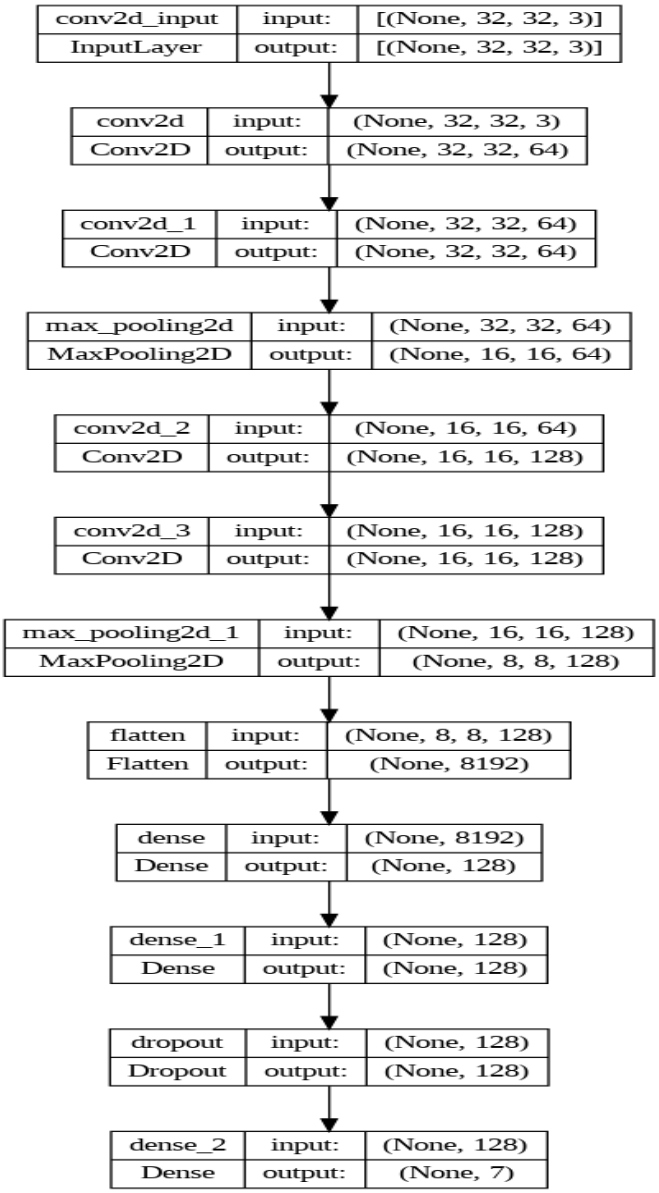
Öğrenci Numarası: G211210372

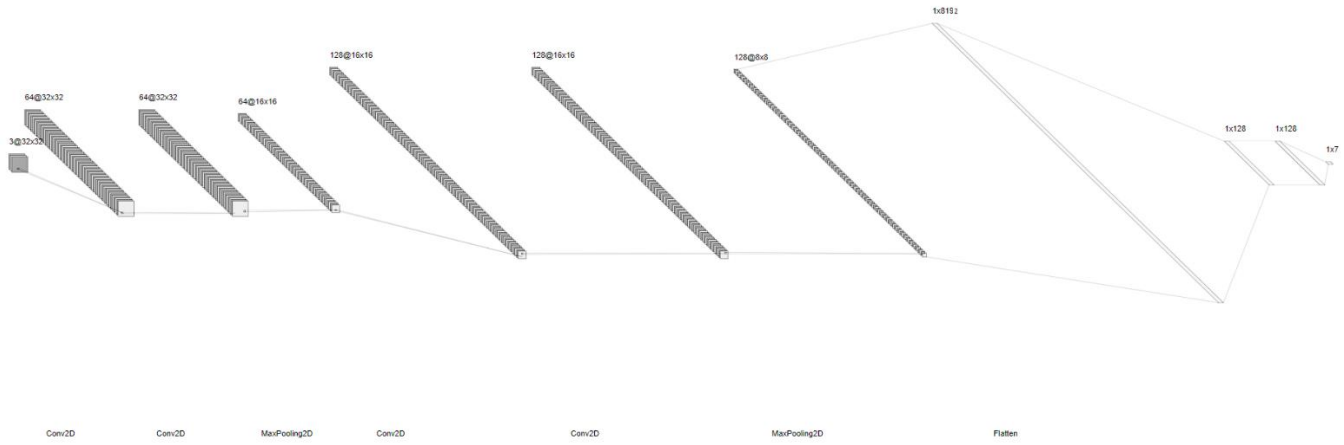
Öğrenim Türü: 2.Öğretim A grubu

1.Kullanılan sınıflar için örnek görüntüler

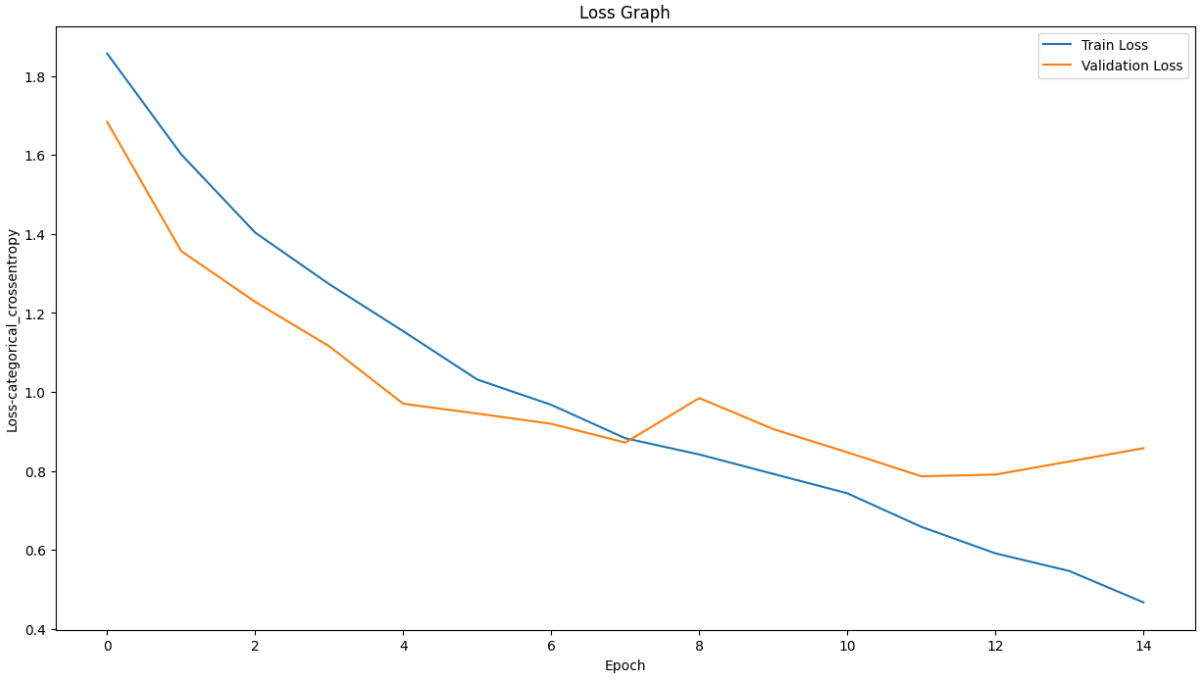


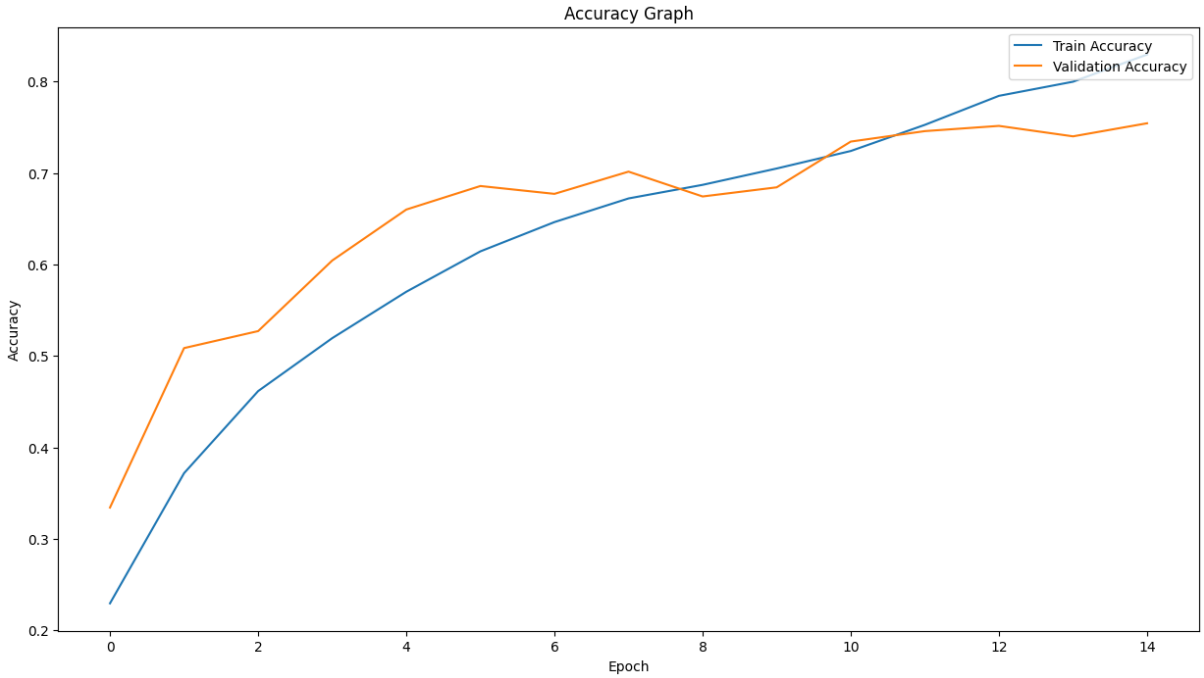
2.Blok Şeması




3@32x32
64@32x32
64@32x32
64@16x16
128@16x16
128@16x16
128@8x8
1x819
1x128
1x128
1x7
Conv2D
Conv2D
MaxPooling2D
Conv2D
Conv2D
MaxPooling2D
Flatten

3.Grafikler


Loss Graph
Train Loss
Validation Loss
Loss-categorical_crossentropy
Epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
2
4
6
8
10
12
14



Modelin kayıp oranı her epoch ta düşerek öğrenmenin git gide iyiye ilerlediğini gösterir. Bunula birlikte validation verisinin de kayıp oranı düşmektedir fakat 10.epochtan sonra val_loss değeri iyileşmemiş ve model overfit olmaya başlamıştır.

Accuracy grafiğinde ise model öğrenmesine devam ederken yine 10.epochtan sonra model overfit olmaya başlamıştır. Yani eğitim verilerini ezberliyor görmediği verilere karşı kötü sonuçlar çıkarıyor.

Modeli eğitirken erken durdurma fonksiyonu kullandım. Burada val_loss değerini takip ederek bu değerin 3 epoch iyileşmemesi durumunda eğitimi durdurdum ve en iyi ağırlıklarla modeli kaydettim. Ve model 12. Epochta durdurulup buradaki ağırlıklarla kaydedilmiştir.

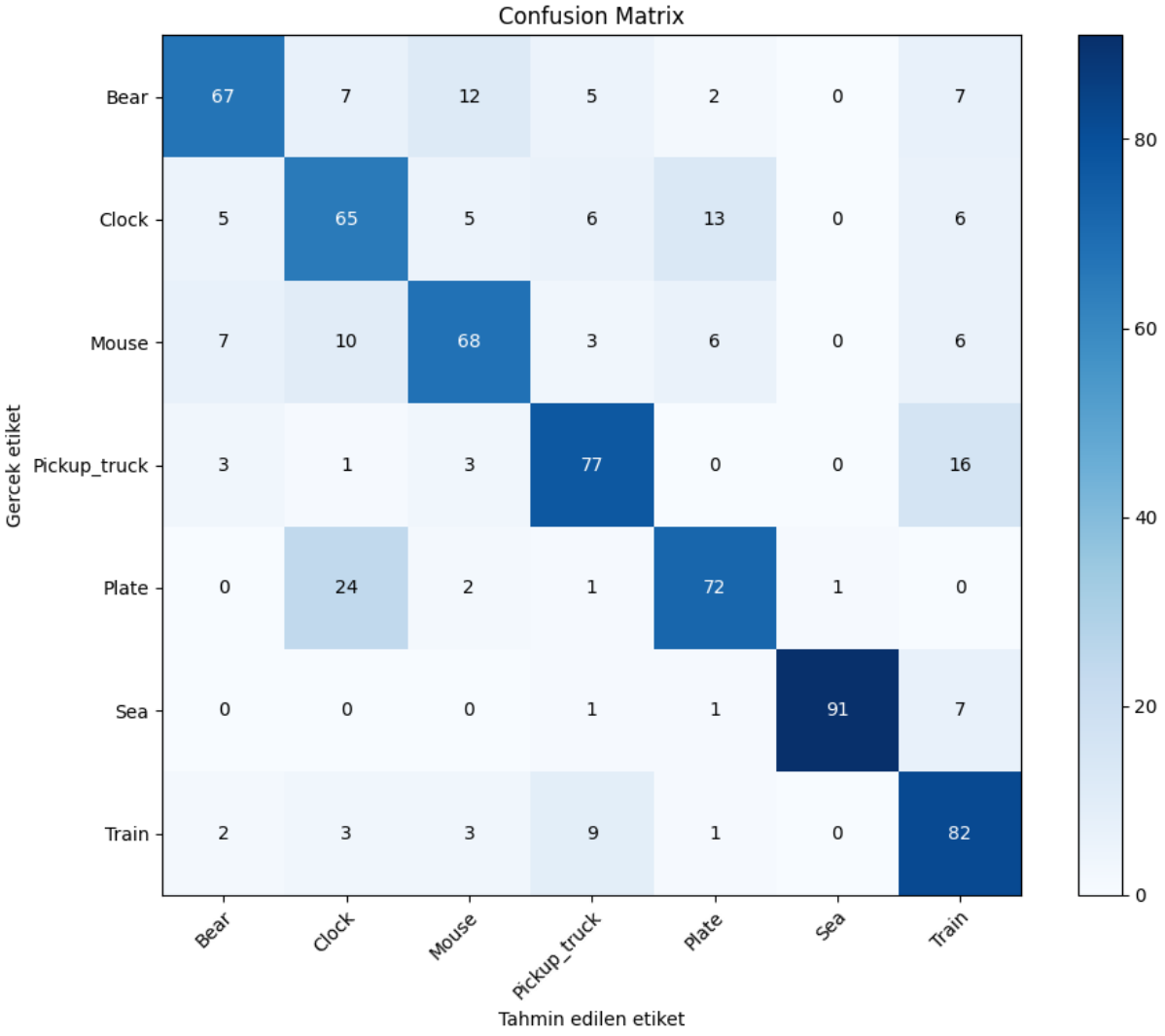
Modelin,

train_accuracy = 0.7516

test_accuracy = 0.7457

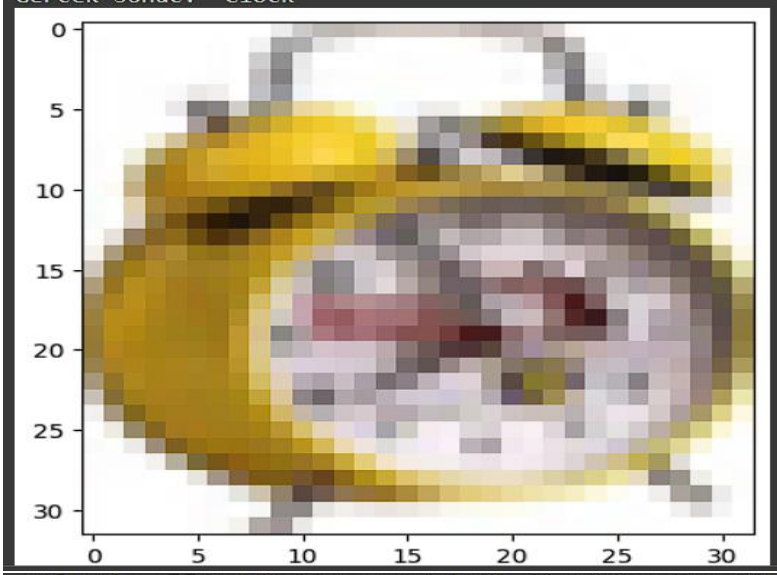
değeri elde edilmiştir. Model hiç görmediği verilere karşı eğitim doğruluğuna çok yakın bir sonuç vermiştir. Modelin hiç görmediği verilere karşı doğruluk performansı %74.57 diyebiliriz.

4.Confusion Matrix

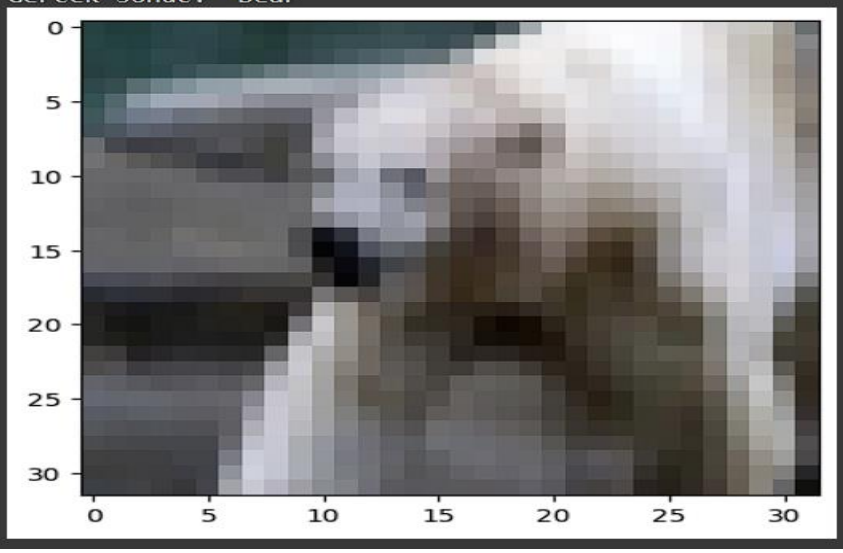


5.Predict

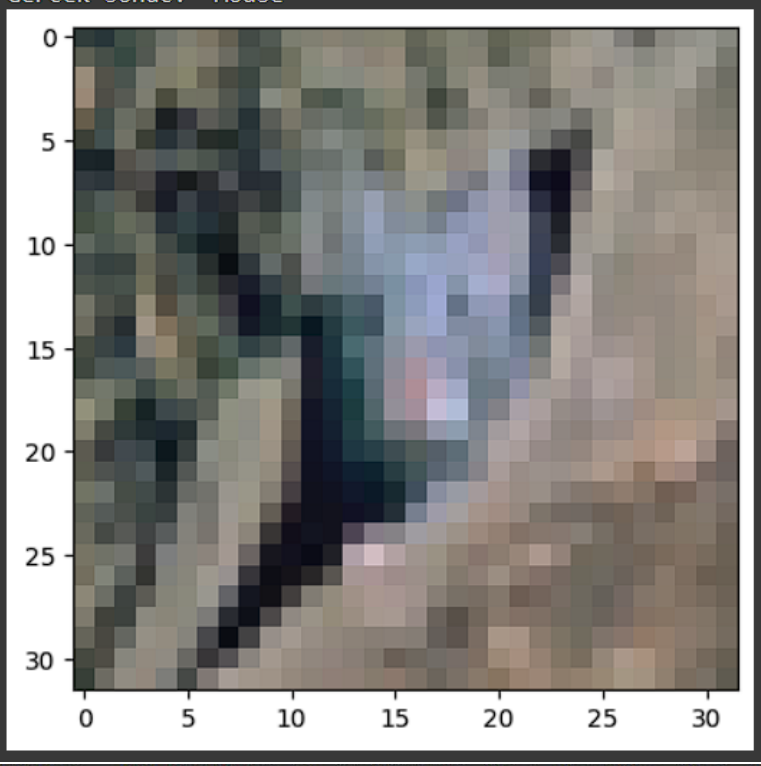
```
Vektor:  [[1.0261596e-05 8.5142934e-01 7.0804270e-04 3.6879668e-05 1.4777270e-01
  2.9934572e-06 3.9830098e-05]]
Tahmin sonucu:  Clock
Gercek sonuc:  Clock
```



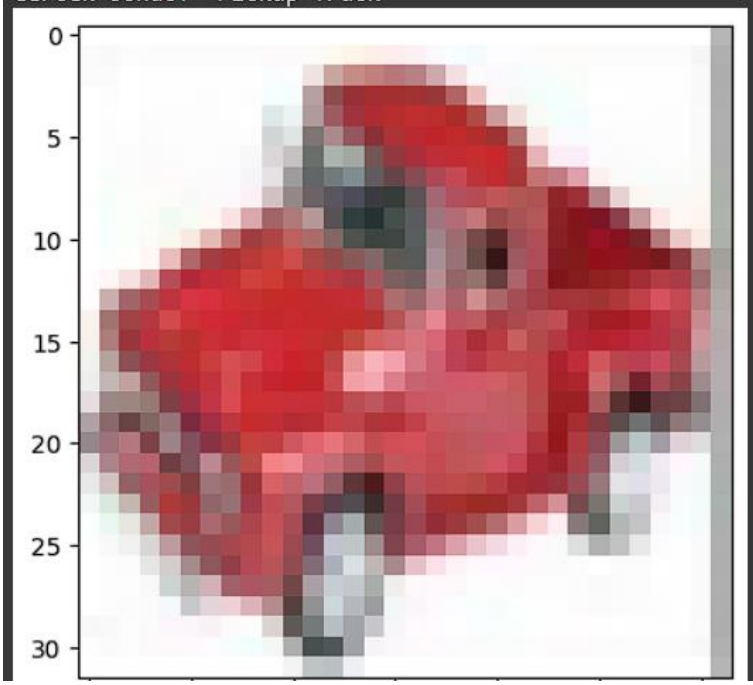

```
Vektor:  [[0.91693324 0.01057601 0.03122309 0.00498996 0.001155   0.00096455
  0.03415816]]
Tahmin sonucu:  Bear
Gercek sonuc:  Bear
```



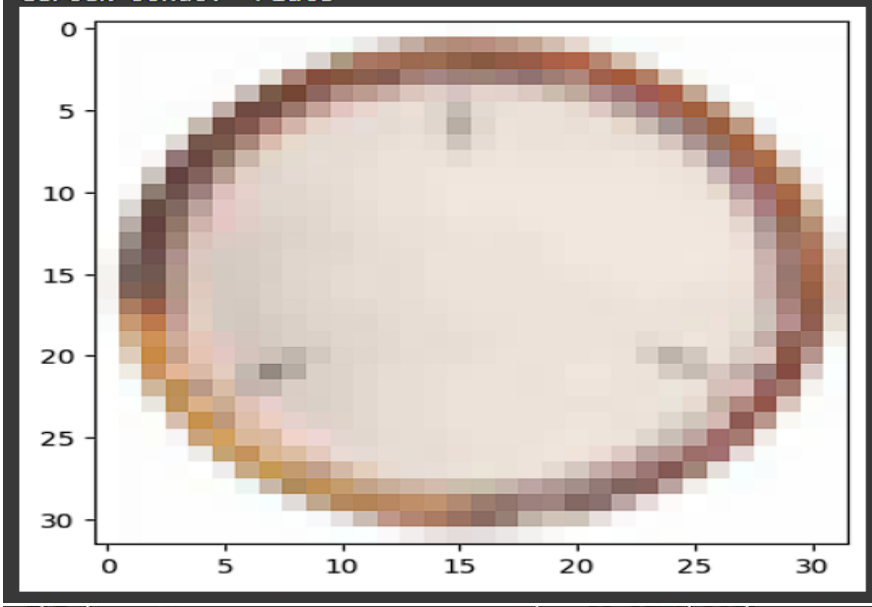
```
1/1 [==============================] - 0s 19ms/step
Vektor:  [[0.14043815 0.08516384 0.05403446 0.5049885  0.0278754  0.01110991
  0.17638974]]
Tahmin sonucu:  Pickup Truck
Gercek sonuc:  Mouse
```



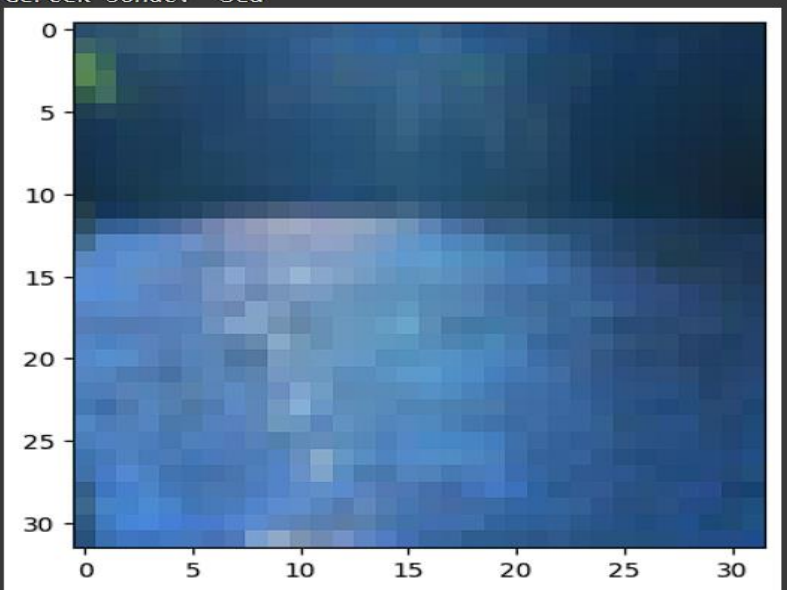

```
Vektor:  [[1.9234882e-05 8.3243139e-03 6.7555825e-06 9.8529053e-01 2.4228402e-04
  1.2558074e-06 6.1154654e-03]]
Tahmin sonucu:  Pickup Truck
Gercek sonuc:  Pickup Truck
```



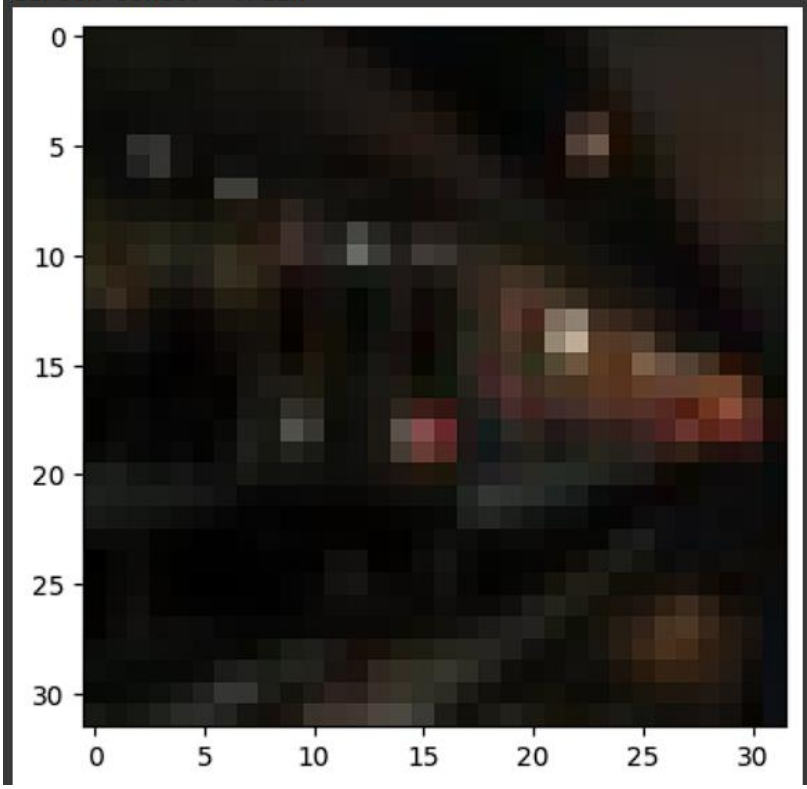
```
Vektor:  [[3.04047482e-11 1.51648875e-02 4.37750280e-09 6.84158930e-15
  9.84835148e-01 1.09809839e-13 1.49307053e-13]]
Tahmin sonucu:  Plate
Gercek sonuc:  Plate
```




```
Vektor:  [[1.7570563e-03 7.7384501e-04 4.5205551e-04 6.0251619e-05 6.8244158e-04
  9.9593687e-01 3.3758619e-04]]
Tahmin sonucu:  Sea
Gercek sonuc:  Sea
```



```
Vektor:  [[0.17398056 0.02756971 0.20546351 0.03810682 0.00407665 0.00464054
  0.5461622 ]]
Tahmin sonucu:  Train
Gercek sonuc:  Train
```



6.Kodlar

```python
# Library
import numpy as np
from keras import layers
from keras import models
from keras.utils import to_categorical
from keras.datasets import cifar100
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import tensorflow as tf

# Load Data
(train_images,train_labels),(test_images,test_labels) = cifar100.load_data()

# Categories
### 3=bear 22=clock 50=mouse 58=pickup_truck 61=plate 71=sea 90=train

selected_classes = [3, 22, 50, 58, 61, 71, 90]

train_images_filtered, train_labels_filtered = [], []
test_images_filtered, test_labels_filtered = [], []
```

```
for i in range(len(train_labels)):
    if train_labels[i][0] in selected_classes:
        train_images_filtered.append(train_images[i])
        train_labels_filtered.append(train_labels[i][0])

for i in range(len(test_labels)):
    if test_labels[i][0] in selected_classes:
        test_images_filtered.append(test_images[i])
        test_labels_filtered.append(test_labels[i][0])

train_images_filtered = np.array(train_images_filtered)
train_labels_filtered = np.array(train_labels_filtered)
test_images_filtered = np.array(test_images_filtered)
test_labels_filtered = np.array(test_labels_filtered)

print("Filtrelenmis egitim veri seti boyutu:", train_images_filtered.shape)
print("Filtrelenmis test veri seti boyutu:", test_images_filtered.shape)

# Visualization
categories = ["Bear","Clock","Mouse","Pickup_truck","Plate","Sea","Train"]
selected_classes = [3, 22, 50, 58, 61, 71, 90]
def show(classes, images, labels, categories, num_images=10):
    for i in range(len(classes)):
        category_images = [images[j] for j in range(len(labels)) if labels[j]
== classes[i]]


        fig,axes = plt.subplots(1, num_images, figsize=(7, 2))
        fig.text(0.01, 0.5, categories[i], ha='center', va='center',
fontsize=12)

        for k in range(min(num_images, len(category_images))): #secilen 10dan
az olabilir bu yuzden min alinir.
            axes[k].imshow(category_images[k])
            axes[k].axis('off')
        plt.show()

show(selected_classes, train_images_filtered, train_labels_filtered,
categories)

# Data Augmentation
datagen =tf.keras.preprocessing.image.ImageDataGenerator(
                                rotation_range=90,
                                brightness_range=[0.4,1.5],
                                width_shift_range=0.10,
                                height_shift_range=0.10,
                                shear_range=0.1,
                                zoom_range=0.1,
```

```
                                horizontal_flip=True,
                                vertical_flip=True,
                               fill_mode='nearest'
)

augmented_data =
datagen.flow(train_images_filtered,train_labels_filtered,batch_size=350)
train_images_augmented = train_images_filtered.copy()
train_labels_augmented = train_labels_filtered.copy()

for i in range(10):
  images, labels = next(augmented_data)
  train_images_augmented = np.concatenate([train_images_augmented, images])
  train_labels_augmented = np.concatenate([train_labels_augmented, labels])
train_images_augmented.shape

# Data Preprocessing

label_mapping = {3: 0, 22: 1, 50: 2, 58: 3, 61: 4, 71: 5, 90: 6}

train_labels_augmented = np.array([label_mapping[label] for label in
train_labels_augmented])
test_labels_filtered = np.array([label_mapping[label] for label in
test_labels_filtered])

train_images_augmented = train_images_augmented.reshape((7000,32,32,3))
train_images_augmented = train_images_augmented.astype('float32') /255

test_images_filtered = test_images_filtered.reshape((700,32,32,3))
test_images_filtered = test_images_filtered.astype('float32') /255



train_labels_augmented = to_categorical(train_labels_augmented)
test_labels_filtered = to_categorical(test_labels_filtered)


# Model
model = models.Sequential()

model.add(layers.Conv2D(
    64,
    (3,3),
    padding='same',
    input_shape=(32,32,3),
    activation='relu'
))
model.add(layers.Conv2D(
    64,
```

```
    (3,3),
    padding='same',
    activation='relu'
))

model.add(layers.MaxPool2D((2,2)))

model.add(layers.Conv2D(
    128,
    (3,3),
    padding='same',
    activation='relu'
))
model.add(layers.Conv2D(
    128,
    (3,3),
    padding='same',
    activation='relu'
))

model.add(layers.MaxPool2D((2,2)))



model.add(layers.Flatten())

model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dropout(0.5))

model.add(layers.Dense(7,activation='softmax'))

model.summary()

# Block Schema
import keras
keras.utils.plot_model(model,"model.png",show_shapes=True)

# Model Compile
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                              mode='min',
                                              patience = 3,
                                              verbose=1,
                                              restore_best_weights = True)

model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy','AUC','Recall']
```

```
)

# Model Fit

history = model.fit(train_images_augmented
                    ,train_labels_augmented
                    ,batch_size=256,
                    epochs=200,
                    validation_data=(test_images_filtered,test_labels_filtered
),
                    callbacks=[early_stop]
                )

model.save("model.h5")

# Loss Val Loss Graph

plt.figure(figsize=(15,8))
plt.plot(history.history['loss'],label ='Train Loss')
plt.plot(history.history['val_loss'],label ='Validation Loss')
plt.ylabel('Loss-categorical_crossentropy')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.title('Loss Graph')
plt.show()

# Accuracy Val Accuracy Graph
plt.figure(figsize=(15,8))
plt.plot(history.history['accuracy'],label='Train Accuracy')
plt.plot(history.history['val_accuracy'],label='Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper right')
plt.title('Accuracy Graph')
plt.show()

# Model Evaluate
test_loss,test_acc,test_auc,test_recall =
model.evaluate(test_images_filtered,test_labels_filtered)
print("Test loss: ",test_loss)
print("Test acc: ",test_acc)
print("Test auc: ",test_auc)
print("Test recall: ",test_recall)

# Confusion Matrix
predictions = model.predict(test_images_filtered)
predicted_classes = np.argmax(predictions, axis=1)

true_classes = np.argmax(test_labels_filtered, axis=1)
```

```
cm = confusion_matrix(true_classes, predicted_classes)

class_labels = ["Bear","Clock","Mouse","Pickup_truck","Plate","Sea","Train"]

fig, ax = plt.subplots(figsize=(10, 8))
im = ax.imshow(cm, interpolation='nearest', cmap='Blues')
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=class_labels, yticklabels=class_labels,
       title='Confusion Matrix',
       ylabel='Gercek etiket',
       xlabel='Tahmin edilen etiket')

plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
ax.set_xlim(-0.5, len(np.unique(class_labels))-0.5)
ax.set_ylim(len(np.unique(class_labels))-0.5, -0.5)

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], 'd'),
                ha="center", va="center",
                color="white" if cm[i, j] > cm.max() / 2. else "black")
plt.tight_layout()
plt.show()
# Model Test
map_list = {0:'Bear',1:'Clock',2:'Mouse',3:'Pickup
Truck',4:'Plate',5:'Sea',6:'Train'}
def testModel(test_images_filtered,test_labels_filtered):
    rand = np.random.randint(0, len(test_images_filtered) - 1)
    test = test_images_filtered[rand]
    plt.imshow(test)
    test = test.reshape(1,32,32,3)
    tahmin = model.predict(test)
    sonuc = np.argmax(tahmin)
    print("Vektor: ",tahmin)
    print("Tahmin sonucu: ",map_list[sonuc])

    gercek = test_labels_filtered[rand]
    gercek = np.argmax(gercek)
    print("Gercek sonuc: ",map_list[gercek])

testModel(test_images_filtered,test_labels_filtered)
```