GÖRÜNTÜ İŞLEME

PROJE

AYŞE HİLAL DOĞAN

17011907

Konu : Konvolüsyonel Sinir Ağı Tasarımı Tabanlı Sınıflandırma ve Öğrenme Aktarımı Tabanlı Görüntü Erişimi Uygulaması

Konu Açıklama : Bu ödevde CINIC-10 verisetin üzerinde konvolüsyonel ağ tasarımı yapılacaktır. Bunun yanı sıra daha önce Imagenet verisetinde eğitilmiş VGG-16 ve Resnet-50 modellerinin öğrenme aktarımı(transfer learning) ile ilgili CINIC-10 ile yeniden sınıflandırılması ve bu modelden elde edilen öznitelik vektörü (feature vector) ile bir resme en çok benzeyen 5 adet resmi bulan bir sistem tasarlanacak ve gerçeklenecektir.

a. YÖNTEM

1. Orijinal bir konvolüsyonel ağının tasarlanması, hiper-parametrelerin denenmesi ve sonuçlarının raporlanması

- CINIC-10 veriseti üzerinde konvolüsyonel ağ tasarımı yapılacağı için kaggle app'ini kullanarak yukarıdaki kod bloğunda verildiği şekilde CINIC-10 veriseti yüklenir.

```
! pip install -q kaggle
```

```
from google.colab import files
files.upload()
```

```
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download -d mengcius/cinic10
```

- Yüklenen verisetindeki veriler train, valid, test şeklindedir. Her bir dosyanın içinde 10 sınıfa ait resimler bulunmaktadır. Eğitimde kullanmak üzere train_generator ve val_generator, testte kullanmak üzere test_generator yukarıda verildiği şekilde tanımlanmıştır. Resimlerin boyutları 224x224x3, batch_size 64 olarak verilmiştir. Datayı alırken resimleri rescale fonksiyonuyla normalize ederek 0-1 aralığına getiririz.

```
batch_size = 64

datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255)

train_generator = datagen.flow_from_directory(
    directory='/content/train',
    target_size=(224,224),
    class_mode='categorical',
    batch_size=batch_size,
    )

val_generator = datagen.flow_from_directory(
    directory='/content/valid',
    target_size=(224,224),
    batch_size=batch_size,
    class_mode='categorical',
    )

test_generator = datagen.flow_from_directory(
    directory='/content/test',
    target_size=(224,224),
    batch_size=batch_size,
    class_mode='categorical',
    )
```

- CINIC-10 veriseti üzerinde 6 farklı konvolüsyonel ağ tasarımı yapılması istenmiştir. Output 10, batch_size 64, filtre sayısı 32, filtre büyüklükleri 3x3 veya 5x5, dropout (0.2) veya (0.7) olarak verilmiştir. 6 farklı ağ tasarımının 3 tanesi 2 katmanlı, 3 tanesi 3 katmanlıdır. Optimizasyon olarak "Adam" optimizasyon algoritması kullanılmıştır. Katmanların başlangıç ağırlıkları GlorotNormal olarak verilmiştir.

  Aşağıda 2 katmanlı, filtre büyüklüğü 3x3, dropout 0.7 olan konvolüsyonel ağ tasarımının kodlaması verilmiştir. 1 katmanda Conv2D, MaxPooling2D ve genelleştirme performansını arttırmak için Dropout bulunmaktadır.

```
model = models.Sequential()

model.add(layers.Conv2D(32, (3,3), activation='relu',kernel_initializer='GlorotNormal', input_shape=(224,224,3)))
model.add(layers.MaxPooling2D((1,1)))
model.add(layers.Dropout(0.7))

model.add(layers.Conv2D(32, (3,3), activation='relu',kernel_initializer='GlorotNormal'))
model.add(layers.MaxPooling2D((1,1)))
model.add(layers.Dropout(0.7))


model.add(layers.Flatten())
model.add(layers.Dense(batch_size, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

- Modelin eğitilmesi için model.fit() fonksiyonundan yararlanılır. 10 epoch kadar eğitilir. Daha sonra model kaydedilir.

```
model.fit(train_generator, validation_data = val_generator, epochs=10)
```

```
model.summary()
```

```
model.save("33_07.h5")
```

- Eğitilen modelin eğitim ve test aşamalarındaki loss ve accuracy değerleri hesaplanır ve ekrana yazdırılır.

```
loss,acc = model.evaluate(train_generator,verbose=2)
print("Train Loss:", loss)
print("Train Accuracy:", acc)
```

```
loss,acc = model.evaluate(test_generator,verbose=2)
print("Test Loss:", loss)
print("Test Accuracy:", acc)
```

- Modelin sınıflandırma sonuçlarının değerlendirilmesi aşamasında hata olduğunda hangi sınıfın hangi sınıfla karıştığının görülebilmesi için karışıklık matrisi hesaplanır ve ekrana yazdırılır.

```
prediction = model.predict(train_generator)
#prediction[:5]
```

```
print("Confusion Matrix:")
classes = [np.argmax(element) for element in prediction]
classes[:10]
```

2. Öğrenim Aktarımının Gerçeklenmesi ve Raporlanması

Bu aşamada VGG-16 ve Resnet-50 modellerinin Imagenet veriseti üzerinde eğitilmiş ağırlık dosyalarına ulaşarak, iki farklı konvolüsyonel sinir ağı (VGG-16, Resnet-50) için iki farklı öğrenme aktarımı yapmamız istenmiştir.

- Burada istenen senaryoya bağlı olarak VGG16 veya Resnet50 modelini base_model olarak yükleyerek untrainable olarak ayarlıyoruz.

VGG16:

```
base_model = keras.applications.VGG16(weights='imagenet',include_top=True)
```

```
#Do the model untrainable
base_model.trainable = False
```

Resnet50:

```
base_model = keras.applications.ResNet50(weights='imagenet',include_top=True)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
102973440/102967424 [==============================] - 0s 0us/step
```

```
#Do the model untrainable
base_model.trainable = False
```

- Modelin son fully connected katmanını silerek, yerine 1024 neuron'luk bir fully connected katman eklememiz istenmiştir. Aşağıda verilen kod bloğunda son katmanı çıkarır ve yerine Dense(1024) eklenir. Çıktı olarak 10 sınfımız olduğu için en sona Dense(10) eklenir.

```
base_inputs = base_model.layers[0].input
base_outputs = base_model.layers[-2].output
final_outputs = layers.Dense(1024,activation='relu')(base_outputs)
final_outputs = layers.Dense(10,activation='softmax')(final_outputs)

new_model = keras.Model(inputs=base_inputs, outputs = final_outputs)

new_model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

print(new_model.summary())
```

- Aşağıda VGG16 için modelin ilk durumu ve son fully connected katmanın silinip, yerine 1024 neuronluk Dense eklendikten sonraki durumlarının görüntüsü verilmiştir.

```
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 0
Non-trainable params: 138,357,544
```

-> >>>

```
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              4195328
_________________________________________________________________
dense_5 (Dense)              (None, 10)                10250
=================================================================
Total params: 138,466,122
Trainable params: 4,205,578
Non-trainable params: 134,260,544
```

- İki farklı model için 2 farklı sinir ağı istenmiştir. Bunlardan birinde sadece eklediğimiz katman eğitilebilir olup, diğerleri eğitilemez olmalıdır. Diğerinde ise eklediğimiz katmanla beraber 4 katman eğitilebilir, diğerleri eğitilemez olmalıdır. Eğer 4 katmanı eğitilebilir istiyorsak aşağıdaki kod bloğu çalışır,son 3 katman hariç diğer katmanlar dondurulur. Diğer türlü sadece eklediğimiz katman eğitilir. Gerisi eğitilemez.

```
for layer in base_model.layers[:-3]:
  layer.trainable = False
```

- Modeli eğitmek için aşağıdaki kod bloğu kullanılır.

```
new_model.fit(train_generator, validation_data = val_generator, batch_size=64, epochs=10)
```

- Eğitim ve test sonuçları aşağıdaki kod bloğunda bulunur ve ekrana yazdırılır.

```
#train loss,accuracy

loss,acc = model.evaluate(train_generator,verbose=2)
print("Train Loss:", loss)
print("Train Accuracy:", acc)
```

```
#test loss,accuracy

loss,acc = model.evaluate(test_generator,verbose=2)
print("Test Loss:", loss)
print("Test Accuracy:", acc)
```

- Her bir senaryo için ayrı ayrı karmaşıklık matrisi hesaplanır ve ekrana yazdırılır.

```
#Confusion Matrix
prediction = new_model.predict(train_generator)
```

```
print("Confusion Matrix:")
classes = [np.argmax(element) for element in prediction]
classes[:10]
```

```
Confusion Matrix:
[7, 8, 4, 6, 5, 1, 7, 8, 5, 9]
```

- Daha sonra da kullanabilmek için model kaydedilir.

```
model.save("vgg16_2.h5")
```

3- Öğrenim Aktarımı Modelinin Özniteliklerini Kullanarak Görüntü Erişiminin Gerçeklenmesi ve Raporlanması

Bu kısımda 2. Kısımda eğitimi tamamlanan farklı öğrenme aktarımlarına sahip modelleri kullanarak veri kümesindeki veriler için öznitelik çıkarımı yaparak, kosinüs benzerliğini kullanarak random seçilen resimlere benzer 5 er resim bulmamız istenmiştir.

- Aşağıda kosinüs benzerliğini kullanarak iki vektör arasındaki benzerliği döndüren bir fonksiyonun kod bloğu bulunmaktadır.

```
#Cosine Similarity Function

import math
from math import sqrt

def cosine(a,b):
    sum = 0
    suma = 0
    sumb = 0
    for i,j in zip(a, b):
        suma += i * i
        sumb += j*j
        sum += i*j
    cosine_sim = sum / ((sqrt(suma))*(sqrt(sumb)))
    return cosine_sim
```

- Eğitim yapıldıktan sonra modelin son katmanının çıktısı kullanılarak öznitelik çıkarımı yapılır.

```
feature_extractor = keras.Model(
    inputs=new_model.inputs,
    outputs=new_model.get_layer(name="Dense_4").output,
)
```

- Random bir resim seçmeye yarar:

```
import random

#take random image
id = int(len(images) * random.random())

#display image
img = image.load_img(images[id])
plt.imshow(img)
```

- Yolu verilen bir resmi ekranda göstermeye yarar:

```
img = image.load_img(path , target_size=model.input_shape[1:3])

x = image.img_to_array(img)
x = np.expand_dims(x,axis=0)
x = preprocess_input(x)

plt.imshow(img)
```

b. UYGULAMA

Bu bölümde yapılan eğitimlerin başarı sonuçları, karmaşıklık sonuçları verilerek karşılaştırma yapılacaktır.

Eğitimler :

1- 2 katman , filtre büyüklüğü 3x3, dropout 0.2 :

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 222, 222, 32)      0
_________________________________________________________________
dropout_3 (Dropout)          (None, 222, 222, 32)      0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 220, 220, 32)      9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 220, 220, 32)      0
_________________________________________________________________
dropout_4 (Dropout)          (None, 220, 220, 32)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1548800)           0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                99123264
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 99,134,058
Trainable params: 99,134,058
Non-trainable params: 0
_________________________________________________________________
```

```
1407/1407 - 93s - loss: 0.2073 - accuracy: 0.9334
Train Loss: 0.2072758823633194
Train Accuracy: 0.9334333539009094
```

```
1407/1407 - 94s - loss: 2.9354 - accuracy: 0.4568
Test Loss: 2.9353833198547363
Test Accuracy: 0.4568111002445221

        Confusion Matrix:
        [3, 8, 4, 8, 5, 1, 0, 5, 0, 5]
```

2- 2 katman , filtre büyüklüğü 3x3, dropout 0.7 :

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 222, 222, 32)      0
_________________________________________________________________
dropout_5 (Dropout)          (None, 222, 222, 32)      0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 220, 220, 32)      9248
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 220, 220, 32)      0
_________________________________________________________________
dropout_6 (Dropout)          (None, 220, 220, 32)      0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1548800)           0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                99123264
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 99,134,058
Trainable params: 99,134,058
Non-trainable params: 0
_________________________________________________________________
```

```
1407/1407 - 99s - loss: 1.9174 - accuracy: 0.1949
Train Loss: 1.91744863986969
Train Accuracy: 0.1948888897895813
```

```
1407/1407 - 136s - loss: 1.9550 - accuracy: 0.1919
Test Loss: 1.9550411701202393
Test Accuracy: 0.19193333387374878

        Confusion Matrix:
        [2, 8, 9, 8, 8, 2, 9, 8, 2, 1]
```

3- 2 katman , filtre büyüklüğü 5x5, dropout 0.2 :

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 220, 220, 32)      2432
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 220, 220, 32)      0
_________________________________________________________________
dropout_7 (Dropout)          (None, 220, 220, 32)      0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 216, 216, 32)      25632
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 216, 216, 32)      0
_________________________________________________________________
dropout_8 (Dropout)          (None, 216, 216, 32)      0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1492992)           0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                95551552
_________________________________________________________________
dense_7 (Dense)              (None, 10)                650
=================================================================
Total params: 95,580,266
Trainable params: 95,580,266
Non-trainable params: 0
_________________________________________________________________
```

```
1407/1407 - 103s - loss: 2.3026 - accuracy: 0.1000
Train Loss: 2.3026061058044434
Train Accuracy: 0.10000000149011612
```

```
1407/1407 - 103s - loss: 2.3026 - accuracy: 0.1000
Test Loss: 2.3026044368743896
Test Accuracy: 0.10000000149011612

Confusion Matrix:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

4- 3 katman , filtre büyüklüğü 3x3, dropout 0.2 :

```
1407/1407 - 99s - loss: 0.1805 - accuracy: 0.9446
Train Loss: 0.18048711121082306
Train Accuracy: 0.9445555806159973
```

```
1407/1407 - 101s - loss: 2.7232 - accuracy: 0.4518
Test Loss: 2.72318696975708
Test Accuracy: 0.451755553483963
```

Confusion matrix:

```
[6, 1, 0, 9, 7, 5, 2, 3, 3, 9]
```

5- 3 katman , filtre büyüklüğü 3x3, dropout 0.7 :

```
1407/1407 - 106s - loss: 1.5279 - accuracy: 0.4052
Train Loss: 1.5279173851013184
Train Accuracy: 0.40521112084388733
```

```
1407/1407 - 105s - loss: 1.5946 - accuracy: 0.3875
Test Loss: 1.5945558547973633
Test Accuracy: 0.38751110434532166

Confusion Matrix:
[6, 9, 1, 2, 6, 5, 8, 8, 3, 8]
```

6- 3 katman , filtre büyüklüğü 5x5, dropout 0.2 :

```
1407/1407 - 116s - loss: 2.3027 - accuracy: 0.1000
Train Loss: 2.302659273147583
Train Accuracy: 0.10000000149011612
```

```
1407/1407 - 116s - loss: 2.3027 - accuracy: 0.1000
Test Loss: 2.302659511566162
Test Accuracy: 0.10000000149011612

Confusion Matrix:
[9, 9, 9, 9, 9, 9, 9, 9, 9, 9]
```

Karmaşıklık matrisleri incelendiğinde, genel olarak dropout 0.7 olanların 0.2 olanlardan daha başarılı olduğunu, katman arttıkça başarının arttığı gözlemlenmiştir.

Transfer eğitimleri:

- **VGG16 – 1 :** Sadece son katman eğitilebilir

```
Confusion Matrix:
[7, 8, 4, 6, 5, 1, 7, 8, 5, 9]
```

- **VGG16 – 2 :** Son 4 katman eğitilebilir

```
1407/1407 - 97s - loss: 2.3176 - accuracy: 0.1024
Train Loss: 2.3176403045654297
Train Accuracy: 0.10243333131074905
```

```
1407/1407 - 99s - loss: 2.3176 - accuracy: 0.1030
Test Loss: 2.3175888061523438
Test Accuracy: 0.1030111089348793
```

```
Confusion Matrix:
[0, 6, 2, 4, 9, 7, 4, 7, 1, 7]
```

- **Resnet50 – 1 :** Sadece son katman eğitilebilir

```
1407/1407 - 101s - loss: 2.3026 - accuracy: 0.1000
Train Loss: 2.302603006362915
Train Accuracy: 0.10000000149011612
```

```
1407/1407 - 101s - loss: 2.3026 - accuracy: 0.1000
Test Loss: 2.3026068210601807
Test Accuracy: 0.10000000149011612
```

```
Confusion Matrix:
[6, 8, 1, 6, 7, 1, 6, 9, 6, 1]
```

- **Resnet50 – 2 :** Son 4 katman eğitilebilir

Vgg16 ve resnet50 nin karmaşıklık matrisleri ve başarı sonuçları incelendiğinde, VGG16 nın Resnet50 ye göre daha başarılı olduğu, sadece son katmanın eğitilebilir olması durumunda başarının daha yüksek olduğu görülmüştür.

Yapılan tüm eğitimlerin sonucunda ise, transfer eğitimi yapılması durumunda başarı oranının daha yüksek olduğu anlaşılmıştır.

c. SONUÇ

Yapılan konvolüsyonel sinir ağı tasarımı tabanlı sınıflandırma ve öğrenme aktarımı tabanlı görüntü erişimi uygulamasında, daha önce böyle bir çalışmada bulunmadığım için çok fazla kazanımım olmuştur. Öncelikle makine öğrenmesinin mantığı anlaşılmış ve temel olmuştur. Sinir ağının tasarımının nasıl yapıldığı, öğrenme aktarımının faydaları, sınıflandırma yaparken

sinir ağı tabanlı bir sınıflandırmanın artıları deneyimlenerek ve çalışmaların sonuçları gözlemlenerek uygulamalı olarak öğrenilmiştir.

Konvolüsyonel sinir ağı tasarımında, katman sayısı nekadar çok olursa okadar başarılı sonuçlar alındığı, genelleştirme performansını arttırmaya yarayan dropout katmanının arttırılmasının model üzerindeki olumlu etkisi, piksel değerlerinin normalleştirilmesinin doğru sonuçlar verme üzerindeki etkisi, karmaşıklık matrisinin nasıl oluştuğu ve sonuçları değerlendirmedeki yararı gözlemlenmiştir.

Yapılan çalışma sonucunda, aynı veri kümesinde farklı sinir ağları tasarlanmış ve öğrenme aktarımı yaparak modeller tasarlanmıştır. Bunların arasındaki fark uygulama bölümünde anlatılmıştır. Öğrenme aktarımının kullanılmasının olumlu yönleri; daha önceden eğitildiği için daha başarılı sonuçlar vermesi, olumsuz yönü ise daha yavaş işlemesi olarak gözlemlenmiştir.