Convolutional Neural Networks

Welcome to HomeWork 3 of the course! In this notebook, you will:

- Implement a CNN model from the scratch for image classification.
- Use a pretrained model and Transfer learning to solve above classification problem.

As usual, we will start by loading the packages.

به نام خدا

در این تمرین می‌خواهیم یک شبکه دسته‌بندی را در دو حالت بررسی کنیم. در حالت اول با استفاده از یک شبکه به صورت آموزش از ابتدا این کار را انجام می‌دهیم. و در حالت دیگر با استفاده از یک شبکه از پیش آموزش داده شده به صورت انتقال یادگیری شبکه این کار را انجام می‌دهیم و نتایج را مقایسه می‌کنیم. در ابتدا ماژول‌های مورد نیاز را به برنامه اضافه می‌کنیم.

In [1]:
```
import numpy as np
import h5py
import matplotlib.pyplot as plt
## Feel free to add pakages depending of your selected frame work.
%matplotlib inline
np.random.seed(1)
import keras
from keras.layers import Flatten, Dense, Dropout, Convolution2D, MaxPool2D
from keras.models import Sequential, Model
from keras.optimizers import sgd, rmsprop
from keras.regularizers import L1L2
import skimage
from skimage import exposure
import sklearn
from sklearn.metrics import confusion_matrix, fbeta_score
```

```
Using TensorFlow backend.
```

حال پایگاه داده را باز می‌کنیم.

In [2]:
```
def load_dataset():
    train_dataset = h5py.File('datasets/train.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your trai
n set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your trai
n set labels

    test_dataset = h5py.File('datasets/test.h5', "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test se
t features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test se
t labels

    classes = np.array(test_dataset["list_classes"][:]) # the list of classe
s

    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape
[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_o
rig, classes
```

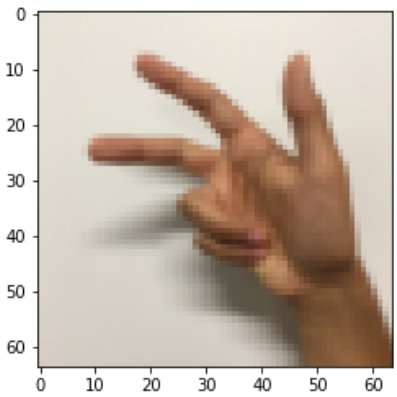
Run the next cell to load the dataset you are going to use.

In [3]:
```
# Loading the data (signs)
X_train_orig, Y_train_orig, X_test_orig, Y_test_orig, classes = load_dataset
()
```

The next cell will show you an example of a labelled image in the dataset. Feel free to change the value of `index` below and re-run to see different examples.

In [4]:
```
# Example of a picture
index = 100
plt.imshow(X_train_orig[index])
print ("y = " + str(np.squeeze(Y_train_orig[:, index])))
```

```
y = 3
```



From now implemnt your model. After that use transfer learning. Please add cells and explain yours developing steps and your results.

استفاده از زبان فارسی برای توضیحات هم مجاز است

موفق باشید

In [5]:
```
X_train_orig.max()
```

Out[5]:
```
244
```

با توجه به این که تصاویر پایگاه داده ساختار ساده‌ای دارند بنابراین یک شبکه ساده مانند LeNet می‌تواند به راحتی روی این داده‌ها آموزش ببیند. بنابراین برای دسته‌بندی از ابتدا، از معماری مشابه همان شبکه LeNet استفاده می‌کنیم:

In [51]:
```
model = Sequential()
model.add(Convolution2D(16, (5,5), input_shape=(64,64,3), padding='same', ac
tivation='relu'))
model.add(MaxPool2D(2,2))
model.add(Convolution2D(32, (5,5), padding='same', activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Convolution2D(64, (5,5), padding='same', activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Convolution2D(64, (5,5), padding='same', activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Convolution2D(64, (5,5), padding='same', activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Convolution2D(128, (5,5), padding='same', activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(6, activation='softmax'))
```

همانطور که دیده می‌شود، مدل شبکه بسیار ساده است و پارامترهای کمی‌دارد.

In [52]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_62 (Conv2D)           (None, 64, 64, 16)        1216
_________________________________________________________________
max_pooling2d_60 (MaxPooling (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 32, 32, 32)        12832
_________________________________________________________________
max_pooling2d_61 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 16, 16, 64)        51264
_________________________________________________________________
max_pooling2d_62 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 8, 8, 64)          102464
_________________________________________________________________
max_pooling2d_63 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 4, 4, 64)          102464
_________________________________________________________________
max_pooling2d_64 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
conv2d_67 (Conv2D)           (None, 2, 2, 128)         204928
_________________________________________________________________
max_pooling2d_65 (MaxPooling (None, 1, 1, 128)         0
_________________________________________________________________
flatten_8 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_14 (Dense)             (None, 6)                 774
=================================================================
Total params: 492,454
Trainable params: 492,454
Non-trainable params: 0
_________________________________________________________________
```

حال برای آموزش مدل، نیاز است که یک روش بهینه سازی انتخاب می‌کنیم، و البته برای این که مطمئن شویم که شبکه روی داده‌ها overfit نمی‌کند، روی وزن‌های شبکه یک تنظیم کننده کنار تابع loss قرار می‌دهیم. ولی از آنجا که شبکه بسیار سبک است، بنابراین نیازی به انتخاب مقدار بزرگی برای آن نداریم.

In [53]:
```
op = rmsprop(decay=1e-6)
model.compile(optimizer=op, loss='categorical_crossentropy', metrics=['acc
'])
```

In [23]:
```
Y_train = keras.utils.np_utils.to_categorical(Y_train_orig[0,:])
Y_test = keras.utils.np_utils.to_categorical(Y_test_orig[0,:])
```

In [54]:
```
model.fit(X_train_orig/255.0, Y_train, batch_size=32, epochs=32, shuffle=Tru
e, validation_data=(X_test_orig/255.0, Y_test))
```

```
Train on 1080 samples, validate on 120 samples
Epoch 1/32
1080/1080 [==============================] - 23s 21ms/step - loss: 1.8133 - a
cc: 0.1463 - val_loss: 1.7912 - val_acc: 0.2417
Epoch 2/32
1080/1080 [==============================] - 20s 18ms/step - loss: 1.7932 - a
cc: 0.1509 - val_loss: 1.7913 - val_acc: 0.1667
Epoch 3/32
1080/1080 [==============================] - 21s 19ms/step - loss: 1.7971 - a
cc: 0.1620 - val_loss: 1.7906 - val_acc: 0.1667
Epoch 4/32
1080/1080 [==============================] - 20s 19ms/step - loss: 1.7872 - a
cc: 0.1657 - val_loss: 1.7251 - val_acc: 0.2583
Epoch 5/32
1080/1080 [==============================] - 19s 18ms/step - loss: 1.6622 - a
cc: 0.3065 - val_loss: 1.5081 - val_acc: 0.4167
Epoch 6/32
1080/1080 [==============================] - 21s 19ms/step - loss: 1.3816 - a
cc: 0.4389 - val_loss: 1.0975 - val_acc: 0.5167
Epoch 7/32
1080/1080 [==============================] - 21s 19ms/step - loss: 1.0330 - a
cc: 0.5917 - val_loss: 0.9516 - val_acc: 0.6833
Epoch 8/32
1080/1080 [==============================] - 20s 19ms/step - loss: 0.7737 - a
cc: 0.6981 - val_loss: 0.6176 - val_acc: 0.7833
Epoch 9/32
1080/1080 [==============================] - 21s 20ms/step - loss: 0.5773 - a
cc: 0.7944 - val_loss: 0.9194 - val_acc: 0.6500
Epoch 10/32
1080/1080 [==============================] - 20s 19ms/step - loss: 0.4863 - a
cc: 0.8167 - val_loss: 0.3542 - val_acc: 0.8750
Epoch 11/32
1080/1080 [==============================] - 20s 19ms/step - loss: 0.3390 - a
cc: 0.8778 - val_loss: 0.3464 - val_acc: 0.8583
Epoch 12/32
1080/1080 [==============================] - 19s 17ms/step - loss: 0.2630 - a
cc: 0.9093 - val_loss: 0.5541 - val_acc: 0.8333
Epoch 13/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.2119 - a
cc: 0.9204 - val_loss: 0.6110 - val_acc: 0.8000
Epoch 14/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.1410 - a
cc: 0.9528 - val_loss: 0.2403 - val_acc: 0.9333
Epoch 15/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.3033 - a
cc: 0.9167 - val_loss: 0.1902 - val_acc: 0.9250
Epoch 16/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.0831 - a
cc: 0.9750 - val_loss: 0.4874 - val_acc: 0.8917
Epoch 17/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.1316 - a
cc: 0.9620 - val_loss: 0.1528 - val_acc: 0.9333
Epoch 18/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.0957 - a
cc: 0.9713 - val_loss: 0.4117 - val_acc: 0.9000
Epoch 19/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.0589 - a
cc: 0.9824 - val_loss: 0.2485 - val_acc: 0.9500
Epoch 20/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.1959 - a
cc: 0.9639 - val_loss: 0.3408 - val_acc: 0.9250
Epoch 21/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.1197 - a
cc: 0.9713 - val_loss: 0.2544 - val_acc: 0.8833
Epoch 22/32
1080/1080 [==============================] - 19s 18ms/step - loss: 0.0219 - a
cc: 0.9944 - val_loss: 0.3725 - val_acc: 0.9417
Epoch 23/32
```

Out[54]: <keras.callbacks.History at 0x7f30b38b8c18>

پس از آموزش در ۳۲ دوره زمانی شبکه به دقت کافی می‌رسد. از مقادیر تابع loss برای هر دو داده آموزشی و ارزیابی مشخص است که شبکه روی داده‌ها overfit نشده است.

در نهایت برای این که روی دسته بندی شبکه مطمئن شویم، ماتریس confusion دسته بندی را نیز بررسی می‌کنیم. این ماتریس به دقت ۹۷/۵ درصد رسده است.

In [55]:
```
Y = model.predict(X_test_orig/255.0,verbose=1)
confusion_matrix(y_pred=Y.argmax(1), y_true=Y_test.argmax(1))
```

```
120/120 [==============================] - 1s 9ms/step
```

Out[55]:
```
array([[20,  0,  0,  0,  0,  0],
       [ 0, 20,  0,  0,  0,  0],
       [ 0,  1, 19,  0,  0,  0],
       [ 0,  0,  0, 20,  0,  0],
       [ 0,  0,  1,  0, 18,  1],
       [ 0,  0,  0,  0,  0, 20]])
```

مسأله ۲:

در بخش دوم این تمرین می‌خواهیم یک شبکه از پیش آموزش داده شده را روی داده‌های خود آموزش دهیم. در انجام این مسأله چند نکته باید در نظر گرفته شود، نخست انتخاب یک مدل مناسب. دوم نحوه آموزش است.

شبکه‌هایی مانند VGG16 و ResNet و... شبکه‌های بسیار بزرگ و با پارامتر زیادی هستند که هم آموزش آن‌ها روی یک رایانه خانگی سخت است و علاوه بر آن با توجه به این که داده‌ها بسیار ساده هستند واقعا نیازی به این حجم از پارامتر برای یک شبکه نداریم.

از طرفی آموزش شبکه در دو مرحله انجام می‌شود، مرحله اول را که معمولا fine-tunning می‌نامند به این صورت است که ما یک شبکه را انتخاب می‌کنیم و بخش دسته بند آن یعنی همان شبکه تمام متصل را از نو و متناسب با کاربرد خود طراحی می‌کنیم. در مرحله اول آموزش فقط شبکه تمام متصل را آموزش می‌دهیم، در این مرحله برای جلوگیری از overfit حتما نیاز است که از یک تنظیم کننده در لایه‌ها استفاده نماییم.

سپس در مرحله دوم که در اصطلاح به آن transfer-learning می‌گویند تمام شبکه را با یک نرخ یادگیری پایین آموزش می‌دهیم.

شبکه‌ای که انتخاب کردیم، معماری MobileNet_v2 است که در کتابخانه Keras به صورت از پیش آموزش دیده شده تعبیه شده است.

اما مشکلی که Keras در تولید شبکه دارد این است که به ما اجازه نمی‌دهد که شبکه‌ای که از پیش آموزش داده شده است را با ورودی به اندازه دلخواه طراحی کنیم. بنابراین ما در اینجا دو مدل طراحی کردیم و مدل اول به صورت پیشفرض خود Keras است ولی همراه با وزن‌ها، و شبکه دوم با اندازه ورودی مورد نظر ما ولی بدون وزن‌های آموزش داده شده.

از آنجا که وزن‌های بخش استخراج ویژگی دو شبکه ارتباطی به اندازه ورودی شبکه ندارد، بنابراین وزن‌های مدل اول را به مدل دوم می‌دهیم و سپس دسته بند یعنی شبکه تمام متصل را به مدل دوم اضافه می‌کنیم.

In [7]:
```
model = keras.applications.mobilenet_v2.MobileNetV2(input_shape=(128, 128,
3), alpha=1.0, depth_multiplier=1, include_top=False, weights='imagenet', in
put_tensor=None, pooling=None, classes=6)
```

In [8]:
```
model2 = keras.applications.mobilenet_v2.MobileNetV2(input_shape=(64, 64,
3), alpha=1.0, depth_multiplier=1, include_top=False, weights=None, input_te
nsor=None, pooling=None, classes=6)
```

In [36]:
```
W = model.get_weights()
model2.set_weights(W)
```

In [23]:
```
Y_train = keras.utils.np_utils.to_categorical(Y_train_orig[0,:])
Y_test = keras.utils.np_utils.to_categorical(Y_test_orig[0,:])
```

در اینجا می‌خواهیم به شبکه یک لایه تمام متصل اضافه کنیم. با توجه به روش کد نویسی در Keras به صورت زیر عمل می‌کنیم:

In [38]:
```
detector = model2.output
```

In [39]:
```
x = Flatten()(detector)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.4)(x)
# and a logistic layer -- let's say we have 6 classes
predictions = Dense(6, activation='softmax')(x)

# this is the model we will train
classifier = Model(inputs=model2.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional MobileNetV2 layers
for layer in model2.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
op = sgd(decay=1e-3)
classifier.compile(optimizer=op, loss='categorical_crossentropy', metrics=['
acc'])
```

حال شبکه را در مرحله اول آموزش می‌دهیم. در مرحله اول فقط بخش تمام متصل شبکه را آموزش می‌دهیم، برای این منظور در Keras برای لایه‌های کانولوشن شبکه پارامتر trainable را به صورت False مقدار دهی می‌کنیم.

در این مرحله از آموزش تنظیم کننده را در کنار تابع loss تعریف می‌کنیم و در این صورت مقدار اولیه آن را برابر 1e-3 قرار می‌دهیم. سپس در شبکه را آموزش می‌دهیم. در این جا صرفا به دو مرحله اپوک زمانی اکتفا شده است.

In [45]:
```
classifier.fit(X_train_orig/255.0, Y_train, batch_size=32, epochs=2, shuffl
e=True, validation_data=(X_test_orig/255.0, Y_test))
```

```
Train on 1080 samples, validate on 120 samples
Epoch 1/2
1080/1080 [==============================] - 60s 55ms/step - loss: 0.7762 - v
al_loss: 0.5541
Epoch 2/2
1080/1080 [==============================] - 45s 42ms/step - loss: 0.2526 - v
al_loss: 0.4070
```

Out[45]:
```
<keras.callbacks.History at 0x7f79ab0da240>
```

سپس با استفاده از بخش کانولوشنی شبکه را هم قابل آموزش می‌کنیم. در این صورت با روش آموزش SGD سعی می‌کنیم شبکه را به صورت Transfer-learning آموزش می‌دهیم. اما نکته‌ای که باید در اینجا توجه کنیم این است که تنظیم کننده در نظر گرفته شده مقدار خیلی زیادی دارد. مقدار زیاد تنظیم کننده هر چند که جلوی overfit شدن شبکه را می‌گیرد، اما مقدار زیاد آن جلوی بهینه شدن جواب را هم می‌گیرد، به این منظور به صورت مرحله‌ای مقدار آن را کاهش می‌دهیم و سپس آموزش شبکه را ادامه می‌دهیم.

In [44]:
```
for layer in model2.layers:
    layer.trainable = True

# compile the model (should be done *after* setting layers to trainable)
classifier.compile(optimizer='sgd', loss='categorical_crossentropy')
```

در نهایت پس از آموزش شبکه، به نتیجه‌ای مشابه حالت قبل می‌رسیم.

In [55]:
```
Y = model.predict(X_test_orig/255.0,verbose=1)
confusion_matrix(y_pred=Y.argmax(1), y_true=Y_test.argmax(1))
```

```
120/120 [==============================] - 1s 9ms/step
```

Out[55]:
```
array([[20,  0,  0,  0,  0,  0],
       [ 0, 20,  0,  0,  0,  0],
       [ 0,  1, 19,  0,  0,  0],
       [ 0,  0,  0, 20,  0,  0],
       [ 0,  0,  1,  0, 18,  1],
       [ 0,  0,  0,  0,  0, 20]])
```

جمع بندی

در اینجا با توجه به این که داده‌های مورد بحث ساختار بسیار ساده‌ای دارند بنابراین با استفاده از یک شبکه ساده و آموزش از ابتدا می‌توانیم به نتیجه مطلوب برسیم. اما برای آموزش شبکه با استفاده از یک شبکه از پیش آموزش دیده شده با معماری MobileNet_v2 نیاز است که به مواردی توجه شود.

شبکه‌های آماده به طور معمول ممکن است دارای پارامترهای خیلی زیادی باشد که آموزش آن‌ها بدون دقت ممکن است باعث overfit شدن شبکه شود، در اینگونه مسائل شبکه را در دو مرحله آموزش می‌دهیم و در هر دو حالت از یک بهینه ساز مانند SGD و یک نرخ یادگیری پایین استفاده می‌کنیم. شبکه را در دو مرحله fine-tunning و Transfer-learning آموزش می‌دهیم. در مرحله اول فقط بخش دسته بند شبکه یعنی شبکه تمام متصل آموزش داده می‌شود، در حالت دوم تمام شبکه آموزش داده می‌شود. به این ترتیب دانشی که شبکه در کاربرد قبل یادگرفته است در کاربرد جدید همچنان باقی می‌ماند.

علاوه بر موارد بالا برای جلوگیری از overfit شدن شبکه می‌توان از تنظیم کننده‌ها نیز بهره برد. در دو جا می‌توانیم تنظمیم کننده را تعریف کنیم. یکی در کنار وزن‌های شبکه تمام متصل، که تعداد وزن‌ها زیاد است. اولین لایه دسته بند معمولا برای این کار مناسب است. و یا این که می‌توانیم در کنار تابع loss شبکه یک تنظیم کننده تعریف کنیم که به این ترتیب از تمام وزن‌های شبکه محافظت می‌کند.

در صورت استفاده از تنظیم کننده، باید توجه داشته باشیم که مقدار زیاد این پارامتر می‌تواند جلوی آموزش مناسب را نیز بگیرد، بنابراین بهتر است که اگر در ابتدا مقدار آن را زیاد در نظر گرفتیم، به طور مرحله‌ای آن را کاهش دهیم.

به طور کلی استفاده از یک شبکه به صورت Transfer-learning می‌تواند به طور چشم‌گیری در نتیجه یادگیری مؤثر باشد، هر چند که در اینجا با توجه به ساده بودن داده‌ها هر دو حالت به یک نتیجه رسیدند.