یادگیری عمیق

گزارش مینی پروژه دوم

نام استاد : دکتر ستوده

نام دانشجو : حمزه قائدی

شماره دانشجویی: 9831419

هدف:

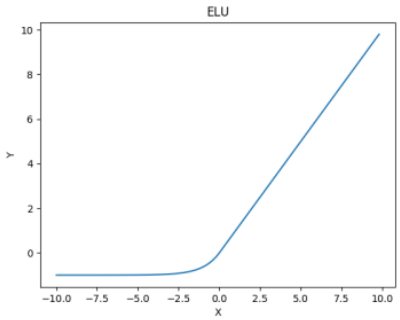
در این پروژه با طراحی یک شبکه عصبی فیدفوروارد و یک شبکه عصبی کانولوشنی برای یک مسئله خاص، به بررسی و مقایسه این دو مدل از شبکه های عصبی میپردازیم و در بخش دوم نیز روش transfer learning را بررسی میکنیم

ملاحضات کلی:

به منظور اجتناب از تکرار، اجزای استفاده شده در طول پروژه در زیر شرح داده شده است:

1) **تابع فعال ساز:** در تمامی لایه ها از تابع ELU به عنوان فعالساز نورن ها استفاده شده است. ضابطه و نمودار این تابع به صورت زیر است:

$$ELU = \begin{cases} x & x \geq 0 \\ a(e^x - 1) & x < 0 \end{cases}$$



2) **تابع هزینه** : از تابع SparseCategoricalCrossEntropy استفاده شده است. گر توزیع دسته های خروجی multinouli (برنولی چندتایی) فرض شود و با این فرض likelihood خروجی برحسب پارمترهای شبکه نوشته شود با ماکسیمم کردن تابع (likelihood)

بر حسب پارامترهای خروجی، به رابطه زیر میرسیم که به آن CategoricalCrossEntropy میگویند (که ااین اصطلاح از نظریه اطلاعات وام گرفته شده است)

در رابطه زیر فرض شده است که هر عکس تنها متعلق به یک دسته است و لذا دسته های خروجی به صورت one hot کد شده اند . اگر دسته های خروجی به صورت اعداد صحیح باشند و یا هر عکس به بیش از یک دسته تعلق داشته باشد در این صورت با اندکی تغییر در کد کردن لیبل های خروجی ، ازهمان رابطه ولی با نام SparseCategoricalCrossEntropy به عنوان تابع هزینه استفاده میشود

$$J(w) = -\frac{1}{N}\sum_{i=1}^{N}[\, y_i \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)\,]$$

3) **بهینه ساز** : الگوریتم RMSprop: در این الگوریتم نرخ یادگیری در هر مرحله به اندازه گرادیان مراحل قبل وابسه است لذا نرخ یادگیری وفقپذیر است. این امر منجر به پایداری بهتر فرایند بهینه سازی میشود. این الگوریتم، وزنها را با توجه به روابط زیر آپدیت میکند:

$$For\ each\ Parameter\ w^j$$
$$(j\ subscript\ dropped\ for\ clarity)$$
$$\nu_t = \rho\nu_{t-1} + (1-\rho) * g_t^2$$
$$\Delta\omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$
$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

$\eta : Initial\ Learning\ rate$
$\nu_t : Exponential\ Average\ of\ squares\ of\ gradients$
$g_t : Gradient\ at\ time\ t\ along\ \omega^j$

بررسی لایه های مورد استفاده :

Flatten: این لایه نمونه های ورودی را که به صورت عکسهای دوبعدی سیاه و سفید (ماتریس) است، به بردار تبدیل میکند

Batch Normalization: این لایه، میانگین و واریانس نمونه های خروجی لایه قبل از خود را نرمالیزه میکند (میانگین را به صفر و واریانس را به یک میل میدهد) و با اینکار سرعت یادگیری و نیز پایداری شبکه را بهبود میبخشد

Dropout: به منظور کاهش overfitting استفاده شده است

Dense: لایه ایست که هر ورودی آن به تمام خروجی هایش مرتبط است

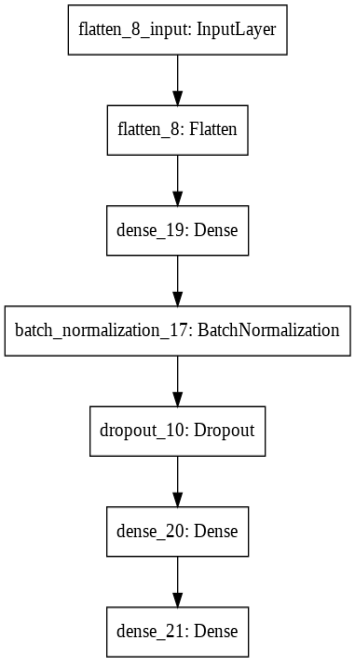
همچنین در تعدادی از لایه ها از نرم مرتبه دو به عنوان regularizer استفاده شده است که معمولا باعث کاهش overfitting میشود همچنین رنج تغییرات وزنهای لایه را کاهش میدهد.

Early Stopping: روشی برای جلوگیری از Overfitting میباشد. بدین صورت که یکی از متریک هارا حین یادگیری بررسی میکند و در صورت عدم تغییر مناسب فرایند یادگیری را خاتمه میدهد

***در هر دو بخش 5000 عکس از مجموعه x_train به عنوان validation set استفاده شده است**

بخش اول

طراحی شبکه فیدفوروارد




```
Model: FFN_model
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_8 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_19 (Dense)             (None, 256)               262400
_________________________________________________________________
batch_normalization_17 (Batc (None, 256)               1024
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_20 (Dense)             (None, 64)                16448
_________________________________________________________________
dense_21 (Dense)             (None, 10)                650
=================================================================
Total params: 280,522
Trainable params: 280,010
Non-trainable params: 512
```

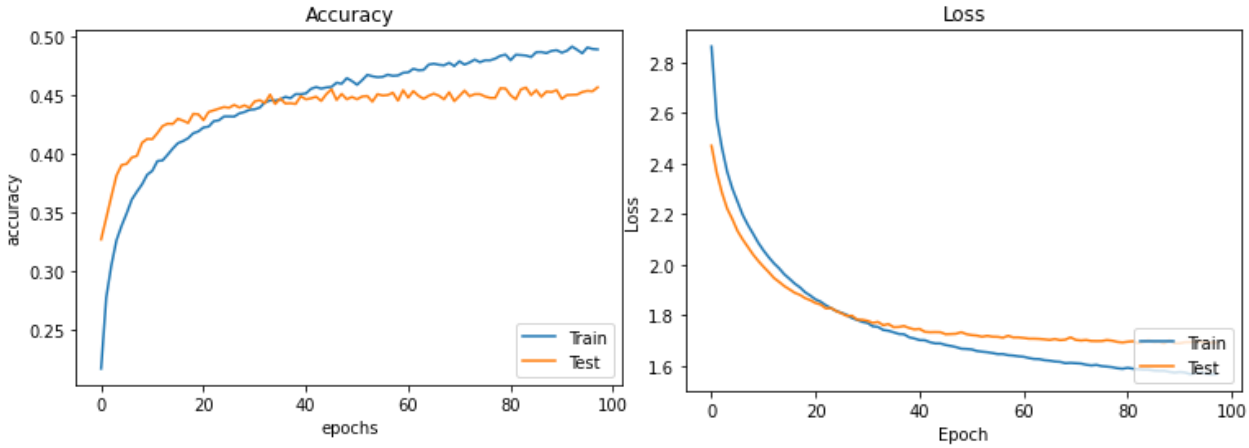
Optimization: RMSprop lr = 0.0001 decay = 10e-6,

Batch_size = 64, epochs = 100

Callbacks: EarlyStopping min_delta = 0.001 patience = 10

نتایج :

```
704/704 [==============================] - 5s 7ms/step - loss: 1.5688 - accuracy: 0.4904 - val_loss: 1.6921 - val_accuracy: 0.4534
Epoch 97/100
704/704 [==============================] - 5s 7ms/step - loss: 1.5667 - accuracy: 0.4891 - val_loss: 1.6921 - val_accuracy: 0.4530
Epoch 98/100
704/704 [==============================] - 5s 7ms/step - loss: 1.5669 - accuracy: 0.4887 - val_loss: 1.6918 - val_accuracy: 0.4564
<tensorflow.python.keras.callbacks.History at 0x7efc98fafc88>
```

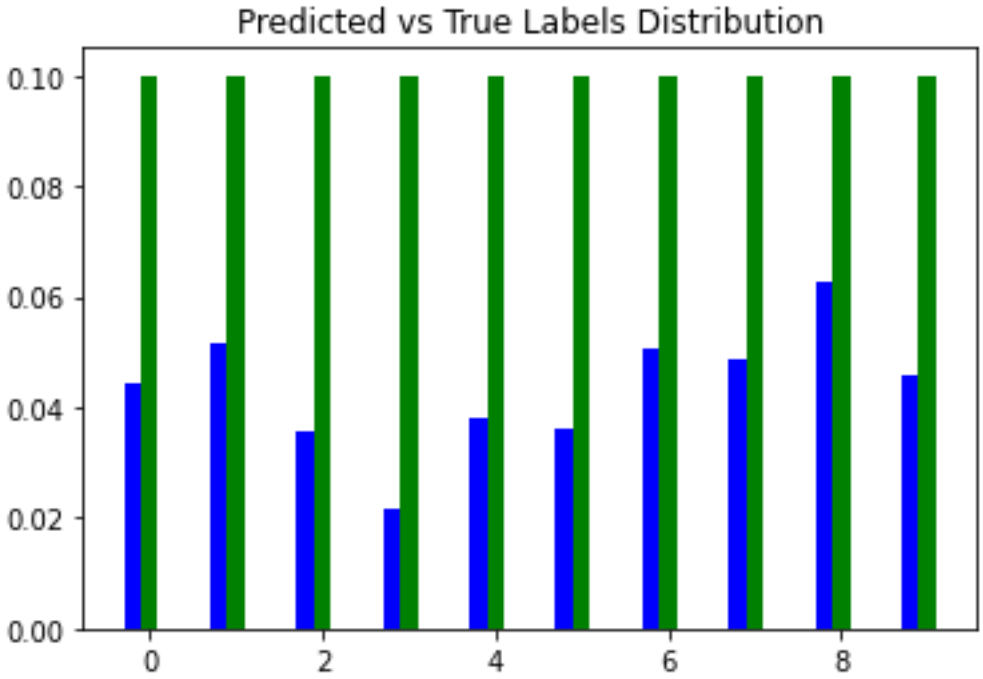


نتایج اعمال مدل روی مجموعه تست:

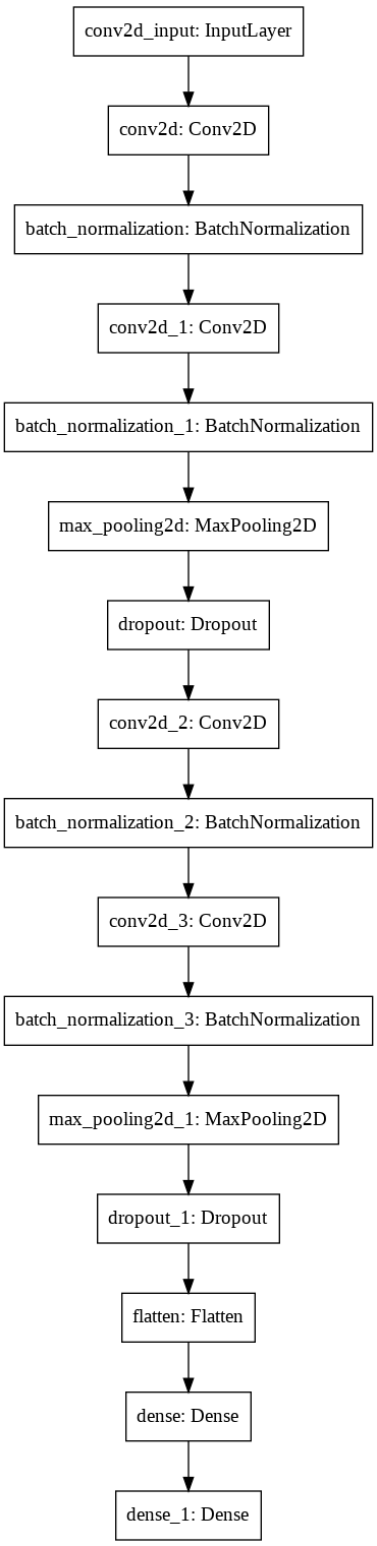
```
FFN_model.evaluate(x_test,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 1.6972 - accuracy: 0.4350
[1.697173833847046, 0.4350000023841858]
```

در نمودار زیر، توزیع حدس های درست برای هر دسته(با رنگ آبی) و نیز توزیع اعضای هر دسته در مجموعه تست (با رنگ سبز) نمایش داده شده است



طراحی شبکه کانولوشنی :




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1048832
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,117,162
Trainable params: 1,116,778
Non-trainable params: 384
```

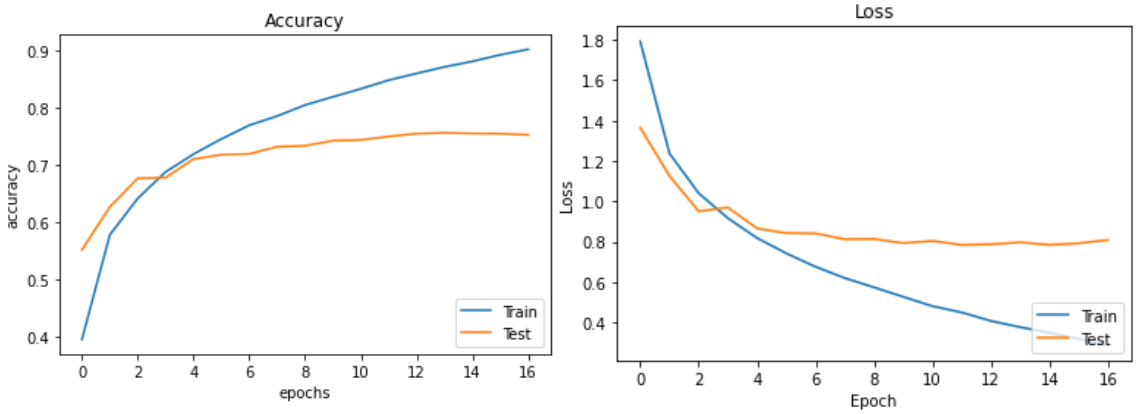
Optimization: RMSprop lr = 0.0001 decay = 10e-6,

Batch_size = 32, epochs = 50

Callbacks: EarlyStopping min_delta = 0.001 patience = 10

نتایج:

```
Epoch 16/50
1407/1407 [==============================] - 302s 215ms/step - loss: 0.3195 - accuracy: 0.8935 - val_loss: 0.7924 - val_accuracy: 0.7554
Epoch 17/50
1407/1407 [==============================] - 298s 212ms/step - loss: 0.2881 - accuracy: 0.9032 - val_loss: 0.8088 - val_accuracy: 0.7534
<tensorflow.python.keras.callbacks.History at 0x7f1de8716e10>
```

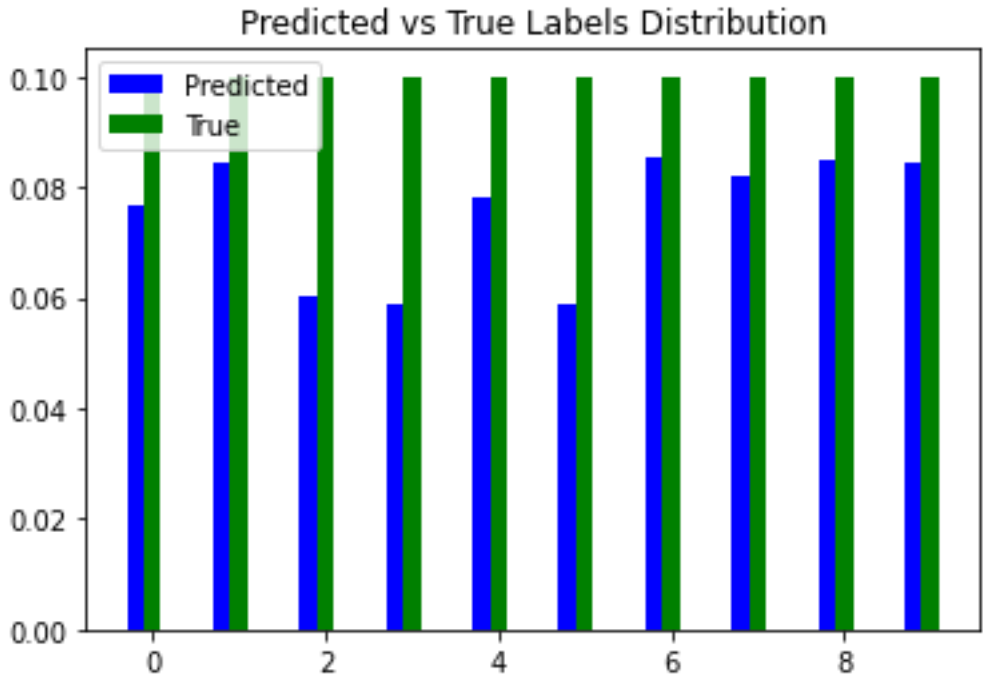


نتایج اعمال مدل بر روی مجموعه تست:

```
CNN_model.evaluate(x_test,y_test)

313/313 [==============================] - 17s 55ms/step - loss: 0.8403 - accuracy: 0.7545
[0.8402857184410095, 0.7544999718666077]
```

در نمودار زیر، توزیع حدس های درست برای هر دسته(با رنگ آبی) و نیز توزیع اعضای هر دسته در مجموعه تست (با رنگ سبز) نمایش داده شده است



مقایسه دو مدل:

شبکه عصبی فیدفوروارد ساختار ساده تری داشته و آموزش آن محاسبات کمتر و لذا زمان کمتری میطلبد اما در مورد این مسئله خاص، و به طور کلی مسائل دسته بندی عکس، این نوع از شبکه ها دقت قابل قبولی ندارند علت این امر را میتوان اینگونه تشریح کرد که هنگام اعمال ورودی به یک شکبه فیدفوروارد عکس ورودی که یک ماتریس دو بعدی است باید به یک بردار یک بعدی تبدیل شود، این امر منجر به از دست رفتن اطلاعات فضایی پیکسل (طرز کنار هم قرار گرفتن پیکسلها) میشود درصورتی که در شبکه ها کانولوشنی از اطلاعات فضایی پیکسلها نیز به منظور طبقه بندی استفاده میشود.. در این پروژه با تغییر هایپرپارامترهای شبکه فیدفوروارد، نظیر تعداد لایه ها و نورونها، نوع توابع فعال ساز و یا تغییر روش بهینه سازی، فرق مشهودی حاصل نشد و دقت تعمیم از حدود 50 درصد فراتر نرفت. (اگر چه ممکن است با افزایش قابل ملاحظه لایه ها و پیچیده کردن مدل دقت کمی بهتر شود). در مقابل شبکه های کانولوشنی، ساختار پیچیده تری دارند و آموزش آنها زمان بسیار بیشتری نسبت به شبکه های فیدفوروارد میطلبد اما نتایج بسیار بهتری خواهند داشت. در این پروژه با یک شبکه عصبی کانولوشنی نسبتا ساده دقت تعمیم 75 درصد حاصل شده است. با تغییر هایپرپارامترهای مدل کانولوشنی، مثلا افزودن دو لایه کانولوشنی با 128 فیلتر و...، میتوان دقت را بالاتر نیز برد (اما فرایند یادگیری بسیار زمان بر خواهد شد)

بخش دوم

استفاده از VGGNet

الف) دو مدل برای شبکه VGGNet در دسترس است. مدل با ساختار 16 لایه و مدل با ساختار 19 لایه، در شکل صفحه بعد ساختار 16 لایه و ویژگی های هر لایه قابل مشاهده است.ورودی این شبکه یک عکس رنگی با اندازه 224*224 میباشد. تنها پیشپردازش اعمال شده روی وردی، کم کردن میانگین محاسبه شده روی داده های آموزشی از پیکسلهای عکس ورودی است. عکس ورودی از چندلایه کانولوشنی با فیلترهای 3*3 و استراید 1 عبور و پنج لایه maxpooling عبور میکند و در نهایت از سه لایه Fully Connected عبور داده میشود. قابل ذکر است که تابع فعال ساز لایه های پنهان Relu بوده و لایه خروجی نیز یک لایه با تابع فعال ساز softmax است. خروجی نیز یک بردار 1000 بعدی است که هر درایه آن احتمال تعلق عکس ورودی به دسته مربوطه (مشخص شده با شماره یا ایندکس درایه) را مشخص مکند. (عکس مدل در صفحه آخر)

VGG19 با بیش از یک میلیون عکس و طی چندین هفته توسط NVIDIA TITAN GPUs آموزش دیده است. آموزش چنین شبکه ای امری هزینه بر بوده و خارج از عهده بسیاری از شرکتهای نوپا و محققین این حوزه میباشد. اما خوشبختانه نتایج حاصله در دسترس است و علاقه مندان میتواند از این شبکه در جهت مقاصد مختلف نظیر شناسایی الگو ،طبقه بندی تصویر و بینایی ماشین و...بهره ببرند.

ب) به کمک کد زیر میتوان از VGG19 استفاده کرد:

```
from keras.applications import vgg19

#load pre-trained model-----------------------
model = vgg19.VGG19()
#---------------------------------------------
```

ج)

```
'30': ['n01641577', 'bullfrog'],'31': ['n01644373', 'tree_frog'],
 '32': ['n01644900', 'tailed_frog'],'33': ['n01664065', 'loggerhead'],
 '34': ['n01665541', 'leatherback_turtle'], '35': ['n01667114', 'mud_turtle'],
 '36': ['n01667778', 'terrapin'], '37': ['n01669191', 'box_turtle'],
 '38': ['n01675722', 'banded_gecko'],'39': ['n01677366', 'common_iguana'],
 '40': ['n01682714', 'American_chameleon'],'41': ['n01685808', 'whiptail'],
 '42': ['n01687978', 'agama'],'43': ['n01688243', 'frilled_lizard'],
 '44': ['n01689811', 'alligator_lizard'],'45': ['n01692333', 'Gila_monster'],
 '46': ['n01693334', 'green_lizard'],'47': ['n01694178', 'African_chameleon'],
 '48': ['n01695060', 'Komodo_dragon'],'49': ['n01697457', 'African_crocodile'],
 '50': ['n01698640', 'American_alligator'],
```

د)

عکس اول :

```
img = image.load_img('clock.jpg', target_size=(224, 224))
img = image.img_to_array(img)
img = img.reshape((1,img.shape[0],img.shape[1],img.shape[2]))

img = vgg19.preprocess_input(img)

y_predicted = model.predict(img)
vgg19.decode_predictions(y_predicted)
```

```
[[('n03706229', 'magnetic_compass', 0.67551917),
  ('n04328186', 'stopwatch', 0.09867849),
  ('n04141975', 'scale', 0.04245477),
  ('n04579432', 'whistle', 0.036351677),
  ('n03814906', 'necklace', 0.027735323)]]
```

عکس دوم:

```
img = image.load_img('clock1.jpg', target_size=(224, 224))
img = image.img_to_array(img)
img = img.reshape((1,img.shape[0],img.shape[1],img.shape[2]))

img = vgg19.preprocess_input(img)

y_predicted = model.predict(img)
vgg19.decode_predictions(y_predicted)
```

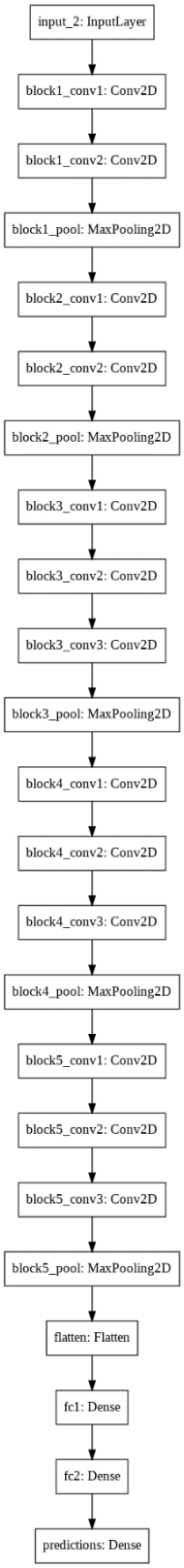
```
[[('n03706229', 'magnetic_compass', 0.658107),
  ('n04328186', 'stopwatch', 0.13404484),
  ('n02794156', 'barometer', 0.085258216),
  ('n02708093', 'analog_clock', 0.05863261),
  ('n04141975', 'scale', 0.03153348)]]
```

نتیجه گیری:

به طور کلی، در مسئله دسته بندی تصاویر، شبکه های کانولوشنی به دلیل در نظر گرفتن اطلاعات فضایی پیکسلهای تصویر به نحو قابل ملاحظه ای بهتر از شبکه های فیدفوروارد عمل میکنند اما در عوض یادگیری آنها نسبت به شبکه های فیدفوروارد بسیار زمانبر است.

زمانبر بودن فرایند یادگیری، باعث ارائه و محبوبیت راهکارهایی نظیر Transfer learning شده است که در این پروژه با آن آشنا شدیم.

مدل شبکه VGG16:




```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
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: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```