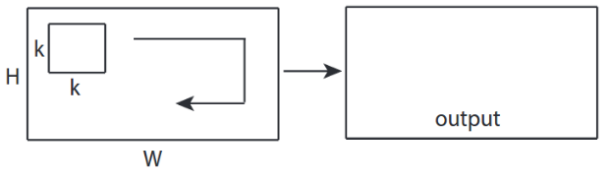
به نام خدا

تمرین نهم یادگیری عمیق

غزل زمانی‌نژاد

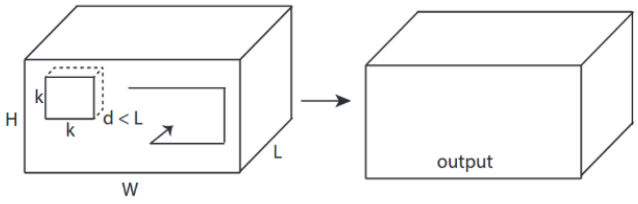
۹۷۵۲۲۱۶۶

1. **الف) conv2d: در کانولوشن 2بعدی کرنل در 2 جهت حرکت می کند. ورودی و خروجی یک لایه conv2d 3بعد دارد. یک فیلتر 3بعدی در کانال های مختلف داده convolve می شود. بیشتر برای تصاویر استفاده می شود. هر تصویر 2بعدی دارای مشخصه طول و عرض به همراه تعداد کانال ها است. در شبکه با تاپل 3تایی (height, width, channel) مشخص می شود به همین دلیل میگوییم ورودی شبکه 3بعد دارد. کرنل را در 2بعد مطابق تصویر زیر حرکت میدهیم:**



مزیت: کرنل میتواند ویژگی های spatial داده را به خوبی استخراج کند. مثلا با استفاده از کرنل مناسب میتوانیم لبه های موجود در یک تصویر را پیدا کنیم. که این کار باعث میشود شبکه های کانولوشنی در دسته بندی داده های دارای ویژگی های فضایی، مقاوم شده و عملکرد خوبی داشته باشند.

Conv3d: کرنل در 3 جهت حرکت می کند. ورودی و خروجی یک لایه conv3d 4بعد دارد. بیشتر برای تصاویر 3بعدی (مثل CT Scan و MRI) و فیلم استفاده می شود چون در این نوع داده، در هر time span فریمی از تصاویر داریم یعنی نسبت به لایه 2بعدی یک محور زمان هم دارد. تعداد پارامترهای قابل آموزش این لایه از conv2d بیشتر است. کرنل را در 3بعد مطابق تصویر زیر حرکت میدهیم:



مزیت: میتوانیم ویژگی های داده را در 3بعد طول، عرض و ارتفاع (یا زمان) به خوبی استخراج کنیم. مثلا در یک کلیپ، تصاویر موجود در فریم ها از نظر زمانی هم به یکدیگر وابستگی دارند. در صورتی که ویژگی ها را در 2بعد استخراج کنیم، از وابستگی زمانی آنها صرف نظر کرده ایم. در صورتی که با استفاده از conv3d، آن را هم درنظر گرفته ایم.

ب) زمانی که از ابعاد ویژگی هایی که شبکه قرار است آموزش ببیند اطلاعی نداریم، بهتر است از فیلتر مربعی استفاده کنیم. زیرا الگوها هم به صورت عمودی و هم افقی میتوانند وجود داشته باشند و با استفاده کردن از فیلتر مربعی میتوانیم آن را به شکل قرینه استخراج کنیم. به این شکل، هر نقطه خروجی، از یک مجموعه مناسب از همسایگان بدست می آید که تعداد نقاط مجاور عمودی و افقی در آن برابر هستند. اما در صورتی که از فیلتر غیر مربعی، مثلا 2*8 استفاده کنیم، در هر بخش از تصویر، تعداد نقاط افقی که آن را به خروجی مپ کرده ایم 4 برابر نقاط عمودی بوده است. و این فیلتر برای استخراج لبه های افقی می تواند مناسب باشد. اما در حالت کلی بهتر است از فیلتر مربعی استفاده شود.

برای انتخاب سایز فیلتر باید به پیچیدگی و جزئیات تصویر دقت داشته باشیم. برای داده های کوچک و ساده همچون تشخیص اعداد minst، از فیلترهای کوچک (مثلا 3*3 و 5*5) استفاده می شود اما برای تصاویر بزرگتر و پیچیده تر از جمله تشخیص چهره، از فیلترهای بزرگتر (مثلا 7*7 و 9*9) نیز می توانیم استفاده کنیم.

به طور معمول، توصیه می شود از فیلترهایی که ابعادشان فرد است استفاده کنیم. زیرا تمامی پیکسل های لایه قبل به صورت متقارن در اطراف پیکسل خروجی قرار میگیرند. اما فیلترهای با ابعاد زوج این خاصیت را ندارند.

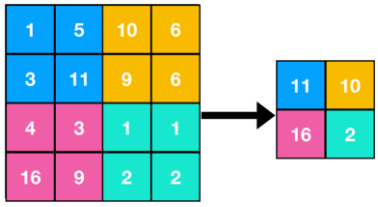
همچنین توصیه می شود از فیلترها با ابعاد کوچکتر استفاده شود. زیرا این فیلترها میتوانند بخش های کوچک تصویر را با دقت بیشتری مورد بررسی قرار دهند. در این میان فیلتر 1*1 برای کاهش عمق و کمتر کردن تعداد پارامترها استفاده می شود و برای استخراج ویژگی توصیه نمی شود. اگر سایز فیلتر بزرگ باشد، تعداد پارامترهای شبکه افزایش می یابد و در نتیجه هزینه محاسباتی بیشتر می شود.

برای نخستین لایه می توانیم سایز فیلتر را بزرگتر در نظر بگیریم. زیرا در لایه های ابتدایی میخواهیم ویژگی های کلی را استخراج کنیم.

با بررسی معماری VGG میبینیم که این شبکه تنها از فیلتر 3*3 استفاده کرده. مثلا به جای استفاده از یک فیلتر 5*5، از دو فیلتر 3*3 و تعداد لایه های بیشتر استفاده شده. این کار کمک می کند شبکه غیرخطی تر و قدرت یادگیری آن بیشتر شود.

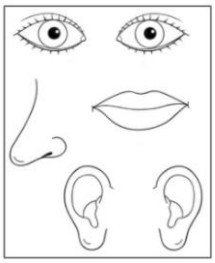
ج) 4 نوع لایه pooling داریم:

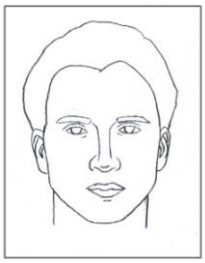
- Max pool: این لایه بزرگترین مقدار موجود درون بخشی از داده که فیلتر روی آن قرار میگیرد را به عنوان خروجی میدهد. برای نگه داشتن پیکسل هایی که بیشترین مقدار را دارند استفاده میشود.



- Average pool: میانگین مقادیر موجود درون بخشی از داده که فیلتر روی آن قرار میگیرد را به عنوان خروجی میدهد. برای کوچک کردن بازنمایی استفاده میشود. نسبت به max pool، اطلاعات کمتری از ورودی را حذف میکند (زیرا میانگین همه مقادیر را در محاسبات دخیل میکند).
- Min pool: این لایه کمترین مقدار موجود درون بخشی از داده که فیلتر روی آن قرار میگیرد را به عنوان خروجی میدهد.
- Adaptive pool: در این لایه، کاربر به جای وارد کردن هایپرپارامترها (از جمله سایز فیلتر) ، اندازه خروجی مورد نظر را وارد می کند. بر اساس آن، لایه خودش اندازه فیلتر را بدست می آورد.

لایه pooling از پارامترهای همسایه یک مشخصه آماری درمی آورد و کمک میکند بازنمایی نسبت به جابه جایی کوچک، حساسیت کمتری داشته باشد. در مسائلی که مکان اشیا اهمیتی ندارد از این نوع لایه استفاده می شود. مثلا برای تشخیص اینکه در تصویر گربه وجود دارد یا خیر، محل قرارگیری تاثیری در جواب ندارد. اما در مسئله تشخیص وجود چهره، محل قرارگیری المان ها اهمیت دارند. مثلا در تصویر سمت چپ، چون اعضای چهره ترتیب قرارگیری درستی ندارند، جواب عدم وجود چهره خواهد بود. پس در این نوع مسائل اگر لایه pooling به درستی استفاده نشود، می تواند اثر منفی بگذارد.





2. ابتدا فایل تصویر را در کولب آپدیت میکنیم. سپس به کمک کتابخانه cv2 آن را میخوانیم و در کنسول نمایش می دهیم.

```
import numpy as np
from google.colab.patches import cv2_imshow
import cv2

path = 'img1.jpg'
img = cv2.imread(path, flags=cv2.IMREAD_GRAYSCALE)
cv2_imshow(img)
print("\n")
```

متد convolve: این متد تصویر و فیلتر را به عنوان ورودی دریافت می کند. با استفاده از متد filter2D موجود در کتابخانه cv2 فیلتر را در تصویر convolve میکنیم. با تابع imshow نتیجه را در خروجی نمایش می دهیم.

```
def convolve(img, filter):
  output = cv2.filter2D(img, -1, filter)
  cv2_imshow(output)
  print("\n")
```

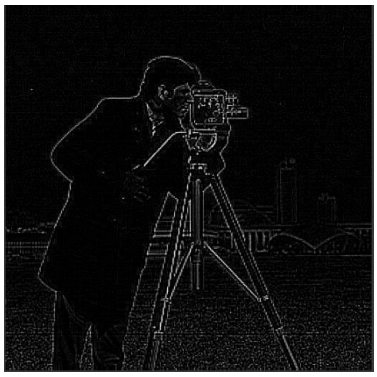
فیلترهای خواسته شده را با استفاده از numpy تشکیل می دهیم و متد convolve را برای تک تک کرنل ها صدا می زنیم.

```
filter1 = (1/9) * np.array(([1, 1, 1],
                            [1, 1, 1],
                            [1, 1, 1]))

filter2 = np.array(([-1, -1, -1],
                    [-1, 8, -1],
                    [-1, -1, -1]))

filter3 = np.array(([-1, -2, -1],
                    [0, 0, 0],
                    [1, 2, 1]))

filter4 = np.array(([-1, 0, 1],
                    [-2, 0, 2],
                    [-1, 0, 1]))

convolve(img, filter1)
convolve(img, filter2)
convolve(img, filter3)
convolve(img, filter4)
```

بررسی فیلترها:

فیلتر اول: میدانیم در یک تصویر سیاه و سفید مقادیر پیکسل ها از 0 تا 255 است. 0 نشان دهنده رنگ سیاه و 255 نشان دهنده رنگ سفید است. با اعمال این فیلتر، مقادیر پیکسل ها کوچکتر میشوند. هرچه مقادیر کمتر باشند، تصویر تیره تر میشود. در خروجی مشاهده میشود تصویر کمی تیره و تار شده.

فیلتر دوم: این فیلتر برای مشخص کردن طرح اصلی موجود در تصویر به کار می رود. بعد از اعمال این فیلتر، یک تصویر سیاه داریم که کلیت اشیا موجود در آن (لبه اشیا) با خطوط سفید نمایش داده می شود. در خروجی یک طرح کلی از مرد و دوربین و منظره اطراف که با رنگ سفید مشخص شده مشاهده می شود.



فیلتر سوم: برای مشخص کردن لبه های افقی موجود در تصویر به کار می رود. در خروجی تصویر با زمینه سیاه دیده می شود که لبه های افقی آن با خطوط سفید مشخص شده اند.

فیلتر چهارم: برای مشخص کردن لبه های عمودی موجود در تصویر به کار می رود. در خروجی تصویر با زمینه سیاه دیده می شود که لبه های عمودی آن با خطوط سفید مشخص شده اند.

3. **الف) میدانیم یک مدل علاوه بر پارامترهای قابل آموزش، تعدادی هایپرپارامتر نیز دارد که به صورت دستی آنها را تنظیم میکنیم تا به بهترین نتیجه برسیم. کتابخانه kerasTuner یک ابزار قدرتمند برای تنظیم هایپرپارامترها به صورت خودکار است. در صورتی که هایپرپارامترهای مسئله به صورت دقیق تنظیم شوند، میتوانند تاثیر بسیار خوبی در بالا بردن دقت آزمون داشته باشند. به بررسی چند موردی که میتوانیم با استفاده از این ابزار تنظیم کنیم می پردازیم:**
   - **تنظیم تعداد نورون های لایه dense با استفاده از hp.Int**
   - **انتخاب تابع فعال سازی با استفاده از hp.Choice**
   - **انتخاب استفاده یا عدم استفاده از لایه dropout با استفاده از hp.Boolean**
   - **تنظیم نرخ یادگیری با استفاده از hp.Float**

   مراحل استفاده از تنظیم خودکار برای یافتن بهترین مدل:
   - **تعریف مدل و مشخص کردن فضای جستجوی هایپرپارامترها**
   - **شروع به جستجو با مشخص کردن کلاس tuner**
   - **چاپ کردن مدلی که بر روی داده آزمون بهترین عملکرد را داشته**

   همچنین میتوانیم فرآیند آموزش دیدن مدل را tune کنیم. مثلا با استفاده از آن مدل تصمیم بگیرد که در هر ایپوک داده ها را شافل کند یا خیر.

   برای پیش پردازش داده نیز میتوانیم tuning را انجام دهیم. مثلا با استفاده از آن تصمیم بگیرد که داده ها را پیش از شروع آموزش نرمالایز کند یا خیر.

   میتوانیم هدف از tuning را مشخص کنیم. مثلا هدف از آموزش دادن مدل های مختلف میتواند رسیدن به بالاترین دقت داده آزمون باشد.

ب) در کراس 3 نوع کلاس tuner وجود دارد که از الگوریتم های مختلف استفاده می کنند:

- **RandomSearch: در میان سایر tunerها کمترین کارآمدی را دارد زیرا از بهترین نتایجی که بدست آورده استفاده نمی کند و هر دفعه در میان فضای حالت، هایپرپارامترها را به صورت رندوم انتخاب میکند.**
- **BayesianOptimization: مشابه RandomSearch است. با این تفاوت که انتخاب ترکیب هایپرپارامترها به صورت تصادفی انجام نمیشود بلکه از یک روش احتمال در الگوریتم خود استفاده می کند.**
- **Hyperband: یک روش بهینه از RandomSearch است که در آن زمان جستجو و تخصیص منابع بهینه تر شده است.**

برای پیاده سازی این سوال از RandomSearch استفاده میکنیم. زیرا این روش شهودی ترین روش ممکن است که به صورت تصادفی مقادیری را از فضای حالت هایپرپارامترها انتخاب میکند و در میان این انتخاب های تصادفی، میتوانیم به یک tuning مناسب برسیم.

ج) سلول اول: keras tuner را در محیط کولب نصب میکنیم.

سلول دوم: دیتاست cifar10 که در کتابخانه کراس موجود است را دانلود کرده و میخوانیم. شکل ورودی داده آزمایشی و برچسب ها و همچنین شکل داده ورودی تست و برچسب را چاپ میکنیم. برای اینکه مقادیر پیکسل های ورودی بازه بزرگی از اعداد را شامل نشود آنها را نرمالایز میکنیم. و در آخر با استفاده از متد to_categorical، برچسب نمونه ها را به صورت one hot درمی آوریم.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

print("train input shape", x_train.shape)
print("train label shape", y_train.shape)
print("test input shape", x_test.shape)
print("test label shape", y_test.shape)

# normalize pixels
x_train = x_train.astype("float32")  / 255
x_test = x_test.astype("float32")  / 255

# encode labels with one-hot
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
train input shape (50000, 32, 32, 3)
train label shape (50000, 1)
test input shape (10000, 32, 32, 3)
test label shape (10000, 1)
```

سلول سوم: متد build_model **را برای ساختن شبکه تعریف میکنیم. در اینجا برای استفاده از امکان تنظیم خودکار کراس،** hp **را به عنوان ورودی به متد میدهیم و در تابع از آن استفاده میکنیم. در ابتدا با** models.Sequential **یک مدل میسازیم.**

```
def build_model(hp):
    p = 0.2
    model = keras.models.Sequential()
```

به عنوان اولین لایه، یک لایه کانولوشنی درنظر می گیریم که تعداد فیلترهای آن را کراس می آموزد. کمترین تعداد 32 و بیشترین 256 است و می توانیم گام های 32تایی برداریم. همچنین انتخاب تابع فعال سازی مناسب از میان relu **و** tanh **را کراس انجام میدهد. برای این مسئله سایز کرنل 3 را در نظر می گیریم.**

```
    model.add(Conv2D(hp.Int('conv1_filters', min_value=32, max_value=256, step=32),
                     (3, 3),
                     input_shape=x_train.shape[1:],
                     activation=hp.Choice("activation", ["relu", "tanh"])))
```

در لایه دوم از max pooling **استفاده میکنیم. بدین ترتیب ابعاد خروجی این شبکه کاهش می یابد.**

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

برای تنظیم لایه های بعدی از یک حلقه استفاده میکنیم. مطابق این حلقه ممکن است تا 4 لایه کانولوشنی به شبکه اضافه شود. در آن سایز کرنل لایه را با کراس تنظیم میکند (میتواند 3 یا 5 باشد). همچنین تعداد فیلترهای مورد استفاده و تابع فعال سازی مشابه قسمت قبل است.

سپس برای اینکه شبکه overfit **نشود از** hp.Boolean **استفاده میکنیم تا در صورتی که** dropout **اثر مثبت** regularization **داشت از آن استفاده کنیم.**

```
    for i in range(hp.Int('conv_layers', 2, 5)):
        kernel_size = hp.Choice(f'kernel_size{i}', values=[3, 5])
        model.add(Conv2D(hp.Int(f'conv{i}_filters', min_value=32, max_value=256, step=32),
                         kernel_size,
                         activation=hp.Choice(f'activation{i}', ["relu", "tanh"])))
        if hp.Boolean("dropout"):
            model.add(Dropout(p))
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

خروجی لایه های قبل را flatten **میکنیم. سپس آن را از یک لایه** Dense **که تعداد نورون های آن از طریق** kears tuner **تنظیم میشود. این تعداد میتواند از 32 تا 256 باشد.**

```
model.add(Flatten())
model.add(Dense(units=hp.Int(f'dense_neurons1', min_value=32, max_value=256, step=64), activation='relu'))
```

برای تنظیم لایه های بعدی از حلقه استفاده میکنیم. مطابق این حلقه ممکن است تا 4 لایه dense **به شبکه اضافه شود. تعداد نورون ها و تابع فعال سازی مشابه قسمت قبل توسط کراس بدست می آید.**

```
for i in range(hp.Int('dense_layers', 2, 5)):
    model.add(Dense(units=hp.Int(f'neurons{i}', min_value=32, max_value=256, step=64), activation='relu'))
```

برای لایه آخر یک لایه Dense **با 10 نورون خروجی و تابع فعال سازی سافت مکس تشکیل می دهیم. بعد برای نرخ یادگیری 3 مقدار را در نظر می گیریم و با استفاده از** keras tuner **آن را** tune **میکنیم. برای انتخاب الگوریتم بهینه سازی مناسب هم از آدما و** sgd **استفاده میکنیم.**

```
    # last layer has 10 neurons
    model.add(Dense(10, activation='softmax'))
    # tune learning rate
    alpha = hp.Choice('learning_rate', values=[0.001, 0.0005, 0.0001])
    # tune optimizer
    optimizer = hp.Choice('optimizer', values=['adam', 'SGD'])
    if optimizer == 'adam':
          optimizer = tf.keras.optimizers.Adam(learning_rate=alpha)
    else: optimizer = tf.keras.optimizers.SGD(learning_rate=alpha)
```

در آخر مدل را با بهینه ساز و نرخ یادگیری مناسب، تابع ضرر categorical cross entropy **و معیار دقت کامپایل میکنیم**

سلول چهارم: یه tuner از نوع random search تشکیل میدهیم و فضای حالت های ممکن را چاپ میکنیم.

```
tuner = kt.RandomSearch(hypermodel=build_model,
                        objective="val_accuracy",
                        max_trials=4,
                        executions_per_trial=3,
                        overwrite=True,
                        directory="result",
                        project_name="cnn_tuning",
                        )

tuner.search_space_summary()
```

```
Search space summary
Default search space size: 15
conv1_filters (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
conv_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 5, 'step': 1, 'sampling': None}
kernel_size0 (Choice)
{'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
conv0_filters (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 32, 'sampling': None}
activation0 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
dropout (Boolean)
{'default': False, 'conditions': []}
kernel_size1 (Choice)
{'default': 3, 'conditions': [], 'values': [3, 5], 'ordered': True}
activation1 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
dense_neurons1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 64, 'sampling': None}
dense_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 5, 'step': 1, 'sampling': None}
neurons0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 64, 'sampling': None}
neurons1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 256, 'step': 64, 'sampling': None}
learning rate (Choice)
```

سلول پنجم: همچنین از early stopping نیز استفاده میکنیم تا در صورتی که مقدار دقت آزمون برای چند ایپوک تغییری نکرد، فرآیند آموزش را متوقف کنیم.

```
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
```

سلول ششم: فرآیند را با اجرای خط tuner.search آغاز میکنیم. با اجرای این خط، keras tuner به صورت تصادفی در تعداد مشخصی ایپوک و trial مدل های گوناگون با هایپرپارامترهای حدید تشکیل میدهد و بهترین نتیجه را ذخیره میکند.

```
tuner.search(x_train, y_train, epochs=8, validation_data=(x_test, y_test), callbacks=[stop_early])
```

این تصویر به طور مثال هنگام فرآیند آموزش ثبت شده است.

```
Best val_accuracy So Far: 0.6780666510264078
Total elapsed time: 00h 06m 18s

Search: Running Trial #2

Hyperparameter    |Value           |Best Value So Far
conv1_filters     |96              |256
activation        |relu            |tanh
conv_layers       |3               |2
kernel_size0      |5               |5
conv0_filters     |224             |224
activation0       |relu            |relu
dropout           |True            |True
kernel_size1      |3               |3
activation1       |relu            |tanh
dense_neurons1    |160             |224
dense_layers      |5               |3
neurons0          |96              |224
neurons1          |96              |96
learning_rate     |0.0005          |0.0005
optimizer         |SGD             |adam
neurons2          |32              |32
```

پس از پایان آموزش (که فرآیندی زمان گیر است) بهترین مدل را از میان مدل های آموزش دیده پیدا کرده و چاپ میکنیم.

```
3 models = tuner.get_best_models(num_models=1)
4 best_model = models[0]
```

```
Trial 4 Complete [00h 03m 56s]
val_accuracy: 0.11963333189487457

Best val_accuracy So Far: 0.6780666510264078
Total elapsed time: 00h 21m 35s
```

سلول هفتم: ابتدا آن را بیلد میکنیم. سپس خلاصه آن را چاپ میکنیم. و میزان دقت و خطای آزمون را بدست می آوریم.

```
best_model.build(input_shape=(None, 32, 32, 3))
print(best_model.summary())
loss, accuracy = best_model.evaluate(x_test, y_test)

print('loss:', loss)
print('accuracy:', accuracy)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 256)       7168
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 256)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 224)       1433824
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 224)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 256)         516352
_________________________________________________________________
dropout_1 (Dropout)          (None, 9, 9, 256)         0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 224)               917728
_________________________________________________________________
dense_1 (Dense)              (None, 224)               50400
_________________________________________________________________
dense_2 (Dense)              (None, 96)                21600
_________________________________________________________________
dense_3 (Dense)              (None, 32)                3104
_________________________________________________________________
dense_4 (Dense)              (None, 10)                330
=================================================================
Total params: 2,950,506
Trainable params: 2,950,506
Non-trainable params: 0
```

```
_________________________________________________________________
None
313/313 [==============================] - 1s 4ms/step - loss: 1.0286 - accuracy: 0.6838
loss: 1.0285815000534058
accuracy: 0.6837999820709229
```

سلول آخر: در آن خلاصه ای از نتایجی که tuner **به آن رسیده را چاپ میکنیم.**

```
1 tuner.results_summary()
```

```
Results summary
Results in result/cnn_tuning
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
conv1_filters: 256
activation: tanh
conv_layers: 2
kernel_size0: 5
conv0_filters: 224
activation0: relu
dropout: True
kernel_size1: 3
activation1: tanh
dense_neurons1: 224
dense_layers: 3
neurons0: 224
neurons1: 96
learning_rate: 0.0005
optimizer: adam
neurons2: 32
Score: 0.6780666510264078
Trial summary
Hyperparameters:
conv1_filters: 96
activation: relu
conv_layers: 3
kernel_size0: 5
conv0_filters: 224
activation0: relu
dropout: True
kernel_size1: 3
activation1: relu
dense_neurons1: 160
```

نتیجه نهایی: در این مسئله بهترین نتیجه از طریق هایپرپارامترهای زیر بدست آمد:

- **استفاده از 256 فیلتر در لایه conv نخست**
- **استفاده از تابع فعال سازی tanh**
- **استفاده از 2 لایه دیگر کانولوشنی**
- **سایز کرنل 5 و تعداد 224 کرنل برای دومین لایه کانولوشن**
- **استفاده از تابع فعال سازی relu**
- **استفاده از dropout**
- **سایز کرنل 3 برای سومین لایه کانولوشن**
- **استفاده از تابع فعال سازی tanh**
- **تعداد 224 نورون برای لایه dense اول**

- **تعداد 3 لایه dense دیگر**
- **اولی دارای 224 نورون**
- **دومی دارای 96 نورون**
- **سومی دارای 32 نورون**
- **استفاده از بهینه ساز آدام**
- **نرخ یادگیری 0.0005**

که این موارد مورد انتظار بودند. در لایه های convolutional از تعداد فیلترهای زیادی برای استخراج ویژگی های بیشتر استفاده شد. از دو لایه dropout برای جلوگیری از overfit استفاده شد. در لایه های fully connected، تعداد نورون ها از زیاد شروع شده و در طی چند لایه کاهش پیدا کرده تا به تعداد 10 نورون لایه آخر که برای دسته بندی است برسد. در اثر fine tuning به این نتیجه رسیدیم بهینه ساز آدام برای این مسئله عملکرد بهتری خواهد داشت و همگرایی آن بهتر است. همچنین نرخ یادگیری عددی میان 1e-3, 1e-4 مناسب ترین عملکرد را کسب کرد.

4. **صفحه بعد**

trainable 1: $[(7\times7)+1]\times20 = 1000$

(7×7): تعداد خانه‌های یک کرنل — +1: bias

ابعاد خروجی = $28-7+1 = 22$ (22, 22, 20)

trainable 2: 0 لایه pooling پارامتر قابل آموزش ندارد

ابعاد خروجی = $\left\lfloor \frac{22-2}{2}+1 \right\rfloor = 11$ (11, 11, 20)

trainable 3: $[(5\times5\times20)+1]\times10 = 5010$

(5×5×20) = 500

ابعاد خروجی = $11-5+1 = 7$ (7, 7, 10)

trainable 4: $[(3\times3\times10)+1]\times[5\times5\times2] = 91\times50 = 4550$

ابعاد خروجی = $7-3+1 = 5$ (5, 5, 2)

trainable 5: 0 لایه flatten پارامتر قابل آموزش ندارد

trainable 6: $(5\times5\times2+1)\times10 = 510$

(5×5×2) = 50 — +1: bias

$$\text{total trainable} = \sum_{i=1}^{6} \text{trainable}_i$$

$$= 1000 + 5010 + 4550 + 510$$

منابع

https://xzz201920.medium.com/conv1d-conv2d-and-conv3d-8a59182c4d6

https://datascience.stackexchange.com/questions/51470/what-are-the-differences-between-convolutional1d-convolutional2d-and-convoluti

https://stackoverflow.com/questions/59946176/non-squared-convolution-kernel-size

https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15#7430

https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel

https://neptune.ai/blog/keras-tuner-tuning-hyperparameters-deep-learning-model

https://www.tensorflow.org/tutorials/keras/keras_tuner