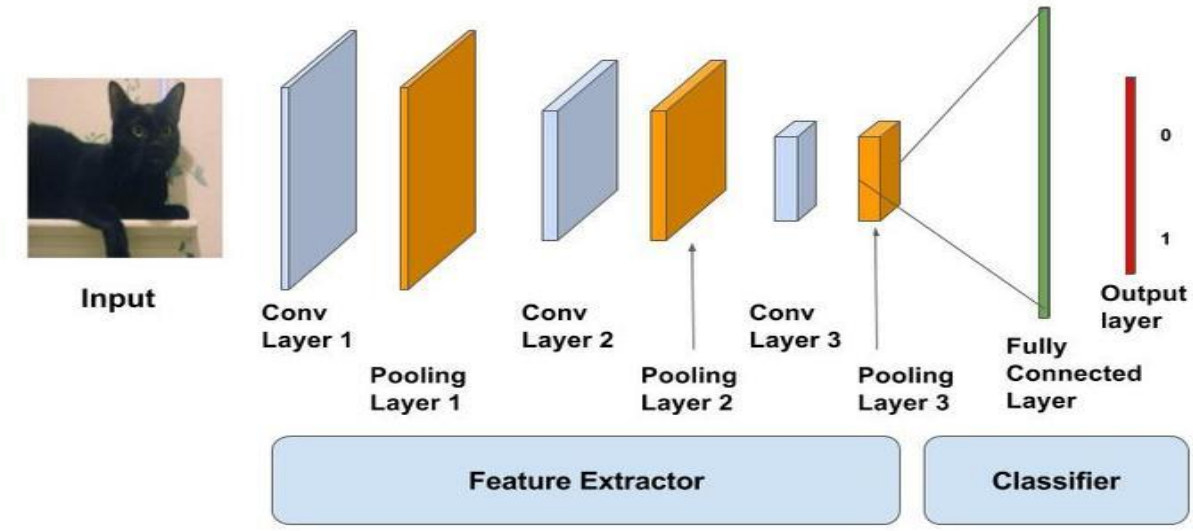
اجزای CNN مدل CNN در دو مرحله کار می‌کند:

- استخراج ویژگی و طبقه‌بندی استخراج ویژگی مرحله‌ای است که در آن فیلترها و لایه‌های مختلفی بر روی تصاویر اعمال می‌شود تا اطلاعات و ویژگی‌ها از آن استخراج شود و پس از انجام آن به تصویر بعدی منتقل می‌شود.
- مرحلهدوم یعنی طبقه بندی که در آن بر اساس متغیر هدف مسئله طبقه بندی می شوند.

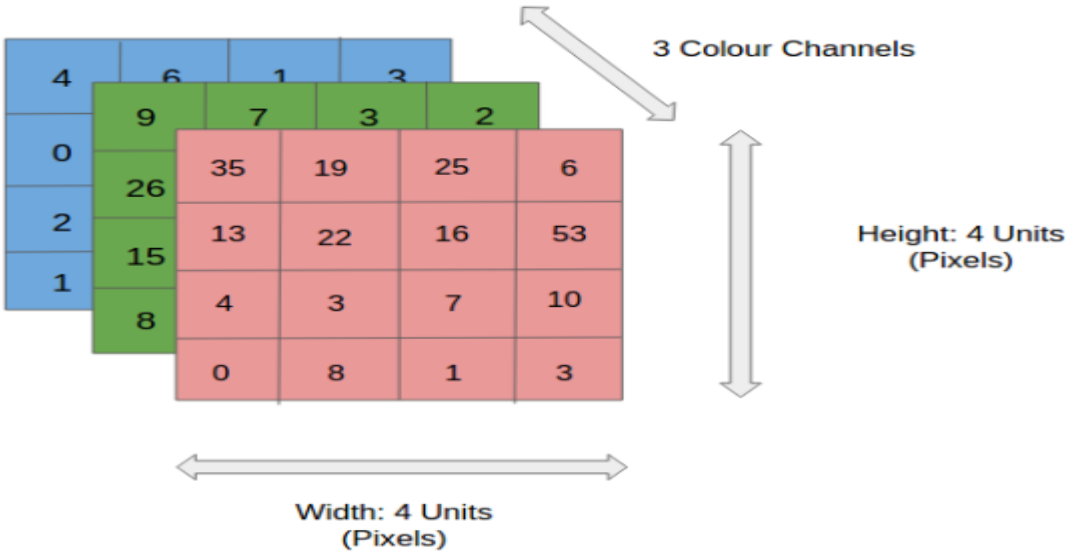
✓ یک مدل معمولی CNN به شکل زیر است:

- Input layer
- Convolution layer + Activation function
- Pooling layer
- Fully Connected Layer



لایه ورودی

همانطور که از نامش می‌گوید، این تصویر ورودی ما است و می‌تواند Grayscale یا RGB باشد. هر تصویر از پیکسل هایی تشکیل شده است که از 0 تا 255 متغیر هستند. ما باید آنها را نرمال سازی کنیم، یعنی قبل از ارسال آن به مدل، محدوده را بین 0 به 1 تبدیل کنیم. در زیر نمونه تصویر ورودی با اندازه 4*4 و دارای 3 کانال یعنی مقادیر RGB و پیکسل است.



import tensorflow as tf:

این خط کد، کتابخانه TensorFlow را به برنامه وارد می کند.

from tensorflow.keras import datasets, layers, models:

این خط کد، برخی از ماژول های مفید از TensorFlow Keras را وارد می کند.

from google.colab import drive:

این خط کد، امکان دسترسی به Google Drive را فراهم می کند.

(train_images, train_labels), (test_images, test_labels) = datasets. cifar10.load_data ():

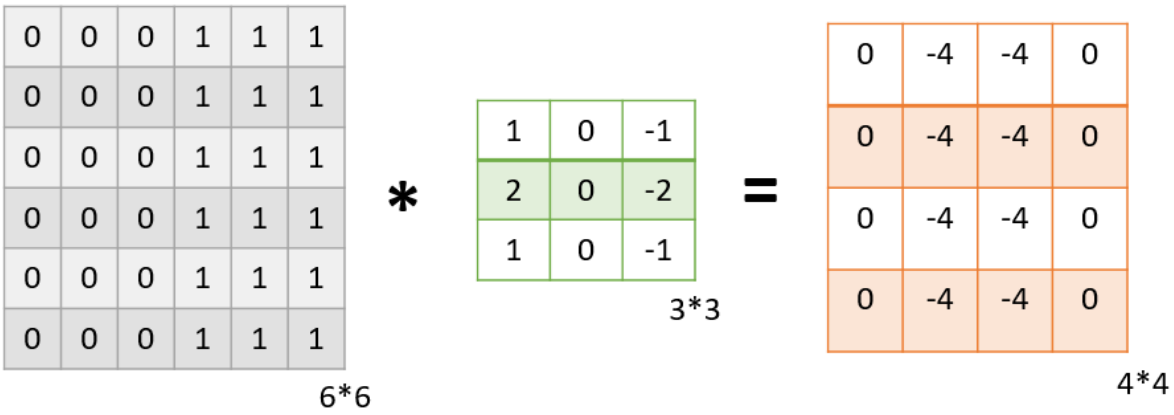
این خط کد، مجموعه داده CIFAR-10 را بارگذاری می کند و آن را به متغیرهای train_images, train_labels, test_images و test_labels اختصاص می دهد.

train_images, test_images = train_images / 255.0, test_images / 255.0:

این خطوط کد، تصاویر را به مقیاس 0 تا 1 نرمال می کنند.

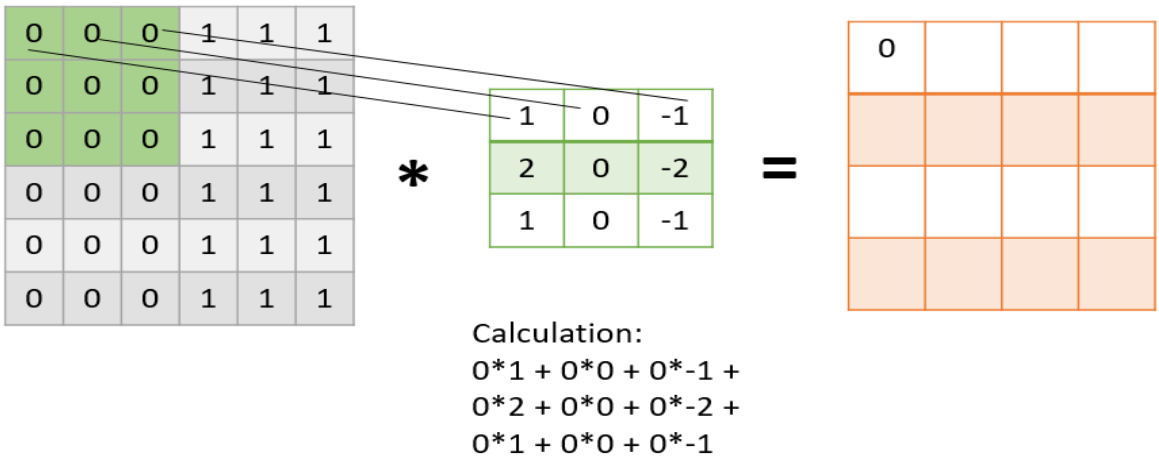
Convolution Layer

Convolution Layer لایه ای است که در آن فیلتر روی تصویر ورودی ما اعمال می شود تا ویژگی های آن استخراج یا شناسایی شود. یک فیلتر چندین بار روی تصویر اعمال می شود و یک نقشه ویژگی ایجاد می کند که به طبقه بندی تصویر ورودی کمک می کند. بیایید این را با کمک یک مثال درک کنیم. برای سادگی، یک تصویر ورودی 2 بعدی با پیکسل های نرمال شده می گیریم.



در شکل بالا یک تصویر ورودی به اندازه 6*6 داریم و فیلتر 3*3 را روی آن اعمال کرده ایم تا برخی ویژگی ها را شناسایی کنیم. در این مثال، ما فقط یک فیلتر را اعمال کرده ایم اما در عمل، فیلترهای زیادی برای استخراج اطلاعات از تصویر اعمال می شوند.

نتیجه اعمال فیلتر روی تصویر این است که ما یک نقشه ویژگی 4*4 دریافت می کنیم که اطلاعاتی در مورد تصویر ورودی دارد. بسیاری از این نقشه های ویژگی در کاربردهای عملی تولید می شوند. برخی ریاضیات پشت دریافت نقشه ویژگی در تصویر بالا



همانطور که در شکل بالا نشان داده شده است، در مرحله اول فیلتر روی قسمت سبز رنگ تصویر اعمال می شود و مقادیر پیکسل تصویر با مقادیر فیلتر (همانطور که در شکل با استفاده از خطوط نشان داده شده است) ضرب می شود و سپس برای به دست آوردن ارزش نهایی خلاصه می شود. در مرحله بعد فیلتر با یک ستون جابه جا می شود که در شکل زیر نشان داده شده است. این پرش به ستون یا ردیف بعدی به عنوان گام شناخته می شود و در این مثال، ما یک گام برداریم که به این معنی است که یک ستون جابجا می شویم.

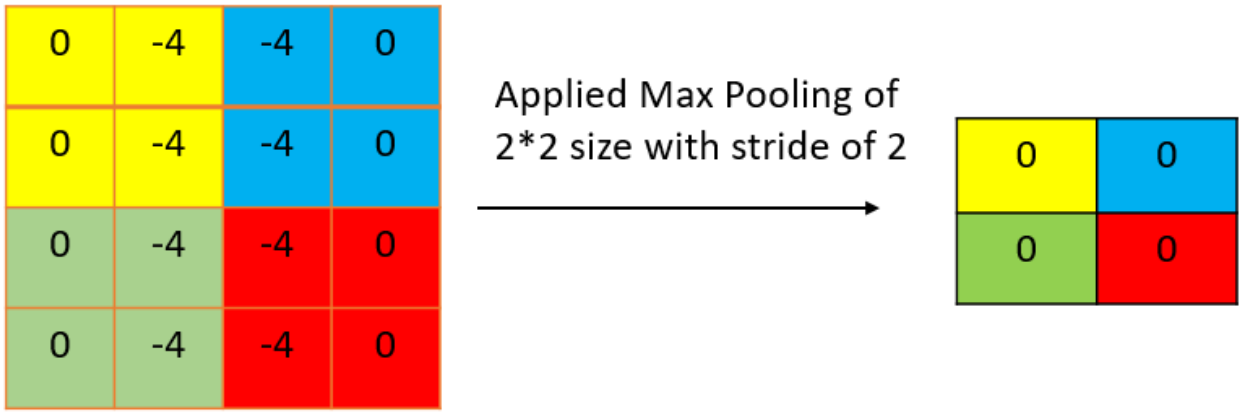
Pooling Layer

لایه Pooling بعد از لایه Convolutional اعمال می شود و برای کاهش ابعاد نقشه ویژگی استفاده می شود که به حفظ اطلاعات مهم یا ویژگی های تصویر ورودی کمک می کند و زمان محاسبه را کاهش می دهد. با استفاده از ادغام، یک نسخه با وضوح پایین تر از ورودی ایجاد می شود که همچنان حاوی عناصر بزرگ یا مهم تصویر ورودی است.

متداول ترین انواع Pooling عبارتند از Max Pooling و .Average Pooling شکل زیر نحوه عملکرد Max Pooling را نشان می دهد. با استفاده از نقشه ویژگی که از مثال بالا به دست آوردیم برای اعمال .Pooling در اینجا ما از یک لایه Pooling به اندازه 2*2 با گام 2 استفاده می کنیم.

حداکثر مقدار از هر ناحیه هایلایت شده گرفته می شود و نسخه جدیدی از تصویر ورودی به دست می آید که در اندازه 2*2

می باشد بنابراین پس از اعمال Pooling بعد نقشه ویژگی کاهش می یابد.



:

Load the pre-trained model weights

model = models.Sequential()

این خط یک مدل سکوئنشال جدید ایجاد می‌کند. مدل سکوئنشال یک نوع معماری شبکه‌ی عصبی است که لایه‌ها به صورت خطی روی هم قرار می‌گیرند.

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))

این خط یک لایه کانولوشن دو بعدی به مدل اضافه می‌کند. لایه Conv2D مجموعه‌ای از فیلترهای قابل آموزش را روی ورودی (تصویر) اعمال می‌کند. پارامتر اول 32 تعداد فیلترها را مشخص می‌کند، پارامتر دوم (3, 3) اندازه‌ی فیلترها را یک مربع(3*3) تعیین می‌کند،

activation='relu' تابع فعال‌سازی ReLU را روی خروجی لایه اعمال می‌کند، و input_shape=(32, 32,3) شکل ورودی تصاویر را 32x32 پیکسل با 3 کانال رنگی مانند (RGB مشخص می‌کند

model.add(layers.MaxPooling2D((2, 2)))

این خط یک لایه حداکثرگیری (max-pooling) به مدل اضافه می‌کند. حداکثرگیری یک نوع کاهش غیرخطی اندازه است که اندازه‌ی فضایی بازنمایی را کاهش داده و به ویژگی‌هایی که استخراج می‌کند بی‌اعتنایی بیشتری نسبت به جابه‌جایی‌های کوچک در ورودی می‌بخشد. پارامتر (2, 2) اندازه‌ی پنجره حداکثرگیری را تعیین می‌کند.

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

این خط یک لایه کانولوشن دو بعدی دیگر با 64 فیلتر اندازه 3 3xو فعال‌سازی ReLU اضافه می‌کند.

```
model.add(layers.MaxPooling2D((2, 2)))
```

این خط یک لایه حداکثرگیری دیگر با پنجره 2* 2 اضافه می‌کند.

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

این خط یک لایه کانولوشن سوم با 64 فیلتر اندازه 3* 3 و فعال‌سازی ReLU اضافه می‌کند.

.

Fully Connected Layer

تا به حال مراحل استخراج ویژگی را انجام داده ایم، اکنون بخش طبقه بندی می آید. لایه کاملاً متصل همانطور که در ANN داری (برای طبقه بندی تصویر ورودی به یک برچسب استفاده می شود. این لایه اطلاعات استخراج شده از مراحل قبلی) یعنی لایه **Convolution**و Pooling) را به لایه خروجی متصل می کند و در نهایت ورودی را در برچسب مورد نظر طبقه بندی می کند.

```
model.add(layers.Flatten())
```

این خط خروجی لایه‌های کانولوشنی قبلی را به یک بردار یک‌بعدی متراکم می‌کند تا بتواند به عنوان ورودی به لایه‌های کاملاً متصل (dense) استفاده شود.

```
model.add(layers.Dense(64, activation='relu'))
```

این خط یک لایه کاملاً متصل (dense) با 64 واحد و فعال‌سازی ReLU اضافه می‌کند.

```
model.add(layers.Dense(10))
```

این خط آخرین لایه کاملاً متصل را با 10 واحد (متناظر با تعداد کلاس‌های خروجی) اضافه می‌کند

```
model.compile(optimizer='adam', ...)
```

این خط مدل را کامپایل می‌کند و پارامترهایی مانند بهینه‌ساز (در اینجا Adam)، تابع هزینه و معیارهای مورد نظر برای رصد در طول آموزش را مشخص می‌کند.

```
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
```

این خط، مدل را برای آموزش پیکربندی می‌کند. اپتیمایزر را به adam تنظیم می‌کند، تابع زیان را به Sparse Categorical Crossentropy (که برای مسائل طبقه‌بندی چندکلاسه مناسب است) تنظیم می‌کند و معیار ارزیابی را به ' accuracy' تنظیم می‌کند.

```
history = model.fit(train_images, train_labels, epochs=20,
validation_data=(test_images, test_labels))
```

این خط، مدل را بر روی train_images و train_labels برای 20 ایپوک آموزش می‌دهد. پارامتر validation_data مجموعه داده‌های آزمون/اعتبارسنجی را مشخص می‌کند که برای نظارت بر عملکرد مدل در طول آموزش استفاده می‌شود. متغیر history مقادیر آموزش و اعتبارسنجی را برای هر ایپوک ذخیره می‌کند، که می‌توان برای تحلیل و ترسیم نمودار استفاده کرد.

```
model.load_weights('/content/gdrive/My Drive/model_weights.h5')
```

این خط، وزن‌های از قبل آموزش‌دیده مدل را از مسیر فایل مشخص‌شده بارگذاری می‌کند. این کار مفید است اگر قبلاً مدل را آموزش داده‌اید و می‌خواهید از وزن‌های ذخیره‌شده برای پردازش یا ارزیابی بیشتر استفاده کنید.

```
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'Test accuracy: {test_acc:.4f}')
```

این کد، عملکرد مدل را بر روی مجموعه داده آزمون ارزیابی می‌کند. متغیرهای test_loss و test_acc مقادیر زیان و دقت را ذخیره می‌کنند، به ترتیب. پارامتر verbose=2 خروجی مفصل‌تری را در طول فرآیند ارزیابی فراهم می‌کند. در نهایت، دقت آزمون چاپ می‌شود.

```
predictions = model.predict(test_images)

predicted_classes = tf.argmax(predictions, axis=1)
```

این کد، پیش‌بینی‌ها را برای تصاویر آزمون با استفاده از مدل بارگذاری‌شده تولید می‌کند. متغیر predictions حاوی خروجی‌های لیبل یا احتمالات مدل برای هر تصویر آزمون است. متغیر predicted_classes حاوی برچسب‌های کلاس پیش‌بینی‌شده برای هر تصویر آزمون است، که با گرفتن argmax خروجی‌های لیبل پیش‌بینی در امتداد محور=1 (بُعد کلاس) به دست می‌آید.

```
for i in range(10):

print(f'Actual class: {test_labels[i][0]}, Predicted class: {predicted_classes[i]}')
```

این حلقه، برچسب کلاس واقعی و برچسب کلاس پیش‌بینی‌شده را برای اولین 10 تصویر آزمون چاپ می‌کند. این کار برای بررسی بصری عملکرد مدل بر روی نمونه‌های فردی مفید است.