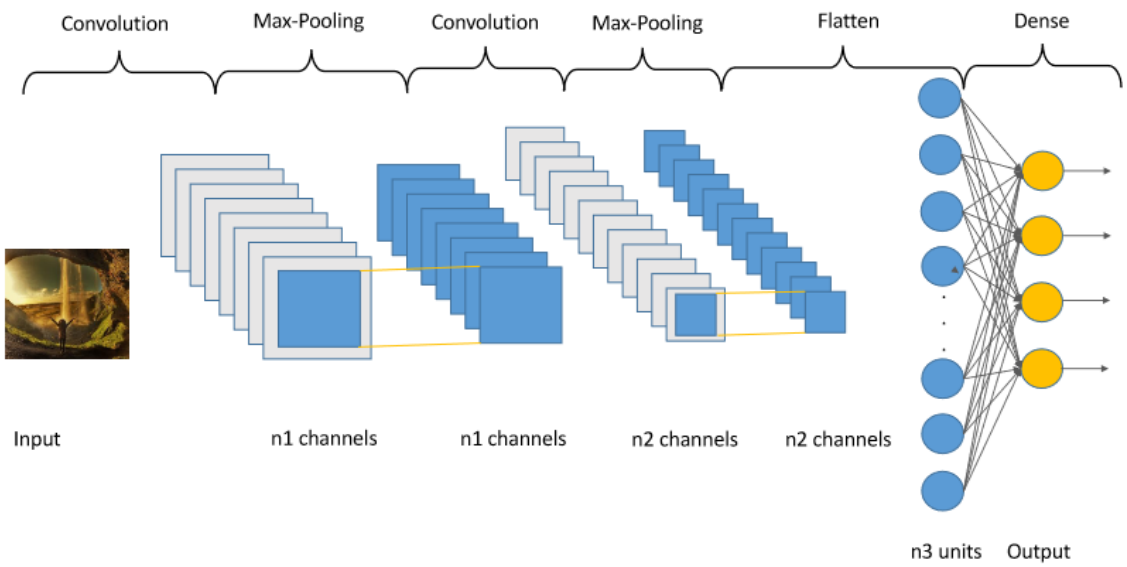


شکل ۱: نمونه ای از شبکه عصبی کانولوشنی

۱ مقدمه

در این آزمایش هدف پیاده سازی و کامپایل یک شبکه کانولوشنی بر روی مجموعه داده $MNIST$ است. ابتدا به مروری کوتاه بر شبکه کانولوشنی پرداخته می شود.

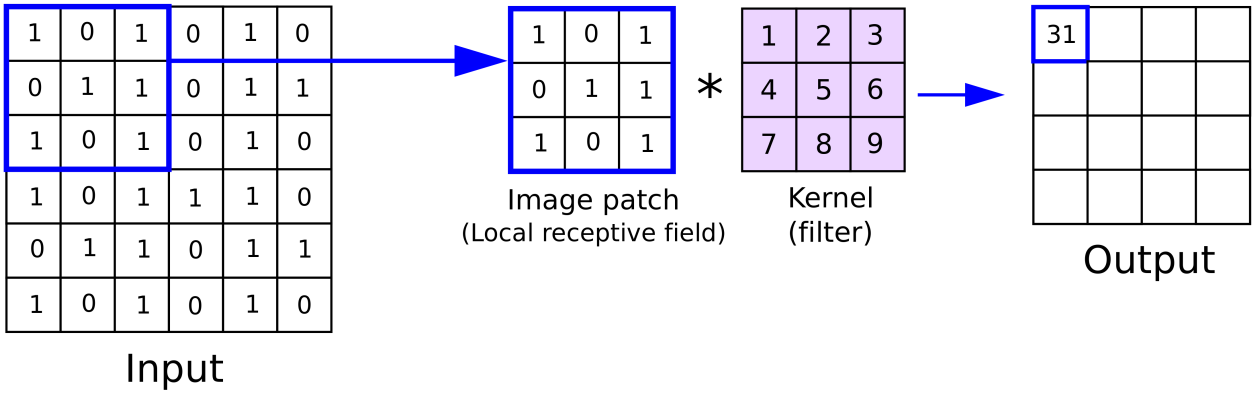
شبکه عصبی کانولوشنی از پرکاربردترین شبکه های عصبی برای پردازش تصویر می باشد. عملکرد بسیاری از حملات نیز بر روی این شبکه عصبی بررسی می شوند. یک نمونه از شبکه کانولوشنی در شکل. ۱ نمایش داده شده است. شبکه کانولوشنی از لایه های اصلی زیر تشکیل شده است:

۱. لایه کانولوشنی

۲. لایه ادغام[۱]

۳. لایه کاملا متصل

[۱] $Pooling$



شکل ۲: عملگر کانولوشن

در ادامه به بررسی هر یک از این لایه ها پرداخته می شود.

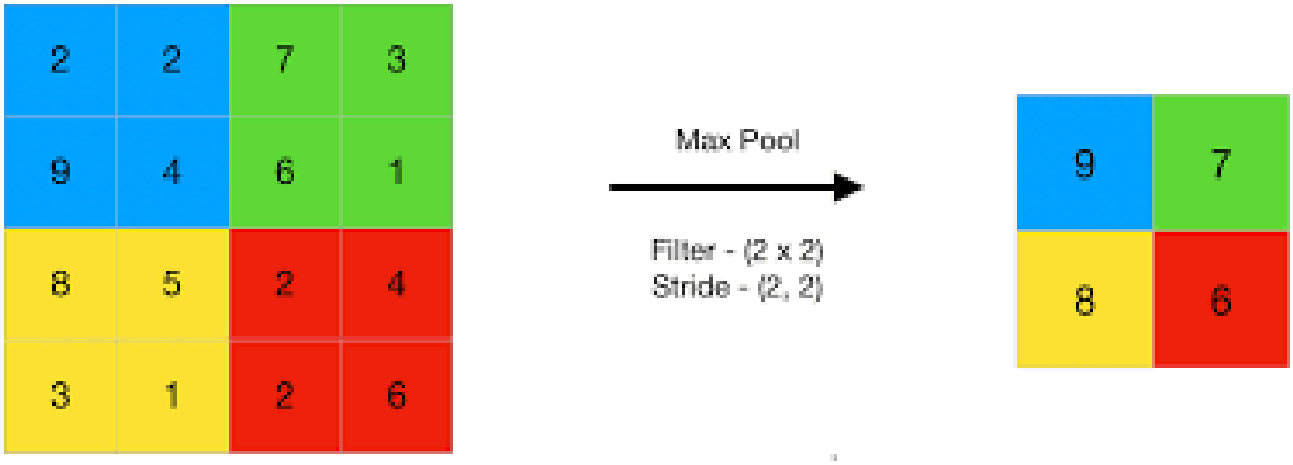
لایه کانولوشنی در این لایه عمل کانولوشن بر روی داده ورودی و با استفاده از تعدادی فیلتر انجام می شود. برای محاسبه هر درایه خروجی، ماتریس فیلتر بر روی ماتریس ورودی لغزانده می شود. عمل کانولوشن به این صورت تعریف می شود که ابتدا اولین عنصر فیلتر بر روی اولین عنصر ماتریس ورودی قرار می گیرد. سپس مجموع ضرب درایه های متناظر فیلتر با درایه های متناظز ماتریس ورودی محاسبه می شود. در نهایت فیلتر بر روی تصویر ورودی به اندازه پارامتر از پیش تعیین شده s یه جلو برده می شود. با تکرار این مراحل ماتریس خروجی محاسبه می شود.

عمل کانولوشن برای محاسبه یک درایه خروجی در شکل.۲ نمایش داده شده است.

عمل کانولوشن در دو حالت یکسان [۲] و اصلی [۳] انجام می پذیرد. تفاوت این دو روش در ابعاد خروجی عملگر

[۲] $same$

[۳] $valid$



شکل ۳: ادغام ماکزیمم

کانولوشن است. در حالت یکسان ماتریس خروجی لایه کانولوشن دارای ابعاد برابر با ماتریس ورودی است. در حالی که در حالت دوم ابعاد ماتریس خروجی می تواند متفاوت باشد.

پارامترهای فیلتر در شبکه عصبی کانولوشنی با استفاده از پس انتشار خطا [۴] حاصل می شود.

لایه ادغام یکی دیگر از لایه های شبکه کانولوشنی لایه ادغام است. این لایه هیچ پارامتر آموزشی ندارد. هدف این لایه کاهش ابعاد ماتریس ورودی ورودی و همزمان حفظ اطلاعات ارزشمند ورودی است. در این لایه ابتدا ماتریس با ابعاد از پیش تعیین شده k در نظر گرفته می شود. این پارامتر معمولا برابر با ۲ در نظر گرفته می شود. سپس با لغزاندن فیلتر بر روی ورودی اندازه ماتریس ورودی کاهش می یابد.

یک نمونه از عملگرهایی که برای نمونه برداری در این لایه استفاده می شود، عملگر بیشینه است. در این حالت ماتریس از پیش تعیین شده بر روی داده ورودی لغزانده می شود و تنها بیشینه عناصری که در هر بخش قرار می گیرند را به عنوان خروجی در نظر می گیرد. یک نمونه از عملگرد ادغام بیشینه در شکل.۳ نمایش داده شده است.

[۴] $Back - propoagation$

لایه کاملا متصل در این لایه یک شبکه عصبی کاملا متصل قرار گرفته است. در این لایه هدف مرتبط کردن ماتریس نهایی با خروجی های نهایی شبکه است. وزن های شبکه کاملا متصل از طریق پس انتشار خطا بدست می آید.

۲ شرح آزمایش

در این آزمایش شبکه کانولوشنی در محیط $Tensorflow$ پیاده سازی می شود.

برای پیاده سازی شبکه کانولوشنی در محیط $Tensorflow$ گام های زیر را دنبال کنید:

۱. فراخوانی مجموعه داده و کتابخانه های ضروری : برخی از کتابخانه های ضروری برای پیاده سازی عبارتند از $Tensorflow, numpy, panda$ و $matplotlib.pyplot$ هستند. با استفاده از دستور $import$ کتابخانه ها را فراخوانی نمایید. با استفاده از دستور $verion-$ از بروز بودن نسخه تنسورفلو خود اطمینان حاصل نمایید (آخرین نسخه تنسورفلو نسخه ۲ می باشد.)

در این آزمایش، از مجموعه داده$MNIST$ استفاده خواهید کرد. این مجموعه از ۶۰۰۰۰ تصویر رقم دست نویس با برچسب های مربوطه برای آزمایش و یک مجموعه از ۱۰۰۰۰ تصویر رقم دست نویس برای تست شبکه تشکیل شده است. تصاویر این مجموعه نرمال سازی شده اند. این مجموعه داده اغلب در تحقیقات یادگیری ماشین استفاده می شود. هدف شما پیاده سازی یک شبکه کانولوشنی برای تشخیص ارقام ورودی است. ارقام ورودی بین ۰ تا ۹ تغییر می کنند. برای بارگذاری مجموعه داده قطعه کد زیر را اجرا نمایید:

۲. پیش پردازش داده ها: برای یادگیری بهتر شبکه تابعی بنویسید که داده های تست و یادگیری شبکه را

```
# Run this cell to load the MNIST data

mnist_data = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist_data.load_data()
```

شکل ۴: کد لازم برای بارگذری مجموعه داده

```
def scale_mnist_data(train_images, test_images):
    """
    This function takes in the training and test images as loaded in the cell above, and scales them
    so that they have minimum and maximum values equal to 0 and 1 respectively.
    Your function should return a tuple (train_images, test_images) of scaled training and test images.
    """

    return (train_images, test_images)
```

شکل ۵: فرمت کلی تابع لازم برای پیش پردازش داده ها

مقیاس بندی کند به گونه ای که مقادیر تصاویر خروجی بین صفر و یک قرار گیرد. شکل کلی تابع در تصویر ۵ نمایش داده شده است برای اطمینان از درست کارکردن تابع خود قطعه کد نمایش داده شده در شکل. (۶) را اجرا کنید.

۳. ساخت شبکه کانولوشنی: در این قسمت هدف پیاده سازی تابعی است که یک شبکه کانولوشنی را بازگرداند.

برای پیاده سازی این شبکه از رابط کاربری $Sequential$ استفاده می کنیم.

قصد ساختن یک شبکه کانولوشنی با ویژگی های زیر را داریم:

(آ) لایه کانولوشن دو بعدی با اندازه فیلتر ۳×۳ و ۸ فیلتر. از پدینگ $same$ و تابع $Relu$ به عنوان تابع

```
# Run your function on the input data

scaled_train_images, scaled_test_images = scale_mnist_data(train_images, test_images)
```

```
# Add a dummy channel dimension

scaled_train_images = scaled_train_images[..., np.newaxis]
scaled_test_images = scaled_test_images[..., np.newaxis]
```

شکل ۶: مقیاس کردن و اضافه کردن ۱ بعد به داده های ورودی

فعالساز استفاده نمایید. مطمئن شوید که آرگومان $input_shape$ در این لایه ارائه شده است.

(ب) یک لایه $Maxpool$ ، با یک پنجره ۲×۲. (مقدار $Stride$ را به صورت پیش فرض در نظر بگیرید)

(ج) یک لایه مسطح، که ورودی را به یک تانسور یک بعدی باز می کند.

(د) دو لایه مخفی متراکم، هر کدام با ۶۴ نورون و تابع فعال ساز $ReLU$.

(ه) یک لایه خروجی متراکم با ۱۰ واحد و تابع فعال سازی $softmax$.

ابعاد تصویر نیز به عنوان ورودی به شبکه کانولوشنی داده می شود. در اولین لایه شبکه کانولوشنی لازم است $input_shape$ را برابر با آرگومان دریافتی از ورودی تابع قرار دهید.

برای پیاده سازی می توانید از کتابخانه های زیر استفاده نمایید:

- $tf.keras.Sequential$
- $tf.keras.layers.Conv2D$
- $tf.keras.layers.MaxPooling2D$
- $tf.keras.layers.Flatten$
- $tf.keras.layers.Dense$

فرم کلی تابع نیز در شکل ۷ نمایش داده شده است.

برای اطمینان از عملکرد تابع قطعه کد نمایش داده شده در شکل ۸ را اجرا نمایید.

۴. کامپایل کردن مدل: اکنون باید مدل را با استفاده از دستور $compile$ کامپایل کنید. برای انجام این کار، باید یک بهینه ساز، یک تابع ضرر و یک متریک برای قضاوت در مورد عملکرد مدل خود مشخص کنید.

```
def get_model(input_shape):
    """
    This function should build a Sequential model according to the above specification. Ensure the
    weights are initialised by providing the input_shape argument in the first layer, given by the
    function argument.
    Your function should return the model.
    """
    model = tf.keras.Sequential([

    ])

    return model
```

شکل ۷: فرم کلی تابع

```
: # Run your function to get the model

  model = get_model(scaled_train_images[0].shape)
```

شکل ۸: دریافت یک شبکه کانولوشنی

مدل را با استفاده از بهینه ساز $Adam$ (با تنظیمات پیش فرض)، تابع ضرر $Cross - entropy$ و و دقت به عنوان تنها معیار اجرا نمایید. توجه داشته باشید لازم نیست تابع نوشته شده آرگومانی را بازگرداند و صرفا با اجرای دستور $compile$ مدل کامپایل می شود.

۵. برازش مدل بر روی داده های ورودی و خروجی: در این قسمت هدف نوشتن تابعی برای برازش بر روی داده های ورودی و خروجی است. مدل و تصاویر مقیاس شده به همراه برچسب های صحیح نیز به عنوان ورودی به تابع داده می شود.اکنون باید با استفاده از روش fit مدل، مدل را بر روی مجموعه داده $MNIST$ آموزش دهید. آموزش را برای ۵ $epoch$ انجام داده و خروجی تابع fit را برای ترسیم نمودارهای دقت به عنوان خروجی تابع برگردانید. فرم کلی تابع در شکل نمایش داده شده است. برای اطمینان از کارکرد صحیح تابع قطعه کد نمایش داده شده در تصویر ۱۰ زیر را اجرا نمایید: در صورت پیاده سازی صحیح باید خروجی نزدیک به خروجی نمایش داده شده در تصویر ۱۱ دریافت نمایید.

```python
def train_model(model, scaled_train_images, train_labels):
    """
    This function should train the model for 5 epochs on the scaled_train_images and train_labels.
    Your function should return the training history, as returned by model.fit.
    """

    return history
```

شکل ۹: شمای کلی تابع برازش

```python
# Run your function to train the model

history = train_model(model, scaled_train_images, train_labels)
```

شکل ۱۰: برازش تابع

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.2034 - sparse_categorical_accuracy: 0.9398
Epoch 2/5
60000/60000 [==============================] - 83s 1ms/sample - loss: 0.0653 - sparse_categorical_accuracy: 0.9804
Epoch 3/5
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.0455 - sparse_categorical_accuracy: 0.9858
Epoch 4/5
60000/60000 [==============================] - 85s 1ms/sample - loss: 0.0330 - sparse_categorical_accuracy: 0.9894
Epoch 5/5
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0260 - sparse_categorical_accuracy: 0.9919
```

شکل ۱۱: برازش تابع

```
frame = pd.DataFrame(history.history)
```

```
# Run this cell to make the Accuracy vs Epochs plot

acc_plot = frame.plot(y="accuracy", title="Accuracy vs Epochs", legend=False)
acc_plot.set(xlabel="Epochs", ylabel="Accuracy")
```

شکل ۱۲: ترسیم نمودار دقت برحسب $epoch$

```
: # Run this cell to make the Loss vs Epochs plot

  acc_plot = frame.plot(y="loss", title = "Loss vs Epochs",legend=False)
  acc_plot.set(xlabel="Epochs", ylabel="Loss")
```

شکل ۱۳: ترسیم نمودار ضرر برحسب $epoch$

۶. نمایش های نمودارهای دقت: با اجرا کردن قطعه کدهای نمایش داده شده در شکل ۱۲ و ۱۳ نمودارهای دقت و ضرر برحسب $epoch$ را برای شبکه ترسیم نماید.

۷. ارزیابی مدل: در این قسمت هدف پیاده سازی تابعی برای ارزیابی مدل است. برای این کار می توانید از تابع $evaluate$ استفاده نمایید. ارزیابی را بر روی مجموعه داده تست مقیاس شده و برچسب های متناظر با آن انجام دهید.دقت و ضرر بر روی مجموعه داده تست را به عنوان خروجی تابع برگردانید. شمای کلی تابع در شکل نمایش داده شده است. با اجرا کردن قطعه کد نمایش داده شده در شکل ۱۵ از درست اجرا شدن تابع اطمینان حاصل نمایید.

```
def evaluate_model(model, scaled_test_images, test_labels):
    """
    This function should evaluate the model on the scaled_test_images and test_labels.
    Your function should return a tuple (test_loss, test_accuracy).
    """
    test_loss, test_accuracy = model.evaluate()

    return (test_loss, test_accuracy)
```

شکل ۱۴: ارزیابی تابع

```
# Run your function to evaluate the model

test_loss, test_accuracy = evaluate_model(model, scaled_test_images, test_labels)
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_accuracy}")
```

شکل ۱۵: ارزیابی تابع

```
# Run this cell to get model predictions on randomly selected test images

num_test_images = scaled_test_images.shape[0]

random_inx = np.random.choice(num_test_images, 4)
random_test_images = scaled_test_images[random_inx, ...]
random_test_labels = test_labels[random_inx, ...]

predictions = model.predict(random_test_images)

fig, axes = plt.subplots(4, 2, figsize=(16, 12))
fig.subplots_adjust(hspace=0.4, wspace=-0.2)

for i, (prediction, image, label) in enumerate(zip(predictions, random_test_images, random_test_labels)):
    axes[i, 0].imshow(np.squeeze(image))
    axes[i, 0].get_xaxis().set_visible(False)
    axes[i, 0].get_yaxis().set_visible(False)
    axes[i, 0].text(10., -1.5, f'Digit {label}')
    axes[i, 1].bar(np.arange(len(prediction)), prediction)
    axes[i, 1].set_xticks(np.arange(len(prediction)))
    axes[i, 1].set_title(f"Categorical distribution. Model prediction: {np.argmax(prediction)}")

plt.show()
```

شکل ۱۶: پیش بینی تابع

۸. پیش بینی شبکه: در نهایت با اجرا کردن کد نمایش داده شده در شکل ۱۶ عملکرد شبکه خود را بر روی برخی از تصاویر $MNIST$ مشاهده کنید.

۳ تمرین

۱. با یک شبکه کانولوشنی با ساختار دلخواه به جداسازی مجموعه داده $Cifar-10$ بپردازید.

- اثر افزایش تعداد لایه های شبکه بر روی دقت پیش بینی شبکه بر روی داده یادگیری را بررسی نمایید
- اثر افزایش تعداد لایه های شبکه بر روی دقت پیش بینی شبکه بر روی داده تست را بررسی نمایید

۲. بررسی کنید چه شبکه هایی توسط $Sequential - API$ قابل پیاده سازی نیستند. برای پیاده سازی این شبکه ها چه راه حلی را پیشنهاد می دهید.