دانشگاه صنعتی امیرکبیر

(پلی‌تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش ششم درس داده کاوی محاسباتی

تجزیه تنسورها

نگارش:

پوریا علیمرادپور ۴۰۳۱۱۲۰۸۸

اساتید درس:

دکتر مهدی قطعی

بهنام یوسفی مهر

پاییز ۱۴۰۳

چکیده

الگوریتم Higher-Order Singular Value Decomposition (HOSVD) یک تکنیک قدرتمند برای تجزیه و تحلیل تنسورها ، به ویژه در کاربردهایی همچون داده‌کاوی و پردازش الگوها است. این الگوریتم به ما امکان می‌دهد تا ساختارهای پیچیده داده را در قالب تنسورها مدل‌سازی کرده و کاهش ابعاد مؤثری انجام دهیم.

در این گزارش، به بررسی کاربرد HOSVD در طبقه‌بندی دیتاست ارقام دست‌نویس پرداخته‌ایم. با اجرای مراحل مختلف الگوریتم و استفاده از مدل‌های طبقه‌بندی مبتنی بر ویژگی‌های استخراج‌شده، توانستیم دقت بالایی در طبقه‌بندی این دیتاست به دست آوریم.

واژه‌های کلیدی:

الگوریتم HOSVD، تنسور، طبقه‌بندی

فهرست مطالب

صفحه

چکیده ..... أ

فصل اول مقدمه ۳

فصل دوم پاسخ تمرین ۵

۲-۱- الگوریتم HOSVD ۶

۲-۲- طبقه‌بندی ارقام دست‌نویس با HOSVD ۷

فصل سوم جمع‌بندی و نتیجه‌گیری ۱۶

منابع و مراجع ۱۸

References ۱۸

فصل اول

مقدمه

با افزایش حجم و پیچیدگی داده‌های موجود در حوزه‌های مختلف همچون تصویربرداری، پردازش سیگنال، و داده‌کاوی، استفاده از روش‌های مؤثر برای کاهش ابعاد و استخراج ویژگی‌ها به یک نیاز اساسی تبدیل شده است. تحلیل و پردازش این داده‌ها به روش‌های مؤثری نیاز دارد که بتواند ساختارهای پنهان درون داده‌ها را شناسایی کند. یکی از روش‌های پیشرفته در این زمینه، تکنیک‌های تجزیه‌وتحلیل تنسورها است که به ما امکان می‌دهد داده‌های چند بعدی را مدل‌سازی و بررسی کنیم. یکی از ابزارهای قدرتمند در این زمینه، تجزیه مقادیر منفرد از بالاتر یا HOSVD است که یک تعمیم از تجزیه مقادیر منفرد (SVD) به فضای چندبعدی یا تنسورها به شمار می‌آید. برخلاف SVD که فقط برای ماتریس‌ها کاربرد دارد، HOSVD قادر به پردازش داده‌های با ابعاد بالاتر بوده و از آن در مسائل مختلفی از جمله فشرده‌سازی داده‌ها، کاهش ابعاد و طبقه‌بندی استفاده می‌شود.

روش HOSVD به طور خاص برای تجزیه تنسورها توسعه یافته و در آن، هر بُعد از تنسور به عنوان یک فضای مستقل در نظر گرفته می‌شود که این ویژگی به HOSVD اجازه می‌دهد تا ساختارهای پیچیده داده را حفظ کرده و ویژگی‌های کلیدی را به طور مؤثری استخراج کند. این روش در مقایسه با روش‌های دیگر، دارای قابلیت‌های بهینه‌تری برای نگهداری ساختار درونی داده‌ها بوده و به همین دلیل در مسائلی همچون تشخیص الگو و طبقه‌بندی، به عنوان یکی از روش‌های کارآمد مطرح است.

در این گزارش، ما به بررسی کاربرد HOSVD در طبقه‌بندی مجموعه داده ارقام دست‌نویس US Postal Service Dataset پرداخته و با طراحی یک طبقه‌بند مبتنی بر HOSVD، دقت این روش را در شناسایی و دسته‌بندی نمونه‌ها ارزیابی کرده‌ایم.

کدهای مربوط به این پروژه داخل محیط گوگل کولب به آدرس زیر در دسترس هستند:

https://colab.research.google.com/drive/1r7T_R4tqWo7rMZ3cP6EFaXm7jH__N6Lk?usp=sharing

فصل دوم

پاسخ تمرین

همانطور که قبل‌تر اشـــاره شـــد، در این تمرین به طبقه بندی دیتاســـت US Postal Service Dataset می‌پردازیم که شامل تصاویر ارقام دست‌نویس اسکن‌شده است و در سیستم‌های پردازش خودکار پست ایالات متحده برای شناسایی ارقام استفاده می‌شود. این دیتاست دارای نمونه‌های متنوعی از ارقام دست‌نویس است که شـامل نویســه‌های مختلف از افراد متعدد با دســت‌خط‌های متفاوت اســت و چالش‌های زیادی در زمینه تشخیص الگو و طبقه‌بندی دقیق فراهم می‌آورد.

این دیتاسـت در فرمت *HDF5* ارائه شـده و شـامل دو گروه اصـلی، یعنی *train* و *test* اسـت. هر گروه دارای دو بخش است: *data* و *target* . بخش *data* حاوی تصاویر ارقام دست‌نویس با ابعاد ۱۶×۱۶ پیکسل خاکستری است و بخش *target* نیز شامل برچسب‌های مرتبط با هر تصویر است که بیانگر رقم صحیح آن تصـویر می‌باشـد. این مجموعه شـامل ۷۲۹۱ تصـویر برای آموزش و ۲۰۰۷ تصـویر برای آزمایش اسـت که به منظور ارزیابی عملکرد مدل‌های طبقه‌بندی استفاده می‌شود.

در ادامه به توضـــیحات این الگوریتم طبق کتاب کاربرد تجزیه ماتریس در الگویابی (Eldén, 2019) و پیاده‌سازی آن می‌پردازیم.

۲-۱- الگوریتم HOSVD

فرض کنیم تنسور $A \in \mathbb{R}^{m\times n\times d}$ داریم؛ در این صورت می‌توان A را به صورت زیر نوشت:

$$A = S \times_1 U^{(1)} \times_2 U^{(2)} \times_3 U^{(3)}$$

که $U^{(1)} \in \mathbb{R}^{m\times m}$، $U^{(2)} \in \mathbb{R}^{n\times n}$ و $U^{(3)} \in \mathbb{R}^{d\times d}$ ماتریس‌هایی متعامد و S یک تانســور هســته[1] هم اندازه با A است به طوری که برش‌های آن در هر سه جهت بر هم عمودند.

$$\langle S(i,:,:), S(j,:,:)\rangle = \langle S(:,i,:), S(:,j,:)\rangle = \langle S(:,:,i), S(:,:,j)\rangle = 0 \ , i \neq j$$

مقادیر منفرد تانسور A صورت زیر تعریف می‌شوند:

[1] Core Tensor

$$\sigma_i^{(1)} = \|S(i,:,:)\|_F, \qquad i = 1, \dots, m \qquad (\sigma_1^{(1)} \geq \sigma_2^{(1)} \geq \cdots \geq \sigma_m^{(1)})$$
$$\sigma_i^{(2)} = \|S(:,j,:)\|_F, \qquad j = 1, \dots, n \qquad (\sigma_1^{(2)} \geq \sigma_2^{(2)} \geq \cdots \geq \sigma_n^{(2)})$$
$$\sigma_i^{(3)} = \|S(:,:,k)\|_F, \qquad k = 1, \dots, d \qquad (\sigma_1^{(3)} \geq \sigma_2^{(3)} \geq \cdots \geq \sigma_d^{(3)})$$

$U^{(1)}$، $U^{(2)}$ و $U^{(3)}$ از تجزیه SVD[1] مقادیر $unfold$ تانسورهای A بدست می‌آیند:

$$unfold_i(A) = A_{(i)} = U^{(i)}\Sigma^{(i)}\left(V^{(i)}\right)^T \qquad ,i = 1,2,3$$
$$S = A \times_1 \left(U^{(1)}\right)^T \times_2 \left(U^{(2)}\right)^T \times_3 \left(U^{(3)}\right)^T$$

۲-۲- طبقه‌بندی ارقام دست‌نویس با HOSVD

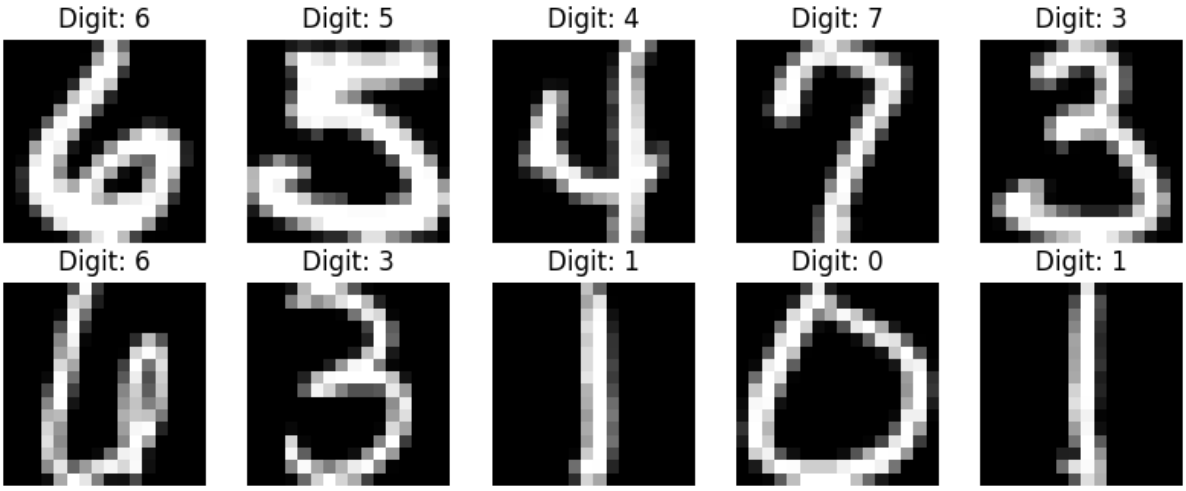
برای طبقه‌بندی مجموعه داده اشاره شده در فصل اول مقدمه، پس از فراخوانی کتابخانه‌های مورد نیاز، آنرا بارگزاری می‌کنیم و مشخصاتش را خروجی گرفته تا بررسی انجام دهیم:

```
[ ]  with h5py.File('/content/sample_data/usps.h5', 'r') as hf:
         train = hf.get('train')
         X_train = train.get('data')[:]
         y_train = train.get('target')[:]
         test = hf.get('test')
         X_test = test.get('data')[:]
         y_test = test.get('target')[:]

     # Print dataset infos
     print("Training set shape:", X_train.shape)
     print("Testing set shape:", X_test.shape)
     print("Each image shape:", X_train[0].shape)
     print("Labels of training data:", set(y_train))

     # Print number of samples for each digit in a table-like format
     print("Digit | Count \t| %")
     print("--------------------------")
     for digit in range(10):
         print(f"{digit}     | {np.sum(y_train == digit)} \t| {np.sum(y_train == digit) / len(y
```

[1] Singular Value Decomposition

```
Training set shape: (7291, 256)
Testing set shape: (2007, 256)
Each image shape: (256,)
Labels of training data: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Digit | Count   | %
-------------------------
0     | 1194    | 16.38%
1     | 1005    | 13.78%
2     | 731     | 10.03%
3     | 658     | 9.02%
4     | 652     | 8.94%
5     | 556     | 7.63%
6     | 664     | 9.11%
7     | 645     | 8.85%
8     | 542     | 7.43%
9     | 644     | 8.83%
```

و همچنین چند نمونه اول آنرا همراه با لیبل (برچسب) بر روی نمودار می‌بریم:



طبق بخش ۲.۳.۱ مطرح شده در مقاله ارسالی همراه تمرین، داده‌های تمرینی را طوری کنارهم قرار می‌دهیم تا یک تنسور سه بعدی تشکیل شود:

3.2.1. Training phase

First we build a tensor with all the digits in the training set. The data can be visualized as in Figure 6. All the digits are reshaped into vectors in $\mathbb{R}^{400}$. They are sorted so that every slice contains digits for one class only.

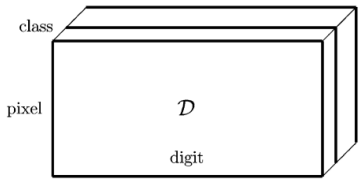


Fig. 6. The tensor with all the digits. The dimensions are 400 for the pixel mode, 10 for the class mode and approximately 1000 for the digit mode.

بدین ترتیب یک تانسور با ابعاد Final tensor shape: (256, 1000, 10) داریم که باید HOSVD آنرا محاسبه و برای مدلمان استفاده کنیم. (لازم به ذکر است که در مقاله برای پیش‌پردازش، فیلتر گاوسی بر روی عکس اعمال کرده اند که باعث شده ابعاد آن ۲۰×۲۰ باشد اما ما از تصاویر بدون تغییر با ابعاد ۱۶×۱۶ استفاده کرده‌ایم.)

```python
# Parameters
num_classes = 10           # 10 digit classes (0 to 9)
pixels_per_image = 256     # Each image is 16x16, flattened to 256
samples_per_class = 1000   # Desired number of samples per class for consistency

# Initialize empty list to hold samples of each class
class_samples = []

# Loop through each digit (0 to 9)
for digit in range(num_classes):
    # Select samples belonging to the current digit
    digit_data = X_train[y_train == digit]
    # If we have more samples than needed, truncate
    if len(digit_data) > samples_per_class:
        digit_data = digit_data[:samples_per_class]
    # If we have fewer samples, pad by repeating some samples
    elif len(digit_data) < samples_per_class:
        repeat_times = samples_per_class // len(digit_data) + 1
        digit_data = np.tile(digit_data, (repeat_times, 1))[:samples_per_class]

    # Append to the list
    class_samples.append(digit_data)

# Stack all classes along the "digit" mode to create the final tensor
tensor_data = np.stack(class_samples, axis=1)  # Shape: (1000, 10, 256)
tensor_data = tensor_data.transpose(2, 0, 1)   # Rearrange to shape: (256, 1000, 10)

print("Final tensor shape:", tensor_data.shape)
```

حال به پیاده‌سازی طبقه‌بند تجزیه مقادیر منفرد ابعاد بالاتر[1] می‌پردازیم:

[1] HOSVD Classifier

```python
class HOSVDClassifier:
    """
    A classifier implementing the HOSVD (Higher-Order Singular Value Decomposition)
    algorithm for digit classification.

    Parameters
    ----------
    image_size : tuple, default=(16, 16)
        The dimensions to reshape the input vectors into.
    n_components : int, default=None
        Number of components to keep after SVD. If None, keep all components.
    """
    def __init__(self, image_size=(16, 16), n_components=None):
        self.image_size = image_size
        self.n_components = n_components
```

با ایجاد یک instance از این کلاس، ورودی ابعاد تصاویر مدنظر و تعداد کامپوننت‌های مدنظرمان برای نگهداری و فشرده‌سازی را در مدل فیکس می‌کنیم.

دو متد اصلی این کلاس `fit(X_train, y_train)` و `predict(X_test)` هستند که مابقی متدها برای انجام این دو مرحله نوشته شده‌اند. بدین صورت که متد فیت، پارامترهای مدل که شامل S, U, V, W که تجزیه HOSVD تنسور است را تنظیم کرده و متد پردیکت از آن پارامترها برای بررسی باقیمانده تصویر جدید و تصویر بازسازی شده که در ادامه توضیح می‌دهیم استفاده می‌کند.

```python
# Train classifier
hosvd = HOSVDClassifier(n_components=64) #
hosvd.fit(tensor_data, y_train)
```

```
HOSVDClassifier(image_size=(16, 16), n_components=64)
```

```python
# Test classifier
y_pred = hosvd.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print(f"\nAccuracy: {accuracy:.3f}")
```

```
[■■■■■■■■■■■■] 100.00%
Accuracy: 0.903
```

فرایند تنظیم پارامترهای مدل زمان بسیار کمی می‌گیرد اما عمل پیش‌بینی به دلیل اینکه باید بر روی تک تک ۲۰۰۷ داده تست انجام شود مقداری زمان‌بر خواهد بود. اما در نهایت مدلی با دقت بالا (۹۰.۳٪) داریم.

در مقاله (جدول ۱) ذکر شده که با n_components=64 فشرده‌سازی ۹۷درصدی برروی داده‌ها اعمال شده و خطای نرمی معادل ۵.۳۳ داشته.

همانطور که در تصویر مشخص است متد fit() تنها برچسب‌های خاص که مدل باید عکس‌های رقم جدید را به آنها اختصاص دهد مشخص کرده و سپس از تابع محاسبه پایه‌های استفاده می‌کند و چیزی باز نمیگرداند.

```
[ ]     def _compute_basis(self, digit_tensor):
            """Compute HOSVD basis matrix for a digit class."""
            # SVD on each unfolding
            U1, _, _ = svd(self._unfold_tensor(digit_tensor, 1), full_matrices=False)
            U2, _, _ = svd(self._unfold_tensor(digit_tensor, 2), full_matrices=False)
            U3, _, _ = svd(self._unfold_tensor(digit_tensor, 3), full_matrices=False)

            S = self._mode_n_product(digit_tensor, U1.T, 1)
            S = self._mode_n_product(S, U2.T, 2)
            S = self._mode_n_product(S, U3.T, 3)

            # Keep only the first n_components
            if self.n_components is not None:
                U1 = U1[:, :self.n_components]
                U2 = U2[:, :self.n_components]
                U3 = U3[:, :self.n_components]
                S = S[:self.n_components, :self.n_components, :]

            # Normalize the basis matrix
            U1 = normalize(U1, axis=0)
            U2 = normalize(U2, axis=0)
            U3 = normalize(U3, axis=0)

            return [U1, U2, U3], S

        def fit(self, X, y):
            """Fit the HOSVD classifier."""
            self.classes_ = np.unique(y) # Unique classes
            self.basis_ = self._compute_basis(X) # Compute bases
```

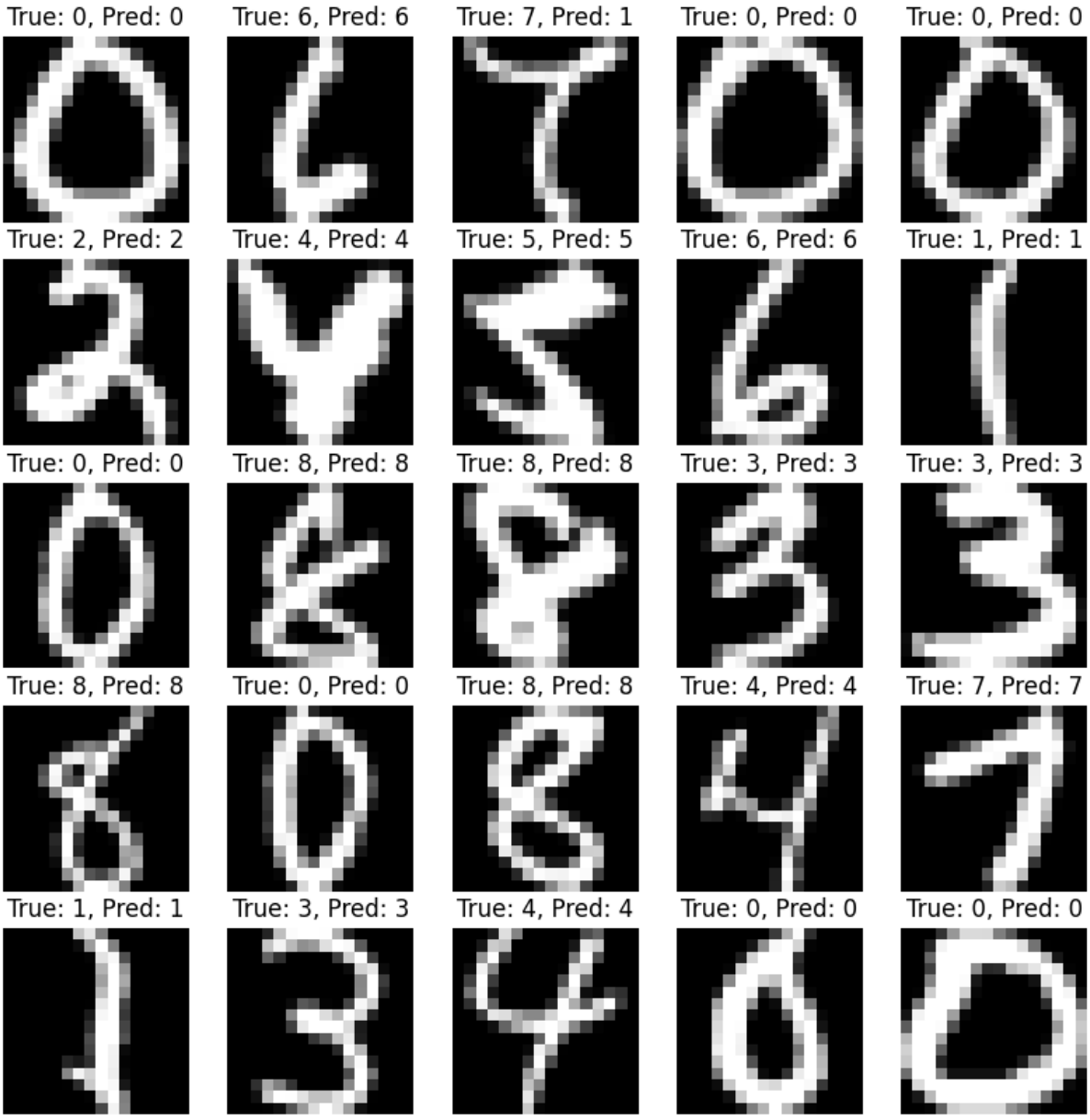
که مراحل محاسبه، در الگوریتم HOSVD شرح داده شد. ضرب مد-n یک تنسور و یک ماتریس هم اگر بخواهیم توضیح مختصری دهیم، با نماد $\times_n$ نمایش داده می‌شود و حاصلضرب ماتریس در اعضای unfold شده‌ی تنسور است که جزئیات هم در جزوه و هم در پیاده‌سازی یکسان است و خروجی آنها تست شده.

متد پیش‌بینی برداری ۱۰تایی از متد ()_compute_residuals دریافت کرده و آرگومان مینیمم آنرا پیدا نموده و به عنوان لیبل پیش‌بینی شده عکس باز می‌گرداند.

```python
def _compute_residual(self, digit_matrix, basis):
    """Compute residual between a digit tensor and its basis."""
    [U, V, W], S = basis

    C = self._mode_n_product(self._mode_n_product(S, U, 1), V, 2)
    C_pseudo_inv = np.stack([np.linalg.pinv(C[:, i, :])
                             for i in range(C.shape[1])], axis=1)
    projected_tensor = C_pseudo_inv @ digit_matrix

    residual = np.linalg.norm(W[:, :, np.newaxis] - projected_tensor, ord=2, axis=1)
    return residual

def predict(self, X):
    """Predict digit classes for samples in X."""
    # Make predictions
    predictions = []
    for i, x in enumerate(X):
        # Compute residuals for each class
        residual = self._compute_residual(x, self.basis_)
        min_index = np.unravel_index(np.argmin(residual), residual.shape)
        pred = self.classes_[min_index[0]]
        predictions.append(pred)
        # Show progress bar
        progress = (i + 1) / len(X) * 100
        bar_length = 10
        filled_length = int(bar_length * progress // 100)
        bar = '■' * filled_length + '□' * (bar_length - filled_length)
        print(f'\r[{bar}] {progress:.2f}%', end='', flush=True)
```

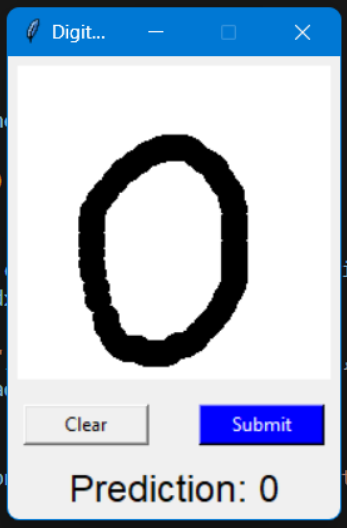
چند نمونه رندوم از دیتاست آزمایشی را همراه با برچسب پیش‌بینی شده خروجی می‌گیریم که خروجی مطلوبی به ما داده است:

```python
[ ] # Show random samples from the test set with their predictions
    plt.figure(figsize=(10, 10))
    for i in range(25):
        idx = np.random.randint(len(X_test))
        plt.subplot(5, 5, i + 1)
        plt.imshow(X_test[idx].reshape(16, 16), cmap='gray')
        plt.title(f"True: {y_test[idx]}, Pred: {y_pred[idx]}")
        plt.axis('off')
```

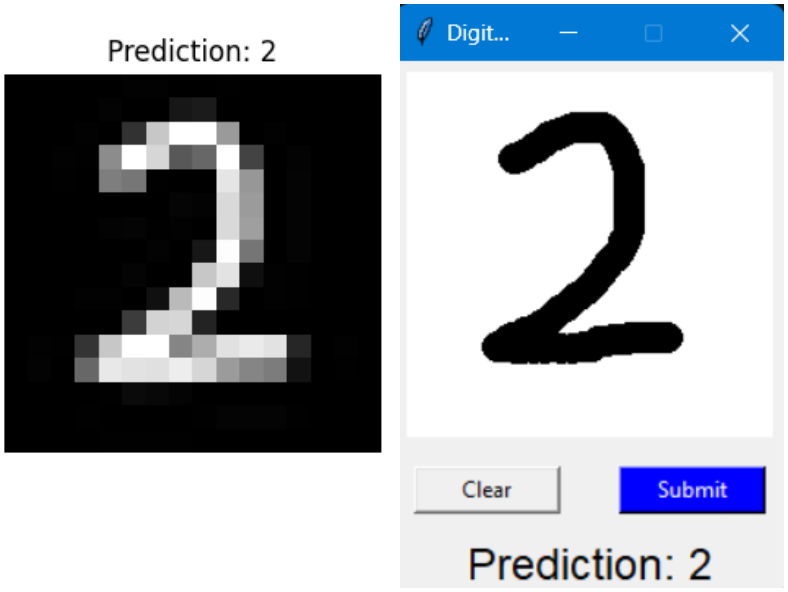


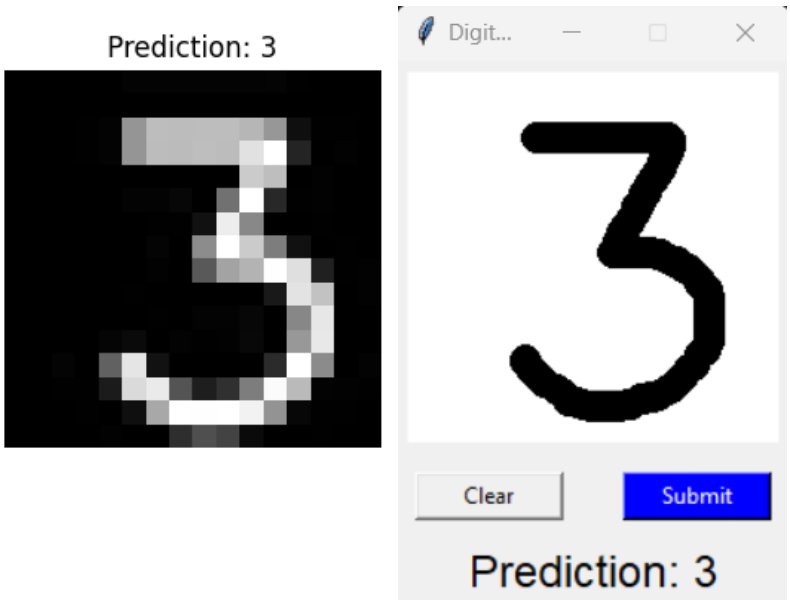
برای جذابیت کار، یک فضای interactive برای نوشتن یک رقم به کمک کتابخانه tkinter ایجاد کردیم که در محیط کولب به دلیل عدم وجود Display قابل اجرا نبود اما خروجی آن به شرح زیر است:

```python
class DigitRecognizerApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Digit Recognizer")

        self.canvas = tk.Canvas(self.master, width=200, h
        self.master.resizable(False, False)
        self.canvas.grid(row=0, column=0, pady=4, padx=4)
        self.canvas.bind("<B1-Motion>", self.paint)

        self.btn_clear = tk.Button(master, text="Clear",
        self.btn_clear.grid(row=1, column=0, pady=10, pad

        self.btn_submit = tk.Button(master, text="Submit"
        self.btn_submit.grid(row=1, column=0, pady=10, pa

        self.prediction_text = tk.StringVar()
        self.label = tk.Label(master, textvariable=self.p
        self.label.grid(row=3, column=0, pady=2)

        self.image = Image.new("L", (200, 200), color=255)
        self.draw = ImageDraw.Draw(self.image)

    def paint(self, event):
        x, y = event.x, event.y
        r = 8
        self.canvas.create_oval(x-r, y-r, x+r, y+r, fill='black')
        self.draw.ellipse([x-r, y-r, x+r, y+r], fill='black')

    def clear_canvas(self):
        self.canvas.delete("all")
        self.image = Image.new("L", (200, 200), color=255)
        self.draw = ImageDraw.Draw(self.image)
        self.prediction_text.set("")

    def predict_digit(self):
        img_resized = self.image.resize((16, 16), Image.LANCZOS)
        img_resized = ImageOps.invert(img_resized)
        img_array = np.array(img_resized) / 255.0
        img_flatten = img_array.flatten().reshape(16, 16)

        prediction = hosvd.predict(img_flatten.flatten().reshape(1, -1))[0]
        self.prediction_text.set(f"Prediction: {prediction}")

        plt.figure(figsize=(3, 3))
        plt.axis('off')
        plt.title(f"Prediction: {prediction}")
        plt.imshow(img_flatten, cmap='gray')

root = tk.Tk()
app = DigitRecognizerApp(root)
root.mainloop()
```



و برای هر رقم:





و به همین ترتیب می‌توایند در پنجره باز شـــده و ترســـیم عدد و انتخاب گزینه Submit پیش‌بینی مدل را مشاهده نمایید.

فصل سوم

جمع‌بندی و نتیجه‌گیری

بررســـی نتایج این گزارش نشـــان می‌دهد که HOSVD به عنوان یک ابزار تجزیه‌ای می‌تواند در کاهش پیچیدگی محاســـباتی و بهبود عملکرد طبقه‌بندی در بحث تنســـورها که داده‌های با ابعاد بالاتری نســـبت به ماتریس‌ها را نگه‌داری می‌کنند، مؤثر باشد.

این مطالعه نشـــان داد که HOSVD چگونه می‌تواند به کاهش ابعاد داده‌ها و حفظ اطلاعات اصـــلی کمک کرده و با بهره‌گیری از ویژگی‌های استخراج شده، طبقه‌بندی با دقت و کارایی را امکان‌پذیر سازد.

منابع و مراجع

References

Eldén, L. (2019). *Matrix methods in data mining and pattern recognition.* Society for Industrial and Applied Mathematics.