گزارش پروژه سوم درس سیگنال سیستم

پریا پاسه‌ورز 810101393

کوثرشیری جعفرزاده 810101456

بخش اول

تمرین 1-1)

در این قسمت با کمک cell، یک mapset ساختیم که سطر اول حروف و کاراکترهای مدنظرمان مشخص شده‌اند و در سطر دوم، با کمک dec2bin، یک عدد باینری را به آنها map کرده‌ایم.

کد:

```
% CREATING THE MAPSET
Nch=32;
mapset=cell(2,Nch);
Alphabet ='abcdefghijklmnopqrstuvwxyz .,!";';
for i=1:Nch
    mapset{1,i}=Alphabet(i);
    mapset{2,i}=dec2bin(i-1,5);
end
```

تمرین 1_2)

اول عکسی که می‌خواهیم پیام را درونش مخفی کنیم، لود کرده و و آن را سیاه سفید می‌کنیم. سپس پیامی که می‌خواهیم code کنیم را مشخص می‌کنیم و آنها را به تابع coding ورودی می‌دهیم.

کد:

```
X=imread('flower.jpg');
X=rgb2gray(X);

message='signal;';

Y=coding(message, X, mapset);
```

تصویر:

این تابع در بخش اول، هر حرف message ورودی را در نظر گرفته، index ای که آن کاراکتر در mapset در آن ذخیره شده را پیدا می‌کند و عدد باینری متناظرش را در یک آرایه ذخیره می‌کند.

سپس این ارایه از اعداد باینری را به ماتریس تبدیل می‌کند.

کد :

```
function encoded_image = coding(message, X, mapset)

message_len=length(message);
message_bin=cell(1,message_len);

for i=1:message_len
    ch=message(i);
    index=strcmp(ch,mapset(1,:));
    message_bin{i}=mapset{2,index};
end

binarymessage=cell2mat(message_bin);
binarymessage_len=length(binarymessage);
```

حالا باید بلوکی که می‌خواهیم پیام را در آن مخفی کنیم پیدا کنیم. برای این کار، واریانس هر بلوک 5*5 را محاسبه می‌کنیم و سپس بر حسب واریانس این بلوک‌ها را sort می‌کنیم. برای این کار از تابع blocking_image استفاده می‌کنیم. این تابع علاوه لیست واریانس‌های sort شده، لیستی از نقطه گوشه سمت چپ بالا این blockها را نیز برمی‌گرداند.

کد:

```
block_size = 5;
[sorted_coordinates, sorted_variances] = blocking_image(X, block_size);
```

داخل تابع، با توجه به مقدار block-size انتخاب شده و سپس با توجه به اندازه عکس‌ها، تعداد بلوک‌های مورد نیاز را محاسبه می‌کنیم.

یک لیست هم با مقدار 0 مقداردهی اولیه می‌کنیم تا واریانس‌ها را در آن ذخیره کنیم.

کد:

```
function [sorted_coordinates, sorted_variances] = blocking_image(img,block_size)

[img_height, img_width] = size(img);
num_blocks_y = img_height / block_size;
num_blocks_x = img_width / block_size;
variances = zeros(fix(num_blocks_y), fix(num_blocks_x));
```

حالا با توجه به شماره هر block و اندازه block-size، خطوط و ستون‌های هر بلوک را مشخص کرده و آن را جدا می‌کنیم. سپس با کمک تابع var، مقدار واریانسش را ذخیره می‌کنیم.

کد:

```
for i = 1:num_blocks_y
    for j = 1:num_blocks_x
        row_start = (i-1) * block_size + 1;
        row_end = i * block_size;
        col_start = (j-1) * block_size + 1;
        col_end = j * block_size;
        block = img(row_start:row_end, col_start:col_end);

        variances(i, j) = var(double(block(:)));
    end
end
```

لیست واریانسها در حال حاضر یک آرایه دو بعدی است، آن را تبدیل به یک لیست کرده و سپس sort می‌کنیم.

کد:

```
variances_flat = variances(:);

[sorted_variances, sorted_indices] = sort(variances_flat, 'descend');
```

حالا می‌خواهیم مختصات نقطه گوشه سمت چپ بالا هر block را به ترتیب واریانس ذخیره کنیم. برای این کار از تابع in2sub استفاده می‌کنیم. این تابع با توجه با شماره بلاک در لست واریانس و تعداد کل بلاک‌هایی که واریانس برای آنها محاسبه شده، row و col را پیدا می‌کند.

حالا که شماره سطر و ستون را داریم، می‌توانیم با کمک block-size، مختصات گوشه چپ بالا را پیدا کنیم.

کد:

```
sorted_coordinates = zeros(length(sorted_indices), 2);
for k = 1:length(sorted_indices)
    [row, col] = ind2sub(size(variances), sorted_indices(k));
    sorted_coordinates(k, :) = [(col-1) * block_size + 1, (row-1) * block_size + 1];
end
end
```

حالا با در دست داشتن این اطلاعات، مقدار block-size و threshold را در تابع coding مشخص می‌کنیم.

کد:

```
encoded_image = X;

threshold = 0.5;
message_index = 1;
```

پیام را به ترتیب واریانس در blockها کدگذاری می‌کنیم، یعنی اول در بلاک‌هایی که تغییر رنگ در آنها بیشتر است. یک threshold هم مشخص می‌کنیم که حداقل مقدار برای واریانس blockها چقدر باید باشد.

سپس خط و ستون گوشه سمت چپ بالا block را از sorted_coordinates استخراج کرده و و با کمک آنها، بلوک مورد نظر را از عکس جدا می‌کنیم.

از خانه اول بلوک شروع می‌کنیم، عدد متناظر با grayscale آن را به باینری تبدیل می‌کنیم و بیت آخر آن را با بیت کنونی پیام جابجا می‌کنیم. آنقدر این کار را ادامه می‌دهیم تا یا پیام تمام شود یا تعداد خانه‌های بلوک کنونی. اگر تعداد خانه‌های بلوک تمام شد سراغ بلوک بعدی با بیشترین واریانس می‌رویم.

کد:

```
for k = 1:length(sorted_variances)
    try
        if(sorted_variances(k) < threshold)
            error("message length is bigger than pixel numbers")
        end
    catch ME
        exit
    end

    coords = sorted_coordinates(k, :);
    row_start = coords(2);
    col_start = coords(1);

    block = encoded_image(row_start:row_start+4, col_start:col_start+4);

    for i = 1:block_size
        for j = 1:block_size
            if message_index <= binarymessage_len
                pixel_value = block(i, j);
                pixel_bin = dec2bin(pixel_value);
                pixel_bin(end) = binarymessage(message_index);
                block(i, j) = bin2dec(pixel_bin);
                message_index = message_index + 1;
            else
                break;
            end
        end
    end

    encoded_image(row_start:row_start+4, col_start:col_start+4) = block;
    if message_index > binarymessage_len
        break;
    end
end
```

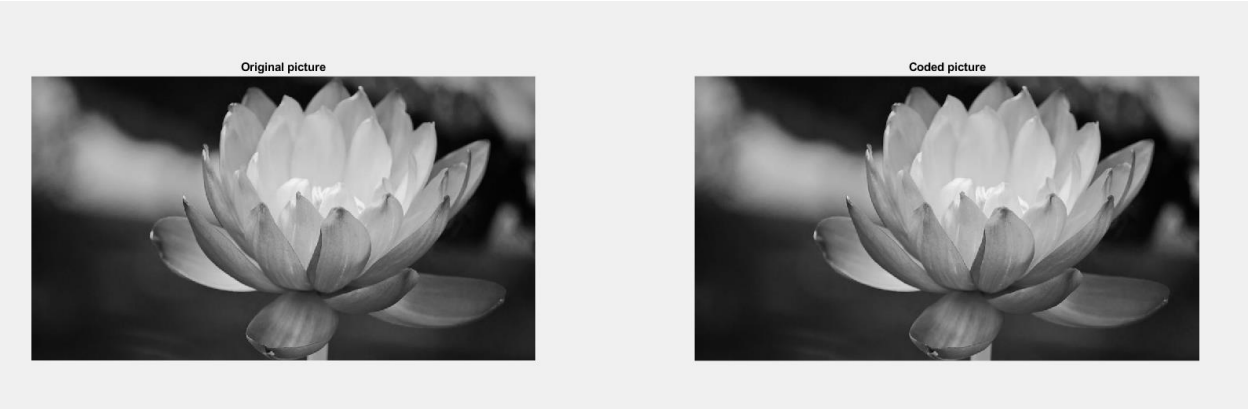
تمرین 1_3)

هم عکس سیاه سفید اولیه و هم عکس شده را رسم می‌کنیم.همانطور که مشاهده می‌کنید، تغییری در تصویر قابل مشاهده نیست، چون بیت‌های پیام در کم اهمیت‌ترین بیت هر پیام و در بلوکی که بیشترین تغییر رنگ در آن وجود دارد code شده اند.

کد:

```
subplot(1,2,1)
imshow(X)
title('Original picture')
subplot(1,2,2)
imshow(Y)
title('Coded picture')
```

خروجی:



تمرین 1_4)

در این بخش می‌خواهیم پس از کد کردن پیام درون تصویر، دوباره آن را بازیابی کنیم.

کد:

```
threshold = 0.5;
block_size = 5;
decoded_message = decoding(Y, mapset, threshold, block_size);
```

اول از طریق mapset داده شده، لیست حروف را می‌سازیم، سپس مشابه بخش coding، عکس را block می‌کنیم و آنها را بر حسب واریانس مرتب می‌کنیم.

متغییر flag مشخص خواهد کرد که کی پیام به اتمام رسیده است.

کد:

```
function DcodedMessageBin = decoding(X, mapset, threshold, block_size)

Alphabet = strjoin(mapset(1, :), '');

[sorted_coordinates, sorted_variances] = blocking_image(X, block_size);

flag=1;
DcodedMessageBin=[];
```

حال دقیقا برعکس عملیات coding عمل می‌کنیم، یعنی از بلاک با بیشترین واریانس شروع می‌کنیم و با توجه به سطر ستون نقطه سمت چپ بالایش، آن block را از عکس اصلی جدا می‌کنیم.

سپس مقدار موجود در هر پیکسل بلاک را به باینری تبدیل می‌کنیم و بیت آخر آن را به عنوان بیتی از پیام جدا می‌کنیم و ذخیره‌سازی می‌کنیم.

چون بلاک‌ها در اینجا 5*5 هستند و هر حرف نیز با یک رشته 5تایی از بیت‌ها code شده است، هر زمان یک سطر از block را خواندیم، کاراکتر متناظر با آن رشته را از Alphabet پیدا می‌کنیم و به message اضافه می‌کنیم.

هر زمان که با کاراکتر semicolon رسیدیم، یعنی پیام را کامل استخراج کرده‌ایم و عملیات متوقف می‌شود.

کد:

```
while flag
    message_index = 1;
    for k = 1:length(sorted_variances)
        if(sorted_variances(k) < threshold)
            break
        end
        coords = sorted_coordinates(k, :);
        row_start = coords(2);
        col_start = coords(1);

        block = X(row_start:row_start+(block_size-1), col_start:col_start+(block_size-1));

        for i = 1:block_size
            characterbin=zeros(1,block_size);
            for j = 1:block_size
                pixel_value = block(i, j);
                pixel_bin = dec2bin(pixel_value);
                characterbin(j)=str2double(pixel_bin(end));
                message_index = message_index + 1;
            end
            num=sum(characterbin.*(2.^(4:-1:0)))+1;
            DcodedMessageBin=[DcodedMessageBin Alphabet(num)];
            if strcmp(Alphabet(num),';')
                flag=0;
            end
            if flag == 0
                break;
            end
        end
        if flag == 0
            break;
        end

    end
```

خروجی:

```
>> pl
signal;
```

تمرین 1_5)

بستگی دارد این نویز در کجای تصویر اضافه شود. اگر در بخش شلوغی باشد که واریانس‌اش زیاد است، احتمال دارد پیام در آن بلوک code شده باشد و این نویز سبب شود بیتهایی که از آن بلوک استخراج می‌کنیم، با بیت‌های code شده تطابق نداشته باشند.

بخش دوم:

ایده کلی:

با توجه به راهنمایی انجام شده از template matching استفاده می کنیم و برخلاف قبل از normalize correlation استفاده می کنیم و برای حل مشکل چرخش عکس IC مربوطه بدین صورت عمل می کنیم که عکس را 180 درجه می گردانیم و دوباره normalize correlation می گیریم .

نکته مهم این است که همانند پروژه قبل نیاز داریم که برای این بخش یک threshold در نظر بگیریم تا مطمئن شویم نویز یا تصاویر غلط در اینجا حساب نشوند.

بعد از اینکه مقادیر بیشتر از threshold را حساب کردیم در قدم بعد مختصات نقطه بالا سمت چپ را پیدا کرده و از آن نقطه به اندازه طول و عرض IC یک مستطیل رسم می کنیم.

کد:

p2.m

```
1   ICrecognition('IC.png','PCB.jpg')
```

کد تابع:

```
function ICrecognition(IC_image,PCB_image)
pcb_image_rgb = imread(PCB_image);
ic_image = imread(IC_image);

pcb_image_gray = rgb2gray(pcb_image_rgb);
ic_image_gray = rgb2gray(ic_image);

ic_image_rotated = imrotate(ic_image_gray, 180);

result1 = normxcorr2(ic_image_gray, pcb_image_gray);
result2 = normxcorr2(ic_image_rotated, pcb_image_gray);

threshold = 0.5;
[y1, x1] = find(abs(result1) >= threshold);
[y2, x2] = find(abs(result2) >= threshold);

y_coords = [y1; y2];
x_coords = [x1; x2];
template_sizes = [repmat(size(ic_image_gray), length(y1), 1); repmat(size(ic_image_rotated), length(y2), 1)];

imshow(pcb_image_rgb); hold on;

for i = 1:length(y_coords)
    y_offset = y_coords(i) - template_sizes(i, 1);
    x_offset = x_coords(i) - template_sizes(i, 2);
    rectangle('Position', [x_offset, y_offset, template_sizes(i, 2), template_sizes(i, 1)], ...
              'EdgeColor', 'b', 'LineWidth', 2);
end

title('Detected ICs');
hold off;

end
```

نحوه عملکرد کد:

در ابتدا باید مطمئن شویم که تابع نوشته شده دو عکس را دریافت می کند.

سپس دو عکس داده شده را خوانده و در متغیر ها ذخیره می کنیم (با توجه به اینکه طبق خواسته سوال باید در نهایت نتیجه یافت شده برروی نسخه rgb مدار چاپ شود پس باید یک نسخه رنگی از مدار را تا انتها نگه داریم.)

در قدم بعدی نیاز است تا هر دو تصویر را از rgb به grayscale تبدیل کنیم این کار به دلیل این است که در ادامه برای correlation گرفتن نیازی به میانگین گیری از ابعاد rgb نباشد.

با توجه به اینکه ذکر شده است که IC ها تنها می توانند 180 درجه دوران داشته باشند پس کافیست عکس دیگری را تحت عنوان rotated picture ایجاد کنیم و عکس چرخانده شده را در آن قرار دهیم.

همانطور که در صورت پروژه راهنمایی شده بود از normalized cross correlation استفاده می کنیم این تابع تمامی مقادیر مربوطه را در result1 نگه می دارد، همین عمل را برای تصویر چرخانده شده نیز تکرار می کنیم و نتیجه را در result2ذخیره میکنیم.

نکته حائز اهمیت این است که لزوما تمامی مقادیر یافت شده در این بخش مد نظر ما نیستند و یک threshold تعیین می کند که آیا نتیجه داخل لیست یک ic را نشان می دهد یا نه . بدین منظور ما threshold را برابر 0.5 قرار می دهیم تا تنها مقادیری که بیش از این 0.5 هستند باقی مانده و بقیه حذف شود (این حد تعیین میکند که آیا شکل واقعا یک IC است یا اجزای دیگر مدار است که لحاظ شده است)

پس همانند آنچه گفته شد در result1,result2 به دنبال مقادیری می گردیم که از این حد تعیین شده بیشتر باشند پس از یافتن این مقادیر x,y آن ها را برمی گردانیم (ما در نظر داریم دور IC ها یک مستطیل آبی رنگ بکشیم پس ضروری است که مختصات آن ها را ذخیره کنیم)

حال که توانستیم تعداد IC ها را در هر دوحالت تشخیص دهیم این مختصات را باهم merge می کنیم تا نتیجه به دست آمده تعداد کل مستطیل هایی که باید رسم شود را نشان دهد.

نکته ی دیگر در این بخش این است که ما باید بتوانیم مختصات را با توجه به تصویرIC به روزرسانی کنیم برای این کار یک ماتریس تشکیل داده و ابعاد تصویر اولیه IC را یعنی (Hight, Width) را به تعداد IC هایی که یافتیم در آن قرار می دهیم.

در قدم بعدی عمل بالا را برای rotated IC نیز تکرار می کنیم و در همان ماتریس ذخیره می کنیم.

بنابراین درنهایت قرار است که یک ماتریس به ابعاد 2*(length y1+length y2) داشته باشیم که 2 در اینجا نمایانگر طول وعرض است .

به منظور رسم ابتدا تصور اصلی pcb_rgb را نمایش می دهیم تا بتوانیم برروی آن مستطیل های آبی را رسم کنیم.

همانطور که در توضیحات تابع normxcorr2 نوشته شده است این تابع نقطه پایین سمت راست را به عنوان مختصات باز می گرداند پس برای یافتن نقطه بالا سمت چپ نیاز است تا طول و عرض IC را از این نقطه کم کنیم و نقطه بالا سمت چپ را بدست بیاوریم.

با استفاده از نقطه ی چپ بالا مستطیل را به رنگ آبی رسم می کنیم (با توجه به داشتن این نقطه کافیست تنها طول و عرض IC را به تابع بدهیم و شرایط خط مثل رنگ و نوع آن را مشخص کنیم)

بعد ازآن که تمامی مستطیل هارا رسم کردیم از حلقه خارج شده و hold off را صدا زده و به اسکریپت اولیه باز می گردیم.

PCB:


BA3240
4T22
BA3240
4T22

IC:


BA3240
4T22

خروجی:

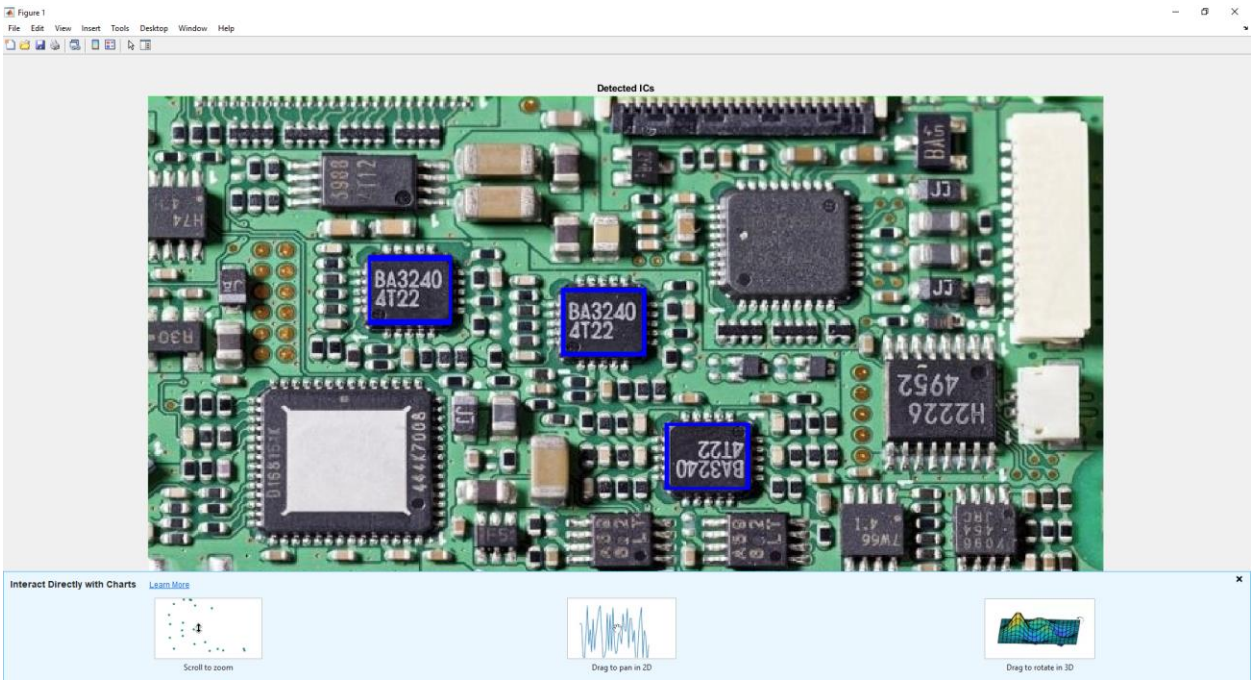


Figure 1
File Edit View Insert Tools Desktop Window Help
Detected ICs
BA3240
4T22
BA3240
4T22
Interact Directly with Charts
Learn More
Scroll to zoom
Drag to pan in 2D
Drag to rotate in 3D

بخش سوم:

در ابتدا همانند آنچه در دستور پروژه آورده شده است دیتاست را وارد workspace می کنیم.

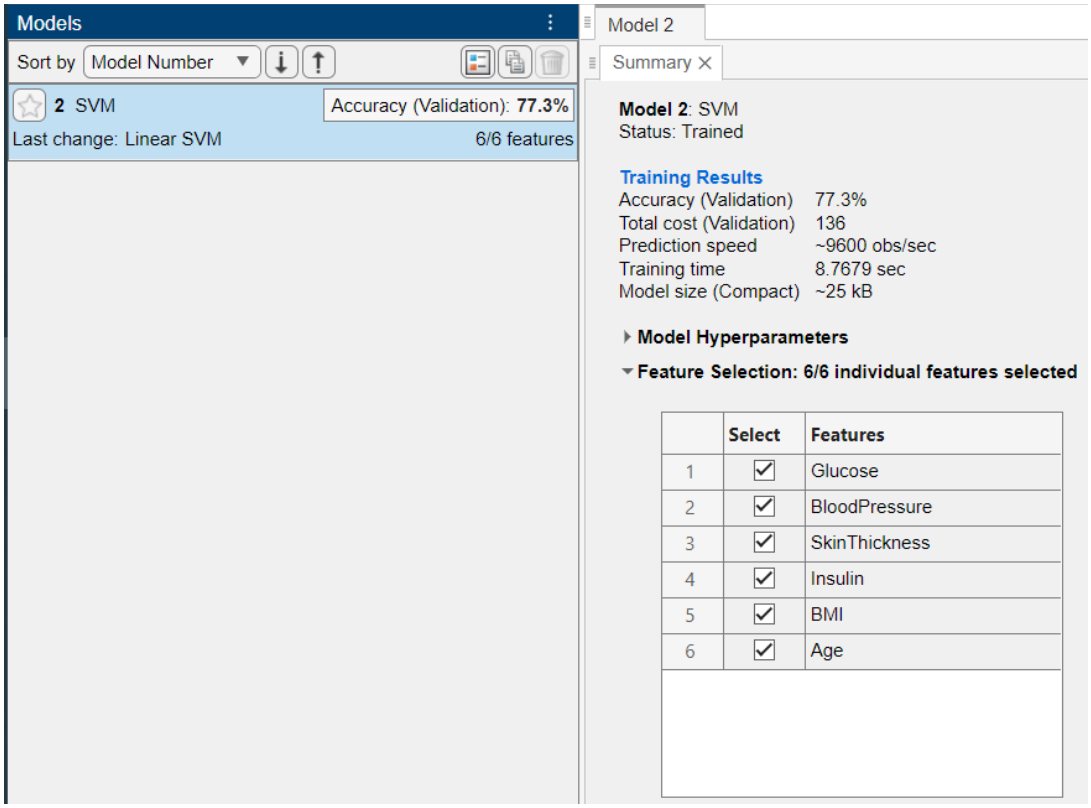
diabetestraining

600x7 table

	1 Glucose	2 BloodPressure	3 SkinThickness	4 Insulin	5 BMI	6 Age	7 label
1	148	72	35	0	33.6000	50	1
2	85	66	29	0	26.6000	31	0
3	183	64	0	0	23.3000	32	1
4	89	66	23	94	28.1000	21	0
5	137	40	35	168	43.1000	33	1
6	116	74	0	0	25.6000	30	0
7	78	50	32	88	31	26	1
8	115	0	0	0	35.3000	29	0
9	197	70	45	543	30.5000	53	1
10	125	96	0	0	0	54	1
11	110	92	0	0	37.6000	30	0
12	168	74	0	0	38	34	1
13	139	80	0	0	27.1000	57	0
14	189	60	23	846	30.1000	59	1
15	166	72	19	175	25.8000	51	1
16	100	0	0	0	30	32	1
17	118	84	47	230	45.8000	31	1
18	107	74	0	0	29.6000	31	1
19	103	30	38	83	43.3000	33	0
20	115	70	30	96	34.6000	32	1
21	126	88	41	235	39.3000	27	0
22	99	84	0	0	35.4000	50	0
23	196	90	0	0	39.8000	41	1
24	119	80	35	0	29	29	1
25	143	94	33	146	36.6000	51	1
26	125	70	26	115	31.1000	41	1

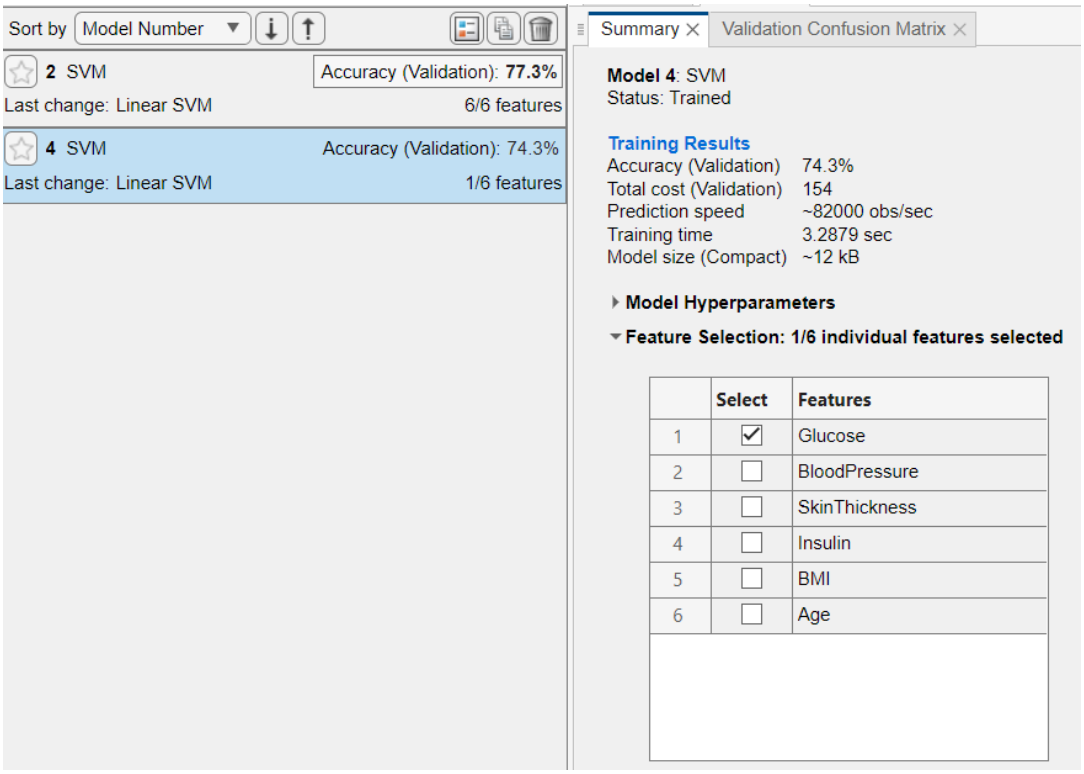
تمرین 3-1)

دقت به دست آمده برابر 77.3% می باشد.

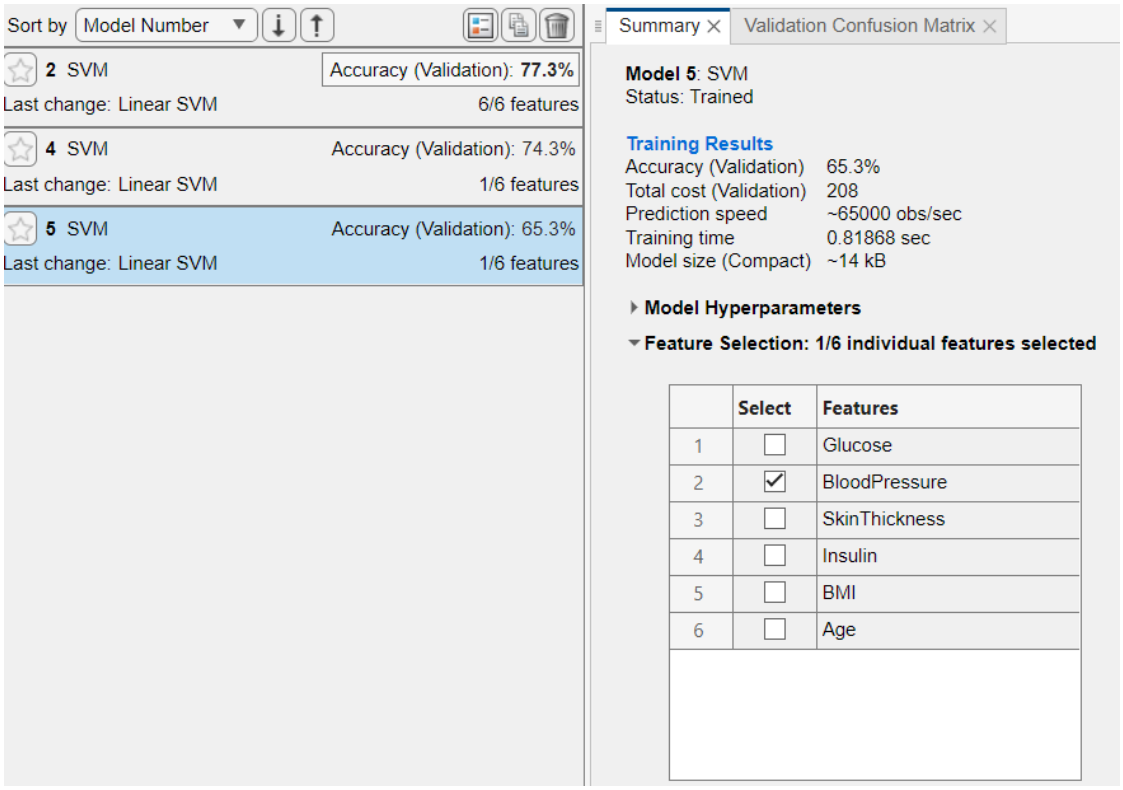


تمرین 3-2)

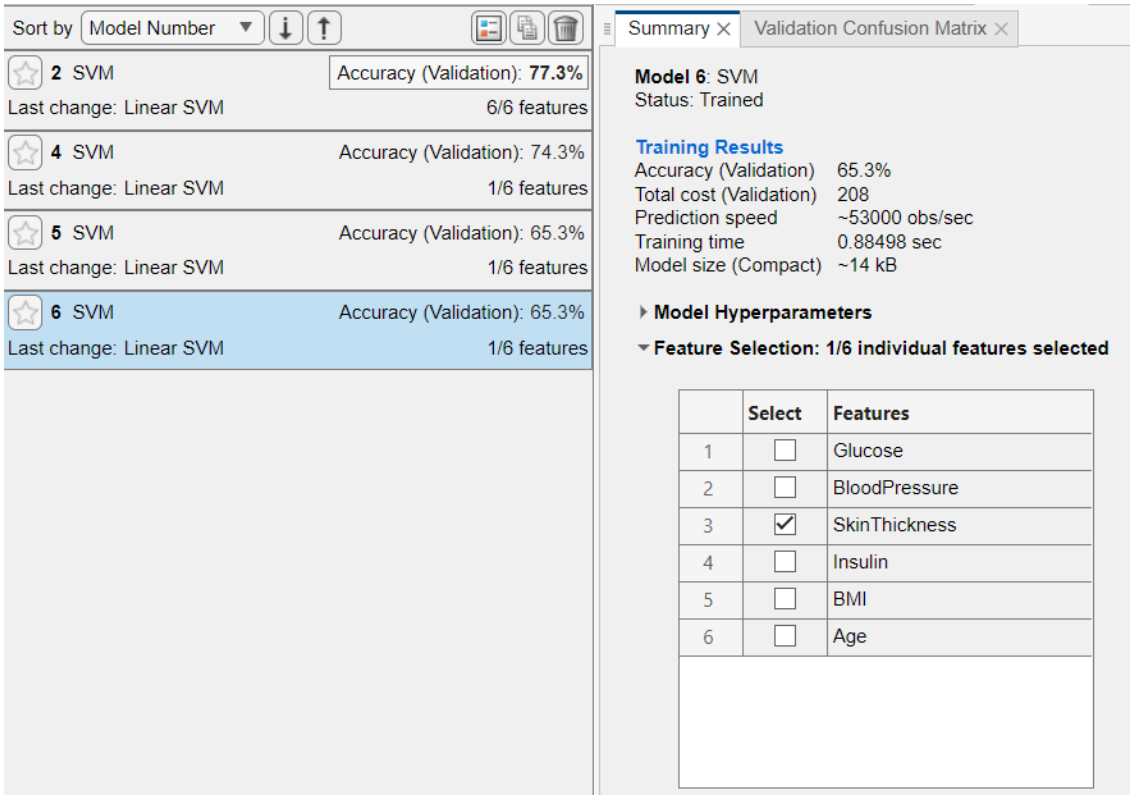
اگر تنها فیچر Glucose را نگه داریم دقت برابر 74.3 درصد می شود.



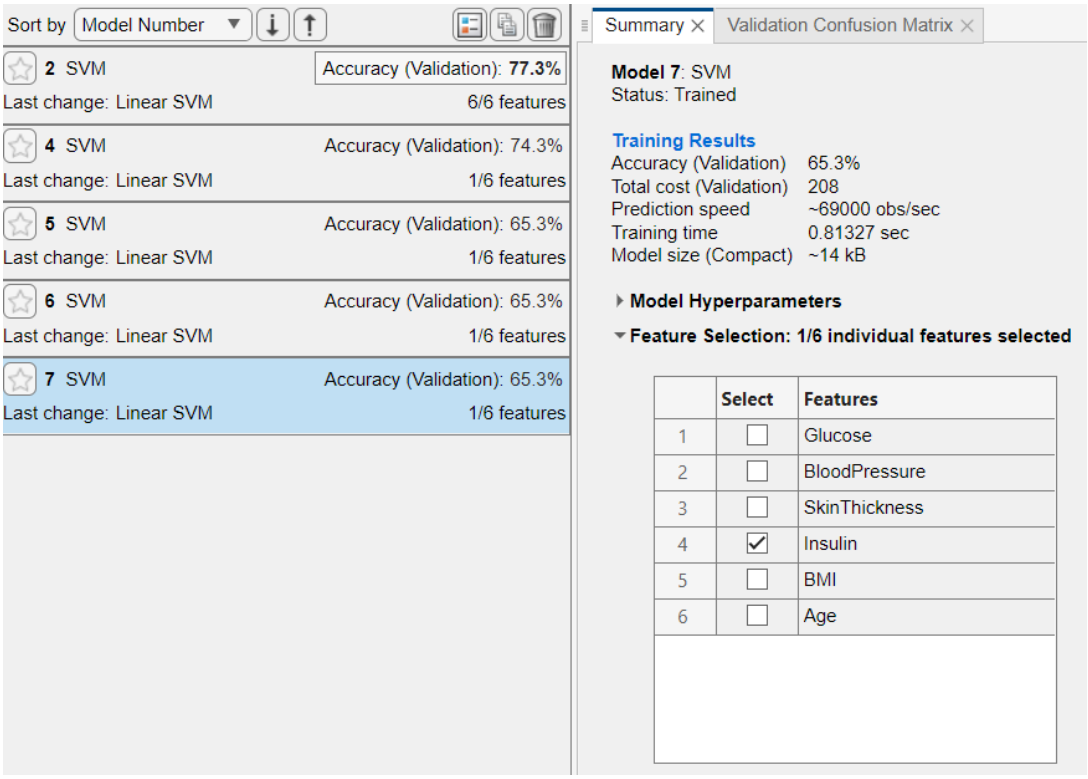
اگر تنها فیچر blood pressure را نگه داریم دقت برابر 65.3 درصد می شود.



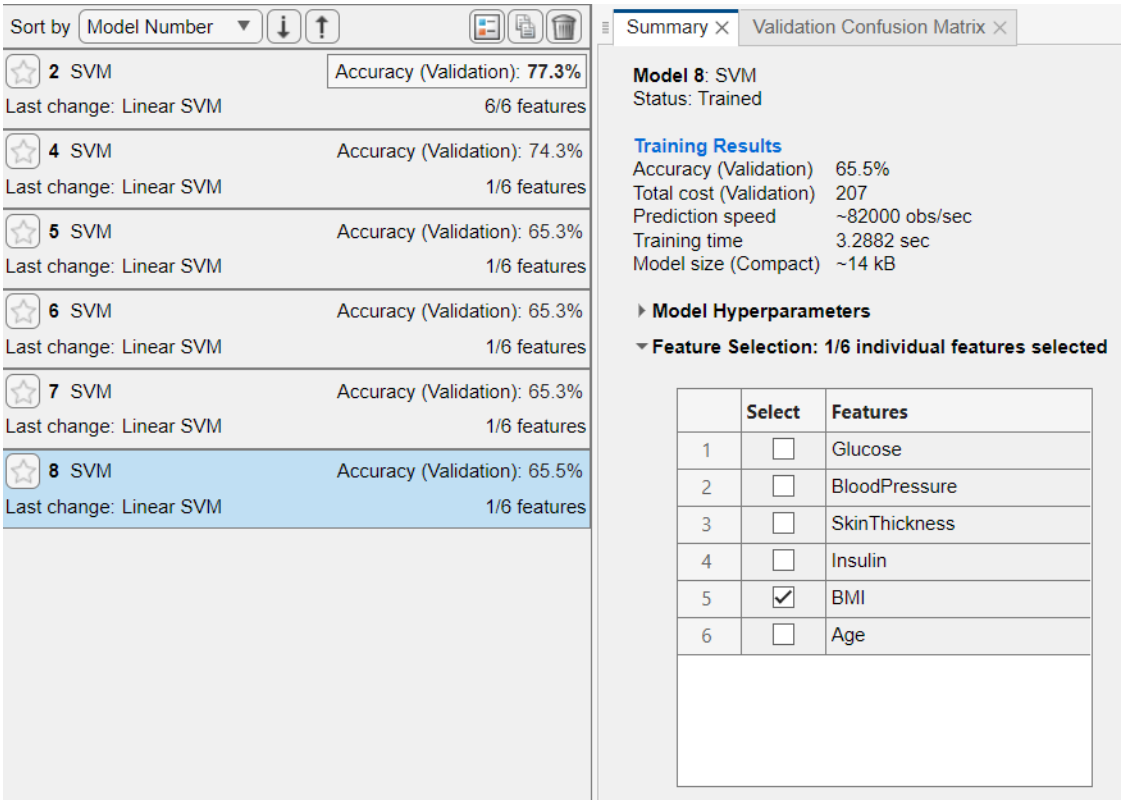
اگر تنها فیچر skin thickness را نگه داریم دقت برابر 65.3 درصد می شود.



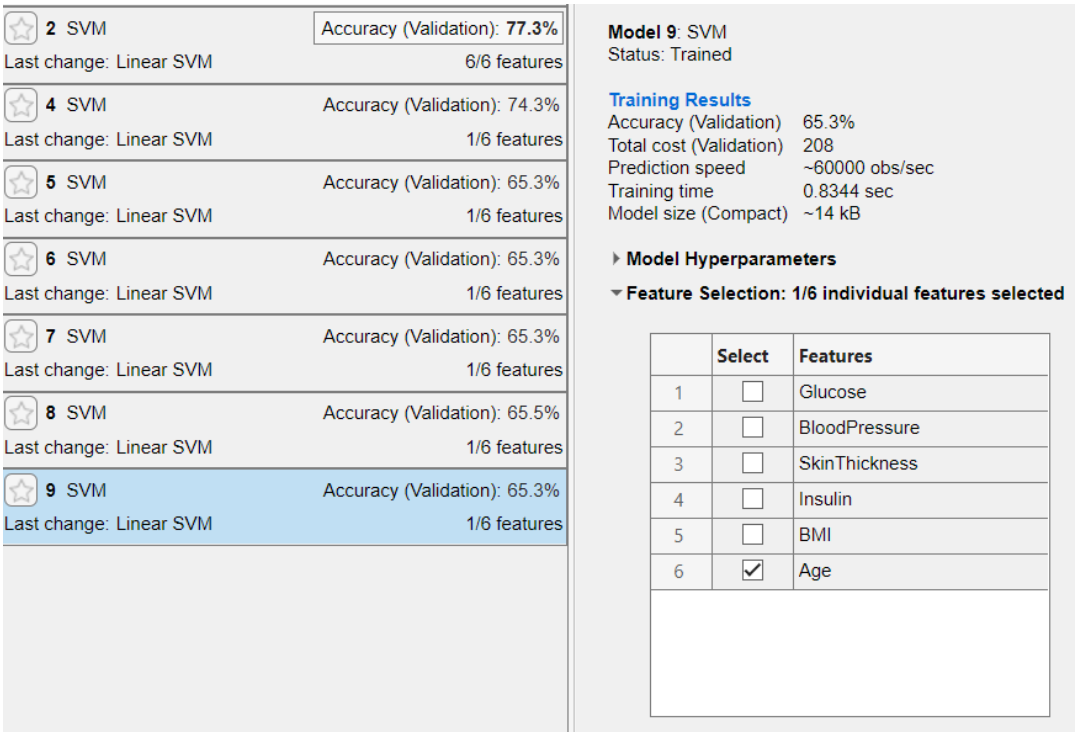
اگر تنها فیچر Insulin را نگه داریم دقت آن برابر 65.3 درصد می شود .



اگر فقط فیچر BMI را نگه داریم دقت آن برابر 65.5 درصد می باشد.



اگر فقط فیچر age را نگه داریم دقت برابر 65.3 درصد می شود.



باتوجه به میزان دقت محاسبه شده بیشترین تاثیر متعلق به Glucose و در قدم بعدی متعلق به BMI می باشد

پس از بیشترین تاثیر به کمترین به شرح زیر می باشد:

1- Glucose
2- BMI
3- Blood Pressure – Skin Thickness – Insulin – Age

تمرین 3-3)

برای محاسبه دقت از کد زیر استفاده می کنیم:

accuracy_train.m

```
dataset = readtable('diabetes-training.csv');
labels = dataset(:, end);
features = dataset(:, 1:end-1);
predictions = TrainedModel.predictFcn(features);
accuracy = mean(predictions == labels).*100;
disp(accuracy);
```

دقت محاسبه شده برابر است با 77.5 درصد است :

```
>> accuracy_train
    label
    _____

    77.5
```

همانطور که مشخص است دقت محاسبه شده با دقت گزارش شده در بالا مطابقت دارد .

تمرین 3-4)

از کد زیر برای این کار استفاده می کنیم:

accuracy_valid.m

```
1  dataset = readtable('diabetes-validation.csv');
2  labels = dataset(:, end);
3  features = dataset(:, 1:end-1);
4  predictions = TrainedModel.predictFcn(features);
5  accuracy = mean(predictions == labels).*100;
6  disp(accuracy);
```

دقت گزارش شده توسط این مدل برابر 78 درصد که دقت فاز تست است:

```
>> accuracy_valid
    label
    _____

     78
```