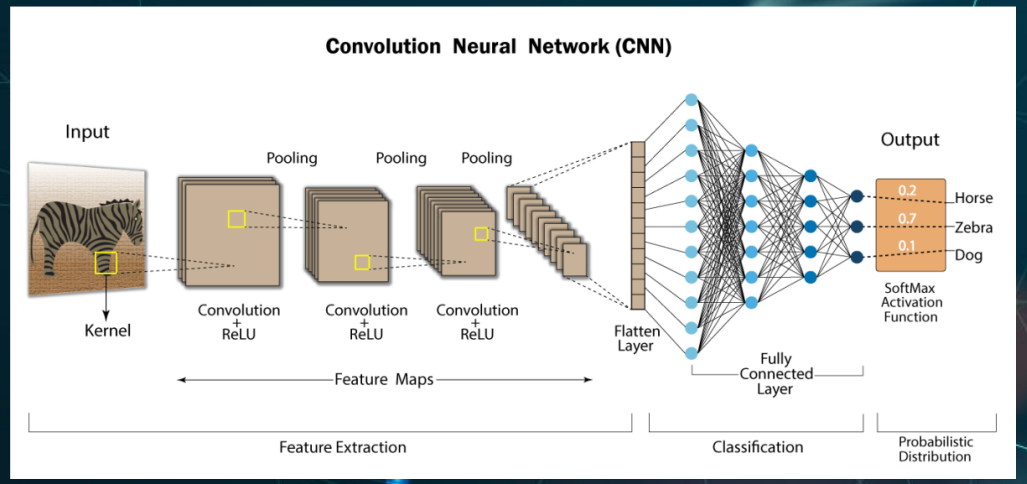
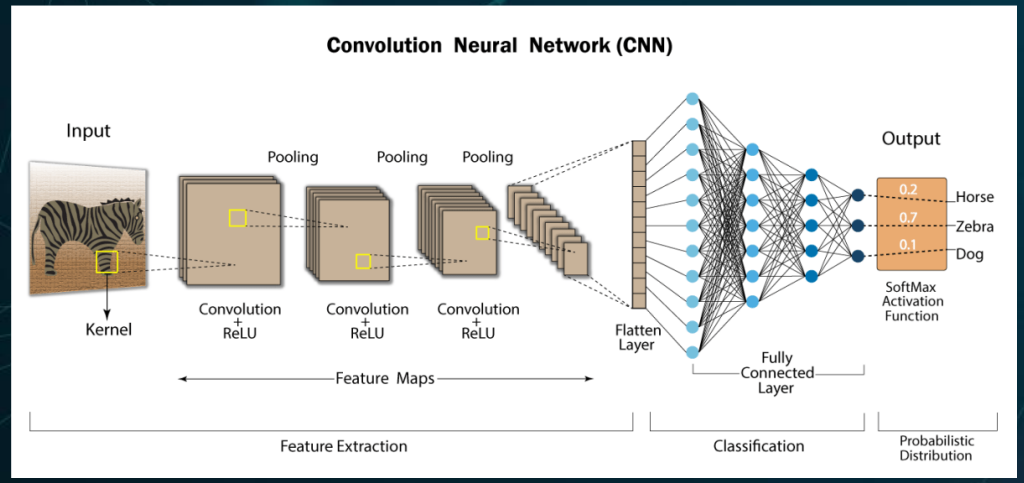
CNN – Convolution Neural Network

גלעד מרקמן

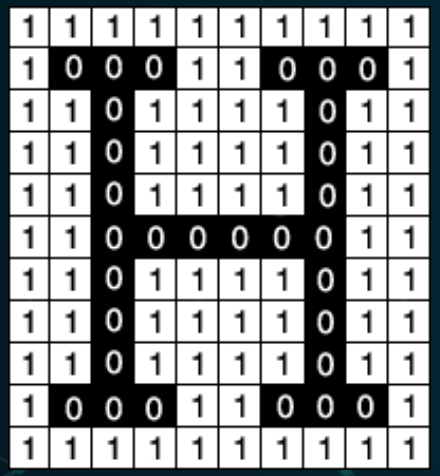


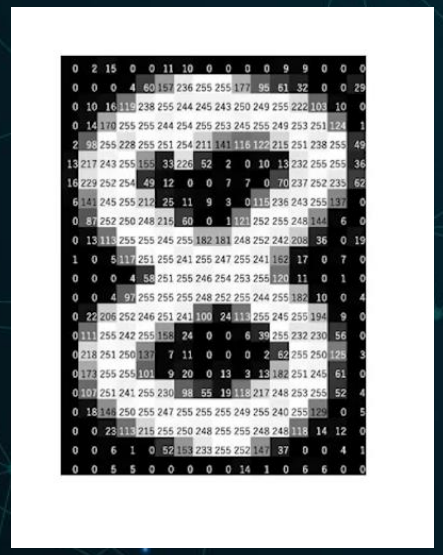
Convolution Neural Network

- רשת CNN הינה רשת נוירונים המבצעת את פעולת הקונבולוציה ומשמשת בעיקר לעיבוד תמונות.
- רשת CNN בנויה בתחילה משכבות קונבולוציה, ולאחר מכן מרשת נוירונים רגילה.
- בטרם נלמד מהי פעולת הקונבולוציה נסביר תחילה כיצד מוצגת תמונה בזכרון המחשב.
-



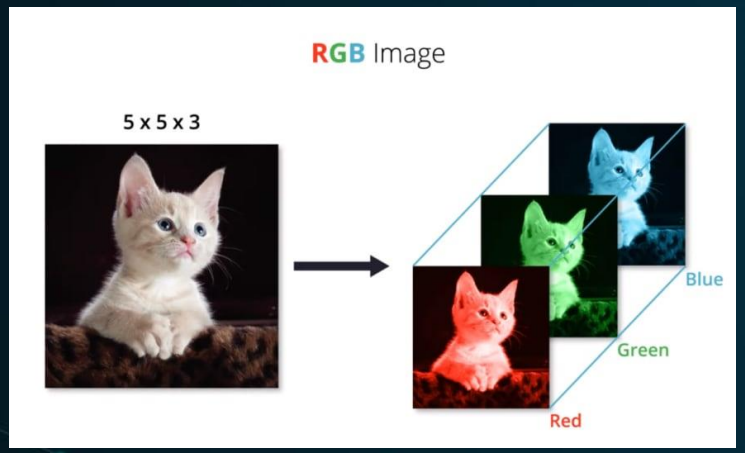
ייצוג תמונה בזכרון – שחור לבן

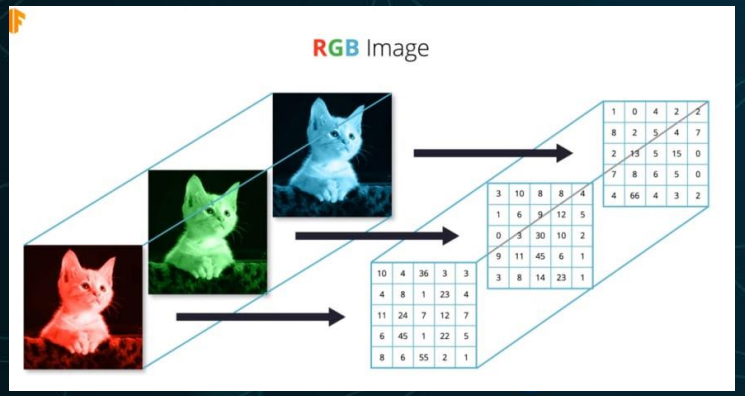




- תמונה מיוצגת בזיכרון המחשב על ידי מערך דו מימדי (מטריצה) של נקודות (פיקסלים).
- הרזולוציה היא למעשה המימדים של המערך. רזולוציה של 1024x768 מייצגת מערך דו מימדי של פיקסלים בגודל זה.
- תמונה שחור לבן תיוצג על ידי מטריצה של הערכים 0,1.
- תמונה עם גוונים של אפור תיוצג על ידי מטריצה של ערכים בין 0-255. שחור - 0; לבן – 255. וכל מספר ביניהם מייצג גוון של אפור.

יצוג תמונה צבעונית בזכרון - RGB

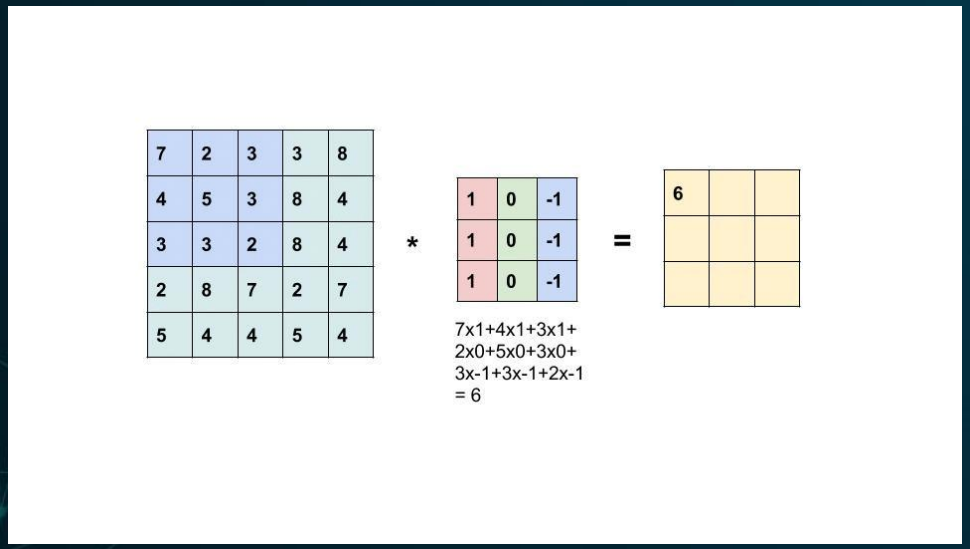


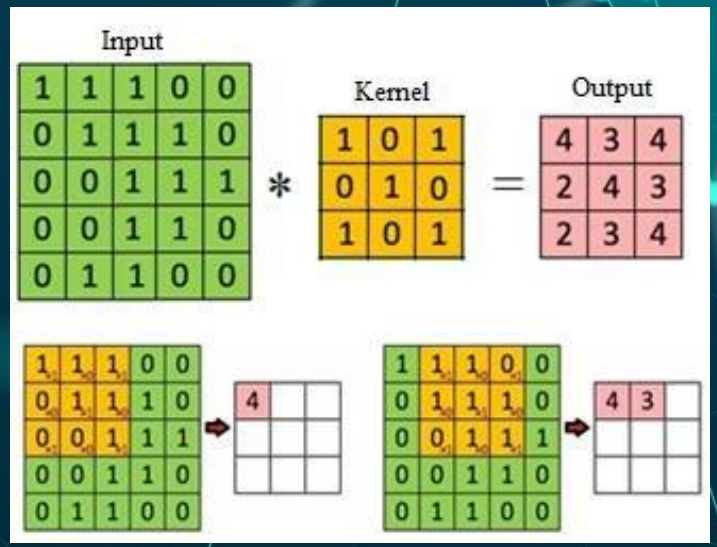


- תמונה צבעונית מיוצגת על ידי מערך של שלושה מימדים, שכל תא מכיל מספר שלם בין 0 – 255.
- למעשה מדובר בשלוש מטריצות של מספרים שלמים 0-255, המייצגים את הגוונים אדום, ירוק וכחול.
- כל נקודה (פיקסל) אם כן, בנויה משלושה ערכי RGB. שחור – (0,0,0). לבן (255,255,255). כל שאר הגוונים מהווים קומבינציה שלש שלושת המספרים הללו.
- את המימד של המטריצות (3) אנחנו מכנים channels.

קונבולוציה תמונות שחור לבן – 2D

- פעולת הקונבולוציה היא למעשה מכפלה סקלרית בין מטריצות בעלות אותן מימדים.
- כופלים כל איבר ממטריצה א' בתא המקביל לו במטריצה ב' וסוכמים את כל התוצאות.
- התמונה היא ה input והפילטר הוא kernel.

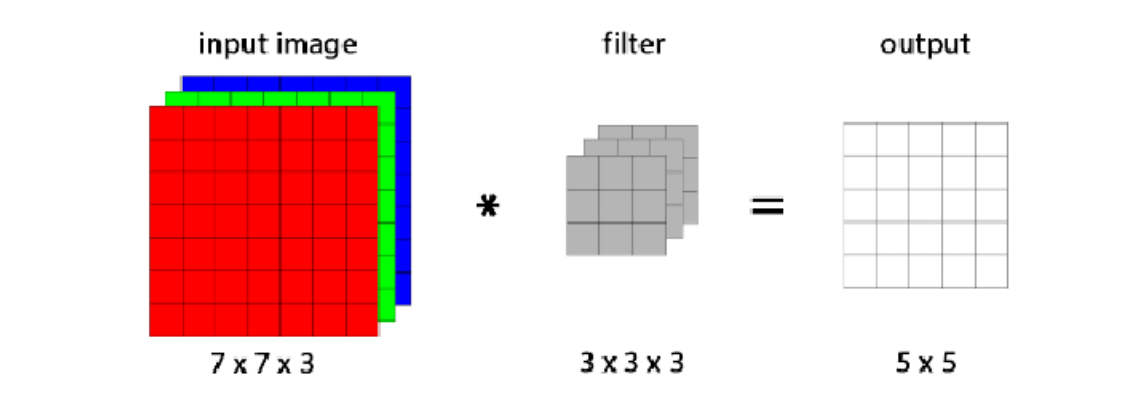




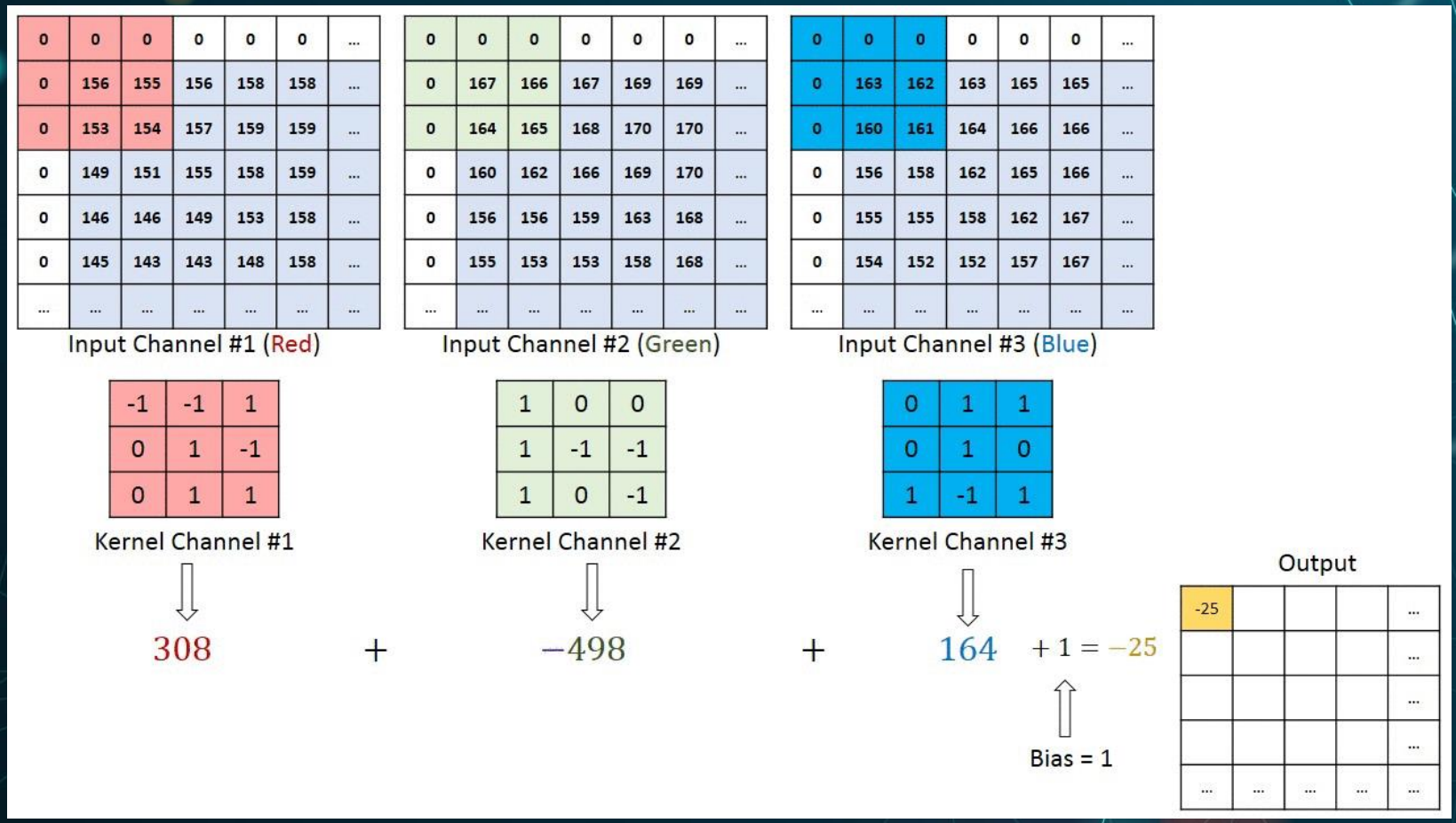
- הפעולה היא מחזורית: מתחילי בפינה שמאלית עליונה של התמונה, ומזיזיפ את הפילטר תחילה ימינה ואחר כך למטה מספר צעדים קבוע.

קונבולוציה של תמונות צבעוניות – 3D

- קונבולוציה של תמונה צבעונית נעשית באמצעות Kernel הכולל מספר מטריצות כמספר הערוצים (channels).
- לאחר חישוב המכפלה הסקלרית של כל מטריצה סוכמים את התוצאה לקבלת מספר בודד.
- קונבולוציה עם kernel (פילטר) אחד של תמונה צבעונית הכוללת שלוש
- מטריצות, תיתן לנו מטריצה אחת.

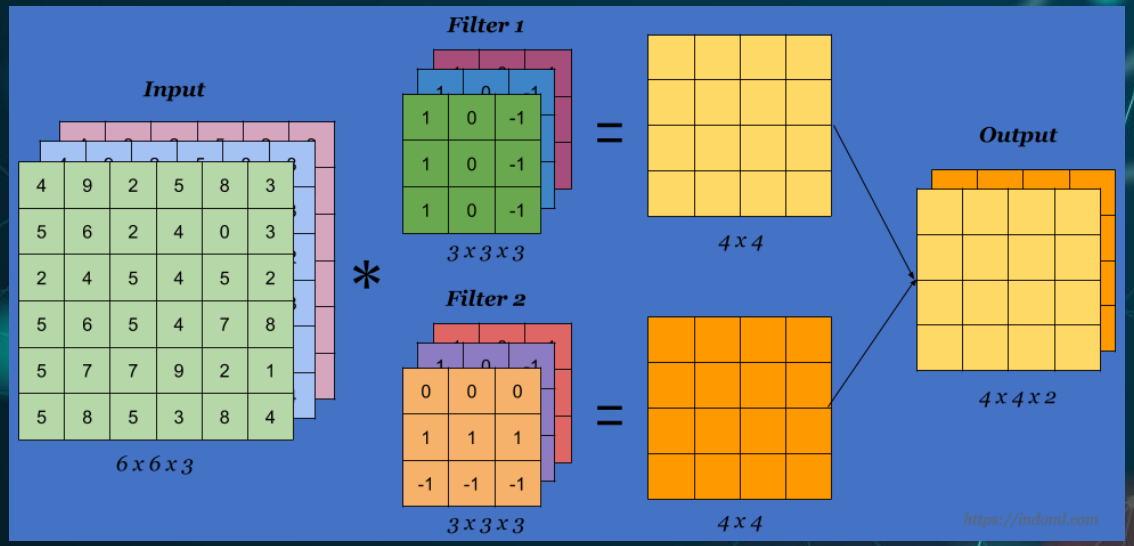


קונבולוציה של תמונה צבעונית



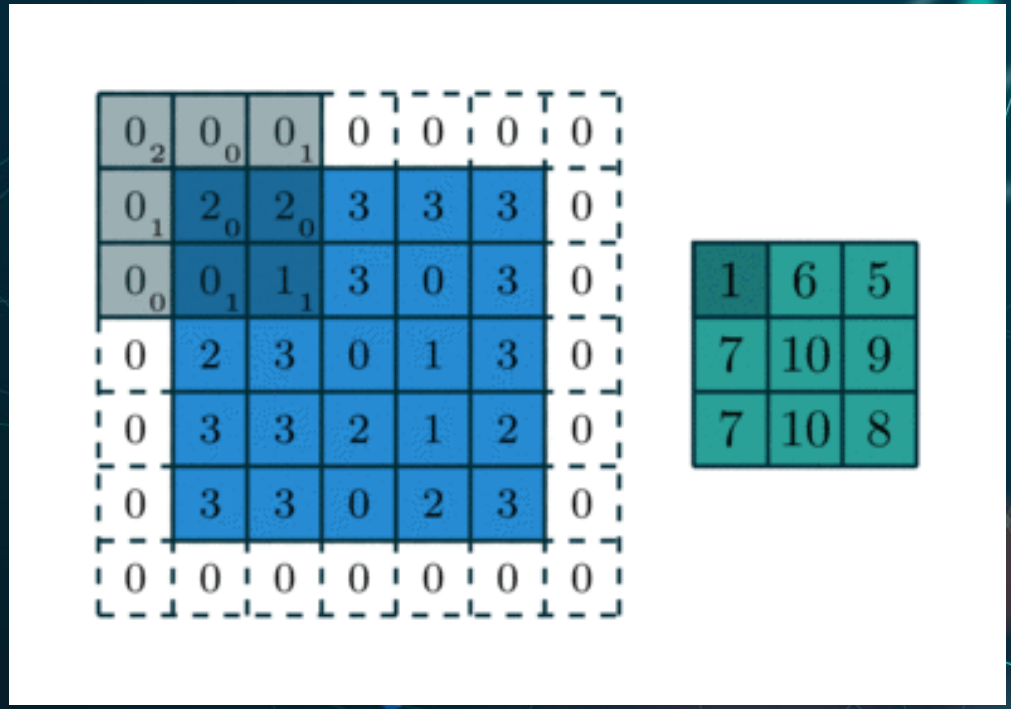
קביעת מספר הערוצים

- ניתן לבצע קונבולוציה על ידי מספר kernels וכך לקבל תוצאה הכוללת מספר מטריצות.
- בהתאם למספר ה kernels ניתן לקבוע את מספר המטריצות שיתקבלו, ואין מניעה אף להגדיל את מספר המטריצות לעומת הקלט המקורי.

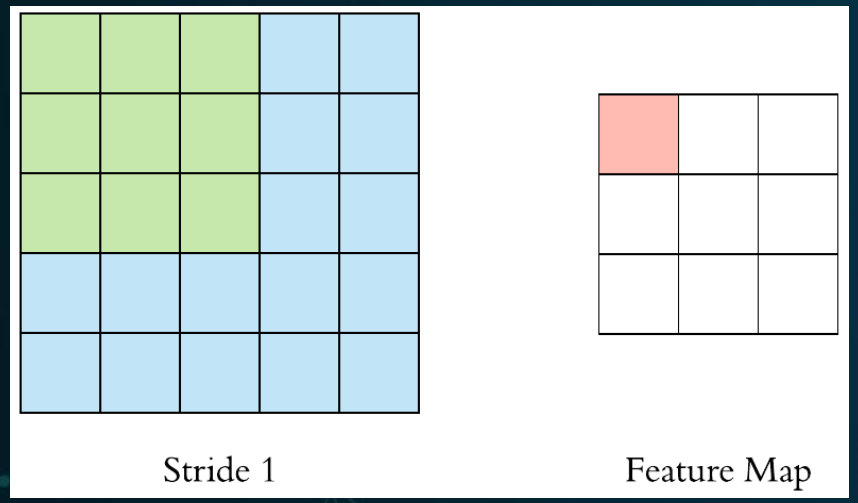


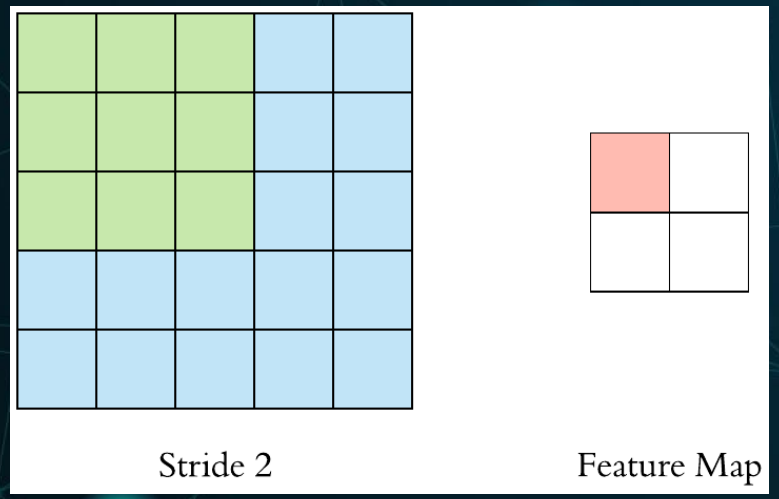
padding

- במקרים מבוצעת פעולה של padding – הגדלת המטריצה של התמונה באמצעות אפסים (0) בכדי שלא לאבד את הנתונים הנמצאים בשולי התמונה.
- פעולת ה padding מגדילה את מטריצת הפלט של הקונבולוציה.



Stride

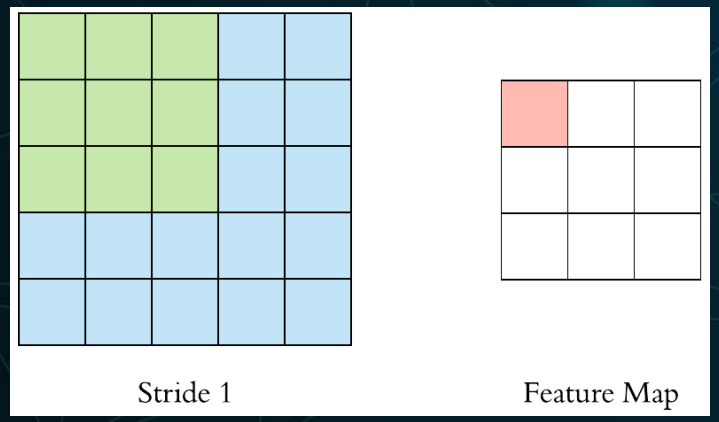


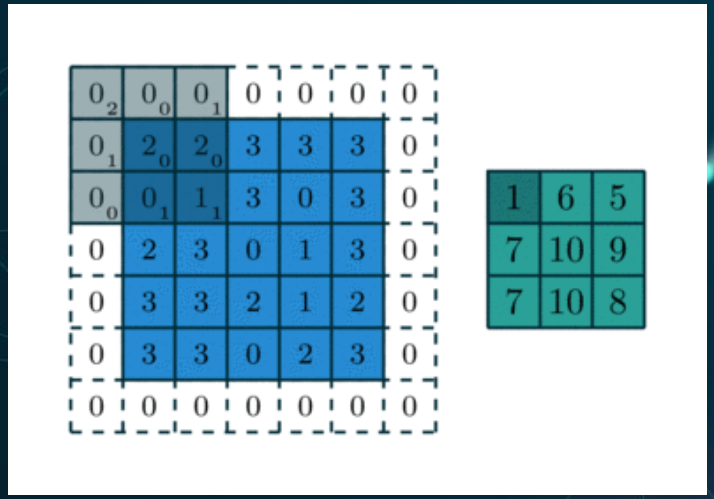


- Stride – מייצג את מספר הצעדים בו הפילטר נע על גבי התמונה.
- בדרך כלל הצעדים האופקיים והאנכיים זהים.
- ה stride קובע את גודל הפלט. ככל שהצעד גדול יותר מימדי הפלט קטנים.
-

נוסחת הקונבולוציה

- ניתן לחשב את מימדי מטריצת הפלט לאחר ביצוע הקונבולוציה, בהתאם לנוסחה הבאה.
- הנוסחה הינה הכרחית לחישוב הגדלים של הפלט בין שכבות רשת הנוירונים (כפי שנראה בהמשך).





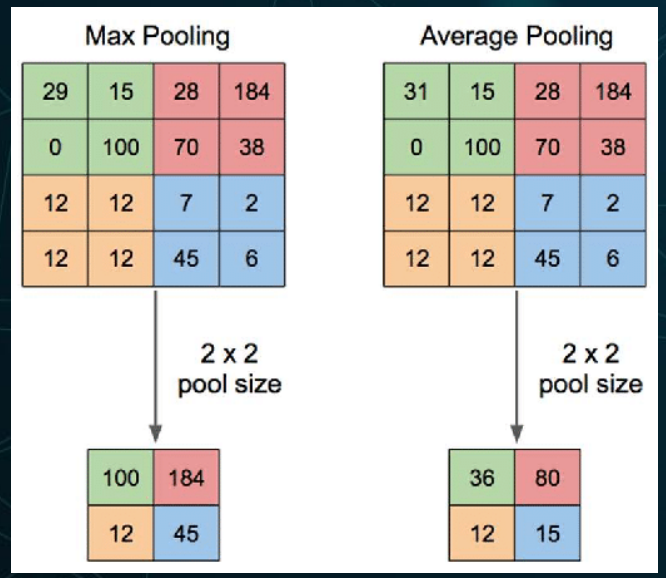
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

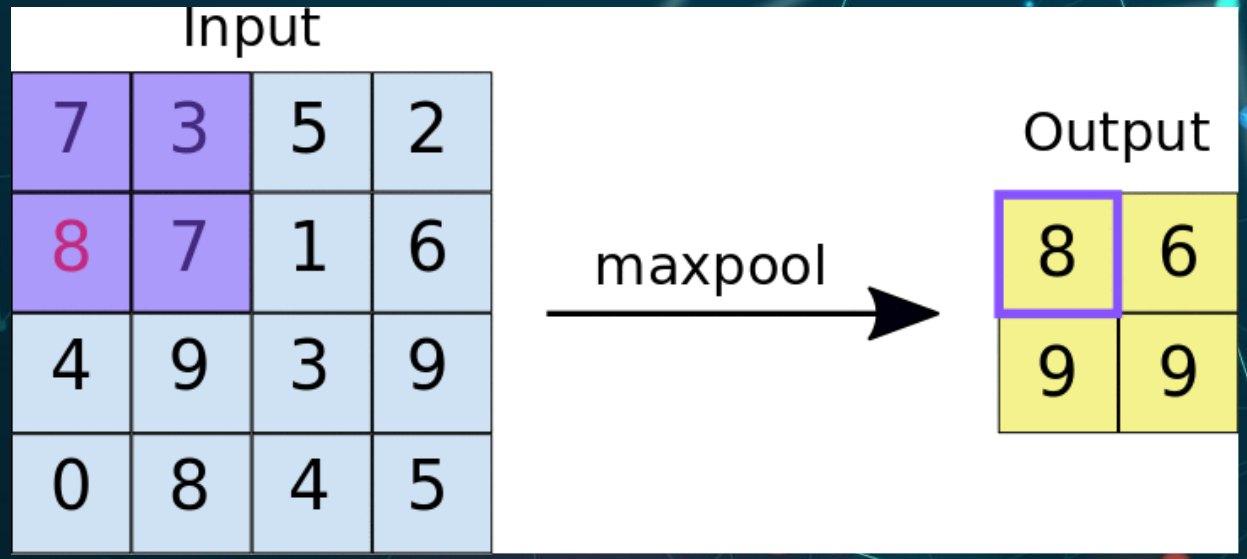
n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

pooling

- הפעולה של pooling נועדה על מנת לצמצם את מימדי התמונות תוך איבוד מינימלי של מידע.
- הפעולה מחזירה מספר אחד למטריצה, בין אם המספר **המירבי** או **הממוצע**.
- הנוסחה לחישוב מימדי הפלט:

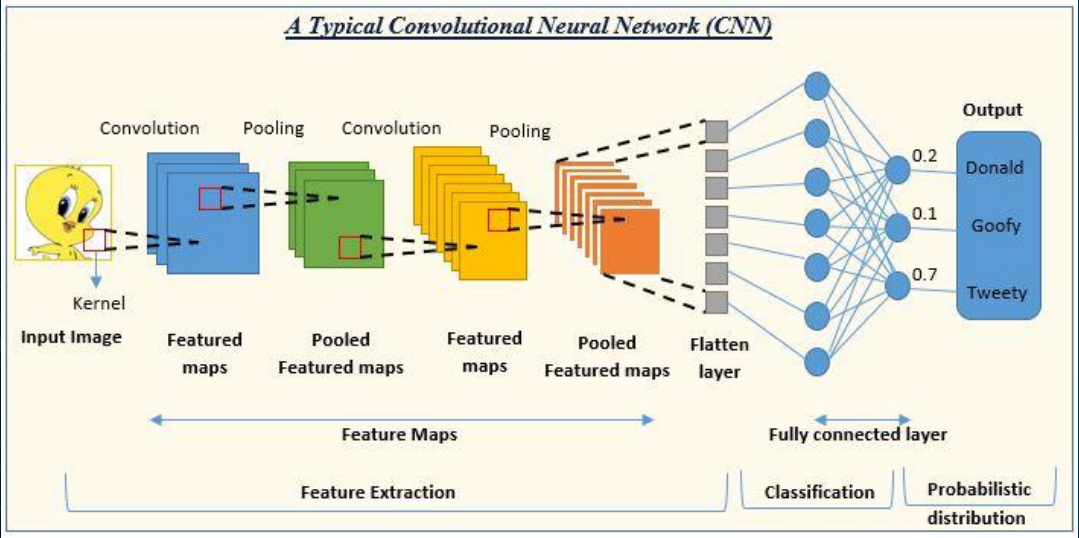
$$\text{Floor}\left(\frac{(W-F)}{S}+1\right)$$

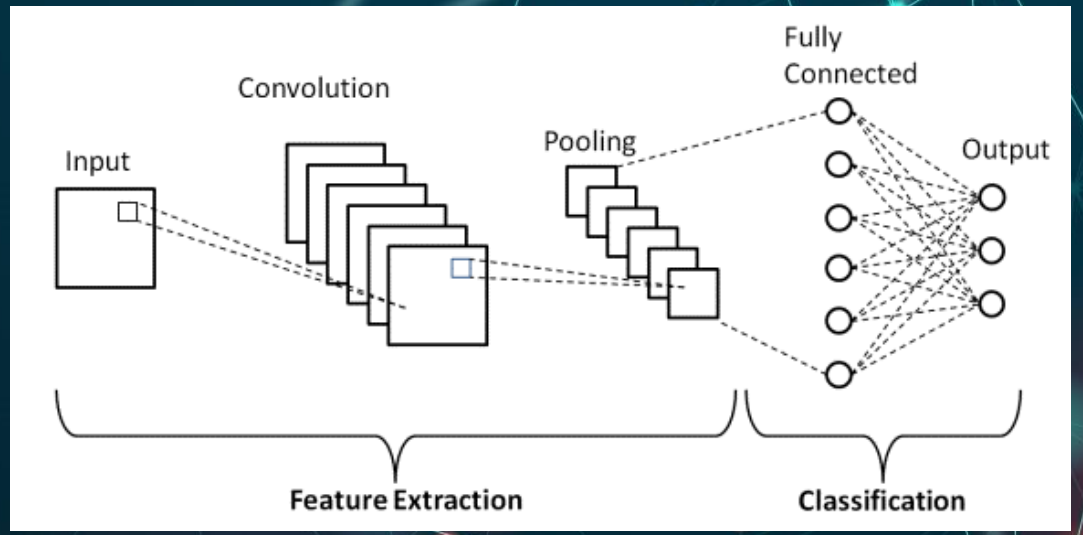




מבנה CNN

- רשת קונבולוציה בנויה בדרך כלל במבנה הבא:
  - שכבת הקלט (קולטת תמונה ללא שינוי של המימדים).
  - שכבת קונבולוציה אחת או יותר: הכוללות כל אחת פעולת קונבולוציה, אקטיבציה, ולאחריה פעולת pooling.
  - פעולה המשטחת את הפלט של הקונבולוציה לטנסר בעל מיימד אחד.
  - רשת נוירונים רגילה.





Pytorch: nn.Conv2d, nn.MaxPool2d

- ברשת CNN הערכים של הפילטרים השונים (Kernel, Pooling) הם הפרמטרים שמתעדכנים תוך כדי צעדי הלימוד.
- המחלקות nn.Conv2d, nn.MaxPool2d יוצרים עבורנו את הפעולות הנדרשות עבור ביצוע הקונבולוציה, בהתאם לפרמטרים הרלוונטיים.
- הפרמטרים מאותחלים באופן ראנדומלי ומעדכנים בכל ביצוע אופטימיזציה של הרשת.
-
- ברירת המחדל של stride = 1 ושל padding = 0.

```
in_Channels, out_channels, kernel, stride, padding = 3, 6, 5, 1, 0
c = nn.Conv2d(in_Channels, out_channels, kernel, stride, padding)

pool_kernel, pool_stride = 2, 2
mp = nn.MaxPool2d(pool_kernel , pool_stride)
```

דוגמה לרשת קונבולוציה - CNN

- בניית רשת קונבולוציה ב pytorch נעשית בדומה לרשת נוירונים רגילה.
- חישוב מימדי הפלט בשקופית הבאה.

```python
class CNN_Model(nn.Module):
    def __init__(self):
        super(CNN_Model, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # -> n, 3, 32, 32
        x = self.pool(F.relu(self.conv1(x)))  # -> n, 6, 14, 14
        x = self.pool(F.relu(self.conv2(x)))  # -> n, 16, 5, 5
        x = x.view(-1, 16 * 5 * 5)            # -> n, 400
        x = F.relu(self.fc1(x))               # -> n, 120
        x = F.relu(self.fc2(x))               # -> n, 84
        x = self.fc3(x)                       # -> n, 10
        return x


Model = CNN_Model().to(device)
```

Convolution calc

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Pooling calc

$$\text{Floor}\left(\frac{(W-F)}{S} + 1 \right)$$

חישוב פלט של שכבת קונבולוציה

- במקרה שלנו הקלט הן התמונות צבעוניות בגודל 3*32*32.
- חישוב מימדי הפלט לאחר השכבה הראשונה:
  - מספר הערוצים channels נקבע במפורש בהגדרת השכבה = 6.
  - רוחב התמונה לפני pooling $n = \left\lfloor \frac{32+2*0-5}{1} \right\rfloor + 1 = 27 + 1 = 28$
  - רוחב התמונה לאחר pooling: $w = \left\lfloor \frac{28-2}{2} + 1 \right\rfloor = 14$

```python
class CNN_Model(nn.Module):
    def __init__(self):
        super(CNN_Model, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # -> n, 3, 32, 32
        x = self.pool(F.relu(self.conv1(x)))  # -> n, 6, 14, 14
        x = self.pool(F.relu(self.conv2(x)))  # -> n, 16, 5, 5
        x = x.view(-1, 16 * 5 * 5)            # -> n, 400
        x = F.relu(self.fc1(x))               # -> n, 120
        x = F.relu(self.fc2(x))               # -> n, 84
        x = self.fc3(x)                       # -> n, 10
        return x


Model = CNN_Model().to(device)
```

Convolution calc

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

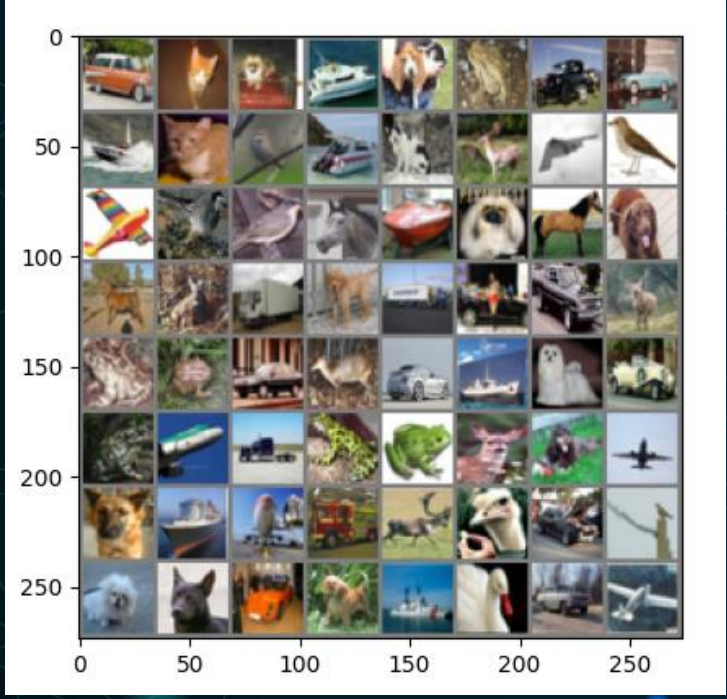
Pooling calc

$$\text{Floor}\left(\frac{(W-F)}{S} + 1 \right)$$

בניית CYPHAR10 – CNN

- נבנה רשת נוירונים אשר תזהה תמונות מבסיס הנתונים CYPHAR10, הכולל תמונות צבעוניות בגודל 32*32 המחולקות לעשר קבוצות:

```
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```



- נטען את בסיס הנתונים באמצעות הספריה torchvision.
- נבנה רשת קונבולוציה הכוללת שתי שכבות קונבולוציה ושלוש שכבות של רשת נוריונים רגילה (FNN).
- הפלט יהיה 10 קטגוריות, ונעשה שימוש ב softmax כדי למצוא את הקטגוריה הנכונה.
- * הקוד מתוך סדרת ההרצאות של Patrick Loeber

טעינת התמונות

```
[19] # dataset has PILImage images of range [0, 1].
     # We transform them to Tensors of normalized range [-1, 1]
     transform = transforms.Compose([
         transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
         ])
```

```
[20] batch_size = 4

     train_dataset = torchvision.datasets.CIFAR10(root='./data',
                                                  train=True,
                                                  transform=transform,
                                                  download=True)

     test_dataset = torchvision.datasets.CIFAR10(root='./data',
                                                 train=False,
                                                 transform=transform)

     # Data loader
     train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                                batch_size=batch_size,
                                                shuffle=True)

     test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                               batch_size=batch_size,
                                               shuffle=False)
     classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

דוגמה מן התמונות

```
[22] # get some random training images
     dataiter = iter(train_loader)
     for i in range(5):
       images, labels = next(dataiter)

       # show images
       imshow(torchvision.utils.make_grid(images))
```

המודל של CNN

- המודל כולל שתי שכבות קונבולוציה ושלוש שכבות של רשת נוירונים רגילה FNN.
- הפלט בגודל 10 בהתאם למספר סוגי התמונות.

```
[26] class CNN_Model(nn.Module):
        def __init__(self):
            super(CNN_Model, self).__init__()
            self.conv1 = nn.Conv2d(3, 6, 5)
            self.pool = nn.MaxPool2d(2, 2)
            self.conv2 = nn.Conv2d(6, 16, 5)
            self.fc1 = nn.Linear(16 * 5 * 5, 120)
            self.fc2 = nn.Linear(120, 84)
            self.fc3 = nn.Linear(84, 10)

        def forward(self, x):
            # -> n, 3, 32, 32
            x = self.pool(F.relu(self.conv1(x)))  # -> n, 6, 14, 14
            x = self.pool(F.relu(self.conv2(x)))  # -> n, 16, 5, 5
            x = x.view(-1, 16 * 5 * 5)            # -> n, 400
            x = F.relu(self.fc1(x))               # -> n, 120
            x = F.relu(self.fc2(x))               # -> n, 84
            x = self.fc3(x)                       # -> n, 10
            return x


    Model = CNN_Model().to(device)
```

Loss + optomization

- כיוון שאנחנו בונים רשת שמטרתה חלוקה לקטגוריות אנו עושים שימוש בשכבה האחרונה בפונקציית האקטיבציה softmax ופונקציית הטעות CrossEntropyLOss.
- ב pytorch הפונקציה nn.CrossEntropyLoss() מבצעת גם את softmax וכלן במודל עצמו לא מוסיפה את פונקציית האקטיבציה.

```
[27] # loss function
     Loss = nn.CrossEntropyLoss () # applies nn.LogSoftmax + nn.NLLLoss, No softmax in last layer

     # init optimizer
     optim = torch.optim.Adam(Model.parameters(), lr=learning_rate)
```

לולאת האימון

- לולאת האימון רגילה בדומה לכל האלגורתמים הקודמים.
- אנחנו משתמשים ב GPU ולכן מעבירים את הנתונים לזכרון המתאים.

```
n_total_steps = len(train_loader)


for epoch in range(epochs):
  for i, (images, lables) in enumerate(train_loader):

    images = images.to(device)
    lables = lables.to(device)

    # forward
    Y_predict = Model(images)

    # backward
    loss = Loss(Y_predict, lables)
    loss.backward()

    # update wights
    optim.step()

    if i % 2000 == 0:
      print(f"epoch= {epoch} i= {i+epoch * n_total_steps} loss={loss.item():.4f} ")

    # zero wights
    optim.zero_grad()
```

בדיקה של המודל

- הרצה של המודל לאחר 5 epochs מביאה לדיוק של 57% (מודל רנדומלי היה מביא לדיוק של 10% בלבד).
- שימו לב ש learning_rate = 0.01. בקצב גבוה יותר המודל לא למד.
- כמובן שניתן לשפר את המודל עם מספר רב יותר של epochs ושינוי פרמטרים או מבנה הרשת.

```
[76] with torch.no_grad():
        n_correct = 0
        n_samples = 0
        for images, lables in test_loader:
            images = images.to(device)
            lables = lables.to(device)
            __,y_predict = torch.max(Model(images),1)
            n_samples += lables.size(0)
            n_correct += (y_predict == lables).sum().item()

        acc = 100 * n_correct / n_samples
        print(f'Accuracy of the network on the {n_samples} test images: {acc} %')
```

```
Accuracy of the network on the 10000 test images: 57.04 %
```