המכללה האקדמית להנדסה ע"ש סמי שמעון

פרויקט זיהוי גזעי כלבים

השוואה בין 2 מודלים לזיהוי :
CNN and ResNet

השוואת דיוק וזמני חישוב של שני מודלים לזיהוי גזעי כלבים **CNN** לעומת **ResNet**.
זיהוי אוטומטי של גזעי כלבים יכול לסייע בווטרינריה, אפליקציות לזיהוי בעלי חיים, מערכות אבטחה, ועוד.
לכן, בחירת המודל המתאים חשובה לשיפור הדיוק והיעילות של המערכת.

המכללה האקדמאית להנגסה ע"ש סמי שמעון

באר-שבע

מחלקת מדעי המחשב

תאריך עברי	תאריך לועזי
ו' בשבט התשפ"ה	04/02/2025

1. תיאור המודלים:

מודל ה CNN :

Output Dimensions	Input Dimensions	Layers	Stage
64×64×64	128×128×3	Conv2D (64 filters, 3×3) + BatchNorm + ReLU +×2 MaxPool(2×2)	**Block 1**
32×32×128	64×64×64	Conv2D (128 filters, 3×3) + BatchNorm + ReLU +×2 MaxPool(2×2)	**Block 2**
16×16×256	32×32×128	Conv2D (256 filters, 3×3) + BatchNorm + ReLU +×2 MaxPool(2×2)	**Block 3**
8×8×512	16×16×256	Conv2D (512 filters, 3×3) + BatchNorm + ReLU +×2 MaxPool(2×2)	**Block 4**
1×1×512	8×8×512	AdaptiveAvgPool2D (1×1)	**GAP**
70	512	Dropout(0.5) + Linear(512 → 256) + ReLU + Dropout(0.5) + Linear(256 → 70)	**Fully Connected**

```python
class DogCNN(nn.Module):
    def __init__(self, num_classes=70):
        super(DogCNN, self).__init__()

        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1),nn.BatchNorm2d(64),nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1),nn.BatchNorm2d(64),nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)  # הקטנת רזולוציה בחצי
        )

        self.block2 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, padding=1),nn.BatchNorm2d(128),nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),nn.BatchNorm2d(128),nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.block3 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=3, padding=1),nn.BatchNorm2d(256),nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),nn.BatchNorm2d(256),nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.block4 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=3, padding=1),nn.BatchNorm2d(512),nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),nn.BatchNorm2d(512),nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )


        self.gap = nn.AdaptiveAvgPool2d((1,1))

        # Fully Connected
        self.classifier = nn.Sequential(
            nn.Dropout(p=0.5),
            nn.Linear(512, 256),
            nn.ReLU(inplace=True),

            nn.Dropout(p=0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)

        x = self.gap(x)
        x = x.view(x.size(0), -1)

        x = self.classifier(x)
        return x

def get_dog_cnn(num_classes=70):
    return DogCNN(num_classes=num_classes)
```

***מודל ה ResNet18** :

Output Dimensions	Input Dimensions	Layers	Stage
64×64×64	128×128×3	Conv2D (64 filters, 7×7, Stride=2) + BatchNorm + MaxPool(3×3)	**Input**
64×64×64	64×64×64	Residual Block (64 filters, 3×3)×2	**Block 1**
32×32×128	64×64×64	Residual Block (128 filters, 3×3, Stride=2)×2	**Block 2**
16×16×256	32×32×128	Residual Block (256 filters, 3×3, Stride=2)×2	**Block 3**
8×8×512	16×16×256	Residual Block (512 filters, 3×3, Stride=2)×2	**Block 4**
1×1×512	8×8×512	AdaptiveAvgPool2D (1×1)	**GAP**
70	512	Dropout(0.5) + Linear(512 → 70)	**Fully Connected**

```python
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.shortcut = nn.Sequential()

        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=stride),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=70):
        super(ResNet, self).__init__()
        self.in_channels = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.pool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        self.layer1 = self._make_layer(block, 64,  num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)

        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512, num_classes)
        self.dropout = nn.Dropout(0.5)

    def _make_layer(self, block, out_channels, num_blocks, stride):
        layers = []
        layers.append(block(self.in_channels, out_channels, stride))
        self.in_channels = out_channels
        for _ in range(1, num_blocks):
            layers.append(block(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.pool(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.avg_pool(out)
        out = torch.flatten(out, 1)
        out = self.dropout(out)  # הוספת Dropout לפני ה-Fully Connected layer
        out = self.fc(out)
        return out

def ResNet18(num_classes=70):
    return ResNet(ResidualBlock, [2, 2, 2, 2], num_classes=num_classes)
```

2. ניסויים ובדיקות:

2.1 מערך הנתונים

שני המודלים אומנו על סט הנתונים "**70 Dog Breeds Image Dataset**" הכולל:

70 גזעי כלבים שונים.

חלוקה ל Train-ו Test-בהתאם למבנה הקובץ (7946 תמונות אימון ו700 תמונות בסט הבדיקה)

שימוש ב Data Augmentation (שינוי תאורה, סיבוב וכו'). לשנות למה שרשמתי בדף!!!!!!!!!

2.2 תהליך האימון:

אימון מודל ה CNN היה למשך 110 אפוקים.

כאשר נוספה פקודה לשמירת הלמידה הטובה ביותר:

```
Epoch [108/110] | Train Loss: 0.2027 | Train Accuracy: 93.24% | Test Accuracy: 88.29%
Epoch [109/110] | Train Loss: 0.2075 | Train Accuracy: 93.28% | Test Accuracy: 88.57%
Epoch [110/110] | Train Loss: 0.1906 | Train Accuracy: 93.75% | Test Accuracy: 89.00%
best model saved:/content/drive/MyDrive/OurCNN-Model.pth
```

אימון מודל ה ResNet היה למשך 50 אפוקים.

כאשר נעשת בדיקה להפסקת הלמידה אם המודל מפסיק ללמוד , כלומר מתחיל overfitting

במקרה שלנו אחרי 34 אפוקים:

```
Epoch [33/50] | Train Loss: 0.4469 | Train Accuracy: 85.65% | Test Loss: 1.0143 | Test Accuracy: 72.29%
Epoch [34/50] | Train Loss: 0.4264 | Train Accuracy: 86.03% | Test Loss: 1.0217 | Test Accuracy: 69.71%
Early stopping triggered after 34 epochs.
Final Model - Train Accuracy: 81.83%, Test Accuracy: 73.14%
```

3. **תוצאות המודל :**

```
The best model was saved with Test Accuracy: 88.29%
Epoch [100/110] | Train Loss: 0.2826 | Train Accuracy: 90.86% | Test Accuracy: 88.29%
Epoch [101/110] | Train Loss: 0.2694 | Train Accuracy: 91.29% | Test Accuracy: 88.00%
Epoch [102/110] | Train Loss: 0.2655 | Train Accuracy: 91.29% | Test Accuracy: 88.14%
Epoch [103/110] | Train Loss: 0.2716 | Train Accuracy: 91.42% | Test Accuracy: 86.86%
Epoch [104/110] | Train Loss: 0.2859 | Train Accuracy: 90.52% | Test Accuracy: 86.57%
Epoch [105/110] | Train Loss: 0.2610 | Train Accuracy: 91.58% | Test Accuracy: 87.00%
Epoch [106/110] | Train Loss: 0.2615 | Train Accuracy: 91.28% | Test Accuracy: 88.14%
The best model was saved with Test Accuracy: 89.86%
Epoch [107/110] | Train Loss: 0.2318 | Train Accuracy: 92.61% | Test Accuracy: 89.86%
Epoch [108/110] | Train Loss: 0.2027 | Train Accuracy: 93.24% | Test Accuracy: 88.29%
Epoch [109/110] | Train Loss: 0.2075 | Train Accuracy: 93.28% | Test Accuracy: 88.57%
Epoch [110/110] | Train Loss: 0.1906 | Train Accuracy: 93.75% | Test Accuracy: 89.00%
best model saved:/content/drive/MyDrive/OurCNN-Model.pth
```

ניתן לראות שבמודל ה CNN הדיוק הטוב ביותר הגיע ל- 89.86% באפוק 107 ואילו במודל ה ResNet הדיוק הטוב ביותר הגיע ל- 73.14% באפוק 34.

Model	Best Training Accuracy	Best Test Accuracy
CNN	92.61%	89.86%
ResNet	81.83%	73.14%

CNN הציג ביצועים טובים יותר במבחן של 89.86%, לעומת **ResNet** שהגיע רק ל-73.14%.
זה מראה כי למרות ש ResNet מיועד לרשתות עמוקות ויעילות יותר, במצב זה הוא לא הצליח להתגבר על ביצועי CNN פשוט.

זמן האימון של מודל ה- CNN ערך הרבה יותר זמן מזמן האימון של מודל ה ResNet:

Model	Training Time
CNN	~1.5 hours
ResNet	~30 minutes

4. **קשיים:**

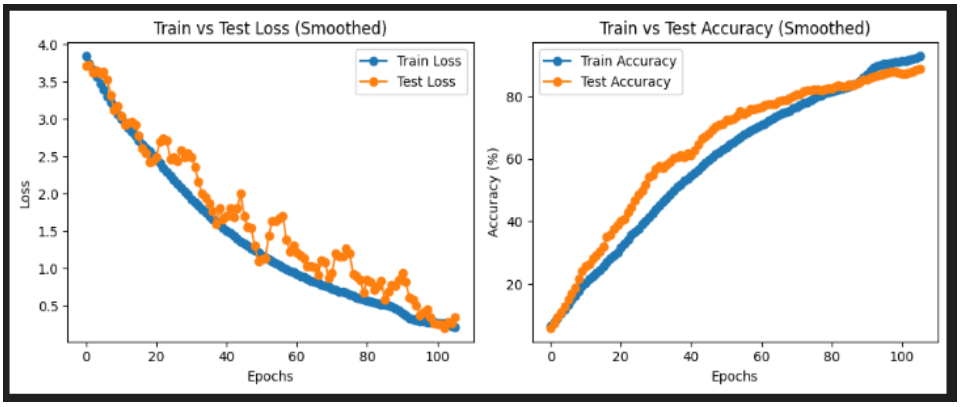
במהלך הפרויקט, אחד האתגרים המשמעותיים היה **איזון בין דיוק למניעת Overfitting**. היה עלינו לשפר את ביצועי המודל כך שיגיע לרמת דיוק גבוהה ככל האפשר, אך מבלי להגיע ל Overfitting-מוקדם מדי או לייצר רשת גדולה מדי שתדרוש משאבי חישוב גבוהים מאוד.

בנוסף, **זמן הריצה של מודל ה CNN-**היווה אתגר משמעותי. בשל הצורך באימון ממושך, פעמים רבות ההרצות ארכו זמן רב, ובמקרים מסוימים הסתיימה הקצאת ה GPU-מה שאילץ אותנו להמתין **יום-יומיים** עד שהתאפשר לנו להפעיל מחדש את ההרצה ולבדוק את ביצועי המודל.

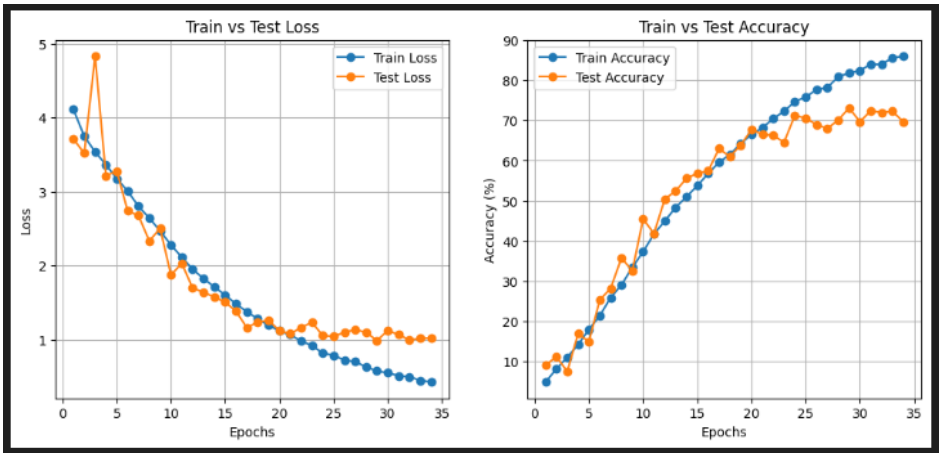
כמו כן, **אימון הדאטה על מגוון רחב של גזעי כלבים** דרש התמודדות עם שונות גבוהה בין התמונות. הבדלים בגודל, בצבע, בזוויות הצילום ובתנאי התאורה הפכו את תהליך הלמידה למורכב יותר, ונדרש שימוש בטכניקות כמו Data Augmentation כדי לשפר את הכללת המודל ולמנוע התאמת יתר למידע מסוים בלבד.

הצגת גרפים של דיוק ואובדן:

CNN:



ResNet:



5. <u>סיכום:</u>

המודל מבוסס CNN הצליח להגיע לדיוק גבוה יותר על סט הבדיקה, עם הפרש משמעותי לעומת ה ResNet.

למרות היתרונות התאורטיים של ResNet ברשתות עמוקות, במקרה זה הוא לא הצליח להתגבר על ביצועי ה CNN הפשוט יותר.

ניתן לראות כי CNN מפגין הכללה טובה יותר על סט הבדיקה, בעוד ש ResNet התחיל להראות סימני Overfitting בשלב מוקדם יחסית. זמן האימון של ResNet היה קצר יותר, אך הדבר הגיע על חשבון ירידה משמעותית בביצועים הסופיים.