Applied ML- Assignment 3:

Part 1:

בחלק זה עלינו היה לממש שכבה נוספת לרשת נוירונים הממומשת בעזרת numpy לקוד קיים הנמצא במחברת הבאה [machine-learning-book].
חילקנו את הדאטה סט ל Train(70%)-Test(30%).
ע"מ לעשות זאת שינינו מספר קטעי קוד, ראשית היה עלינו לשנות את המחלקה של הרשת. המחלקה בנויה ממספר פונקציות :

Init:

הפונקציה אשר נקראת בבניית המודל, עלינו להוסיף את המשקלים של השכבה החדשה(weight & bias).

```
self.weight_hh = rng.normal(loc=0.0, scale=0.1, size=(num_hidden, num_hidden))
self.bias_hh = np.zeros(num_hidden)
```

Forward:

הפונקציה משמשת עבור חישוב ערך פרדיקציה על הקלט, ניתן לראות את ההוספת השכבה הנוספת כך שהחישוב יעבור דרכה.

```
def forward(self, x):
    # First Layer
    z_h = np.dot(x, self.weight_h.T) + self.bias_h
    act_h = sigmoid(z_h)
    # Second Layer
    z_hh = np.dot(act_h, self.weight_hh.T) + self.bias_hh
    act_hh = sigmoid(z_hh)
    z_out = np.dot(act_hh, self.weight_out.T) + self.bias_out
    act_out = sigmoid(z_out)
    return act_h, act_hh, act_out
```

Backward:

לאחר החיזוי, נחשב את הגרדיאנט של כל אחד ממשקלי היחס להפסד.
לאחר החישוב נחזיר את הערך שהתקבל זאת כדי שנוכל לעדכן את המשקלי הרשת ביחס לאותו גראדיאנט.

(מאחר ותוכן חלק זה מכיל הרבה קוד נציג דוגמה רק עבור השכבה אמצעית, שאר הקוד נמצא במחברת).

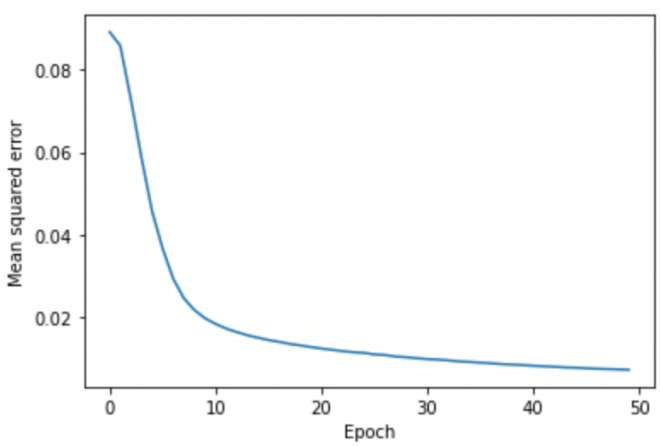
```
#########################
### Middle layer weights
#########################
# dLoss/dWeightsHH = dLoss/dZOut * dZOut/dActHH * dActHH/dZHH * dZHH/dWHH
dZOut_dActHH = self.weight_out
dLoss_dActHH = np.dot(dLoss_dZOut,dZOut_dActHH)
dActHH_dZHH =  act_hh * (1. - act_hh)
dZHH_dWHH = act_h
dLoss_dZHH = dLoss_dActHH * dActHH_dZHH
# Lets Multiply
dLoss_dWeightsHH = np.dot(dLoss_dZHH.T,dZHH_dWHH)
dLoss_dBHH = np.sum(dLoss_dZHH, axis=0)
```

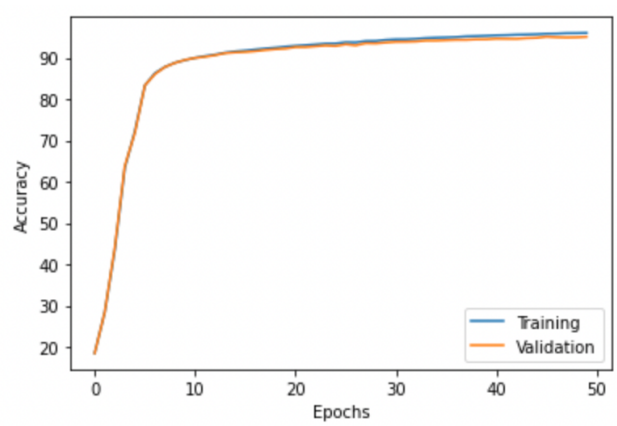
Update:

הפונקציה זו מקבלת את הגרדיאנטים ביחס לכל אחד מהמשקולות מפונקציית ה-Backward. בהתאם לגרדיאנט ולקצב הלמידה מתבצע עדכון למשקלים.

```
def update(self, grad, lr):
    self.weight_h -= lr * grad[0]
    self.bias_h -= lr * grad[1]

    self.weight_hh -= lr * grad[2]
    self.bias_hh -= lr * grad[3]

    self.weight_out -= lr * grad[4]
    self.bias_out -= lr * grad[5]
```

לאחר זאת ביצענו אימון עבור המודל וקיבלנו את התוצאות הבאות:





לאחר מכן ביצענו אימון של רשת נוירונים בעזרת torch.

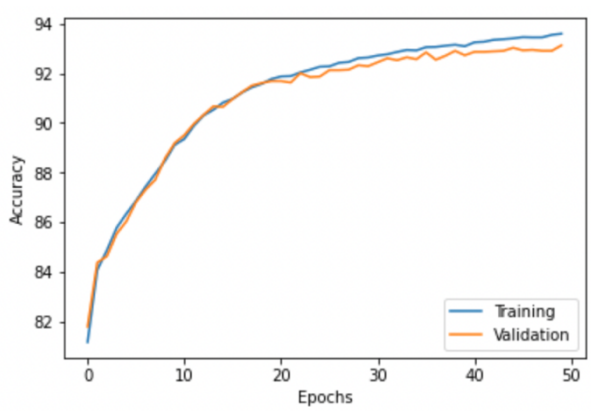
```python
class TorchModel(torch.nn.Module):
    def __init__(self,num_features,num_hidden,num_classes):
        super(TorchModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(num_features,num_hidden),
            nn.Sigmoid(),
            nn.Linear(num_hidden,num_classes),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.net(x)
```

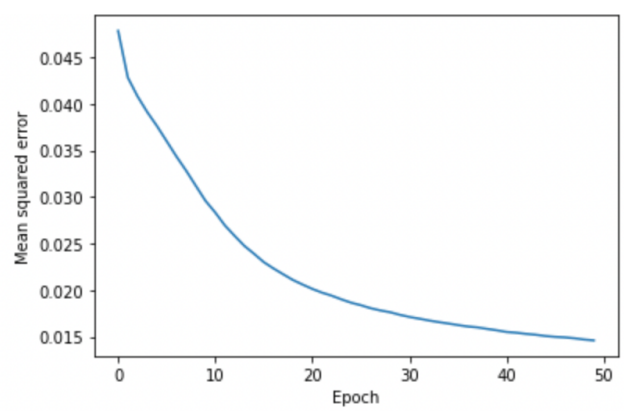
בנוסף הגדרנו אופטימייזר ופונקציית הפסד בדומה למודל הממומש בnumpy.

```python
optimizer = torch.optim.SGD(torch_model.parameters(), lr=0.01, momentum=0.9)
criterion = torch.nn.MSELoss()
```

בחרנו ב- MSE על מנת שהשוואה בין המודלים תהיה כמה שיותר מדויקת דבר אשר נכון גם ל-Lr.
בנוסף השתמשנו בקוד השינוי היה המרת האינפוט של המודל החדש ממערכים לTensor.

לבסוף קיבלנו את התוצאות הבאות:





לאחר שראינו כי הדיוק מתחיל מערך גבוהה הרצנו הרצה נוספת וקיבלנו תוצאות דומות(מאחר וה-seed זהה) ,
לבסוף יצא כי הדיוק על הטסט עבור המודל ב- Numpy הוא: 94.9% והדיוק עבור המודל ב-torch הוא :
93.35%.

Part 2:

בדומה לחלק הראשון , בחרנו להשתמש ב- Pytorch ע״מ לבנות ואמן מודל.
כמו שהוסבר במטלה השתמשנו בtransfer learning וביצענו hyper-tuning ע״ג הרשת החדשה.
המודלים אותם בחרנו לייבא הם: VGG16, RESNET50. המודלים יובאו עם משקלים מאומנים.
כחלק מהתהליך דאגנו למנוע שינוי במשקלי השכבות התחתונות, כך שהעדכון יהיה רק בשכבה האחרונה.
בנוסף, ביצענו שינוי בשכבה האחרונה כך שתתאים לבעיה שלנו (מספר שונה של classes), כמו כן הוספנו שכבה של softmax.

```
def init_by_name(name):
    if name is 'vgg':
        model = models.vgg16(pretrained=True)
    elif name is 'resnet':
        model = models.resnet50(pretrained=True)
    else:
        raise ValueError("need right name")
    # freeze unwanted weight change
    for param in model.parameters():
        param.requires_grad = False
    # update last layer for our problem
    if name is 'vgg':
        modelOutputFeats = model.classifier[6].in_features
        model.classifier[6] = torch.nn.Linear(modelOutputFeats, N_CLASS)
        model.softmax = torch.nn.Softmax(dim=1)
    else:
        modelOutputFeats = model.fc.in_features
        model.fc = torch.nn.Linear(modelOutputFeats, N_CLASS)
        model.softmax = torch.nn.Softmax(dim=1)
    model = model.to(device)
    return model
```

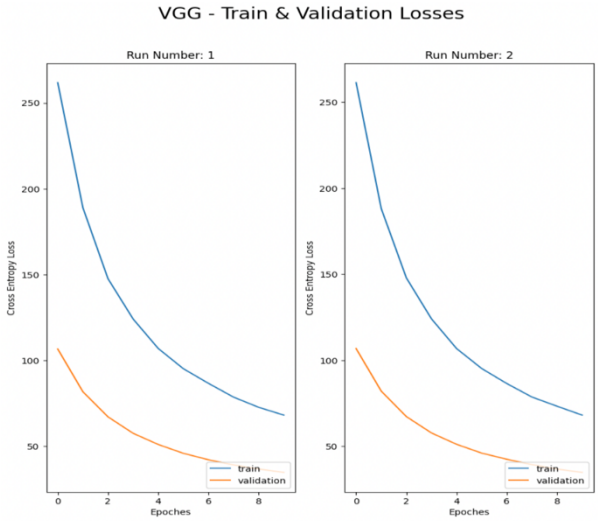
לאחר מכן יצרנו את הערכים הדרושים ב- Pytorch לשם ההרצה. הערכים כוללים dataset, dataloader.
כאשר פונקציית ההפסד שלנו היית CrossEntropyLoss, והאופטימיזר היה SGD .

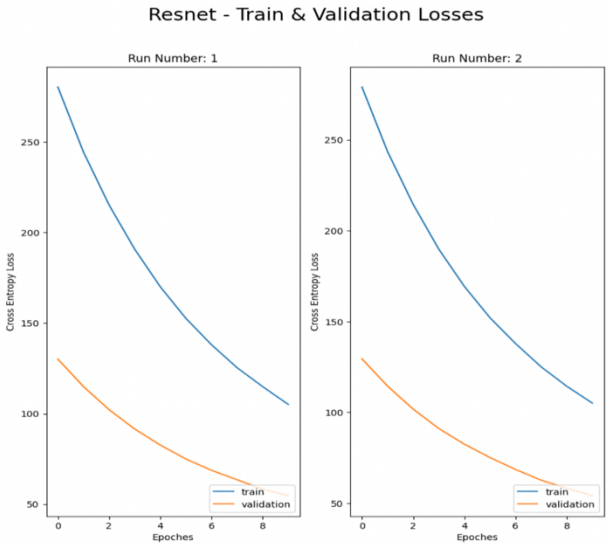
על מנת למקסם את דיוק המודל השתמשנו טרנספורמציה , שינינו את גודל התמונה ל 256x256, ומתוכה חתכנו תמונה בגדול 224x224 של מרכז התמונה, כמו כן עכשיו נורמליזציה ע״פ הערכים המופעים באיור הבא:

```
transforms.Compose([transforms.Resize(256),
                    transforms.CenterCrop(224),
                    transforms.ToTensor(),
                    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                         std=[0.229, 0.224, 0.225])])
```

בצענו 2 ריצות עבור כל מודל (איפוס של המודל למשקלים המקוריים) ואימון מחדש.
וקיבלנו את התוצאות הבאות:

עבור **אימון וולידציה**:





עבור **סט הבדיקה**: