**Deep learning Final project report**  
Ofek Manor, ID: 206949489  
Itamar Podolsky, ID: 206026254

Before Starting the first experiment, we came to the conclusion from exercise 2   
that the following parameters benefited us the best results:  
filter per layer: 64  
layer per block: 8  
Those will be the following parameters that we will be using during our experiments.

**Experiment 1:**In this experiment we checked what would happen if we try out different hidden dimension.  
The results are the following:  
1. Hidden dim: [50,50] – Train accuracy: Around 90%, Test accuracy- 71.67  
2. Hidden dim: [100, 100] -Train accuracy: Around 90%, Test accuracy – 73.94  
3. Hidden dim: [150, 150] - Train accuracy: Around 90%, Test accuracy – 74.94  
4. Hidden dim: [200, 200] – Train accuracy: Around 90%, Test accuracy – 73.48

As we can see, the best results on the test came from the hidden dim: [150, 150]  
so going onwards we will be using this hidden dim.

**Experiment 2: Checking out different optimizers and learning rates.**In this experiment we were comparing between SGD with momentum and the ADAM optimizers with different learning rates.   
We found out that the best test result belonged to the optimizer **ADAM** with a learning rate of **0.001**, and chose to proceed using it.

**Why should we add momentum to the SGD?**  
We should add momentum to the SGD method because the momentum helps accelerate gradient’s vectors in the average right directions, so it leads to faster converge.   
What SGD without momentum does is taking small batches of the data, and calculates their gradients, so we might end up going for one direction instead of another, this adds noise.  
SGD with momentum sums up the gradients results and averages it. Then we move the average’s way. This adds denoise to the gradient and allows for better and faster converge.  
Another thing that SGD with momentum helps with is Ravines.   
Ravine is an area, where the surface curves much more steeply in one dimension than in another. Ravines are common near local minima in deep learning and SGD has troubles navigating them. SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum.

**What is the main two improvements Adam has comparing to the simple SGD?**Adams adds themomentum mentioned above, with all the benefits mentioned.  
Additionally, Adam also normalizes the directions of the optimizer, thus improving the chance to avoid the aforementioned ravines and avoiding zig-zags.

**Experiment 3: adding batch normalization**

**What is the purpose of batch normalization (why do we use it)?**   
The purpose of batch normalization are as follows:  
1.  Batch normalization enables us to normalize the activations from each previous layer which will improve the gradient descent, allowing it to converge better during training.  
2. Batch normalization allows us to use higher learning rates.  
3. Reduces the influence of the starting weight initialization on the weights.  
4. Batch normalization is used as regularization method, because it adds noise to model.

**Did the Batch normalization improve the network preformance?**  
Yes, the batch normalization did improve our network performance.  
The batch normalization helped the fitting of our model, in previous versions we had overfits. We had higher results in the training batches, while our test batches performed poorly in comparison. But not we batch normalization. With batch normalization we had a fitting result!  
in the training batch we had 81% accuracy and in our test batches we had a result of 78%!  
in comparison, our best previous result had 90% training accuracy while the test had 74.89% accuracy.  
Overall, we think the batch normalization improved our model, It added a needed regularization, improved the convergence rate, improved test accuracy and improved the fitting of the model.  
  
**Experiment 4: adding Regularization  
  
Why should we use regularization?**We should use regularization as a method to prevent overfit, it does so by adding penalty as the model complexity increase, so it would reduce some features weights and strengthen others.   
The main goal is to let each feature in a test data the same importance, and prevent a model from concentrating on a few features.

**How does the regularization affect the train accuracy and loss?**The regularization affects the train accuracy by decreasing it and the loss by increasing it.  
However, due to its generalizing idea, both validation and test accuracy should increase and the loss should reduce. Overall it shall improve the model and make it more stable.

**How does it affect the val and test accuracy and loss?**The regularization affects the validation and test accuracy by improving it and making the loss lower. Because the regularization makes the model to be more “generalized” it helps the model to be correct on new and unfamiliar examples, making the model more accurate and trustworthy.

**What was the best regulrization method?**In this experiment we checked L2 regularization and the dropout methods.   
We checked some values of weight decay for the L2 regularization. From out of the checked values we found out that the best weight decay value was 0.001, we got a training accuracy of 96% and 81.8% test accuracy.

Despite that, the best result for us came from the dropout regularization. With dropout = 0.5 we got training accuracy of 95% and 85% training accuracy. Making it the best regularization method that we checked.

**Summary Up To Here:  
What is your best architecture?**:  
Our best architecture is the following:  
Optimizer – ADAM  
Learning rate – 0.001  
Regularization Method- dropout = 0.5  
Batch normalization with Xaviar initialization  
Hidden dims – [150,150]  
Layer block = 8, Filter per layer = 64  
With this architecture we received a training accuracy of 95% and test accuracy of 85%.  
 **What its best accuracy and loss on the test set? Compare your result to assignment 2**:  
The best accuracy for the test set was 85% and loss of 0.56.  
In comparison to assignment 2, there the best result for the test set was 68% with 1.05 loss

**How did you mange to improve the model?**We managed to improve the model with number of factors, the first (and perhaps the most significant) is finding the most fitting learning rate to the problem. Afterwards we managed to add Batch normalization and Xaviar initialization (their advantages and uses are written above), those managed to further improve our test accuracy.  
Then we added Regularization. After checking between two regularization methods we found out that the dropout method gave us the best result and significantly improved out test accuracy.

**Experiment 5: experimenting with ResNet:**In this experiment we tested out ResNet with 3 different learning rate: 0.0001, 0.0005, 0.001  
we got our best results from the 0.0001 learning rate, so we will discuss it.  
  
**What was the pretrained ResNet50 accuracy and loss?**   
The training results on the pretrained ResNet50 was 99.4% and the loss was 0.018  
On the other hand, the test result was 83.75% with 0.822 loss  
 **Is it overfit/underfit/well-fit the data?**  
We think that it is in between overfit and well fit. On one hand we got 99.4% training result, which is a very high accuracy that may indicate overfit, but on the other hand, we got 83.75% test accuracy, which is also quite impressive accuracy.  
We are leaning more to the overfit option, and that because we got such high training accuracy but not as high test accuracy, with such high training accuracy we expect the test accuracy to be over 90%. This may point out that our model “memorized” (שינן) the given examples, and performed worse then expected.

**Has it got better accuracy than the non-pretrained ResNet?**   
No. on our previous attempt with “YourCodeNet” we got 95% training accuracy and 85% test accuracy which is better than our 83.75% test accuracy on ResNet.  
We came with the expectation that ResNet will perform better then YourCodeNet, but alas, we were proven wrong. The cause for that may be because the hyperparameters weren’t optimized to ResNet but to our YourCodeNet architecture.

**Project Summary**  
At the beginning of the project, we tested out different hidden dimension. We were conflicted with [100,100] and [150,150] hidden dimension, but further testing lead us to choosing the [150,150] hidden dim.  
Afterwards we tested out different learning rates. We tried out those 5 parameters in the list because we thought the distribution between the values were sufficient.  
The checking showed us that the best parameter was the 0.001 learning rate that we chose.  
In the third experiment we added batch normalization and weight initialization (Xaviar initialization). The batch normalization significantly improved our model performance, leading us to 78% test accuracy from previous 74%  
In the fourth experiment we added Regularization. First, we tried the L2 regularization method (added weight decay to ADAM), for our surprise, the best performance with the L2 regularization was with the lowest hyperparameter (0.0001), where we received 81% accuracy, an improvement of 3%.  
Second, we tested out dropout. We made the hyperparameter to L2 0 and tested dropout = 0.5  
The results were better than we expected. We got 95% training accuracy and 85% test accuracy. Outperforming anything we did before.  
  
In the last experiment we experimented with ResNet50.  
We checked out the ResNet50 with the following learning rates: 0.0001, 0.0005 and 0.001  
(0.001 is the best lr that we previously got).  
The results were a little bit disappointing, the best result we got was with learning rate of 0.0001, with 99.4% training accuracy and 83.75% test accuracy which may show that our model is overfited, we expected better test accuracy.  
  
**which architecture and hyperparameters led to the best preformance?**   
The best architecture was “YourCodeNet” with the following architecture:  
Optimizer – ADAM  
Learning rate – 0.001  
Regularization Method- dropout = 0.5  
Batch normalization with Xaviar initialization  
Hidden dims – [150,150]  
Layer block = 8, Filter per layer = 64  
With this architecture we received a training accuracy of 95% and test accuracy of 85%.

**BERT Article Summary**

BERT הוא מודל בעל ביצועים מדהימים, אשר מבוסס על ה-encoder ב-transformer, אשר פועל בעיקר בשלב ה-pre-training. בניגוד למודלי NLP אחרים אשר "קוראים" את טקסט הקלט משמאל לימין או ימין לשמאל, BERT קורא את כל הרצף בבאת אחת. BERT משתמש בעיקר בשני טכניקות:

Masked Language Modeling(MLM):

לפני שהמודל מעבד את המילים, 15% מהמילים(כולל תווים מיוחדים כגון EOS) בכל רצף מוחלפים בטוקן [MASK]. המודל מנסה לחזות את הערך המקורי של הטוקן, לפי הקונטסט של שאר המילים שלא החביאו אותם (כלומר שלא הפכו אותם ל-[MASK] ). כדי לממש זאת, מוסיפים שכבת classification לסוף ה-encoder, מכפילים את התוצאה במטריצת ה-embeddings, ומפעילים softmax על התוצאה כדי לחשב את ההסתברות של כל מילה באוצר-המילים להיות המילה המוסתרת. פונקציית ה-loss של BERT בודקת רק את הפרקדציות של המילים המוחבאות. כתוצאה מכך, המודל מתכנס לאט יותר, אבל יש לה מודל יותר מודעות context של המילים.

Next sentence Prediction(NSP):

בתהליך אימון BERT, המודל מקבל זוגות של שני משפטים ומנסה לחזות האם המשפט שני המשך של המשפט הראשון בטקסט המקורי.ב 50% מהזוגות המשפט השני הוא אכן המשך של המשפט הראשון, וב50% המשפט השני הוא משפט שנלקח באופן רנדומלי מהטקסט. כדי לעשות זאת, משנים את הקלט טיפה: מוסיפים טוקן [CLS] לאחר המשפט הראשון וטוקן [SEP] בסוף כל משפט, ועושים embedding כולל positional embedding.

לאחר השינויים, כל הקלט נכנס למודל והפלט של ה[CLS] טוקן נהפך לוקטור 2x1 שעליו מפעלים softmax כדי לחשב את התוצאה.

כשמאמנים מודל BERT, מאמנים את ה-MLM ואת ה NSP ביחד, כאשר המטרה היא להגיע ל-loss נמוך בשניהם.

אפשר להשתמש ב-BERT במשימות רבות, עם שינויים קלים למודל הבסיס בלבד,בנוסף רוב ה-hyper-parameters נשארים זהים(שלב זה הוא ה-fine-tuning).

לסיכום, BERT הוא מודל שימושי מאוד בעולם ה-NLP.

**Vit Article Summary**

Vit או Vision Transformes היא למעשה שיטה המתבססת על הtransformers המשומשים בטקסט. למעשה מדובר במודל דומה עם שינויים קטנים בו ככה שיותאם לדאטה המגיע מתמונה.  
השינויים מתבטאים בtokenization וב-embedding. למעשה מידע התמונה מתפצל למידעים קטנים יותר הנקראים visual tokens. ה visual tokens למעשה מוטבעים לווקטור בעל מימדים קבועים. מיקום הvisual tokens אשר מייצג את חלקו היחסי בתמונה מוטבע גם הוא בווקטור ולמעשה מזין את מידעיו   
למקודד הtransformers (שלו תפקיד דומה כמו בtransformers לטקסטים).   
ישנם במקודדים מספר בלוקים, בכל בלוק מתקיימים 3 תהליכים עיקריים: נרמול שכבה,  
 Multi-head Attention Network (MSP) ו- Multi-Layer Perceptrons (MLP).   
נרמול השכבה עוזר לתהליך האימון להתרחש ונותן למודל ללמוד את השינויים. הMSP אחראי על ייצור מפות של attention מהvisual tokens המוטבעים. מפות הattentions עוזרות למודל להתמקד על המקומות החשובים בתמונה בשביל תהליך הלמידה של האובייקט אותו אנו מעוניינים ללמוד.   
הMLP הינה רשת קלסיפיקציה בעלת שתי שכבות המשתמשת שבסופו היא משתמשת בGELU. למעשה הבלוק האחרון של הMLP משומש כתוצאה של הטרנספורמר, בה בסופו משתמשים בsoftmax על מנת לתת קלסיפיקציה התתאים לתוויות המוצאות.

**שימושים:**למעשה Vit משומש כמעט בכל ההיבטים של הvision, כולל קלסיפיקציה של תמונה, תמונה לטקסט וטקסט לתמונה ועוד. השימושים המוצלחים יותר הינם קלסיפיקציה של תמונה ותיאור של תמונה.  
  
**קלסיפיקציה של תמונה:**קלסיפיקציה של תמונה הינה אחת הבעיות הנפוצות בראייה ממוחשבת, Vit אינה מביאה תוצאות המשתוות לCNN בדאטה סטים קטנים או בינוניים, אך בדאטה סטים גדולים Vit מביאה תוצאות מוצלחות יותר.  
הCNN מוצלח בדאטה סטים קטנים יותר מכיוון שהוא מקדד את המידע באופן מקומי באופן יעיל, וזאת לעומת Vit שבה יש מוגבלות מקומית בשדה הקליטה.  
למעשה בVit אנו טוענים מודל מאומן, ואז לוקחים סט של תמונות רנדומליות (בין אם מהאינטרנט או ממקומות אחרים) ועם התמונות הללו אנו מאמנים את המודל, לאחר מכן אנו מקבלים את המיתוגים המתאימים ביותר לתמונות.  
  
**תיאור תמונה:**למעשה בעזרת הטכנולוגיה הזאת אנו יכולים ליצור תיאור של תמונות מהתמונות עצמם, וזאת על ידי ייצור של כותרות המתארות תמונה במקום מילה אחת המתארת אותה. Vit לומד רפרזנטציה כללית של דאטה מובנה במקום סט גס של תיאורים, על ידי כך ניתן ליצור תיאור מילולי יותר של תמונה.  
  
**יצירת תמונה מתיאור:**בעזרת Vit אנו יכולים לעשות פעולה מורכבת שקשה\לא ממש ניתן לעשות בשיטות אחרות, ליצור תמונה מתיאור. בשביל לעשות את הפעולה הנ"ל אנו זקוקים לתיאור מופשט יותר של טקסט ותמונה. ניתן לעשות זאת על ידי אימון של שני טרנספורמרים מקודדים שונים של הטקסט ושל התמונה. ניתן להשוות את התמונה המקודדת ואת הטקסט על ידי יצירת מטריצה קוסינוסים המתארת דמיון.   
ניתן לומר שלמעשה התהליך הבא מתקיים: ראשית אנו לוקחים כמה תמונות, לאחר מכן אנו כותבים טקסט המתאר אותן. הטקסט והתמונה מתקודדים לסט פיצ'ארים שמיוצג על ידי ווקטורים (וקטורים שונים ויחודיים לטקסט ולתמונה) וזאת על ידי שימוש במודל מאומן מראש של Vit.  
לאחר מכן אנו מחשבים את הדמיון בין הטקסט לתמונה על ידי מטריצת קוסינוסים, כך שככל שתוצאה בתא גבוהה יותר, כך יש דמיון רב יותר בין התמונה לטקסט.

**סיכום:**טכנולוגית הVision Transformers הינה טכנולוגיה חדשנית אשר ניתנת לשימוש בעקבות פיתוח הTransformers הרגיל וזאת על ידי שינוי במקודדים. הטכנולוגיה נתנה לנו שיפור במשימה הקלאסית של קלסיפיקציה של תמונה עם דאטה סט גדול ופתחה לנו דלתות לפיתרון בעיות כמו תיאור תמונות ואף יצירת תמונה חדשה מתיאור טקסט כללי ושיוך טקסט כלשהו לתמונה.