Tom Duany:206282477, Iris Dreizenshtok: 315790410 and Yuval Mor Yosef: 312475528

Assignment 4: Image super-resolution exercise

Contents

Stage 1: Selecting the algorithm for evaluation

בחרנו באלגוריתם:

“Sgdr: Stochastic gradient descent with warm restarts by Ilya Loshchilov, Frank Hutter”

a. תיאור האלגוריתם:

אלגוריתם הממש אופטימייזר Stochastic Gradient Decent עם מומנטום.

- תחילה נסביר מה הוא האופטימייזר הבסיסי, gradient decent, עוברים על כל הנתונים שיש לנו, מחשבים את הנגזרת בנקודה שאנחנו נמצאים בה ונעים בניגוד לגרדיאנט. האטא בנוסחה אומר כמה נזוז בניגוד לגרדיאנט. אם הוא מאוד קטן אנחנו נתכנס מאוד מאוד לאט. אם הוא מאוד גדול יכול להיות שלא נתכנס ואף נתבדר כלומר מצעד לצעד נלך לפתרון פחות טוב מהקודם). בשיטה זו אנו נתקדם רק לאחר מעבר על כל הנתונים ולכן יהיה מאוד קל להיתקע במינימום מקומי, ואין אקראיות כיוון שעוברים תמיד על כל הנתונים. האופטימייזר היותר מתקדם הוא Stochastic Gradient Decent: עדכון המשקולות לאחר כל חלק מהנתונים, דבר המאפשר מהירות ואקראיות בבחירה של הנתונים באותו חלק של נתונים (הגודל תלוי בגודל הram של הcpu).
- המאמר מציע אלגוריתם משופר לעיל ע"י שימוש במזמן עם מומנטום, צובר תאוצה, המשמעות היא שהוספת המומנטום עוזר לנו בהתמודדות עם מינימום מקומי ע"י כך שאם הוא צובר תאוצה ועובר את אותו מינימום מקומי, נגיע למצב הרצוי. נייצר צובר תאוצה בעזרת התחשבות בגרדיאנט הנוכחי בתוספת לאיזשהו משקל על הגרדיאנטים הקודמים.

Warm restarts - קצב למידה התחלתי גבוה לאחר הפעלה מחדש, משמש למעשה להקניית הפרמטרים מהמינימום שאליו הם התכנסו בעבר וזה מאפשר למודל להתכנס במהירות לפתרון חדש וטוב יותר.

ו

$$\eta_t = \eta_{min}^i + \frac{1}{2}(\eta_{max}^i - \eta_{min}^i)(1 + \cos(\frac{T_{cur}}{T_i}\pi)),$$

ו

השתמשנו בספרייה Keras שבאמצעותה בנינו את הרשת ומימשנו את הנוסחה מתוך המאמר המייצגת את המתזמן בשילובStochastic Gradient Decent עם מומנטום.

השתמשנו בפונקציית בשדה LearningRateScheduler של **Callbacks** כדי שתהיה לנו גישה לפרמטר lr שבעזרתו נוכל לבצע כיוונון ל- hyper parameters בשלה האימון.

b. יתרנות האלגוריתם:

- השגת ביצועים טובים יותר ברוב המקרים ע"י רשת עם אופטימייזר משופר.
- ע"י הרצה אחת בלבד של קובץ האימון, ניתן לקבל הרמה מודלים שניתן לשלב אותם (models to ensemble).
- תוצאות אימון הרשת מתכנסות מהר הרבה יותר ע"י הוספת המתזמן בהשוואה לכניקות בסיסיות.
- ביצועי האימון ע"י SGDR טובים אפילו כשמורכבות סט האימונים מורכב.
- השימוש במומנטום עוזר בהתמודדות עם מינימום מקומי.
- ניכרת למידה מאד מהירה ממעט דוגמאות במידה והדוגמאות בעלות תבנית זהה.

c. חסרנות האלגוריתם:

- שימוש בפרמטרים נוספים – הנוסחה של אופטימייזר SGDR דורשת שני ערכים אשר יש לבצע חישובים רבים עד שמוצאים את הערכים האידאליים המביאים ביצועים הגבוהיים ביותר. שמות הפרמטרים הם: T_0 ו- T_{Mult}.
- SGDR דורש כיוונון רב של הlearning rate והschedule ביחס לAdam למשל.

Stage 2: Suggesting an improvement

```
t0 = 10
ti = t0
tmult = 1
lr_warmup_current = 0.02 #max
lr_warmup_next = lr_warmup_current
tcur = 1
lr_min = 0

def lr_scheduler_improved(epoch, lr):

    global tcur
    tcur += 1
    global ti
    lr_scheduler_basic(epoch, lr)
    if tcur > ti:
        ti = int(tmult * ti)
        tcur = 1
        # global lr_warmup_current
        # lr_warmup_current = lr_warmup_next
    if ti==0:
      ti=0.0001
    lr = float(lr_min + (lr_warmup_current - lr_min) * (1 + np.cos(tcur/ti*np.pi)) / 2.0)
    return lr
```

בחרנו לשפר את האלגוריתם המוצג במאמר ע"י שילובו עם האלגוריתם המוכר (הבסיסי) בכך ושבאיפוק מסויים נבצע הקטנה בפאקטור וכן שימוש במזמן כך שנוכל להרוויח מיכולתו.

ע"י שיפור זה, אנחנו מקבלים שיפור במצב של התבדות מצד לצד, הימנעות ממצב של היתקעות במינימום מקומי והתחשבות בהיסטוריה של הLR.

Stage 3: Select a well-known algorithm for comparison

השתמשנו ברשת בסיסית בעלת אופטימייזר Adam אשר מתחשב בהיסטוריה של ה learning rate הקודם (במידה ויש כמה learning rates הוא יתייחס אליהם בעזרת משקלים ממושקלים, ככל שהlearning rate יותר עדכני הוא יקבל משקל יותר גדול). וכן בשני היפר פרמטרים.

השיפור הוא:

```
reduce_lr_epoch = 3
reduce_factor = 10

def lr_scheduler_basic(epoch, lr):
    if epoch == reduce_lr_epoch:
      print("Reduce lr from {} to {}".format(lr, lr/reduce_factor))
      return lr/reduce_factor
    else:
      return lr
```

בכל הרצה נקבע LR חדש וכן באיפוק מסויים שנקבע ע"י Random Search, אנחנו מקטינים את הLR בפאקטור שנקבע מראש, דבר התורם לנו במצב בו הlearning rate גבוה מידי ובמתבדר מצד לצד.

Stage 4: Evaluating the algorithms you selected in stages 1, 2 and 3

- **בניית הרשת:**

```
def build_model(lr,x_size,y_size):

  act_label=[]
  pred_label=[]
  model = Sequential()
  model.add(Dense(256, activation='relu', input_shape=(x_size,)))
  model.add(Dense(256, activation='relu'))
  model.add(Dense(256, activation='relu'))
  model.add(Dense(256, activation='relu'))
  model.add(Dense(y_size, activation='softmax'))
  optimizer = keras.optimizers.Adam(lr)
  model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy',keras.metrics.Precision()])
  return model
```

יצרנו מודל גנרי לשלושת האלגוריתמים כאשר פונקציית בנייתו מקבלת שלושה פרמטרים, Learning Rate – גודל התחלתי, גודל הפרמטרים וגודל הקלאסים, מכיוון שאנחנו עובדים בmulty labels.

- <u>Datasets:</u>
  - עבור כל Dataset ביצענו עיבוד בו העברנו את ה-class למטריצה כמספר הסיווגים האפשריים בעזרת ()to_categorical.

- בחרנו 20 Datases מתוך המאגר שעליהם בדקנו את שלושת האלגוריתמים.

20 Datases	
pittsburg-bridges-REL-L	teachingAssistant
post-operative	tae
acute-inflammation	veteran
pittsburg-bridges-TYPE	hayes-roth
pittsburg-bridges-MATERIAL	lowbwt
teaching	iris
acute-nephritis	prnn_synth
rabe_131	pittsburg-bridges-SPAN
chscase_vine1	pittsburg-bridges-T-OR-D_R
transplant	lenses

- <u>Evaluation Protocol:</u>
  - החלוקה לקובץ אימון וקובץ מבחן הינו:
    יצרנו פונקציה שמקבלת את X, Y המכילים את הנתונים שבאמצעותם חוזים ואת התוצאות, בהתאמה.
  - Hyperparameters - פרמטרים כמו Learnung Rate, במקרה שלנו, בחרנו עבור כל אלגוריתם, היפר פרמטרים ספציפיים ורלוונטיים שיעזרו לנו להגיע ליצירת מודל אידאלי.
    עבור האלגוריתם של המאמר, בחרנו: T0, Tmult שהם משתנים גם בנוסחה, ובנוסף LR.
    עבור האלגוריתם של המשופר, בחרנו: T0, Tmult שהם משתנים גם בנוסחה של המאמר, מספר epoch ובנוסף LR.
    עבור האלגוריתם של המוכר, בחרנו: מספר epoch ו- LR.

hyperParameters:

```
[ ] array_T0 = list(range(1, 51))
    array_Tmult =[1,2]
    array_lr = list(np.arange(0.00001, 0.1, 0.001))

    hp_warm = list(itertools.product(array_T0, array_Tmult,array_lr))
    random.shuffle(hp_warm)
    hp_warm = hp_warm[:50]
    # print(hp_warm)

    array_epoch_update_lr = list(range(1, 51))
    hp_basic = list(itertools.product(array_epoch_update_lr, array_lr))
    random.shuffle(hp_basic)
    hp_basic = hp_basic[:50]#epoch, lr
    # hp_basic

    hp_improved = list(itertools.product(array_epoch_update_lr,array_T0, array_Tmult,array_lr))
    random.shuffle(hp_improved)
    hp_improved = hp_improved[:50]#epoch, T0, Tmult,lr

    hp={"basic":hp_basic, "warm": hp_improved, "improved":hp_improved }
```

מכיוון שנזקקנו לבצע אופטימיזציה באמצעות Random Search ולא מצאנו תמיכה של Keras לשינוי מלבד בfit אז מימשנו בעצמנו ע"י מתן טווח לכל אחד מהפרמטרים, יצירת כלל האפשרויות ולקיחה באופן רנדומאלי של 50 קומבינציות.

```
def run(x,y):
  stage_name = ["basic","warm","improved"]
  cv_outer = KFold(n_splits=10, shuffle=True, random_state=1)
  i =0
  save_best_models=[]
  save_best_hp=[]
  for train_index, test_index in cv_outer.split(x,y):
    i += 1
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]
    cv_inner = KFold(n_splits=3, shuffle=False, random_state=1)
    val_acc_inner = {}
    for j, (train_inner_idx, val_inner_idx) in enumerate(cv_inner.split(x_train, y_train)):
      print("10kfold: ",i , "3kfold: ",j)
      x_train_inner, x_val_inner = x[train_inner_idx], x[val_inner_idx]
      y_train_inner, y_val_inner = y[train_inner_idx], y[val_inner_idx]
      val_acc_inner[j] = (hp_search(x_train_inner, x_val_inner, y_train_inner, y_val_inner, hp[stage_name[2]], stage_name[2]))
    # Do average between lists
    val_acc_inner_avg = []
    for k in range(0,len(val_acc_inner)):
      print("k: ", k)
      val_acc_inner_avg.append((val_acc_inner[0][k]+val_acc_inner[1][k]+val_acc_inner[2][k])/3)
    max_acc = max(val_acc_inner_avg)
    max_index_of_acc = val_acc_inner_avg.index(max_acc)
    best_hp = hp[stage_name[2]][max_index_of_acc]
    best_train_history = train_model(x_train,y_train,x_test, y_test, best_hp, stage='improved', epochs=50)
    save_best_models.append(best_train_history.history)
    save_best_hp.append(best_hp)
  return save_best_models,save_best_hp
```

- תחילה ביצענו 10fold Cross Validation כל fold שלחנו ל-3 fold Cross Validation שבו ביצענו קריאה לפונקציית hp_search אשר מימשנו כי לא מצאנו מימוש בkeras המתאים לדרישות.

```
def hp_search(x_train_inner, x_val_inner, y_train_inner, y_val_inner, hp, stage_name):
    # Return a tuple length len(hp) with each field contain the val accuracy of the corresponding hp
    val_acc = []
    counter = 0
    # for hyperparameters in hp:
    # print(hp[0:5])
    for hyperparameters in hp:
      print("hp_search counter:", counter)
      train_history = train_model(x_train_inner,y_train_inner,x_val_inner, y_val_inner, hyperparameters, stage=stage_name, epochs=50)
      # print("times: {}", format(train_history.history['train_time']))
      val_acc.append(train_history.history['val_accuracy'][-1])

      counter+=1
    return val_acc
```

בפונקציה זו ביצענו את הקריאות לtrain_model.

גילוי נאות: מימשנו קוד המבצע 50 קריאות פנימיות אך מפאת חוסר במשאבים, ביצענו 5 קריאות פנימיות.

- לא נוכל לבחור את המודל בעל התוצאות הטובות ביותר כי זה לא מחייב שאלו יהיו גם התוצאות הכי טובות עבור הdata המלא ולכן ביצענו ממוצע בין שלוש התוצאות הפנימיות ולקחנו את ההיפר פרמטר המתאים עפ"י הערך הגבוה שמצאנו בהתאמה.

```
def train_model(x_train, y_train,x_test, y_test, hp, stage, epochs=50):

  # before = timeit.default_timer()
  if stage=='basic':
    model = build_model(hp[1],x_train.shape[1],y_train.shape[1])
    # print("Train model with lr: {}".format(hp[1]))
    global reduce_lr_epoch
    reduce_lr_epoch = hp[0]
    callbacks_lr = LearningRateScheduler(lr_scheduler_basic, verbose=1)
  if stage=='warm':
    model = build_model(hp[2],x_train.shape[1],y_train.shape[1])
    # print("Train model with lr: {}".format(hp[1]))
    global tmult
    tmult = hp[1]
    global ti
    ti = hp[0]
    callbacks_lr = LearningRateScheduler(lr_scheduler_warm, verbose=1)
  if stage=='improved':
    model = build_model(hp[3],x_train.shape[1],y_train.shape[1])
    # print("Train model with lr: {}".format(hp[1]))
    # global tmult
    tmult = hp[3]
    # global ti
    ti = hp[1]
    # global reduce_lr_epoch
    reduce_lr_epoch = hp[0]
    callbacks_lr = LearningRateScheduler(lr_scheduler_improved, verbose=1)

  time_callback = TimeHistory()
  history = model.fit(x_train,y_train, validation_data=(x_test, y_test), batch_size=256, epochs=epochs, verbose=1,callbacks=[time_callback, callbacks_lr])

  history.history['train_time'] = time_callback.times
  return history
```

- פונקציה זו מעדכנת את ערכי ההיפר פרמטרים לפי כל ריצה באמצעות callbacks, בנוסף השתמשנו עבור חישוב הזמנים.

```
import time
class TimeHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.times = []

    def on_epoch_begin(self, batch, logs={}):
        self.epoch_time_start = time.time()

    def on_epoch_end(self, batch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)
```

- מדובר במודל קלאסיפיקציה ולכן לקחנו Dataset מתוך classification_datasets.

- <u>Metrics:</u>
  - Hyper-Parameters Values – מה שאנו קובעים לגבי אופן הלמידה של המודל הlearning rate.
  - Accuracy – נכונות המודל.
  - Precision – דיוק המודל.
  - Training Time – מדדנו את הזמן מתחילת תהליך ה – 'fit' ועד הסוף.
  - Inference Time –
  - **PR-Curve, AUC , FPR , TPR**
  - ראינו לנכון שאין צורך להציג מידע אודות פרמטרים שלהלן מכיוון שה Data שלנו קטן והמידע אינו סיפק לנו מידע שהועיל לנו.

מצורף קובץ אקסל המכיל את התוצאות:

דוגמה:

Dataset Name	Algorithm Name	Cross Validation	Hyper-Parameters Values	Accuracy	Precision	Training Time	Inference Time
pittsburg-bridges-SPAN.csv	Basic	1	(20, 0.06501)	[0.8780487775802612]	[0.8780487775802612]	3.179732084	47.79137323
pittsburg-bridges-SPAN.csv	Basic	2	(20, 0.06501)	[0.8292682766914368]	[0.8292682766914368]	3.043545961	45.7444958
pittsburg-bridges-SPAN.csv	Basic	3	(20, 0.06501)	[0.8433734774589539]	[0.8500000238418579]	2.879115343	43.27310361
pittsburg-bridges-SPAN.csv	Basic	4	(31, 0.06101)	[0.8192771077156067]	[0.8374999761581421]	3.048721313	45.82228134
pittsburg-bridges-SPAN.csv	Basic	5	(20, 0.06501)	[0.759036123752594]	[0.762499988079071]	3.020359039	45.39599636
pittsburg-bridges-SPAN.csv	Basic	6	(34, 0.07400999999999999)	[0.7108433842658997]	[0.7108433842658997]	3.068953991	46.12637848
pittsburg-bridges-SPAN.csv	Basic	7	(20, 0.06501)	[0.7710843086242676]	[0.7710843086242676]	2.987842798	44.90727726
pittsburg-bridges-SPAN.csv	Basic	8	(34, 0.07400999999999999)	[0.9156626462936401]	[0.9156626462936401]	2.778160095	41.75574623
pittsburg-bridges-SPAN.csv	Basic	9	(34, 0.07400999999999999)	[0.7349397540092468]	[0.7692307829856873]	3.141021967	47.20956016
pittsburg-bridges-SPAN.csv	Basic	10	(20, 0.06501)	[0.8795180916786194]	[0.9090909361839294]	2.98210454	44.82103123

Dataset Name	Algorithm I	Cross Validation [1-10]	Hyper-Parameters Values	Accuracy	Precision	Training Time	Inference Time
pittsburg-bridges-SPAN.csv	SGDR	1	(22, 2, 0.00101)	[0.9146341681480408]	[0.9146341681480408]	4.116652489	61.87328691
pittsburg-bridges-SPAN.csv	SGDR	2	(29, 1, 0.06901)	[0.9512194991111755]	[0.9512194991111755]	4.624874353	69.51186153
pittsburg-bridges-SPAN.csv	SGDR	3	(22, 2, 0.00101)	[0.8313252925872803]	[0.8313252925872803]	4.004602432	60.18917456
pittsburg-bridges-SPAN.csv	SGDR	4	(22, 2, 0.00101)	[0.9277108311653137]	[0.9277108311653137]	4.122070074	61.95471321
pittsburg-bridges-SPAN.csv	SGDR	5	(22, 2, 0.00101)	[0.9277108311653137]	[0.9277108311653137]	4.311454535	64.80116165
pittsburg-bridges-SPAN.csv	SGDR	6	(29, 1, 0.06901)	[0.9518072009086609]	[0.9518072009086609]	4.230414629	63.58313187
pittsburg-bridges-SPAN.csv	SGDR	7	(29, 1, 0.06901)	[0.9759036302566528]	[0.9759036302566528]	3.975452662	59.7510535
pittsburg-bridges-SPAN.csv	SGDR	8	(22, 2, 0.00101)	[0.9397590160369873]	[0.9397590160369873]	3.988569498	59.94819956
pittsburg-bridges-SPAN.csv	SGDR	9	(11, 2, 0.050010000000000006)	[0.9638554453849792]	[0.9638554453849792]	4.119925261	61.92247667
pittsburg-bridges-SPAN.csv	SGDR	10	(29, 1, 0.06901)	[0.9518072009086609]	[0.9518072009086609]	3.924676657	58.98789015

Dataset Name	Algorithm Name	Cross Validation [1-10]	Hyper-Parameters Values	Accuracy	Precision	Training Time	Inference Time
pittsburg-bridges-SPAN.csv	improved	1	(23, 9, 2, 0.034010000000000006)	[0.9390243887901306]	[0.9390243887901306]	2.796408653	42.03002206
pittsburg-bridges-SPAN.csv	improved	2	(7, 13, 2, 0.033010000000000005)	[0.9634146094322205]	[0.9634146094322205]	2.910122156	43.73913601
pittsburg-bridges-SPAN.csv	improved	3	(23, 9, 2, 0.034010000000000006)	[0.9638554453849792]	[0.9638554453849792]	3.401731968	51.12803148
pittsburg-bridges-SPAN.csv	improved	4	(23, 9, 2, 0.034010000000000006)	[0.9638554453849792]	[0.9638554453849792]	2.989461184	44.93160159
pittsburg-bridges-SPAN.csv	improved	5	(33, 18, 2, 0.07801)	[0.9397590160369873]	[0.9397590160369873]	2.768976927	41.61772321
pittsburg-bridges-SPAN.csv	improved	6	(23, 9, 2, 0.034010000000000006)	[0.9397590160369873]	[0.9397590160369873]	2.956847906	44.44142403
pittsburg-bridges-SPAN.csv	improved	7	(7, 13, 2, 0.033010000000000005)	[0.9518072009086609]	[0.9518072009086609]	2.843258381	42.73417346
pittsburg-bridges-SPAN.csv	improved	8	(7, 13, 2, 0.033010000000000005)	[0.9759036302566528]	[0.9759036302566528]	2.82281208	42.42686557
pittsburg-bridges-SPAN.csv	improved	9	(23, 9, 2, 0.034010000000000006)	[0.9879518151283264]	[0.9879518151283264]	2.817795038	42.35145942
pittsburg-bridges-SPAN.csv	improved	10	(33, 18, 2, 0.07801)	[0.9518072009086609]	[0.9518072009086609]	2.998357296	45.06531016

Stage 5: Statistical significance testing of the results

מבחן הוק & מבחן פרידמן:

מבחן פרידמן – השתמשנו במבחן זה על מנת לבדוק האם קיים מובהק בין המודל הבסיסי, המודל המוצע והמודל המושפר או לחילופין שלא קיים הבדל. המדד להשוואה הינו Accuracy. השתמשנו ב – stats. Friedmanchisquare. הרצה כוללת 600 תוצאות, 200 לכל אלגוריתם.

- statistic=77.09138381201097
- , pvalue=1.8189413843193766e-17

```
Friedman test
FriedmanchisquareResult(statistic=77.09138381201097, pvalue=1.8189413843193766e-17)
```

pvalue < 0.05

דחינו את השערת האפס ונסיק שקיים הבדל בין האלגוריתמים. כעת נבצע מבחן הוק.

מבחן הוק:

במבחן הוק השתמשנו בהפלט של אלגוריתם פרידמן. פונקציה הוק יוצרת מטריצה X על X ככמות אלגוריתמים. כל תא מייצג pvalue של אלגוריתם אחד מול השני . השתמשנו ב - scikit_posthocs.posthoc_nemenyi_friedman והגדרנו את α=0.05.

במידה והערך(pvalue) קטן מ α הקבוצות שונות. - α >pvalue .

מטריצת הוק:

```
Hoc test
       0        1        2
0  1.000  0.00100  0.00100
1  0.001  1.00000  0.00595
2  0.001  0.00595  1.00000
```

- ע"פ מבחן הוק ניתן להסיק כי קיים הבדל בין האלגורתים הבסיסי (Basic) לבין אלגוריתם שהוצע ע"י המאמר(SGDR).

- ע"פ מבחן הוק ניתן להסיק כי קיים הבדל בין האלגורתים הבסיסי (Basic) לבין אלגוריתם המשופר (improve).
- ע"פ מבחן הוק ניתן להסיק כי קיים הבדל בין האלגוריתם שהוצע ע"י המאמר (SGDR) לבין אלגוריתם המשופר (improve), מכאן שהשיפור שביצענו השפיע על תוצאות – לחיוב או לשלילה.

Stage 6: conclusions

- האלגוריתם SDGR המתואר במאמר, אכן מציג שיפור ל Stochastic gradient descent וניתן לראות זאת גם עפ"י הנוסחה שלו:

$$w := w - \eta \nabla Q_i(w).$$

קיימת התייחסות לגרדיאנט אך יכולה להיווצר בעיה של מינימום מקומי ללא פתרון ולכן המתזמן במאמר פותר זאת יחד עם המומנטום.

- האלגוריתם המוצג במאמר אכן הציג תוצאות טובות יותר מהאלגוריתם הבסיסי שיצרנו.
- האלגוריתם המשופר הציג תוצאות שוות או משופרות ולכן אנחנו מסיקים כי לעיתים הקטנה בבפאקטור של הLR אכן תורמת במזעור נקים של התבדרות.
- נוכחנו לדעת ולהבין את משמעות האופטימיזציה, דרך היצירה וכן הבנה עמוקה של משך זמני הריצה.
- השתמשנו בפלפטורמה של google colab כי מספק GPU , דבר שהיה נחוץ מאד בעבודה זו.

Stage 7: References

Out git code –

https://github.com/tomdua/Final-Course-Project-2021---SGDR.git

Source code git –

https://github.com/loshchil/SGDR

Article –

https://arxiv.org/abs/1608.03983

Summary article –

https://debuggercafe.com/stochastic-gradient-descent-with-warm-restarts-paper-explanation/

Data –

https://drive.google.com/drive/folders/1UBgL2M0JHLN6HLRhC8yqK4hFv4QlQXZT?usp=sharing

Colab notebook`s:

Basic –

https://colab.research.google.com/drive/1xfkf77W__Ofnm9S3VvunVMVWSB56IC1k?usp=sharing

Sgdr -

https://colab.research.google.com/drive/1TDbkKr6TpWjlGpC4Wcyt7UsLrk7OTEx3?usp=sharing

Improve –

https://colab.research.google.com/drive/1n2ZuDjsFuPxA8sg5cf0GO_zOukELTUKo?usp=sharing

Stage 5 -

https://colab.research.google.com/drive/1-nBaZZmpnKVnIJ0dwqK2gdLWpSnMB_Oc?usp=sharing

Excel results –

https://drive.google.com/file/d/1SJu6y1sFxJSvYCb1NmGqmD1xnPKkfcqO/view?usp=sharing

Csv results –

https://drive.google.com/file/d/1SJu6y1sFxJSvYCb1NmGqmD1xnPKkfcqO/view?usp=sharing