Лабораторна робота 5 – Штучні нейронні мережі.

Завдання 1. Розв'яжіть лабораторну №3 за допомогою нейронної мережі. Кількість шарів, нейронів, метод оптимізації – на Ваш розсуд. Дозволяється використання обгортки Keras для побудови моделі, однак реалізація на чистому TensorFlow (або PyTorch) заохочується додатковими балами.

Завдання було виконано за допомогою TensorFlow та Keras.

Код класу LogisticRegression з використанням чистого TensorFlow:

```
24  class LogisticRegression(tf.Module):
25      def __init__(self, x):
26          self.mean = tf.Variable(tf.math.reduce_mean(x, axis=0))
27          self.std = tf.Variable(tf.math.reduce_std(x, axis=0)) + pow(10, -8)
28      def normalize(self, x):
29          return (x - self.mean)/self.std
30      def GetTrainAndTestDataSets(self, BatchSize, x_train, y_train, x_test, y_test):
31          TrainDataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
32          TrainDataset = TrainDataset.shuffle(buffer_size=x_train.shape[0]).batch(BatchSize)
33          TestDataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
34          TestDataset = TestDataset.shuffle(buffer_size=x_test.shape[0]).batch(BatchSize)
35          return TrainDataset, TestDataset
36      def ComputeTheModelOutput(self, x, w, b):
37          return tf.nn.softmax(tf.add(tf.matmul(x, w), b))
38      def GetAccuracy(self, y_pred, y):
39          prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
40          return tf.reduce_mean(tf.cast(prediction, tf.float32))
41      def GetLoss(self, y_pred, y):
42          return tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=y_pred)
43      def RunModel(self, CountOfEpochs, LearningRate, TrainDataset, TestDataset,
        DetailedOutputOn):
44          w, b = tf.Variable(tf.zeros([784,10])), tf.Variable(tf.zeros([10]))
```

```
87              for epoch in range(CountOfEpochs):
88                  BatchTrainLosses, BatchTrainAccuracies = [], []
89                  BatchTestLosses, BatchTestAccuracies = [], []
90                  for XBatch, YBatch in TrainDataset:
91                      with tf.GradientTape() as tape:
92                          YPredBatch = self.ComputeTheModelOutput(XBatch, w, b)
93                          BatchLoss = self.GetLoss(YPredBatch, YBatch)
94                      BatchAccuracy = self.GetAccuracy(YPredBatch, YBatch)
95                      gradients = tape.gradient(BatchLoss, [w, b])
96                      for g, v in zip(gradients, [w, b]):
97                          v.assign_sub(LearningRate * g)
98                      BatchTrainLosses.append(BatchLoss)
99                      BatchTrainAccuracies.append(BatchAccuracy)
100                     gradients=[]
101                     tape=[]
102                 for XBatch, YBatch in TestDataset:
103                     YPredBatch = self.ComputeTheModelOutput(XBatch, w, b)
104                     BatchLoss = self.GetLoss(YPredBatch, YBatch)
105                     BatchAccuracy = self.GetAccuracy(YPredBatch, YBatch)
106                 BatchTestLosses.append(BatchLoss)
107                 BatchTestAccuracies.append(BatchAccuracy)
108                 TrainLoss, TrainAccuracy = tf.reduce_mean(BatchTrainLosses), tf.reduce_mean(
                    BatchTrainAccuracies)
109                 TestLoss, TestAccuracy = tf.reduce_mean(BatchTestLosses), tf.reduce_mean(
                    BatchTestAccuracies)
110         return TrainAccuracy, TestAccuracy, TrainLoss, TestLoss
```

Опис проведених досліджень

Реалізація логістичної регресії за допомогою TensorFlow для вирішення задач багатокласової та бінарної класифікації дещо відрізняється: для обчислення функції активації та коефіцієнта втрат використовувався інструмент softmax замість sigmoid.

На відміну від попередньої роботи, ми аналізували як точність, так і значення втрат на тренувальних та тестових даних, а також кількість навчальних епох, проте вплив поліноміальності та коефіцієнтів alpha1 і alpha2 не досліджувались.

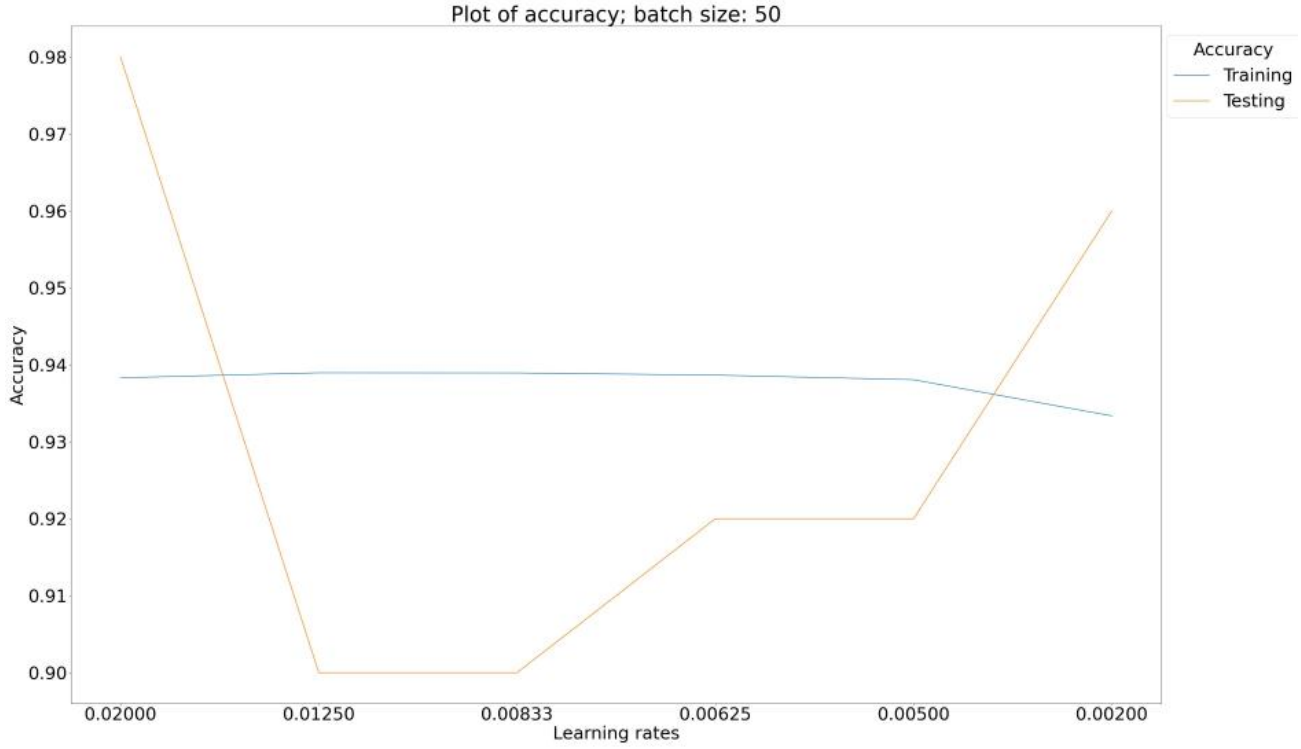
Запустивши код із стандартними параметрами, ми отримали наступний вивід:

```
Count of epochs:  10 ; batch size:  32 ; learning rate:  0.00100...
Training accuracy.:  0.92805 ; testing accuracy.:  0.93750;
Training loss:  1.54361 ; testing loss:  1.52456; time:  0 : 1 : 17
```

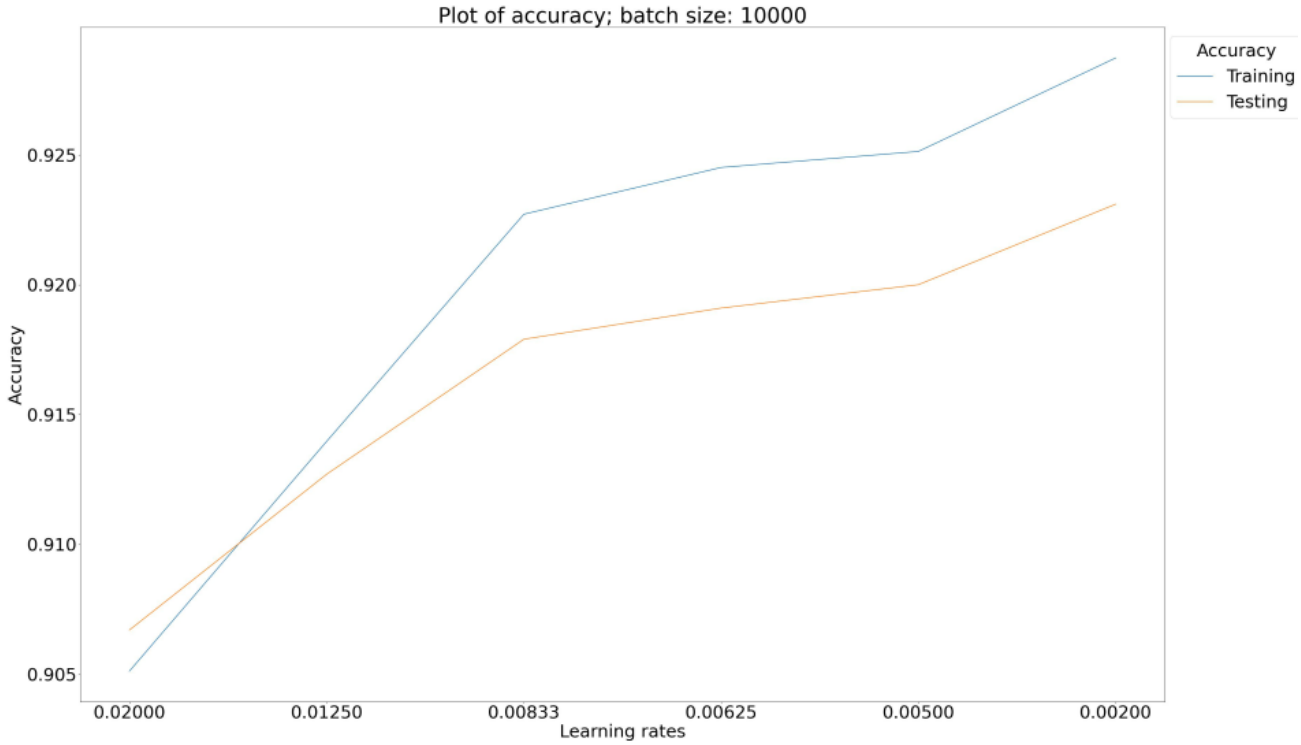
Видно, що точність нашої моделі на тестових даних, реалізованої за допомогою за допомогою TensorFlow, виявилась більшою на 6.56%.

Як і в попередніх роботах, з ціллю оптимізації ми винесли деякі підготовчі дії в окремий метод, який виконується перед початком нової серії експериментів з іншим розміром частини вибірки (BatchSize). Програма також містить код для визначення часових витрат, фіксації результатів у вигляді графіків, діаграм різних типів та текстового csv-файлу, який при необхідності може бути проаналізований табличним процесором.

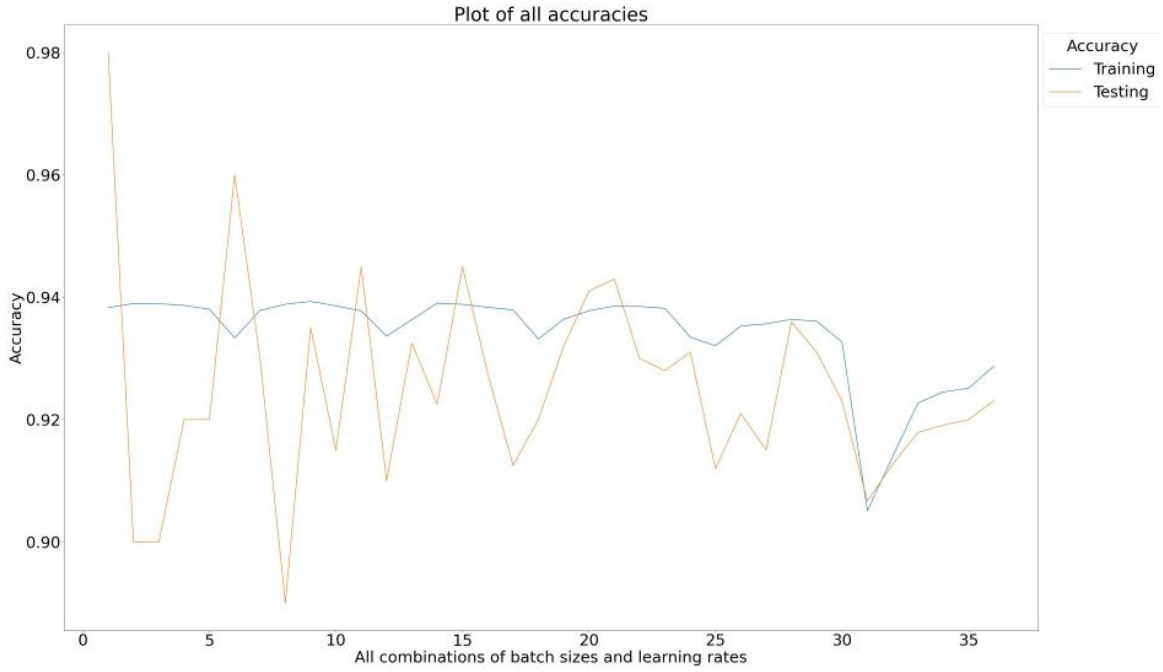
Тестування моделі включало в себе перебір різних розмірів частини вибірки (BatchSize) з коефіцієнтами швидкості навчання (LearningRate), а кількість епох при цьому була 10. Після визначення найкращих значень цих параметрів було виконано один експеримент із кількістю епох 1000. Наведемо графік, який ілюструє зміну точності при певному BatchSize з різними значеннями LearningRate:



Коефіцієнт швидкості навчання достатньо сильно впливає на якість моделі, проаналізованої на тестових даних, та менше впливає на точність при аналізі тренувальної вибірки. Схожу історію помітно при інших BatchSize окрім 10000:

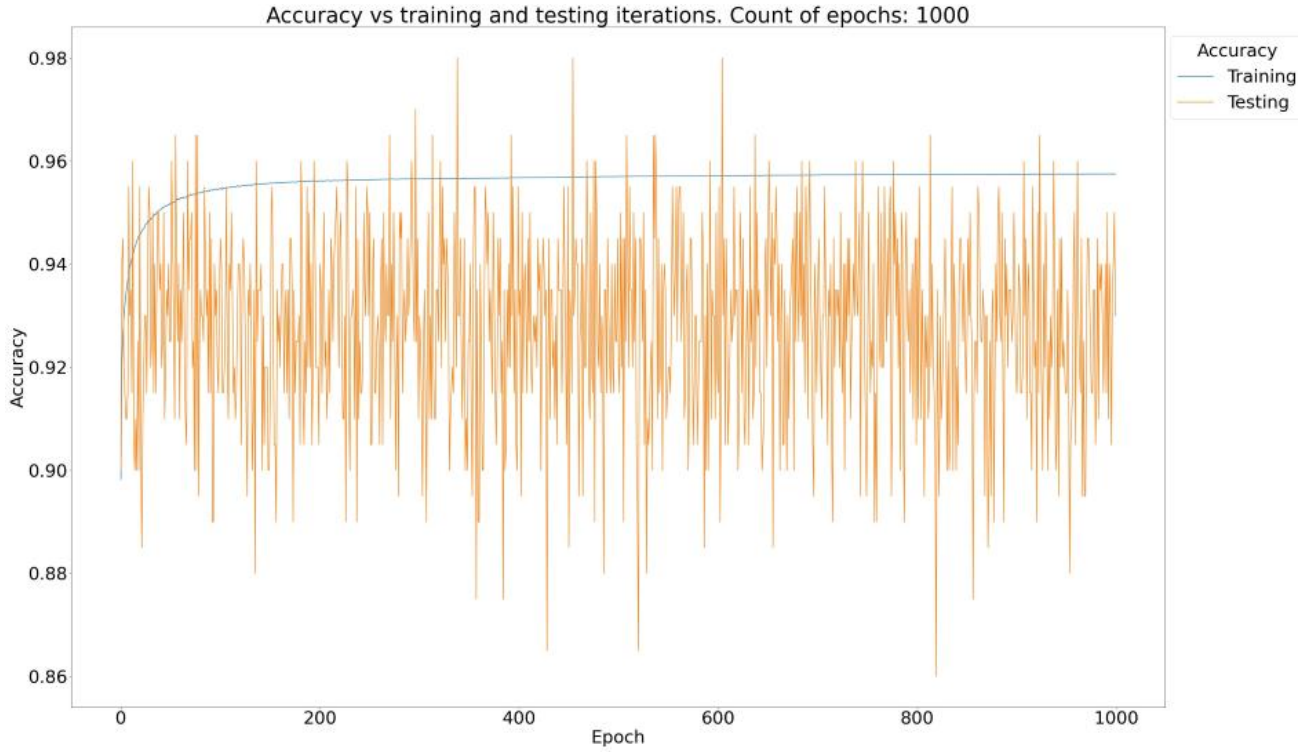


Тут ми бачимо правильну тенденцію, коли зростання коефіцієнта швидкості навчання призводить до збільшення точності моделі на всіх двох типах даних, причому рівень якості на тестовій вибірці у переважній більшості випадків нижчий за тренувальний. Графіки функції втрат є повністю оберненими до наданих. Проілюструємо глобальну зміну якості моделі при всіх комбінаціях описаних параметрів на тренувальних та тестових даних:



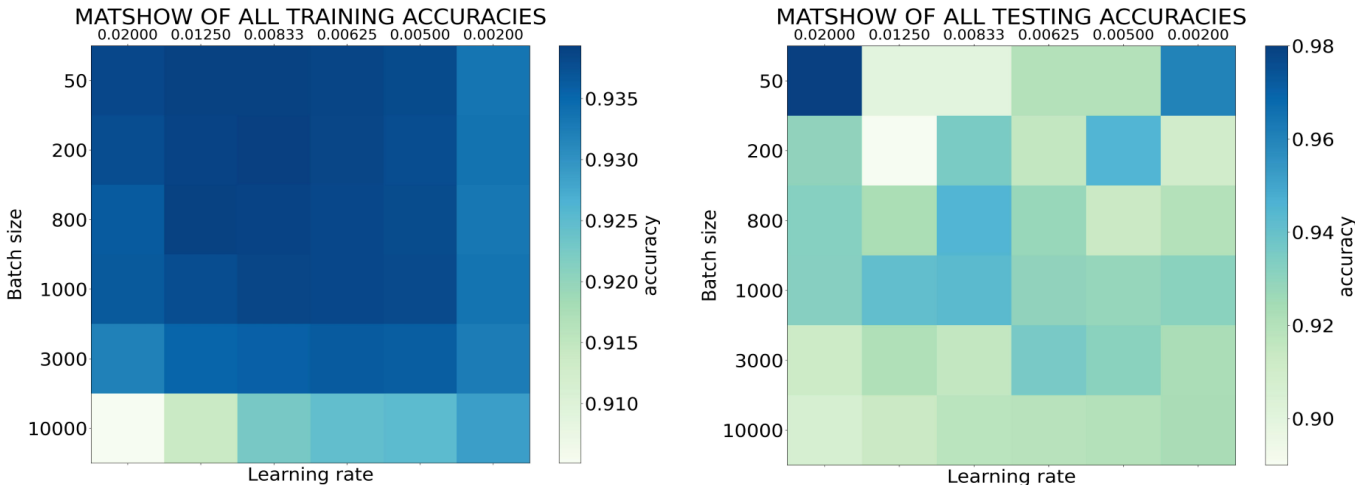
Видно, що точність, отримана на тренувальній вибірці, є більш стабільною.

Наведемо графік зміни якості моделі на кожній з епох:



Ми бачимо подібну тенденцію: точність на тренувальній вибірці є набагато стабільнішою за точність на тестовій вибірці, і в більшості випадків вона є вищою.

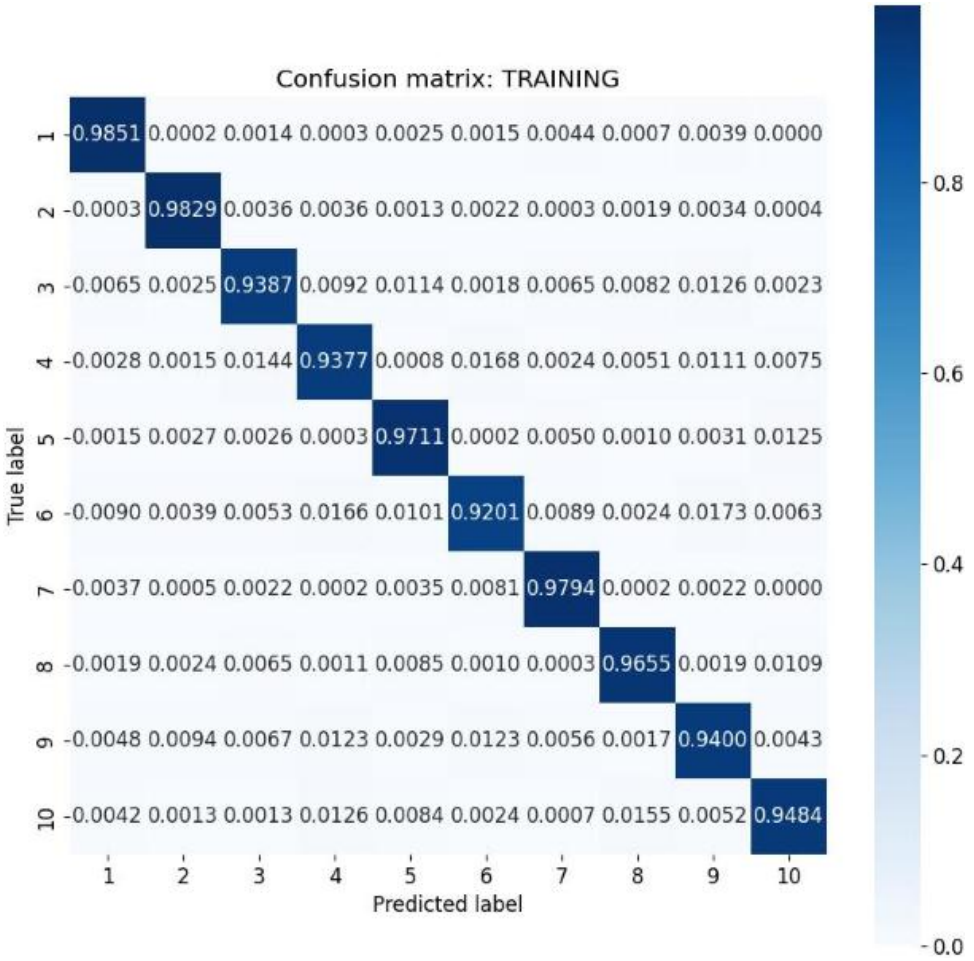
Такі ж результати показують і кольорові діаграми:

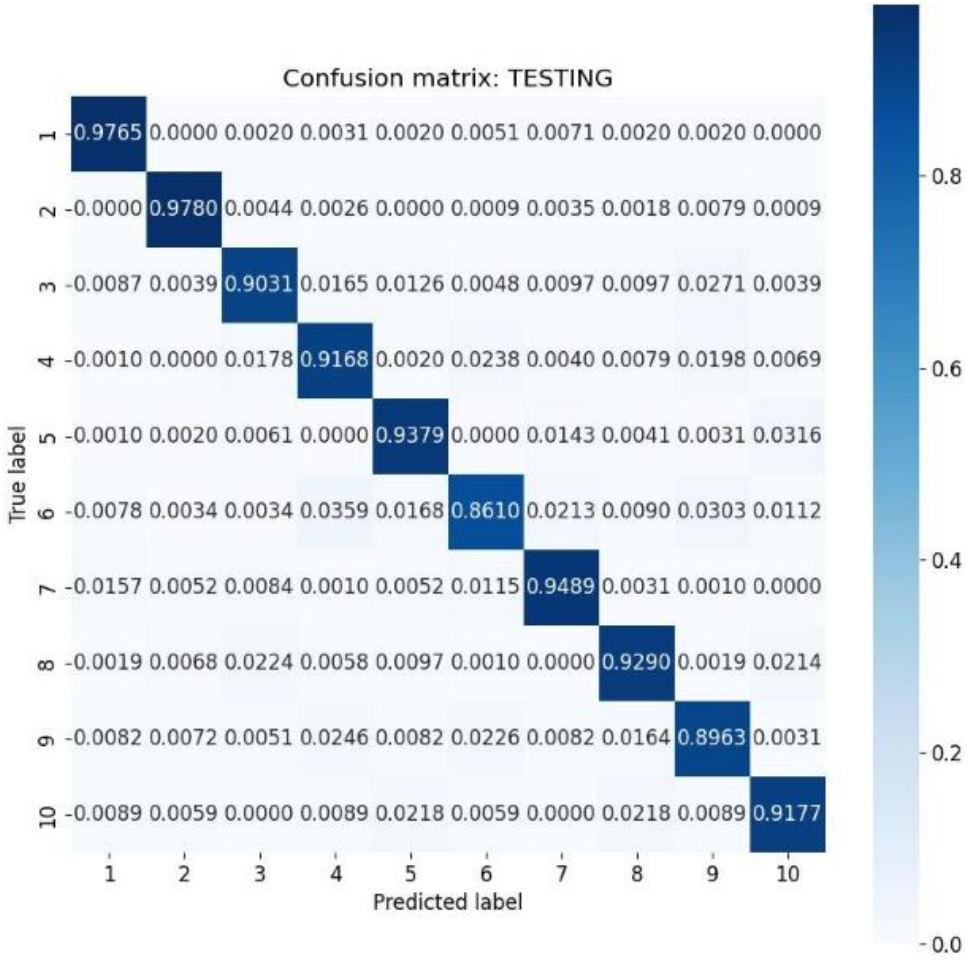


Після завершення всіх випробувань був отриманий наступний вивід на екран:

```
Epoch  998 / 1000... Training accuracy.:  0.95752 ; testing accuracy.:  0.94000
Training loss:  1.50404 ; testing loss:  1.52290;
------------------------------------------------------------------------------
Epoch  999 / 1000... Training accuracy.:  0.95752 ; testing accuracy.:  0.95000
Training loss:  1.50404 ; testing loss:  1.51187;
------------------------------------------------------------------------------
Epoch  1000 / 1000... Training accuracy.:  0.95752 ; testing accuracy.:  0.93000
Training loss:  1.50403 ; testing loss:  1.53117;
------------------------------------------------------------------------------
Training accuracy.:  0.95752 ; testing accuracy.:  0.93000;
Training loss:  1.50403 ; testing loss:  1.53117; time:  0 : 25 : 43
Total time of all calculations:  0 : 33 : 51
Building plots... Time:  0 : 0 : 7
```

Для кращої моделі були побудовані діаграми типу «Confusion matrix»:





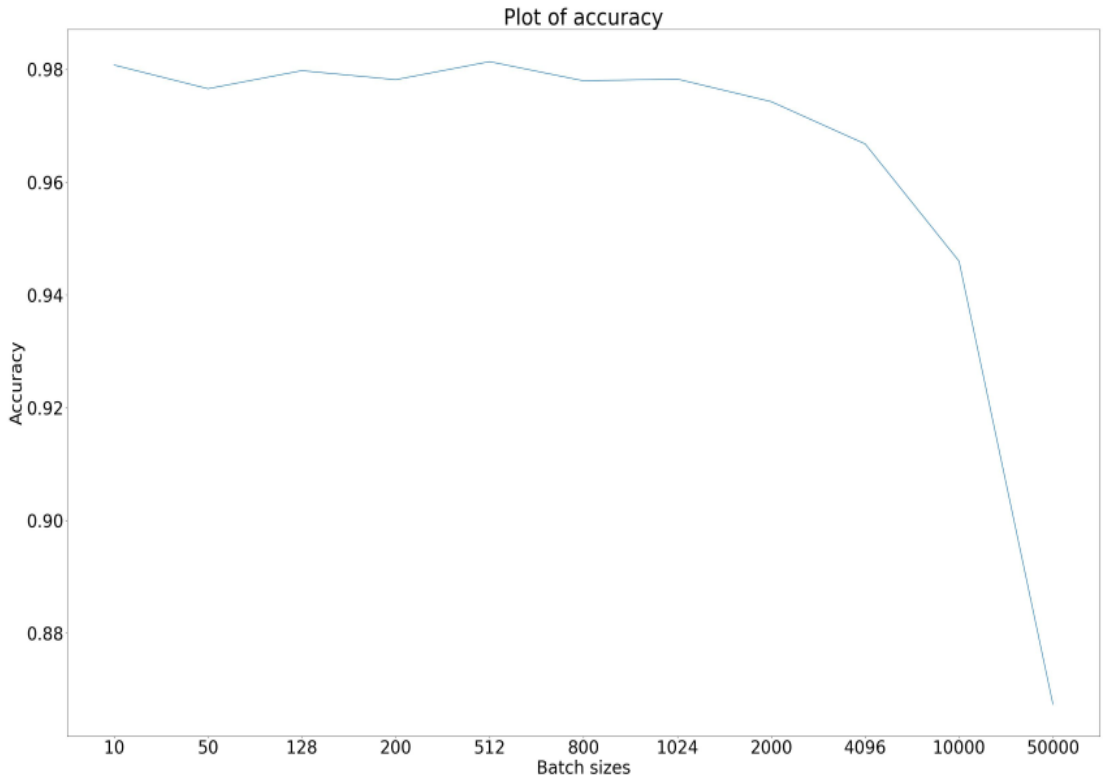
Завдання даної лабораторної роботи також було виконано за допомогою Keras:

```
31  def KerasLogisticRegression(x_train, y_train, x_test, y_test, CountOfEpochs, BatchSize,
    DetailedOutputOn):
32      model = Sequential()
33      model.add(Dense(512, input_dim=28 * 28, activation='relu', name='Hidden-1'))
34      model.add(Dense(256, activation='relu', name='Hidden-2'))
35      model.add(Dense(10, activation='softmax', name='Output'))
36      model.compile('adam', loss='categorical_crossentropy', metrics=['accuracy'])
37      history = model.fit(x_train, y_train, epochs=CountOfEpochs, batch_size=BatchSize,
        validation_split=0.2, verbose=0)
38      AccuracyAndLoss = model.evaluate(x=x_test, y=y_test, verbose=0)
39      if DetailedOutputOn:
40          train_pred = model.predict(x_train, verbose=0)
41          test_pred = model.predict(x_test, verbose=0)
42          return AccuracyAndLoss[1], AccuracyAndLoss[0], history, train_pred, test_pred
43      return AccuracyAndLoss[1], AccuracyAndLoss[0]
```

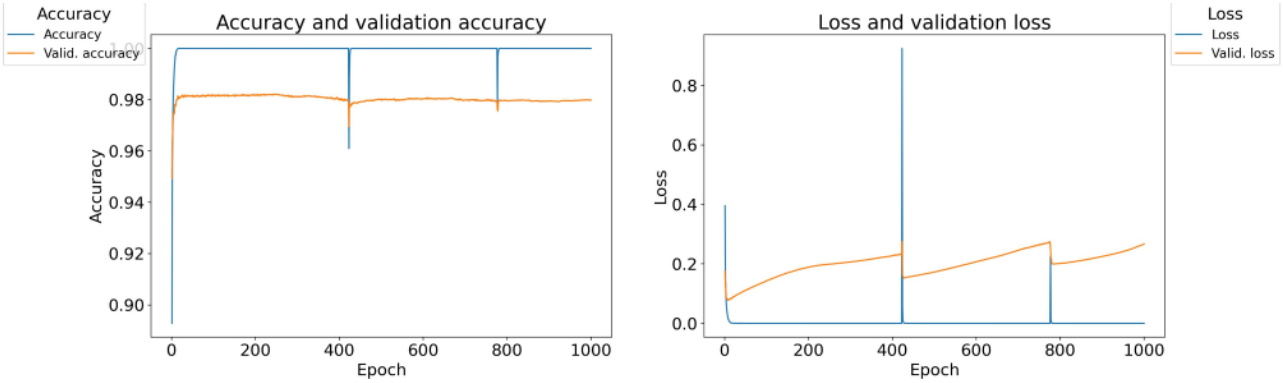
Ми бачимо, що модель, створена за допомогою Keras, майже на 5% точніша, ніж TensorFlow з параметрами, як у попередніх двох роботах:

```
Count of epochs:  10 ; batch size:  32... Time:  0 : 1 : 34
Accuracy:  0.97560 ; loss:  0.12142
```

Оскільки обраний метод оптимізації («adam») автоматично задає значення параметра LearningRate (коефіцієнт швидкості навчання) в залежності від отриманого градієнта, ми його не аналізували, хоча в Keras, можливо, є способи змінити його. Тому ми перевіряли вплив BatchSize (розмір частини вибірки) на точність та коефіцієнт втрат лише на тестових даних, з кількістю епох 10:

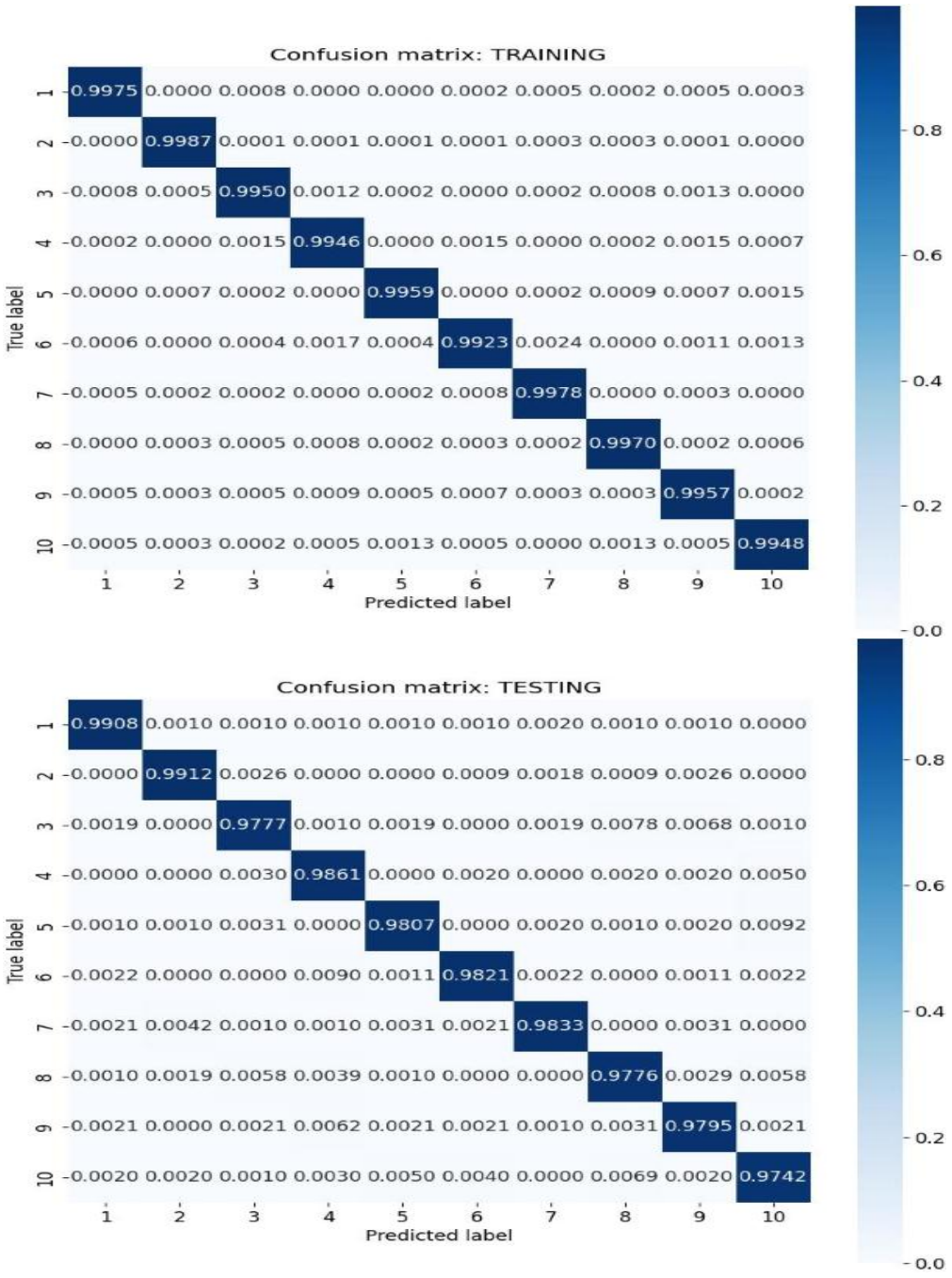


Видно, що розмір частини вибірки, більший за 2000, призводить до стрімкого погіршення якості моделі. Після отримання найкращого значення (512) було виконано один експеримент з кількістю епох 1000, отримано такі графіки:



Можна помітити, що загальний рівень точності на тестових даних у переважній більшості випадків вищий, ніж на валідаційних.

Для кращої моделі були побудовані діаграми типу «Confusion matrix»:



Оскільки попереднє рішення має ряд недоліків, ми створили ще одне, з більшою кількістю шарів Keras, а також з можливістю аналізувати LearningRate:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.metrics import confusion_matrix
import random
from sklearn.utils import shuffle
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '1'
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
import tensorflow as tf
import tensorflow_datasets as tfds
from sklearn.preprocessing import OneHotEncoder
from datetime import datetime
from datetime import timedelta
def preprocess(dataset):
    image = tf.cast(dataset['image'], dtype=tf.float32) / 255.
    label = tf.cast(dataset['label'], dtype=tf.float32)
    return image, label
```

```
def SimpleConvolutionalNeuralNetworkModel(NumberOfClasses):
    input_ = tf.keras.layers.Input(shape=(28, 28, 1,))
    x = tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu')(input_)
    x = tf.keras.layers.MaxPool2D(2, 2)(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu')(x)
    x = tf.keras.layers.MaxPool2D(2, 2)(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Conv2D(16, (3, 3), padding='same', activation='relu')(x)
    x = tf.keras.layers.MaxPool2D(2, 2)(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(128, activation='relu')(x)
    output_ = tf.keras.layers.Dense(NumberOfClasses, activation='softmax')(x)
    return tf.keras.models.Model(input_, output_, name='Classifier')
def RunLogisticRegression(CountOfEpochs,BatchSize,LearningRate,TrainData,ValidationData,
TestData,MetaData):
    TrainData = TrainData.map(preprocess).shuffle(buffer_size=1024).batch(BatchSize)
    ValidationData = ValidationData.map(preprocess).batch(BatchSize)
    TestData = TestData.map(preprocess).batch(BatchSize)
    NumberOfClasses = MetaData.features['label'].num_classes
    model = SimpleConvolutionalNeuralNetworkModel(NumberOfClasses)
    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(LearningRate,decay_steps=
    100000,decay_rate=0.96)
    model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule),loss = tf.
    keras.losses.SparseCategoricalCrossentropy(),metrics = ['accuracy'])
    history = model.fit(TrainData, epochs=CountOfEpochs, validation_data=ValidationData,
    verbose=0)
```

```
    if(CountOfEpochs>1):
        fig,ax=plt.subplots(ncols=2, figsize=(20, 6))
        ax[0].set_title('Accuracy and validation accuracy')
        ax[0].set_xlabel('Epoch')
        ax[0].set_ylabel('Accuracy')
        ax[0].plot([i+1 for i in range(CountOfEpochs)],history.history['accuracy'], label =
        'Train accuracy')
        ax[0].plot([i+1 for i in range(CountOfEpochs)],history.history['val_accuracy'], label
         = 'Validation accuracy')
        ax[0].legend(bbox_to_anchor = (0.0005,1.13),loc = 'upper right', title="Accuracy",
        labels=['Accuracy','Valid. accuracy'],fontsize=14)
        ax[1].set_title('Loss and validation loss')
        ax[1].set_xlabel('Epoch')
        ax[1].set_ylabel('Accuracy')
        ax[1].plot([i+1 for i in range(CountOfEpochs)],history.history['loss'], label =
        'Train loss')
        ax[1].plot([i+1 for i in range(CountOfEpochs)],history.history['val_loss'], label =
        'Validation loss')
        ax[1].legend(bbox_to_anchor = (1.263,1.13),loc = 'upper right', title="Loss",labels=[
        'Loss','Valid. loss'],fontsize=14)
        plt.savefig("Plots\\CountOfEpochs "+str(CountOfEpochs)+" BatchSize "+str(BatchSize)+
        " LearningRate "+f'{LearningRate:.5f}'+".jpg",dpi=200)
        plt.close()
    return model
```

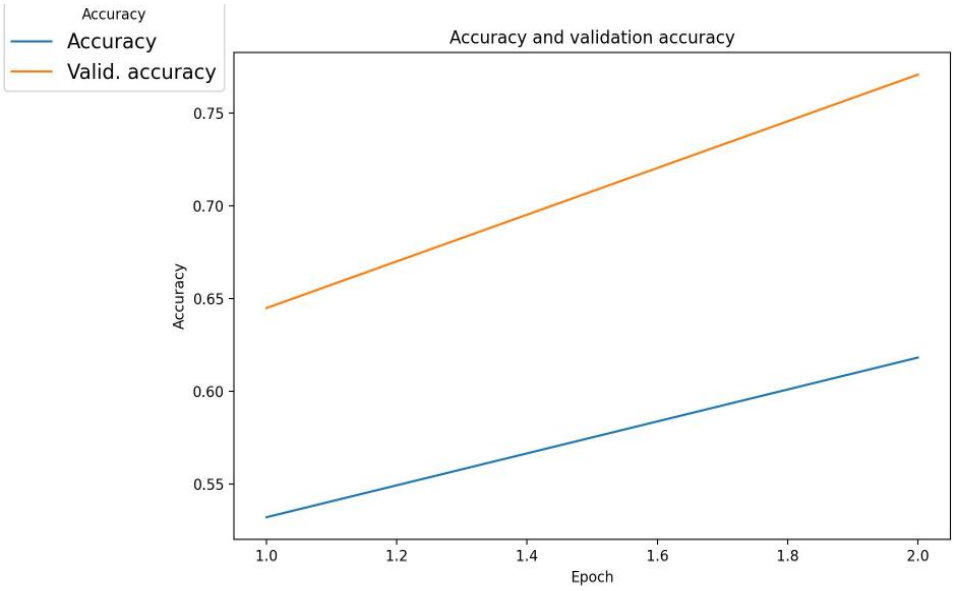
Запустивши код із стандартними параметрами, ми отримали наступний вивід:

```
Count of epochs:  10 ; batch size:  32 ; learning rate:  0.00100
Processing... Time:  0 : 7 : 37 ; accuracy:  0.98456
```

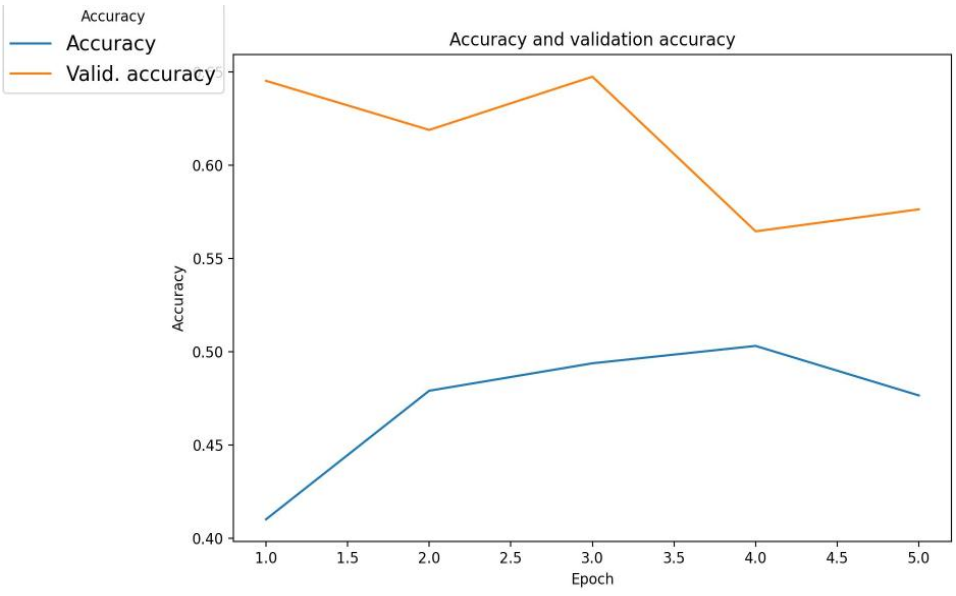
Бачимо, що точність даної моделі більша за попередню лише на майже 1%, проте обчислення тривали в рази довше з тими самими параметрами.

Нами було проведено багато експериментів з усіма комбінаціями кількості епох (від 1 до 5), розмірів частини вибірки (BatchSize) та коефіцієнтів швидкості навчання (LearningRate), для кожної з них було побудовано графіки такого вигляду:

CountOfEpochs 2 BatchSize 32 LearningRate 0.02000

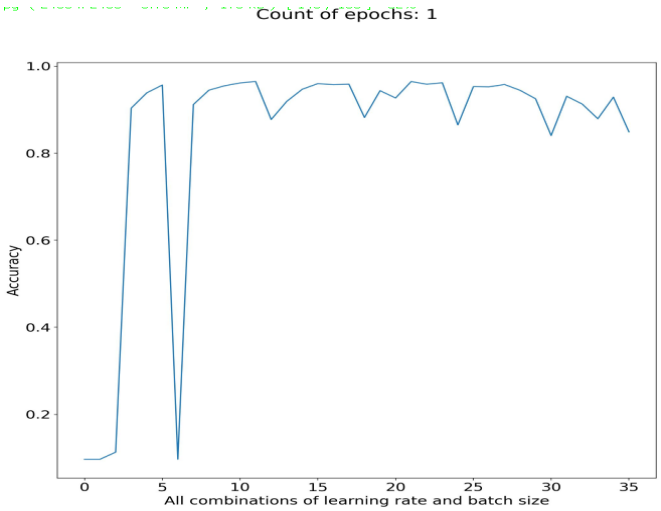


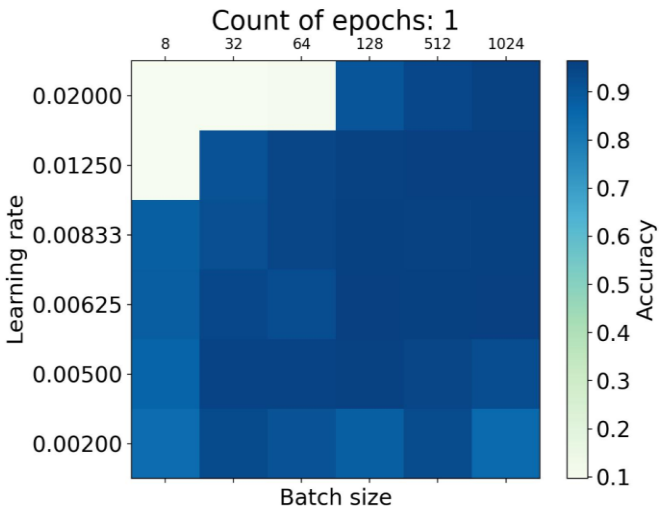
CountOfEpochs 5 BatchSize 32 LearningRate 0.02000

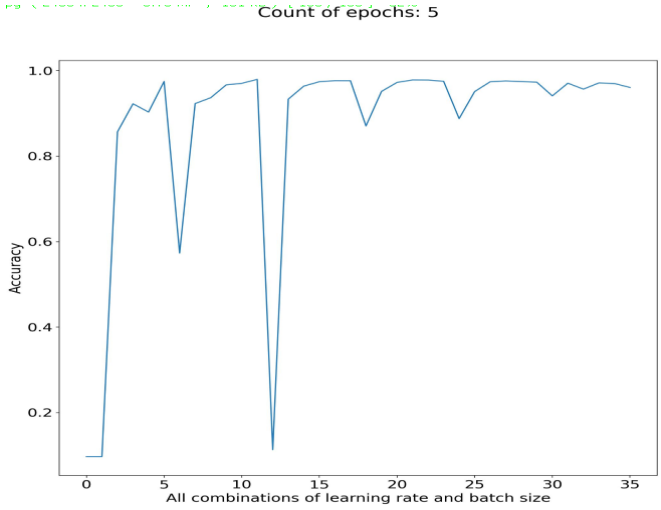


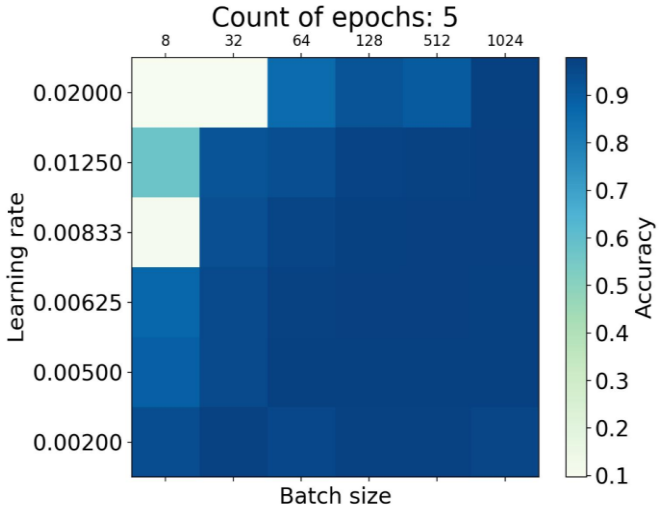
Видно, що зі збільшенням кількості навчальних епох загальний рівень точності роботи моделі на валідаційних та тестових даних збільшується, а у переважній більшості випадків валідаційна точність вища за тестову.

Для кожної епохи також отримали більш загальні графіки та діаграми:









Підсумковий графік та кольорова діаграма виглядають таким чином: