МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студентка гр. 7381 Процветкина А.В.

Преподаватель Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).

Постановка задачи.

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Написать функцию, позволяющая загружать изображение пользователи и классифицировать его

Ход работы.

Посмотрим на формат данных перед обучением модели. С помощью shape можно увидеть, что в тренировочном датасете 60,000 изображений, каждое размером 28 x 28 пикселей. Изображения черно-белые и по сути каждый элемент тензора есть число от 0 до 255, где 0 -- черный, а 255 -- белый промежуточное значение соответственно оттенок серого.

Первый слой этой сети - tf.keras.layers.Flatten, преобразует формат изображения из двумерного массива (28 на 28 пикселей) в одномерный (размерностью 28 * 28 = 784 пикселя). Слой извлекает строки пикселей из изображения и выстраивает их в один ряд. Этот слой не имеет параметров для обучения; он только переформатирует данные.

Далее стандартные полносвязные слои Dense и обучение модели. Начальные параметры оказались весьма оптимальны – получаемая точность составила 97.6%. На рис.1 приведены потери и точность обучения модели.

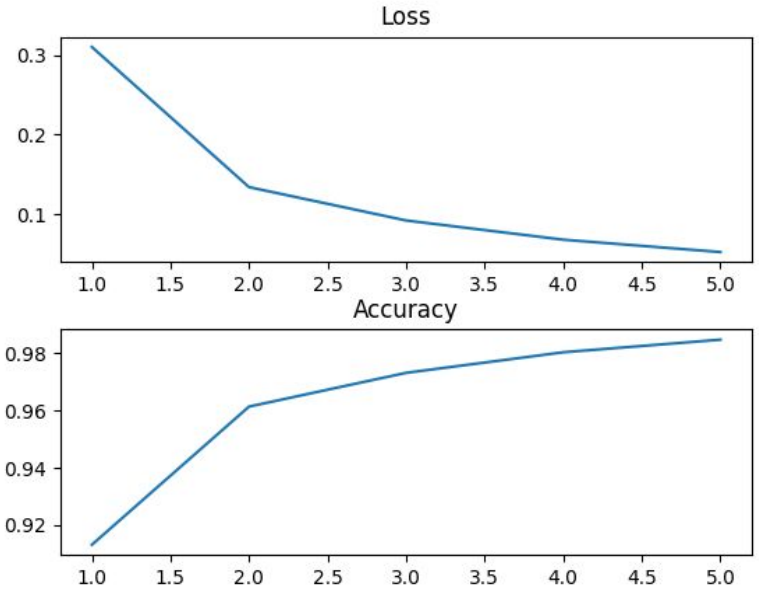

Рисунок 1 -- Потери и точность обучения модели

Исследование оптимизаторов и их параметров на процесс обучения:

Adam является самым популярным и эффективным оптимизатором, поэтому он будет изучен наиболее подробно. Так же исследуем оптимизаторы AdaGrad, RMSProp и SGD. (Большее число оптимизаторов было бы неудобно рассматривать на одном графике)

Оптимизаторы обладают параметрами, которые для универсальности принимают значения по умолчанию (оптимальные в том или ином смысле значения). Вполне естественно будет сравнить результаты обучения одной и той же модели с разными оптимизаторами. Результат такого сравнения приведен на рис.2 и в таблице 1. (Зеленая и синяя линии практически совпадают, так что на рисунке их не отличить, но в легенде указаны оба оптимизатора)

Таблица 1 -- Результаты обучения модели

	Adam	Adagrad	RMSProp	SGD
test_accuracy	0.9782	0.9232	0.979	0.9103

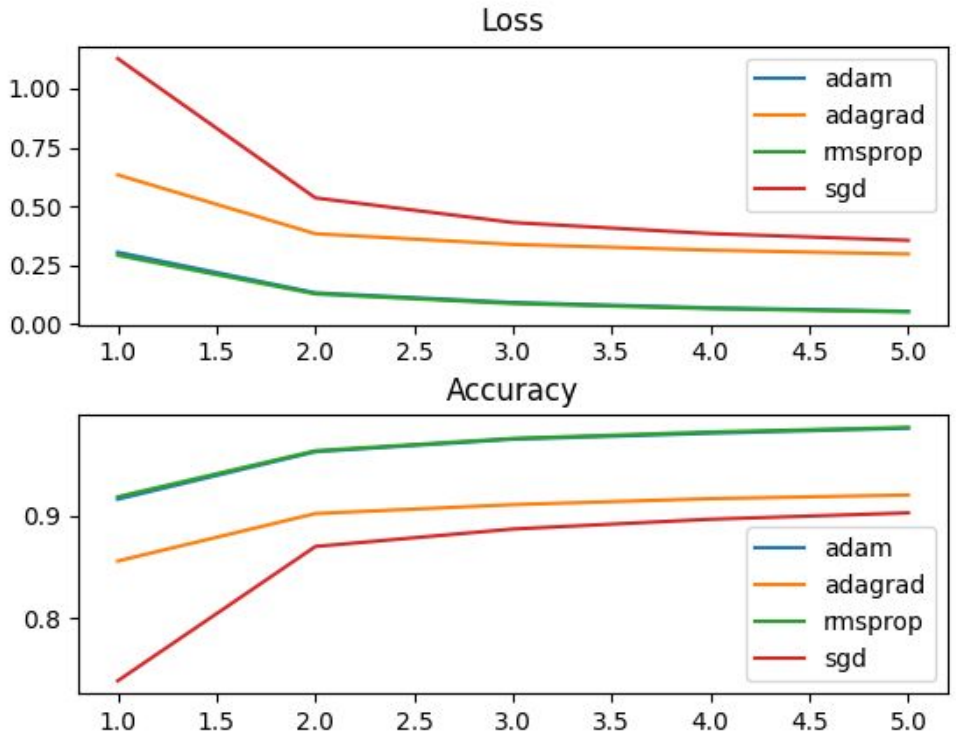

Рисунок 2 -- Сравнение оптимизаторов с параметрами по умолчанию.

Отметим, что Adam и RMSProp показали примерно одинаковый процесс обучения и схожие результаты теста. В то время как Adagrad и SGD справились заметно хуже.

SGD (Stochastic Gradient Descent)

Данный оптимизатор является наиболее простым.

```
keras.optimizers.SGD(learning_rate=0.01, momentum=0.0, nesterov=False)
```

Ограничимся 4-мя вариантами.

1) По умолчанию
2) `learning_rate=0.1`
3) `momentum=0.9`
4) `learning_rate=0.1, momentum=0.9`

Результаты обучения показаны на рис. 3.

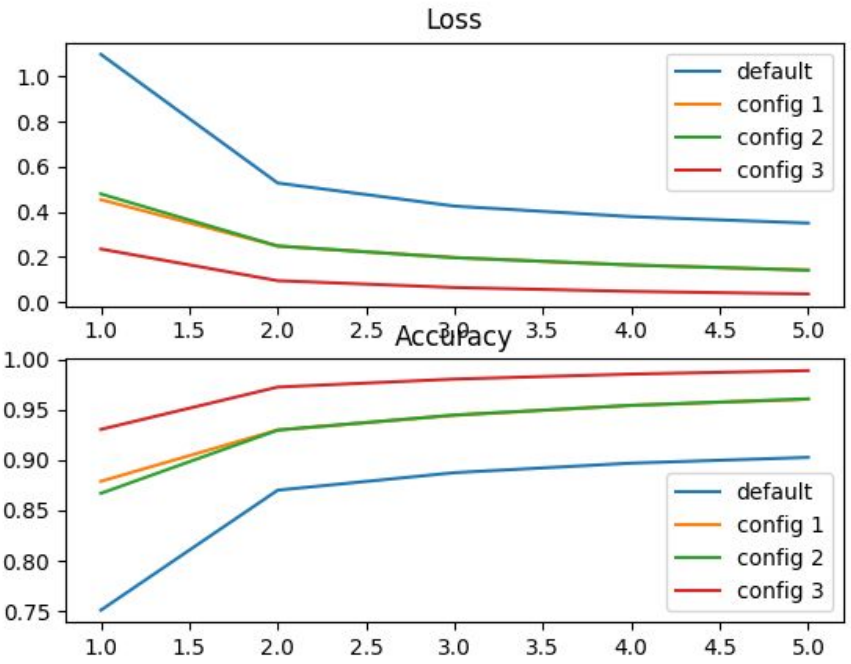

Рисунок 3 -- Исследование оптимизатора SGD

Изменение момента и скорости обучения по отдельности улучшает процесс обучения. Одновременное изменение данных параметров ускоряет минимизацию еще лучше.

Adagrad (Adaptive gradient)

```
keras.optimizers.Adagrad(learning_rate=0.01)
```

У оптимизатора всего один параметр, так что трех конфигураций будет вполне достаточно:

1. По умолчанию
2. `learning_rate=0.001`
3. `learning_rate=0.1`

Результаты обучения приведены на рис. 4.

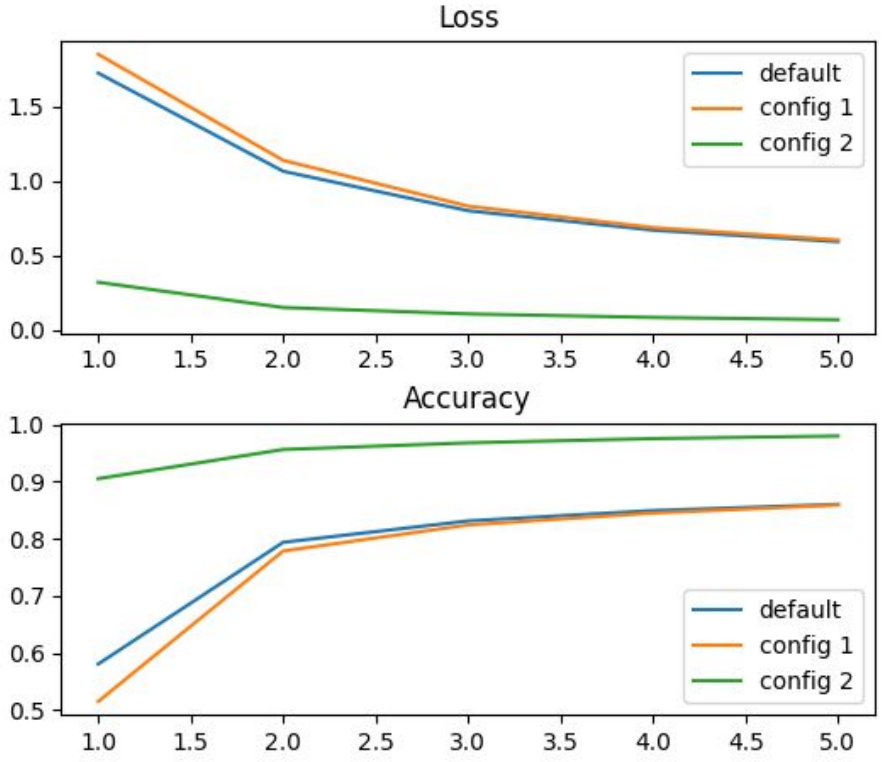

Рисунок 4 -- Исследование оптимизатора Adagrad

В данном случае конфигурация по умолчанию оказалась не оптимальна, и увеличение параметра скорости обучения ускорило процесс минимизации.

RMSProp (Root Mean Square Propagation)

```
keras.optimizers.RMSprop(learning_rate=0.001, rho=0.9)
```

Исследуем следующие конфигурации:

1) По умолчанию
2) `learning_rate=0.01`
3) `rho=0.1`
4) `learning_rate=0.01, rho=0.1`

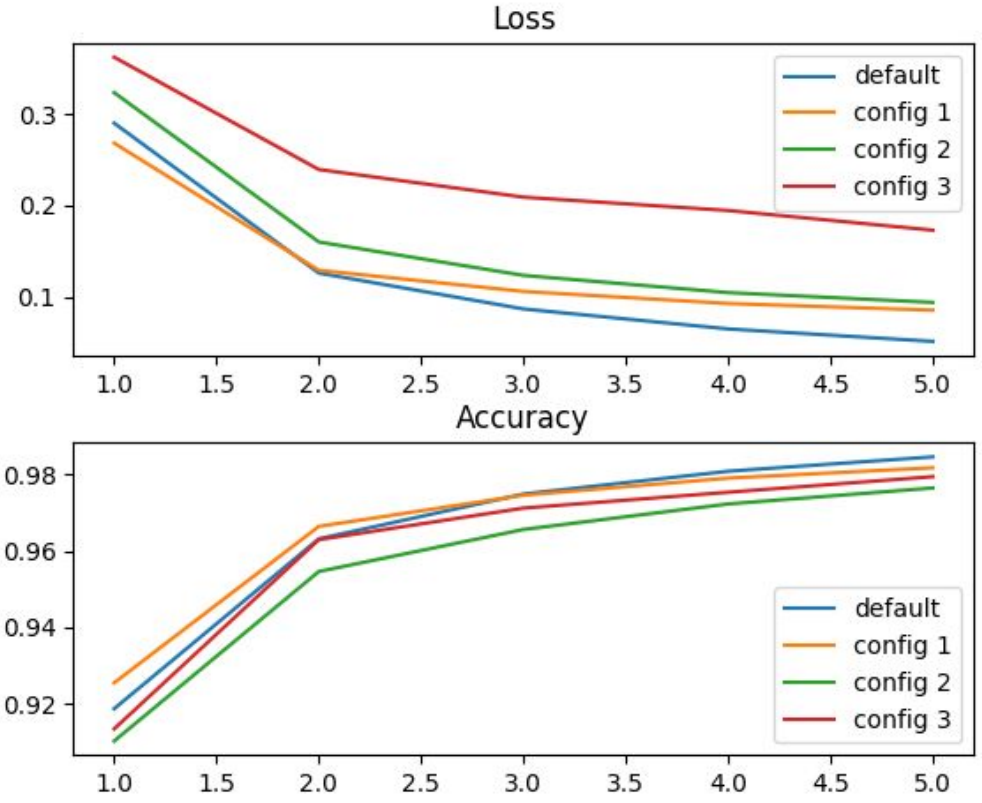

Рисунок 5 -- Исследование оптимизатора RMSProp

Конфигурация по умолчанию оказалась самой удачной.

Adam(Adaptive moment)

```
keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,
amsgrad=False)
```

Исследуем следующие конфигурации:

1) По умолчанию
2) `amsgrad=True`
3) `learning_rate=0.01`
4) `beta_1=0.1, beta_2=0.1`

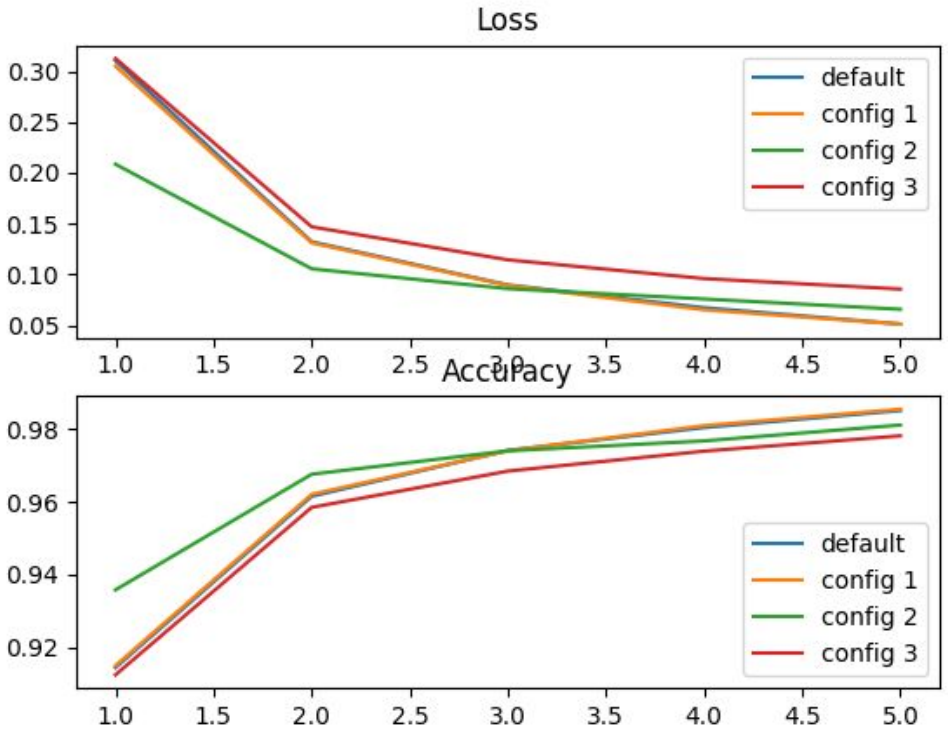

Рисунок 6 -- Исследование оптимизатора Adam

Конфигурация по умолчанию данного оптимизатора оказалась весьма удачна (синяя линия совпадает с оранжевой).

Сравним самые удачные варианты конфигураций всех четырех оптимизаторов:

1) `SGD(learning_rate=0.1, momentum=0.9)`
2) `Adagrad(learning_rate=0.1)`
3) `RMSprop()`
4) `Adam()`.

Результаты приведены на рис. 7 и в таблице 2.

Таблица 2 -- Сравнение наиболее удачных конфигураций

	SGD	Adagrad	RMSProp	Adam
test_accuracy	0.9813	0.9747	0.9785	0.9768

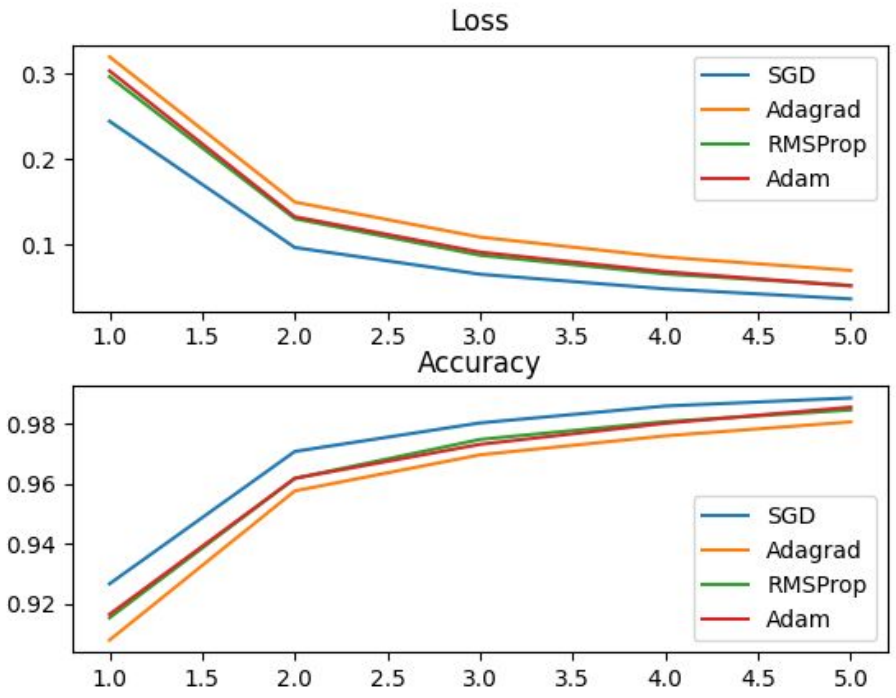

Рисунок 7 -- Сравнение наиболее удачных конфигураций

Неожиданно, но стохастический градиент показал лучшие результаты, нежели популярные Adam и RMSprop.

Также была написана функция, позволяющая загружать собственное изображение и предсказывающая написанную цифру. С помощью библиотеки Pillow изображение считывается в numpy array и сразу же конвертируется двумерный тензор, поскольку модель обучалась на монохромных изображениях.

С предсказанием модель справляется, как показано на рис. 8 (хотя и не всегда).

```
prediction for file 0.jpeg -- 0
prediction for file 1.jpeg -- 1
```

Рисунок 8 -- Предсказания модели для пользовательских изображений.

Выводы.

В ходе данной лабораторной работы было исследовано представление и передача в модель простейших черно-белых изображений из специального датасета MNIST. Была написана функция для загрузки пользовательского изображения в пригодном для последующей передачи в модель виде.

Также был исследован процесс обучения модели с различными конфигурациями четырех оптимизаторов: Adam, Adagrad, RMSprop и SGD.

Не станет открытием тот факт, что в различных ситуациях применимы различные методы оптимизации. Единого и всегда подходящего варианта не существует, однако есть довольно универсальные алгоритмы, удовлетворяющие требованиям в большинстве случаев. В рамках данной задачи лучше всего себя проявил метод SGD с параметрами `learning_rate=0.1, momentum=0.9`. Однако это вовсе не означает, что остальные методы хуже -- Adam более гибкий и универсальный нежели SGD, а увеличение параметра скорости обучения просто далеко не всегда применимо.

Приложение А
Исходный код программы

```
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Flatten
    from keras.utils import to_categorical
    from tensorflow import optimizers
    from matplotlib import pyplot

    from PIL import Image
    from numpy import asarray
    import tensorflow as tf
    import numpy as np


    mnist = tf.keras.datasets.mnist
    (train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

    train_images = train_images / 255.0
    test_images = test_images / 255.0

    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)

    def looping_optimizers(opt_list, labels):
        acc_list = []
        history_loss_list = []
        history_acc_list = []

        # create model
        for opt in opt_list:
            model = Sequential()
            model.add(Flatten(input_shape=(28, 28)))
            model.add(Dense(256, activation='relu'))
            model.add(Dense(10, activation='softmax'))
            model.compile(optimizer=opt,
loss='categorical_crossentropy', metrics=['accuracy'])

            # train the model
            res = model.fit(train_images, train_labels, epochs=5,
batch_size=128)
            history_loss_list.append(res.history['loss'])
            history_acc_list.append(res.history['accuracy'])
            test_loss, test_acc = model.evaluate(test_images,
test_labels)
            acc_list.append(test_acc)
```

```
        print('test_acc:', acc_list)
        x = range(1, 6)

        pyplot.subplot(211)
        pyplot.title('Loss')
        for loss in history_loss_list:
            pyplot.plot(x, loss)
        pyplot.legend(labels)

        pyplot.subplot(212)
        pyplot.title('Accuracy')
        for acc in history_acc_list:
            pyplot.plot(x, acc)
        pyplot.legend(labels)
        pyplot.show()
        return model


    test = 6
    opt_list = []

    # comparing default
    if test == 0:
        opt_list = ("adam", "adagrad", "rmsprop", "sgd")
        looping_optimizers(opt_list, opt_list)

    # tuning sgd
    if test == 1:
        opt_list.append(optimizers.SGD())
# default
        opt_list.append(optimizers.SGD(learning_rate=0.1))
# conf 1
        opt_list.append(optimizers.SGD(momentum=0.9))
# conf 2
        opt_list.append(optimizers.SGD(learning_rate=0.1, momentum=0.9))
# conf 3
        looping_optimizers(opt_list, ("default", "config 1", "config 2",
"config 3"))

    # tuning adagrad
    if test == 2:
        opt_list.append(optimizers.Adagrad())                    #
default (0.01)
        opt_list.append(optimizers.Adagrad(learning_rate=0.001))   #
conf 1
        opt_list.append(optimizers.Adagrad(learning_rate=0.1))     #
conf 2
        looping_optimizers(opt_list, ("default", "config 1", "config
2"))
```

```
    # tuning rmsprop
    if test == 3:
        opt_list.append(optimizers.RMSprop())
# default learning_rate=0.001, rho=0.9
        opt_list.append(optimizers.RMSprop(learning_rate=0.01))
# conf 1
        opt_list.append(optimizers.RMSprop(rho=0.1))
# conf 2
        opt_list.append(optimizers.RMSprop(learning_rate=0.01, rho=0.1))
# conf 3
        looping_optimizers(opt_list, ("default", "config 1", "config 2",
"config 3"))

    # tuning adam
    if test == 4:
        opt_list.append(optimizers.Adam())                          #
default (lr=0.001, beta_1=0.9, beta_2=0.999)
        opt_list.append(optimizers.Adam(amsgrad=True))              #
conf 1
        opt_list.append(optimizers.Adam(learning_rate=0.01))        #
conf 2
        opt_list.append(optimizers.Adam(beta_1=0.1, beta_2=0.1))    #
conf 3
        looping_optimizers(opt_list, ("default", "config 1", "config 2",
"config 3"))

    # comparing best
    if test == 5:
        opt_list.append(optimizers.SGD(learning_rate=0.1, momentum=0.9))
        opt_list.append(optimizers.Adagrad(learning_rate=0.1))
        opt_list.append(optimizers.RMSprop())
        opt_list.append(optimizers.Adam())
        looping_optimizers(opt_list, ("SGD", "Adagrad", "RMSProp",
"Adam"))


    def read_file(path):
        # load the image
        image = Image.open(path).convert('L')
        # convert image to numpy array and without rgb stuff
        data = asarray(image)
        data = data.reshape((1, 28, 28))
        return data


    model = looping_optimizers([optimizers.SGD(learning_rate=0.1,
momentum=0.9)], ["SGD"])
    filename = "0.jpeg"
    img = read_file(filename)
```

```
    Y = model.predict_classes(img)
    print("prediction for file " + filename + " -- " +
np.array2string(Y[0]))
    img = read_file("1.jpeg")
    filename = "1.jpeg"
    Y = model.predict_classes(img)
    print("prediction for file " + filename + " -- " +
np.array2string(Y[0]))
    model.save("model.h5")
```