Documentatie Proiect

Gheorghe Cosmina

grupa 231

Pentru implementarea acestui proiect am ales sa testez modelele de clasificare CNN (Convolutional Neural Network) si K-NN (K-Nearest Neighbors). Am optat pentru aceste doua modele din urmatoarele motive:

- K-NN a fost abordat in cadrul primelor laboratoare si am considerat ca este un punct de start potrivit. In acest clasificator, se calculeaza distanta imaginii de test curente fata de toate celelalte imagini din set si se selecteaza primii K cei mai apropiati vecini. Clasa majoritara este atribuita ulterior imaginii test
- CNN este o retea de invatare profunda proiectata pentru procesarea structurilor de date in forma de matrice, cum ar fi imaginile, motiv pentru care l-am utilizat in proiect. Acesta transforma imaginile trecandu-le printr-un set de straturi. Acestea pot fi alese in multiple feluri, rezultatele fiecarei configurari variind in functie de setul de date.

La inceputul ambelor implementari am realizat importurile necesare (am adaugat si pe parcurs) si am stocat imaginile si etichetele din setul de date. Mai intai am citit datele de antrenare, validare si testare din csv utilizand functia read_csv din biblioteca pandas. Pe urma am stocat in liste numele imaginilor de antrenare, testare si validare si etichetele de antrenare si validare corespunzatoare.

Am concatenat fiecare nume de imagine din setul de antrenare cu path-ul catre folderul de imagini de antrenare si am stocat aceste cai intr-o lista. Pentru a incarca si redimensiona datele apelez o functie specifica, trimitand ca parametrii caile catre imagini si dimensiunile dorite. La inceput am utilizat dimensiunea 28x28 pixeli atat pentru K-NN cat si pentru CNN, testand mai tarziu si cu 64x64 pixeli si observand ca in cazul CNN-ului acuratetea era mult mai buna (voi detalia mai jos), in timp ce pentru KNN aceasta a scazut de la 0.137 la 0.109.

In functia mentionata deschid fiecare imagine, o incarc si o redimensionez adaugand-o in lista in care stochez imaginile redimensionate. Pentru acest pas folosesc functiile Image.open() si resize(). Repet acest proces si pentru imaginile de validare si testare pentru a ma asigura ca toate au aceeasi forma si pentru a le putea prelucra ulterior fara impedimente.

Din acest punct cele doua implementari se diferentiaza din caza formei diferite a datelor de care fiecare clasificator are nevoie.

1. *K-NN*

Deoarece pentru acest model de invatare este necesara o reprezentare unidimensionala a vectorilor trebuie sa aduc imaginile de antrenare la acest format folosind functia flatten() din biblioteca NumPy. Inainte de a apela functia transform fiecare imagine in numpy array pentru a asigura compatibilitatea si eficienta in utilizarea functiei. Fac acelasi lucru si penutru noua lista de imagini aplatizate pentru viitoare operatii. Repet acest proces si pentru imaginile de validare si testare.

Urmeaza pasul de normalizare a datelor in care ma asigur ca valorile pixelilor se afla in intervalul [0,1]. Folosesc astype('float32') deoarece vreau ca pixelii sa fie reprezentati in floating point si impart la 255.0 deoarece valorile acestora se afla in range-ul 0-255 in mod normal.

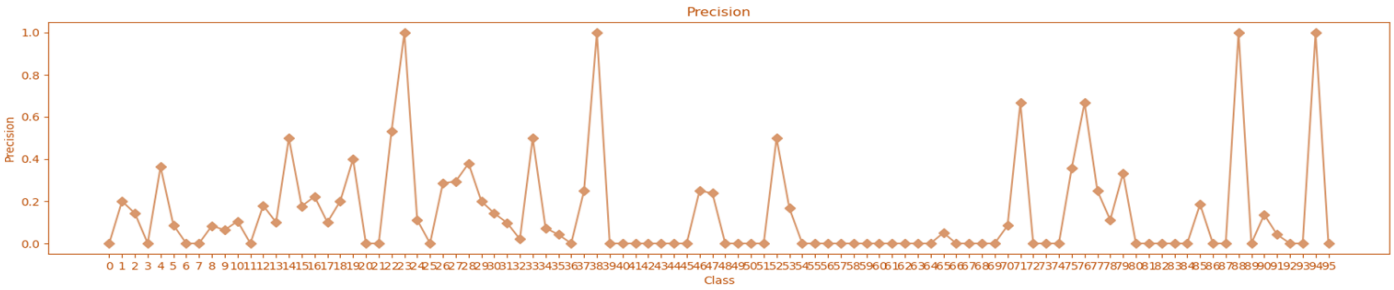
In continuare initializez modelul. Initial am ales valoare 5 pentru k, aceasta fiind chiar valoarea implicita din scikit, cu acuratetea 0.137. Acuratetea in acest caz a fost mai mare decat cea pentru k=20 (0.109), k=50 (0.106) sau k=1 (0.127), dar mai mica decat pentru k=3 (0.138) .

Antrenez modelul folosind functia fit() pe imaginile normalizate si etichete si prezic etichetele imaginilor de validare cu functia predict(). Calculez acuratetea (Exemple clasificate corect) / (Număr total de exemple) , precizia (Exemple pozitive clasificate corect) / (Număr total de exemple clasificate pozitiv) si recuperarea (Exemple pozitive clasificate corect) / (Număr total de exemple pozitive) in urma testului facut pe datele de validare si le afisez folosind matplotlib.pyplot. Pe urma declar si matricea de confuzie si o plotez, marind fontul pentru o imagine mai clara.

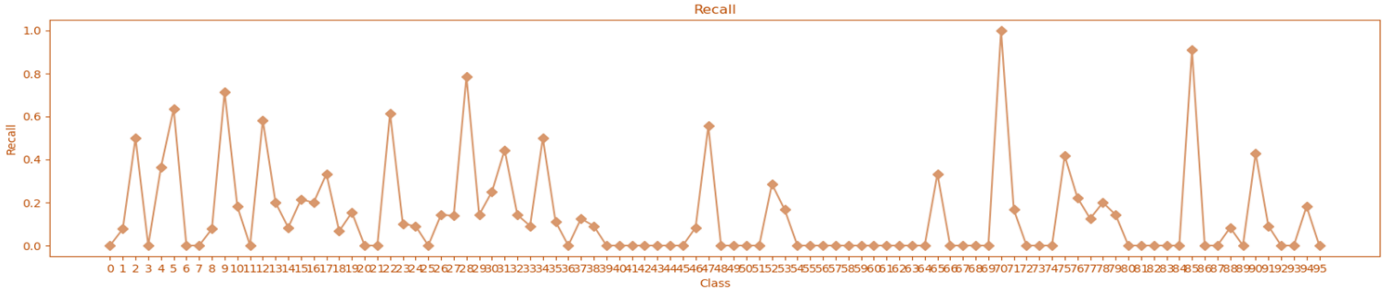
Acestea au fost rezultatele pentru dimensiunea pozelor 28x28 si k=3:

Acuratete: 0.138

Precizie:



Recuperare:

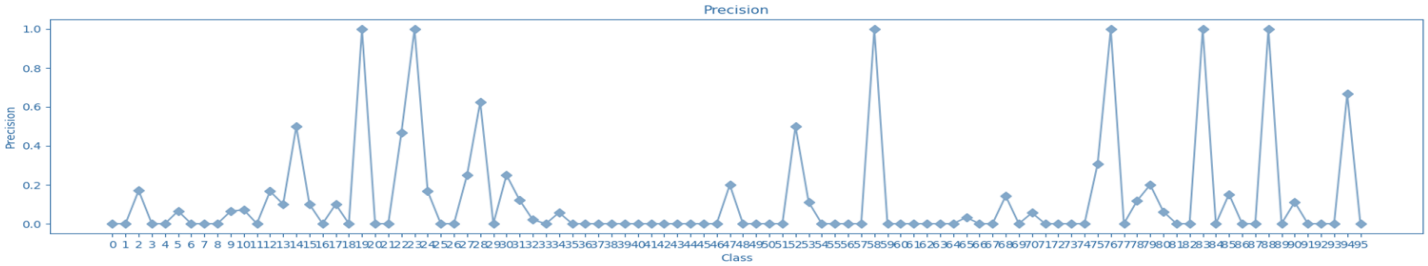


Matricea de confuzie:

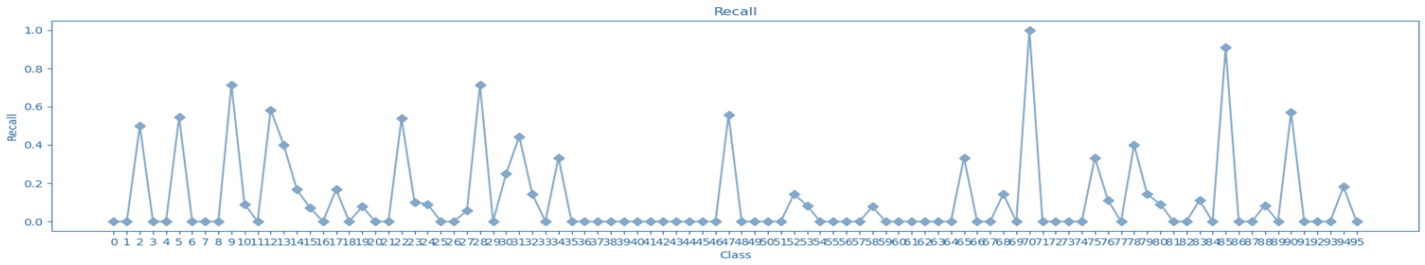
Acestea au fost rezultatele pentru dimensiunea pozelor 64x64 si k=5:

Acuratete: 0.109

Precizie:



Recuperare:



2. CNN

In cazul acestei abordari nu mai e nevoie sa reprezint datele drept un vector unidimensional asa ca dupa citire, incarcare si redimensionare transform listele de imagini in numpy arrays prin acelasi proces ca si KNN, omitand utilizarea functiei de aplatizare. Repet si pasul de normalizare a datelor, valoare pixelilor fiind acum un numar in floating point intre 0 si 1.

In continuare, vreau sa transorm etichetele claselor intr-un format potrivit invatarii neuronale asa ca aplic functia to_categorical pentru a le transforma in one hot vectors, adica fiecare eticheta este schimbata intr-un vector care va avea lungimea egala cu numarul de clase. Respectivul vector contine valori de 0 pe toate pozitiile in afara de aceea care corespunde clasei adevarate a etichetei, aceasta fiind setata la 1.

Urmeaza implementarea modelului CNN, pentru inceput il instantiez si selectez o valoare pe care o voi folosi in startul de dropout. In urma a multiple teste am observat ca valoarea 0.4 se potriveste cel mai bine pentru implementarile mele. Am incercat si valori precum 0.1, 0.5, 0.8. Voi enumera straturile folosite in teste, pe care le-am adaugat si sters pe parcurs:

Stratul de convolutie- Conv2D, stratul de baza care extrage caracteristicile imaginilor de intrare. Acest pas consta in aplicarea unui filtru asupra imaginii pentru a calcula produse punctuale si suma acestora, obtinandu-se o feature map. In primul strat de acest tip setez si forma inputului: (64, 64, 3), (28, 28, 3), 3 reprezentand numarul de canale de culoare, iar aceasta se pastreaza pe parcurs. Alti parametrii sunt numarul de filtre, adica cate harti sunt generate, pe care il setez initial la 32 si il cresc pe parcurs, fiecare filtru extragand o anumita caracteristica a imaginii. Kernel size- marimea ferestrei de convolutie care se deplaseaza pe imagine, Stride- numarul de pixeli cu care se deplaseaza fereastra, Activation- se selecteaza functia de activare, “relu” in cazul meu. Padding- “same” asigura ca dimensiunea hartii de iesire e aceeasi cu cea a imaginii de intrare, folosit in combinatie cu stride.

Stratul de Pooling-reduce dimensiunea hartii. In primele rezolvari am incercat sa foloses max pooling in care se selecteaza cel mai mare element dintr-o regiune, dar ulterior l-am scos din model observand ca nu aduce rezultatele dorite.

Stratul de normalizare (BatchNormalization)- accelereaza antrenarea retelei prin realizarea normalizarii pe stratul precedent calculand media si deviatia standard.

Stratul de Dropout- ajuta la evitarea overfittingului prin dezactivarea la intamplare a unor neuroni, facand reteaua mai generala.

Stratul de aplatizare (Flatten) – pregateste harta de carateristici pentru straturile complet conectate, transformanu-o intr-un vector unidimensional.

Stratul complet conectat(Dense)- realizeaza calsificarea finala a caracteristicilor. In acesta se pot adauga functii de activare dar si de regularizare care controleaza overfittingul. La final utilizez functia “softmax” cu scopul de a genera probabilitati de clasificare pentru fiecare clasa.

Setez o valoare initiala pentru rata de invatare pe care o folosesc drept parametru la compilarea modelului CNN unde utilizez metrica de evaluare “accuracy” si functia de loss “categorical_crossentropy” specifica problemelor de clasificare multipla (mai mult de doua clase). Ca urmator pas am implementat o functie care o data la n epoci scade rata de invatare cu 10%, utilizata in antrenarea modelului. Aici setez si numarul de epoci si batch size-ul, valori pentru care am realizat multe incercari.

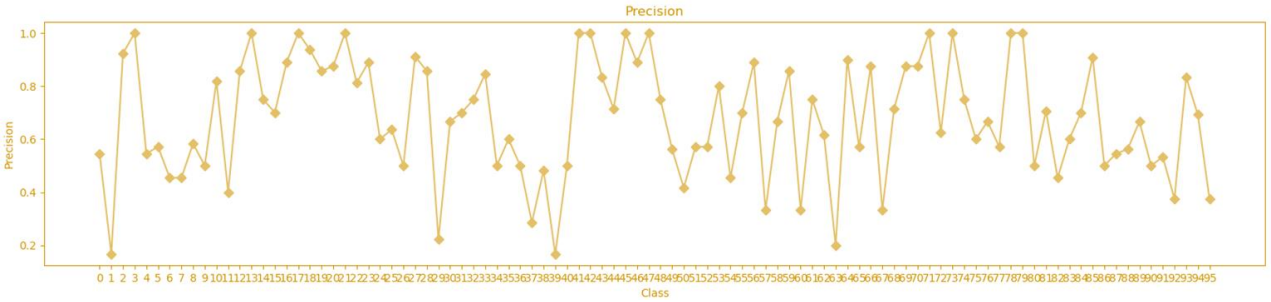
In final calculez acuratetea, precizia, recuperarea si matricea de confuzie ca si in cazul KNN-ului, fac predictia pentru imaginile de testare si salvez numele imaginilor si etichetele intr-un fisier csv.

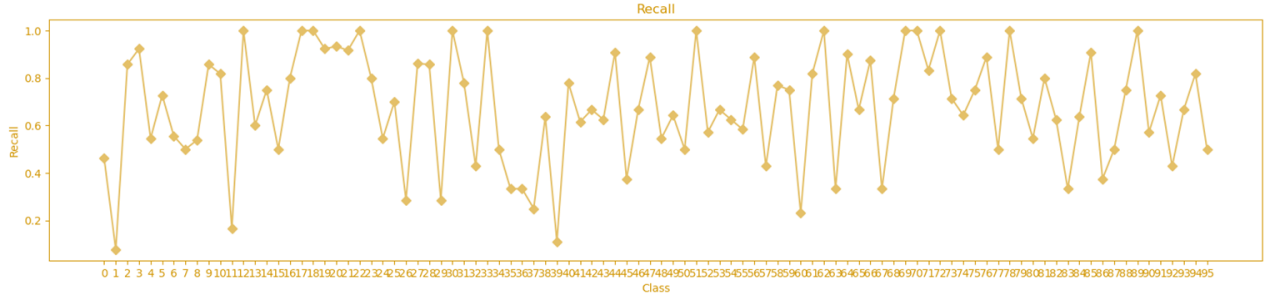
Voi descrie abordarile pentru implementarea modelului CNN:

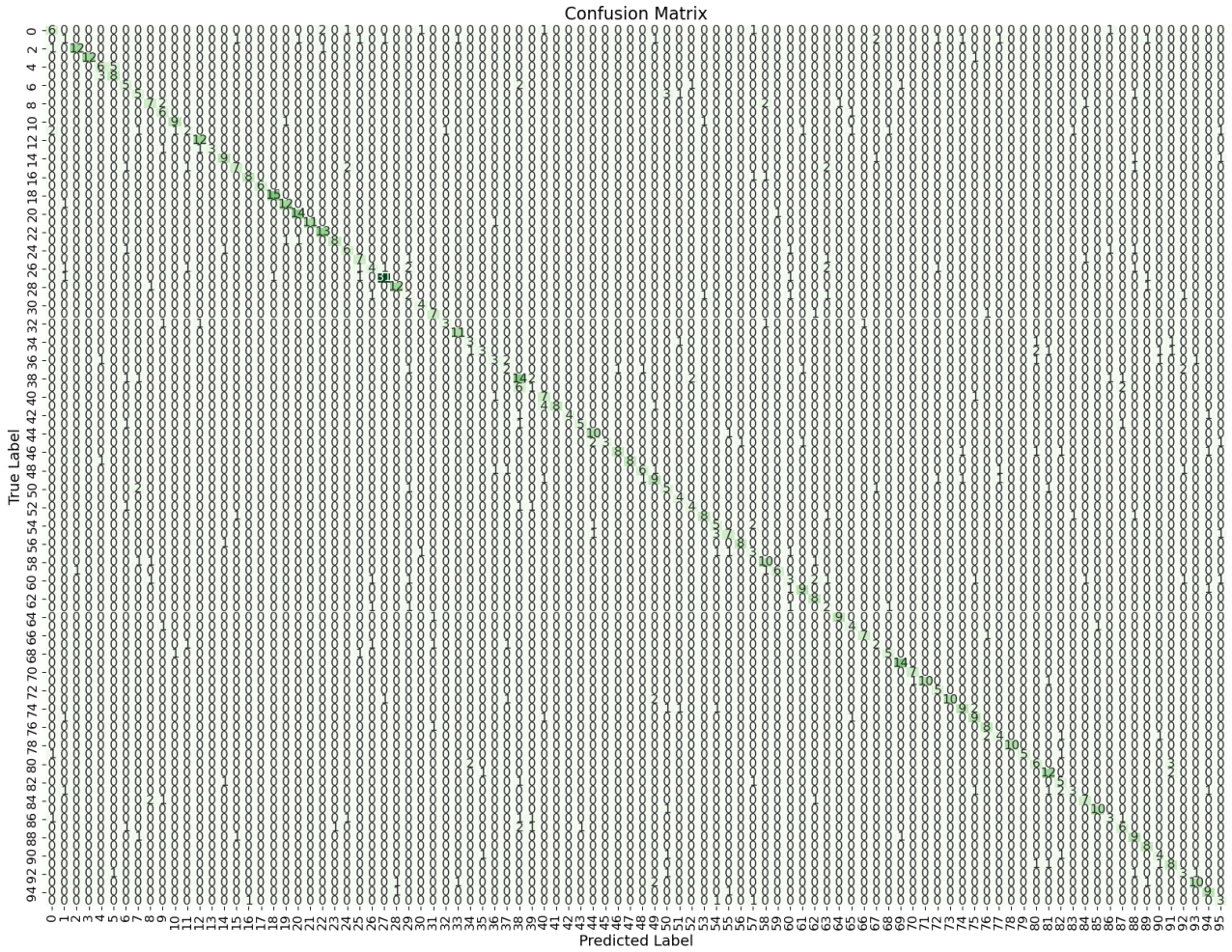
1. Am ales forma detlor de 28x28 si am folosit doua straturi de convolutiecu 32 de filtre, fiecare urmat de unMaxPooling. Dupa acest set am adaugat un dropout cu parametrul 0.1 si inca un strat de convolutie cu 64 de filtre cu un strat de pooling. Am adaugat si un strat de aplatizare, si un strat final Dense. Pentru acest format au functionat cel mai bine 30 de epoci si batch_size-ul 32. Pentru fiecare configurare am testat combinari intre valorile (20,30,40) pentru epoci si (16,32,64) pentru batch size. Au fost si cazuri in care, testand initial cu 30 de epoci, am observat din evolutia valorilor de loss ca sunt necesare mai multe epoci si nu am mai relizat teste cu mai putine epoci.
   In acest caz am obtinut acuratetea 0.54, urmand sa obtin 0.56 dupa adaugarea unui nou strat Dense si a unui Dropout. Ulterior am schimbat forma datelor, alegand 64x64 si am obtinut o acuratete de 0.68 cu formatul precizat.
2. De acesta data am decis sa inlocuiesc max pooling cu batch normalization si rezultatul a fost unul pozitiv. Am adaugat 4 straturi de convolutie cu 32 de filtre fiecare urmat de batch normalization, dupa acestea un dropout si inca 2 straturi de convolutie cu 64 de filtre cu batch normalization si dropout. Pentru doua din straturile de convolutie folosesc si stride. Am testat si fara acesta dar acuratetea este putin mai mica in acel caz. In final am iarasi doua Dense -uri, primul facand si regularizarea datelor cu L2. Am ales L2 pentru ca a dat rezultate mai bune decat L1 cu aproximativ 0.06.
   Pentru aceasta structura de straturi am obtinut acuratetile 0.72 pe tip de date 28x28 si dropout 0.1 si 0.833 pe date 64x64 si dropout 0.4, cu un numar de epoci de 30 si batch size de 32.
3. Am mai incercat si alte configurari, adaugand sau scazand numarul de straturi de convulatie, un alt rezultat remarcabil la care am ajuns fiind: 3 straturi de convolutie cu 32 de filtre, unul cu strides=2 urmate de batch normalization si de un dropout, 2 straturi de convulatie cu 64 de filtre, cu batch normalization aferent si dropout in final si cele doua Dense uri ulterioare, cu dropout 0.4, date 64x64, 40 de epoci si 32 batch size. In acest caz am obtinut acurateti in jurul valorii de 0.83-0.84.
4. Pentru structura finala am adaugat la (3) inca un strat de convolutie cu 32 de filtre si un batch normalization pe 30 de epoci in loc de 40 si am obtinut acuratetea 0.85333.

Voi adauga si grafice de acuratete, precizie recuperare si matricea de confuzie pentru cea mai buna solutie din fiecare situatie descrisa.

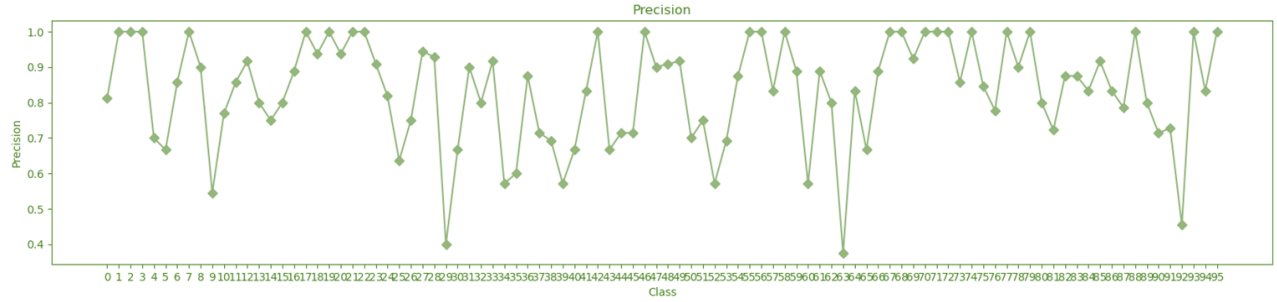
1.


Precision
Precision
Class


Recall
Recall
Class


Confusion Matrix
True Label
Predicted Label

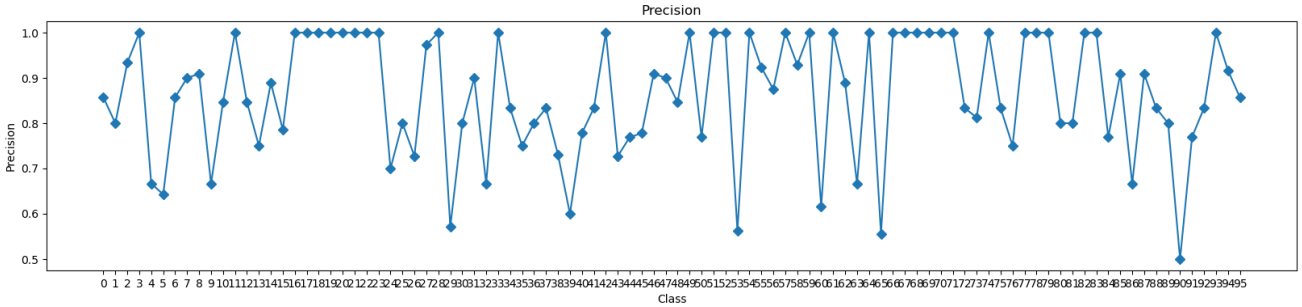
2.

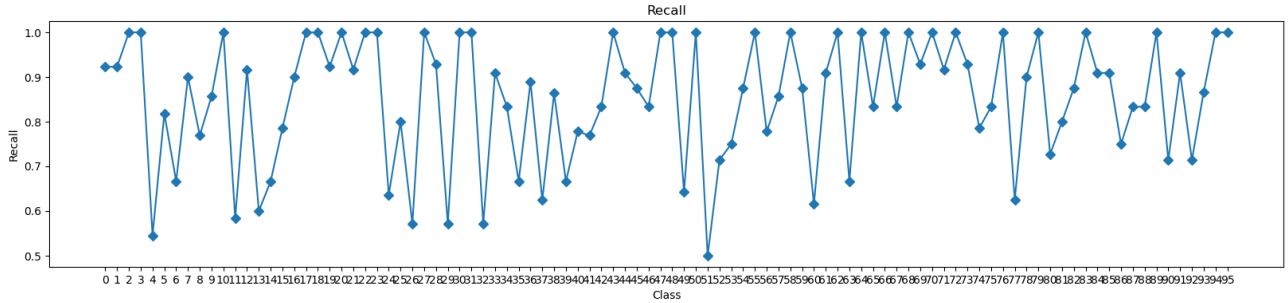

Precision
Precision
Class


Recall
Recall
Class


Confusion Matrix
True Label
Predicted Label

4.


Precision
Precision
Class


Recall
Recall
Class


Confusion Matrix
True Label
Predicted Label