Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki

Programowanie Komputerowe

CNN

autor	Mateusz Kucharczyk
prowadzący	mgr inż Grzegorz Wojciech Kwiatkowski
rok akademicki	2021/2022
kierunek	informatyka
rodzaj studiów	SSI
semestr	4
termin laboratorium	wt, 15:15 - 16:45
grupa	3
termin oddania sprawozdania	2022-06-13

1 Treść zadania

Napisać obiektowo program do klasyfikacji obrazków wykorzystując głębokie uczenie z warstwami konwolucyjnymi oraz funkcje aktywacji sigmoid. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-t` - tryb pracy
- `-c` - plik konfiguracyjny
- `-i` - obrazki wejściowe
- `-p` - adres (ip:port)

2 Algorytmy

2.1 Forward propagation

Dane wejściowe są przekazywane przez sieć do przodu. Każda ukryta warstwa przyjmuje dane z poprzedniej warstwy, przetwarza je w odpowiedni sposób i wysyła do kolejnej warstwy.

`-Dense`	matmul wag z wejściem plus bias następnie wynik jest poddawany funkcji aktywacji
`-Conv`	algorytm opisany w [1] plus bias następnie wynik jest poddawany funkcji aktywacji
`-Flatten`	spłaszczenie ND tablicy wejściowej do 1D ta warstwa nie posiada funkcji aktywacji

Wyjście ostatnie warstwy jest wynikiem, każdy neuron prezentuje procentową wartość dla każdej etykiety

2.2 Backpropagation

Najpierw liczony jest gradient ostatniej warstwy. W tym celu wykorzystywana jest pochodna funkcji starty Mean Square Error [2] oraz pochodna funkcji aktywacji. Gradientem ostatniej warstwy jest wynik pochodnej MSE przemnożony przez pochodną funkcji sigmoid wykorzystując chain rule. Następnie są liczone gradienty warstw od ostatniej do drugiej

`-Dense` Gradient wyjściowy dla poprzednej warstwy to
matmul gradientu wyjściowego
z tansponowanymi wagami

`-Conv` Gradient wyjściowy dla poprzednej warstwy to
pełna konwolucja obróconych o 180 stopni wag
z gradientem wyjściowym [3]

`-Flatten` Gradient wyjściowy dla poprzednej warstwy to
gradient wyjściowy o przywróconych kształtach

Następnie aktualizowane sa biasy oraz wagi

`-Dense` Nowe wagi to stare wagi minus
gradient wyjściowy przemnożony przez
pochodną funkcji aktywacji
przemnożony prez wyjście poprzedniej warstwy
przemnożony przez learning rate
Nowe biasy to stare biasy minus
gradient wyjściowy przemnożony przez
pochodną funkcji aktywacji
przemnożony przez learning rate

`-Conv` Nowe wagi to stare wagi minus
konwolucja gradientu wyjściowego z
wyjściem poprzedniej warstwy
przemnożona przez pochodną funkcji aktywacji
przemnożona przez learning rate
Nowy bias to stary bias minus
suma gradientu wyjściowego
przemnożonego przez funkcji aktywacji
przemnożona przez learning rate

`-Flatten` Tutaj wagi i biasy nie są aktualizowane

3 Specyfikacja zwenętrzna

Program uruchamiany jest z linii poleceń. Ma pięć trybów (eval, server, client, learnNew i learnContinue).

- -t eval -c plik konfiguracyjny -i folder z obrazkami do klasyfikacji

  Program pozyskuje wyuczone wagi i odtwarza model sieci. Następnie wykonuje wielowątkowo propagację dla każdego obrazka. Ilość wątków odpowiada zmiennej „batchSize” w pliku konfiguracyjnym. Na koniec zwraca ścieżkę do obrazka, najbardziej prawdopodobną predykcje oraz predykcje dla każdej etykiety wyrażoną w procentach

- -t server -c plik konfiguracyjny -p adres na którym ma pracować serwer

  Działa tak samo jak tryb eval tylko że wejście jest obrazkiem który przyśle klient zamiast folderu lokalnego. Na koniec zwraca klientowi odpowiedź

- -t client -p adres serwara -i obrazek do klasyfikacji

  Łączy się z serwerem, wysyła obrazek i wyświetla odpowiedź serwera czyli predykcje dla naszego obrazka

- -t learnNew -c plik konfiguracyjny

  Uczenie nowego modelu zawartego w pliku konfiguracyjnym. Tworzy model sieci neuronowej, wagi losuje zgodnie z rozkładem jednostajnym Xavier'a Glorot'a [4] z limitem

  $$\sqrt{\frac{6}{fan\_in + fan\_out}}$$

  gdzie fan_in oraz fan_out są liczone zgodnie z [5]. Algorytm uczenia jest opisany w punkcie 2. Najpierw jest wykonywana propagacja, następnie propagacja wsteczna po czym są aktualizowane wagi i baiasy. Jeśli ustawy „batchSize" na jeden otrzymamy stochastic gradient descent czyli propagujemy jeden obrazek liczymy stratę, gradienty warstw, aktualizujemy wagi, pobieramy następny losowy obrazek(obrazki w epoce nie powtarzają się). Kiedy wykonamy uczenie na każdym obrazku minie epoka. Jeśli ustawy „batchSize" na wartość większą od jeden otrzymamy mini batch gradient descent czyli wielowątkowo propagujemy N obrazków, liczymy straty oraz gradienty warstw dla każdego z N obrazków. Następnie liczymy średnią deltę wag i biasów po czym aktualizujemy je. Następnie pobierane jest N losowych obrazków. Zastosowanie MB GD z wykorzystaniem wielowątkowości znacznie skraca trawnie epoki natomiast uczy gorzej w porównaniu do SGD [6]

- -t learnContinue -c plik konfiguracyjny

  Kontunuacja uczenia

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem obiektowym z wykorzystaniem regexa, wątków, filesystem oraz modułów. Dodatkowe biblioteki to stb_image.h do odczytywania obrazków, json.h do przechowywania

pliku konfiguracyjnego oraz wag (nlohmann json) oraz asio do obsługi połączeń internetowych.

Literatura

[1] https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148

[2] Error https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error

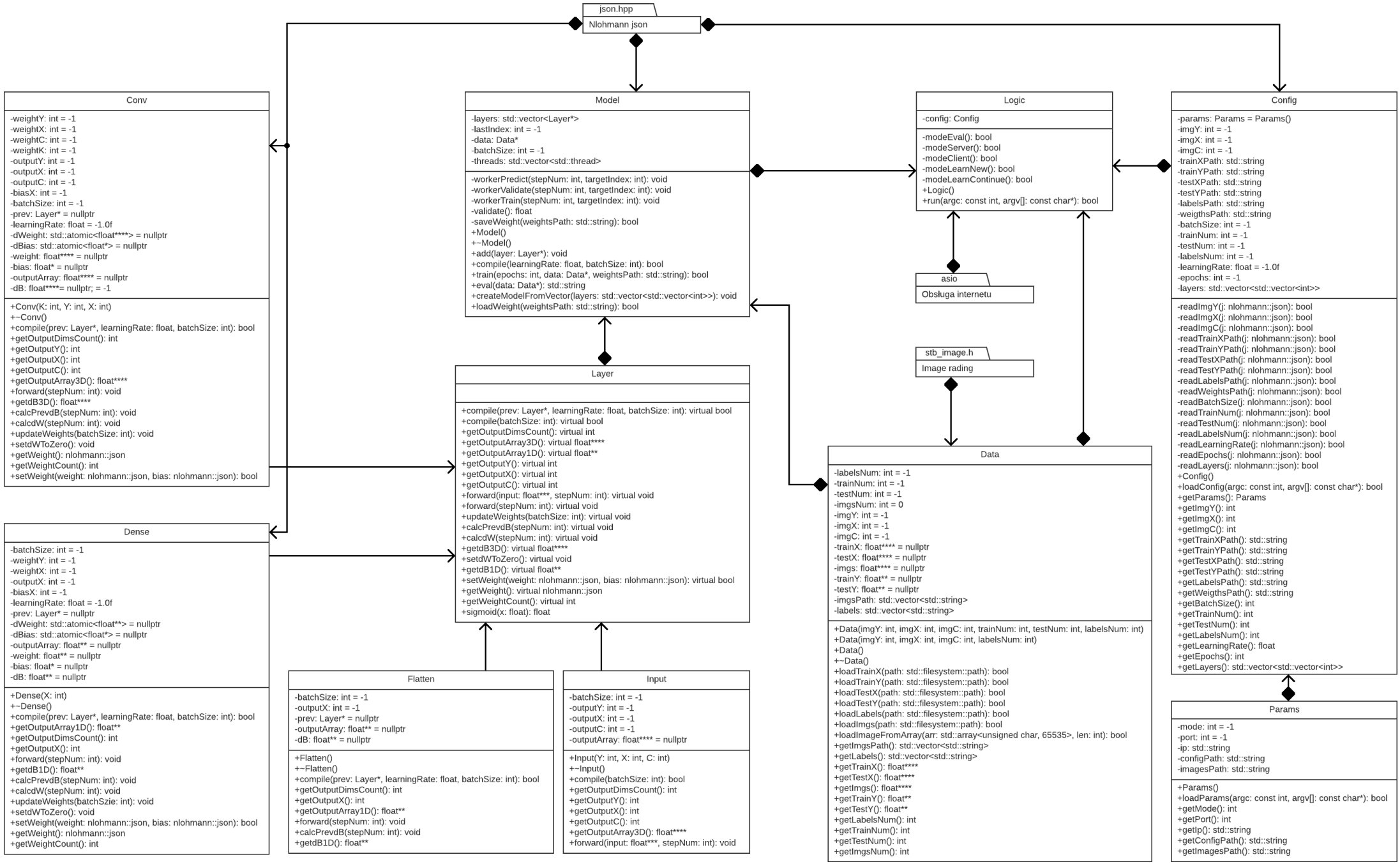
[3] https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c

[4] https://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

[5] https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/ops/init_ops.py#L1788-L1812

[6] https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

Diagram klas


json.hpp
Nlohmann json
Conv
-weightY: int = -1
-weightX: int = -1
-weightC: int = -1
-weightK: int = -1
-outputY: int = -1
-outputX: int = -1
-outputC: int = -1
-biasX: int = -1
-batchSize: int = -1
-prev: Layer* = nullptr
-learningRate: float = -1.0f
-dWeight: std::atomic<float****> = nullptr
-dBias: std::atomic<float*> = nullptr
-weight: float**** = nullptr
-bias: float* = nullptr
-outputArray: float**** = nullptr
-dB: float****= nullptr; = -1
+Conv(K: int, Y: int, X: int)
+~Conv()
+compile(prev: Layer*, learningRate: float, batchSize: int): bool
+getOutputDimsCount(): int
+getOutputY(): int
+getOutputX(): int
+getOutputC(): int
+getOutputArray3D(): float****
+forward(stepNum: int): void
+getdB3D(): float****
+calcPrevdB(stepNum: int): void
+calcdW(stepNum: int): void
+updateWeights(batchSize: int): void
+setdWToZero(): void
+getWeight(): nlohmann::json
+getWeightCount(): int
+setWeight(weight: nlohmann::json, bias: nlohmann::json): bool
Model
-layers: std::vector<Layer*>
-lastIndex: int = -1
-data: Data*
-batchSize: int = -1
-threads: std::vector<std::thread>
-workerPredict(stepNum: int, targetIndex: int): void
-workerValidate(stepNum: int, targetIndex: int): void
-workerTrain(stepNum: int, targetIndex: int): void
-validate(): float
-saveWeight(weightsPath: std::string): bool
+Model()
+~Model()
+add(layer: Layer*): void
+compile(learningRate: float, batchSize: int): bool
+train(epochs: int, data: Data*, weightsPath: std::string): bool
+eval(data: Data*): std::string
+createModelFromVector(layers: std::vector<std::vector<int>>): void
+loadWeight(weightsPath: std::string): bool
Logic
-config: Config
-modeEval(): bool
-modeServer(): bool
-modeClient(): bool
-modeLearnNew(): bool
-modeLearnContinue(): bool
+Logic()
+run(argc: const int, argv[]: const char*): bool
asio
Obsługa internetu
Config
-params: Params = Params()
-imgY: int = -1
-imgX: int = -1
-imgC: int = -1
-trainXPath: std::string
-trainYPath: std::string
-testXPath: std::string
-testYPath: std::string
-labelsPath: std::string
-weigthsPath: std::string
-batchSize: int = -1
-trainNum: int = -1
-testNum: int = -1
-labelsNum: int = -1
-learningRate: float = -1.0f
-epochs: int = -1
-layers: std::vector<std::vector<int>>
-readImgY(j: nlohmann::json): bool
-readImgX(j: nlohmann::json): bool
-readImgC(j: nlohmann::json): bool
-readTrainXPath(j: nlohmann::json): bool
-readTrainYPath(j: nlohmann::json): bool
-readTestXPath(j: nlohmann::json): bool
-readTestYPath(j: nlohmann::json): bool
-readLabelsPath(j: nlohmann::json): bool
-readWeightsPath(j: nlohmann::json): bool
-readBatchSize(j: nlohmann::json): bool
-readTrainNum(j: nlohmann::json): bool
-readTestNum(j: nlohmann::json): bool
-readLabelsNum(j: nlohmann::json): bool
-readLearningRate(j: nlohmann::json): bool
-readEpochs(j: nlohmann::json): bool
-readLayers(j: nlohmann::json): bool
+Config()
+loadConfig(argc: const int, argv[]: const char*): bool
+getParams(): Params
+getImgY(): int
+getImgX(): int
+getImgC(): int
+getTrainXPath(): std::string
+getTrainYPath(): std::string
+getTestXPath(): std::string
+getTestYPath(): std::string
+getLabelsPath(): std::string
+getWeigthsPath(): std::string
+getBatchSize(): int
+getTrainNum(): int
+getTestNum(): int
+getLabelsNum(): int
+getLearningRate(): float
+getEpochs(): int
+getLayers(): std::vector<std::vector<int>>
Layer
+compile(prev: Layer*, learningRate: float, batchSize: int): virtual bool
+compile(batchSize: int): virtual bool
+getOutputDimsCount(): virtual int
+getOutputArray3D(): virtual float****
+getOutputArray1D(): virtual float**
+getOutputY(): virtual int
+getOutputX(): virtual int
+getOutputC(): virtual int
+forward(input: float***, stepNum: int): virtual void
+forward(stepNum: int): virtual void
+updateWeights(batchSize: int): virtual void
+calcPrevdB(stepNum: int): virtual void
+calcdW(stepNum: int): virtual void
+getdB3D(): virtual float****
+setdWToZero(): virtual void
+getdB1D(): virtual float**
+setWeight(weight: nlohmann::json, bias: nlohmann::json): virtual bool
+getWeight(): virtual nlohmann::json
+getWeightCount(): virtual int
+sigmoid(x: float): float
stb_image.h
Image rading
Data
-labelsNum: int = -1
-trainNum: int = -1
-testNum: int = -1
-imgsNum: int = 0
-imgY: int = -1
-imgX: int = -1
-imgC: int = -1
-trainX: float**** = nullptr
-testX: float**** = nullptr
-imgs: float**** = nullptr
-trainY: float** = nullptr
-testY: float** = nullptr
-imgsPath: std::vector<std::string>
-labels: std::vector<std::string>
+Data(imgY: int, imgX: int, imgC: int, trainNum: int, testNum: int, labelsNum: int)
+Data(imgY: int, imgX: int, imgC: int, labelsNum: int)
+Data()
+~Data()
+loadTrainX(path: std::filesystem::path): bool
+loadTrainY(path: std::filesystem::path): bool
+loadTestX(path: std::filesystem::path): bool
+loadTestY(path: std::filesystem::path): bool
+loadLabels(path: std::filesystem::path): bool
+loadImgs(path: std::filesystem::path): bool
+loadImageFromArray(arr: std::array<unsigned char, 65535>, len: int): bool
+getImgsPath(): std::vector<std::string>
+getLabels(): std::vector<std::string>
+getTrainX(): float****
+getTestX(): float****
+getImgs(): float****
+getTrainY(): float**
+getTestY(): float**
+getLabelsNum(): int
+getTrainNum(): int
+getTestNum(): int
+getImgsNum(): int
Params
-mode: int = -1
-port: int = -1
-ip: std::string
-configPath: std::string
-imagesPath: std::string
+Params()
+loadParams(argc: const int, argv[]: const char*): bool
+getMode(): int
+getPort(): int
+getIp(): std::string
+getConfigPath(): std::string
+getImagesPath(): std::string
Dense
-batchSize: int = -1
-weightY: int = -1
-weightX: int = -1
-outputX: int = -1
-biasX: int = -1
-learningRate: float = -1.0f
-prev: Layer* = nullptr
-dWeight: std::atomic<float**> = nullptr
-dBias: std::atomic<float*> = nullptr
-outputArray: float** = nullptr
-weight: float** = nullptr
-bias: float* = nullptr
-dB: float** = nullptr
+Dense(X: int)
+~Dense()
+compile(prev: Layer*, learningRate: float, batchSize: int): bool
+getOutputArray1D(): float**
+getOutputDimsCount(): int
+getOutputX(): int
+forward(stepNum: int): void
+getdB1D(): float**
+calcPrevdB(stepNum: int): void
+calcdW(stepNum: int): void
+updateWeights(batchSzie: int): void
+setdWToZero(): void
+setWeight(weight: nlohmann::json, bias: nlohmann::json): bool
+getWeight(): nlohmann::json
+getWeightCount(): int
Flatten
-batchSize: int = -1
-outputX: int = -1
-prev: Layer* = nullptr
-outputArray: float** = nullptr
-dB: float** = nullptr
+Flatten()
+~Flatten()
+compile(prev: Layer*, learningRate: float, batchSize: int): bool
+getOutputDimsCount(): int
+getOutputX(): int
+getOutputArray1D(): float**
+forward(stepNum: int): void
+calcPrevdB(stepNum: int): void
+getdB1D(): float**
Input
-batchSize: int = -1
-outputY: int = -1
-outputX: int = -1
-outputC: int = -1
-outputArray: float**** = nullptr
+Input(Y: int, X: int, C: int)
+~Input()
+compile(batchSize: int): bool
+getOutputDimsCount(): int
+getOutputY(): int
+getOutputX(): int
+getOutputC(): int
+getOutputArray3D(): float****
+forward(input: float***, stepNum: int): void