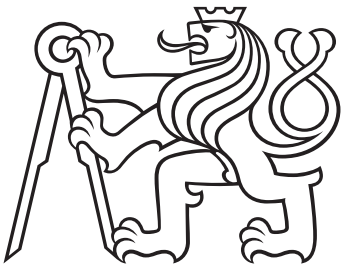
České vysoké učení technické v Praze
Fakulta strojní

Ústav technické matematiky
Studijní program: Aplikované vědy ve strojním inženýrství
Specializace: Matematické modelování v technice



Využití strojového učení pro simulace obtékání profilu

Application of machine learning for simulation of flows over a profile

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. Josef Černík
Vedoucí práce: prof. Ing. Jiří Fürst, Ph.D.
Rok: 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Černík Josef** Personal ID number: **483131**

Faculty / Institute: **Faculty of Mechanical Engineering**

Department / Institute: **Department of Technical Mathematics**

Study program: **Applied Sciences in Mechanical Engineering**

Specialisation: **Mathematical Modeling in Engineering**

II. Master’s thesis details

Master’s thesis title in English:

Application of machine lerning for simulation of flows over a profile

Master’s thesis title in Czech:

Využití strojového učení pro simulace obtékání profilu

Guidelines:

Bibliography / sources:

- Ribeiro, Mateus Dias, et al. "DeepCFD: Efficient steady-state laminar flow approximation with deep convolutional neural networks." arXiv preprint arXiv:2004.08826 (2020).
- H. G. Weller, G. Tabor, H. Jasak, C. Fureby, A tensorial approach to computational continuum mechanics using object-oriented techniques, Computers in Physics 12 (1998) 620–631
- . Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org

Name and workplace of master’s thesis supervisor:

prof. Ing. Jiří Fürst, Ph.D. Department of Technical Mathematics FME

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **19.04.2023** Deadline for master's thesis submission: **13.08.2023**

Assignment valid until: _____

prof. Ing. Jiří Fürst, Ph.D. Supervisor’s signature	prof. Ing. Jiří Fürst, Ph.D. Head of department’s signature	doc. Ing. Miroslav Španiel, CSc. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne .....

.....
Bc. Josef Černík

Poděkování

Děkuji prof. Ing. Jiřímu Fürstovi, Ph.D. za vedení mé diplomové práce, ochotu a neocenitelné rady, které práci obohatily. Dále bych chtěl poděkovat své rodině a blízkým za významnou podporu při studiu.

Bc. Josef Černík

Název práce:
Využití strojového učení pro simulace obtékání profilu

Autor: Bc. Josef Černík

Studijní program: Aplikované vědy ve strojním inženýrství
Specializace: Matematické modelování v technice
Druh práce: Diplomová práce

Vedoucí práce: prof. Ing. Jiří Fürst, Ph.D.

Abstrakt: Tato práce popisuje základní problematiku neuronových sítí. Dále jsou různé typy neuronových sítí aplikovány na problém obtékání profilu, konkrétně jde o predikci tlakového pole. Učící datová sada je vytvořena pomocí numerického řešiče OpenFoam. Na této sadě je natrénována hustá a konvoluční neuronová síť architektury Unet. Práce obsahuje použité algoritmy pro přípravu a zpracování dat a metodiku automatizace dílčích procesů. Dále demonstruje výhodnost použití architektury Unet pro daný problém na základě získaných podkladů. Obsaženo je také vyhodnocení přesnosti predikce modelů a porovnání výpočetního času.

Klíčová slova: NACA, profil, OpenFoam, SIMPLE, hluboké učení, neuronové sítě, konvoluční vsrtvy, Unet, trénink, optimalizace

Title:
Application of machine learning for simulation of flows over a profile

Author: Bc. Josef Černík

Abstract: Thesis describes basic problematic around neural networks. Different types of neural nets are applied on flow over profile problem, models are specifically used for pressure field prediction. Learning dataset is generated with OpenFoam numerical solver. On this dataset is trained models of dense and fully convolutional Unet neural networks. Thesis contains explanation of used algorithms for data processing, manipulation and process automation. Also advantage of using convolutional nets for given problem is demonstrated based on acquired results. Precission evaluation and computational time comparison is also included.

Key words: NACA, airfoil, OpenFoam, SIMPLE, deep learning, neural networks, convolutional layers, Unet, training, optimalization

Obsah

Seznam použitých zkratek viii

Seznam obrázků xi

Úvod 1

1 Seznámení s problematikou 2

2 Architektura a princip neuronových sítí 5
2.1 Aktivační funkce . . . 5
2.2 Husté neuronové sítě . . . 7
2.3 Trénink hustých neuronových sítí . . . 10
2.3.1 Gradient descent . . . 11
2.3.2 Backpropagation . . . 12
2.3.3 Rozšíření pro více vzorků . . . 14
2.4 Konvoluční neuronové sítě . . . 16
2.4.1 Konvoluční vrstvy . . . 16
2.4.2 Podvzorkovací vrstvy . . . 18
2.4.3 Transponované konvoluční vrstvy . . . 19
2.4.4 Trénink konvolučních sítí . . . 19

3 Formulace úlohy a numerické výpočty 20

4 Aplikace neuronových sítí na řešení problému 23
4.1 Formulace úlohy pro neuronovou síť . . . 23
4.2 Příprava tréninkového a validačního souboru dat . . . 24
4.2.1 Výpočet vstupního pole vzdáleností **I** . . . 25
4.2.2 Příprava výstupního tlakového pole . . . 29
4.2.3 Automatické generování vzorků . . . 30
4.3 Tvorba neuronové sítě . . . 32
4.3.1 Hustá neuronová síť . . . 33
4.3.2 Neuronová síť U-net . . . 35
4.3.3 Trénink a optimalizace U-net sítě . . . 37
4.4 Potřebné procesy pro interpretaci výsledků . . . 40
4.4.1 Numerický výpočet gradientu veličiny . . . 40
4.4.2 Rekonstrukce tvaru profilu z pole vzdáleností . . . 41
4.4.3 Seřazení bodů na hranici profilu . . . 43
4.4.4 Zahuštění hranice profilu . . . 43
4.4.5 Interpolace tlaku na hranici z tlakového pole . . . 45
4.4.6 Numerická integrace pro získání tlakového koeficientu . . . 46

5 Vyhodnocení výsledků 47
5.1 Srovnání výpočetních časů . . . 47

5.2 Profil NACA 4415 48
5.3 Profil SD7003 50
5.4 Analýza rozlišení 53
5.5 Vyhodnocení přesnosti na vybraných vzorcích 54

Závěr **56**

Bibliografie **57**

Přílohy **59**
A První příloha 59
B Druhá příloha 59
B.1 Profil S826 59
B.2 Profil K3 61

Seznam použitých zkratek

CFD Výpočetní dynamika tekutin (*Computational fluid dynamics*)

CNN Konvoluční neuronová síť (*Convolutional neural network*)

FNN Dopředná neuronová síť (*Feedforward neural network*)

Seznam použitých symbolů

α	Úhel náběhu
χ	Parametr *stride*
δ	Pomocný parametr při *Backpropagation*
η	Parametr *learning rate*
Γ	Hranice
$\hat{\mathbf{O}}$	Pole hodnot "správného řešení" získané z OpenFoam
$\hat{\xi}$	Průměrná chybová funkce
$\hat{d}$	Euklidovská vzdálenost
κ	Rozměr filtru
$\mathbb{N}$	Množina přirozených čísel
$\mathbb{R}$	Množina reálných čísel
$\mathbb{Z}$	Množina celých čísel
$\nu_t, \tilde{\nu}$	Parametry turbulence
ψ	Parametr *padding*
σ	Aktivační funkce Sigmoid
$\mathbf{E}$	Pole chyby
$\mathbf{I}$	Transformované pole vzdáleností
$\mathbf{O}$	Pole predikce neuronové sítě
$\tilde{\mathbf{I}}$	Vstupní pole vzdáleností
$\tilde{p}$	Kinematický tlak
φ	Aktivační funkce
$\vec{b}$	Vektor posunutí
$\vec{G}$	Vektor bodů hranice profilu

$\vec{u}$	Rychlost ve 2D
$\vec{v}$	Vektor trénovatelných parametrů
$\vec{Y}$	Vektor správných hodnot
ξ	Chybová funkce
b_c	Celkový počet posunutí
c_l	Koeficient vztlaku
c_p	Koeficient tlaku
d_i	Rozměr i-tého filtru
i, j, k, l, m, p, h, r	Indexy
I^p	Počet vstupních kanálů p-té vrstvy
K	Parametry filtru
N	Počet vrstev neuronové sítě
n_i	Počet neuronů i-té vrstvy
O^p	Počet výstupních kanálů p-té vrstvy
$P(q)$	Čtvercové pole s q body v každém směru
q	Parametr určující počet bodů pole
S	Počet vzorků v sadě dat
s	Parametr *batch size*
v_c	Celkový počet trénovatelných parametrů
w, W	Parametry vah
w_c	Celkový počet vah
x, y, z	Hodnoty vrstev neuronových sítí

Seznam obrázků

2.1 Aktivační funkce SiLU 6
2.2 Schéma FNN 7
2.3 Schéma k-tého neuronu p-té vrstvy 8
2.4 Část neuronové sítě 12
2.5 Aplikace filtru na data 16
2.6 Posouvání submatice Ω_{kl} 17

3.1 Schéma řešené úlohy 20
3.2 Výpočetní síť (NACA 5412, úhel náběhu 2°) 22

4.1 Profil $\vec{G}$ a body oblasti $P(9)$ 24
4.2 Reprezentace geometrie profilu 25
4.3 Určení normálové vzdálenosti 26
4.4 Určení normálové vzdálenosti 27
4.5 Vstupní pole vzdáleností $\tilde{\mathbf{I}}$ a transformovaných vzdáleností $\mathbf{I}$ pro NACA 0012, $\alpha = 0°$ 28
4.6 Detail pole transformovaných vzdáleností $\mathbf{I}$ 29
4.7 Výstupní tlakové pole pro NACA 0012, $\alpha = 0°$ 30
4.8 Vybrané příklady ze vzorku dat s 4-cifernými NACA profily 32
4.9 Průběh hodnoty chyby při tréninku husté sítě 34
4.10 Predikce hustou neuronovou sítí pro posunutý profil 35
4.11 Konvoluční blok 36
4.12 U-net architektura 37
4.13 Závislost chybové funkce na epochu při tréninku modelu Unet1 40
4.14 Oblast výskytu bodů ležící blízko hranice $\vec{G}$ 41
4.15 Vybrané body P^h 42
4.16 Rekostruovaný profil z pole vzdáleností $\tilde{\mathbf{I}}$ 43
4.17 Interpolace tlaku c_p z pole $\mathbf{O}$ na hranici 45

5.1 Vstupní data a predikce pro 4415, $\alpha = 3°$ 48
5.2 Porovnání predikce a výsledku z OpenFoam pro NACA 4415, $\alpha = 3°$ 48
5.3 Rozložení chyby $\mathbf{E}$ pro výsledky z obr. 5.2 49
5.4 NACA 4415, $\alpha = 3°$, interpolace na hranici 49
5.5 Závislost $c_l(\alpha)$ pro profil NACA 4415 50
5.6 Porovnání predikce a výsledku z OpenFoam pro SD7003, $\alpha = -8°$ . . 51
5.7 Rozložení chyby $\mathbf{E}$ z obrázku 5.6 51
5.8 SD7003 $\alpha = -8°$, interpolace veličin na hranici 52
5.9 Závislost $c_l(\alpha)$ pro profil SD7003 53
5.10 interpolace c_p na hranici pro porovnání rozlišení ,profil SD7003, $\alpha = -8°$ 53

11 Predikce koeficientu tlaku pro posunutý profil NACA 0012, $\alpha = 0°$ (o $\Delta y = -0.3$) 59
12 Porovnání predikce a výsledku z OpenFoam pro S826, $\alpha = 3°$ 59
13 Rozložení chyby **E** z obrázku 12 60
14 S826, $\alpha = 3°$, interpolace veličin na hranici 60
15 Závislost $c_l(\alpha)$ pro profil S826 61
16 Porovnání predikce a výsledku z OpenFoam pro K3, $\alpha = 3°$ 61
17 Rozložení chyby **E** z obrázku 16 62
18 K3, $\alpha = 3°$, interpolace veličin na hranici 62
19 Závislost $c_l(\alpha)$ pro profil K3 63

Úvod

Obor strojírenství a aplikovaných věd je zároveň úzce spjat s oborem softwarového inženýrství. Vzhledem k tomu, že neustále dochází k vývoji výkonnějšího hardwaru (především procesorů a grafických karet), pojem *umělá inteligence* začínáme slýchat v běžném životě čím dál více. Pro současného inženýra je výhodné znát tyto principy sofwarového inženýrství a programování, protože mohou významně ušetřit výpočetní čas a přinést nové možnosti například při optimalizaci. Zajímavou aplikací se naskytuje využití umělé inteligence pro CFD[1]. Tato práce je zaměřena na jednu z její podoblastí, a to hluboké učení neboli *deep learning*. Jedná se o obor, který se zabývá problematikou neuronových sítí. Jejich úkolem je napodobit chování lidského mozku. Lidský mozek, když chybuje, zapamatuje si situaci a přizpůsobí se tak, aby příště při stejném ději chyboval méně. Stejně tak neuronová síť konkrétní jev vyhodnocuje, aby stanovila odchylku od správného výsledku nebo chování a jejím cílem je přizpůsobit si parametry jednotlivých buněk (neuronů) tak, aby výsledná chyba byla minimální. Tyto algoritmy dříve nebyly příliš uplatňovány, protože byly limitovány výkonem hardwaru. Vzhledem k tomu, že zpracovávají a vyhodnocují objemný soubor dat, nemohly být dostatečně efektivní a jejich tzv. trénink (viz dále) by trval příliš dlouho.

Cílem této práce je vytvořit a vyhodnotit neuronovou síť, která bude umět na základě dané geometerie predikovat pole tlakového koeficientu v případě stacionárního turbulentního 2D proudění nestlačitelné tekutiny za daných okrajových podmínek pro případ obtékání profilu. Jedná se tedy o přípravu a zpracování dat k formulaci problému pro neuronovou síť, nalezení vhodné struktury a vhodných parametrů neuronové sítě, přípravu a provedení tréninku konkrétního modelu a také diskuzi získaných dat na testovacích případech.

[1] Výpočetní dynamika tekutin (*Computational fluid dynamics*)

Kapitola 1

Seznámení s problematikou

Obor zabývající se neuronovými sítěmi se nazývá *deep learning* a spadá pod discilínu *machine learning*. Dle [1] str. 1 byly neuronové sítě jedním z prvních algoritmů modelů strojového učení. První koncepty byly představeny už roku 1940. V roce 2006 se Geoffreymu Hintonovi podařilo vylepšit a optimalizovat učící algoritmus (*backpropagation*) tak, že bylo možné konečně efektivně trénovat vícevrstvé modely. Geoffrey Hinton a jeho 3 další spolupracovníci (Yann LeCun, Yoshua Bengio a Andrew Ng), kteří se znamenitě podíleli na vývoji, jsou považováni za novodobé zakladatele tohoto směru. V této době začala současná éra neuronových sítí. Slovo *deep* je v názvu použito, protože neuronové sítě, které se skládají ze vstupní a výstupní vrsty a několika dalších skrytých vrstev neuronů (tzv. *hidden layers*), se nazývají *hluboké*. *Deep learning* (Hluboké učení) poskytuje principy, jak tyto hluboké neuronové sítě trénovat.

Základní architekturou neuronových sítí je tzv. FNN[1], která se skládá pouze z několika hustých vrstev (alespoň dvou) [2] str. 21. Na jejích principech je mnoho pojmů z oblasti neuronových sítí v této práci vysvětleno. Vrstva se nazývá hustá, pokud každý neuron příslušné vrstvy ovlivňuje každý neuron té následující a zároveň je ovlivněn každým neuronem předchozí vrsty. Pojem neuron lze chápat jako samostatnou buňku vrstvy, která v sobě nese hodnotu. Propojení neuronů je reprezentováno relací, podle které se hodnoty neuronů odlišných vrstev navzájem ovlivňují.

K provedení tréninku je potřeba obsáhlý soubor dat, který pro každý vzorek obsahuje vstupní data a požadovaný výstup (správné řešení). Tréninkem modelu se rozumí přizpůsobování parametrů relačních vztahů mezi neurony tak, aby byla chyba mezi ideálním výstupem a výstupem modelu minimální. Relačními vztahy mezi neurony jsou v případě FNN jednoduché lineární funkce. Jelikož je účelem modelovat obecně nelineární problémy, je potřeba vnést nelinearitu i do modelu. To se řeší použitím tzv. aktivačních funkcí (které jsou pro tento účel nelineární) na hodnoty jednotlivých neuronů. Trénink se realizuje cyklem přes jednotlivé vzorky množiny učících dat tak, že se poskytnou vstupní vrstvě sítě vstupní data vzorku, proběhne vyhodnocení výsledku (predikce) a ten se porovná s ideálním výstupem vzorku učící sady. Slovem porovnání je myšleno vyhodnocení chybovosti pomocí chybové funkce (což je jedním z hyper-parametrů tréninku), která přijme dva argumenty (požadovaný výsledek a predikci) a vrací hodnotu chyby (reálné číslo). Po inicializaci modelu jsou parametry jednotlivých relací mezi neurony nastaveny zpravidla náhodně[2], tudíž predikce modelu má obecně bez jakéhokoliv tréninku vysokou hodnotu chyby.

[1] Dopředná neuronová síť (*Feedforward neural network*)

[2] Převážně se jedná o čísla kolem nuly z intervalu $(-1, 1)$. Více podrobností lze najít např. v [3].

Poté, co je vyčíslena chyba, se dají pomocí sofistikovaných algoritmů parametry vztahů mezi neurony upravit tak, aby při dalším vyhodnocení byla chyba menší. Iterováním přes soubor dat se zaručí, aby se parametry modelu co nejlépe přizpůsobily všem vzorkům souboru. Po první iteraci přes všechny vzorky většinou nelze očekávat uspokojivé výsledky, proto se běžně provádí více cyklů přes všechna data, dokud není hodnota chyby minimální. Stručně řečeno, cílem je najít bod minima chybové funkce.

Uživatel při použití modelu komunikuje pouze se vstupní vrstvou, která přijímá data a výstupní vrstvou, která vrací predikci. Většina výpočtů probíhá uvnitř skrytých vrstev. Počet vstupních a výstupních neuronů závisí na formulaci problému. Přidáním skrytých vrstev dojde k přidání parametrů modelu a zvýší se tak komplexnost modelu a schopnost lépe pracovat s rozmanitějšími daty. Lze si to představit tak, že čím více skrytých vrstev model obsahuje, tím více vzorů a souvislostí v poskytnutých datech je schopný najít a naučit se.

Hlavním účelem neuronových sítí je klasifikace (zjednodušeně rozřazování vzorků do kategorií) a regrese (přiřazení vzorku spojitou hodnotu), popřípadě jejich kombinace. Přesto, že neurovové sítě nepřinášejí signifikantně vyšší přesnost než jiné machine learning algoritmy, jako např. *Support Vector Machines*, *Random Forests* a *K-Means Clustering* [1] str. 4, při aplikaci na nízkodimenzionální data, hrají významnou roli při zpracovávání videa, obrazu, textu, zvuku apod. Existují různé jejich modifikace, jako např. CNN[3], které dokážou na obraze rozpoznávat jednotlivé vzory a na základě toho se rozhodovat. Ukázkou využití konvolučních neurovnových sítí pro zpracování obrazu při CFD je i tato práce. Vzhledem k tomu, že neuronové sítě lze při vhodné formulaci problému aplikovat na různé děje v různých oborech, mají v současnosti velice široké využití.

Jedním z nejznámějších a nejjednodušších příkladů aplikace konvoluční neuronové sítě je model pro klasifikaci psů a koček na obraze, což je blíže popsáno např. v literatuře [4]. Jde spíše o ilustrační příklad. Model se tak po absolvování tréninku dokáže naučit rozpoznávat důležité vzory v obraze a na základně nich s vysokou přesností klasifikovat, zda je na obraze pes nebo kočka.

Zajímavou konstrukcí konvoluční neuronové sítě je tzv. U-net architektura, která byla poprvé použita pro zvýraznění a segmentaci hranic buněk v publikaci [5]. Tato architektura je výkonná v rozpoznávání hran v obraze, protože pracuje na principu předávání informací o kontrastních přechodech z minulých vrstev (viz dále). Jak vyplývá z výsledků práce, model je schopen spolehlivě provádět segmentaci různých biomedikálních obrazů a vyznačit hranice mezi buňkami nebo látkami.

Použití CFD pro řešení fyzikálních problémů pomocí složitých numerických algoritmů s sebou často nese vysoké náklady, zejména potom na množství výpočetního času. Tento náklad omezuje např. možnosti optimalizace nebo výběr správné geometrie pro dosažení požadovaných výsledků. Pokud je možné pro konkrétní popis CFD problému najít souvislosti a podobnosti ve smyslu řešených rovnic, okrajových podmínek, geometrií, vlastností látek apod, a zároveň je potřeba numerický experiment provádět opakovaně pro odlišnou sadu parametrů, může být výhodné použít neuronovou síť. Při vhodné formulaci úlohy, model za cenu dostatečně malé chyby umožní ušetřit významnou část výpočetního času.

Pro CFD výpočty byla konvoluční neuronová síť aplikována v práci [6]. Autoři zde neuronovou síť formulovali jako produkt pro řešení Navier-Stokesových rovnic

[3]Konvoluční neuronová síť (*Convolutional neural network*)

problému obtékání vzduchu kolem různých geometrií náhodných útvarů ve 2D, přičemž experimentovali s různými architekturami a parametry modelu. Jejich vstupem do modelu byla reprezentace geometrie obtékaného tělesa a výstupem 3 fyzikální pole: rychlost ve směru x, rychlost ve směru y a tlak. Ve výsledcích je posouzena přesnost modelů různých architektur srovnáním predikce modelu s CFD výpočty softwarovým balíčkem OpenFoam. Z této publikace vyplývá výhodnost použití konvolučních neuronových sítí a obzvláště U-net architektury pro podobné problémy.

V práci [7] se autor zabýval řešením tlakového pole pomocí CNN při obtékání NACA profilu pro stanovení vztlakového a odporového koeficientu. Použitý model neobsahoval pouze konvoluční vrstvy, ale mezi částmi *encoder* a *decoder* bylo vloženo několik hustých vrstvev. CFD výpočty byly prováděny na C-síti a model, který obsahoval aktivační funkce ReLU (viz dále), byl trénován na datech vygenerovaných pomocí open-source softwaru FlowPro. V práci je uvedeno, že predikce tlakových polí všech tréninkových vzorků pomocí modelu je oproti CFD výpočtům zhruba 1500x rychlejší. V závěrech je uvedena relativní chyba vztlakového koeficientu získaného z predikce maximálně okolo 5% na datech neobsažených v tréninkovém souboru.

Kapitola 2

Architektura a princip neuronových sítí

2.1 Aktivační funkce

Dá se dokázat, že neuronová síť může aproximovat libovolnou funkci [8] přinejhorším po částech hladkou funkcí. Jak bude výsledná aproximace vypadat závisí na volbě aktivačních funkcí jednotlivých vrstev. Aktivační funkce φ je obvykle neklesající zobrazení

$$\varphi : \mathbb{R} \mapsto \mathbb{R}.$$

Jedná se o funkce, kterými disponují určené neurony, resp. vrstvy neuronů, a používají ji k výpočtu své výstupní hodnoty. Obvykle se tyto nelineární funkce aplikují na skryté vrstvy neuronů za účelem narušení linearity modelu či filtrování nebo redukování vysokých hodnot amplitud neuronů. Bez použití aktivačních funkcí by byl výstup modelu y pouze lineární funkcí, což je pouze polynom prvního řádu, $y \in P_1(y)$. Přestože pracovat s polynomem prvního řádu je snadné, nemá schopnost rozpoznávat složitější vzory a vztahy mezi vstupními daty a model by byl identický s modelem lineární regrese [9] str. 310 s limitovanou přesností. Použití lineární regrese je nevhodné pro komplikovanější problémy jako např. rozpoznávání zvuku nebo obrazu. Toto je důvodem pro zavedení aktivačních funkci, které pomohou komplexním modelům pro práci s vysokodimenzionálními daty obsahujícím několik skrytých vrstvev docílit větší přesnosti a výkonu. Existuje celá řada aktivačních funkcí, které mají pro určité případy své využití. Volba aktivační funkce může ovlivnit rychlost konvergence, rychlost tréninku a přesnost modelu. Několik důležitých je dále uvedeno.

Pro pochopení je důležité zmínit nejjednodušší tzv. aktivační funkci *Threshold*, která funguje jako přepínač a rozhoduje, jestli neuron na základě své hodnoty bude aktivní nebo ne. Tato funkce se dá matematicky definovat jako

$$\varphi_{th}(z) = \begin{cases} 1, & \text{pro z} \geq 0 \\ 0, & \text{pro z} < 0 \end{cases} \tag{2.1}$$

Nevýhodou jejího použití je, že její derivace, která je potřeba při *Backpropagation* algoritmu (sekce 2.3), je na celém intervalu nulová a nedává žádnou informaci, jak přizpůsobit parametry modelu pro lepší výsledek.

Nejpoužívanější aktivační funkcí je *Sigmoid* [9]. Jedná se o nelineární hladkou funkci popsanou vztahem

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{2.2}$$

která vrací spojitou hodnotu $\sigma(z) \in (0, 1)$ pro $z \in \mathbb{R}$. Vzhledem k tomuto omezení je možné ji škálovat. Sigmoid je výhodné při úloze klasifikace aplikovat na výstupní vrstvu neuronů, kdy je požadováno, aby neurony vracely hodnoty reprezentující pravděpodobnost v procentech. Výhodou jejího použití je snadný výpočet její derivace, což se dá odvodit jako

$$\sigma(z)' = \sigma(z) \cdot (1 - \sigma(z)) \,. \tag{2.3}$$

Dalším příkladem je aktivační funkce *Rectified Linear Unit* [9], neboli ReLU. Má pro neuronové sítě široké využití, protože je jednoduché ji vyhodnotit a zároveň není lineární. Deaktivuje neurony se zápornými hodnotami a tím je způsobeno, že příslušné neurony reagují pouze na parametry, na které jsou trénovány a neznehodnocují data pro parametry, se kterými nemají nic společného. Tato funkce se dá jednoduše matematicky vyjádřit.

$$\varphi_{ReLU}(z) = \max(0, z) \tag{2.4}$$

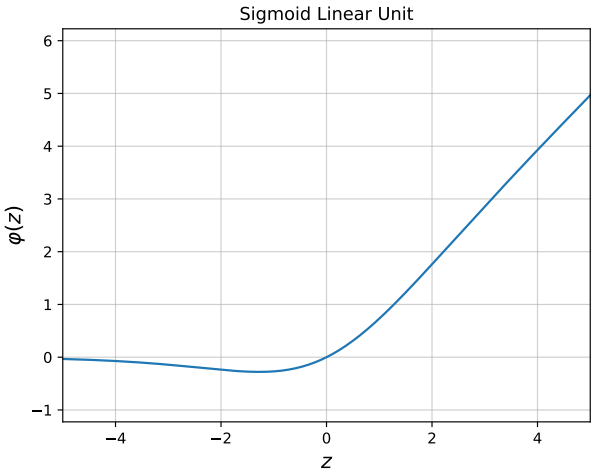
Jak lze odvodit, její derivace je skoková funkce

$$\varphi_{ReLU}(z)' = \begin{cases} 1, & \text{pro z} > 0 \\ 0, & \text{pro z} \leq 0. \end{cases} \tag{2.5}$$

Protože ReLU nemá matematicky jednoznačně definovanou derivaci v nule, různí autoři se liší v její umělé definici. Tato vlastnost má zásadní vliv při tréninku, protože se tímto dosáhne pouze přizpůsobování těch parametrů, které na daný vstup kladně reagují a nejsou tak narušeny parametry reagující na odlišné vstupy.

Pro tuto práci je také významná aktivační funkce *Sigmoid Linear Unit* (SiLU) viz. [10]. Tato funkce má podobný průběh jako ReLU, ale navíc do určité míry zohledňuje mírně záporné hodnoty. Její předpis je

$$\varphi_{SiLU}(z) = z \cdot \sigma(z). \tag{2.6}$$



Obrázek 2.1: Aktivační funkce SiLU

V práci je dále znázorněno, jak použití této aktivační funkce ve srovnání s ReLU příznivě ovlivní přesnost modelu pro daný případ. Důvodem je hladký průběh na celém intervalu a existence derivace v nule.

Přesto, že jsou aktivační funkce specifikovány jako $\varphi : \mathbb{R} \mapsto \mathbb{R}$, je zavedeno následující pravidlo k jejich jednoduchému využití pro vektory (dimenze n) a matice (dimenze $m \times n$).

$$\varphi\left(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix}\right) = \begin{bmatrix} \varphi(z_1) \\ \varphi(z_2) \\ \vdots \\ \varphi(z_n) \end{bmatrix} \tag{2.7}$$

$$\varphi(Z) = \varphi\left(\begin{bmatrix} z_{11} & z_{12} & \dots & z_{1,n} \\ z_{21} & \ddots & \dots & z_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ z_{m,1} & z_{m,2} & \dots & z_{m,n} \end{bmatrix}\right) = \begin{bmatrix} \varphi(z_{11}) & \varphi(z_{12}) & \dots & \varphi(z_{1,n}) \\ \varphi(z_{21}) & \ddots & \dots & z_{2,n}) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi(z_{m,1}) & \varphi(z_{m,2}) & \dots & \varphi(z_{m,n}) \end{bmatrix} \tag{2.8}$$

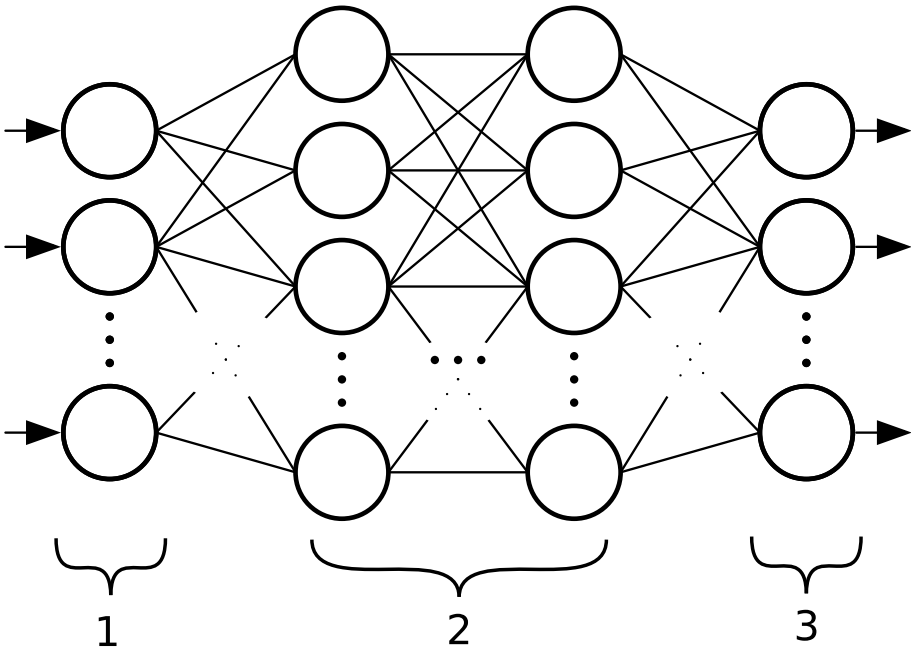
2.2 Husté neuronové sítě

Nechť vektor $\vec{Y} = \vec{Y}(\vec{x})$ je vektor požadovaných výstupních hodnot (správných hodnot) a vektor $\vec{x}$ je vektor vstupních hodnot modelu. Cílem FNN je co nejlépe najít vztah $\vec{y} = \vec{y}(\vec{x})$, který aproximuje funkci $\vec{Y}(\vec{x})$

$$\vec{y}(\vec{x}) \approx \vec{Y}(\vec{x}),$$

kde hodnoty $\vec{y}$ jsou hodnoty výstupní předpovědi modelu. Dále v této sekci bude provedena bližší úvaha o dimenzích vektorů $\vec{Y}, \vec{y}$ a $\vec{x}$.

Jak již bylo zmíněno, základní architekturou je FNN. Tato architektura je schematicky znázorněna na obr. 2.2.

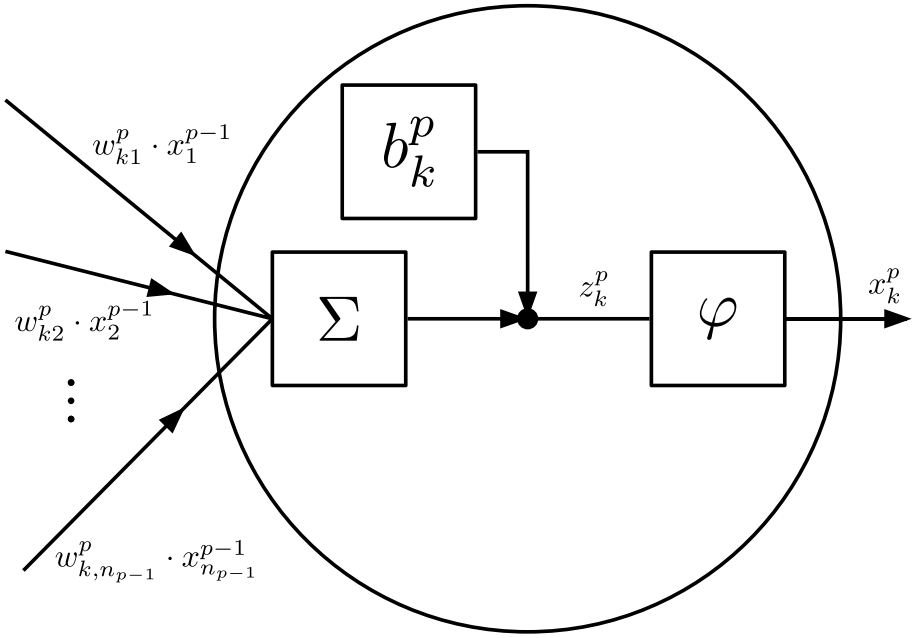


Obrázek 2.2: Schéma FNN

Na obr. 2.2 jednotlivé kružnice mají význam neuronů a úsečky představují jejich propojení. Jeden sloupec neuronů reprezentuje vrstvu sítě. Výstupem každého neuronu je pouze jedna skalární hodnota a propojení mezi neurony vyjadřuje jejich vzájemný vztah. Skupina 1 znázoňuje vstupní vrstvu, skupina 2 skryté vstvy a skupina 3 výstupní vrstvu. Nechť má neuronová síť celkem $N > 2$ vrstev, a nechť p-tá vrstva $p = \{\langle 1, N\rangle \cap \mathbb{N}\}$ obsahuje celkem n_p neuronů. Dle literatury [2] str. 11 je možné si k-tý

$$k = \{\langle 1, n_p\rangle \cap \mathbb{N}\}$$

neuron p-té vrstvy představit následovně (obr. 2.3), pokud je tato vrstva skrytou vrstvou.



Obrázek 2.3: Schéma k-tého neuronu p-té vrstvy

Vstupními hodnotami neuronu k-tého neuronu p-té vrstvy jsou hodnoty neuronů $x^{(p-1)} \in \mathbb{R}^{n_{(p-1)}}$ minulé vrstvy násobené vahami w^p_{ki}, které jsou reprezentovány propojením neuronů vrstvy $(p-1)$ a p-té vrstvy. Tyto váhy jsou jedním z parametrů neuronové sítě, který je nutný tréninkem optimalizovat. Neuron (k-tý, vrstvy p) v sobě nese další parametr - posunutí (bias) b^p_k, který je dalším cílem oprimalizace, a skalární tzv. aktivační funkci p-té vrstvy $\varphi_p(x) : \mathbb{R} \mapsto \mathbb{R}$ (viz sekce 2.1). Výstupní hodnota x^p_k neuronu se dá určit vztahy

$$z^p_k = \sum_{i=1}^{n_{(p-1)}} w^p_{ki} \cdot x^{(p-1)}_i + b^p_k \tag{2.9}$$

$$x^p_k = \varphi_p\left(z^p_k\right), \tag{2.10}$$

kde z^p_k je vážená vstupní hodnota neuronu, x^p_k je aktivační hodnota neuronu a n_p je počet neuronů p-té vrstvy [11]. Aktivační hodnoty neuronů p-té vrstvy lze potom zapsat vektorově jako

$$\vec{x}^p = \begin{bmatrix} x^p_1 \\ x^p_2 \\ \vdots \\ x^p_{n_p} \end{bmatrix} \in \mathbb{R}^{n_p}.$$

Je-li p-tá vrstva vstupní vrstvou, neobsahuje aktivační funkce a hodnoty x_k^p jsou přímo vstupními hodnotami modelu, což závisí na formulaci úlohy. V této práci je pro tyto hodnoty zavedeno označení bez horního indexu

$$x_k = x_k^p.$$

Skládá-li se daný model z celkem N vrstev, hodnoty x_k^N představují predikci (tedy hodnoty výstupní vrstvy) modelu a jsou dále označeny

$$y_k = x_k^N.$$

Výstupní vrstva může obsahovat aktivační funkce.

Při úvaze o neuronové síti, která má skalárních $m = n_1$ vstupů, neboli vstupní vektor $x \in \mathbb{R}^m$ a $n = n_N$ skalárních výstupů ($y \in \mathbb{R}^n$), lze takový model považovat za zobrazení

$$f : \mathbb{R}^m \mapsto \mathbb{R}^n.$$

Z indexových zápisů výše uvedených vyplývá maticový zápis pro výpočet hodnot neuronů p-té vrstvy

$$\vec{z}^p = \mathrm{W}^p \cdot \vec{x}^{p-1} + \vec{b}^p \tag{2.11}$$

$$\vec{x}^p = \varphi_p\left(\vec{z}^p\right), \tag{2.12}$$

kde matice $\mathrm{W}^p \in \mathbb{R}^{n_p} \times \mathbb{R}^{n_{p-1}}$ je složena z jednotlivých vah w_{ik}^p tak, že platí

$$\mathrm{W}^p = \begin{bmatrix} w_{11}^p & w_{12}^p & \dots & w_{1,n_{p-1}}^p \\ w_{21}^p & \ddots & \dots & w_{2,n_{p-1}}^p \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_p,1}^p & w_{n_p,2}^p & \dots & w_{n_p,n_{p-1}}^p \end{bmatrix} \tag{2.13}$$

a vektor $\vec{b}^p \in \mathbb{R}^{n_p}$ je vektor posunutí p-té vrstvy

$$\vec{b}^p = \begin{bmatrix} b_1^p \\ b_2^p \\ \vdots \\ b_{n_p}^p \end{bmatrix}. \tag{2.14}$$

Postupným aplikováním vztahu 2.12 a 2.11 pro $p \in \{\langle 2, N\rangle \cap \mathbb{N}\}$ lze projít přes všechny vrstvy až k té poslední a vyjde tak výsledná predikce $\vec{y}$ modelu. Je-li třeba konstruovat model bez aktivačních funkcí na výstupní vrstvě, může být položeno jednoduše

$$\varphi_N(\vec{x}) = \vec{x}.$$

Celkově se dá spočítat, že pro použití FNN je potřeba optimalizovat w_c vah a b_c posunutí, tedy $v_c = w_c + b_c$ parametrů, kde

$$w_c = \sum_{p=2}^{N} (n_p \cdot n_{p-1}), \tag{2.15}$$

$$b_c = \sum_{p=2}^{N} n_p. \tag{2.16}$$

Naskytá se tak otázka, jak přizpůsobit tyto parametry W^p a $\vec{b}^p$ tak, aby celkový výstup modelu byl co nejpřesnější. Pojem "co nejpřesnější" znamená najít parametry tak, aby hodnota chyby $\xi(\cdot,\cdot)$ (definována dále) byla minimální. Z pohledu chybové funkce je nutné brát v potaz, že výstupní vektor $\vec{y}$ je funkcí všech optimalizovaných parametrů a vstupní data se dají brát jako konstanty. Cílem totiž není měnit vstupní data, ale parametry modelu W^p a $\vec{b}^p$.

$$\vec{y} = \vec{y}(\vec{x}, \mathrm{W}^2, \mathrm{W}^3, \ldots, \mathrm{W}^N, \vec{b}^2, \vec{b}^3, \ldots, \vec{b}^N) \tag{2.17}$$

Nechť je dále skalární nezáporná funkce $\xi(\vec{Y}, \vec{y})$ [8] označena jako chybová funkce o dvou parametrech, která vrací nezáporné reálné číslo, přičemž každým z parametrů je vektor, tedy jde o zobrazení

$$\xi : (\mathbb{R}^n, \mathbb{R}^n) \mapsto \mathbb{R}^+$$

a nechť je dále označen požadovaný výstup modelu symbolem $\vec{Y} \in \mathbb{R}^n$. Existuje mnoho formulací chybových funkcí, nicméně v této práci je věnována pozornost funkci, která se v knihovně *PyTorch* [10] nazývá *MSELoss*. Tato funkce lze matematicky zapsat následovně.

$$\xi(\vec{Y}, \vec{y}) = \sum_{i=1}^{n} (Y_i - y_i)^2 \tag{2.18}$$

Je vidět, že výsledná hodnota funkce $\xi(\vec{Y}, \vec{y}) \geq 0$. Tato formulace se mimochodem rovná euklidovské normě $||\cdot||^2$ pro vektory.

2.3 Trénink hustých neuronových sítí

Je uvažováno označení všech paramterů k optimalizaci

$$\begin{gathered}\mathbf{W} = \{w^p_{ki} \text{ pro } \forall k, i;\ p \in \langle 2, N\rangle\} \\ \mathbf{b} = \{b^p_k \text{ pro } \forall k;\ p \in \langle 2, N\rangle\} \\ \vec{v} = \{\mathbf{W}_1, \ldots, \mathbf{W}_{w_c}, \mathbf{b}_1, \ldots, \mathbf{b}_{b_c}\},\end{gathered}$$

kde $\mathbf{W}$ je vektor všech parametrů w^p_{ij}, $\mathbf{b}$ je vektor parametrů b^p_i a $\vec{v}$ je vektor všech tréninkových parametrů, jehož dimenze je $\dim(\vec{v}) = v_c$.
Vzhledem k závislosti $\xi = \xi(\vec{Y}, \vec{y})$ a 2.17 bude dále funkce ξ pro účely tréninkových algoritmů uvažována jako funkce

$$\xi = \xi(\mathbf{W}, \mathbf{b}) = \xi(\vec{v}).$$

a tréninkem se rozumí pokusit se najít bod minima $min\,(\xi(\mathbf{W}, \mathbf{b}))$ vzhledem k těmto parametrům. Parametry $\mathbf{W}, \mathbf{b}$, při kterých ξ dosahuje své minimální hodnoty se dají považovat za optimalizované. Je velmi obtížné (v mnoha případech nemožné) dokázat, že nalezené minimum funkce ξ s vysokým počtem parametrů je globálním minimem, proto bude dále uspokojivé přiblížit se alespoň lokálnímu minimu. Pro tento účel je možné využít např. algoritmus zvaný *Gradient descent* [8] str. 16.

2.3.1 Gradient descent

Díky obecně vysokému množství parametrů v_c nelze vhodně uplatnit metody analytické matematiky pro nález minima funkce $\xi(\vec{v})$, jak se uvádí v [8] na str. 18. Proto se hledá minimum pomocí tohoho algoritmu, který funguje na principu využívání gradientu $\nabla\xi(\vec{v})$ k přizpůsobování parametrů $\vec{v}$ za účelem minimalizace hodnoty ξ. Diskrétní přírustek chybové funkce lze vyjádřit jako

$$\Delta\xi \approx \sum_{i=1}^{v_c} \frac{\partial\xi}{\partial v_i} \cdot \Delta v_i = \nabla\xi \cdot \Delta\vec{v}, \tag{2.19}$$

kde

$$\nabla\xi = \left[\frac{\partial\xi}{\partial v_1}, \frac{\partial\xi}{\partial v_2}, \ldots, \frac{\partial\xi}{\partial v_{v_c}}\right]^T \tag{2.20}$$

a $\Delta\vec{v}$ je přírustek vektoru $\vec{v}$.

Gradient $\nabla\xi$ v daném bodě $\vec{v}$ má význam směru nejvyššího růstu chybové funkce. Protože je účelem přiblížit se minimu chybové funkce, nejkratším směrem je $-\nabla\xi$ a proto se uvažuje přírustek $\Delta\vec{v}$ následovně:

$$\Delta\vec{v} = -\eta\nabla\xi, \tag{2.21}$$

přičemž parametr $\eta > 0$ se nazývá *learning rate*. Dosazením tohoto vztahu 2.21 do 2.19 lze odvodit, že přírustek $\Delta\xi$ je nekladný.

$$\Delta\xi \approx -\eta\nabla\xi \cdot \nabla\xi = -\eta \cdot ||\xi||^2 \leq 0$$

To ovšem neplatí vždy, neboť vztah 2.19 je pouze přibližný. Platil by přesně, pokud by se přírustek Δv_i nahradil diferenciálem dv_i, který je nekonečně malý. V diskrétním počtu není možné nekonečně malý přírustek realizovat, proto je potřeba volit parametr η dostatečně malý ($\eta \ll 1$), aby změna chybové funkce $\Delta\xi$ byla stále záporná. Velikost η samozřejmě ovlivňuje rychlost přizpůsobování parametrů, proto pro volbu příliš malého η k tomu dochází velmi pomalu a je potřeba provádět vysoký počet tréninkových cyklů. V praxi je možné tento problém řešit změnou parametru η během tréninku, jako například inicializovat s určitou hodnotou a během tréninku tuto hodnotu postupně snižovat, aby docházelo k postupnému zpřesňování parametrů $\vec{v}$. Tuto problematiku je vhodné testovat na konkrétním případu s konkrétními hodnotami η.

Přírustek $\Delta\vec{v}$ má význam

$$\Delta\vec{v} = \vec{v}_{new} - \vec{v}_{old},$$

kde, jak z označení vyplývá, $\vec{v}_{old}$ je vektor se starými hodnotami a $\vec{v}_{new}$ vektor s novými. Ze vztahu 2.21 se pak dají určit nové hodnoty

$$\vec{v}_{new} = \vec{v}_{old} - \eta\nabla\xi(\vec{v}_{old}). \tag{2.22}$$

Postupným iterováním je možné se přiblížit hodnotě lokálního minima funkce $\xi(\vec{v})$ pro každý ze vzorků. Jeden cyklus upravení parametrů může vypadat následovně.

Algorithm 1 Gradient descent - jeden cyklus

Require: : $\vec{v}_old$

1: Forward pass through neural net (for given sample) with parameters $\vec{v}_{old}$
2: Calculation of average value $\xi(\vec{v}_{old})$ for several samples
3: Calculation of gradient $\nabla\xi(\vec{v}_{old})$
4: Computation $\vec{v}_{new}$
5: Replace old values: $\vec{v}_{old} \leftarrow \vec{v}_{new}$

2.3.2 Backpropagation

Tyto úvahy přinášejí další otázku, jak najít hodnoty gradientu $\nabla\xi$. Odpovědí je algoritmus zvaný *Backpropagation* [2] str. 129, [8] str. 39. Tento algoritmus využívá pravidel o derivování složených funkcí k postupnému nalazení všech hodnot $\nabla\xi$. Jak z jeho názvu vyplývá, jednotlivé derivace jsou dopočítávány v opačném směru než probíhá průchod neuronovou sítí, tudíž se začíná výstupní vrstvou a postupuje se zpět celou architekturou modelu. V dále uvedených rovnicích bude zřejmé, že hodnoty vrstvy $p-1$ se dají dopočítat z hodnot vrstvy p.

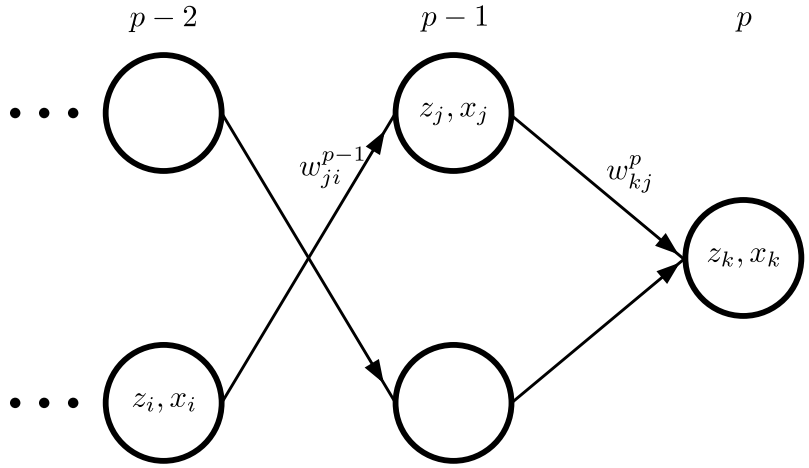
Backpropagation algoritmus spočívá ve zjišťování, jak jednotlivé změny vah w^p_{ki} a posunutí b^p_k ovlivňují celkovou hodnotu ξ, což znamená určení hodnot derivací

$$\frac{\partial\xi}{\partial w^p_{ki}},\ \frac{\partial\xi}{\partial b^p_k}, \text{nebo jen zkráceně } \frac{\partial\xi}{\partial v_i}.$$

Před samotným stanovením výpočtu gradientu $\nabla\xi$ je dle [8] str. 43 efektivní zavést Hadamardův operátor $\odot$, který dále umožní maticový zápis rovnic pro tento algoritmus. Je tedy pro libovolné vektory $\vec{p}$ a $\vec{q}$ stejné dimenze n definován vztah

$$\vec{p}\odot\vec{q} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{bmatrix} \odot \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} = \begin{bmatrix} p_1 \cdot q_1 \\ p_2 \cdot q_2 \\ \vdots \\ p_n \cdot q_n \end{bmatrix}. \tag{2.23}$$

Například v knihovně *PyTorch* je řešena tato operace pomocí funkce `torch.mul`$(\vec{p},\vec{q})$ [10]. Dále pro představu je z celé neuronové sítě vyjmuta pouze její část, která je zjednodušeně zobrazena na následujícím obrázku 2.4.



Obrázek 2.4: Část neuronové sítě

Indexem k jsou značeny neurony p-té vrstvy, indexem j neurony vrstvy $(p-1)$ a indexem i neurony vrstvy $(p-2)$. Parciální derivace neboli citlivost chybové funkce na parametr w_{kj}^p by vypadala takto:

$$\frac{\partial \xi}{\partial w_{kj}^p} = \underbrace{\frac{\partial \xi}{\partial x_k^p} \cdot \frac{\partial x_k^p}{\partial z_k^p}}_{\delta_k^p} \cdot \frac{\partial z_k^p}{\partial w_{kj}^p} = \delta_k^p \cdot x_j^{p-1}. \tag{2.24}$$

Z ronice 2.9 (po přejmenování sčítacího indexu) je snadné vidět, že

$$\frac{\partial z_k^p}{\partial w_{kj}^p} = x_j^{p-1}.$$

Je použito značení hodnoty δ_k^p, která je použita v tzv. δ-pravidle [11] str. 35. Veličina δ_k^p výrazně usnadňuje dopočítávání derivací chybové funkce i podle parametrů z vrstev předcházející p-té, což bude ukázáno dále. Dá se odvodit, že rovnici 2.24 je možné aplikovat pro $\forall p \in \langle 2, N\rangle \cap \mathbb{N}$ při znalosti příslušného δ_k^p.

$$\delta_k^p = \frac{\partial \xi}{\partial z_k^p} = \frac{\partial \xi}{\partial x_k^p} \cdot \frac{\partial x_k^p}{\partial z_k^p} = \frac{\partial \xi}{\partial x_k^p} \cdot \varphi'(z_k^p) \tag{2.25}$$

V rovnici 2.25 se objevuje derivace aktivační funkce, o kterých byla zmínka v předchozí kapitole. Veličinu δ_k^p jde vnímat jako k-tou složku vektoru δ^p a v literatuře [8] str. 45 je uveden elegantní maticový zápis pomocí výše zavedeného operátoru pro vyjádření tohoto vektoru, kde operátor $\nabla_{x,p}$ představuje zjednodušený zápis pro derivaci chybové funkce podle aktivačních hodnot pomocí gradientu.

$$\nabla_{x,p}(\xi) = \left[\frac{\partial \xi}{\partial x_1^p}, \ldots, \frac{\partial \xi}{\partial x_{n_p}^p}\right]^T$$

$$\delta^p = \nabla_{x,p}(\xi) \odot \varphi'(\vec{z}^p) \tag{2.26}$$

Za zmínku stojí, že rozměr vektoru δ^p se rovná počtu neuronů p-té vrstvy, tedy n_p. Obdobným postupem se dá spočítat i derivace chybové funkce podle parametru posunutí p-té vrstvy (analogicky jako v rovnici 2.24) a dá se opět odvodit, že toto platí pro $\forall p \in \langle 2, N\rangle \cap \mathbb{N}$.

$$\frac{\partial \xi}{\partial b_k^p} = \delta_k^p \tag{2.27}$$

Nyní je na místě se zaměřit na derivace podle parametrů předchozí vrsty $p-1$. Znovu v návaznosti na schematický obrázek 2.4 je cílem vyjádřit $\partial\xi/\partial w_{ji}^{p-1}$.

$$\frac{\partial \xi}{\partial w_{ji}^{p-1}} = \underbrace{\frac{\partial \xi}{\partial x_j^{p-1}} \cdot \frac{\partial x_j^{p-1}}{\partial z_j^{p-1}}}_{\delta_j^{p-1}} \cdot \underbrace{\frac{\partial z_j^{p-1}}{\partial w_{ji}^{p-1}}}_{x_i^{p-2}} = \delta_j^{p-1} \cdot x_i^{p-2} \tag{2.28}$$

Analogicky k rovnici 2.25 se první dva činitelé dohromady dají označit jako δ_j^{p-1} a opět z rovnice 2.9 se třetí činitel rovná x_i^{p-2}. Lze si všimnout, že tento vztah je v souladu se vztahem 2.24, pouze došlo k posunutí indexů. Zaměřením na člen δ_j^{p-1} a rozepsáním derivace podle x_j^{p-1} pomocí pravidel o derivování složené funkce vyplynou další podstatné souvislosti. Jsou k tomu využity výše již uvedené vztahy a pro přehlednost vyznačeny již známé členy.

$$\delta_j^{p-1} = \left(\underbrace{\frac{\partial \xi}{\partial x_k^p} \cdot \frac{\partial x_k^p}{\partial z_k^p}}_{\delta_k^p} \cdot \underbrace{\frac{\partial z_k^p}{\partial x_j^{p-1}}}_{w_{kj}^p} \right) \cdot \underbrace{\frac{\partial x_j^{p-1}}{\partial z_j^{p-1}}}_{\varphi'(z_j^{p-1})} \tag{2.29}$$

Násobení mezi závorkou a čtvrtým činitelem není myšleno jako skalární součin. Pro získání j-té složky vektoru δ^{p-1} je potřeba mezi sebou vynásobit j-té složky dvou násobených vektorů přesně tak, jak je definován operátor $\odot$. Po těchto úvahách je možné vztah 2.29 zapsat maticově:

$$\delta^{p-1} = \left((W^p)^T \cdot \delta^p \right) \odot \varphi'(\vec{z}^{p-1}). \tag{2.30}$$

Horní index ^{T} značí transpozici matice. Tento vztah dává informaci, jak se efektivně přenést k hodnotám δ^{p-1} minulé vrstvy, proto je potřeba při tomto algoritmu postupovat po zpátku a postupně propagovat hodnoty vázané na poslední vrstvu N až k té první (resp. druhé, protože první vrstva je pouze vstupní a nenese žádné parametry pro optimalizaci).

Shrnutí: Výše jsou odvozeny 4 důležité rovnice (2.26, 2.30, 2.24, 2.27) pro realizaci celého algoritmu. Pro přehled jsou zde uvedeny pohromadě:

$$\begin{aligned}
\delta^p &= \nabla_{x,p}(\xi) \odot \varphi'(\vec{z}^p) \\
\delta^{p-1} &= \left((W^p)^T \cdot \delta^p \right) \odot \varphi'(\vec{z}^{p-1}) \\
\frac{\partial \xi}{\partial w_{kj}^p} &= \delta_k^p \cdot x_j^{p-1} \\
\frac{\partial \xi}{\partial b_k^p} &= \delta_k^p
\end{aligned}$$

Ve zkratce první rovnice poskytuje informaci o tom, jak dopočítat pomocné hodnoty δ^p, druhá rovnice, jak se z nich dostat k hodnotám δ^{p-1}. Třetí a čvrtá rovnice říkají, jak z dopočteného příslušného δ^p dostat členy gradientu $\nabla\xi$, který je následně použit pro *Gradient descent* (alg. 1). Všechny tyto rovnice byly odvozeny obecně a platí pro $\forall p \in \langle 2, N \rangle \cap \mathbb{N}$, kromě druhé rovnice, která pro $p = 2$ nedává smysl, protože je zbytečné počítat pomocné hodnoty δ^1.

2.3.3 Rozšíření pro více vzorků

Dosavadní úvahy s sebou však nesou značné omezení. Výše definované vztahy a popsané algoritmy umožňují práci pouze s jedním vzorkem datového souboru. V této pozici je možné vyhodnotit gradient $\nabla\xi$ a provést *Gradient descent* algoritmus, tedy přizpůsobit parametry neuronové sítě jednomu jedinému vzorku.

Proto je uvažována skupina s vzorků. Hodnota chybové funkce l-tého vzorku je vyjádřena v souladu s obecným vztahem 2.18, tedy

$$\xi_l(\vec{Y}^{\,l}, \vec{y}^{\,l}) = \sum_{i=1}^{n} (Y_i^l - y_i^l)^2 = ||\vec{Y}^{\,l} - \vec{y}^{\,l}||^2, \tag{2.31}$$

kde $\vec{Y}^{\,l}$ jsou požadované hodnoty l-tého vzorku a $\vec{y}^{\,l}$ predikované hodnoty pro l-tý vzorek. K realizaci algoritmu je dále potřeba zavést průměrnou hodnotu chybové funkce přes s vzorků

$$\hat{\xi_s} = \frac{1}{s}\sum_{l=1}^{s}\xi_l(\vec{Y}^{\,l}, \vec{y}^{\,l}), \tag{2.32}$$

aby finální přizpůsobení parametrů $\vec{v}$ mělo co nejpříznivější vliv na všech s vzorků. Správné hodnoty $\vec{Y}^{\,l}$ jsou poskytnuté a z hlediska tréninku jsou brány za konstantní.

Díky této definici se zmíněné algoritmy dají rozšířit. *Gradient descent* navíc musí určit $\hat{\xi_s}$ pro s vzorků a algoritmus *Backpropagation* s ní pak může pracovat (namísto ξ pro jeden vzorek). Dojde tak k nalezení gradientu $\nabla\hat{\xi_s}$, který se dá použít v *Gradient descent* k upravení trénovatelných parametrů takto:

$$\vec{v}_{new} = \vec{v}_{old} - \eta\nabla\hat{\xi_s}. \tag{2.33}$$

Přístup k tréninku, který volí počet vzorků skupiny $s = 1$ je např. v [2] str. 128 pojmenován jako *on-line learning*. Po přizpůsobení parametrů $\vec{v}$ danému vzorku se posune na další a takto pokračuje dokud neprojde celou sadu. Jeden průchod celou tréninkovou množinou se nazývá *epoch*.

V praxi se však častěji provádí tréninky metodou, která spočívá v přizpůsobování vah w^p_{ki} a posunutí b^p_k modelu po vyhodnocení skupiny s vzorků, kde $1 < s \leq S$ a S je počet vzorků v celé datové sadě. Průměrnou chybu přes tyto vybrané vzorky lze určit pomocí vztahu 2.32. Metoda výpočtu gradientu, která se nazývá *Stochastic Gradient descent* [8] str. 22 spočívá v náhodném výběru s vzorků ze souboru dat. Hodnota s se podle literatury uvedené v minulé větě nazývá *mini-batch*, ale v této práci v souladu s knihovnou *PyTorch* [10] bude dále nazývána *batch size*. Průměrné hodnoty gradientu chybové funkce z náhodného výběru potom aproximují průměrné hodnoty gradientu chyby celé sady

$$\nabla\hat{\xi} \approx \frac{1}{s}\sum_{l=1}^{s}\nabla\xi_l = \nabla\hat{\xi_s}, \tag{2.34}$$

kde $\nabla\xi_l$ je gradient l-tého vzorku náhodného výberu, $\nabla\hat{\xi}$ je průměrný chybový gradient celé sady a $\nabla\hat{\xi_s}$ souvisí pouze s náhodným výběrem s vzorků. Se znalostí hodnoty $\Delta\hat{\xi_s}$ je teprve provedeno přizpůsobení vah. Po realizaci tohoto kroku se vybraných s vzorků a vyřadí z možnosti výběru, provede se nový výběr a další krok algoritmu *Gradient descent*. Proces se opakuje, dokud nedojde k vyčerpání celé množiny vzorků, což znamená provedení jednoho epochu. Tento přístup urychluje proces tréninku a umožňuje paralelizaci.

Samozřejmě, že nejoptimálnějším směrem přizpůsobení optimalizovaných parametrů by byla hodnota $\nabla\hat{\xi}$ ($s = S$), ale to s sebou nese několik komplikací, jako např. rychlost výpočtu $\nabla\hat{\xi}$ či vysoké nároky na místo v paměti. Proto se častěji provádí více "téměř" optimálních kroků v aktuálním směru $\nabla\hat{\xi_s}$.

Volba tréninkového přístupu a hodnoty s opět závisí na konkrétním problému a často se musí ověřovat experimentálně. Na závěr této sekce je vhodné uvést, že knihovny jako např. *PyTorch* [10] mají algoritmy jako *Gradient descent* nebo *Backpropagation* již implemtované a připravené ke konkrétnímu použití. Vedle *Stochastic Gradient descent* existují i další, složitější, optimalizační procesy trénovatelných parametrů. Velmi populární je např. algoritmus *ADAM* [10], který je používán v této práci.

2.4 Konvoluční neuronové sítě

V této sekci je nastíněna základní myšlenka CNN. Architektura CNN kromě hustých vrstev zmíněných v minulé sekci navíc obsahuje, jak již z názvu vyplývá, konvoluční vrstvy. Jejich silnou stránkou je rozpoznávání určitých vzorů v datech a důsledkem toho mohou pak v určitých situacích přesněji předpovídat výsledky. Poté, co konvoluční vrstvy vybraný vzor v rámci nějaké lokální oblasti identifikují, jeho absolutní poloha už není pro výslednou predikci tolik důležitá. Toto je velký rozdíl od klasických hustých vrstev, které daná data sledují absolutně. Pro lepší náhled je možné si představit příklad rozpoznávání objektů na 2D obrazu, přičemž každý vstupní neuron husté vrstvy reprezentuje jeden konkrétní pixel. Při posunutí hledaného objektu o několik pixelů již nebude hustá vrstva schopna spolehlivě objekt poznat.

Pro tuto práci je dále problematika CNN omezena pouze na 2D případy, čímž jsou myšlena data, která se dají reprezentovat jako 2D objekty. Příkladem může být klasifikace plošných obrazů nebo také řešení numerických problémů ve 2D.

Nejdůležitějšími elementy CNN jsou konvoluční vrstvy a podvzorkovací vrstvy, nazývané jako *pooling* nebo *subsampling layers* [2] str. 201. Obě tyto vrstvy jsou charakterizovány jednotlivými filtry (v literatuře bývá mimo jiné uváděn pojem *kernel* [12] str. 6). Filtr si lze představit jako matici $K : \mathbb{R}^{\kappa_1} \times \mathbb{R}^{\kappa_2}$, ale pro účel této práce postačí předpoklad $\kappa_1 = \kappa_2 = \kappa$, kde parametr κ je rozměrem "čtvercového" filtru, neboli *kernel size* [10]. Dále je stanoven předpoklad, že vstupní data lze reprezentovat maticí $X : \mathbb{R}^{\mu} \times \mathbb{R}^{\mu}$ a $\mu > \kappa$. Čtverová matice hodnot p-té vrstvy bude značena X^p s rozměrem d_p ($d_1 = \mu$), $X^p \in \mathbb{R}^{d_p} \times \mathbb{R}^{d_p}$.

2.4.1 Konvoluční vrstvy

Na obrázku 2.5 je pro představu znázorněn filter K^{p+1} konvoluční vrstvy $(p+1)$ o rozměru $\kappa_{p+1} = 3$ a matice dat X^p o rozměru $d_p = 5$.

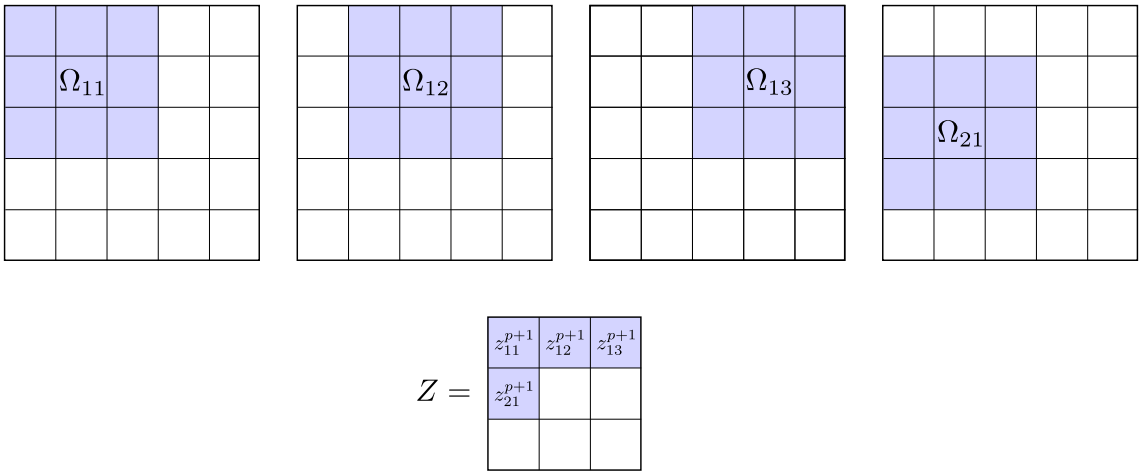
$$\begin{array}{|c|c|c|c|c|}\hline x_{11}^p & x_{12}^p & x_{13}^p & x_{14}^p & x_{15}^p \\ \hline x_{21}^p & x_{22}^p & x_{23}^p & x_{24}^p & x_{25}^p \\ \hline x_{31}^p & x_{32}^p & x_{33}^p & x_{34}^p & x_{35}^p \\ \hline x_{41}^p & x_{42}^p & x_{43}^p & x_{44}^p & x_{45}^p \\ \hline x_{51}^p & x_{52}^p & x_{53}^p & x_{54}^p & x_{55}^p \\ \hline \end{array} \quad * \quad \begin{array}{|c|c|c|}\hline w_{11}^{p+1} & w_{12}^{p+1} & w_{13}^{p+1} \\ \hline w_{21}^{p+1} & w_{22}^{p+1} & w_{23}^{p+1} \\ \hline w_{31}^{p+1} & w_{32}^{p+1} & w_{33}^{p+1} \\ \hline \end{array}$$

Obrázek 2.5: Aplikace filtru na data

Na barevně zvýrazněnou část (ozn. submatice Ω_{kl}) matice X^p je aplikován filtr. Aplikace se realizuje pomocí operace konvolučního součinu ($*$). Tato operace funguje pro matice stejných dimenzí následovně:

$$A * B = \sum_{i=1}^{d} \sum_{j=1}^{d} a_{ij} \cdot b_{ij} \in \mathbb{R}, \tag{2.35}$$

kde d je rozměr čtvercové matice $A : \mathbb{R}^d \times \mathbb{R}^d$. Výsledkem konvolučního součinu je reálné číslo. V kontextu aplikace filtru na matici dat X^p má dále konvoluční součin takový význam, že výsledkem je čtvercová matice rozměru d_{p+1} složená z jednotlivých konvolučních součinů získaných "posouváním" (po řádcích) submatice Ω_{kl} tak, aby celá Ω_{kl} byla v matici X^p obsažena (proces vysvětlen také v [13] str. 326). Schematicky je tento proces znázorněn na obrázku 2.6.



Obrázek 2.6: Posouvání submatice Ω_{kl}

Myšlenka s volením submatice Ω_{kl} je ilustrativní a je určená pro pochopení principu. Formálně se tento proces dá zapsat následujícími vztahy.

$$\tilde{Z}^{p+1} = \left\{\tilde{z}_{kl}^{p+1}\right\} = \sum_{i=1}^{\kappa_{p+1}} \sum_{j=1}^{\kappa_{p+1}} x_{k+i-1,l+j-1}^{p} \cdot w_{ij}^{p+1} \tag{2.36}$$

$$Z^{p+1} = \tilde{Z}^{p+1} + b^{p+1}, \tag{2.37}$$

kde $\forall k, l \in \langle 1, d_{p+1}\rangle \cap \mathbb{N}$ a $b^{p+1} \in \mathbb{R}$ je posunutí (*bias*) související s příslušným filtrem. O rozměru d_{p+1} je provedena dále úvaha. Parametry filtru K^p, tj. váhy w_{ij}^p a posunutí b^p, jsou parametry určené pro optimalizaci pomocí tréninku a mají vliv na přesnost predikce modelu. Pro získání aktivačních hodnot další vrstvy je obecně potřeba znovu aplikovat příslušnou aktivační funkci tak, jako v případě hustých vrstev.

$$X^{p+1} = \varphi^{p+1}(Z^{p+1}) \tag{2.38}$$

Na obrázku 2.6 lze vidět, že vždy dochází k posunutí oblasti Ω_{kl} o jeden element a děje se tak v obou směrech. Tento parametr, který je pro celou konvoluční vrstvu konstantní, se nazývá podle [10] *stride* (dále označen jako χ) a podílí se na rozměru výstupní matice Z^{p+1}. Rovnice 2.36 platí za předpokladu $\chi = 1$, ale je možné ji po několika jednoduchých úvahách rozšířit i pro obecné χ. Dalším parametrem hrajícím roli na rozměru výstupním matice je odsazení $\psi \in \mathbb{N}_0$, neboli dle [10] *padding*. Tento parametr se podílí na změně rozměrů vstupní matice X^p a má význam přidání ψ hodnot (např. 0) na začátek a konec každého sloupce, řádku a každé hlavní diagonály (blíže popsáno v [12], nebo přímo v dokumentaci *PyTorch* [10]). Toto se podepíše i na rozměru d_{p+1}. Nyní po vysvětlení základních parametrů κ, χ, ψ je možné diskutovat o dimenzi d_{p+1}, pro kterou platí vztah viz. [12]

$$d_{p+1} = \left[\frac{d_p + 2\psi - \kappa_p}{\chi}\right] + 1. \tag{2.39}$$

Je vhodné zmínit, že kombinace parametrů $\psi = 1, \chi = 1, \kappa = 3$ zaručí stejný rozměr vstupní i výstupní matice, což má také význam v této práci.

Dosud je uvažováno, že vstupem i výstupem konvoluční vrsty je pouze jedna matice. Každá konvoluční vrstva (v tomto případě p-tá) však může mít více vstupních kanálů $I^p \in \mathbb{R}$ a výstupních kanálů $O^p \in \mathbb{R}$ - v literatuře označováno jako *channels* nebo *features* [12]. Aby na sebe mohly vrstvy spojitě navazovat, musí platit

$$O^p = I^{p+1}.$$

Množství výstupních kanálů I^{p+1} vznikne aplikováním I^{p+1} filtrů na vstupní data vrstvy. Díky více filtrům je možné v datech hledat více vlastností. V případě $p = 1$ závisí I^p na definici úlohy. V případě, že $I^{p+1} > 1, O^{p+1} > 1$, přibyde v rovnicích 2.36, 2.37 a 2.38 suma přes vstupní kanály (sčítací index c) a index r výstupního kanálu následovně

$$\tilde{Z}^{p+1,r} = \left\{\tilde{z}_{kl}^{p+1,r}\right\} = \sum_{c=1}^{O^p} \sum_{i=1}^{\kappa_{p+1}} \sum_{j=1}^{\kappa_{p+1}} x_{k+i-1,l+j-1}^{p,c} \cdot w_{ij}^{p+1,r} \tag{2.40}$$

$$Z^{p+1,r} = \tilde{Z}^{p+1,r} + b^{p+1,r} \tag{2.41}$$

$$X^{p+1,r} = \varphi^{p+1}(Z^{p+1,r}), \tag{2.42}$$

kde $x_{k+i-1,l+j-1}^{p,c}$ značí hodnoty matice $X^{p,c}$ c-tého kanálu v p-té vrstvě a index r reprezentuje veličiny spojené s r-tým výstupním kanálem ($r \in \langle 1, O^{p+1}\rangle$). Je důležité zmínit, že rovnice 2.40 je opět omezená pro $\chi = 1$. Popsané vztahy poskytují přehled, jak se v modelu uplatňuje konvoluční vrstva. Vzhledem k množství indexů je při programování vhodné pracovat s vícedimenzionálními daty, což neuronové sítě oproti jiným algoritmům umožňují. Pro tyto potřeby je např. v knihovně *PyTorch* [10] zavedena datová struktura zvaná *tensor*.

2.4.2 Podvzorkovací vrstvy

Podvzorkovací vrstvy slouží k vytažení důležitých hodnot ze vstupní matice a tím snižují její rozměr. Fungují velmi podobně jako konvoluční vrstvy z minulé sekce. Princip posouvání submatice Ω_{kl}, kde Ω_{kl} má stejný rozměr (κ_p) jako příslušný aplikovaný filtr, je použit stejně. Liší se však tím, že tyto vrstvy neprovádí operaci konvoluce mezi filtrem K^p a vybranou maticí Ω_{kl}, tudíž neobsahují parametry k tréninku w_{ij}^p a b^p. I přesto výstupem aplikace podvzrkovacího filtru na danou submatici je skalár. Pro tuto operaci se ve většině případů jednotlivé submatice Ω_{kl} nepřekrývají, jsou disjunktní a

$$\bigcup_{k,l} \Omega_{kl} = X^p$$

viz [12] str. 18. Tento jev se realizuje nastavením parametrů $\chi^p = \kappa^p$ a např. při volbě $\chi^p = \kappa^p = 2$ a za předpokladu, že d^{p-1} je sudé číslo, bude platit $d^{p-1} = 2 \cdot d^p$ (výstupní čtvercová matice má poloviční rozměr než vstupní). Nejpoužívanější podvzorkovací vrstva se nazývá *Max Pool* viz. [13] str. 335, jejíž filtry jednoduše vrátí maximální

hodnotu submatice Ω_{kl}, na kterou je daný filtr použit. Existují i další, např. *Min Pool* nebo *Average Pool* viz. [10]. Aritmetika těchto vrstev funguje stejně jako u konvolučních vrstev.

2.4.3 Transponované konvoluční vrstvy

V této práci jsou dále v CNN použity transponované konvoluční vrstvy. Tyto vrstvy fungují přesně opačně oproti konvolučním vrstvám. Základní princip si lze představit jako aplikování nějakého filtru obsahujícího parametry (váhy a posunutí) na skalární hodnotu. Zde není potřeba používat operaci konvoluce, ale stačí jen vynásobit matici filtru danou skalární hodnotou (a případně přičíst posunutí) a výsledkem je výše používaná submatice Ω_{kl}. Prováděním tohoto postupu na prvky vstupní matice dojde sjednocením submatic Ω_{kl} k vytvoření nové výstupní matice o větším rozměru. Tuto vrstvu je vhodné použít, pokud mimo hledání vzorů v datech je potřeba také zvětšit jejich rozměr. Aritmetiku těchto vrstev lze dohledat v literatuře [12].

2.4.4 Trénink konvolučních sítí

Trénink konvolučních vrstev probíhá velmi podobně, jako trénink hustých vrstev. Je snadné odvodit, že p-tá konvoluční obsahuje celkem $v^p_{c,conv}$ parametrů k tréninku, kde

$$v^p_{c,conv} = O^p \cdot (\kappa_p^2 + 1). \tag{2.43}$$

Je potřeba znovu definovat vektor $\vec{v}$ všech parametrů k optimalizaci, stejným způsobem jako v sekci 2.3. Nechť má neuronová síť celkem N_c konvolučních vrstev a N_d hustých vrstev. Potom v takovém případě je cílem přizpůsobit v_c parametrů, přičemž

$$v_c = N_c \cdot v_{c,conv} + \sum_{p=2}^{N_d} n_p \cdot (n_{p-1} + 1). \tag{2.44}$$

Pro tento účel je možné opět použít algoritmus *Gradient Descent* (alg. 1). Hlavní odlišností v tréninku konvolučních sítí je způsob nalezení vektoru

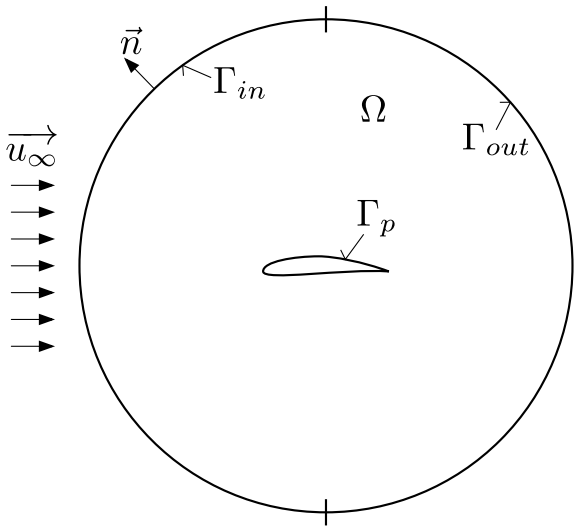
$$\nabla\xi = \left[\frac{\partial\xi}{\partial v_1}, \ldots, \frac{\partial\xi}{\partial v_c}\right]^T,$$

čehož se dá dosáhnout pomocí algoritmu *Backpropagation*. Pro konvoluční vrstvy funguje velmi podobně jako v případě hustých vrstev na principu derivací složených funkcí a postupného derivování od poslední vrstvy. Detailnější vysvětlení přesahuje rozsah této práce a dá se dohledat např. v literatuře [14], [15].

Kapitola 3

Formulace úlohy a numerické výpočty

Účelem této práce je řešit 2D úlohu obtékání profilu pomocí neuronové sítě. Konkrétně se jedná o úlohu nalezení řešení (rychlosti $\vec{u}$ a tlaku[1] $\tilde{p}$) stacionárního turbulentního proudění nestlačitelné tekutiny popsaného Navier-Stokesovými rovnicemi. Vzorová data s očekávanými výsledky jsou generována pomocí softwaru *OpenFoam* [16], tedy data získaná tímto řešičem jsou z pohledu neuronové sítě považována za správná.



Obrázek 3.1: Schéma řešené úlohy

Na obrázku 3.1 je schematicky načrtnut řešený problém. Proudové pole se řeší v oblasti Ω, která je ohraničena hranicemi Γ_{in}, Γ_{out} a Γ_p s tím, že sjednocením $\Gamma_{in} \cup \Gamma_{out}$ vznikne uzavřená vnější hranice oblasti a tyto části jsou navzájem disjunktní $\Gamma_{in} \cap \Gamma_{out} = \emptyset$. Vstupní a výstupní část hranice se dá definovat tak, že při uvažování vnější normály $\vec{n}$ na Γ_{in} platí:

$$\vec{u}_\infty \cdot \vec{n} < 0$$

a na Γ_{out}:

$$\vec{u}_\infty \cdot \vec{n} > 0,$$

[1] Tlakem je v této práci rozuměn tzv. kinematický tlak, který se rovná skutečnému tlaku děleném hustotou

kde vektor $\vec{u}_\infty$ je vektor rychlosti v dostatečné vzdálenosti od obtékané geometrie. Jelikož je uvažován stacionární problém, počáteční podmínky nemají na ustálené řešení vliv. Nastaveny jsou na hodnoty $\vec{u} = \vec{u}_\infty, p = p_\infty$. Na hranicích jsou okrajové podmínky předepsány následovně.

$$\Gamma_{in} : \begin{aligned} \vec{u} &= \vec{u}_\infty \\ \frac{\partial \tilde{p}}{\partial n} &= 0 \end{aligned} \qquad \Gamma_{out} : \begin{aligned} \frac{\partial \vec{u}}{\partial n} &= 0 \\ \tilde{p} &= \tilde{p}_\infty \end{aligned} \qquad \Gamma_p : \begin{aligned} \vec{u} &= 0 \\ \frac{\partial \tilde{p}}{\partial n} &= 0 \end{aligned}$$

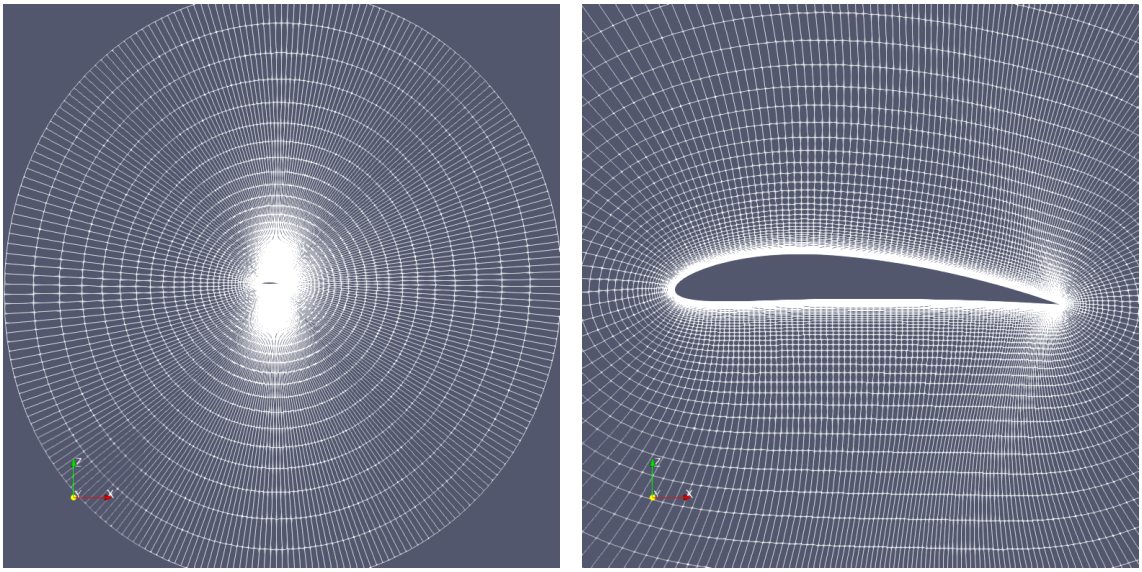
Veličina p_∞ představuje tlak ve vzdáleném okolí. Pro tuto práci jsou voleny okrajové hodnoty

$$\begin{aligned} \vec{u}_\infty &= [10, 0]^T , \\ \tilde{p}_\infty &= 0. \end{aligned}$$

V praxi v *OpenFoam* jsou podmínky na Γ_{in} a Γ_{out} nastaveny jako tzv. *freestream* viz. [16], které kombinují Neumannovy a Dirichletovy okrajové podmínky. Na hranici profilu Γ_p je nastavena tzv. *noSlip* podmínka pro rychlost a *zeroGradient* pro tlak.

Stacionární případ může být řešen pomocí algoritmu *SIMPLE* [17], který je v *OpenFoam* implementovaný v řešiči *simpleFoam*. Pro řešení turbulence je použit *Spalartův-Allmarasův* [18] model. Počáteční hodnota pro parametry turbulence ν_t a $\tilde{\nu}$ je 0.14. K řešení proudového pole v mezní vrstvě je aplikována stěnová funkce *nutSpaldingWallFunction* v OpenFoam. Uvažovaný režim proudění odpovídá Reynoldsovu číslu $Re = 10^6$.

Výpočetní síť (O-síť), uvažována s jednotkovou délkou tětivy profilu, je generována pomocí softwaru *construct2d* [19]. Jedná se o software pro tvorbu kvalitních hyperbolických výpočetních sítí pro případy řešení obtékání profilu. Na obrázku 3.2 je zobrazena vzorová výpočetní síť. Na levém obrázku 3.2a má oproti jednotkovému profilu síťovaná oblast poloměr 15 (tzv. *farfield radius*). Taková síť obsahuje celkem 24651 buněk. Všechny použité sítě pro různé profily jsou tvořeny se stejným nastavením v softwaru, tudíž počet buněk se vždy pohybuje kolem hodnoty 25000.



(a) O-síť **(b)** Detail sítě

Obrázek 3.2: Výpočetní síť (NACA 5412, úhel náběhu 2°)

Jednotlivé geometrie profilů jsou normované a mají jednotkovou délku, což umožňuje případnou škálovatelnost úlohy. Bylo otestováno, že k dosažení rezidua menšího než 10^{-6} pro výše uvedené nastavení postačí 500 iterací.

Veškeré ovládání a nastavení softwarů *OpenFoam*, *construct2d* a manipulace se soubory byla automatizována pomocí skriptů v Pythonu pro možnost generování souboru dat - "správných" výsledků k jednotlivým případům.

Kapitola 4

Aplikace neuronových sítí na řešení problému

Tato kapitola obsahuje popis tvorby modelu neuronové sítě, metodiku zpracování dat a vysvětlení všech potřebných algoritmů pro jejich úpravu. Všechny potřebné skripty jsou psány v jazyce Python a ke tvorbě neuronové sítě je použita knihovna PyTorch [10]. Důvodem použití této knihovny je možnost provádění tréninku na grafické kartě, její dobře zpracované rozhraní pro jazyk Python a možnost mít jako uživatel kontrolu např. nad tréninkovým cyklem.

Jak již bylo zmíněno v předešlých kapitolách, pro řešení CFD simulace obtékání profilu je v této práci použita konvoluční neuronová síť a architektura U-net. V sekci 2.4 bylo zmíněno, že pro účely této práce konvoluční síť pracuje s daty, které jsou reprezentována čtvercovou maticí. Proto je v prvním kroce nutné formulovat danou úlohu pro neuronovou síť.

4.1 Formulace úlohy pro neuronovou síť

Při numerických výpočtech je získáván výsledek na základě dané vstupní geometrie, okrajových a počátečních podmínek. Účelem je takto připravit problém i v případě neuronové sítě, která má být efektivním řešičem tlakového pole Navier-Stokesových rovnic pro různé geometrie profilu, čímž jsou míněny dva parametry: tvar profilu a úhel natočení α. Pro všechny predikce neuronové sítě jsou však okrajové a počáteční podmínky konstantní.

V minulé kapitole bylo zmíněno, že všechny profily jsou normované, mají tedy jednotkovou délku a jejich průmět na osu x leží v intervalu $x \in \langle 0, 1\rangle$. Geometrie profilu je definována skupinou g bodů $\vec{G} = [G_1, G_2, \ldots, G_g]$ v tzv. *Selig* [20] formátu, který obsahuje souřadnice seřazených bodů $G_i = G_i(x_i, y_i)$ hranice profilu. Body jsou seřazeny "proti směru hodinových ručiček" tak, že první bod leží na horní straně odtokové hrany. Vstupní i výstupní data jsou reprezentována čtvercovým polem $q \times q$ bodů $P(q)$ s konstantním rozestupem, které překrývá danou geometrii a blízké okolí profilu. Konkrétně jsou voleny body, které leží ve čtvercové oblasti $\langle -0.5, 1.5\rangle \times \langle -1, 1\rangle$ a jsou v rámci jednoho modelu voleny konstantně. Jako vstupní data jsou v těchto bodech uvažovány normálové vzdálenosti od hranice profilu (transformované vhodným vztahem, viz dále) a výstupní data z modelu mají reprezentovat informaci o hodnotě tlaku (resp. koeficientu tlaku) v každém z těchto bodů. Díky této volbě je pak možné vstupní data zapsat maticí $\mathbf{I} \in \mathbb{R}^q \times \mathbb{R}^q$ a výstupní data maticí $\mathbf{O} \in$

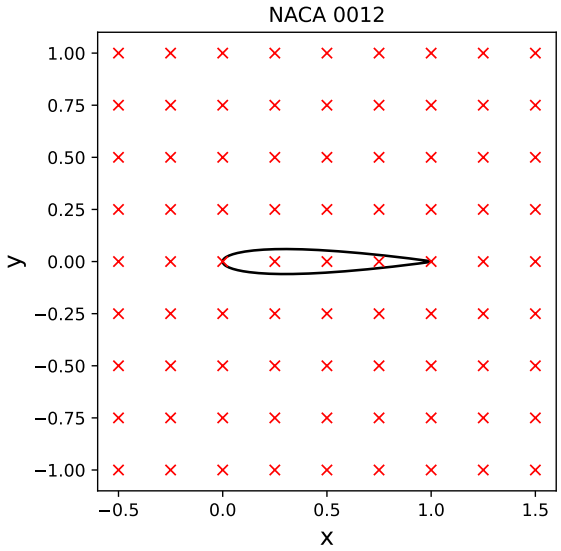
$\mathbb{R}^q \times \mathbb{R}^q$. Symbol $\mathbf{O}$ má význam predikce modelu. Neuronová síť je poté zobrazení

$$f : \mathbf{I} \mapsto \mathbf{O}.$$

Tlakové pole, které je v této práci značené jako $\hat{\mathbf{O}}$ představuje hodnoty tlaků v bodech $P(q)$ získané pomocí řešiče OpenFoam. Cílem je získat neuronovou síť, jejíž predikce $\mathbf{O}$ co nejlépe aproximuje vzorové pole $\hat{\mathbf{O}}$.

$$\mathbf{O} - \hat{\mathbf{O}} \to 0$$

Konstatní souřadnice bodů pole $P(q)$ jsou externí informací a není potřeba je modelu poskytovat. Na obrázku 4.1 je znázorněn ilustrační příklad konkrétní geometrie profilu a pole bodů $P(q)$ pro volbu $q = 9$ a $\alpha = 0°$.



Obrázek 4.1: Profil $\vec{G}$ a body oblasti $P(9)$

K dalším účelům je uvažován počet bodů v každém směru $q = 64$, tudíž pole, které celkem obsahuje 4096 bodů. Pro vybranou geometrii je pak možné přímo vypočítat vstupní pole vzdáleností $\mathbf{I}$ a z numerické simulace v *OpenFoam* získat požadované tlakové pole reprezentované maticí $\hat{\mathbf{O}}$. Obě tato pole jsou uvažována jako dvoudimenzionální a jeden prvek tohoto pole je značen pomocí dvou indexů, stejně jako u matic. Např. $\mathbf{I}_{15}$ znamená prvek pole $\mathbf{I}$, který se nachází v 1. řádku a 5. sloupci.

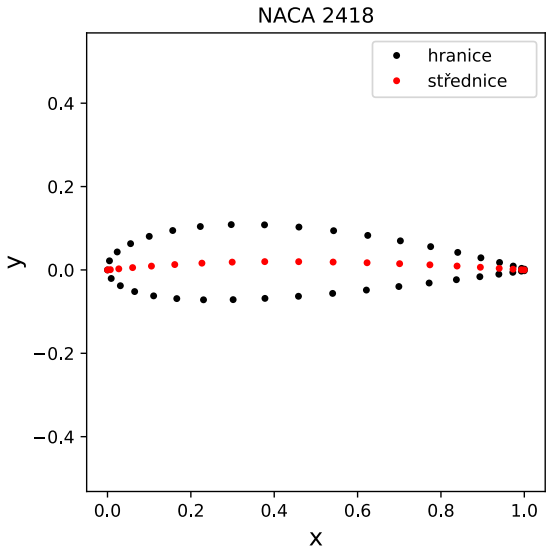
Tímto způsobem je možné vytvořit tréninkový, validační i testovací soubor dat. Soubory mají pro každý vzorek obsahovat vstup $\mathbf{I}$ a požadovaný výstup $\hat{\mathbf{O}}$.

4.2 Příprava tréninkového a validačního souboru dat

K dalším postupům je nutné získat geometrii profilu. Jak je zmíněno, ta je reprezentována souřadnicemi jednotlivých bodů G_i. Je zavedeno omezení, že pro tréninkový soubor dat obsahuje profily typu NACA, konkrétně 4-ciferné (označeny

např. NACA 0012, NACA 2418 atd.). Validační datový soubor obsahuje NACA profily stejného typu, ale zahrnuje takové kombinace geometrie a úhlu náběhu, které nejsou přítomny v tréninkovém souboru. 4-ciferné NACA profily jdou popsat parametricky (vysvětleno např. na webu [21]) a jednotlivé cifry v jejich názvu mají svůj význam, který hraje roli v tvaru profilu. Jako další předpoklad byly uvažovány profily s nenulovým radiusem na odtokové hraně.

Pro tento účel byla vytvořena funkce v Pythonu, která jako vstupní parametry přijme specifikace 4-ciferného označení profilu a počet bodů na střednici profilu, a vrátí seřazené body reprezentující hranici.



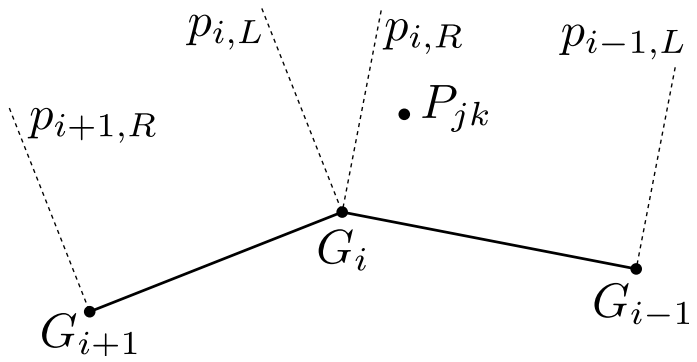
Obrázek 4.2: Reprezentace geometrie profilu

Na obrázku 4.2 jsou znázorněny body hranice profilu NACA 2418 s celkem 20 body na střednici. Při tvorbě tréninkových vzorků je pro lepší rozlišení uvažováno 100 bodů na střednici. V rámci tohoto generátoru geometrií ještě nedochází k aplikaci úhlu natočení α.

Kolem samotného pole bodů popisujících geometrii je vytvořena třída `Profile`, která se stará o operace související s geometrií, jako např. natočení profilu o úhel α (čelní bod náběhové hrany je považován za střed rotace), zápis a čtení bodů ze souboru, zahuštění bodů hranice a další výpočty popsány dále. Výhodou je, že tato třída není omezena pouze na profil NACA, ale je připravena pro práci s libovolným profilem.

4.2.1 Výpočet vstupního pole vzdáleností I

Po obdržení geometrie $\vec{G}$, definování pole bodů $P(q)$ a zadání úhlu náběhu α, je možné vypočítat vstupní pole vzdáleností, tedy matici $\mathbf{I}$. Zmíněná třída `Profile` realizuje natočení dané geometrie o poskytnutý úhel α. K určení pole vzdáleností je potřeba pracovat s již natočeným profilem. Pro vybraný bod P_{jk} z pole $P(q)$ je normálová vzdálenost od hranice profilu $\vec{G}$ určována pomocí následující myšlenky.



Obrázek 4.3: Určení normálové vzdálenosti

K bodu P_{jk} je nalezen nejbližší bod hranice profilu $G_i \in \vec{G}$. Potom pro body G_i, G_{i+1}, G_{i-1} jsou určeny normálové přímky $p_{i+1,R}$, $p_{i,L}$, $p_{i,R}$, $p_{i-1,L}$ k příslušným úsečkám viz obrázek 4.3. Nejbližší vzdálenost bodu P_{jk} k hranici $\vec{G}$ je následně určena jako

$$\min\left(\hat{d}(P_{jk}, |G_i, G_{i-1}|), \hat{d}(P_{jk}, |G_i, G_{i+1}|), \hat{d}(P_{jk}, G_i)\right),$$

kde funkce $\hat{d}$ představuje euklidovskou vzdálenost v $\mathbb{R}^2$ od daných entit, a označení $|AB|$ značí úsečku mezi body A a B. Funkce $\hat{d}$ určuje vzdálenost bodu od úsečky jako vzdálenost od přímky, kterou úsečka definuje, v případě, že daný bod leží v normálovém pásu znázorněném čárkovaně na obr. 4.3. Pokud daný bod v pásu neleží, je brána vzdálenost k nejbližšímu krajnímu bodu úsečky. Algoritmicky tento problém je zpracován následovně.

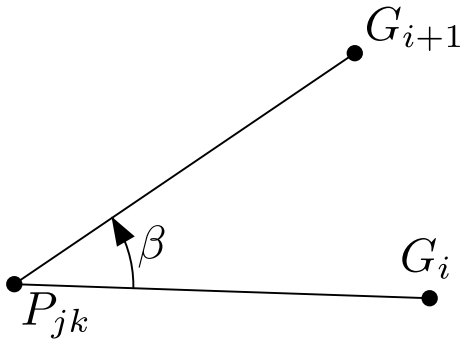
Algorithm 2 Algoritmus pro výpočet pole vzdáleností

Require: $\vec{G}, P(q)$
  for $j, k = 1 \ldots q$ **do**
    find nearest G_i from P_{jk}
    find lines $p_{i+1,R}$, $p_{i,L}$, $p_{i,R}$, $p_{i-1,L}$
    if p_{jk} is between $p_{i+1,R}$, $p_{i,L}$ **then**
      $\mathbf{I}_{j,k} = \hat{d}(p_{jk}, |G_i, G_{i+1}|)$
    else if p_{jk} is between $p_{i,R}$, $p_{i-1,L}$ **then**
      $\mathbf{I}_{j,k} = \hat{d}(p_{jk}, |G_i, G_{i-1}|)$
    else
      $\mathbf{I}_{j,k} = \hat{d}(p_{jk}, G_i)$
    end if
  end for

Vzhledem k vysokému počtu cyklů jsou funkce potřebné pro tento algoritmus implementovány v *Cythonu* za účelem úspory výpočetního času. *Cython* je statický kompilátor umožňující pomocí specifické syntaxe definovat funkce v C, které lze v Pythonu použít. *Cython* je kompatibilní s knihovnou `numpy`.

Výše uvedeným postupem vznikne pole vzdáleností $\mathbf{I}$, které ještě není v požadovaném formátu. Všechny prvky $\mathbf{I}_{jk} \geq 0 \ \forall j, k$, protože vzdálenost je nezáporné reáné číslo. Je vhodné pro neuronovou síť odlišit body uvnitř geometrie profilu, které nepatří do výpočetní oblasti. Proto jsou v těchto bodech uvažovány normálové vzdálenosti od hranice profilu se záporným znaménkem.

Je tedy nutné se zamyslet, jak poznat, zda bod P_{jk} leží uvnitř diskrétně definované hranice (křivky) $\vec{G}$.



Obrázek 4.4: Určení normálové vzdálenosti

Výhodou je, že profil v *Selig* formátu obsahuje seřazené body od odtokové hrany "proti směru hodinových ručiček". Pak lze snadno vybrat dva sousední body hranice (G_i, G_{i+1}) a pro libovolný bod P_{jk} je možné určit úhel β viz obrázek 4.4, který svírají vektory $\overrightarrow{P_{jk}G_i}$ a $\overrightarrow{P_{jk}G_{i+1}}$. Obecně úhel β, který je svírán $\vec{a}$ a $\vec{b}$ se dá jednoduše určit pomocí formule

$$\beta = \arcsin\left(\frac{\|\vec{a} \times \vec{b}\|}{\|\vec{a}\| \cdot \|\vec{b}\|}\right), \tag{4.1}$$

kde symbol $\times$ představuje vektorový součin. Jsou-li vektory

$$\vec{a} = [a_1, a_2, 0],\ \vec{b} = [b_1, b_2, 0]$$

pouze vektory ve 2D, dá se předpokládat, že jejich součin $\vec{a} \times \vec{b}$ bude mít nenulovou pouze složku z s hodnotou

$$(\vec{a} \times \vec{b})_z = a_1 b_2 - a_2 b_1.$$

Norma vektoru je vždy nezáporné číslo, ale pokud s omezením na 2D ve vztahu 4.1 dojde k nahrazení čitatele výrazem $(\vec{a} \times \vec{b})_z$, vztah bude respektovat i znaménko a vrátí informaci, v jakém smyslu je úhel β orientován.

$$\beta = \arcsin\left(\frac{(\vec{a} \times \vec{b})_z}{\|\vec{a}\| \cdot \|\vec{b}\|}\right) \tag{4.2}$$

Tento fakt se hodí k rozpoznání, zda je bod P_{jk} uvnitř nebo vně profilu. Zpětně k referenci na obrázek 4.4, po vyčíslení všech úhlů β_i vztažených k P_{jk}, pro $i \in \langle 1, g-1\rangle \cap \mathbb{N}$, a následné sumě

$$\gamma_{jk} = \sum_{i=1}^{g-1} \beta_i,$$

lze rozhodnout o poloze bodu P_{jk}. V teoretickém ideálním případě, pokud by bod P_{jk} ležel uvnitř profilu, úhel $\gamma_{jk} = 360°$, v opačném případě $\gamma_{jk} = 0°$. V reálném případě tato situace vlivem diskretizačních chyb přesně nenastane, ale je nastavena mez, že pokud $\gamma_{jk} \geq 180°$, je bod P_{jk} považován za vnitřní a normálová vzdálenost k hranici v tomto bodě bude mít záporné znaménko.

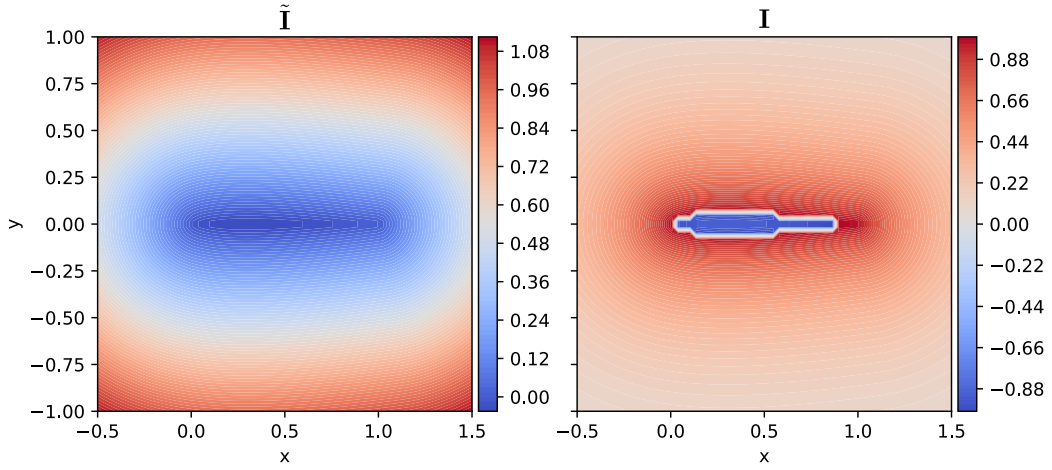
Aby mohla dobře konvoluční neuronová síť rozpoznat hrany, hodí se vzdálenosti přetransformovat tak, aby hrana profilu v poli vzdáleností $\mathbf{I}$ vynikla. V této pozici jsou v normálovém směru od hranice vzdálenosti lineární funkcí. Je velmi minimální rozdíl ve vzdálenostech vyčíslených v bodech, kdy jeden z bodů se limitně blíží hranici zvenčí a druhý zevnitř. Pro narušení této linearity je na každou hodnotu celého pole vzdáleností aplikována tato funkce.

$$f(x) = \begin{cases} e^{-2.3x} & \text{pro } x > 0 \\ -e^{2.3x} & \text{pro } x \leq 0 \end{cases} \tag{4.3}$$

Původní netransformované pole vzdáleností je označeno jako $\tilde{\mathbf{I}}$ a stále tak platí výrok, že model představuje zobrazení $f : \mathbf{I} \mapsto O$, neboť jako vstup jsou brány transformované vzdálenosti. Konstanta "2.3" v rovnici 4.3 je volena experimentálně tak, aby malé vzdálenosti vynikly, ale zároveň aby byly řádově srovnatelné s většími vzdálenostmi.

Na obrázku 4.5 je vyobrazen jeden vzorek vstupního pole vzdáleností. Levý graf představuje originální pole $\tilde{\mathbf{I}}$ normálových vzdáleností od hranice $\vec{G}$, zatímco vpravo je vykresleno pole $\mathbf{I}$, které slouží jako vstupní data pro neuronovou síť.

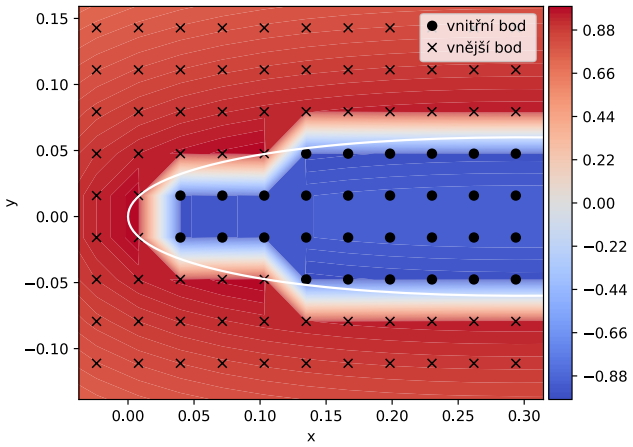
Popsanými postupy jsou tvořeny všechny vstupní data tréninkového datového souboru. Zároveň skripty zpracovávající tyto procesy jsou nezbytnými k použití neuronové sítě, protože správně definují vstupní formát a souřadnice bodů pole $P(q)$. K vytvoření správně formulovaného vstupu pro model tak stačí pouze přiložit soubor s geometrií profilu v *Selig* formátu a zadefinovat úhel α.



Obrázek 4.5: Vstupní pole vzdáleností $\tilde{\mathbf{I}}$ a transformovaných vzdáleností $\mathbf{I}$ pro NACA 0012, $\alpha = 0°$

Na obrázku 4.5 profil vypadá "hranatě" kvůli vlivu diskretizace pole vzdáleností do sítě bodů $P(q)$. Na obrázku 4.6 je pro lepší představu zobranen detail transformovaného pole vzdáleností i s hranicí profilu (bíle) a rozlišenými vnitřními a vnějšími body pole $P(q)$.

Tato část detekce bodu uvnitř profilu je také implementována v *Cythonu*. Je tedy vhodné uvést porovnání. Určit celé pole $\mathbf{I}$ pro profil definovaný 199ti body na



Obrázek 4.6: Detail pole transformovaných vzdáleností **I**

hranici a pole bodů $P[64]$, tj.celkem 4096 bodů, trvalo díky implementaci skriptů v *Cythonu* 0.46 sekund oproti použití čistého Pythonu a knihovny `numpy`, které vyžadovalo 1.85 sekund. První přístup tedy urychlí proces zhruba čtyřnásobně. Výsledné pole **I** je ukládáno v datové struktuře `numpy.array` ke snadnému načtení pomocí knihovny PyTorch.

4.2.2 Příprava výstupního tlakového pole

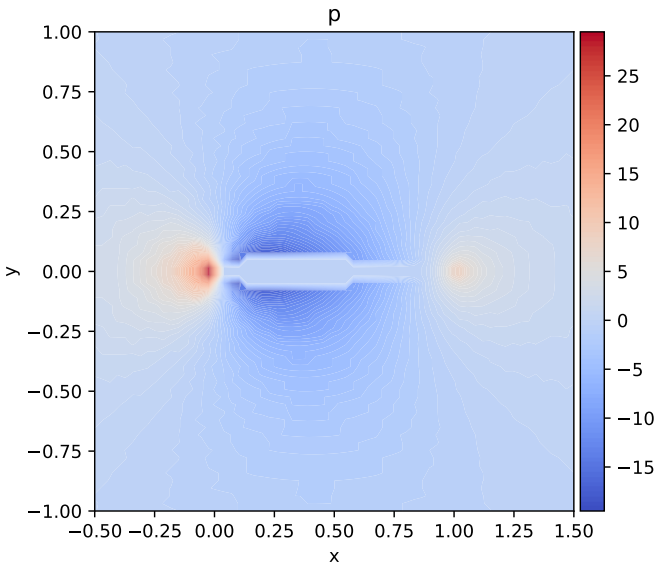
V této sekci je popsána příprava tlakového pole jednoho vzorku. Parametry geometrie $\vec{G}$, pole bodů $P(q)$ a úhel náběhu α jsou již známy a definovány z předchozích procesů. Aby byly data připravena pro trénink, je potřeba ve stejných bodech (tj. v bodech pole $P(q)$), ve kterých jsou vyčísleny vzdálenosti **I**, získat i hodnoty tlaku a tedy i zmíněné vzorové výstupní pole $\hat{\mathbf{O}}$. V rámci automatizace tohoto procesu je připraven výchozí OpenFoam případ, který má již definované vše, co je konstantní, zmíněno v kapitole 3. Jedná se o nastavení výpočtů, okrajové podmínky, apod. V softwaru OpenFoam to znamená správně definovat soubory v adresářích `system/` a `0/`.

Před zahájením numerických výpočtů je na místě získat výpočetní síť. Software *construct2d* je aplikace běžící v konzoli, která na základě dané hranice profilu (v *Selig* formátu) a dalších specifikovaných parametrů dokáže vytvořit výpočetní síť. Toho je využito a pomocí skriptů v Pythonu je tvorba sítě automatizována. Pro ovládání *construct2d* stačí jednotlivé instrukce nadefinovat do textového souboru a poté pomocí *pipe* operátotu ("|") v *bash* je lze použít jako vstupní data pro tento software. Všechny výpočetní sítě jsou tvořeny s celkem 250 body na hranici profilu a 100 body v normálovém směru. Výsledná síť je ve formátu `.p3d`, který ji umožňuje nahrát do OpenFoam.

K načtení výpočetní sítě do kopie repozitáře vzorového případu je možné využít funkce připravené v OpenFoam. Prvním příkazem je `plot3dToFoam`, který přijme jako argument cestu k přítomnému `.p3d` souboru a načte výpočetní síť do složky `constant/polyMesh`. Následně je potřeba mít správně nadefinovaný soubor `system/createPatchDict` a použít funkce `autoPatch` a `createPatch`, které automaticky rozdělí vnější plochy výpočetní sítě a přiřadí jim definované parametry. Jako poslední bylo potřeba používat funkci `transformPoints` pro natočení sítě

o úhel α kolem správné osy a tím i realizaci úhlu náběhu. Vzhledem k tomu, že jsou požadovány jako výstup hodnoty tlaku v definovaných bodech pole $P(q)$, je nutné do případu zahrnout funkci `probes` a definovat její konfigurační soubor, což obnáší vypsání souřadnic všech bodů, ve kterých má být tlak vyčíslen - tedy body $P(q)$. Po těchto popsaných procedurách lze už jen zavolat příkaz `simpleFoam` pro spuštění numerického řešiče *SIMPLE*. Vypočtené výsledky v poslední iteraci (500) nejsou pro samotnou neuronovou síť důležité, ale je potřeba extrahovat ze složky `postProcessing` soubor s nainterpolovanými hodnotami tlaku do zadaných bodů.

Pro vytažení hodnot z tohoto souboru je vytvořen skript, který navíc uloží data jako `numpy.array` (pole $\hat{\mathbf{O}}$) pro snadné a rychlé použití v Pythonu. Na obrázku 4.7 je zobrazeno výstupní tlakové pole vypočtené v *OpenFoam* interpolované do poskytnutých bodů $P(q)$.



Obrázek 4.7: Výstupní tlakové pole pro NACA 0012, $\alpha = 0°$

Mimo `probes` jsou také použity OpenFoam funkce `surfaces` a `forceCoeffs`, kterých je potřeba k určení rozložení tlaku na hranici profilu a vztlakového koeficientu. Tyto informace slouží k analýzám přesností modelů, které jsou v práci dále uvedeny.

4.2.3 Automatické generování vzorků

Datová sada zahrnuje 29 různých 4-ciferných NACA profilů a každý je počítán pro úhly náběhu od $-10°$ do $10°$ po $0.5°$, což je celkem 41 různých hodnot pro veličinu α. To dává dohromady 1189 unikátních kombinací. Z počtu 29 profilů je náhodně vybráno 6 do validačního souboru dat a zbylých 23 tvoří tréninkový soubor. Se zahrnutím všech uvedených úhlů náběhu má pak validační soubor 246 vzorků a tréninkový 943. Pro trénink neuronové sítě stačí, aby každý z těchto vzorků obsahoval dva soubory. Jeden z nich je vždy pojmenován jako `input.npy`, který obsahuje vstupní pole $\mathbf{I}$ pro konkrétní vzorek a druhý `output.npy` s výstupním polem $\hat{\mathbf{O}}$. Dále je souhrnně shrnut celý postup pro vytvoření jednoho vzorku dat.

K vytvoření všech vzorků jsou použity skripty zajišťující výše popsané procesy, které zahrnují krom popsaných postupů i vytváření adresářů pro jednotlivé vzorky,

Algorithm 3 Algoritmus pro vytvoření jednoho vzorku

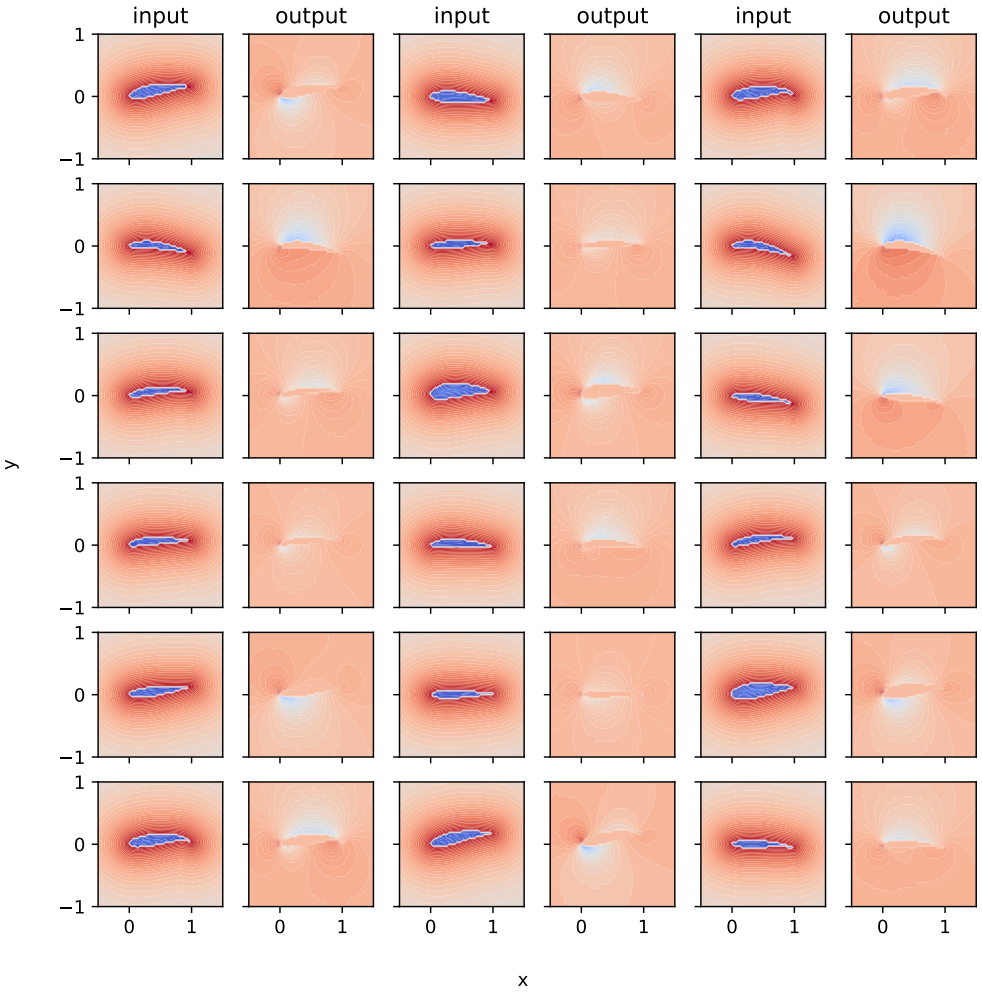
Require: $P(q), \alpha$, 4-digit profile label
Generate points $\vec{G}$ and write to file (*Selig*, with file suffix *.dat*)
Load *.dat* file and create instance of `Profile`
Apply angle α
From $P(q)$ and $\vec{G}$ compute input field $\mathbf{I}$ (*input.npy*)
Use *construct2d* with *.dat* file as input and generate mesh
Copy defined OpenFoam case and load mesh
Run solver *simpleFoam*
Extract results from directory `postProcessing` *OpenFoam* of OpenFoam case
Save extracted values as $\mathbf{O}$ (*output.npy*)
Delete unnecesary files and directories

manipulaci se soubory, promazávání nepotřebných dat z *OpenFoam* a *construct2d* a vytažení stěžejních dat z výstupních souborů použitých softwarů.

Celá příprava jednoho vzorku trvala s použitím jednoho jádra procesoru *AMD Ryzen 7 5800H* kolem 37 s, což dohromady pro celý soubor (tréninkový i validační) vyžadovalo zhruba 12 hodin výpočetního času. Pro úsporu času byl proces tvorby dat v Pythonu paralelizován použitím vestavěné knihovny *concurrent.futures*. S touto paralelizací na 4 jádrech procesoru se průměrný čas přípravy jednoho vzorku snížil na 10 sekund a celkový čas pro všechny vzorky tedy zhruba na 3 hodiny 20 minut.

Na obrázku 4.8 je znázorněno 18 náhodně vybraných vzorků tréninkového souboru složeného z 4-číselných NACA profilů, které obsahují připravená vstupní a ideální výstupní data pro neuronovou síť. Graf je uveden pouze pro představu o formě dat a neobsahuje zavedené barevné stupnice. Každý vzorek obsahuje dvojici grafů ležících vedle sebe, kdy levý z nich je vstup $\mathbf{I}$ a pravý správný výstup $\hat{\mathbf{O}}$.

V rámci této práce je zároveň vytvořen i další datový soubor pro účel porovnání. Tato sada vzorků neobsahuje pouze 4-ciferné NACA profily, ale vybrané profily z airfoil databáze [21]. Jde o 30 profilů (20 tréninkových a 10 validačních), které jsou obtékány pod úhly náběhu od $-10°$ do $10°$ po $1°$. Soubor tak obsahuje 420 tréninkových a 210 validačních vzorků. Odlišností od předchozího souboru dat je, že spektrum tvarů profilů je rozmanitější než v případě omezení na 4-ciferné NACA profily. Zmíněné geometrie jsou získávány z databáze profilů [21] metodou *web scraping*, která se dá jednoduše v Pythonu implementovat pomocí knihovny *requests*. Celý proces tvorby dat tohoto typu, se oproti algoritmu 3 liší pouze v prvním kroce, kdy nejsou geometrie parametricky dopočítávány.



Obrázek 4.8: Vybrané příklady ze vzorku dat s 4-ciferným NACA profily

4.3 Tvorba neuronové sítě

Problematika neuronových sítí je v této práci prakticky řešena prostřednictvím vytvořených tříd, funkcí a objektů, které obsahuje knihovna PyTorch. V situaci, kdy je připravený tréninkový a validační soubor dat a data jsou uložena ve struktuře `numpy.array`, která má pro jeden vzorek rozměr 64×64 prvků, je potřeba data vhodně zprostředkovat, aby s nimi mohla neuronová síť pracovat. Pro tyto účely slouží struktura `torch.Tensor`. K dynamickému načítání dat ze souborů, převodu do struktury `torch.Tensor` a transformace dat je použit modul `torch.util.data` a zejména třídy `Dataset` a `DataLoader`, které slouží jako abstraktní komponenty, které je potřeba v závislosti na problému dodefinovat. Pro neuronovou síť je vhodné, aby pracovala s "malými" čísly zhruba v intervalu $(-1, 1)$. Jinými slovy, data by před použitím měla být normalizována. Toto je proporcionálně dáno vzhledem k spektru hodnot trénovatelných parametrů modelu, hodnotě *learning rate* a zároveň vstupní a výstupní hodnoty by pro přesnost predikce měly dosahovat stejných řádů.

Tento požadavek výše definované pole $\mathbf{I}$ splňuje díky normované velikosti profilu,

volbě oblasti pro pole bodů $P(q)$ a jednotkovému zesílení transformační funkce 4.3. Výstupní data $\hat{\mathbf{O}}$ však představují tlak, který v tréninkovém a validačním souboru dat dosahuje i hodnot kolem ≈ 100. Z toho důvodu jsou před zahájením tréninku výstupní data transformována vztahem

$$\mathbf{O}_{jk} \leftarrow \frac{\mathbf{O}_{jk}}{\frac{1}{2}\rho u_\infty^2}, \tag{4.4}$$

kde ρ je referenční hustota obtékajícího média uvažovaná jako $\rho = 1$ a $u_\infty = \vec{u}_{\infty,x}$ je rychlost vzduchu ve směru x v dostatečné vzdálenosti od profilu, nebo také rychlost letu profilu. Tato hodnota je definována v kapitole 3. Ve vztahu 4.4 nabývá jmenovatel konstantní hodnotu

$$\frac{1}{2}\rho u_\infty^2 = 50.$$

Transformace je řešena pomocí třídy vlastní `AirfoilDataset`, která dědí ze zmíněné třídy `Dataset`. Takto upravené pole $\hat{\mathbf{O}}$ (a zároveň i pole $\mathbf{O}$) fyzikálně představuje pole koeficientu tlaku c_p.

Načítání dat je tedy nastaveno tak, že každý načtený vzorek (vstupní i výstupní) je ve struktuře `torch.Tensor` o tvaru $[1, 64, 64]$. První číslo (1) představuje jeden kanál ve smyslu konvolučních neuronových sítí, a zbylé reprezentují rozměr vzorku daný rozměrem $P(64), \mathbf{I}$ a $\hat{\mathbf{O}}$. Aby byla pole $\mathbf{O}$ a $\hat{\mathbf{O}}$ porovnatelná, je na neuronovou síť kladen požadavek, aby i její výstupní pole $\mathbf{O}$ nabývalo těchto stejných rozměrů. Tento princip procesu načítání je používán pro všechny modely v rámci této práce. Datovou strukturu `torch.Tensor` lze snadno převést do `numpy.array`, která je dále používána pro manipulaci s daty.

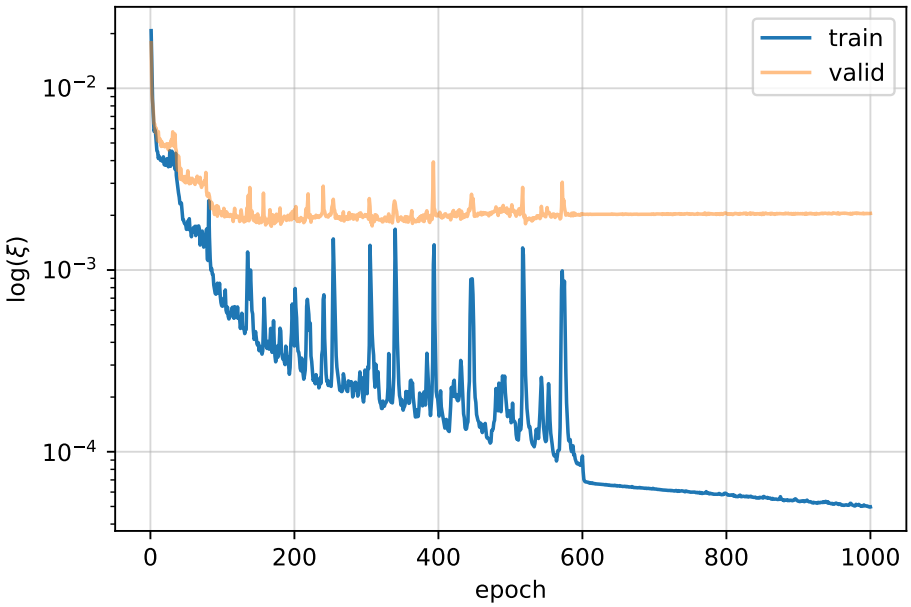
4.3.1 Hustá neuronová síť

Prvním z trénovaných modelů je hustá neuronová síť složená pouze z hustých vrstev popsaných v sekci 2.2. Účelem je ukázat, že pouze husté vrstvy nestačí k predikci pole koeficientu tlaku výše zmíněného problému. Je modelována síť, která obsahuje celkem 13 vrstev, avšak první vrstva nemá kromě formování vstupních dat žádný význam. Počet neuronů v první vrstvě musí být 4096 vzhledem k počtu elementů vstupní matice $\mathbf{I}$. Totéž platí pro výstupní vrstvu vzhledem k požadavkům na $\mathbf{O}$. Znamená to, že k jednomu neuronu vstupní vrstvy existuje právě jeden příslušný neuron výstupní vrstvy a tyto neurony představují příslušnou veličinu vyčíslenou v jednom bodě $P(q)$. Model je složen z částí *encoder* (E) a *decoder* (D), které jsou pro schematické účely popsány vektory. Výstup části E je vstupem do části D. Počty neuronů v jednotlivých částech a vrstvách jsou popsány pomocí vektorů následovně (popořadě):

$$\begin{aligned} E =& [2048, 1024, 256, 64, 16, 4] \\ D =& [16, 64, 256, 1024, 2048, 4096]. \end{aligned}$$

Takto definovaný model obsahuje celkem 21 541 668 trénovatelných parametrů.

Model je trénován na výše popsaném tréninkovém datovém souboru NACA profilů s 943 vzorky a validován na přidruženém validačním souboru dat. K přizpůsobování trénovatelných parametrů je použit optimizer *ADAM*. Při tréninku nabývajícím 1000 epoch je hodnota chyby určena pomocí funkce *MSELoss* [10] a také je volen



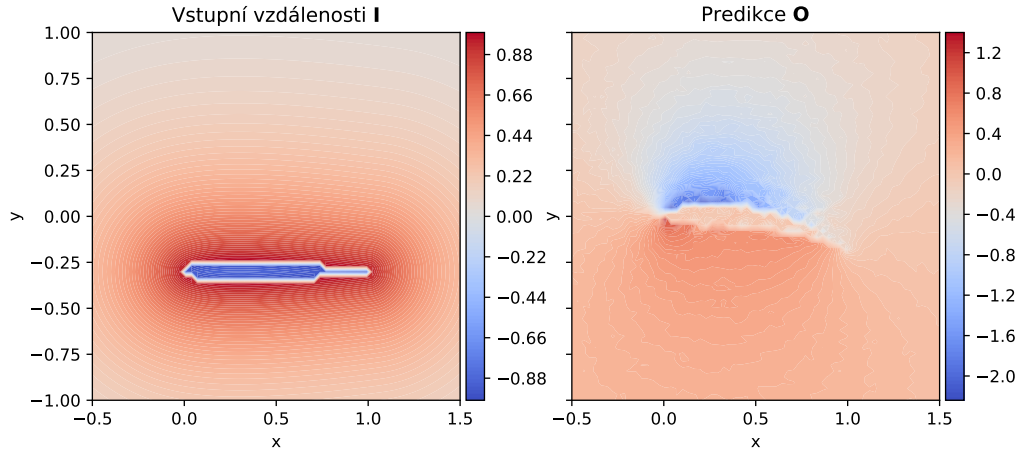
Obrázek 4.9: Průběh hodnoty chyby při tréninku husté sítě

hyper-parametr *batch size* $s = 32$. Hyper-parametr *larning rate* η je nastaven na počáteční hodnotu 10^{-3} a po 600. epochu redukován kvocientem 0.1, tudíž na $\eta = 10^{-4}$. Podobný přístup lze odůvodnit tak, že po určitém počtu epochů se hodnota chybové funkce stále dokola pohybuje kolem lokálního minima a je potřeba zmenšit krok v přizpůsobování trénovatelných parametrů, aby mohla lépe "sklouznout" ke svému minimu. Podobná myšlenka ohledně *learning rate* je využívána v této práci i dále. Trénink tohoto modelu trval celkem 873 sekund s využitím grafické karty GeForce RTX 3060 6GB.

Na obrázku 4.9 je znázorněn proces tréninku jako průběh hodnoty chyby (v logaritmických souřadnicích) v závislosti na epochu. Z obrázku je patrné, že hodnota chybové funkce na validačním souboru dat, se již zhruba od 100. epochu nemění a rozdíl mezi chybovými hodnotami tréninkové a validační sady je téměř dva řády. Hodnota chyby na konci tréninku navíc není u validačního souboru minimální. Toto nejsou přívětivé ukazatele.

Ovšem použití zmíněné architektury s sebou nese mnohem závažnější problém. Vzhledem k tomu, že konkrétní neuron vstupní vrstvy modelu je vázán ke konkrétnímu neuronu bodu pole $P(q)$, závisí celková predikce na umístění profilu $\vec{G}$ v poli $P(q)$. Jednoduše řečeno, neuronová síť je trénovaná tak, že profil leží uprostřed pole, tudíž neurony odpovídající pozici "uprostřed" pole jsou citlivé na výraznější gradienty tlaku kvůli výskytu profilu. Neurony odpovídající polohám vzdáleným od "prostředka" nemusí tak intenzivně reagovat na výskyt profilu a tomu jsou přizpůsobeny i jejich parametry. Pokud by došlo k posunutí profilu $\vec{G}$ v poli $P(q)$, model by začal poskytovat naprosto mylné predikce. Tento problém ilustruje obr. 4.10.

Vstupní data (pole **I**, 4.10 vlevo) jsou formulována tak, že dochází k posunutí profilu ve směru y o hodnotu -0.3. Predikce modelu (pole **O**, 4.10) jednoznačně dokazuje, že model není schopen tento jev zachytit a vrací nesmyslné výsledky. Při používání modelu sice není důvod takto posouvat vstupní geometrii, ale tato analýza dává najevo, že pro "správnost" predikce je potřeba, aby profil a jeho blízké



Obrázek 4.10: Predikce hustou neuronovou sítí pro posunutý profil

okolí překrýval stále stejné body pole $P(q)$. Obecně toto nelze vlivem různých tvarů geometrie profilu a různých úhlů náběhu α dodržet. Proto je tento model obecně nepoužitelný a poukazuje na aplikaci konvolučních vrstev, které tyto jevy eliminují (viz přílohy).

4.3.2 Neuronová síť U-net

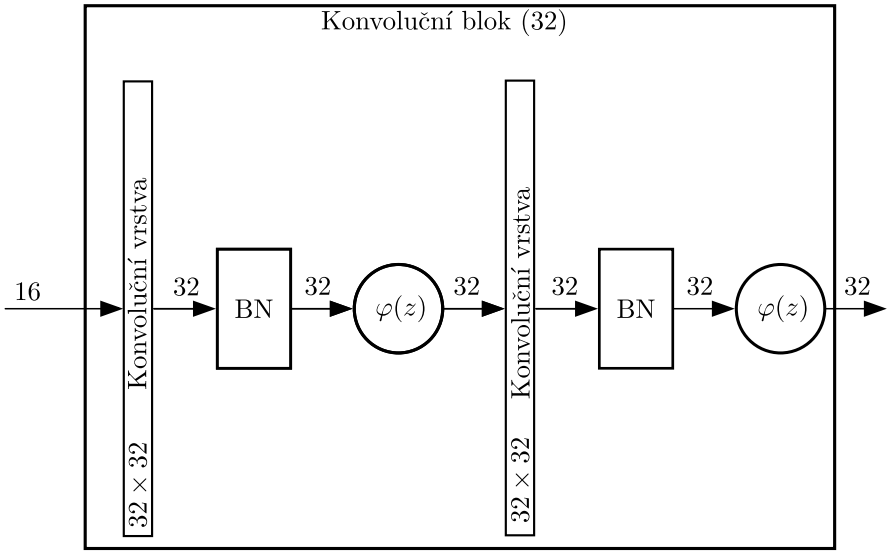
V návaznosti na zkušenosti z [5] a [6] je pro řešení problému predikce tlakového pole použita architektura U-net. Jedná se dle [5] o čistě konvoluční neuronovou síť, která je složena ze dvou větví navzájem propojených můstkem. Tak jako v případě husté neuronové sítě ze sekce 4.3.1 jde o větev kódovací a dekódovací. Tyto větve se však skládají z komplikovanějších prvků.

Stavebním kamenem popisovaného modelu je tzv. konvoluční blok. Skládá se ze dvou konvolučních vrstev, kdy na každou z nich je navázána vrstva *BatchNorm* (BN) [22], [10], [6] (nepovinně) a aktivační funkce. První vrstva se stará o manipulaci s počtem kanálů a druhá vrstva počet kanálů zachovává, pouze vnáší do modelu trénovatelné parametry. Konvoluční bloky jsou navrženy tak, aby zachovaly rozměry dat (*same convolution*, $\kappa = 3,\ \chi = 1,\ \psi = 1$), ale pouze aplikací několika filtrů došlo k jejich namnožení, neboli zvýšení počtu kanálů (pojem vysvětlen v sekci 2.4.1). Schéma jednoho konvolučního bloku, který pracuje s daty o rozmerech 32×32 a má 16 vstupních a 32 výstupních kanálů, je znázorněno na obr. 4.11. Podle počtu výstupních kanálů je konvoluční blok označen, protože počet souhlasí s počtem filtrů pro každou konvoluční vrstvu.

Kódovací větev obsahuje konvoluční bloky, které jsou propojeny podvzorkovacími vrstvami (v této práci konkrétně *Max Pool*), vysvětlenými v sekci 2.4.2. Tyto podvzorkovací vrstvy jsou navrženy tak, aby jejich výstup měl poloviční rozměr v obou dimenzích oproti vstupu. Počet kanálů zachovávají. Toho se dá docílit nastavením κ (*kernel size*) $= \chi$ (*stride*) $= 2$, protože rozměr pole je v řešeném případě vždy sudé číslo. Jejich účelem je snížit rozměry dat. Tímto způsobem je neuronová síť schopna rozložit daný vstup na velký počet různých jednoduchých vzorů.

Můstek je poslední konvoluční blok, který rozšiřuje počet kanálů a přechází mu podvzorkovací vrstva, která snižuje jejich rozměry.

Smyslem dekódovací části je zkonstruovat data zpět do původního rozměru



Obrázek 4.11: Konvoluční blok

64×64 o jednom kanálu. Tato větev obsahuje opět konvoluční bloky, které tentokrát snižují počet kanálů, a transponované konvoluční vrsty (sekce 2.4.3) plnící opačný úkol než podvzorkovací vrstvy, jímž je zvýšení rozměrů dat a navíc redukují počet kanálů na polovinu. Toho je opět docíleno nastavením paramerů $\chi = \kappa = 2$ v PyTorch.

Pokud by transponované konvoluční vrstvy neredukovaly počet kanálů, poskytnutý popis by odpovídal čistě konvolučním architekturám neuronových sítí, které se nazývají *Autoencoder* [6]. Model U-net však navíc koná ještě jednu důležitou operaci zvanou *skip connections* [5]. Pro U-net je potřeba, aby kódovací větev měla stejný počet konvolučních bloků jako dekódovací větev. Poté dochází k interakci mezi konvolučními bloky stejných úrovní kódovací a dekódovací větve. Princip spočívá v tom, že před provedením konvoluce v dekódovací části jsou vstupní data spojena s výstupními daty bloku kódovací části[1]. Model tak dostane hodnotnou informaci o výskytu hran a prudkých přechodů přímo ze vzorů vytvořených v kódovací větvi. Toto výrazně ovlivňuje přesnost predikce modelu, viz [6].

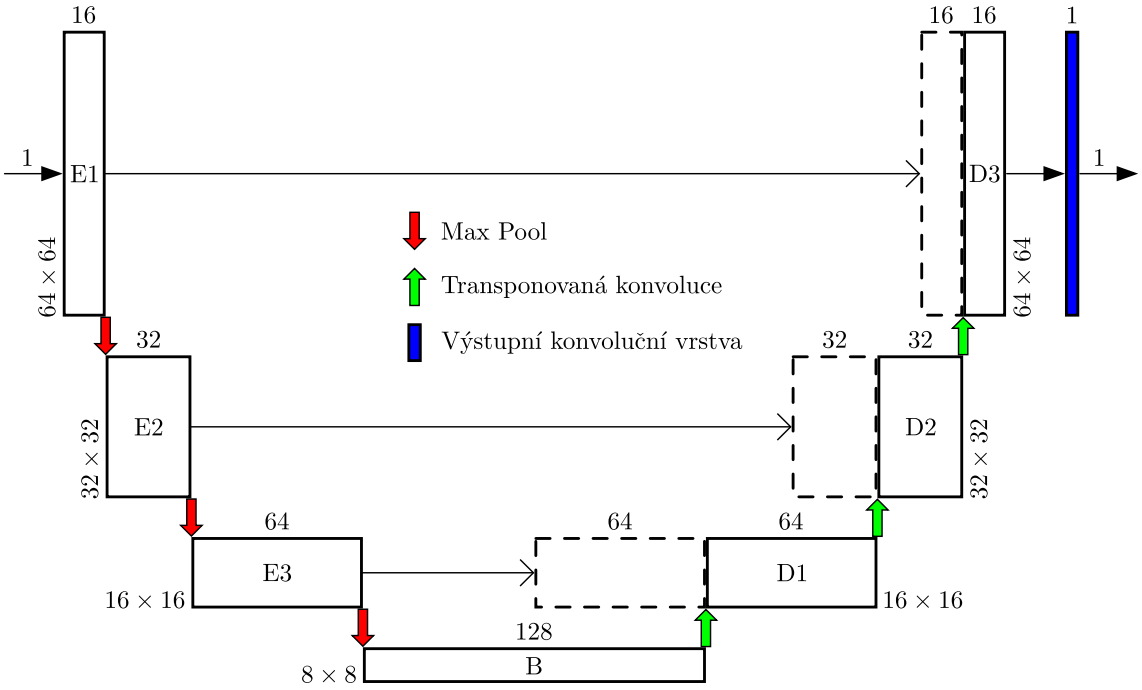
Na obrázku 4.12 je schématicky proporcionálně znázorněna neuronová síť U-net, jejíž každá z větví obsahuje 3 vrstvy mající 16, 32 a 64 filtrů. Takové nastavení je reprezentováno parametrem

$$filters = [16, 32, 64].$$

Bloky označené písmenem E patří do kódovací větve, a bloky označené D do dekódovací. Obrázek zároveň vysvětluje i původ názvu "U-net".

Značením E1, E2, E3, D1, D2, D3 a B jsou myšleny konvoluční bloky. Číslo nad konvolučním blokem má význam počtu výstupních kanálů z bloku, a popisek vedle konvolučních bloků definuje rozměr dat, se kterými blok pracuje. Čárkovaně naznačeny obdélníky symbolizují pouze referenci na výstupní data příslušného bloku kódovací části, která dekódovací konvoluční blok potřebuje k provedení konvoluce. Například blok D1 jako vstup očekává `torch.tensor` s 128 kanály o rozměru 16×16. Z toho 64 dostane transponovanou konvolucí dat z bloku B (můstek, *bridge*)

[1]Proto transponované konvoluční vrstvy musí redukovat počet filtrů, aby seděly rozměry datové struktury. Je možné si uvědomit z obrázku 4.12



Obrázek 4.12: U-net architektura

a druhých 64 převezme od bloku E3. Poté až provede samotnou konvoluci, jejíž výstupem je uvedených 64 kanálů.

Tímto postupem je možné se průchodem sítí díky trénovatelným parametrům dostat z jednoho kanálu - pole vzdáleností **I** na pole o jednom kanálu koeficientu tlaku **O**. Dále následuje provedení tréninku a přizpůsobení tak všech trénovatelných parametrů modelu.

4.3.3 Trénink a optimalizace U-net sítě

Trenink je proveden pomocí principů popsaných v sekci 2.3 přizpůsobených pro konvoluční sítě. K přizpůsobování trénovatelných parametrů dochází po vyhodnocení dávky s vzorků, přičemž během každého epochu jsou vzorky seskupovány do dávek náhodně. Ke zjištění nejoptimálnější volby nastavení hyper-parametrů tréninku a modelu je natrénováno 18 modelů U-net sítě.

Následující nastavení v tomto odstavci je uvažováno pro všechny modely identické. Jako chybová funkce je opět jako v případě husté sítě použita *MSELoss*, popsaná vztahem 2.32 a algoritmus pro optimalizaci parametrů *ADAM*. Tréninkový cyklus je nastaven také na 1000 epoch. Počáteční hodnota *learning rate* se rovná $\eta = 10^{-3}$. Během tréninku docházelo k přizpůsobování této hodnoty. Po 333. epochu byla hodnota *learning rate* redukována kvocientem 0.5 a po 666. epochu kvocientem 0.2. Finální hodnota na konci tréninku tedy dosáhla $\eta = 10^{-4}$. Toto nastavení bylo nalezeno experimentálně a mnohokrát testováno. Níže na grafu 4.13 je možné si všimnout změn v *learning rate* podle snížení amplitudy oscilací chybové funkce.

Pro nejvýkonnější model jsou hledány nejoptimálnější kombinace počtu filtrů (a konvolučních vrstev), *batch size* a použité aktivační funkce na výstupu z konvolučních vrstev. Jsou testovány následující varianty počtu filtrů:

- $[16, 32, 64]$

- $[16, 32, 64, 64]$
- $[16, 32, 64, 128]$,

velikosti *batch size*:

- 16
- 32
- 64,

a aktivační funkce:

- RELU - *Rectified Linear Unit* (viz. 2.1)
- SILU - *Sigmoid Linear Unit* (viz. 2.1).

Kombinace uvedených možností dává celkem zmíněných 18 unikátních modelů. V rámci zvýšení efektivity je díky knihovně PyTorch snadno nastaveno provedení tréninku na grafické kartě GeForce GTX 3060 6GB. Trénink jednoho modelu trval kolem 1300 sekund, což je zhruba 1.5krát déle než u modelu husté neuronové sítě. Časové porovnání má pouze informační charakter, neboť se jedná pouze o jednorázový proces a není to kritérium kvality tréninku nebo modelu. Tabulka 4.1 poskytuje vyhodnocení vybraných natrénovaných modelů na testovacím souboru dat. Testovací soubor slouží k posouzení přesnosti modelů. V tomto případě zahrnuje vybrané vzorky z databáze [21], které nejsou mezi trénovacími daty a zároveň některé z nich nepatří do kategorie 4-ciferných NACA profilů. Jedná se konkrétně o následující profily (v souladu s označením v [21]): *dae11*, *n63012a*, *naca0018*, *naca2410*, *naca24112-jf*, *naca632615*, *naca632a015*, *naca64a210*, *oaf128*, *usa26*. Každý z uvedených profilů je v souboru zachycen v úhlech náběhu ($\alpha \in \langle -10, 10 \rangle \cap \mathbb{Z}$) stupňů. Jedná se celkem o 210 testovacích vzorků. Značení ξ_{avg} má význam průměrné hodnoty chybové funkce přes celý soubor a ξ_{max} značí maximální hodnotu chyby ze všech testovaných vzorků souboru.

název modelu	počet filtrů	aktivační funkce	*batch size*	$\xi_{avg}(\cdot 10^{-5})$	$\xi_{max}(\cdot 10^{-5})$
Unet1	$[16, 32, 64, 64]$	SILU	16	25	120
Unet2	$[16, 32, 64, 64]$	RELU	16	27	174
Unet3	$[16, 32, 64, 64]$	SILU	32	32	284
Unet4	$[16, 32, 64, 64]$	RELU	32	42	315
Unet5	$[16, 32, 64, 64]$	SILU	64	36	221
Unet6	$[16, 32, 64, 64]$	RELU	64	39	265
Unet7	$[16, 32, 64]$	SILU	16	32	217
Unet8	$[16, 32, 64]$	RELU	16	34	227
Unet9	$[16, 32, 64]$	SILU	32	30	158
Unet10	$[16, 32, 64]$	RELU	32	39	266
Unet11	$[16, 32, 64]$	SILU	64	37	201
Unet12	$[16, 32, 64]$	RELU	64	42	228
Unet13	$[16, 32, 64, 128]$	SILU	16	25	211
Unet14	$[16, 32, 64, 128]$	RELU	16	28	202
Unet15	$[16, 32, 64, 128]$	SILU	32	29	186
Unet16	$[16, 32, 64, 128]$	RELU	32	29	180
Unet17	$[16, 32, 64, 128]$	SILU	64	35	213
Unet18	$[16, 32, 64, 128]$	RELU	64	34	168

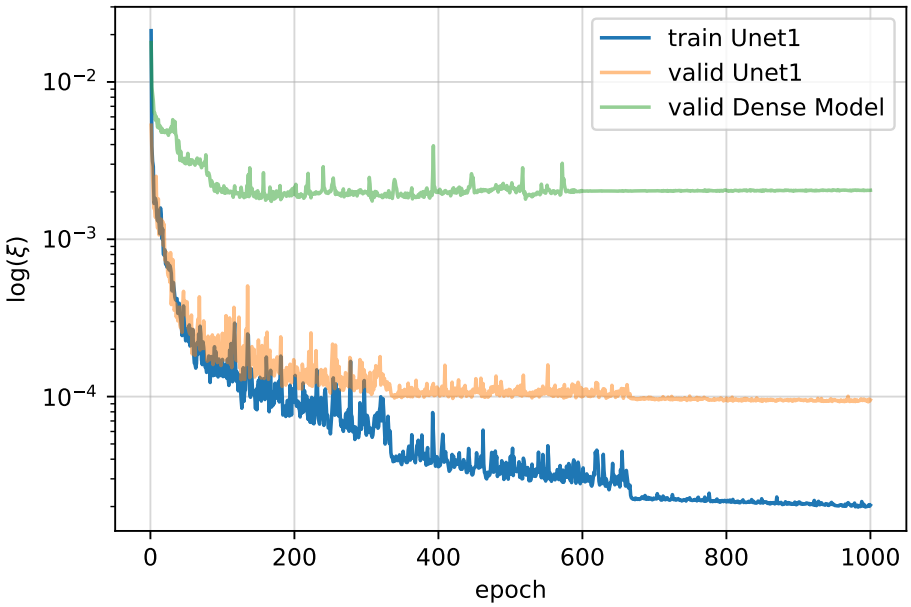
Tabulka 4.1: Hodnoty chybové funkce na testovacím souboru dat pro různé modely

Z výsledků je vidět, že nejmenších hodnot ξ_{avg} dosahují modely trénované s velikostí *batch size* $s = 16$. Také je patrné, že o něco lepších výsledků na testovacím souboru dosáhly většinou modely s aktivační funkcí *Sigmoid Linear Unit*. Nejoptimálnějším modelem se zdá být "Unet1", proto s ním je dále pracováno a jeho volba nastavení je brána za nejlepší kombinaci z uvedených možností.

Trénink tohoto modelu zabral 1150 sekund oproti 873 sekundám tréninku husté sítě, přičemž obsahuje přesně 684 817 trénovatelných parametrů, což je o 96.8% méně než model husté neuronové sítě. Z tohoto výsledku je zřejmé, že průchod a nalezení $\nabla\hat{\xi}$ je v případě konvolučních vrstev časově a výpočetně mnohem náročnější operace než u hustých.

Na obrázku 4.13 je zobrazen průběh tréninku vybraného modelu "Unet1". Z obrázku je patrné, že na konci tréninku dosáhla chybová funkce svého minima (při zanedbání oscilací) a to v obou souborech dat pro model "Unet1". Zásadní ukazatel je křivka reprezentující validační soubor dat (oranžová). Je vidět, že zhruba od 700. epochu se hodnota ξ prakticky nemění, tudíž by pro trénink stačil tento počet iterací. Důležité je však, že od tohoto bodu zůstala téměř konstantní a nedošlo k tzv. přetrénování (nárustu chyby na validační sadě při vyšší hodnotě epoch). To by znamenalo, že by se model příliš přizpůsobil tréninkovým datům a ztratil by přesnost na validačních, která jsou zásadní. Graf 4.13 zároveň reprezentuje výkonnost konvolučních sítí pro tento problém. Zeleně je totiž vykreslen průběh chyby na stejném validačním souboru při tréninku husté sítě. Model "Unet1" dosahuje více než o jeden řád nižší chyby na stejných datech.

V této situaci, kdy je k dispozici natrénovaný model schopný predikovat, je potřeba formulovat ještě pár dalších procesů pro zpracování a vyhodnocení získaných predikcí.



Obrázek 4.13: Závislost chybové funkce na epochu při tréninku modelu Unet1

4.4 Potřebné procesy pro interpretaci výsledků

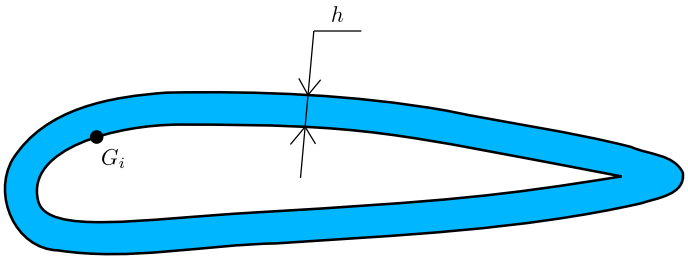
4.4.1 Numerický výpočet gradientu veličiny

Tato sekce obsahuje popis numerického výpočtu 2D gradientu obecné skalární veličiny u. Předpokladem je, že tato veličina u je definována v těch bodech pole $P(q)$, které leží vně geometrie profilu $\vec{G}$. Obecně daná veličina u může být například tlak, nebo normálová vzdálenost od hranice. Jelikož je účelem popisovat proudové pole, nemělo by smysl počítat gradient z hodnot uvnitř profilu a navíc zde nemusí být u definována. Informace, zda se vybraný bod P_{ij} pole $P(q)$ vyskytuje vně profilu je dána znaménkem normálové vzdálenosti od hranice vyčíslené v tomto bodě. Index i představuje sloupec a index j řádek dvourozměrného pole $P(q)$. Na základě této informace se dá postupovat podle následujících pravidel. Myšlenka je demonstrována na x-ové složce výsledného gradientu. Derivace v bodě P_{ij} je pak vyčíslena jako

$$\left(\frac{\partial u}{\partial x}\right)_{ij} \approx \begin{cases} \frac{u_{i+1,j}-u_{i-1,j}}{2\cdot\Delta x} & \text{, pro } d_{i+1,j} > 0 \text{ a } d_{i-1,j} > 0 \\ \frac{u_{i+1,j}-u_{i,j}}{\Delta x} & \text{, pro } d_{i+1,j} > 0 \\ \frac{u_{i,j}-u_{i-1,j}}{\Delta x} & \text{, pro } d_{i-1,j} > 0 \\ 0 & \text{, v ostatních případech,} \end{cases} \tag{4.5}$$

kde $u_{i,j} = u(P_{ij})$, $d_{i,j} = \tilde{\mathbf{I}}_{ij}$ a Δx je rozestup bodů ve směru x, který je pro pole $P(q)$ konstantní. Podmínky ve vztahu 4.5 jsou zároveň řazeny podle priority od shora dolů. Tj. platí-li první podmínka, je využita první definice. Neplatí-li první podmínka, je vyhodnocena následující atd. Tímto způsobem funguje i mechanika `if-else` v Pythonu. Hlavní vůlí je využívat první variantu, kterou je centrální diference, kvůli přesnosti 2. řádu. Pokud není možné ji použít, je zvolena dopředná nebo zpětná diference (1. řádu přesnosti) v závislosti na poloze bodu.

Derivace ve směru y je dopočítávána pomocí stejného principu následovně.



Obrázek 4.14: Oblast výskytu bodů ležící blízko hranice $\vec{G}$

$$\left(\frac{\partial u}{\partial y}\right)_{ij} \approx \begin{cases} \frac{u_{i,j+1}-u_{i,j-1}}{2\cdot\Delta y} & \text{, pro } d_{i,j+1} > 0 \text{ a } d_{i,j-1} > 0 \\ \frac{u_{i,j+1}-u_{i,j}}{\Delta y} & \text{, pro } d_{i,j+1} > 0 \\ \frac{u_{i,j}-u_{i,j-1}}{\Delta y} & \text{, pro } d_{i,j-1} > 0 \\ 0 & \text{, v ostatních případech,} \end{cases} \tag{4.6}$$

kde Δy je rozestup bodů ve směru y, který je také konstantní. Gradient veličiny u v bodě P_{ij} je vektor ve 2D, jehož složky jsou vypočtené derivace.

$$\nabla(u_{ij}) = \left[\left(\frac{\partial u}{\partial x}\right)_{ij}, \left(\frac{\partial u}{\partial y}\right)_{ij}\right]$$

4.4.2 Rekonstrukce tvaru profilu z pole vzdáleností

Je uvažováno vstupní pole vzdáleností $\tilde{\mathbf{I}}$, ze kterého jsou vytvořena vstupní data $\mathbf{I}$ pro neuronovou síť. Pole $\tilde{\mathbf{I}}$ obsahuje normálové vzdálenosti od hranice profilu vyčíslené v bodech pole $P(q)$. Tentokrát jde o inverzní úlohu, kdy je potřeba z pole $\tilde{\mathbf{I}}$ znovu vytvořit geometrii profilu. Důvodem potřeby této informace je pak následující proces interpolace koeficientu tlaku c_p na hranici profilu, přičemž je potřeba znát konkrétní souřadnice bodů, do kterých bude tlak interpolován.

Nejdříve je potřeba vybrat z pole $P(q)$ ty body, které leží vně profilu $\vec{G}$ a zároveň jsou vzdálené dostatečně blízko od hranice. Důvod druhého požadavku je ten, že tvar profilu je interpolován na principu gradientů vzdáleností. Pokud by byl použit gradient vzdálenosti vypočtený v bodě, který je daleko od hranice, nemusela by interpolace být dostatečně přesná. Nepřesnosti by byly zapříčiněny tím, že interpolovaná funkce (značena u v sekci 4.4.1) je obecně nelineární a její gradient aproximuje průběh v daném směru pouze v dostatečně malé vzdálenosti. I přesto, že pole $\tilde{\mathbf{I}}$ normálových vzdáleností od hranice profilu by teoreticky měla být ve směru gradientu v dostatečné oblasti lineární, pro interpolaci funkce tlaku by to rozhodně neplatilo.

Pro určení bodů pole $P(q)$, které leží nejblíže a vně geometrie

$$\vec{G} = [G_1, \ldots, G_i, \ldots, G_g]$$

je zaveden parametr h, který reprezentuje šířku pásu kolem geometrie profilu. V tomto pásu, který je na obr. 4.14 vyznačen modře, by měly ležet požadované body pole $P(q)$.
Daný parametr h je počítán jako

$$h = max(\Delta x, \Delta y) \tag{4.7}$$

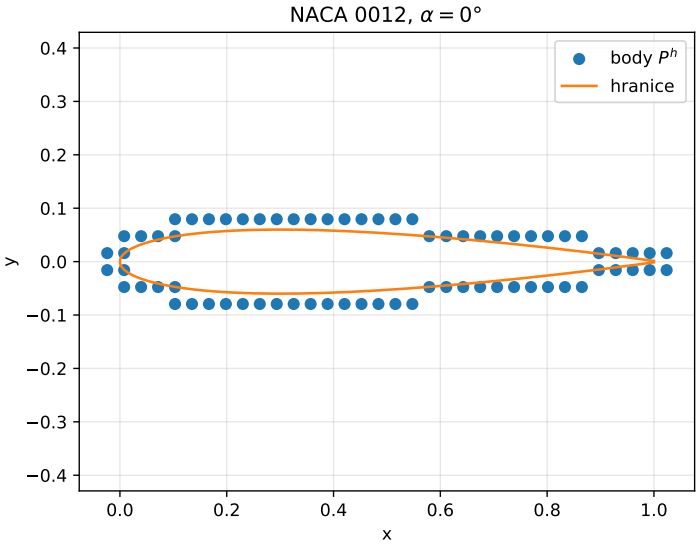
pro vybrání nejbližších vnějších bodů hranice. Vzhledem ke konstantním hodnotám Δx a Δy pole $P(q)$, je i hodnota h konstantní. Dále je vybrána množina bodů P^h ze všech možných bodů $P(q)$, pro které platí, že jejich vzdálenost od hranice profilu je kladná a menší než h.

$$P^h = \left\{ P_i^h = P_{jk} \in P(q), \text{ pokud } 0 < \tilde{\mathbf{I}}_{jk} < h \right\}. \tag{4.8}$$

Výběru bodů P^h odpovídají i elementy matic $\tilde{\mathbf{I}}^h$ a $\mathbf{O}^h$, díky vazbě $\tilde{\mathbf{I}}$ a $\mathbf{O}$ na pole $P(q)$. V množině P^h jsou body řazeny v tuto chvíli náhodně a není potřeba dodržet dvoudimenzionální strukturu jako je $P(q)$. Proto stačí značit body pole P^h jedním indexem, např. $P_i^h \in P^h$. Je potřeba mít na vědomí, že platí $P^h \subset P(q)$. Navíc mezi strukturami platí jednoznačnost a o každém bodu P_i^h lze říci, že je identický právě s jedním bodem pole $P(q)$. Proto je abstraktně zavedeno zobrazení

$$\mathcal{H} : \mathbf{X}^h \mapsto \mathbf{X},$$

které najde k vybranému prvku $\mathbf{X}_i^h \in \mathbf{X}^h$ jeho identický prvek $\mathbf{X}_{jk} \in \mathbf{X}$, tj. pro index i najde příslušné indexy j, k. Použitý symbol $\mathbf{X}$ lze substituovat za $P, \tilde{\mathbf{I}}$ a později $\mathbf{O}$.



Obrázek 4.15: Vybrané body P^h

Na základě této informace je možné pro $\forall P_i^h \in P^h$ najít jeho sousední body v $P(q)$, které jsou potřeba pro výpočty gradientů. Na grafu 4.15 jsou modře znázorněny body P^h vybrané podle uvedených pravidel, kterých je v tomto případě celkem 72. Ukázka je demonstrována na profilu NACA 0012.

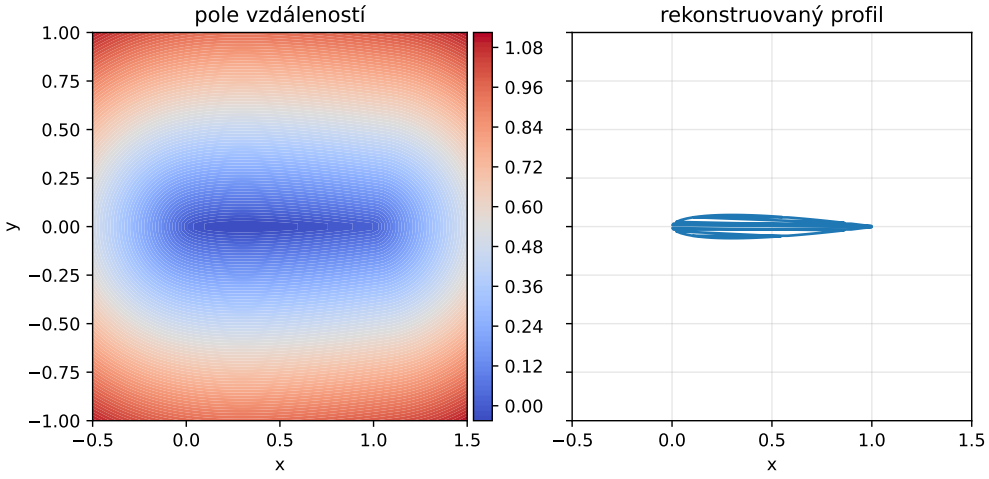
Pro skupinu bodů P^h je možné podle sekce 4.4.1 spočítat gradienty vzdáleností $\nabla(\mathcal{H}(\tilde{\mathbf{I}}_i^h))$. Navíc je zavedeno označení

$$\nabla^h(\tilde{\mathbf{I}}_i) = \nabla(\mathcal{H}(\tilde{\mathbf{I}}_i^h)).$$

Konečně interpolovaný bod $\hat{R}_i \in \mathbb{R}^2$ na hranici profilu leží ve vzdálenosti $\tilde{\mathbf{I}}_i^h$ v opačném směru než je směr gradientu $\nabla^h(\tilde{\mathbf{I}}_i)$. Potom tedy platí

$$\hat{R}_i = P_i^h - \tilde{\mathbf{I}}_i^h \cdot \frac{\nabla^h(\tilde{\mathbf{I}}_i)}{||\nabla^h(\tilde{\mathbf{I}}_i)||}. \tag{4.9}$$

Aplikací těchto postupnů pro $\forall P_i^h$ lze zrekonstruovat tvar původního profilu - množinu $\hat{R}$, což je vidět na obr. 4.16. Pravý graf ukazuje, že hraniční body profilu jsou sice nalezeny, ale nejsou správně seřazeny dle *Selig* formátu. Tento problém je vyřešen dále.



Obrázek 4.16: Rekostruovaný profil z pole vzdáleností $\tilde{\mathbf{I}}$

4.4.3 Seřazení bodů na hranici profilu

K dispozici je množina bodů $\hat{R}$ a cílem je získat uspořádaný vektor $\vec{R}$, ve kterém jsou seřazeny body množiny $\hat{R}$ od odtokové hrany "proti směru hodinových ručiček" (*Selig* formát [20]). Tento problém je řešen algoritmem využívající naivní přístup. Je předpokládána přítomnost stejného typu profilu, se stejným úhlem náběhu α, který je popsán interpolovanými body $\hat{R}$. Takový profil je vždy k dispozici, protože je z něj tvořeno vstupní pole pro neuronovou síť $\mathbf{I}$. Jeho geometrie $\vec{G}$ slouží jako šablona a je značena $\vec{G}^t$. Na šablonu $\vec{G}^t$ jsou však kladeny požadavky, aby obsahovala vhodně seřazené body (což původní profil $\vec{G}$ splňuje) a počet bodů byl zhruba o řád vyšší než počet bodů množiny $\hat{R}$ (to zatím obecně splněno není). Poté je možné pro každý bod $\hat{R}_i$ najít nejbližší bod šablony $\vec{G}^t$ a přiřadit jeho index bodu $\hat{R}_i$. Dále už zbyde jen seřadit $\hat{R}$ podle přiřazených indexů a výsledkem je uspořádaný vektor $\vec{R}$. Algoritmus řešící tento problém by mohl vypadat následovně (alg. 4).

4.4.4 Zahuštění hranice profilu

Tato sekce popisuje postupy umožňující definovat požadovaný počet b hraničních bodů profilu $\vec{G}$. Díky tomu je umožněno vytvořit šablonu $\vec{G}^t$ z minulé sekce, která splňuje všechny specifikované požadavky.

Každý bod G_i má svou x_i a y_i souřadnici. Dále je zaveden diskrétní parametr $t = [t_1, \ldots, t_i, \ldots, t_g]$ a souřadnice x_j^{new} a y_j^{new}, které jsou jeho funkcí:

Algorithm 4 Algoritmus pro seřazení množiny $\hat{R}$

for $\hat{R}_i$ in $\hat{R}$ **do**
  $d = \infty$
  $k = -1$
  for G_j^t in $\vec{G}^t$ **do**
    $d_{temp} = \text{distance}(\hat{R}_i, G_j^t)$
    if $d_{temp} < d$ **then**
      $d \leftarrow d_{temp}$
      $k \leftarrow j$
    end if
  end for
  $\hat{R}_i(index) = k$
end for
$\vec{R} = \text{sort}(\hat{R} \text{ by } \hat{R}_i(index))$

$$x_j^{new} = x^{new}(t_j),$$
$$y_j^{new} = y^{new}(t_j).$$

Je také předpokládán interval, ve kterém se hodnoty t vyskytují: $t_j \in \langle 0, 1 \rangle$. Tento parametr představuje měřítko o poloze na hranici profilu. Hodnotě $t_i = 0$ odpovídá první bod (na odtokové hraně) a hodnotě $t_i = 0.5$ bude odpovídat některý z bodů na náběhové hraně. Všechny hodnoty $t_1, \ldots, t_g$ se dají přímo z hranice $\vec{G}$ získat. Princip demonstruje algoritmus 5.

Algorithm 5 Algoritmus pro určení hodnot parametru t

$t = [0, 0, \ldots, 0], \text{length}(t) = g$
for $i = 2 \ldots g$ **do**
  $t_i = t_{i-1} + \text{distance}(G_i, G_{i-1})$
end for
for t_i in t **do**
  $t_i = t_i / t_g$
end for

Vektory $\vec{G}$ a t mají tak stejný rozměr g. Následně je pro určení křivek x^{new}, y^{new} použita funkce `interp1d` z Python knihovny `scipy.interpolation`, která využije zadaných bodů. Tato funkce je aplikována ve skriptu následovně:

$$x^{new} = \texttt{interp1d}(t, x_i \text{ pro } i = 1 \ldots g)$$
$$y^{new} = \texttt{interp1d}(t, y_i \text{ pro } i = 1 \ldots g),$$

kde x_i, y_i jsou x-ové a y-ové souřadnice bodu G_i. Využitím nalezených funkcí je už jednoduché dostat profil $\vec{G}^t$ o specifikovaném počtu bodů b. Stačí zavést nový diskrétní parametr $\tau = [\tau_1, \ldots, \tau_b]$ s tím, že tentokrát má celkem b prvků a platí

$$\tau_i = (i - 1) \cdot \frac{1}{b - 1}. \tag{4.10}$$

Z rovnice 4.10 je napřímo vidět, že $\tau_b = 1$ a $\tau_1 = 0$. Aplikace

$$\vec{G}^t = \left[G_i^t = (x^{new}(\tau_i), y^{new}(\tau_i)) \text{ pro } i = 1 \dots b\right] \tag{4.11}$$

poskytne profil popsaný b body.

4.4.5 Interpolace tlaku na hranici z tlakového pole

Na základě postupů v minulých sekcí lze nyní jednoduše interpolovat koeficient tlaku na hranici profilu. Polem představující tuto informaci o koeficientu tlaku je výstupní predikce neuronové sítě $\mathbf{O}$ (a nebo také $\hat{O}$ získané z OpenFoam). Dle sekce 4.4.1 je opět potřeba určit gradient a zavést označení:

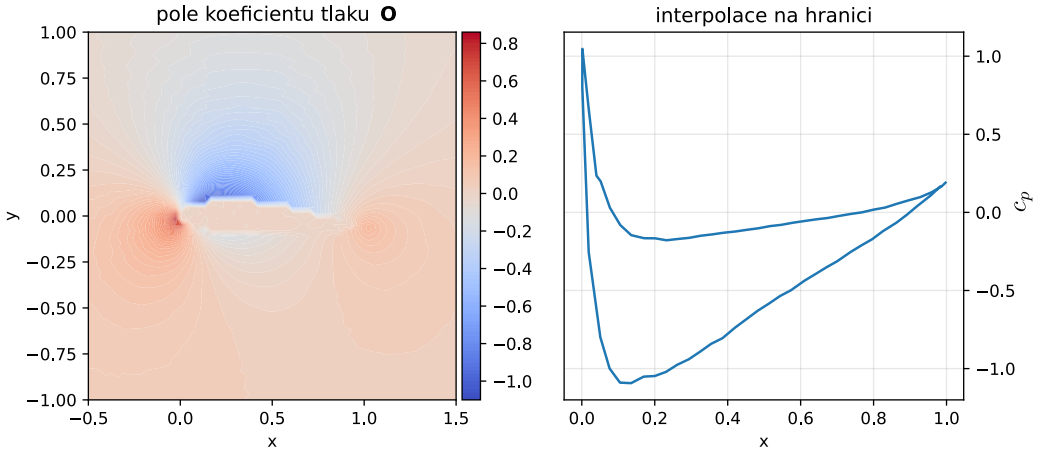
$$\nabla^h(\mathbf{O}_i) = \nabla(\mathcal{H}(\mathbf{O}_i^h)).$$

Poté zbývá určit normovaný tlak $\hat{S}_i$ na hranici profilu vztahem

$$\hat{S}_i = \mathbf{O}_i^h - \underbrace{\left[\tilde{\mathbf{I}}_i^h \cdot \frac{\nabla^h(\tilde{\mathbf{I}}_i)}{||\nabla^h(\tilde{\mathbf{I}}_i)||}\right]}_{\in\mathbb{R}^2} \cdot \nabla^h(\mathbf{O}_i). \tag{4.12}$$

Výraz v hranaté závorce byl již použit v rovnici 4.9 a není potřeba ho počítat znovu. Proto je vhodné oba procesy interpolace provádět pro každý bod současně. Navíc také výraz v hranaté závorce v rovnici 4.12 má význam vektoru, který má normálový směr k hranici v daném bodě P_i^h a velikost vzdálenosti $\tilde{\mathbf{I}}_i^h$. Skalární součin v rovnici 4.12 pouze značí promítnutí gradientu tlaku do požadovaného směru a vzdálenosti.

Výsledek se dá interpretovat tak, že hodnota tlaku $\hat{S}_i$ je nainterpolována do hraničního bodu $\hat{R}_i$. Aplikací algoritmu ze sekce 4.4.3 dojde k seřazení $\hat{R}$ do vektoru $\vec{R}$. Protože bod $\hat{R}_i$ odpovídá i prvku $\hat{S}_i$, je možné řazení při stejném procesu aplikovat i na $\hat{S}$ a získat tak seřazený $\vec{S}$ dle *Selig* formátu. Značení $\vec{S}$ a $\vec{R}$ je zavedeno pro matematický popis.



Obrázek 4.17: Interpolace tlaku c_p z pole $\mathbf{O}$ na hranici

V praxi je tento problém v Pythonu řešen pomocí struktury `numpy.array` ve 2D, kdy každý řádek drží souřadnice nainterpolovaného bodu a v něm vyčíslenou hodnotu koeficientu tlaku. Seřazení řádků v dané `numpy.array` zajistí seřazení $\hat{R}$ a $\hat{S}$ použitých pro matematický popis zároveň.

Přesto, že rovnice 4.9 a 4.12 vypadají složitě, nejedná se o nic víc než lineární interpolaci pro entity ve 2D. Obrázek 4.17 poskytuje ukázku výsledné interpolace normovaného tlaku p_n na hranici profilu ze získaného tlakového pole **O** pro profil NACA 2418, $\alpha = 4°$. Horní křivka na pravém grafu reprezentuje přetlakovou hranu profilu, a spodní křivka podtlakovou. Jedná se o vzorek z tréninkového souboru dat.

4.4.6 Numerická integrace pro získání tlakového koeficientu

Vzhledem k tomu, že výstupní pole **O** obsahuje hodnoty koeficientu tlaku c_p, které jsou interpolovány na rekonstruovanou hranici profilu $\vec{R}$, je možné z této informace spočítat koeficient vztlaku c_l. Pokud by c_p byla spojitá funkce (označ. $\bar{c}_p$) a hranice spojitá křivka $\bar{G}$, platil by vztah

$$c_l = \frac{1}{L_{ref}} \cdot \int_{\bar{G}} \bar{c}_p dx, \tag{4.13}$$

kde L_{ref} je referenční délka profilu, která je pro uvažované úlohy vždy jednotková $L_{ref} = 1$. V řešeném diskrétním případě se dá použít jednoduše numerická integrace. Skalární hodnota c_l totiž odpovídá ploše uvnitř křivky interpolovaného c_p na hranici (např. pravý graf na obr. 4.17). Je však potřeba při integraci dodržet orientaci křivky. K dispozici je seřazený vektor $\vec{R}$ hraničních bodů a díky tomu integrace může vypadat takto:

Algorithm 6 Algoritmus numerické integrace pro získání c_l

Require: $\vec{R}, \vec{S}$
  $c_l = 0$
  for $i = 2, \ldots \text{length}(\vec{R})$ **do**
    $\Delta x = R_i - R_{i-1}$
    $\Delta c = S_i - S_{i-1}$
    $c_l \leftarrow (c_l + \Delta x \cdot \Delta c/2)$
  end for

Koeficient tlaku c_p je bezrozměrná veličina a tedy koeficient c_l je také bezrozměrný. Pro příklad NACA 4415, $\alpha = 4°$ z obr. 4.17 vychází hodnota $c_l = 0.681$.

Kapitola 5

Vyhodnocení výsledků

V situaci, kdy jsou definovány všechny potřebné nástroje, je možné zhodnotit přesnost predikce sítě (konkrétně modelu "Unet1"). K vyhodnocení výsledků je použito několik geometrií profilu z databáze [21], které nejsou zahrnuty do tréninkového ani validačního souboru dat a navíc vzorky převážně nepatří do kategorie 4-číselných NACA profilů. Mezi tyto testovací vzorky patří např. profily SD7003 a S826. Je potřeba uvést, že v této sekci jsou hodnoty koeficientu tlaku c_p na hranici profilu a vztlakového koeficientu c_l počítány vždy na základě znalosti pole koeficientu tlaku $\mathbf{O}$ nebo $\hat{\mathbf{O}}$ pomocí algoritmů uvedených v sekci 4, pokud není u konkrétního grafu specifikováno jinak.

5.1 Srovnání výpočetních časů

V potřebném výpočetním čase nastávají extrémní rozdíly. Výsledky jsou měřeny a porovnány na použitém tréninkovém souboru dat obsahující 943 vzorků. Při tvorbě tréninkových dat bylo uvedeno, že výpočet všech vzorků v OpenFoam vyžadoval kolem 12 hodin výpočetního času na jednom jádru procesoru *AMD Ryzen 5800H*. Vyhodnocení stených vzorků zabralo modelu "Unet1" přesně 2.18 sekund na stejném procesoru na jednom jádře. Při použití grafické karty Nvidia GTX 3060 6GB se výpočetní čas snížil dokonce na 1 sekundu. Natrénovaná síť je tak nepoměrně rychleji schopna predikovat přibližné výsledky pro daný případ.

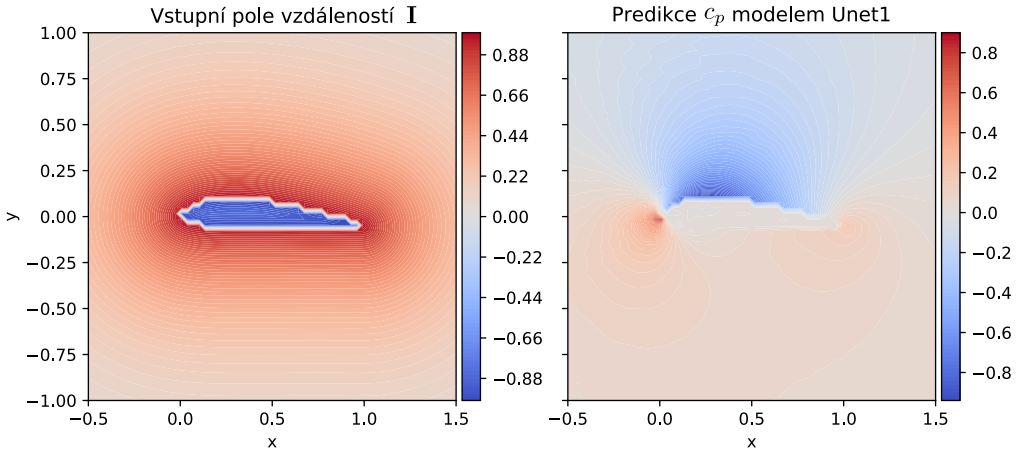
Vedle tréninkového souboru se vzorky vytvořenými z pole $P(64)$, které obsahuje 4096 bodů byl vytvořen další soubor se stejnými profily a úhly náběhu, jejichž pole $\mathbf{I}$ a $\hat{\mathbf{O}}$ jsou diskretizována do bodů $P(128)$. Úloha má tak čtyřnásobný rozměr co se týče množství dat. Na novém trénikovém souboru je natrénována síť Unet se stejnými hyperparametry jako má model "Unet1", která je pojmenována "UnetHighRes1". Tento model tedy pracuje s daty o rozměru 128x128. Zvýšením počtu bodů, do kterých se intepolují hodnoty tlaku, narostl potřebný výpočetní čas v OpenFoam k vytvoření tohoto "většího" souboru na 29 hodin na jednom jádru. Model "UnetHighRes1" zvládne na stejném procesoru predikovat všechny vzorky "většího" souboru za 9.37 sekund a s využitím grafické karty za 1.24 sekund.

Je nutné zmínit další potenciálně časově náročné procesy pro použití neuronové sítě mimo samotnou predikci. Prvním z nich je vytvoření vstupního pole $\mathbf{I}$. Operace díky optimalizaci použitím Cython byla provedena za čas 0.57 sekund pro profil s 200 body na hranici a polem $P(64)$ bodů. Dalším kandidátem je interpolace koeficientu tlaku c_p na hranici profilu ze získaného pole $\mathbf{O}$. Doba běhu tohoto procesu byla

naměřena 0.19 sekund pro pole $P(64)$. Numerická integrace ze sekce 4.4.6 vyžaduje prakticky nulový výpočetní čas vzhledem k malému počtu bodů popisujících křivku.

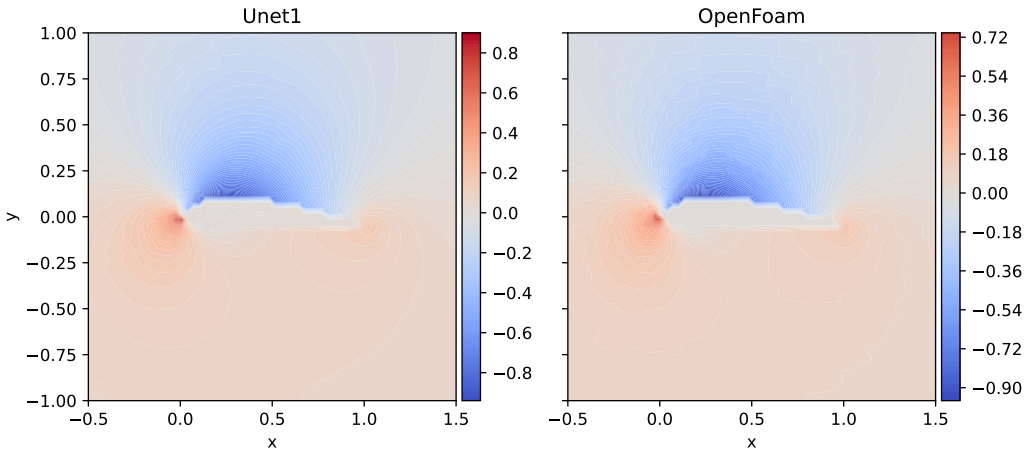
5.2 Profil NACA 4415

Nejprve je model "Unet1" otestován na 4-ciferném profilu NACA 4415. Jedná se o vzorek, který model "nikdy neviděl". Od této predikce lze očekávat velmi uspokojující výsledek, protože tento profil patří do stejné kategorie jako tréninková data. Pro ukázku je zvolen případ $\alpha = 3°$. Na obrázku 5.1 je zobrazeno vstupní pole vzdáleností $\mathbf{I}$ (vlevo) a výstupní pole predikce $\mathbf{O}$ (vpravo) představující rozložení koeficientu tlaku.



Obrázek 5.1: Vstupní data a predikce pro 4415, $\alpha = 3°$

Další graf 5.2 porovnává predikci modelu $\mathbf{O}$ s řešením v OpenFoam nainterpolovaném do bodů $P(q)$, tedy pole $\hat{\mathbf{O}}$. Barevné spektrum je voleno tak, aby konkrétní hodnoty odpovídaly stejným barvám na obou grafech. Na první pohled výsledky vypadají téměř identicky.

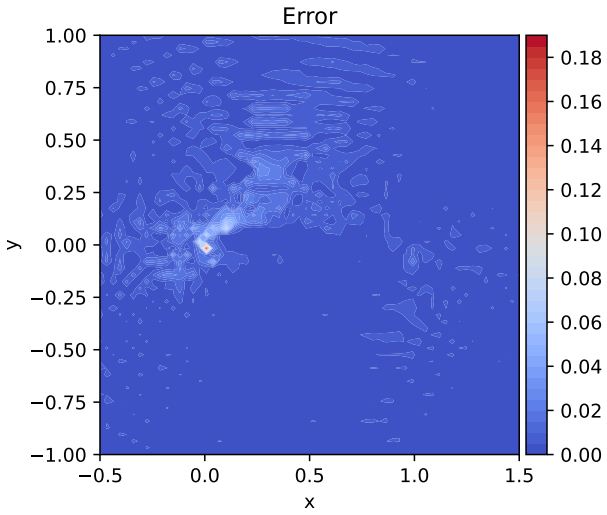


Obrázek 5.2: Porovnání predikce a výsledku z OpenFoam pro NACA 4415, $\alpha = 3°$

Rozdíl mezi výstupními daty $\mathbf{O}$ a $\hat{\mathbf{O}}$ je počítán jako

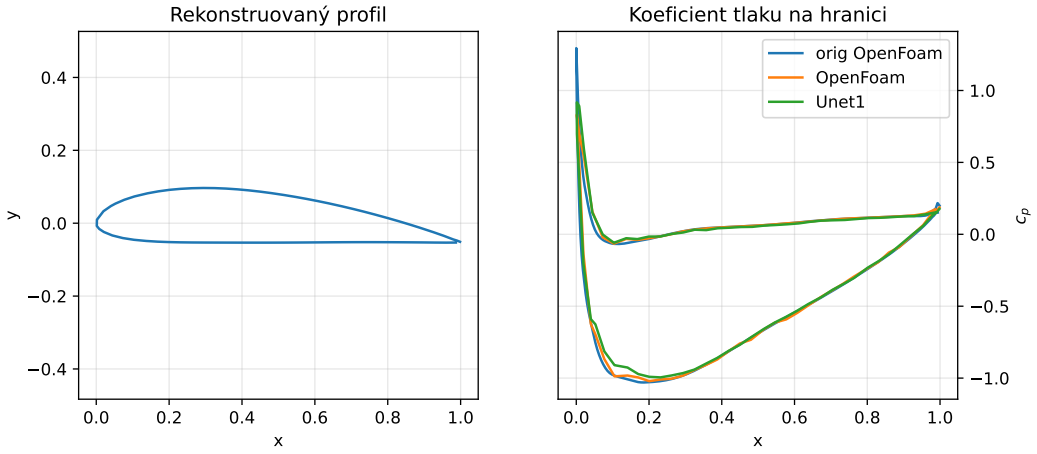
$$\mathbf{E} = |\mathbf{O} - \hat{\mathbf{O}}|. \tag{5.1}$$

Toto rozložení chyby $\mathbf{E} \in \mathbb{R}^{64} \times \mathbb{R}^{64}$ je zachyneno na grafu 5.3. Lze vidět, že k největšímu rozdílu dochází v bodě v okolí náběžné hrany. Odlišnosti ve zbytku pole jsou zanedbatelné.



Obrázek 5.3: Rozložení chyby $\mathbf{E}$ pro výsledky z obr. 5.2

K dalšímu vyhodnocení je použit postup popsaný v sekci 4.4.5 a 4.4.6. Cílem je ukázat, jak se liší průběh koeficientu c_p na hranici profilu a koeficient c_l získaný neuronovou sítí a výpočty v OpenFoam. Je nutné podotknout, že výsledky značené na obrázcích jako "OpenFoam" nejsou data vypočtená pomocí softwaru OpenFoam, ale jsou získána popsanými algoritmy z pole $\hat{\mathbf{O}}$, které je spočteno v řešiči OpenFoam. Na grafu 5.4 popisek v legendě "orig OpenFoam" reprezentuje data, která jsou získána přímo z OpenFoam pomocí funkce *surfaces* a nejsou jakkoliv zasažena zvenčí.



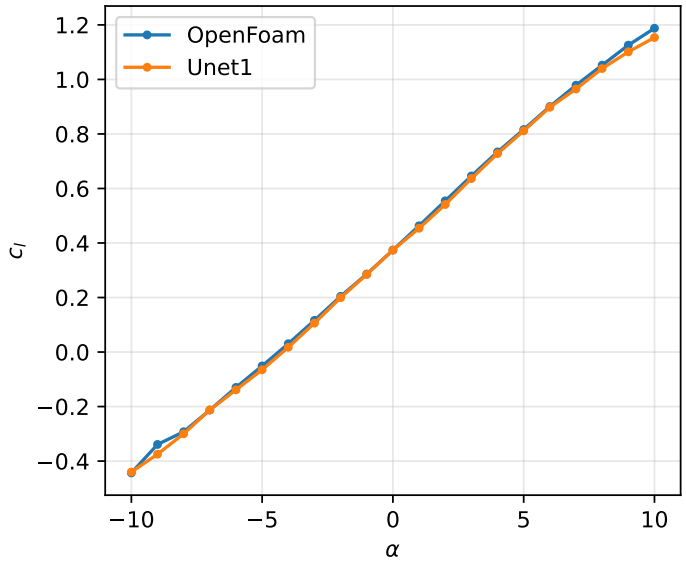
Obrázek 5.4: NACA 4415, $\alpha = 3°$, interpolace na hranici

Rozdíly mezi oranžovou a modrou čarou na grafu 5.4 (vpravo) jsou způsobeny diskretizací do pole bodů $P(64)$, které zdaleka neobsahuje tolik entit v blízkosti hranice profilu jako výpočetní síť pro numerické OpenFoam výpočty. Navíc v případě oranžové křivky jsou hodnoty c_p získávány lineární interpolací z okolních bodů. Tento graf tedy dokazuje, že přesto, že je použita takto "hrubá" diskretizace do bodů

$P(64)$, se výsledek silně podobá originálním výstupům z OpenFoam. Zároveň predikce modelu "Unet1" (zelená čára) velmi dobře napodobuje požadovaný výstup a tím aproximuje i rozložení c_p na hranici profilu, které je možné získat numericky. Koeficienty c_l tak pro tyto tři křivky numerickou intergací vychází velmi podobně:

$$\begin{aligned} c_l(\text{orig OpenFoam}) &= 0.655 \\ c_l(\text{OpenFoam}) &= 0.646 \\ c_l(\text{Unet1}) &= 0.636 \end{aligned}$$

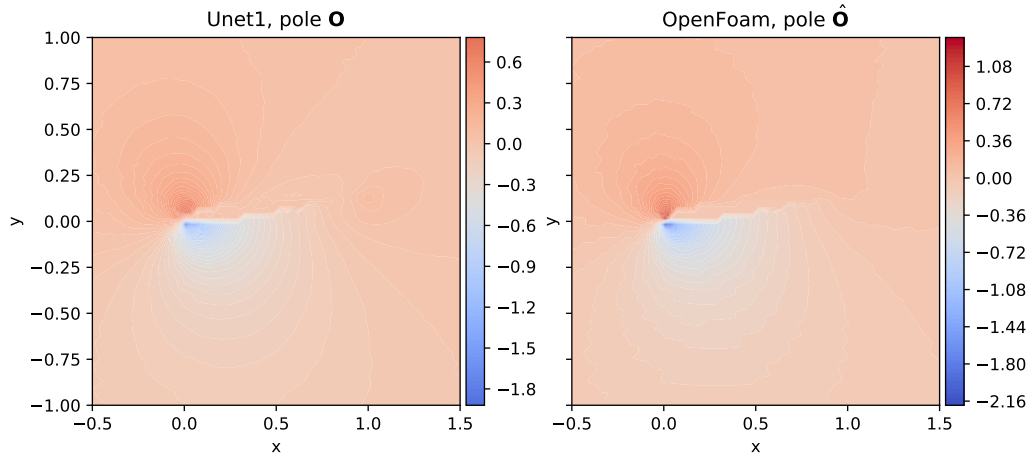
Následující obrázek 5.5 znázorňuje závislost $c_l(\alpha)$ pro daný profil NACA 4415. Výpočty jsou prováděny pro $\alpha \in \langle -10°, 10° \rangle \cap \mathbb{Z}$, což jsou hodnoty, na které byl model trénován. Predikce jsou srovnávány s daty získanými z pole $\hat{\mathbf{O}}$. Dle očekávání si model "Unet1" na analyzovaném profilu vede velmi dobře.



Obrázek 5.5: Závislost $c_l(\alpha)$ pro profil NACA 4415

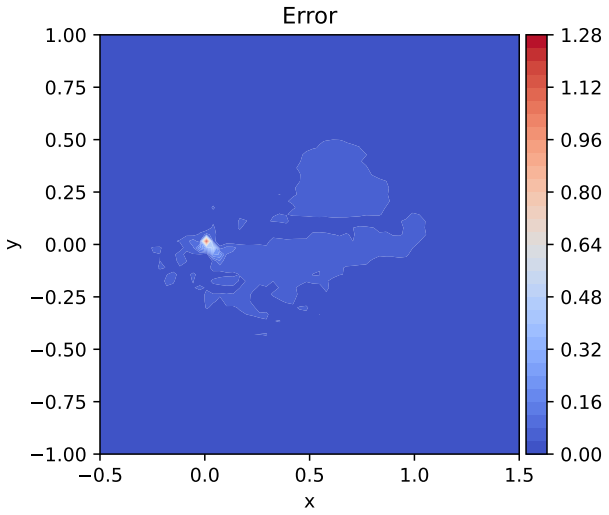
5.3 Profil SD7003

Tvar profilu SD7003 se liší od třídy 4-ciferných NACA profilů. Nicméně na základě zkušeností z tréninkové sady dat by se dalo očekávat, že konvoluční Unet síť si s tímto problémem dokáže s dostatečnou přesností poradit. Otázka je, zda nepřesnosti nejsou příliš velké.



Obrázek 5.6: Porovnání predikce a výsledku z OpenFoam pro SD7003, $\alpha = -8°$

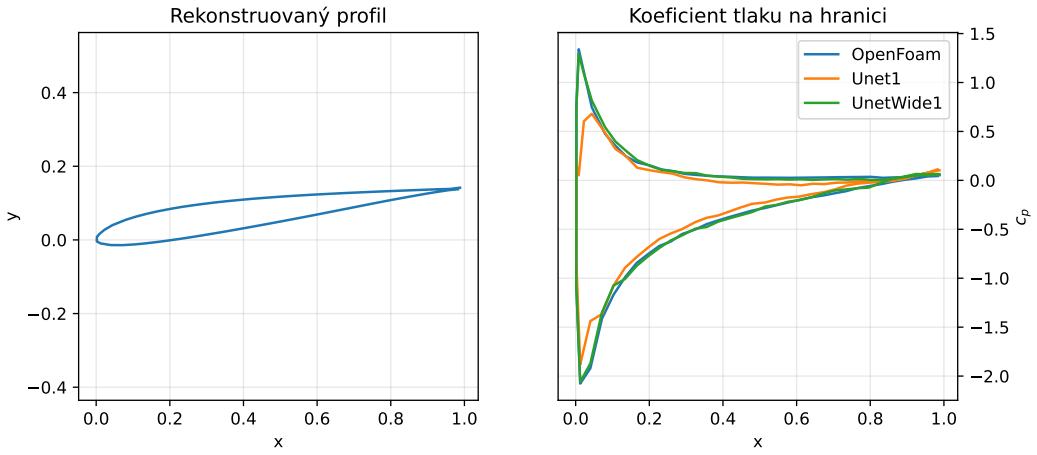
Na grafech 5.6 je porovnána predikce pole koeficientu tlaku s výsledkem z OpenFoam. Na první pohled data vypadají téměř totožně, ale při zaměření na barevnou škálu je vidět, že model "Unet1" nezvládá tak spolehlivě zachytit lokální přetlak a podtlak v oblasti náběžné hrany. Tento jev je zároveň vidět na obrázku 5.7 představující rozložení chyby **E** mezi daty z obrázku 5.6.



Obrázek 5.7: Rozložení chyby **E** z obrázku 5.6

Průběh c_p ineterpolovaný z predikce modelu "Unet1" na hranici (obr. 5.8) nedosahuje takových extrémů u náběžné hrany jako "správné" řešení získané z pole $\hat{\mathbf{O}}$ (modrá křivka). To je způsobeno pro model neznámým tvarem náběžné hrany, která má lehce odlišnou geometrii než u 4-ciferných NACA profilů. Přesto, že se jedná o jednu z horších predikcí, průběh c_p na hranici je až na lokální extrémy zachycen uspokojivě. Tento typ nepřesnosti je však daň za omezené zkušenosti z tréninkových dat. Na pravém grafu jsou navíc vyobrazeny data zelenou čarou a legendou "UnetWide1". Jedná se o koeficient tlaku c_p, který je získán interpolací z pole predikovaného modelem "UnetWide1". Zmíněný model má také stejné hyperparametry jako model "Unet1", pouze se liší tím, že je trénován na jiné tréninkové

množině vzorků. Jde o soubor (zmíněný v sekci 4) vybraných profilů z Airfoil databáze [21]. Díky tomu, že vybrané profily zahrnují širší škálu tvarů geometrie, model tak umí obecně lépe zachytit jev.

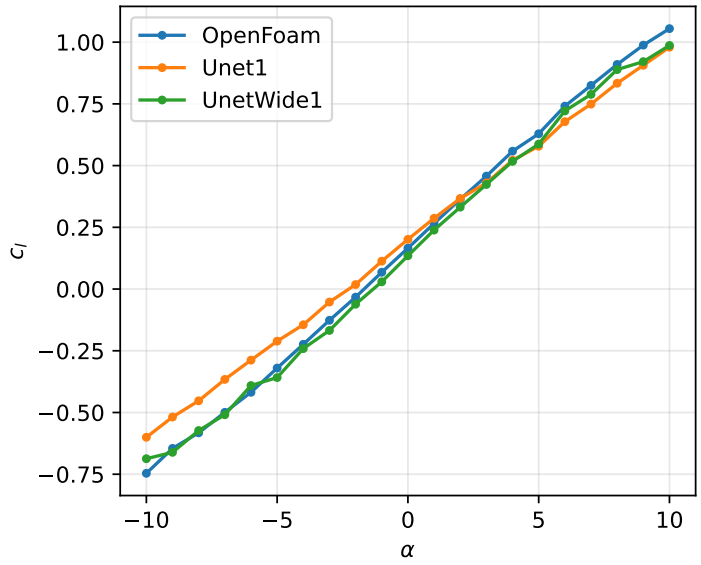


Obrázek 5.8: SD7003 $\alpha = -8°$, interpolace veličin na hranici

Zde jsou uvedeny koeficienty c_l vypočtené numerickou integrací pro tři vyobrazené případy:

$$c_l(\text{OpenFoam}) = -0.582$$
$$c_l(\text{UnetWide1}) = -0.573$$
$$c_l(\text{Unet1}) = -0.453$$

Na obrázku 5.9 jsou zobrazeny hodnoty koeficientů c_l získaných interpolací z polí koeficientu tlaku v závislosti na úhlu náběhu α. Model "UnetWide1" podle dostupných informací zvládl předpovídat výsledky pro tuto úlohu velmi spolehlivě a křivka kopíruje průběh získaný z polí z OpenFoam. Dokonce i předpověď modelu "Unet1", omezeného na 4-číselná NACA profily, se příliš nevzdaluje od požadovaného výsledku. Vzhledem k délce intervalu výskytu hodnot c_l je jeho maximální chyba kolem 10% a to při úhlu náběhu $\alpha = -10°$. Chyba je způsobena nedostatečným zachycováním extrémů c_p v oblasti náběžné hrany. Z celkových analýz pak vyplývá, že model "Unet1" pro profil SD7003 funguje relativně špatně a 10% chyba v koeficientu c_l se dá považovat vysokou.



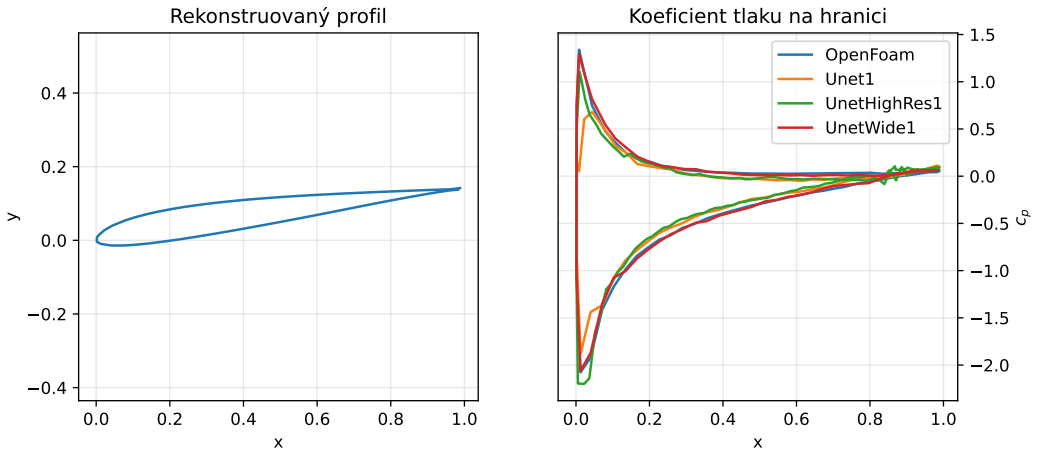
Obrázek 5.9: Závislost $c_l(\alpha)$ pro profil SD7003

Vzhledem k možným zdrojům chyb, jako např. hrubá diskretizace úlohy, predikce neuronové sítě, lineární interpolace atd., jsou takto přesné výsledky (i přes popsané nepřesnosti) překvapující.

V přílohách jsou uvedeny tyto grafické analýzy pro profily S826 a K3.

5.4 Analýza rozlišení

Výše uvedený problém vede k úvaze, zda podobný model trénovaný na 4-ciferných profilech jako "Unet1" s vyšším rozlišením zpracovávaných dat dokáže extrémy lépe zachytit. Tomuto popisu odpovídá model "UnetHighRes1", který pracuje s hodnotami vyčíslenými v bodech $P(128)$. Obrázek 5.10 tuto analýzu demonstruje.



Obrázek 5.10: interpolace c_p na hranici pro porovnání rozlišení ,profil SD7003, $\alpha = -8°$

Síť s vyšším rozlišením skutečně zachytla v tomto konkrétním případě lokální extrémy. Přesto je ale vidět, že výsledek není tak přesný, jako v případě modelu

"UnetWide1". Z tohoto lze usoudit, že použití širší škály geometrií profilů má v mnoha případech větší váhu, než hrubou silou zvyšovat rozlišení dat.

5.5 Vyhodnocení přesnosti na vybraných vzorcích

Poslední sekce obsahuje závěrečné vyhodnocení přesnosti predikcí některých zmíněných modelů. Cílem je poskytnout informaci, jak se predikce v širším měřítku liší od "správného" řešení $\hat{\mathbf{O}}$. V oblasti umělé inteligence lze velmi obtížně určit maximální nepřesnost, jakou lze od modelu očekávat. Obzvláště jedná-li se o případ regrese. Pokud by tato sekce takové informace obsahovala, byly by to velmi odvážné výroky. Je totiž téměř jisté, že by každý uživatel našel takové zadání, pro které by neuronová síť předpověděla výsledek s větší chybou, než by zde byla specifikovaná. Posudek o maximální chybě by bylo možné učinit, pokud by byly modely posouzeny na nekonečně mnoho (tj. na všech možných) vzorcích.

Proto je vyhodnocení omezeno na zmíněnou testovací datovou sadu, která obsahuje konečně mnoho vzorků (konkrétně 210). Prvním skalárním kritériem pro posouzení přesnosti modelů je již mnohokrát diskutována chybová funkce ξ. Zde jsou uvedeny hodnoty průměrné a maximální hodnoty ξ_{avg} a ξ_{max} (hodnoty uvedeny v řádu 10^{-5}).

$$\begin{aligned} \text{Unet1} : \xi_{avg} &= 24.8 \\ \xi_{max} &= 120 \end{aligned} \qquad \begin{aligned} \text{UnetWide1} : \xi_{avg} &= 17.1 \\ \xi_{max} &= 196.6 \end{aligned}$$

Hodnoty slouží spíše pro porovnání s tabulkou 4.1. V případě modelu "Unet1" jde o stejná čísla jako jsou uvedeny v odkazované tabulce. Znalost informace o hodnotě chyby je dobré znát např. za účelem porovnání oproti chybám při tréninku. Nicméně tato veličina nemá fyzikálně žádný význam. Navíc je vyčíslena na základě celého pole (resp. hodnot v bodech $P(q)$) koeficientu tlaku c_p a neříká nic o chování c_p v blízkosti hranice profilu a zachycení extrémů v těchto místech.

Z tohoto důvodu je vhodné znát také informaci o chybě v koeficientu c_l vypočteného z predikovaných polí. Jsou proto uvedeny průměrné a maximální odchylky (Δc_l^{avg} a Δc_l^{max}) od koeficientu c_l získaného z očekávaného řešení $\hat{\mathbf{O}}$.

$$\begin{aligned} \text{Unet1} : \Delta c_l^{avg} &= 0.027 \\ \Delta c_l^{max} &= 0.126 \end{aligned} \qquad \begin{aligned} \text{UnetWide1} : \Delta c_l^{avg} &= 0.022 \\ \Delta c_l^{max} &= 0.110 \end{aligned}$$

Poskytnutá data jednoznačně ukazují potřebu znalosti obou vyhodnocovaných kritérií. I přes větší maximální hodnotu chyby ξ_{max} v predikci modelu "UnetWide1", dochází k menší maximální odchylce Δc_l^{max} oproti predikcím modelu "Unet1". Je potřeba brát na vědomí, že úhly náběhu všech vyhodnocovaných vzorků se pohybují v hodnotách $\alpha \in \langle -10°, 10° \rangle$. Délka intervalu, ve kterém koeficient c_l v případě analyzovaných dat leží, je 1.82. Dá se tedy říci, že maximální odchylka Δc_l^{max} vztažená k délce intervalu je 6.9% pro model "Unet1" a 6% pro "UnetWide1". Průměrná

Δc_l^{avg} však činí hodnoty pro oba modely menší než 1.5%. Uvedené výsledky nepřesností jako cena za úsporu převážné většiny výpočetního času oproti numerickému řešení jsou velmi uspokojivé.

Závěr

Práce splňuje všechny zadané cíle. V prvních kapitolách jsou detailněji popsány základní matematické přístupy, díky kterým neuronové sítě poskytují predikce a jsou schopné tréninkem přizpůsobovat své parametry. Je formulována úloha obtékání profilu ve 2D a specifikovány numerické metody pro její řešení pomocí softwaru OpenFoam. Pomocí numerického řešiče jsou získána potřebná tréninková, validační a vybraná testovací data. Úloha je následně přeformulována pro konvoluční neuronovou síť. Práce navíc poskytuje použité přístupy pro zpracování vstupních a reprezentaci výstupních dat. Dílčí algoritmy jsou realizovány v programovacím jazyku Python. K urychlení procesů vyžadujících vysoký počet cyklů je použita implementace v Cythonu. Tvorba datových souborů a manipulace s daty je kompletně automatizována.

Je popsána a vysvětlena architektura, metodika tréninku a nastavení hyperparametrů použitých neuronových sítí, realizovaných pomocí knihovny PyTorch. Práce dokazuje, že pro formulovaný problém nestačí základní husté sítě a jsou potřeba výhodné vlastnosti konvolučních vrstev. Provedena je i optimalizace Unet konvoluční sítě hledáním ideální kombinace vybraných hyper-parametrů na tréninkové sadě 4-ciferných NACA profilů. Nejlepší z analyzovaných kombinací je dále použita i pro trénování dalšího modelu na tréninkové sadě vybraných profilů z airfoil databáze.

Poslední kapitola obsahuje vyhodnocení predikcí vybraných modelů pro neznámé externí profily. Analýzy pro vyhodnocení používají popsané algoritmy. Chyba v určení koeficientu vztlaku menší než 7% je překvapující vzhledem k potřebnému výpočetnímu času neuronové sítě oproti numerickým výpočtům. Daní za ušetřený čas je však kromě uvedené chyby také omezenost modelu na konkrétní nastavení výpočtu a geometrie profilu. Proto nelze říci, že je možné numerické výpočty aplikací neuronových sítí nahradit. Při správném použití a formulaci úlohy může však umělá inteligence inženýrům ušetřit spoustu práce a času. Například v případě optimalizace tvaru profilu pro dosažení požadovaných vlastností může být podobná neuronová síť, jako vytvořená v této práci, velmi užitečným nástrojem. Přesto je potřeba mít nad výsledky neuronových sítí určitý nadhled.

Je také třeba zvážit omezení této práce. Neuronové sítě mají relativně "malé" rozměry, protože jsou používány a trénovány na hardwaru, který je součástí běžně dostupných počítačů. Přesun na dedikovaný server by mohl nabídnout nepoměrně širší možnosti a prohloubit výhodnost použití umělé inteligence.

Náměty pro budoucí práce, navazující na tuto, by mohlo být např. rozšíření úlohy pro řešení proudového pole v lopatkové mříži turbíny, uvažování nestacionárního děje nebo rozšíření úlohy do 3D, což by vyžadovalo výkonnější hardware.

Bibliografie

1. HEATON, Jeff. *Applications of Deep Neural Networks*. 2020. Dostupné z arXiv: 2009.05673 [cs.LG].
2. HAYKIN, Simon S. *Neural networks and learning machines*. Third. Upper Saddle River, NJ: Pearson Education, 2009.
3. BOULILA, Wadii; DRISS, Maha; AL-SAREM, Mohammed; SAEED, Faisal; KRICHEN, Moez. *Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives*. 2021.
4. LIU, Bang-Gui; LIU, Yan. Image Classification for Dogs and Cats. In: 2014.
5. RONNEBERGER, Olaf; FISCHER, Philipp; BROX, Thomas. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv, 2015. Dostupné z DOI: 10.48550/ARXIV.1505.04597.
6. RIBEIRO, Mateus Dias; REHMAN, Abdul; AHMED, Sheraz; DENGEL, Andreas. *DeepCFD: Efficient Steady-State Laminar Flow Approximation with Deep Convolutional Neural Networks*. arXiv, 2020. Dostupné z DOI: 10.48550/ARXIV.2004.08826.
7. BUBLÍK, O. *Fast pressure prediction along the NACA airfoil using the convolution neural network*. Západočeská univerzita v Plzni, 2018. Dostupné také z: http://hdl.handle.net/11025/36269.
8. NIELSEN, Michael A. *Neural Networks and Deep Learning*. Determination Press, 2018. Dostupné také z: http://neuralnetworksanddeeplearning.com/.
9. SHARMA, Siddharth; SHARMA, Simone; ATHAIYA, Anidhya. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology*. 2020, roč. 04, s. 310–316. Dostupné z DOI: 10.33564/IJEAST.2020.v04i12.054.
10. PASZKE, Adam; GROSS, Sam; CHINTALA, Soumith; CHANAN, Gregory; YANG, Edward; DEVITO, Zachary; LIN, Zeming; DESMAISON, Alban; ANTIGA, Luca; LERER, Adam. Automatic differentiation in PyTorch. In: *NIPS-W*. 2017.
11. KRÖSE, Ben J. A.; SMAGT, Patrick P. van der. *An Introduction to Neural Networks*. Fourth. Amsterdam, The Netherlands: The University of Amsterdam, 1991.
12. DUMOULIN, Vincent; VISIN, Francesco. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*. 2016. Dostupné z eprint: 1603.07285.
13. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

14. LIU, Tianyi; FANG, Shuangsang; ZHAO, Yuehui; WANG, Peng; ZHANG, Jun. *Implementation of Training Convolutional Neural Networks*. 2015. Dostupné z arXiv: 1506.01195 [cs.CV].

15. BOUVRIE, Jake V. Notes on Convolutional Neural Networks. In: 2006.

16. WELLER, H.G.; TABOR, Gavin; JASAK, Hrvoje; FUREBY, Christer. A Tensorial Approach to Computational Continuum Mechanics Using Object Orientated Techniques. *Computers in Physics*. 1998, roč. 12, s. 620–631. Dostupné z DOI: 10.1063/1.168744.

17. MANGANI, Luca; BIANCHINI, Cosimo. United Kingdom HEAT TRANSFER APPLICATIONS IN TURBOMACHINERY. In: 2007.

18. NASA, [b.r.]. Dostupné také z: https://turbmodels.larc.nasa.gov/spalart.html.

19. *Construct2D*. 2007. Ver. v1.2.2. Dostupné také z: https://sourceforge.net/projects/construct2d/.

20. [B.r.]. Dostupné také z: https://m-selig.ae.illinois.edu/ads.html.

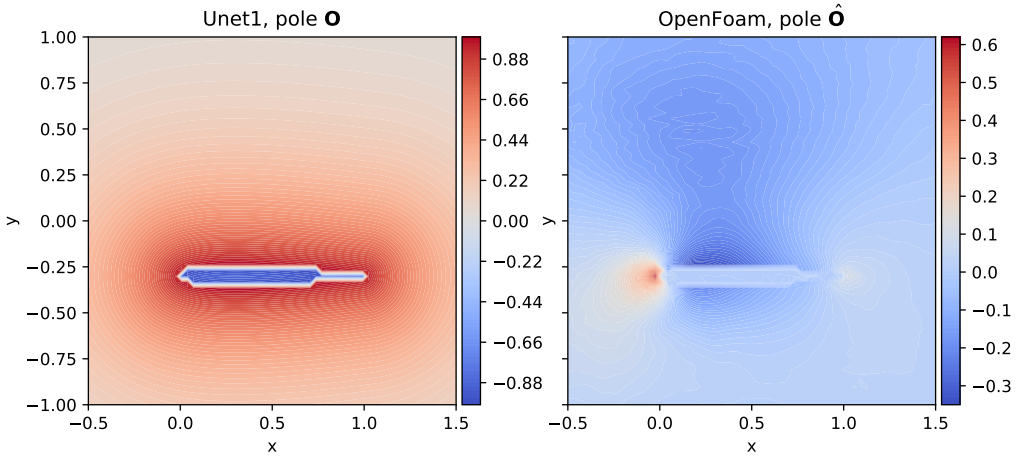
21. [B.r.]. Dostupné také z: http://airfoiltools.com/.

22. IOFFE, Sergey; SZEGEDY, Christian. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. Dostupné z arXiv: 1502.03167 [cs.LG].

Přílohy

A První příloha

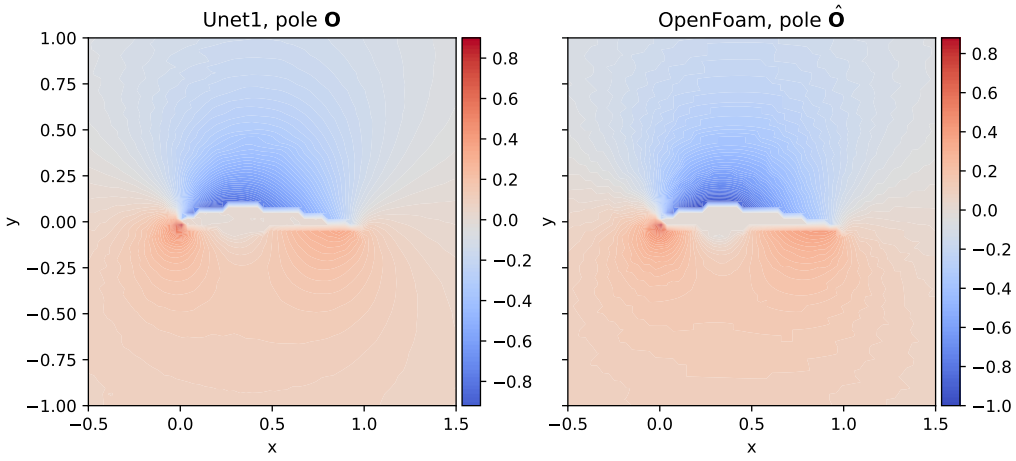
Predikce není věrohodná, ale konvoluční vrstvy umí přítomnost profilu zachytit.



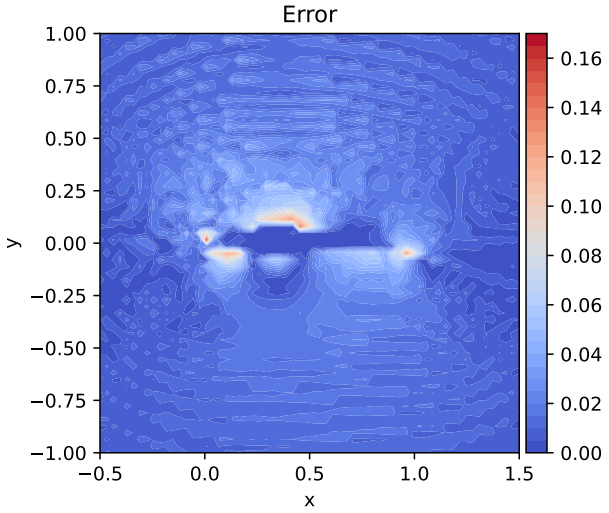
Obrázek 11: Predikce koeficientu tlaku pro posunutý profil NACA 0012, $\alpha = 0°$ (o $\Delta y = -0.3$)

B Druhá příloha

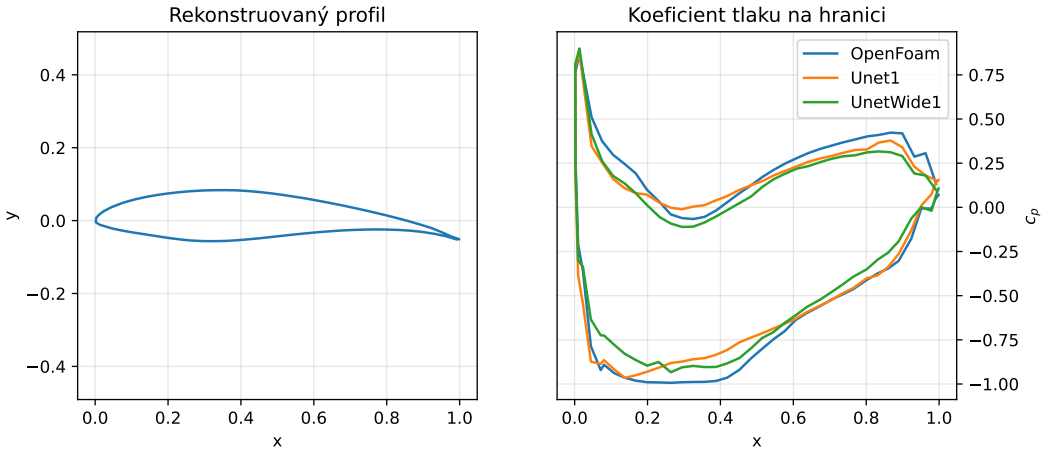
B.1 Profil S826



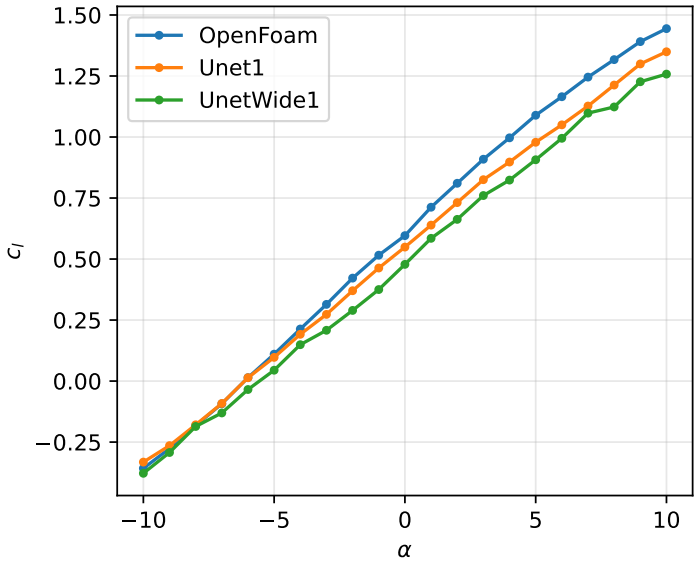
Obrázek 12: Porovnání predikce a výsledku z OpenFoam pro S826, $\alpha = 3°$



Obrázek 13: Rozložení chyby **E** z obrázku 12

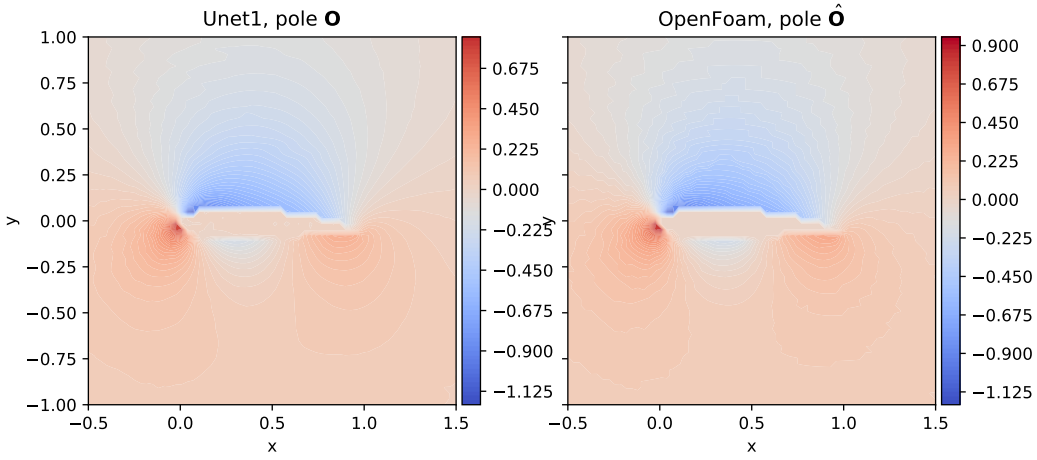


Obrázek 14: S826, $\alpha = 3°$, interpolace veličin na hranici

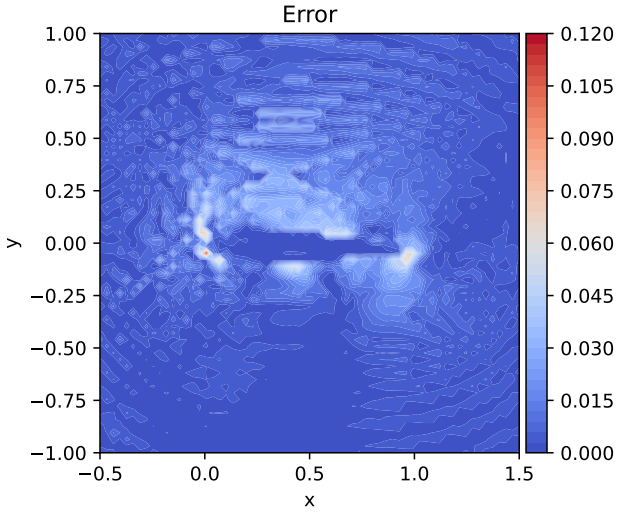


Obrázek 15: Závislost $c_l(\alpha)$ pro profil S826

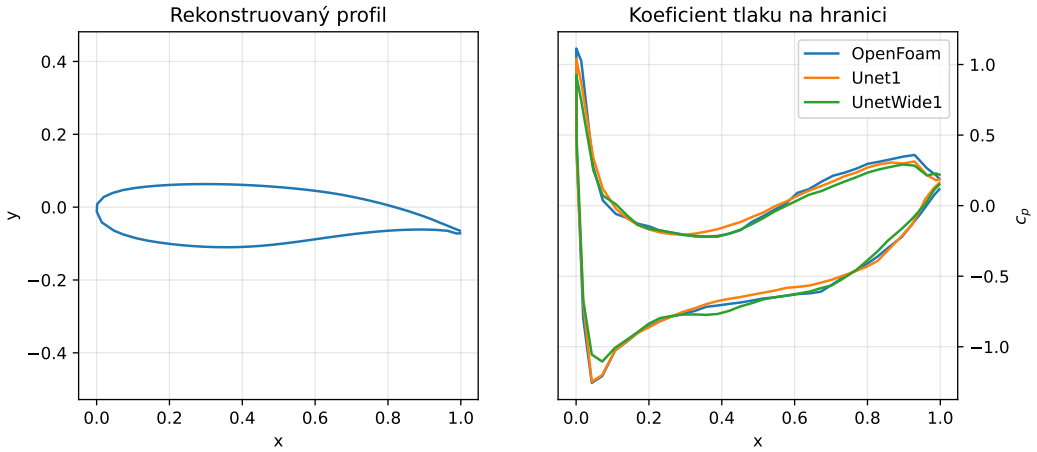
B.2 Profil K3



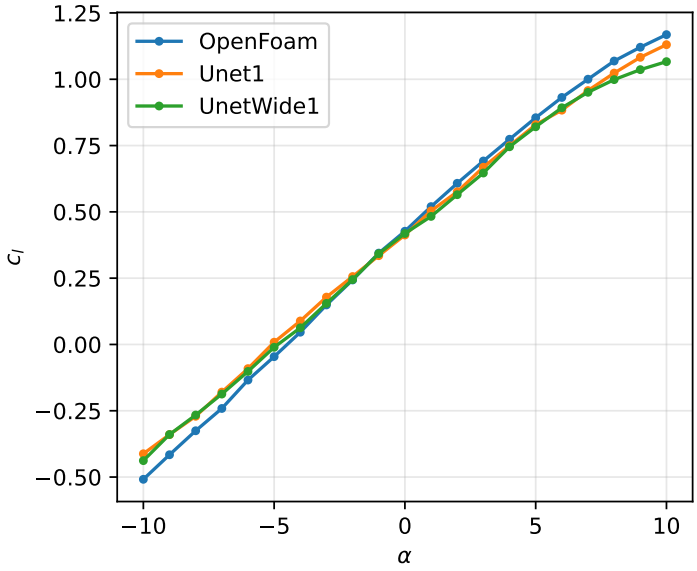
Obrázek 16: Porovnání predikce a výsledku z OpenFoam pro K3, $\alpha = 3°$



Obrázek 17: Rozložení chyby **E** z obrázku 16



Obrázek 18: K3, $\alpha = 3°$, interpolace veličin na hranici



Obrázek 19: Závislost $c_l(\alpha)$ pro profil K3