Střední průmyslová škola, Ústí nad Labem, Resslova 5

Správce neuronových sítí

Dokumentace k maturitní práci

Autor: Mykhailo Ivashchenko

Třída: 4ITC

Vedoucí práce: Ing. Petr Haberzettl

2025/2026

Prohlášení

Prohlašuji, že jsem ročníkovou práci na téma „Správce neuronových sítí“ vypracoval samostatně a s použitím uvedené literatury a pramenů.

V Ústí nad Labem dne 14.09.2025

……

Poděkování

Chtěl bych poděkovat Ing. Petru Haberzettlovi za vedení mé ročníkové práce, cenné rady a odborný dohled.

Anotace

V úvodu je popsán původní cíl projektu. Dokument dále obsahuje teoretickou část, která se dělí na rešerši a technologie. V rešerši je popsáno, čím byl projekt inspirován, a v části „technologie“ jsou popsány všechny technologie použité při vytvoření projektu. Následuje praktická část, která se skládá ze tří částí: návrh, produktizace a popis z pohledu uživatele. Návrh obsahuje screenshoty ze stránek. V produktizaci jsou ukázky kódu a popsané algoritmy. Popis z pohledu uživatele se zaměřuje na to, jak web vypádá z pohledu uživatele. V závěru je popsáno, co se skutečně podařilo vytvořit z původního cíle uvedeného v úvodu.

Klíčová slova

Neuronová síť, trénovací data, umělá inteligence, strojové učení, hluboké učení, zpracování obrazů

Obsah

Úvod

Hlavním cílem ročníkové práce je vytvořit webovou aplikaci sloužící jako správce neuronových sítí, která umožní uživatelům vytvářet, konfigurovat, učit a spravovat vlastní modely neuronových sítí v přehledném a uživatelsky přívětivém prostředí. Projekt je zaměřen na propojení teoretických principů umělé inteligence s praktickým využitím moderních technologií a má za cíl zpřístupnit práci s neuronovými sítěmi i méně zkušeným uživatelům.

Motivací k výběru tohoto tématu je rostoucí význam umělé inteligence a strojového učení v současném světě, zejména v oblasti zpracování obrazu, automatizace a analýzy dat. Neuronové sítě se dnes využívají v mnoha praktických aplikacích, avšak jejich používání bývá často spojeno s technickou složitostí a nutností práce v programovacím prostředí. Tento projekt si klade za cíl tuto bariéru snížit a nabídnout nástroj, který umožní experimentování s neuronovými sítěmi prostřednictvím webové aplikace bez nutnosti hlubokých znalostí programování. Zároveň je téma zvoleno z osobního zájmu o oblast umělé inteligence a její praktické využití.

Dalším cílem práce je návrh systému, který umožní práci s datovými sadami, jejich stahování, zpracování a opakované použití. Aplikace by měla podporovat ukládání projektů, modelů a datasetů tak, aby bylo možné navázat na předchozí práci bez nutnosti opakované konfigurace. Součástí cíle je také vizualizace průběhu učení neuronových sítí pomocí grafů, které uživateli poskytnou přehled o vývoji ztrátové funkce a celkové kvalitě učení modelu.

Projekt si dále klade za cíl umožnit nastavování základních parametrů učení, jako je volba optimalizátoru, ztrátové funkce nebo plánovače učící rychlosti, a nabídnout možnost rozšíření systému o pokročilejší modely, například pro detekci objektů v reálném čase. Aplikace by měla být navržena modulárně tak, aby bylo možné její další rozšiřování a úpravy.

Očekáváním od této práce je vytvoření funkčního a stabilního systému, který bude schopen demonstrovat celý proces práce s neuronovou sítí – od vytvoření projektu, přes přípravu dat, až po učení modelu a vyhodnocení výsledků. Výsledná aplikace by měla sloužit nejen jako praktický nástroj, ale také jako vzdělávací pomůcka, která usnadní pochopení principů neuronových sítí a strojového učení.

1 Teoretická část

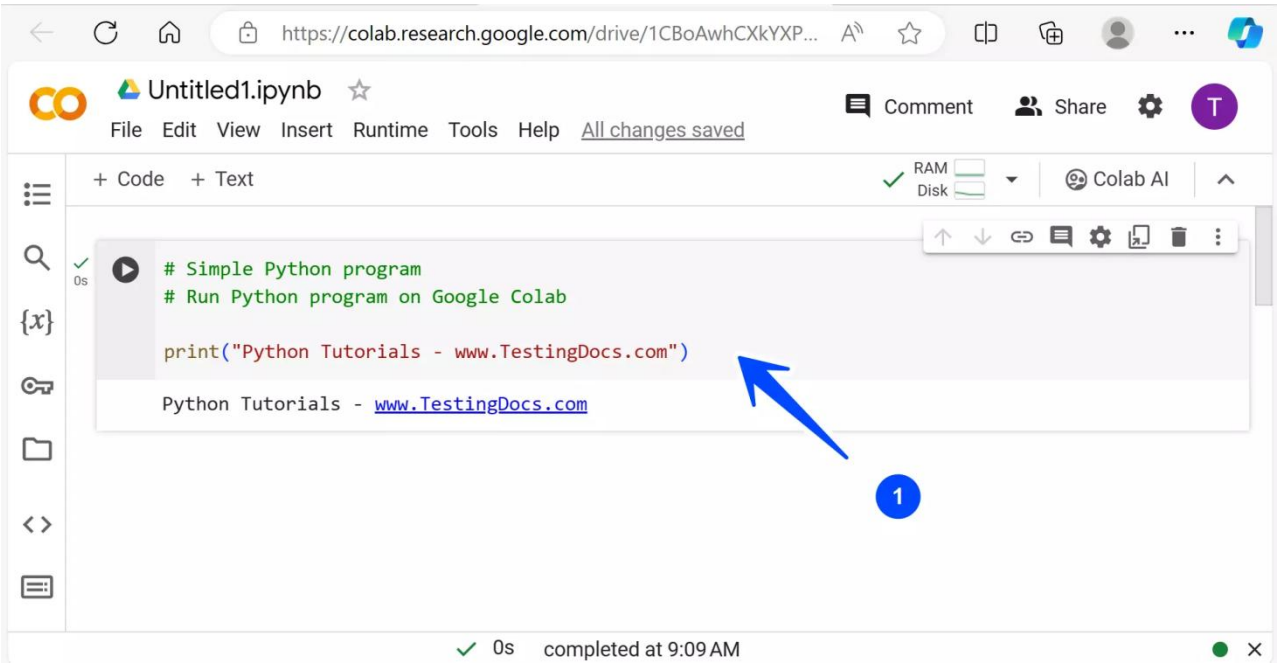
1.1 Rešerše

Sem napište text vaší rešerše.

1.1.1 Google Colab

Google Colab je webová platforma umožňující spouštění Python kódu přímo v prohlížeči bez nutnosti lokální instalace vývojového prostředí. Je často využívána při výuce strojového učení, protože nabízí snadný přístup k výpočetním prostředkům a podporuje populární knihovny, jako je PyTorch nebo TensorFlow.

Inspirací pro tento projekt byla především jednoduchost práce v prostředí, kde se uživatel může soustředit na samotné experimentování s modely a nemusí řešit technické detaily instalace či konfigurace systému. Google Colab rovněž umožňuje ukládání a opětovné načítání výsledků práce, což je důležitý aspekt i pro navrhovanou aplikaci. (1)



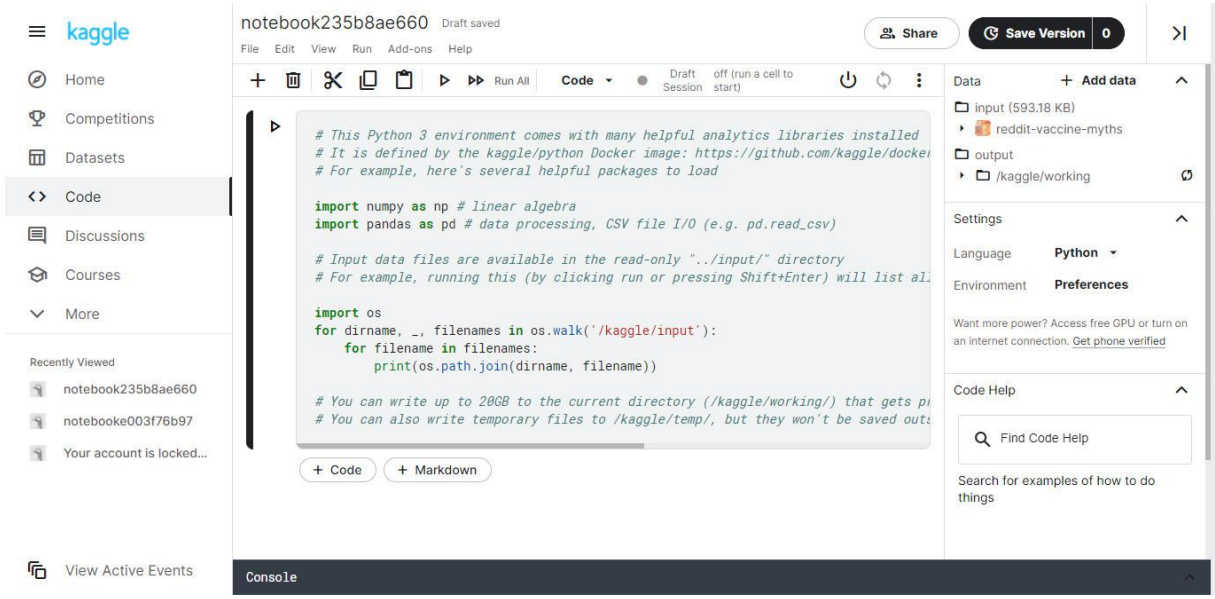
Obrázek 1 - Prostředí Google Colab

1.1.2 Kaggle

Platforma Kaggle je známá především jako prostředí pro datové soutěže a práci s datovými sadami. Nabízí velké množství veřejně dostupných datasetů a nástroje pro jejich zpracování a analýzu. Pro uživatele je výhodné, že má na jednom místě data, kód i možnost vizualizace výsledků.

V rámci této práce byl Kaggle inspirací zejména v oblasti správy datových sad. Myšlenka mít dataset uložený na jednom místě a být schopen jej opakovaně využívat bez nutnosti neustálého

ručního nastavování byla jedním z důvodů, proč je správa datasetů důležitou součástí navrhované aplikace. (2)



Obrázek 2 - Prostředí Kaggle

1.1.3 Hugging Face

Knihovna Hugging Face Datasets poskytuje standardizované rozhraní pro práci s velkým množstvím datasetů určených pro strojové učení. Umožňuje jejich snadné stahování, načítání a předzpracování.

V tomto projektu byla knihovna využívána především jako zdroj inspirace pro strukturovaný přístup k práci s daty. Důraz je kladen na to, aby uživatel nemusel opakovaně psát stejný kód pro načítání a zpracování datasetů, ale mohl se soustředit na samotné učení neuronových sítí. (3)

1.2 Technologie

1.2.1 Python

Python je vysokoúrovňový programovací jazyk, který je широко využíván v oblasti datové analýzy, strojového učení a vývoje backendových aplikací. V rámci projektu je Python použit jako hlavní jazyk serverové části aplikace. Slouží k implementaci logiky trénování neuronových sítí, zpracování datasetů a komunikace mezi jednotlivými službami systému. Volba Pythonu byla ovlivněna především autorovou předchozí zkušeností s tímto jazykem a jeho silnou podporou v oblasti AI. (4)

1.2.2 Py Torch

PyTorch je knihovna pro strojové učení a hluboké neuronové sítě, vyvinutá s důrazem na flexibilitu a práci s dynamickými výpočetními grafy. V projektu je PyTorch využit pro návrh architektury neuronových sítí, jejich trénování a vyhodnocování výsledků. Volba této knihovny vychází z praktických zkušeností autora a její časté využití v akademickém i průmyslovém prostředí. (5)

1.2.3 Docker

Docker je nástroj pro kontejnerizaci aplikací, který umožňuje spouštět software v izolovaném a reprodukovatelném prostředí. V projektu je Docker použit k zajištění konzistentního běhu jednotlivých služeb (např. backendu a databáze) bez ohledu na konkrétní operační systém. Díky tomu je možné aplikaci snadno nasadit a spustit i na jiném zařízení. (6)

1.2.4 Alembic

Alembic je nástroj pro správu verzí databázového schématu.
V rámci projektu slouží k řízení změn struktury databáze, například při přidávání nových tabulek nebo úpravě existujících polí. Tím je zajištěna konzistence databáze během vývoje aplikace. (7)

1.2.5 PostgreSQL

PostgreSQL je relační databázový systém s otevřeným zdrojovým kódem.
V projektu je využit pro ukládání informací o projektech, nastaveních trénování, datech a výsledcích experimentů. PostgreSQL byl zvolen díky své stabilitě, rozšířeným možnostem a častému využití v serverových aplikacích. (8)

1.2.6 Redis

Redis je databázový systém typu key-value, optimalizovaný pro rychlou práci s daty v operační paměti. V tomto projektu je Redis využíván pro ukládání autentizačních tokenů uživatelů. Díky tomu je možné efektivně spravovat přihlášení, ověřování uživatelů a případné odhlašování bez nutnosti častého přístupu k hlavní relační databázi. Použití Redis přispívá ke zvýšení výkonu a bezpečnosti systému, zejména při práci s časově omezenými přístupovými tokeny. (9)

1.2.7 React

React je knihovna pro tvorbu uživatelského rozhraní webových aplikací. V tomto projektu je React použit k vytvoření webového rozhraní, které umožňuje uživateli nastavovat parametry projektu, vybírat dataset, volit ztrátovou funkci a sledovat průběh trénování modelu. React byl zvolen kvůli přehledné správě stavu aplikace a modulárnímu přístupu k tvorbě rozhraní. (10)

1.2.8 ChatGPT

ChatGPT je jazykový model založený na umělé inteligenci. V rámci projektu byl využíván jako podpůrný nástroj při návrhu architektury aplikace, ladění kódu a formulaci textové dokumentace.

ChatGPT nebyl použit jako součást výsledné aplikace, ale pouze jako pomocný prostředek během vývoje. (11)

2 Praktická část

Praktická část práce se zaměřuje na implementaci distribuované aplikace pro správu a trénování neuronových sítí.

2.1 Návrhy

Na začátku projektu bylo nutné navrhnout celkovou architekturu aplikace. Cílem bylo vytvořit modulární a přehledné řešení, které umožní snadné rozšiřování o další funkce v budoucnu. Aplikace je rozdělena na backendovou a frontendovou část, které spolu komunikují pomocí rozhraní API.

Backendová část byla navržena jako sada samostatných služeb, zodpovědných za správu projektů, uživatelů, datasetů a procesu učení neuronových sítí. Důraz byl kladen na oddělení logiky učení modelů od uživatelského rozhraní. Pro ukládání dat byl navržen relační databázový model, který zahrnuje entity jako projekt, uživatel, dataset a konfigurace učení.

Obrázek 3 - Seznam jednotlivých služeb

Databázový kontejner PostgreSQL obsahuje relační databázi s tabulkami určenými pro ukládání uživatelských dat, informací o projektech a jejich konfiguracích. Tato databáze slouží jako hlavní perzistentní úložiště aplikace.

Služba Redis je využita jako paměťové úložiště pro autentizační tokeny. Tyto tokeny umožňují ověřování uživatelů bez nutnosti opakovaného přístupu do databáze, čímž se zvyšuje výkon a bezpečnost systému.

DB-service představuje nadřazený servis nad databází PostgreSQL. Zajišťuje komunikaci s databází prostřednictvím API endpointů a abstrahuje databázovou logiku od ostatních částí systému. Ostatní služby tak s databází komunikují výhradně přes tento servis.

User service je zodpovědná za uživatelskou logiku aplikace. Zajišťuje registraci, přihlášení a odhlášení uživatelů, správu uživatelských profilů a práci s autentizačními tokeny.

Frontendový kontejner obsahuje uživatelské rozhraní aplikace, tvořené HTML stránkami, CSS styly a JavaScriptovými skripty. Tato část aplikace umožňuje uživateli interakci se systémem, nastavování projektů a sledování průběhu učení neuronových sítí.

Služba Projects manager spravuje všechny uživatelské projekty a jejich konfigurace. Je zodpovědná za řízení procesu učení neuronových sítí, včetně spouštění trénování, ukládání výsledků a sledování průběhu učení.

Datasets manager zajišťuje práci s trénovacími daty. Umožňuje stahování datasetů, jejich zpracování, přípravu pro učení neuronových sítí a jejich ukládání pro opakované použití.

2.2 Produktizace

2.2.1 Celkový popis

Aplikace je navržena jako systém mikroslužeb, které jsou provozovány v samostatných kontejnerech pomocí technologie Docker. Tento přístup umožňuje oddělení jednotlivých funkčních částí systému, jejich nezávislé nasazování a jednodušší údržbu. Pro orchestraci služeb je využit nástroj Docker Compose, který definuje závislosti mezi službami, jejich konfiguraci a způsob spuštění.

Databázový kontejner PostgreSQL obsahuje relační databázi s tabulkami určenými pro ukládání uživatelských dat, informací o projektech a jejich konfiguracích. Databáze slouží jako hlavní perzistentní úložiště celé aplikace.

Služba Redis je využita jako paměťové úložiště pro autentizační tokeny. Tyto tokeny umožňují ověřování uživatelů bez nutnosti opakovaného přístupu do databáze, čímž se snižuje zátěž databázového systému a zvyšuje se výkon aplikace.

Služba DB-service představuje aplikační vrstvu nad databází PostgreSQL. Zajišťuje komunikaci s databází prostřednictvím API endpointů a abstrahuje databázovou logiku od ostatních částí systému. Ostatní služby s databází komunikují výhradně prostřednictvím tohoto rozhraní.

User service je zodpovědná za uživatelskou logiku aplikace. Zajišťuje registraci, přihlášení a odhlášení uživatelů, správu uživatelských profilů a práci s autentizačními tokeny.

Frontendový kontejner obsahuje uživatelské rozhraní aplikace tvořené HTML stránkami, CSS styly a JavaScriptovými skripty. Umožňuje uživateli interakci se systémem, nastavování projektů a sledování průběhu učení neuronových sítí.

Služba Projects manager spravuje uživatelské projekty a jejich konfigurace. Je zodpovědná za řízení procesu učení neuronových sítí, včetně spouštění trénování, ukládání výsledků a sledování průběhu učení.

Datasets manager zajišťuje práci s trénovacími daty. Umožňuje stahování datasetů, jejich zpracování, přípravu pro učení neuronových sítí a jejich ukládání pro opakované použití.

2.2.2 Databazový model

Následující část popisuje databázový model aplikace, který je realizován pomocí ORM knihovny SQLAlchemy. Datový model definuje základní entity systému a jejich vzájemné vztahy, zejména uživatele, projekty a datasety používané pro učení neuronových sítí.

```python
from enum import Enum
from sqlalchemy import Column, Integer, String, Date, DateTime, func,
ForeignKey, Enum as EnumType, Text
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

class DataType(Enum):
    IMAGE = "image"
    VECTOR = "vector"

class LossType(Enum):
    CROSS_ENTROPY = "CrossEntropyLoss"
    MSE = "MSELoss"
    L1 = "L1Loss"
    SMOOTH_L1 = "SmoothL1Loss"

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(50), nullable=False)
    surname = Column(String(50), nullable=False)
    username = Column(String(50), unique=True, nullable=False)
    password_hash = Column(String(255), nullable=False)
    born_date = Column(Date, nullable=False)
    bio = Column(String(100), default="")
    registration_date = Column(DateTime(timezone=True),
server_default=func.now())

    projects = relationship("Project", back_populates="owner")

class Project(Base):
    __tablename__ = "projects"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    description = Column(String(255), default="")
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    owner_username = Column(String(50), ForeignKey("users.username"),
nullable=False)
    input_type = Column(EnumType(DataType), nullable=False)
    output_type = Column(EnumType(DataType), nullable=False)
    architecture_json = Column(Text(), nullable=True)
    dataset_id = Column(Integer, ForeignKey("datasets.id"), nullable=True)
    dataset_preprocess_json = Column(Text(), nullable=True)

    optimizer_json = Column(Text(), nullable=True)
    scheduler_json = Column(Text(), nullable=True)
    loss_function = Column(EnumType(LossType), nullable=True,
default=LossType.MSE)

    dataset = relationship("Dataset")
```

```
    owner = relationship("User", back_populates="projects")

class Dataset(Base):
    __tablename__ = "datasets"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    description = Column(String(255), default="")
    created_at = Column(DateTime(timezone=True), server_default=func.now())
    storage_id = Column(String(128), unique=True, nullable=False)
    dataset_type = Column(String(50), nullable=False, default="unknown")

    owner_id = Column(Integer, ForeignKey("users.id"), nullable=False)
```

Kód 1 - Struktura databáze

Pro práci s databází je využit objektově-relační mapovací přístup (ORM), který umožňuje reprezentovat databázové tabulky jako objektové třídy v programovacím jazyce Python. Tento přístup zjednodušuje práci s databází, zvyšuje čitelnost kódu a snižuje riziko chyb při ručním psaní SQL dotazů.

Třídy `DataType` a `LossType` představují výčtové typy (Enum), které slouží k omezení možných hodnot některých atributů projektu.

Typ `DataType` určuje formát vstupních a výstupních dat neuronové sítě (např. obrazová nebo vektorová data). Typ `LossType` definuje typ ztrátové funkce použité během procesu učení neuronové sítě. Použití výčtových typů zajišťuje konzistenci dat a zabraňuje ukládání neplatných hodnot do databáze.

Entita `User` reprezentuje registrovaného uživatele systému. Obsahuje základní identifikační údaje, přihlašovací informace a doplňující metadata, jako je datum registrace nebo krátký popis profilu.

Každý uživatel může vlastnit více projektů, což je realizováno pomocí relační vazby typu *one-to-many* mezi entitami `User` a `Project`.

Entita `Project` představuje základní pracovní jednotku aplikace. Každý projekt odpovídá jedné konfiguraci neuronové sítě a jejímu procesu učení.

Projekt obsahuje informace o typu vstupních a výstupních dat, architektuře sítě, použitých optimalizačních metodách a ztrátové funkci. Konfigurační části jsou ukládány ve formátu JSON, což umožňuje flexibilní ukládání různých architektur bez nutnosti měnit databázové schéma.

Entita `Dataset` slouží k evidenci trénovacích dat používaných v projektech. Obsahuje základní popis datasetu a identifikátor fyzického úložiště, kde jsou data uložena.

Dataset je přiřazen konkrétnímu uživateli a může být opakovaně využíván v různých projektech, což podporuje efektivní práci s daty a jejich znovupoužitelnost.

Navržený databázový model reflektuje modulární architekturu celé aplikace a umožňuje její další rozšiřování bez zásadních zásahů do stávající struktury.

2.2.3 Učení neuronových sítí

Aplikace poskytuje modul pro řízení procesu učení neuronových sítí. Hlavní logiku trénování zajišťují dvě metody: `go_epochs` a `go_steps`. Obě metody umožňují iterativní optimalizaci váh modelu s využitím gradientního sestupu a umožňují monitorovat průběh trénování v reálném čase.

```
    def go_epochs(self, data_loader, project_id, epochs=1.0, device="cpu"):
        self.model.to(device)
        self.model.train()

        steps_per_epoch = len(data_loader)
        total_steps = int(epochs * steps_per_epoch)

        loader = infinite_loader(data_loader)

        for step in range(total_steps):
            inputs, targets = next(loader)
            inputs, targets = inputs.to(device), targets.to(device)

            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.criterion(outputs, targets)
            loss.backward()
            self.optimizer.step()

            self.losses.append(loss.item())

            print(f"Step [{step+1}/{total_steps}] Loss: {loss.item():.4f}")

            if self.scheduler and not isinstance(self.scheduler,
ReduceLROnPlateau):
                self.scheduler.step()

            TRAINING_PROGRESS[project_id] = {
                "status": "running",
                "current": step / steps_per_epoch,
                "total": epochs,
                "loss": loss.item()
            }

        if self.scheduler and isinstance(self.scheduler,
ReduceLROnPlateau):
            self.scheduler.step(loss)

        TRAINING_PROGRESS[project_id] = {
            "status": "finished",
            "current": total_steps / steps_per_epoch,
            "total": epochs,
            "loss": loss.item()
        }

    def go_steps(self, data_loader, project_id, steps=1000, device="cpu"):
        self.model.to(device)
        self.model.train()
```

```
        loader = infinite_loader(data_loader)

        for step in range(steps):
            inputs, targets = next(loader)
            inputs, targets = inputs.to(device), targets.to(device)

            self.optimizer.zero_grad()
            outputs = self.model(inputs)
            loss = self.criterion(outputs, targets)
            loss.backward()
            self.optimizer.step()

            self.losses.append(loss.item())

            print(f"Step [{step+1}/{steps}] Loss: {loss.item():.4f}")

            if self.scheduler and not isinstance(self.scheduler,
ReduceLROnPlateau):
                self.scheduler.step()

            TRAINING_PROGRESS[project_id] = {
                "status": "running",
                "current": step,
                "total": steps,
                "loss": loss.item()
            }

        if self.scheduler and isinstance(self.scheduler,
ReduceLROnPlateau):
            self.scheduler.step(loss)

        TRAINING_PROGRESS[project_id] = {
            "status": "finished",
            "current": steps,
            "total": steps,
            "loss": loss.item()
        }
```

Kód 2 - Trenování neuronové sítí

Metoda `go_epochs` slouží k trénování modelu po zadaný počet epoch. Model je nejprve přesunut na vybraný výpočetní device, například CPU nebo GPU, a přepnut do trénovacího režimu. Data jsou načítána z datového loaderu v nekonečné smyčce, aby bylo možné provést libovolný počet epoch bez nutnosti ručního resetování loaderu. Pro každou dávku dat se nejprve vymažou předchozí gradienty, následně se spočítají výstupy modelu, vyhodnotí ztrátová funkce a provede se zpětná propagace gradientů. Parametry modelu jsou poté aktualizovány pomocí optimalizátoru. V průběhu trénování se průběžně ukládají hodnoty ztrátové funkce a stav projektu do globální struktury `TRAINING_PROGRESS`, což umožňuje sledování aktuálního postupu trénování. Scheduler učící rychlosti, pokud je použit, je volán buď po každém kroku, nebo po každé epoše, v závislosti na jeho typu. Po dokončení všech epoch je stav projektu aktualizován na „finished“, čímž se signalizuje dokončení trénování.

Metoda `go_steps` poskytuje alternativní způsob trénování, kdy je počet kroků explicitně určen, nezávisle na počtu epoch. Princip jejího fungování je obdobný jako u `go_epochs`: model je přesunut na požadovaný device a přepnut do trénovacího režimu, data jsou načítána v nekonečné smyčce, a pro každou dávku se provádí výpočet výstupu, vyhodnocení ztráty, zpětná propagace gradientů a aktualizace parametrů. Během trénování je rovněž sledován průběh prostřednictvím `TRAINING_PROGRESS`, který zaznamenává aktuální krok, celkový počet kroků a hodnotu ztráty. Scheduler učící rychlosti je volán podle typu, stejně jako u metody `go_epochs`. Po dosažení zadaného počtu kroků je stav projektu nastaven na „finished“, což signalizuje ukončení trénování.

2.2.4 Neuronové vrstvy

V aplikaci je implementován modul pro definici šablon neuronových vrstev, které lze využít při sestavování architektury modelu. Konstantní objekt `LAYER_TEMPLATES` obsahuje předdefinované vrstvy, jako jsou lineární vrstvy, konvoluční vrstvy, aktivace typu ReLU, LeakyReLU, Sigmoid či Tanh, normalizační vrstvy typu BatchNorm a LayerNorm, poolingové vrstvy a speciální bloky, například SEBlock, ResidualBlock nebo ConvolutionalAttention. Každá šablona obsahuje výchozí parametry, například počet vstupních a výstupních kanálů, velikost jádra, krok nebo parametr redukce, což umožňuje rychlé vytváření vrstev bez nutnosti manuálního nastavování všech atributů.

Takový přístup zajišťuje modulární a flexibilní architekturu, kdy je možné snadno experimentovat s různými konfiguracemi modelu, přidávat nové vrstvy či měnit jejich parametry bez zásahu do základního kódu. Šablony rovněž zahrnují operace pro změnu rozměrů dat, jako jsou flatten, upsampling nebo pixel shuffle, což podporuje tvorbu komplexních sítí s různými topologiemi. Tento systém umožňuje dynamicky sestavovat modely podle potřeb jednotlivých projektů a experimentů, čímž se zvyšuje efektivita práce s neuronovými sítěmi a minimalizuje riziko chyb při ruční konfiguraci vrstev.

```
const LAYER_TEMPLATES = {
  Linear: { in_features: 1, out_features: 1 },
  Conv2D: { in_features: 1, out_features: 8, kernel_size: 3, stride: 1,
padding: 1 },
  DSConv2D: { in_features: 1, out_features: 8, kernel_size: 3, stride: 1,
padding: 1 },
  ReLU: { in_features: 1, out_features: 8 },
  LeakyReLU: { in_features: 1, out_features: 8, alpha: 0.01 },
  PReLU: { in_features: 1, out_features: 8 },
  Sigmoid: { in_features: 1, out_features: 8 },
  Tanh: { in_features: 1, out_features: 8 },
  Softmax: { in_features: 1, out_features: 8 },
  BatchNorm1d: { in_features: 1, out_features: 8 },
  BatchNorm2d: { in_features: 1, out_features: 8 },
  LayerNorm: { in_features: 1, out_features: 8 },

  Conv2DTranspose: { in_features: 1, out_features: 8, kernel_size: 2,
stride: 1, padding: 0, scale_factor: 2 },
  UpscaleBlock: { in_features: 1, out_features: 8, scale_factor: 2, mode:
"nearest", kernel_size: 3, padding: 1 },
  Upsample: { in_features: 1, out_features: 8, scale_factor: 2, mode:
"nearest" },
  MaxPool2D: { in_features: 1, out_features: 8, kernel_size: 2, stride: 2
},
```

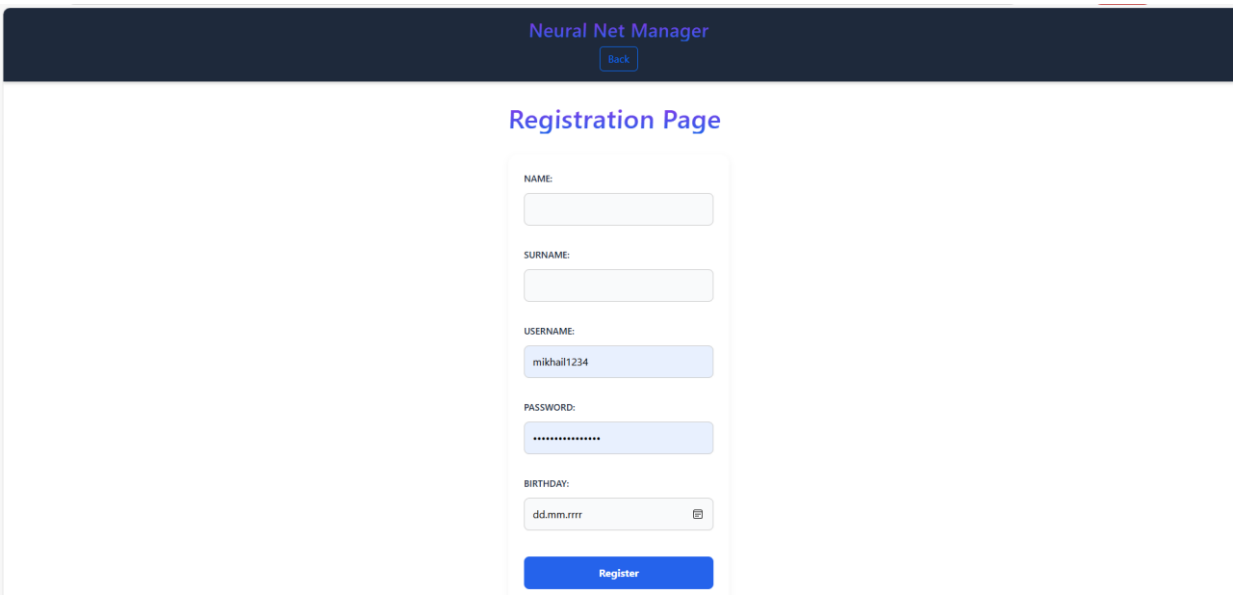
```
  AvgPool2D: { in_features: 1, out_features: 8, kernel_size: 2, stride: 2
},
  ConvolutionalAttention: { in_features: 1, out_features: 8, num_heads: 4},
  SEBlock: { in_features: 1, out_features: 8, reduction: 16 },
  ResidualBlock: { in_features: 1, out_features: 8, kernel_size: 3, stride:
1, padding: 1, depth: 2 },
  InstanceNorm1D: { in_features: 1, out_features: 8},
  InstanceNorm2D: { in_features: 1, out_features: 8},
  GroupNorm: { in_features: 1, out_features: 8, num_groups: 1 },
  Dropout: { in_features: 1, out_features: 8, p: 0.5 },
  PixelShuffle: { in_features: 8, out_features: 8, upscale_factor: 1 },

  // Reshapes
  Flatten: { in_features: 1, out_features: 1 },
  GlobalAveragePool2D: { in_features: 1, out_features: 1 },
};
```

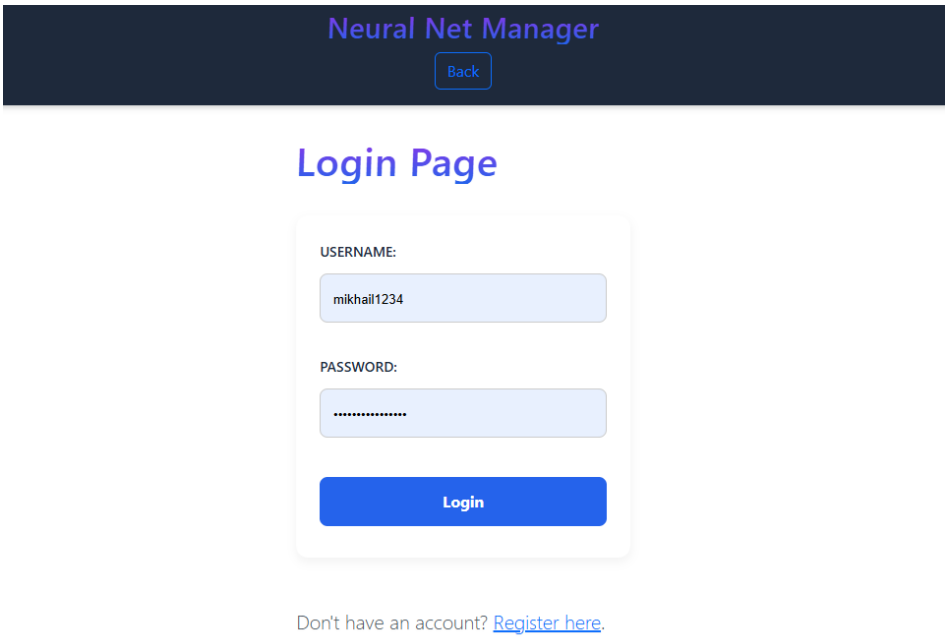
Kód 3 - Definice seznamu vrstev neuronových sítí

2.3 Popis pro uživatele

Pro práci se systémem se uživatel nejprve musí zaregistrovat. Uživatel vyplní své základní údaje, jako je jméno, příjmení, datum narození a unikátní uživatelské jméno. Po zadání hesla a potvrzení registrace systém vytvoří účet a uloží informace do databáze. Po úspěšné registraci uživatel vidí potvrzení o vytvoření účtu a je připraven přihlásit se do aplikace.



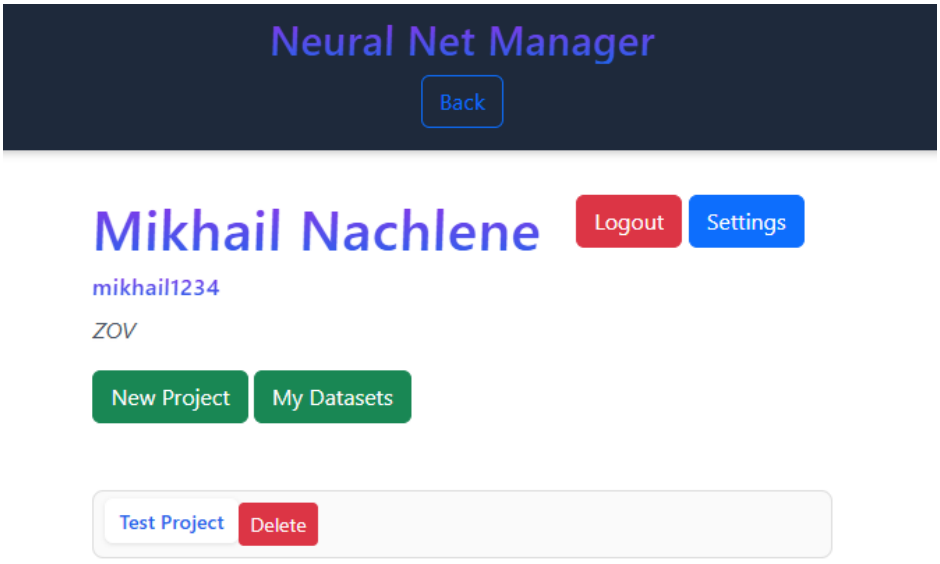
Obrázek 4 - Registrační stránka



Obrázek 5 - Příhlašovácí stránka

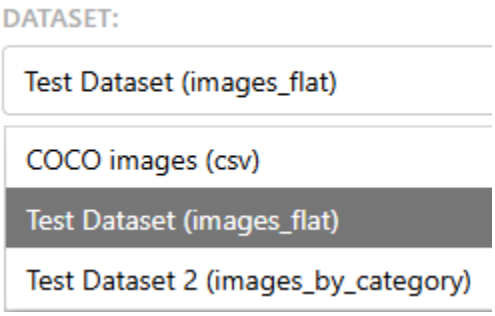
Pro přihlášení musí uživatel zadat své uživatelské jméno a heslo. Po ověření údajů systém vygeneruje autentizační token, který umožňuje bezpečný přístup ke všem funkcím aplikace bez opakovaného zadávání přihlašovacích údajů. Uživatel vidí informační zprávu o úspěšném přihlášení a je přesměrován na hlavní stránku, kde může spravovat své projekty, sledovat trénování modelů a pracovat s datovými sadami. V případě nesprávného uživatelského jména nebo hesla systém zobrazí chybovou hlášku a umožní uživateli opakovat přihlášení.

Po úspěšném přihlášení se uživatel dostává na hlavní stránku aplikace, nazvanou „Neural Net Manager“. Uživatel vidí své jméno a uživatelské jméno, což potvrzuje, že je přihlášen do správného účtu. Na hlavní stránce jsou zobrazeny všechny existující projekty uživatele. Stránka poskytuje intuitivní rozhraní pro rychlý přístup k projektům, umožňuje přidávání nových projektů, správu datasetů a sledování průběhu učení neuronových sítí.



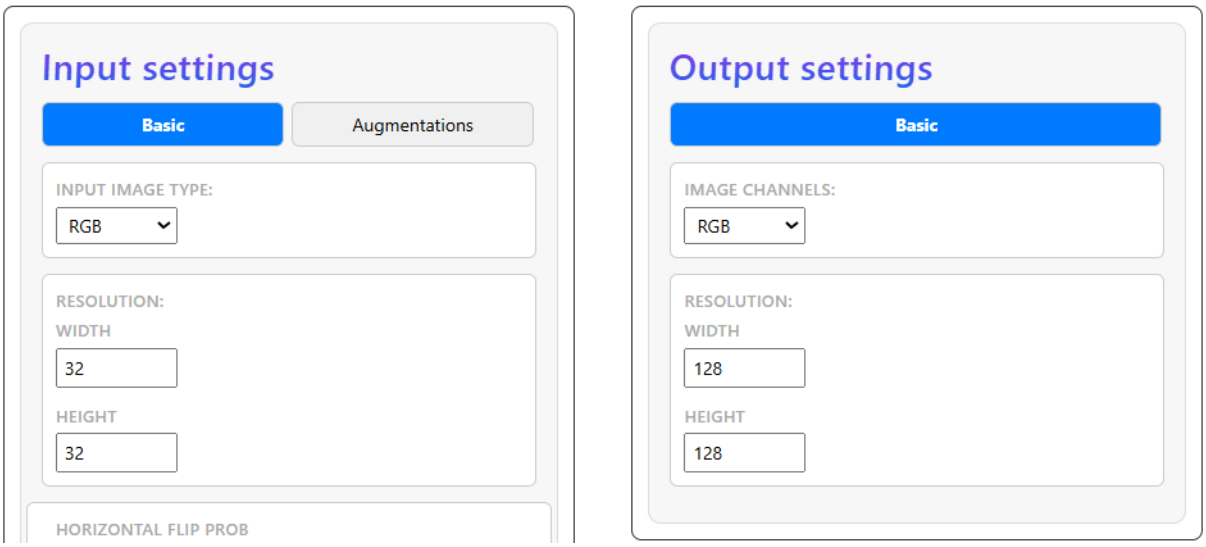
Obrázek 6 - Hlavní stránka

Uživatel má k dispozici ovládací prvky pro vytvoření nového projektu, spuštění trénování nebo zobrazení detailů stávajícího projektu. Každý projekt je vizualizován jako samostatná položka, která zobrazuje základní informace a stav procesu učení. Uživatel může vybrat konkrétní projekt a přejít na stránku s detailními nastaveními, kde má možnost upravovat konfiguraci modelu, přidávat datové sady a monitorovat metriky trénování.



Obrázek 7 - Seznám datasetů, ze kterých uživatel si může výbrat

Součástí uživatelského rozhraní je rozbalovací seznam pro výběr datasetu, který bude použit v rámci daného projektu. Uživatel vidí přehled všech dostupných datasetů, včetně jejich názvu a typu zpracování dat. Po výběru konkrétní položky je zvolený dataset přiřazen k projektu a následně využit během trénování neuronové sítě.

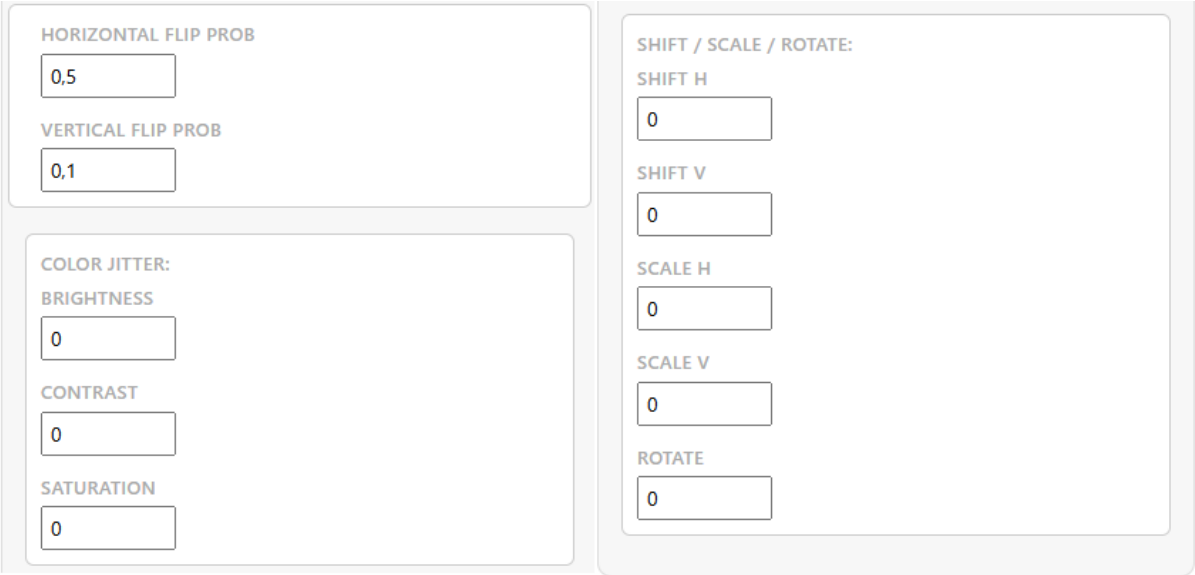


Obrázek 8 - Nastavení vstupu a výstupu

V části nastavení vstupů a výstupů si uživatel určuje základní transformace dat, které definují, v jaké podobě jsou vzorky předávány neuronové síti. U vstupních dat je možné zvolit typ obrazu, například RGB, a cílové rozlišení obrazu pomocí šířky a výšky. Tím je zajištěno, že všechna vstupní data mají jednotný formát bez ohledu na původní podobu datasetu. Tyto transformace jsou využívány například při úlohách, kde je potřeba převést černobílé vstupy na barevné obrazy nebo při práci s nízkým rozlišením vstupních dat.

Výstupní nastavení umožňuje definovat požadovaný formát výstupu neuronové sítě. Uživatel volí počet výstupních kanálů a cílové rozlišení výsledného obrazu. Typickým příkladem je úloha super-resolution, kde síť přijímá vstupní obraz o nižším rozlišení, například 32×32 pixelů, a generuje

výstupní obraz ve vyšším rozlišení, například 128×128 pixelů. Podobným způsobem lze definovat i úlohy, kde síť zpracovává obraz v jednom barevném prostoru a produkuje výstup v jiném, například při kolorování černobílých snímků.

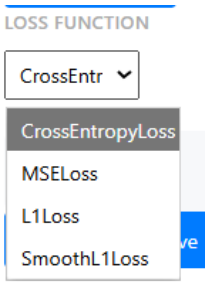


Obrázek 9 - Augmentace

Část nastavení augmentací slouží k rozšíření a variabilitě trénovacích dat během učení neuronové sítě. Augmentace umožňují náhodně měnit vstupní obrazy, aniž by bylo nutné vytvářet nové datasety, čímž se zvyšuje robustnost modelu a snižuje riziko přeučení. Tyto transformace jsou aplikovány pouze během trénování a nemění původní data uložená v systému.

Mezi dostupné augmentace patří horizontální a vertikální převrácení obrazu s nastavitelnou pravděpodobností. Tyto transformace jsou vhodné zejména pro úlohy, kde orientace objektu nehraje zásadní roli, například při klasifikaci nebo generativních obrazových úlohách. Dále je možné upravovat barevné vlastnosti obrazu pomocí parametrů jasu, kontrastu a saturace, což simuluje různé světelné podmínky a kvalitu vstupních dat.

Součástí augmentací jsou také geometrické transformace, jako je posun, změna měřítka a rotace obrazu. Tyto operace umožňují modelu lépe generalizovat na mírně posunuté nebo natočené vstupy a zvyšují jeho odolnost vůči drobným změnám ve vstupních datech. Díky možnosti jemného nastavení jednotlivých parametrů má uživatel plnou kontrolu nad mírou augmentace a může ji přizpůsobit konkrétní úloze a typu datasetu.



Obrázek 10 - Ztrátové funkce

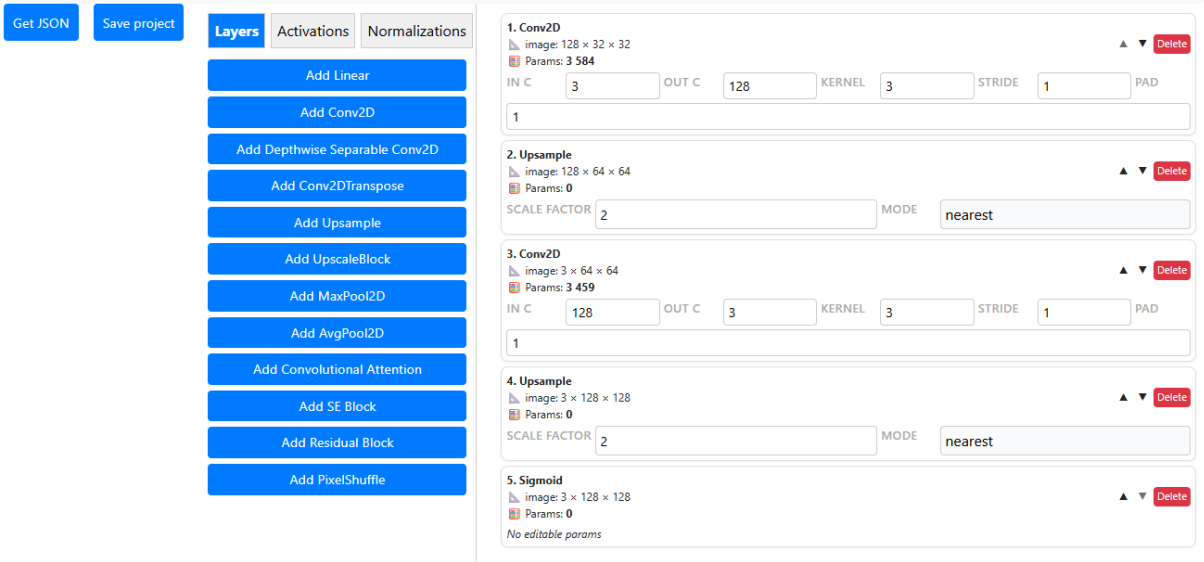
V části nastavení ztrátové funkce si uživatel vybírá způsob, jakým je během trénování vyhodnocována chyba mezi výstupem neuronové sítě a požadovaným cílem. Zvolená ztrátová funkce má přímý vliv na průběh učení i výslednou kvalitu modelu a je volena podle typu řešené úlohy.

Funkce CrossEntropyLoss je určena především pro klasifikační úlohy, kde síť rozhoduje mezi několika třídami. Typickým příkladem je rozpoznávání objektů nebo třídění obrazů do kategorií. Tato ztrátová funkce penalizuje nesprávné predikce výrazněji a podporuje jednoznačné rozdělení výstupních pravděpodobností mezi jednotlivé třídy.

Funkce MSELoss (Mean Squared Error) je využívána zejména v regresních a generativních úlohách, kde je cílem co nejpřesněji aproximovat numerickou hodnotu nebo obraz. Je vhodná například pro úlohy super-resolution nebo rekonstrukce obrazu, kde se hodnotí rozdíl mezi predikovaným a cílovým obrazem na úrovni pixelů.

Funkce L1Loss měří absolutní rozdíl mezi výstupem sítě a cílovými hodnotami. Oproti MSELoss je méně citlivá na extrémní odchylky a může vést k robustnějším výsledkům v situacích, kde se ve datech vyskytují šumy nebo odlehlé hodnoty. Tato funkce je často využívána při generování obrazů nebo regresních úlohách s vyšší mírou nejistoty.

Funkce SmoothL1Loss představuje kompromis mezi MSELoss a L1Loss. Pro malé chyby se chová podobně jako MSELoss, zatímco pro větší odchylky přechází k chování L1Loss. Díky tomu poskytuje stabilnější průběh trénování a je vhodná pro úlohy, kde je potřeba kombinovat přesnost s odolností vůči extrémním chybám.



Obrázek 11 - Editor

Editor slouží k vizuálnímu návrhu architektury neuronové sítě bez nutnosti psát zdrojový kód. Uživatel v něm sestavuje model postupným přidáváním vrstev, které jsou aplikovány sekvenčně v přesně daném pořadí. Každá změna architektury se okamžitě promítá do výpočtu tvaru tenzoru i celkového počtu parametrů, což umožňuje průběžně kontrolovat složitost a správnost navrhovaného modelu.

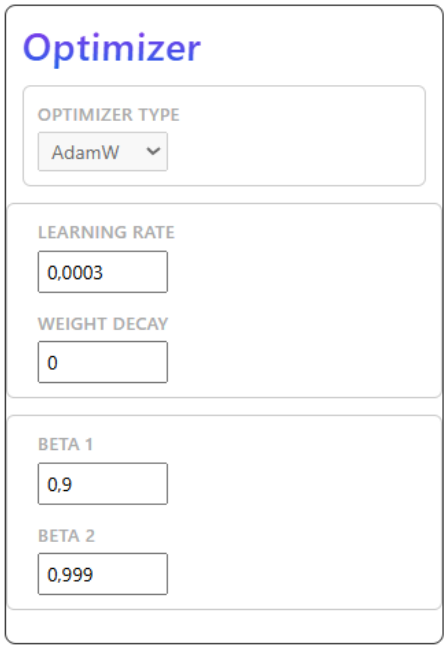
Rozhraní editoru je rozděleno na boční panel a hlavní pracovní plochu. V postranním panelu si uživatel vybírá typy stavebních bloků, které chce do sítě přidat. K dispozici jsou základní vrstvy,

aktivační funkce a normalizační vrstvy, mezi nimiž lze přepínat pomocí záložek. Po zvolení konkrétní vrstvy je tato vrstva přidána na konec aktuální architektury.

Hlavní část editoru zobrazuje jednotlivé vrstvy ve formě karet uspořádaných shora dolů podle jejich pořadí ve výpočetním grafu. Každá karta obsahuje název vrstvy, aktuální rozměry výstupního tenzoru a počet parametrů, které daná vrstva přidává do modelu. Uživatel může vrstvy mezi sebou přesouvat, čímž mění jejich pořadí, nebo je zcela odstranit. Tím je umožněna rychlá iterace nad návrhem architektury a experimentování s různými konfiguracemi.

U vrstev, které obsahují nastavitelné parametry, editor nabízí formulář s příslušnými vstupy, například počet vstupních a výstupních kanálů, velikost jádra nebo krok konvoluce. U vrstev bez parametrů je tato skutečnost explicitně zobrazena. Editor tak slouží nejen jako nástroj pro sestavení modelu, ale i jako přehledná dokumentace jeho struktury.

Výslednou architekturu může uživatel uložit jako projekt nebo exportovat do strukturovaného formátu JSON, který je následně použit backendem pro vytvoření a natrénování neuronové sítě. Tím editor propojuje uživatelské rozhraní s technickou částí systému a tvoří klíčovou součást celého workflow aplikace.

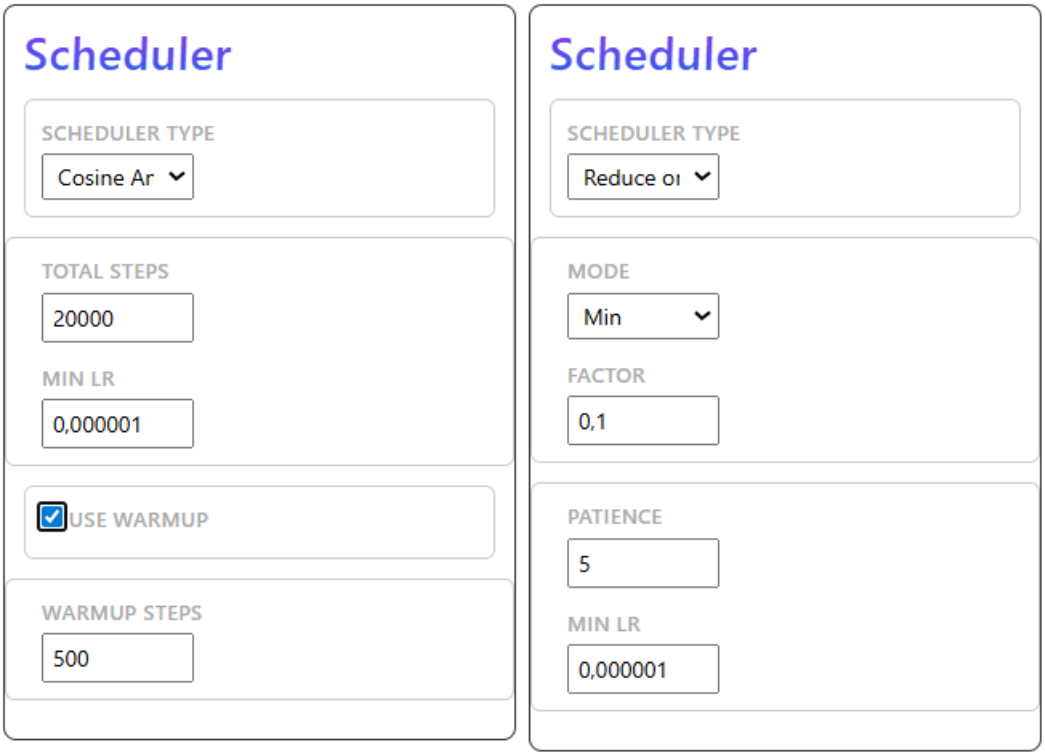


Obrázek 12 - Optimizátor

Tento blok rozhraní slouží k nastavení optimalizačního algoritmu, který je použit během trénování neuronové sítě. V aktuální verzi aplikace je k dispozici optimalizátor AdamW, který kombinuje adaptivní úpravu rychlosti učení s regularizací pomocí váhového úbytku. Typ optimalizátoru je zobrazen informativně a není možné jej měnit, což zajišťuje stabilní a předvídatelné chování trénovacího procesu.

Uživatel má možnost upravit klíčové hyperparametry optimalizátoru. Learning rate určuje velikost kroku při aktualizaci vah a má zásadní vliv na rychlost i stabilitu učení. Parametr weight decay slouží jako forma regularizace a omezuje příliš velké hodnoty vah, čímž pomáhá snižovat přeučení modelu. Hodnoty beta 1 a beta 2 určují chování exponenciálních klouzavých průměrů prvního a druhého

momentu gradientu, které AdamW používá k adaptivnímu řízení aktualizací. Díky těmto nastavením může uživatel jemně doladit průběh trénování podle konkrétní úlohy a charakteru dat.

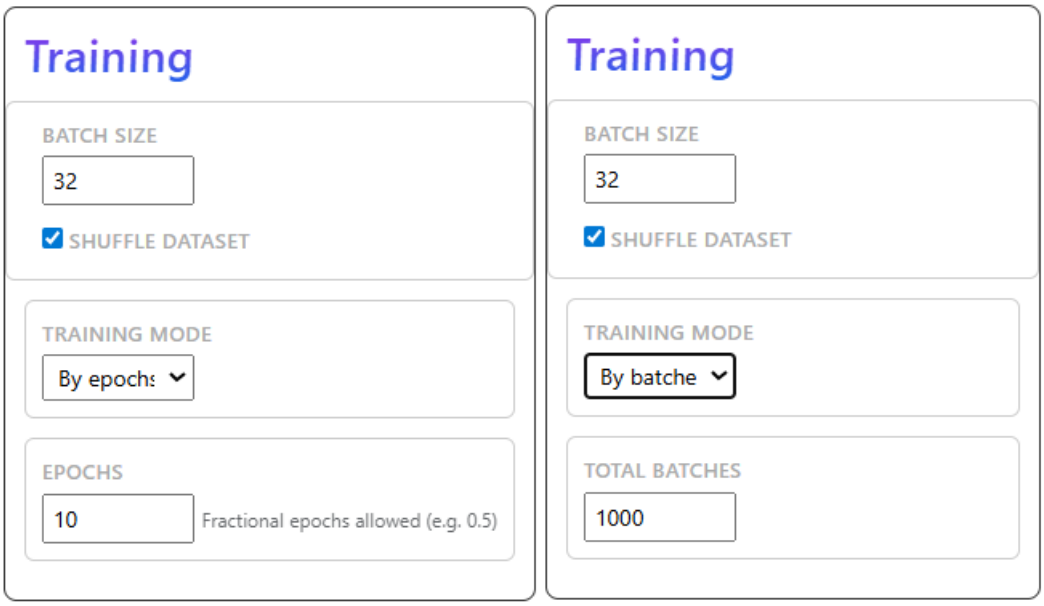


Obrázek 13 - Nastavení scheduleru

Tento blok rozhraní slouží k nastavení plánovače učící rychlosti (learning rate scheduler), který řídí změny hodnoty learning rate v průběhu trénování. Uživatel si nejprve volí typ plánovače. Možnost None znamená, že learning rate zůstává po celou dobu tréninku konstantní. Ostatní varianty umožňují dynamicky upravovat rychlost učení v závislosti na čase nebo chování trénovací metriky, což může výrazně zlepšit konvergenci modelu.

Plánovač Cosine Annealing postupně snižuje learning rate podle kosinové funkce od počáteční hodnoty až k minimální hodnotě během zadaného počtu kroků. Uživatel zde nastavuje celkový počet kroků a minimální learning rate. Volitelně lze zapnout warmup fázi, během které se learning rate na začátku trénování plynule zvyšuje z velmi malé hodnoty, což pomáhá stabilizovat učení zejména u hlubších modelů.

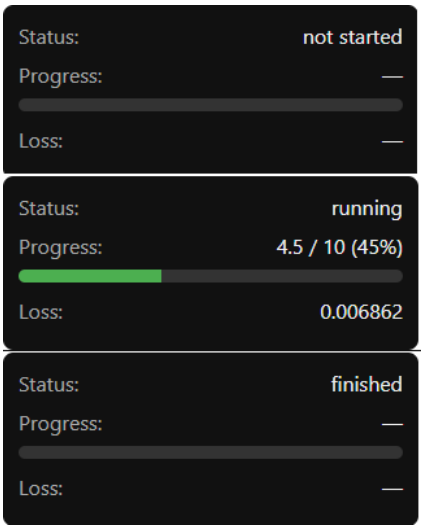
Plánovač Reduce on Plateau reaguje na vývoj zvolené metriky, typicky hodnoty loss nebo validační přesnosti. Uživatel určuje, zda má být metrika minimalizována nebo maximalizována, o jaký faktor se learning rate sníží a kolik epoch bez zlepšení je tolerováno, než ke snížení dojde. Parametr minimální learning rate zabraňuje tomu, aby se hodnota learning rate snížila pod rozumnou mez. Tento přístup je vhodný zejména pro delší trénování, kde není předem znám optimální průběh učení.



Obrázek 14 - Nastavení trenovacích parametrů

Tento blok slouží k nastavení základních parametrů samotného trénovacího procesu. Uživatel zde určuje velikost dávky (batch size), tedy počet vzorků zpracovaných během jednoho kroku učení. Menší batch size obvykle vede k stabilnějšímu učení, ale delší době trénování, zatímco větší hodnoty zvyšují výpočetní efektivitu za cenu vyšších nároků na paměť. Volba shuffle dataset umožňuje náhodné promíchání dat před každou epochou, což pomáhá omezit závislost modelu na pořadí vzorků a zlepšuje generalizaci.

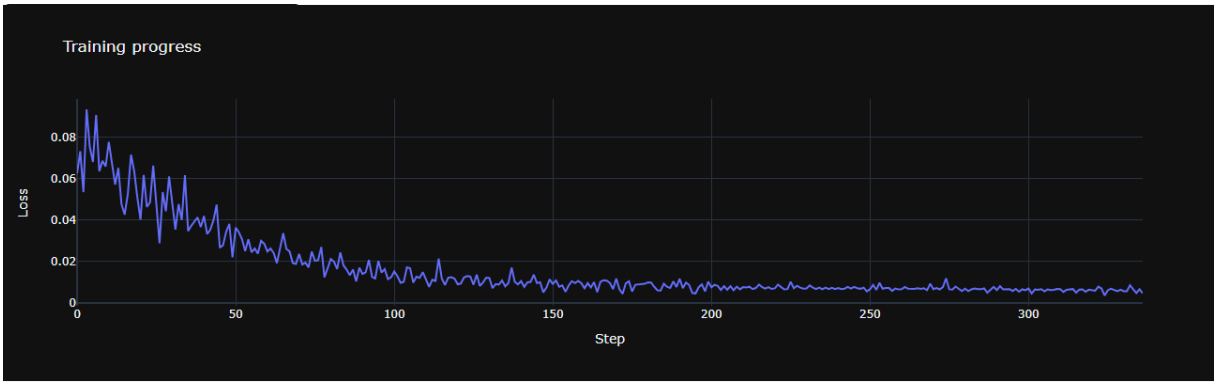
Dále si uživatel volí režim trénování. V režimu podle epoch je trénování řízeno počtem průchodů celým datasetem, přičemž aplikace umožňuje i zlomkové hodnoty, například pro kratší experimentální běhy. Alternativní režim podle počtu batchů umožňuje přesně určit celkový počet trénovacích kroků bez ohledu na velikost datasetu, což je vhodné zejména při ladění hyperparametrů nebo při práci s velmi rozsáhlými daty. Rozhraní dynamicky zobrazuje pouze relevantní nastavení podle zvoleného režimu, čímž snižuje riziko chybné konfigurace.



Obrázek 15 - Průběh učení

Tato část rozhraní slouží k průběžnému sledování stavu trénování neuronové sítě. Uživatel zde vidí aktuální stav procesu, například zda je trénování spuštěno, probíhá nebo již bylo dokončeno. Textový indikátor je doplněn informací o postupu, která vyjadřuje, jaká část trénování již byla zpracována vzhledem k celkovému rozsahu úloh.

Součástí bloku je také grafický indikátor průběhu ve formě progresního pruhu, který poskytuje rychlou vizuální představu o tom, jak daleko se trénování nachází. Dále je zobrazována aktuální hodnota ztrátové funkce (loss), která umožňuje uživateli sledovat, zda se model v průběhu učení skutečně zlepšuje. Tento přehled poskytuje základní, ale dostatečně informativní zpětnou vazbu bez nutnosti nahlížet do logů nebo externích nástrojů.



Obrázek 16 - Graf

Tento graf znázorňuje průběh trénování neuronové sítě pomocí vývoje hodnoty ztrátové funkce v závislosti na počtu trénovacích kroků. Na vodorovné ose jsou zobrazeny jednotlivé kroky učení (Step), zatímco svislá osa reprezentuje hodnotu ztrátové funkce (Loss). Graf je opatřen názvem *Training progress*, který jednoznačně určuje jeho význam v kontextu trénovacího procesu.

Závěr

V úvodu práce byl stanoven cíl vytvořit webovou aplikaci, která umožní uživatelům spravovat neuronové sítě od přípravy dat až po samotné učení modelu v přehledném prostředí. Tento cíl se podařilo naplnit především po technické stránce. Výsledná aplikace umožňuje vytvářet projekty, pracovat s datasety, konfigurovat architekturu neuronové sítě, nastavovat parametry učení a spouštět samotný trénovací proces. Součástí systému je také vizualizace průběhu učení pomocí grafu ztrátové funkce, což odpovídá původnímu záměru demonstrovat celý proces práce s neuronovou sítí v jednom nástroji. Funkční je rovněž ukládání projektů a datasetů, takže je možné navázat na předchozí práci bez nutnosti opakované konfigurace.

Zároveň se však během realizace ukázalo, že některé původní představy byly příliš optimistické. Neuronové sítě jsou velmi komplexní systémy s velkým množstvím detailů, výjimek a skrytých souvislostí. Přestože se podařilo vytvořit grafický editor architektury a rozhraní pro nastavení učení, vytvoření skutečně intuitivního uživatelského rozhraní, které by bylo snadno pochopitelné i pro méně zkušené uživatele, se ukázalo jako velmi obtížné. V praxi by takové rozhraní vyžadovalo buď výrazné zjednodušení možností, nebo dlouhodobý vývoj a testování s reálnými uživateli. Projekt tak spíše ukazuje hranici mezi snahou o zpřístupnění neuronových sítí a jejich přirozenou složitostí.

Práce na projektu mi přinesla hlubší pochopení nejen principů neuronových sítí a jejich učení, ale také praktických problémů spojených s návrhem větších systémů, správou dat a tvorbou uživatelského rozhraní. Projekt mi umožnil propojit teoretické znalosti z oblasti umělé inteligence s reálnou implementací a získat zkušenosti s návrhem modulární aplikace. Do budoucna by bylo možné systém dále rozšířit například o validační metriky, pokročilejší modely nebo lepší nápovědu pro uživatele. I přes své limity lze výslednou aplikaci považovat za funkční demonstraci celého procesu práce s neuronovou sítí a za solidní základ pro další rozvoj.

Seznam použitých zdrojů

1. GOOGLE. Google Colab. Online. 2017. Dostupné z: https://colab.research.google.com/. [cit. 2026-01-07].

2. Kaggle. Online. 2010. Dostupné z: https://www.kaggle.com/datasets. [cit. 2026-01-07]. [Online]

3. Hugging Face. Online. 2016. Dostupné z: https://huggingface.co/docs/datasets/index. [cit. 2026-01-07]. [Online]

4. Python. Online. 2024. Dostupné z: https://www.python.org/doc/. [cit. 2026-01-07].

5. Py Torch. Online. 2024. Dostupné z: https://docs.pytorch.org/docs/stable/index.html. [cit. 2026-01-07].

6. Docker. Online. 2024. Dostupné z: https://docs.docker.com/. [cit. 2026-01-07].

7. Alembic. Online. 2024. Dostupné z: https://alembic.sqlalchemy.org/en/latest/. [cit. 2026-01-07].

8. PostgeSQL. Online. 2024. Dostupné z: https://www.postgresql.org/docs/. [cit. 2026-01-07].

9. Redis. Online. 2024. Dostupné z: https://redis.io/docs/latest/. [cit. 2026-01-07].

10. React. Online. 2024. Dostupné z: https://react.dev/learn. [cit. 2026-01-07].

11. Chat GPT. Online. 2024. Dostupné z: https://chatgpt.com/. [cit. 2026-01-07].

12. Py Torch. Online. 2016. Dostupné z: https://docs.pytorch.org/docs/stable/index.html. [cit. 2026-01-07].

Seznam obrázků

Seznam kódů

Sem vygenerujte seznam kódů (Reference -> Titulky).