Gimnazija Franca Miklošiča Ljutomer
Prešernova 34
9240 Ljutomer

CELOSTRANSKA PREPOZNAVA ROČNO NAPISANIH MATEMATIČNIH ZNAKOV S POMOČJO NEVRONSKIH MREŽ

INFORMATIKA

RAZISKOVALNA NALOGA

Avtorja: Nino Gubič
Savo Prelog
Mentor: Damjan Erhatič

Ljutomer, januar 2024

POVZETEK

Cilj raziskovalne naloge je bil celostranska prepoznava ročno napisanih matematičnih znakov s pomočjo nevronskih mrež. Nevronske mreže so gradniki strojnega učenja, ki temeljijo na človeških možganih. Zgrajeni so iz povezanih vozlišč, ki uporabljajo uteži, da prilagodijo pomembnost vhodnih podatkov pri odločitvah ali napovedih.

Najprej smo poiskali model, ki že uspešno prepoznava ročno pisavo, in ga uporabili kot osnovo za naš model. Nato smo ga nadgradili, da je lahko sprejel in se učil tudi iz drugih podatkov z namenom, da bi se lahko naučil prepoznati tudi različne matematične simbole. To smo dosegli tako, da smo s programskim jezikom Python ustvarili razred, ki smo ga nato vstavili v izhodni model, narejen izključno za prepoznavo črk. Model nam je uspelo naučiti, da prepozna tudi matematične enačbe in jih zapiše v Latex jeziku z nizko natančnostjo. Natančnost bi lahko povečali, če bi model učili dlje časa in z več podatki ter če bi preoblikovali slike na isto velikost ali jih združili v paragrafe. Ker smo bili pri učenju omejeni na slike s črno pisavo na beli podlagi, bi pri nadaljnjih raziskavah lahko uporabili različne barve in podlage.

KLJUČNE BESEDE: strojno učenje, nevronska mreža, prepoznava ročne pisave, matematični simboli, prepoznava slik

ABSTRACT

This thesis focuses on the recognition of handwritten mathematical characters in full pages using neural networks. The neural networks utilized in this research are based on the structure of the human brain and consist of interconnected nodes that adjust the importance of input data through the use of weights.

Our approach involved finding a pre-existing model that was successful in recognizing handwriting and using it as a starting point for our model. We then improved the model to recognize various mathematical symbols by incorporating additional data. Using the Python programming language, we created a class and replaced the existing text recognition class with our own. The result was a model that was able to recognize mathematical equations and translate them into latex format with a limited degree of accuracy. The accuracy could have improved by extending the training period with a larger dataset and standardizing the image size or grouping the images into paragraphs. While limited to only black font on a white background in the training dataset, exploring different font colors and backgrounds could be a direction for future research.

KEY WORDS: Machine learning, neural networks, handwritten text recognition, mathematical symbols, image recognition

KAZALO VSEBINE

KAZALO SLIK

1. UVOD

Zaradi napredka v algoritmih in procesorski moči je postalo strojno učenje zelo močno orodje. Prepoznavanje vzorcev je nekaj, kar je za ljudi naravno, vendar s povečevanjem količine podatkov in večjo kompleksnostjo hitro dosežemo meje naše zmogljivosti. Algoritme strojnega učenja danes uporabljamo za odkrivanje vzorcev v podatkih, ki na prvi pogled niso vidni. Zasnovani so tako, da se tako rekoč "učijo" s tem, da prilagajajo svoje parametre v znani zbirki podatkov. Po dolgotrajnem učenju potem oblikujejo naučen model, ki ga je mogoče pozneje uporabiti za neznane podatke. [6]

Danes je strojno učenje že krepko preseglo okvire preprostih računalniških iger in z njim živimo že kar nekaj časa, vede ali nevede. Uporablja se denimo na področjih finančnih storitev, državne uprave, zdravstva, maloprodaje, prometa in energetske industrije; skratka v večini panog, ki se ukvarjajo z velikimi količinami podatkov. [10]

Izraz umetna inteligenca nama ni tuj, saj že dlje časa raziskujeva njeno uporabo v sodobnem svetu. Eno področje umetne inteligence, ki sva ga raziskovala, je tudi optična prepoznava znakov (angl. »OCR« ali »optical character recognition«). Pri raziskovanju tega področja sva ugotovila, da obstaja zelo veliko že naučenih modelov, ki lahko prepoznavajo ročno pisavo. Ugotovila sva tudi, da je veliko pomanjkanje modelov, ki so zmožni poleg pisave prepoznavati tudi matematične simbole, kar nama je dalo navdih in motivacijo za najino raziskovalno nalogo. S svojim delom sva želela omogočiti hitrejše prepisovanje matematičnih simbolov učiteljem, profesorjem in vsem ostalim, ki jih uporabljajo pri svojem delu.

1.1. Opredelitev teme raziskovanja in cilji

Na področju strojnega učenja je bilo ustvarjenih veliko modelov in programov, ki so tako rekoč »naučeni«, da prepoznavajo ročno pisavo in jo spreminjajo v elektronsko obliko za boljšo preglednost in hitro prepisovanje dokumentov. Cilj seminarske naloge je bil nadgraditi že obstoječe modele tako, da bi lahko prepoznali tudi ročno zapisane račune, ki vsebujejo matematične simbole in najrazličnejše matematične operacije ter jih prepisali v elektronsko obliko čim uspešneje in hitreje – torej celostranska prepoznava ročno napisanih matematičnih znakov in njihova pretvorba v zapis Latex.

V raziskovalni nalogi smo si zadali naslednje cilje:

1. Predstaviti, kaj je strojno učenje in kje se uporablja.
2. Predstaviti, kako so nevronske mreže povezane s strojnim učenjem in kako delujejo.
3. Razviti model za celostransko prepoznavo ročno napisanih matematičnih znakov in besedila.

1.2. Hipoteze

V raziskovalni nalogi smo preučili naslednji hipotezi:

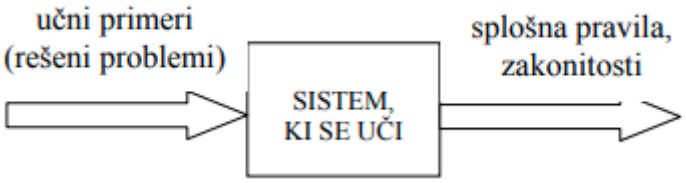
- Hipoteza 1: S pomočjo nevronskih mrež je možno ustvariti program, ki lahko iz fotografije prepozna ročno pisavo in jo zapiše v digitalni obliki.
- Hipoteza 2: Program lahko iz fotografije prepozna tudi matematične simbole in rezultate zapiše v jeziku Latex.

1.3. Metode dela

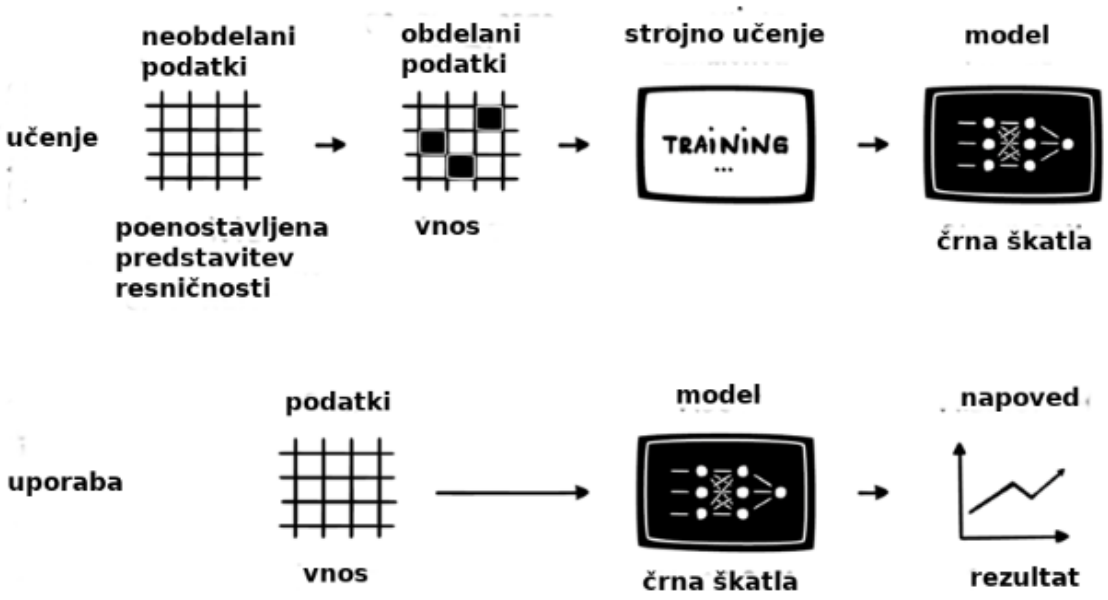
V teoretičnem delu raziskovalne naloge smo za opis delovanja nevronskih mrež in strojnega učenja uporabili metodo deskripcije. V praktičnem delu sta bila vključena pregled in pridobitev že obstoječega modela in dopolnjevanje omenjenega modela. V tem delu smo uporabili metodo analize. Za predstavitev rezultatov raziskave smo uporabili metodo sinteze. Za zapisovanje kode smo uporabili programski jezik Python.

2. STROJNO UČENJE

Strojno učenje je področje umetne inteligence, ki preučuje učljive algoritme. Učljivi algoritmi izboljšujejo svoje delovanje na podlagi podatkov ali izkušenj iz preteklih izvajanj. Strojno učenje (angl. machine learning, tudi statistical learning) pogosto poteka v obliki algoritmične analize podatkov s ciljem iskanja vzorcev v podatkih ali pa gradnje napovednih modelov iz podatkov, kot je prikazano na sliki (Slika 2). Drugače povedano: računalniki se lahko naučijo določenih stvari iz podatkov in nato obstoječe znanje o nekem področju nadgradijo, ne da bi bili prej eksplicitno programirani. Vzorci in napovedni modeli so uporabni iz preprostih razlogov: kratka in razumljiva predstavitev večje količine podatkov, izboljšava razumevanja podatkov, pojasnjevanje obnašanja opazovanega sistema ali pojava, napoved obnašanja opazovanega sistema ter odločanje o ustrezni kontroli njegovega bodočega obnašanja. Na sliki (Slika 1) je prikazan proces strojnega učenja na osnovi podanih primerov. [2] [6]



Slika 1: Učenje na osnovi primerov



Slika 2: Proces strojnega učenja

2.1. Tipologija strojnega učenja

Obstajajo tri glavne vrste strojnega učenja: nadzorovano učenje, nenadzorovano učenje in vzpodbujevalno učenje.

2.1.1. Nenadzorovano učenje

Prva vrsta je nenadzorovano učenje (angl. unsupervised learning), pri katerem gre za učenje vzorcev iz učnih podatkov, ki ne vsebujejo posebnih, eksplicitnih navodil ali oznak konceptov za učenje. K nenadzorovanemu učenju je mogoče šteti tudi vizualizacijo podatkov. [2]

2.1.2. Nadzorovano učenje

Nasprotje nenadzorovanemu učenju je nadzorovano učenje (angl. supervised learning). Pri slednjem so učni primeri v podatkovni množici označeni s posebnimi izhodnimi spremenljivkami, ki so cilj učenja. Učni primeri so pari tipa (vhod, izhod), naloga algoritma za stojno učenje pa je iskanje neznane funkcijske povezave med vhodi in izhodom. [2]

2.1.3. Vzpodbujevalno učenje

Vzpodbujevalno učenje vključuje usposabljanje algoritma za sprejemanje zaporedja odločitev z učenjem na podlagi njegovih preteklih izkušenj. Algoritem je nagrajen ali kaznovan na podlagi svojih dejanj, cilj pa je izpopolniti nagrado skozi čas. Samovozeči avtomobili na primer uporabljajo vzpodbujevalno učenje za sprejemanje odločitev, kot so pospeševanje, zaviranje in zavijanje, na podlagi nagrad, ki jih prejmejo za varno in učinkovito doseganje cilja. [9]

Uspeh strojnega učenja je odvisen od kakovosti in količine podatkov, na katerih se algoritem uči. Več podatkov kot ima algoritem na voljo, bolje se lahko uči. Preveč podatkov lahko privede tudi do pretiranega prilagajanja, kjer algoritem postane preveč specializiran za podatke za učenje in se ne posplošuje dobro na nove podatke (angl. overfitting). Da bi se izognili pretiranemu prilagajanju, lahko uporabimo tehnike, kot sta prečno preverjanje in regularizacija. Prečno preverjanje je metoda, ki uporablja različne dele podatkov za testiranje in usposabljanje modela v različnih iteracijah. Regularizacija pa uporabi "kazen" za vhodne parametre z večjimi koeficienti, kar posledično omeji variacijo modela. [2]

2.2 Uporaba

Strojno učenje se uporablja v različnih primerih: od napovedovanja vedenja strank do oblikovanja sistema za samovozeče avtomobile. Stojno učenje lahko na primer pomaga podjetjem razumeti svoje stranke na globlji ravni. Algoritmi strojnega učenja so se z zbiranjem podatkov in povezovanjem le-teh z njihovim vedenjem skozi čas naučili, kako pomagati ekipam prilagoditi razvoj svojih izdelkov in s tem zvišati povpraševanje. [3]

Nekatera podjetja uporabljajo strojno učenje kot glavno gonilo v svojih poslovnih modelih. Uber, na primer, uporablja algoritme za usklajevanje voznikov s potniki. Google uporablja strojno učenje za prikazovanje oglasov v iskalnikih. Strojno učenje ima tudi pomanjkljivosti. Največja je ta, da je zelo drag proces. Projekte strojnega učenja običajno vodijo podatkovni znanstveniki, ki imajo visoke plače. Ti projekti zahtevajo tudi programsko infrastrukturo, ki pa predstavlja velike stroške. Obstaja tudi problem pristranskosti strojnega učenja. Algoritmi, usposobljeni na podatkovnih nizih, ki izključujejo določene populacije ali vsebujejo napake, lahko privedejo do netočnih modelov, ki so v najboljšem primeru neuspešni, v najslabšem pa diskriminacijski. [3]

Kljub številnim prednostim so s strojnim učenjem povezani tudi izzivi in etični pomisleki. Eden od izzivov je razlaga rezultatov, saj so algoritmi strojnega učenja lahko zapleteni in težko razumljivi. To lahko privede do pristranskosti in napak v rezultatih, zlasti če so podatki, uporabljeni za usposabljanje

algoritma, pristranski ali nepopolni. Drug izziv je tveganje izgube zasebnosti, saj algoritmi strojnega učenja za napovedovanje pogosto uporabljajo velike količine osebnih podatkov. [6]

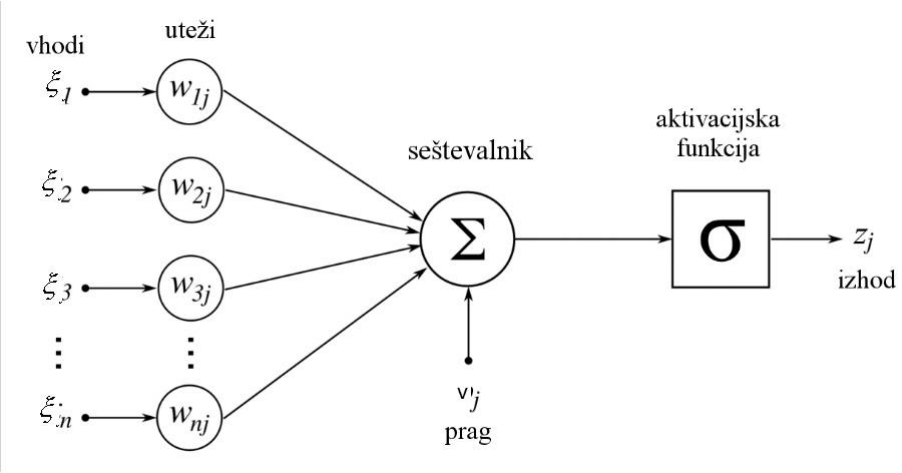
2.3. Nevronske mreže

Umetna nevronska mreža je paradigma za obdelavo informacij, ki se zgleduje po delovanju biološkega živčnega sistema. Ključni element predstavljenega modela je drugačna struktura obdelave informacij. Sestavlja ga veliko število prepletajočih se procesnih elementov – nevronov, ki jih je potrebno za vsako aplikacijo posebej prilagoditi z učnim procesom, določenim za specifičen problem.

Obstaja več vrst nevronskih mrež, med drugim konvolucijske nevronske mreže (CNN) in transformatorske nevronske mreže. Ti dve vrsti nevronskih mrež imata različne arhitekture in se uporabljata za različne naloge. [8]

2.3.1. Umetni nevron

Umetni nevron je naprava z mnogimi vhodi in enim samim izhodom. Matematični modeli so sestavljeni iz med seboj povezanih vozlišč. Parametri za določanje povezav so prilagodljivi, kar omogoča prilagajanje signalov, ki potujejo skozi nevron. Umetne nevronske mreže so sestavljene po vzoru bioloških. Najprej so definirani vsi njihovi sestavni deli, nato pa še njihove povezave. Računalnik sprogramiramo tako, da te lastnosti simuliramo. Na sliki (Slika 3) je grafično prikazana zgradba takega nevrona. Nevronska mreža, tako biološka kot umetna pa je sestavljena iz številnih elementov – nevronov, ki delujejo paralelno za rešitev zastavljenega problema [8].



Slika 3: Zgradba nevrona

2.2.1. Konvolucijske nevronske mreže (CNN)

Eno od področij, na katerem je globoko učenje doseglo osupljiv uspeh, je obdelava slik. Z dodajanjem več plasti v mrežo in uporabo vzvratnega razširjanja za določitev uteži načeloma rešimo težavo, vendar se pojavi nova: število uteži postane izredno veliko, zato je lahko za dosego zadovoljive natančnosti potrebna prevelika količina učnih podatkov, da bi bila realna. [7]

Na srečo obstaja zelo elegantna rešitev problema prevelikega števila uteži, in sicer posebna vrsta nevronske mreže, ali bolje rečeno, posebna vrsta plasti, ki jo je mogoče vključiti v globoko nevronsko mrežo. Ta posebna vrsta plasti je tako imenovana konvolucijska plast. Mreže, ki vključujejo konvolucijske plasti, imenujemo konvolucijske nevronske mreže (CNN). Njihova ključna lastnost je,

da lahko zaznajo atribute na sliki, kot so svetle ali temne pike (ali pike določene barve), različno usmerjeni robovi, vzorci itd. [7]

Ena od glavnih prednosti CNN je njihova sposobnost samodejnega učenja uporabnih lastnosti iz slikovnih podatkov, kar pomaga izboljšati njihovo delovanje. To je v nasprotju s tradicionalnimi metodami obdelave slik, pri katerih je bilo treba značilnosti določiti ročno. Poleg tega lahko CNN obdelujejo velike količine podatkov, zato so primerne za obdelavo slik in videoposnetkov. [4][7]

2.2.2. Transformatorske nevronske mreže

Transformatorske nevronske mreže so bile predstavljene leta 2017 in se pogosto uporabljajo za naloge obdelave naravnega jezika, kot so: strojno prevajanje, analiza razpoloženja in generiranje besedil. Te nevronske mreže uporabljajo mehanizme samopozornosti, ki mreži omogočajo, da se osredotoči na ustrezne informacije v vhodu. To mreži omogoča, da se nauči razmerja med različnimi deli vhoda, zaradi česar je primerno za obdelavo zaporednih podatkov. [1]

Transformacijska mreža je sestavljena iz več plasti samopozornosti (ang. self-attention layers) in naprej povezanih plasti (ang. feed-forward layers). Plasti samopozornosti omogočajo mreži, da se nauči, kateri deli vhoda so najpomembnejši za napovedovanje. Naprej povezane plasti pa uporabljajo naučena razmerja za izdelavo napovedi. [1]

Arhitektura dekodirnika v modelih NLP, kot so transformatorji, je sestavljena iz več plasti, ki delujejo na napovedih žetonih, da ustvarijo končni izhod. Ključni elementi arhitekture dekoderja so:

- plast za vstavljanje (angl. embedding layer): Ta plast prevzame napovedi žetonov in jih pretvori v zgoščene vektorje v visokodimenzionalnem prostoru. To modelu omogoča, da zazna zapletene odnose med žetoni in razlikuje med podobnimi žetoni.
- popolnoma povezana plast (angl. fully connected layer): Ta plast vzame visokodimenzionalne vektorje iz plasti za vstavljanje in pripravi končne napovedi za vsak žeton v izhodnem zaporedju. Uporablja aktivacijske funkcije in uteži za preslikavo vektorjev v želeni izhodni prostor.
- mehanizem pozornosti: Ta mehanizem dekoderju omogoča, da se pri ustvarjanju izhodnih napovedi osredotoči na določene dele vhodnega zaporedja. Mehanizem pozornosti pomaga dekoderju, da se izogne izgubi informacij in ustvari koherentne izhode.
- normalizacija: plasti, kot sta normalizacija ali normalizacija paketov, se uporabljata za izboljšanje stabilnosti modela in zmanjšanje pretiranega prilagajanja. [1]

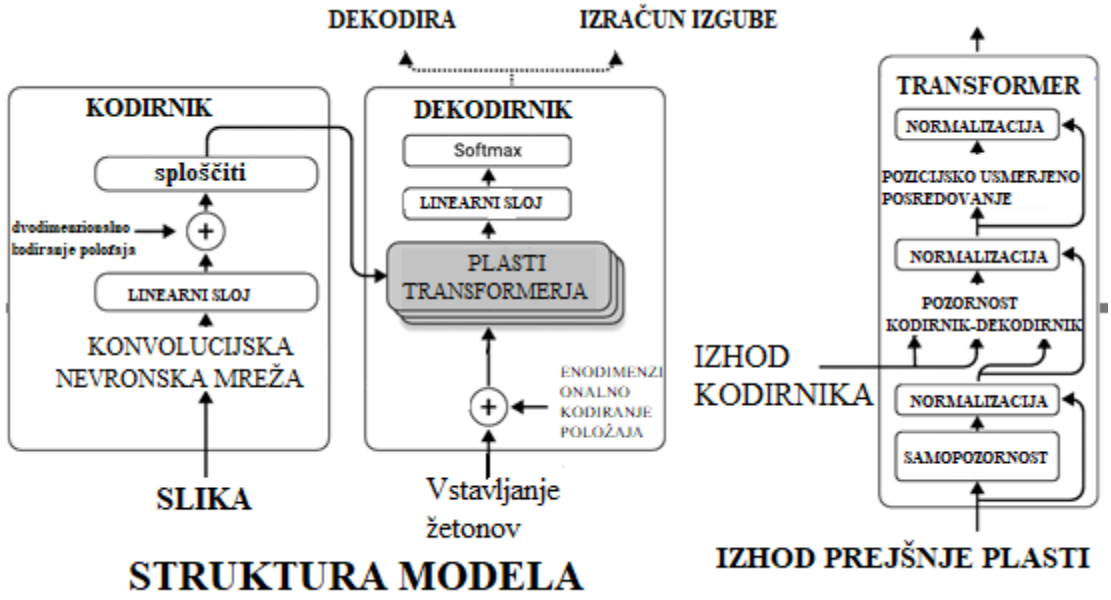
Te komponente delujejo skupaj, da ustvarijo končni izhodni rezultat dekoderja. Arhitektura se lahko razlikuje glede na specifičen primer uporabe, vendar so ti elementi skupni večini arhitektur dekoderjev v modelih NLP.

Ena od glavnih prednosti transformatorskih mrež je njihova zmožnost obdelave dolgih zaporedij podatkov, kot so dolge povedi ali obširni dokumenti. Zaradi tega so zelo primeri za naloge obdelave naravnega jezika, pri katerih so pomembna razmerja med različnimi besedami v vhodu. Poleg tega je mogoče transformatorske mreže usposabljati vzporedno, kar omogoča učinkovito usposabljanje na velikih količinah podatkov. [1]

3. RAZVOJ

Na spletu je že veliko modelov, ki so naučeni, da prepoznavajo in prepisujejo ročno zapisano pisavo. Model z največjo natančnostjo, ki smo ga našli, je od »Full Stack Deep Learning«. Kodo za njihov model so avtorji zapisali in opisali v spletnih učbenikih, ki so jih izdali kot vodilo z namenom poučevanja drugih in nadaljnjega ustvarjanje modelov. Njihove učbenike in kodo smo uporabljali kot osnovo za naš model, ki pa smo ga kasneje nadgradili, da bi zadovoljil naše želje. [5]

3.1 Opis izhodnega modela



Slika 4: Struktura modela, ki smo ga uporabili

Slika gre najprej skozi konvolucijsko nevronsko mrežo, ki poišče značilnosti slike in jih pošlje v naslednjo plast modela. Značilnosti nato prevzame transformator, ki je zasnovan za obdelavo zaporednih podatkov, kot so zaporedja znakov v primeru prepoznavanja besedila. Transformator nato preračuna razmerja med značilnostmi, ki jih pridobi konvolucijska nevronska mreža, in določi pravilno zaporedje znakov na sliki. Celoten postopek je prikazan na sliki (Slika 4).

V nadaljevanju je opisana koda izhodnega modela »Full Stack Deep Learning«, ki smo jo uporabili kot osnovo za naš model.

```python
np.random.seed(42)
torch.manual_seed(42)


def _setup_parser():
    """Set up Python's ArgumentParser with data, model, trainer, and other arguments."""
    parser = argparse.ArgumentParser(add_help=False)

    # Add Trainer specific arguments, such as --max_epochs, --gpus, --precision
    trainer_parser = pl.Trainer.add_argparse_args(parser)
    trainer_parser._action_groups[1].title = "Trainer Args"
    parser = argparse.ArgumentParser(add_help=False, parents=[trainer_parser])
    parser.set_defaults(max_epochs=1)

    # Basic arguments
    parser.add_argument(
        "--data_class",
        type=str,
        default="MNIST",
        help=f"String identifier for the data class, relative to {DATA_CLASS_MODULE}.",
    )
    parser.add_argument(
        "--model_class",
        type=str,
        default="MLP",
        help=f"String identifier for the model class, relative to {MODEL_CLASS_MODULE}.",
    )
    parser.add_argument(
        "--load_checkpoint", type=str, default=None, help="If passed, loads a model from the provided path."
    )
    parser.add_argument(
        "--stop_early",
        type=int,
        default=0,
        help="If non-zero, applies early stopping, with the provided value as the 'patience' argument."
        + " Default is 0.",
```

Slika 5: Izsek kode iz datoteke »run experiment«

Na sliki (Slika 5) je izsek kode, ki je ogrodje za izvajanje eksperimentov, ki zažene eksperiment z različnimi podatki, modeli in argumenti učenja, določenimi z različnimi ukazi za ponovljivost, ter določi naključno seme (angl. Random seed).

V kodi se najprej nastavi naključna semena za ponovljivost, nato pa se definira funkcijo »setup_parser()«, ki ustvari razčlenjevalnik argumentov za obdelavo argumentov ukazne vrstice, ki jih posreduje uporabnik.

Funkcija »main()« je vstopna točka kode, ki najprej pokliče funkcijo »_setup_parser()«, da vzpostavi razčlenjevalnik argumentov in razčleni argumente, ki jih posreduje porabnik. Funkcija nato pokliče »setup_data_and_model_from_args«, da nastavi podatkovni in modelni razred s pomočjo argumentov, ki jih je posredoval uporabnik.

Nato inicializiramo razred »LitModel« (bodisi »BaseLitModel« bodisi »TransformerLitModel«, odvisno od argumenta »args.loss«), nato nastavi imenik za beleženje in usposobi model. Če je zagotovljena pot do kontrolne točke, koda naloži kontrolno točko in nadaljuje postopek učenja. Na koncu koda shrani model in potek usposabljanja.

```
class ConvBlock(nn.Module):
    """
    Simple 3x3 conv with padding size 1 (to leave the input size unchanged), followed by a ReLU.
    """

    def __init__(
        self,
        input_channels: int,
        output_channels: int,
        kernel_size: Param2D = 3,
        stride: Param2D = 1,
        padding: Param2D = 1,
    ) -> None:
        super().__init__()
        self.conv = nn.Conv2d(input_channels, output_channels, kernel_size=kernel_size, stride=stride, padding=padding)
        self.relu = nn.ReLU()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Applies the ConvBlock to x.

        Parameters
        ----------
        x
            (B, C, H, W) tensor

        Returns
        -------
        torch.Tensor
            (B, C, H, W) tensor
        """
        c = self.conv(x)
        r = self.relu(c)
        return r
```

Slika 6: Izsek kode iz datoteke »LineCnnTranformer«

Na sliki (Slika 6) je izsek kode, ki opredeljuje model nevronske mreže za napovedovanje besedila, imenovan »LineCNNTransformer«. Model je sestavljen iz dveh delov: »LineCNN« in transformatorja.

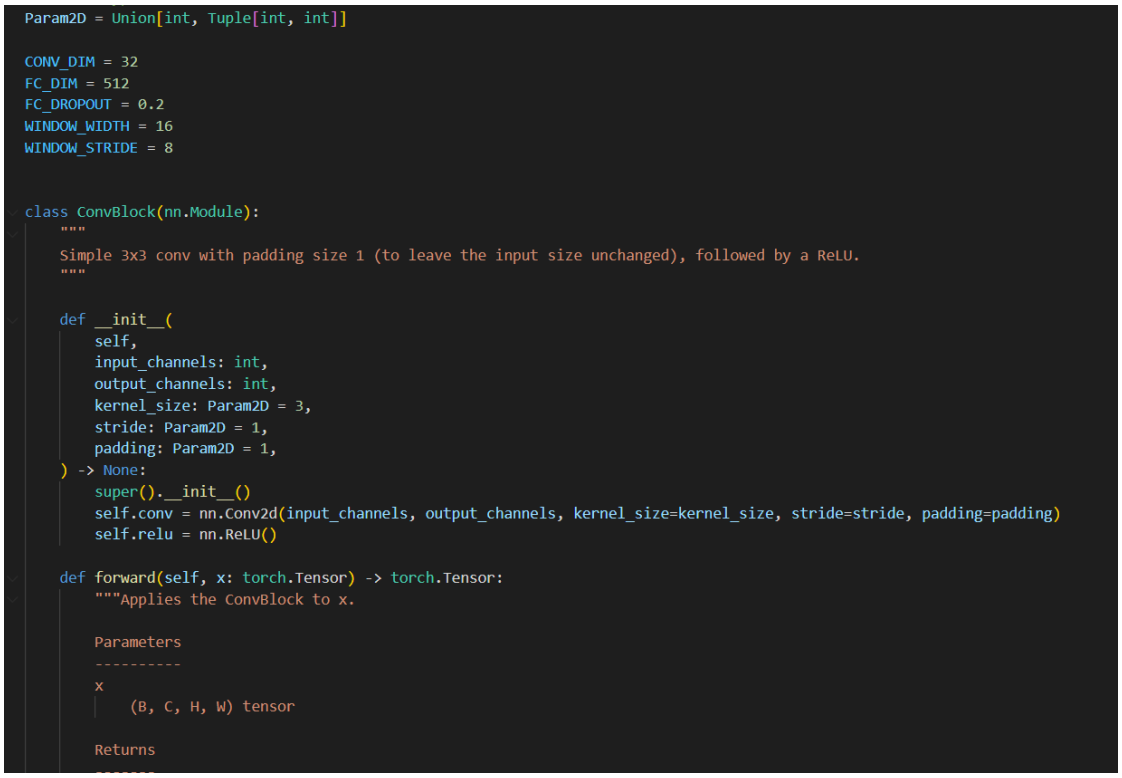
»LineCNN« prejme vhodni tenzor oblike (B, H, W), kjer je B velikost serije, H je višina in W je širina slike. Na izhodu so (B, E, S) logaritemske verjetnosti, kjer je E število razredov, S pa dolžina zaporedja.

Pretvornik je dekoder, ki obdeluje kodiran vhodni podatek iz »LineCNN«. Vhod za transformator so tenzorji (Sx, B, E), kjer je Sx dolžina vhodnega zaporedja, B je velikost serije, E pa je dimenzija kodiranja. Transformator je sestavljen iz več plasti dekoderjev, od katerih ima vsak mehanizem za samopozornost in popolnoma povezano usmerjeno mrežo (angl. feedforward network). Dekoder ima tudi položajno kodiranje (angl. positional encoding), ki dodaja informacije o položaju vsakega elementa v zaporedju.

Nazadnje ima model izhodno plast, ki kodiran izhod iz pretvornika preslika v napovedane razrede prek linearne plasti, ki mu sledi aktivacijska funkcija softmax.

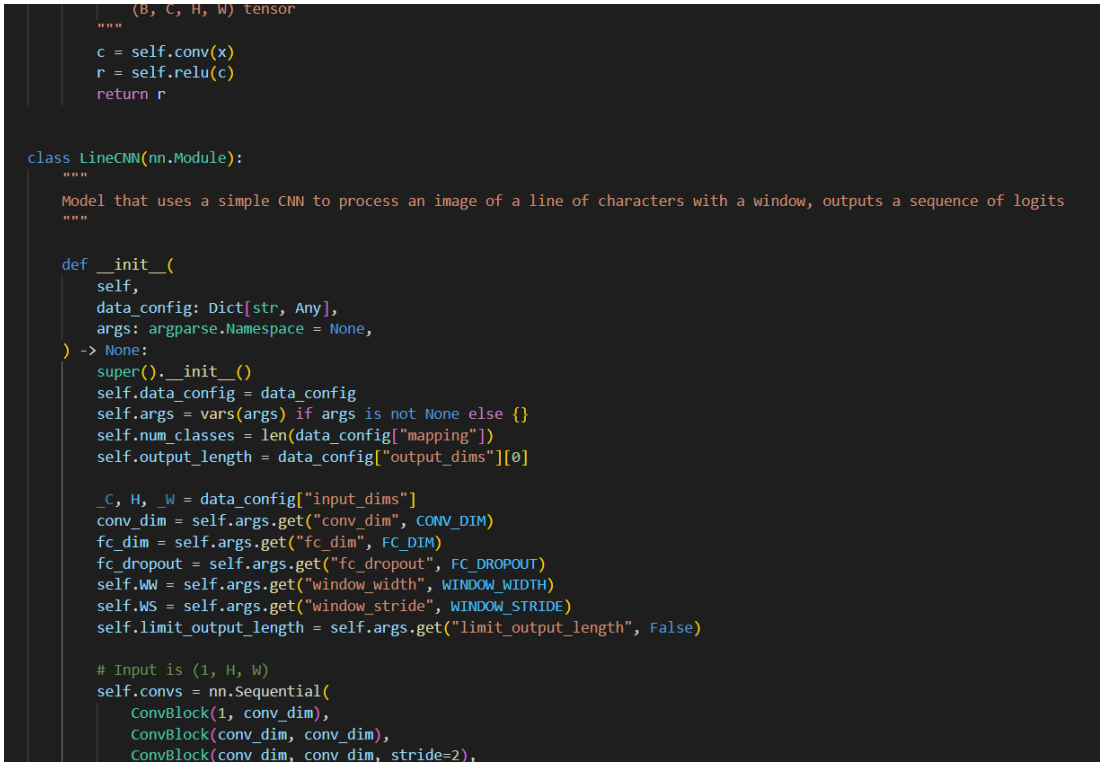
Koda opredeljuje tudi metodo za inicializacijo uteži modela in dve metodi: kodiranje in dekodiranje. Metoda kodiranja sprejme vhodni tenzor (B, H, W) in vrne kodirano zaporedje (Sx, B, E). Metoda dekodiranja sprejme dva vhoda: kodirano zaporedje (Sx, B, E) in osnovno resnico (B, Sy) ter vrne napoved za naslednje znake Sy v zaporedju.

Napoved se naredi tako, da se kodirano zaporedje pošlje skozi mrežo za dekodiranje, ki nato ustvari verjetnostno porazdelitev nad možnimi znaki Sy. Znak z največjo verjetnostjo se nato izbere in doda izhodnemu zaporedju dekoderja. Ta postopek se ponavlja, dokler dekoder ne doseže merila za zaustavitev, kot je največja dolžina ali označevalnik konca zaporedja. Izhodno zaporedje dekodirnika je predvideno nadaljevanje vhodnega zaporedja.



```
Param2D = Union[int, Tuple[int, int]]

CONV_DIM = 32
FC_DIM = 512
FC_DROPOUT = 0.2
WINDOW_WIDTH = 16
WINDOW_STRIDE = 8


class ConvBlock(nn.Module):
    """
    Simple 3x3 conv with padding size 1 (to leave the input size unchanged), followed by a ReLU.
    """

    def __init__(
        self,
        input_channels: int,
        output_channels: int,
        kernel_size: Param2D = 3,
        stride: Param2D = 1,
        padding: Param2D = 1,
    ) -> None:
        super().__init__()
        self.conv = nn.Conv2d(input_channels, output_channels, kernel_size=kernel_size, stride=stride, padding=padding)
        self.relu = nn.ReLU()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Applies the ConvBlock to x.

        Parameters
        ----------
        x
            (B, C, H, W) tensor

        Returns
```

Slika 7: Izsek iz datoteke »line_cnn« (1)



```
            (B, C, H, W) tensor
        """
        c = self.conv(x)
        r = self.relu(c)
        return r


class LineCNN(nn.Module):
    """
    Model that uses a simple CNN to process an image of a line of characters with a window, outputs a sequence of logits
    """

    def __init__(
        self,
        data_config: Dict[str, Any],
        args: argparse.Namespace = None,
    ) -> None:
        super().__init__()
        self.data_config = data_config
        self.args = vars(args) if args is not None else {}
        self.num_classes = len(data_config["mapping"])
        self.output_length = data_config["output_dims"][0]

        _C, H, _W = data_config["input_dims"]
        conv_dim = self.args.get("conv_dim", CONV_DIM)
        fc_dim = self.args.get("fc_dim", FC_DIM)
        fc_dropout = self.args.get("fc_dropout", FC_DROPOUT)
        self.WW = self.args.get("window_width", WINDOW_WIDTH)
        self.WS = self.args.get("window_stride", WINDOW_STRIDE)
        self.limit_output_length = self.args.get("limit_output_length", False)

        # Input is (1, H, W)
        self.convs = nn.Sequential(
            ConvBlock(1, conv_dim),
            ConvBlock(conv_dim, conv_dim),
            ConvBlock(conv_dim, conv_dim, stride=2),
```

Slika 8: Izsek iz datoteke »line_cnn« (2)

Na slikah 7 in 8 je koda implementacije preprostega modela konvolucijske nevronske mreže (CNN) za obdelavo slike, vrstice znakov in izpis zaporedja logitov v jeziku PyTorch. Vrstica znakov se obdela z »WINDOW_WIDTH« in korakom »WINDOW_STRIDE«. Nastale podslike so nato obdelane s konvulacijsko nevronsko mrežo, da dobimo zaporedje logitov, ki se nato uporabijo za napovedovanje.

Koda je sestavljena iz dveh glavnih razredov: »ConvBlock« in »LineCNN«. »ConvBlock« je gradnik za glavni razred »LineCNN«. Gre za preprosto operacijo konvolucije, ki ji sledi aktivacija »ReLU«. »LineCNN« je glavni model, ki je sestavljen iz več »ConvBlock« objektov, razporejenih v zaporedju, skupaj z dvema popolnoma povezanima (fc) plastema in izpadno plastjo za preprečevanje pretiranega prilagajanja.

Model »LineCNN« med inicializacijo sprejme dva parametra: »data_config« in »args.data_config«. Tu se nahajajo vse informacije o podatkih, kot so vhodne dimenzije in število razredov. Args pa je objekt imenskega prostora, ki lahko vsebuje dodatne parametre za model.

Naslednji prehod modela »LineCNN« je sestavljen iz prenosa vhodne slike skozi zaporedje objektov »ConvBlock«, ki jim sledijo popolnoma povezane plasti in izpustna plast. Rezultat je zaporedje logitov, ki se lahko nato uporabijo za napovedi. Model ima tudi metodo »_init_weights«, ki boljše inicializira uteži modela. Ta koda zagotavlja osnovne gradnike za ustvarjanje konvolucijskih modelov nad vrsticami besedila.

```
1  from .mlp import MLP
2  from .cnn import CNN
3  from .line_cnn_simple import LineCNNSimple
4  from .resnet.transformer import ResnetTransformer
5  from .line_cnn_transformer import LineCNNTransformer
```

Slika 9: Izsek kode iz inicializacije razreda

Na sliki (Slika 9) je koda, ki opredeljuje več modelov strojnega učenja za prepoznavanje znakov in besedila v slikah. Modeli so uvoženi iz različnih Python datotek.

»MLP«: To je implementacija večplastnega perceptrona, ki je vrsta umetne nevronske mreže.

CNN: To je implementacija konvolucijske nevronske mreže, ki je vrsta nevronske mreže, zasnovane posebej za naloge prepoznavanja slik.

»LineCNNSimple«: To je preprosta linijska konvolucijska nevronska mreža, ki je različica CNN, zasnovana za prepoznavanje znakov v slikah.

»ResnetTransformer«: To je izvedba preostalega omrežja z arhitekturo transformatorja, ki je vrsta nevronske mreže, zasnovane za naloge prepoznavanja slik.

»LineCNNTransformer«: To je linijska konvolucijska nevronska mreža z arhitekturo transformatorja, ki je različica »ResnetTransformerja«, zasnovana posebej za prepoznavanje znakov v slikah. [5]

3.2. Nadgradnja modela

Zgoraj opisan osnovni model je že sposoben prepoznati ročno pisavo in se učiti, toda naš cilj je prepoznava matematičnih simbolov, zato smo ga morali nadgraditi, da se bo lahko učil iz drugih vhodnih podatkov in se tako naučil prepoznavati tudi matematične simbole.

To smo dosegli tako, da smo napisali kodo za razred »dataset«, ki se uporablja za obdelavo vseh podatkov in njihovo pretvorbo v pravilno obliko za učenje. Ta datoteka se imenuje »dataset_enacbe«. Ta razred smo potem vstavili v naš izhodni model in tako omogočili, da se lahko uči prepoznati tudi matematične simbole.

Naša koda opredeljuje razred »DatasetEnacb« za nalaganje in obdelavo slik matematičnih enačb in njihovih ustreznih enačb.

```
def __init__(self, pot_do_podatkov=r"", augment=False):
    self.dictionary = {}
    self.equations = []
    self.vektorsko=[]
    self.img_paths = []
    self.your_path = pot_do_podatkov
    self.numOfData = 0
    print(self.your_path)
    self.datoteka = open(self.your_path)
    self.vrste = self.datoteka.read().splitlines()
    self.augment = augment
    self.slike_spomin = {}
    if self.augment:
        self.augmentacija_init()
```

Slika 10: Inicializacija razreda »dataset«

Na sliki (Slika 10) je prikazana » __init__« funkcija, ki inicializira razred, določi pot do imenika podatkov, odpre datoteko, ki vsebuje enačbe in imena ustreznih slikovnih datotek, prebere datoteko in razdeli vsebino v vrstice. Prav tako inicializira slovar »dictionary«, seznam enačb, seznam poti slik »img_paths«, spremenljivko »numOfData« in nastavi zastavico »augment".

```
def enacbeinpoti(self):
    for k in range(len(self.vrste)):
        k+=1
        vrstica = self.vrste[k - 1]
        letter = "  "
        i = vrstica.rfind(letter)
        slika = (vrstica[:i])
        h=slika
        ground_truth = (vrstica[i + 1:])
        if "2013" in self.your_path:
            kali = 'test/2013'
        elif "2014" in self.your_path:
            kali='test/2014'
        else:
            kali='train'
        druge_path = pathlib.Path(self.your_path).parent.joinpath(kali)
        path_slike = druge_path.joinpath(f'{h}.png').as_posix()
        self.equations.append(ground_truth)
        self.img_paths.append(path_slike)
    return self.equations, self.img_paths
```

Slika 11: Prikaz funkcije, ki določi poti do ustrezne slike

Na sliki (Slika 11) je prikazana funkcija »enacbeinpoti(self)«, ki obdela vsako vrstico datoteke ter poda enačbo in pot do ustrezne slike. Nato to enačbo shrani v seznam enačb, pot pa v seznam poti slik.

```
def enacba_v_vektor(self):
    sam={0: 'K', 1: 'A', 2: 'I', 3: 'S', 4: 'T', 5: '/', 6: 'r', 7: 'a', 8:
', 46: '0', 47: 'd', 48: 'M', 49: 'E', 50: 'G', 51: 'B', 52: 'C', 53: 'p', 54
    dictionary = {y: x for x, y in sam.items()}
    self.razred_v_crka = sam
    self.crka_v_razred = dictionary
    for j,equation in enumerate(self.equations):
        enacba=list(equation)
        for i, char in enumerate(enacba):
            enacba[i]=dictionary[char]
        enacba.insert(0, dictionary["<S>"])
        enacba.insert(len(enacba), dictionary["<E>"])
        wakanda_1 = [dictionary["<P>"]] * (311-len(enacba))
        enacba.extend(wakanda_1)
        self.vektorsko.append(enacba)
    return self.vektorsko
```

Slika 12: Prikaz funkcije, ki pretvori enačbe v vektorje

Na sliki (Slika 12) je prikazana funkcija »enacba_v_vektor(self)«, ki pretvori vrstice enačb v vektorski zapis, ki ga nato shrani v seznam vektorskih zapisov.

```
def __getitem__(self, n):
    vektor=self.vektorsko[n]
    enacba_v = torch.tensor(vektor)
    slika = self.slike_spomin[n]
    if self.augment:
        slika = self.transforms(slika)
    return slika, enacba_v
```

Slika 13: Prikaz funkcije za dostopanje elementa v zbirki podatkov

Na sliki (Slika 13) je prikazana funkcija »__getitem__« , ki se pokliče, ko dostopamo do elementa v zbirki podatkov z uporabo oglatih oklepajev. Vrne tenzor enačbe in ustrezni tenzor slike.

```
def __len__(self):
    self.numOfData=len(self.equations)
    return self.numOfData
```

Slika 14: Prikaz funkcije za izračun števila elementov

Na sliki 14 je prikazana funkcija »__len__« , ki vrne število elementov v podatkovni zbirki.

```python
def augmentacija_init(self):
    color_jitter_kwargs = {"brightness": 0.4, "contrast": 0.4}
    random_affine_kwargs = {
        "degrees": 3,
        "shear": 6,
        "scale": (0.95, 1),
        "fill" : 1.0,
        "interpolation": transforms.InterpolationMode.BILINEAR,
    }
    random_perspective_kwargs = {
    "distortion_scale": 0.2,
        "p": 0.5,
        "fill" : 1.0,
        "interpolation": transforms.InterpolationMode.BILINEAR,
    }
    gaussian_blur_kwargs = {"kernel_size": (3, 3), "sigma": (0.1, 1.0)}
    sharpness_kwargs = {"sharpness_factor": 2, "p": 0.5}

    self.transforms = transforms.Compose(
        [
            transforms.RandomCrop(
                size=(256, 256), padding=None, pad_if_needed=True, fill=1.0, padding_mode="constant"
            ),
            transforms.RandomPerspective(**random_perspective_kwargs),
            transforms.ColorJitter(**color_jitter_kwargs),
            transforms.GaussianBlur(**gaussian_blur_kwargs),
            transforms.RandomAdjustSharpness(**sharpness_kwargs),
        ]
    )
```

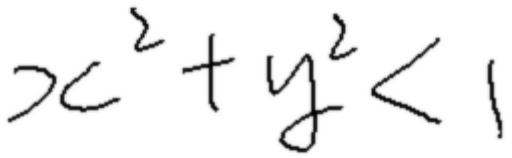
Slika 15: Prikaz funkcije za augmentacijo slik

Slika 15 prikazuje funkcijo »augmentacija_init(self)«, ki se uporablja za razširitev podatkov s preoblikovanjem slik (npr. vrtenje, obračanje).

```python
from text_recognizer.data.base_data_module import BaseDataModule
class EnacbeDataModule(BaseDataModule):
    def __init__(self, args):
        super().__init__(args)
        augment = self.args.get("augment_data", "true").lower() == "true"
        listpotk =[
                '../data/downloaded/enacbe/data/groundtruth_train.tsv',
                '../data/downloaded/enacbe/data/groundtruth_2013.tsv',
                '../data/downloaded/enacbe/data/groundtruth_2014.tsv',
                ]
        for potko in listpotk:
            if "2013" in potko:
                dataset_enacb_validacija = DatasetEnacb(potko) # 2013
            elif "2014" in potko:
                dataset_enacb_testiranje = DatasetEnacb(potko) # 2014
            else:
                dataset_enacb_trening = DatasetEnacb(potko, augment)
        dataset_enacb_trening.nalozi()
        dataset_enacb_validacija.nalozi()
        dataset_enacb_testiranje.nalozi()
        slika, enacba_vektor = dataset_enacb_trening[0]
        self.input_dims = slika.shape
        self.output_dims = enacba_vektor.shape + (1,)
        razred_v_crka = dataset_enacb_trening.razred_v_crka
        self.mapping = [
                razred_v_crka[i] for i in range(len(razred_v_crka))]
        self.data_train = dataset_enacb_trening
        self.data_val = dataset_enacb_validacija
        self.data_test = dataset_enacb_testiranje
    @staticmethod
    def add_to_argparse(parser):
        BaseDataModule.add_to_argparse(parser)
        parser.add_argument("--augment_data", type=str, default="false")
        return parser
```

Slika 16: Prikaz razreda, ki naloži in obdela podatke

Na sliki 16 je prikazan razred, ki je zgrajen z uporabo razreda podatkovne množice PyTorch in se uporablja za nalaganje in obdelavo podatkov za učenje ali testiranje nevronske mreže in za optično prepoznavanja znakov za matematične enačbe.

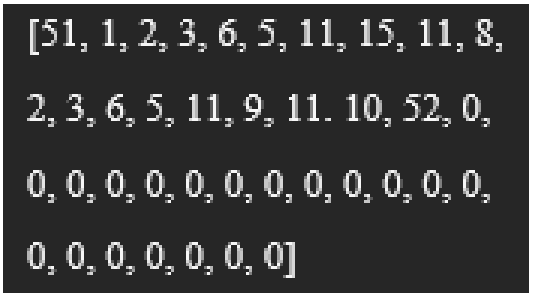


Slika 17: Primer podatkov za učenje

Slika 17 prikazuje primer slike v naši zbirki podatkov, ki smo jih uporabljali za učenje našega modela. Besedilo, ki je priloženo temu primeru, je:

»KAIST/TrainData1_6_sub_3 x^{2} + y^{2} \lt 1«

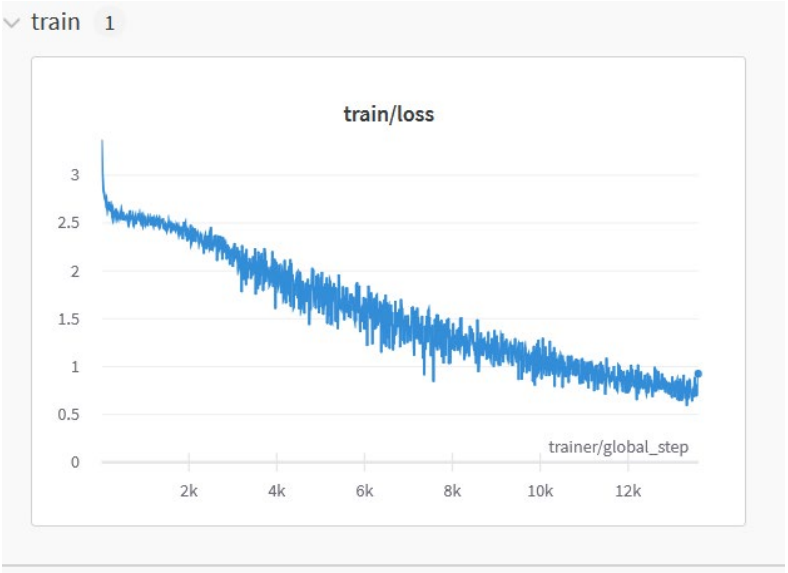
Besedilo je razdeljeno na dva dela: besedilo do presledka in besedilo za presledkom. Del do presledka predstavlja pot do slike, del po presledku pa Latex zapis, ki opisuje enačbo na tej sliki. Latex zapis smo nato preoblikovali v vektorsko obliko, da smo ga lahko vstavili v nevronske mreže. To smo naredili tako, da smo vsakemu simbolu dodelili številko, določili števili za začetek in konec enačbe ter zastavili število, ki je zapolnilo vsa ostala mesta. Nato smo poiskali najdaljšo enačbo v naši zbirki, ki določa dolžino naše matrike, ki smo jo zapolnili s števili. Na primer določimo, da je najdaljša enačba v zbirki 40 znakov, da znak »x« predstavlja število 1, znak »^« število 2, znak »{« število 3, znak »}« število 5, znak »2« število 6, znak »+« število 15, znak »y« število 8, znak »\lt« število 9, znak »1« število 10, presledek število 11, začetek število 51 in konec število 52 ter naj bo število za zapolnitev 0. V tem primeru bi naša koda za zgornji Latex zapis izpisala vektorsko obliko, kot je prikazano na sliki (Slika 18).



Slika 18: Vektorska oblika

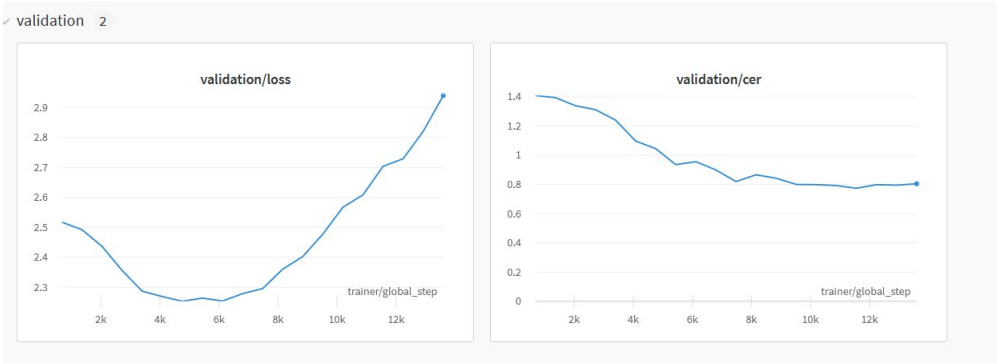
4. REZULTATI

S pomočjo izhodnega modela, opisanega zgoraj, in naših dodatkov nam je uspelo ustvariti model, ki se uspešno uči iz podanih podatkov. S pomočjo mentorjev smo model začeli učiti lokalno. Po kratkem učenju je naš model prepoznaval in zapisoval ročno zapisane matematične enačbe s 35-odstotno natančnostjo, kar pa je manjša natančnost, kot bi si jo želeli. Rezultati, ki smo jih dobili, so bili pričakovani, saj nismo imeli na voljo dovolj časa, podatkov ali opreme, ki bi nam omogočali dolgotrajno in učinkovito učenje, s katerim bi lahko prišli do želenih rezultatov.



Slika 19: Prikaz izgube pri učenju modela

Na slikah so prikazani rezultati učenja, ki smo jih dobili z našim modelom. Kot lahko vidimo na sliki 19, ki prikazuje, kako se je spreminjala izguba skozi čas učenja, se je izguba skozi čas učenja zmanjšala, kar pomeni, da se je model uspešno učil in nadgrajeval, vendar je bila na koncu učenja izguba še vedno previsoka, da bi bil lahko model zelo natančen.



Slika 20: Prikaz rezultatov validacije učenja

Na sliki 20 so prikazani rezultati validacije učenja, kjer lahko opazimo, da je med časom učenja izguba začela naraščati, kar nam sporoča, da je med učenjem prišlo do napake, kar je razlog za slabo natančnost modela. Na levem grafu imamo prikazano validacijo izgube, ki se izračuna s primerjanjem

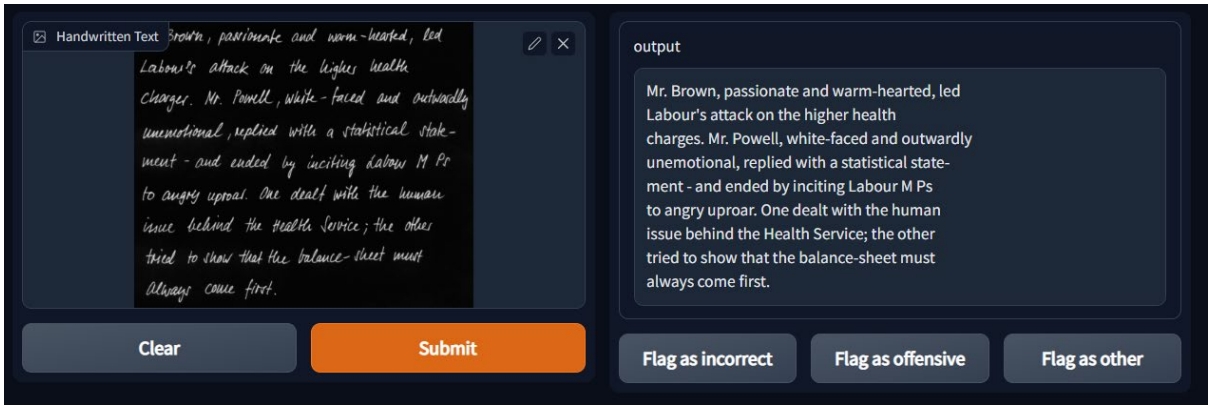
napovedanega izhoda modela in dejanskega izhoda validacije. Na desnem pa validacijo CER ali pogostost napake pri znakih (angl. Character Error Rate), kjer se zapiše število napačnih znakov v napovedih modela v primerjavi z osnovno resnico (angl. Ground Truth).

5. UGOTOVITVE IN SKLEPI

Na osnovi rezultatov in znanja, ki smo jih pridobili z raziskovalnim delom, smo prišli do naslednjih ugotovitev:

- Hipoteza 1: S pomočjo nevronskih mrež je možno ustvariti program, ki lahko iz fotografije prepozna ročno pisavo in jo zapiše v digitalni obliki.

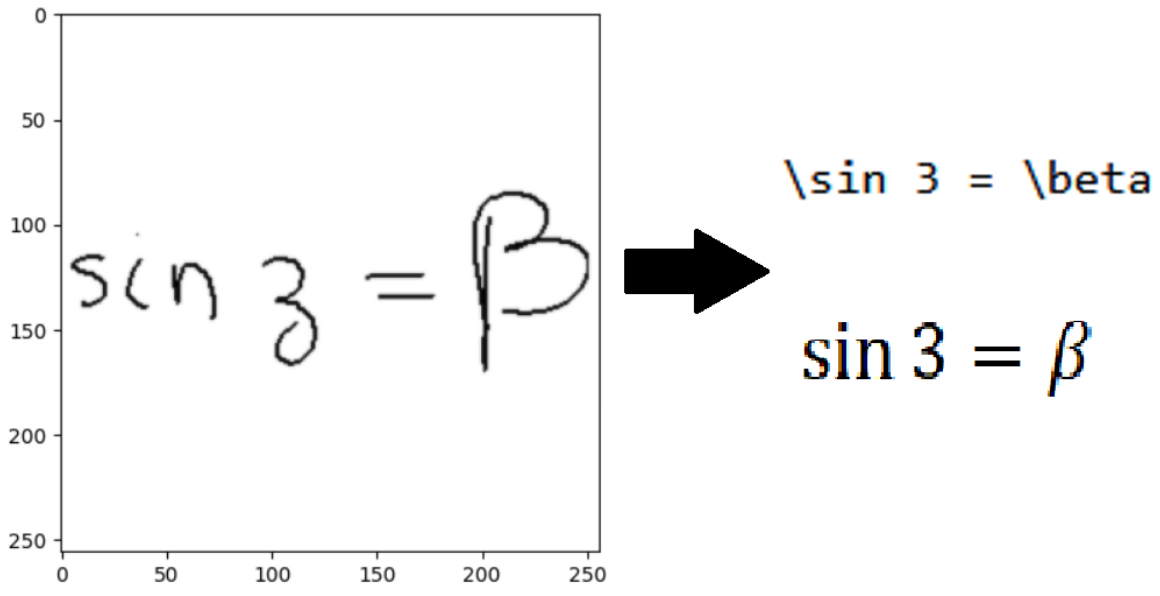
Hipoteza je potrjena. Med raziskovanjem smo na spletu našli veliko modelov, ki temeljijo na nevronskih mrežah in z visoko natančnostjo prepoznavajo ročno zapisano pisavo in jo prepisujejo v digitalno pisavo. Model z najvišjo natančnostjo smo omenili in opisali zgoraj. Na sliki 21 sta prikazani ročna pisava, ki smo jo vstavili v model, in računalniška pisava, ki jo je izpisal.



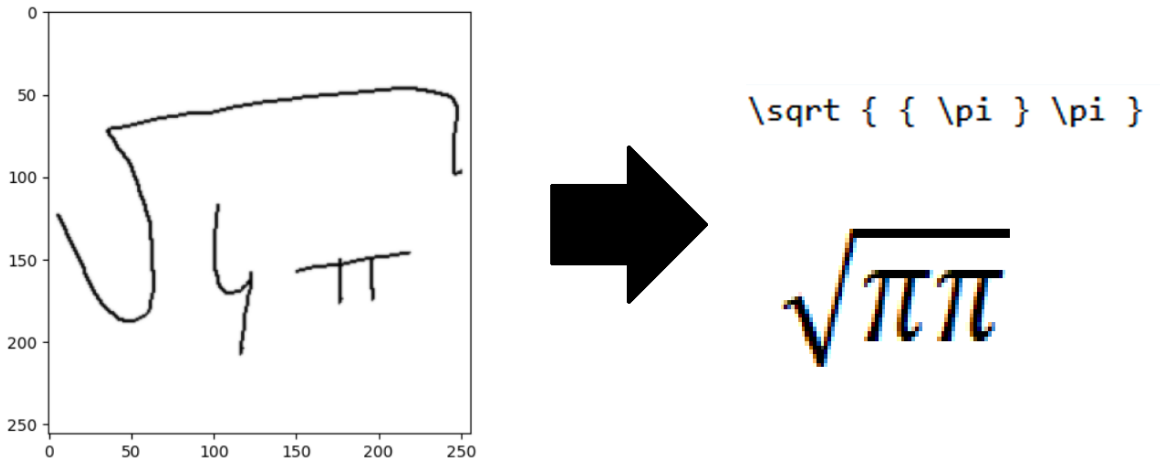
Slika 21: Prikaz delovanja najdenega modela

- Hipoteza 2: Nadgrajen model lahko iz fotografije prepozna tudi matematične simbole in rezultate zapiše v Latex jeziku.

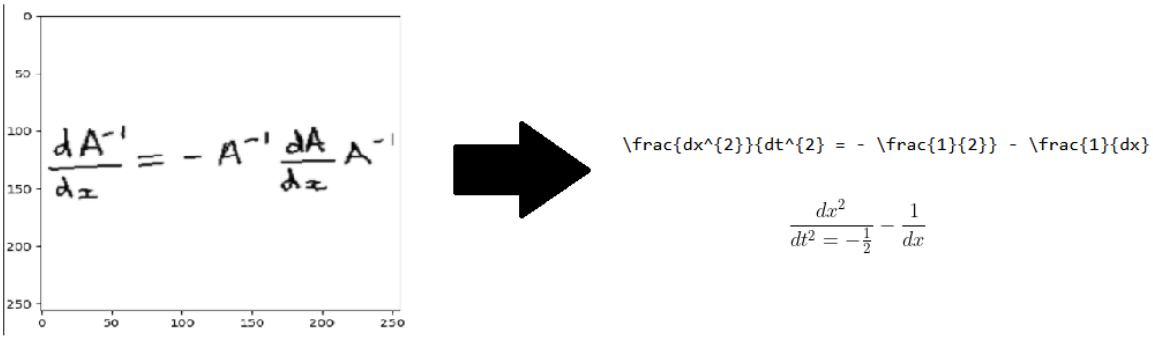
Hipoteza je potrjena. S tem da smo nadgradili zgornji model, nam je uspelo, da se model uspešno uči in prepoznava ročno zapisane matematične simbole in jih zapiše v Latex jeziku.



Slika 22: Primer rezultata modela (1)



Slika 23: Primer rezultata modela (2)



Slika 24: Primer rezultata modela (3)

Na slikah (Slika 22, Slika 23 in Slika 24) so prikazani primeri rezultatov našega modela glede na vstavljeno sliko. Na levi strani so slike, ki jih je model sprejel, na desni pa sta napovedana slika modela in Latex zapis, ki ju je podal model.

6. NADALJNJE RAZISKAVE

Naslednji korak pri naši raziskavi bi bil doseči večjo natančnost modela (vsaj 90-odsotno) in iz njega razviti aplikacijo ali program, s katerim bi lahko kdor koli vstavljal slike in uporabljal naš model, da bi prepisoval matematične simbole v Latex jezik. Večjo natančnost bi dobili z optimiziranjem učenja tako, da bi model učili dlje časa z več podatki in bi vhodne slike preoblikovali na enako velikost ter jih združili v paragrafe, s tem bi dosegli učinkovitejše učenje. V našem primeru smo model učili s slikami z belim ozadjem in črno pisavo, zato lahko prepozna samo pisavo iz tovrstnih primerov. Za nadaljnje raziskave bi lahko prilagodili model, da bi prepoznal tudi različne barve pisave na različnih podlagah, ker bi tako lahko prepoznal pisavo najrazličnejših primerov in ne bi bil omejen na črno pisavo z belim ozadjem.

7. ZAKLJUČEK

V sklopu raziskovalne naloge smo raziskali in se poučili o tem, kako delujejo strojno učenje in nevronske mreže na splošno ter za kaj vse se lahko uporabljajo. Sledila sta iskanje že obstoječih modelov za prepoznavanje ročne pisave in analiza teh modelov. Natančno smo pregledali model z največjo natančnostjo ter postavili načrt dela. Najden model smo nadgradili in mu omogočili učenje na drugih podatkih z namenom, da bi se lahko naučil prepoznavati tudi matematične simbole. Naš nadgrajeni model smo uspešno naučili in na koncu vse zbrane informacije ter rezultate zbrali in jih zapisali v seminarski nalogi.

Zaključimo lahko, da je strojno učenje hitro razvijajoče se področje s široko paleto aplikacij in koristi. Čeprav lahko revolucionarno spremeni številne panoge in nam olajša življenje, predstavlja tudi izzive in etične pomisleke, ki jih je treba obravnavati. Ker strojno učenje še naprej napreduje, je pomembno upoštevati posledice njegove uporabe in razviti odgovorne prakse za njegovo uporabo.

8. VIRI IN LITERATURA

[1] Ankit, U. (28. junij 2022). *Transformer Neural Networks: A Step-by-Step Breakdown*. Pridobljeno iz BuiltIn: https://builtin.com/artificial-intelligence/transformer-neural-network

[2] Brglez, D. (2019). Pridobljeno iz http://pefprints.pef.uni-lj.si/5710/1/Magistrska_Domen_Brglez.pdf

[3] Burns, E. (marec 2021). *Machine learning*. Pridobljeno iz TechTarget: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[4] Dhalla, A. (1. marec 2021). *The Complete Mathematics of Neural Networks and Deep Learning*. Pridobljeno iz YouTube: https://www.youtube.com/watch?v=Ixl3nykKG9M&t=5932s

[5] *Full Stack Deep Learning - Course 2022*. (25. julij 2022). Pridobljeno iz Full Stack Deep Learning: https://fullstackdeeplearning.com/course/2022/

[6] *Kaj je umetna inteligenca in zakaj je pomembna?* (14. maj 2021). Pridobljeno iz Mreža nevladnih organizacij za vključujočo informacijsko družbo (NVO-VID): https://www.informacijska-druzba.org/2021/05/14/kaj-je-umetna-inteligenca-in-zakaj-je-pomembna/

[7] MinnaLearn. (2019). *Nevronske mreže.* Pridobljeno iz Elements of AI: https://course.elementsofai.com/sl/5

[8] *Nevronske mreže*. (21. december 2018). Pridobljeno iz Dijaski.net: https://dijaski.net/gradivo/rif_ref_nevronske_mreze_01

[9] *Strojno učenje*. (28. januar 2023). Pridobljeno iz Wikipedija: https://sl.wikipedia.org/wiki/Strojno_u%C4%8Denje

[10] Žontar, R. (19. februar 2022). *Ali se lahko stroji naučijo več kot ljudje?* Pridobljeno iz Slovenec: https://www.slovenec.org/2022/02/19/ali-se-lahko-stroji-naucijo-vec-kot-ljudje/

[11] 3Blue1Brown. (5. oktober 2017). *But what is a neural network? | Chapter 1, Deep learning?* Pridobljeno iz Youtube; https://www.youtube.com/watch?v=aircAruvnKk

[12] freeCodeCamp.org. (30. Julij 2020). *Deep Learning Crash Course for Beginners*. Pridobljeno iz Youtube: https://www.youtube.com/watch?v=VyWAvY2CF9c

[13] Aurelien, G. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc.

9. PRILOGA

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import torch
import torchvision
import torchvision.transforms as transforms
import pathlib


class DatasetEnacb:
    def __init__(self, pot_do_podatkov=r"", augment=False):
        self.dictionary = {}
        self.equations = []
        self.vektorsko = []
        self.img_paths = []
        self.your_path = pot_do_podatkov
        self.numOfData = 0
        print(self.your_path)
        self.datoteka = open(self.your_path)
        self.vrste = self.datoteka.read().splitlines()
        self.augment = augment
        self.slike_spomin = {}
        if self.augment:
            self.augmentacija_init()

    def enacbeinpoti(self):
        for k in range(len(self.vrste)):
            k += 1
            vrstica = self.vrste[k - 1]
            letter = " "
            i = vrstica.rfind(letter)
            slika = (vrstica[:i])
            h = slika
            ground_truth = (vrstica[i + 1:])
            if "2013" in self.your_path:
                kali = 'test/2013'
            elif "2014" in self.your_path:
                kali = 'test/2014'
            else:
                kali = 'train'

            druge_path = pathlib.Path(self.your_path).parent.joinpath(kali)
            path_slike = druge_path.joinpath(f'{h}.png').as_posix()
            self.equations.append(ground_truth)
            self.img_paths.append(path_slike)
        return self.equations, self.img_paths

    def enacba_v_vektor(self):
        sam = {0: 'K', 1: 'A', 2: 'I', 3: 'S', 4: 'T', 5: '/', 6: 'r', 7:
'a', 8: 'i', 9: 'n', 10: 'D', 11: 't',
               12: '1', 13: '_', 14: '6', 15: 's', 16: 'u', 17: 'b', 18:
'3', 19: '\t', 20: 'x', 21: '^', 22: '{',
               23: '2', 24: '}', 25: ' ', 26: '+', 27: 'y', 28: '\\', 29:
'l', 30: '9', 31: '8', 32: 'm', 33: 'g',
               34: 'h', 35: 'o', 36: 'w', 37: 'f', 38: 'c', 39: '4', 40: '-
', 41: '5', 42: 'e', 43: '(', 44: ')', 45:
                   '7', 46: '0', 47: 'd', 48: 'M', 49: 'E', 50: 'G', 51:
```

```
'B', 52: 'C', 53: 'p', 54: '=', 55: '[',
              56: ']', 57: 'q', 58: 'z', 59: 'F', 60: 'j', 61: 'k', 62:
'X', 63: 'R', 64: 'H', 65: 'V', 66: ',',
              67: 'P', 68: 'N', 69: 'L', 70: 'Y', 71: '.', 72: 'v', 73:
'"', 74: '|', 75: '!', 76: 'W', 77: "'",
              78: ';', 79: '<', 80: 'O', 81: '>', 82: '<S>', 83: '<E>',
84: '<P>'}
        dictionary = {y: x for x, y in sam.items()}
        self.razred_v_crka = sam
        self.crka_v_razred = dictionary
        for j, equation in enumerate(self.equations):
            enacba = list(equation)
            for i, char in enumerate(enacba):
                enacba[i] = dictionary[char]
            enacba.insert(0, dictionary["<S>"])
            enacba.insert(len(enacba), dictionary["<E>"])
            g = [dictionary["<P>"]] * (311 - len(enacba))
            enacba.extend(g)
            self.vektorsko.append(enacba)
            # vektorsko=np.asarray(vektorsko)
            # novi_list=np.asarray(enacba)
        # self.vektorsko=np.asarray(self.vektorsko)
        return self.vektorsko

    def nalozi(self):
        self.enacbeinpoti()
        self.enacba_v_vektor()
        for n, pot_tslike in enumerate(self.img_paths):
            slika = torchvision.io.read_image(pot_tslike) / 255.
            self.slike_spomin[n] = slika

    def __getitem__(self, n):
        # slika = mpimg.imread(self.img_paths[n])
        # slika=mpimg.imread(self.img_paths[n])
        # vektor = self.vektorsko[n]
        vektor = self.vektorsko[n]
        enacba_v = torch.tensor(vektor)
        slika = self.slike_spomin[n]
        if self.augment:
            slika = self.transforms(slika)
        return slika, enacba_v

    def __len__(self):
        self.numOfData = len(self.equations)
        return self.numOfData

    def augmentacija_init(self):
        color_jitter_kwargs = {"brightness": 0.4, "contrast": 0.4}
        random_affine_kwargs = {
            "degrees": 3,
            "shear": 6,
            "scale": (0.95, 1),
            "fill": 1.0,
            "interpolation": transforms.InterpolationMode.BILINEAR,
        }
        random_perspective_kwargs = {
            "distortion_scale": 0.2,
            "p": 0.5,
            "fill": 1.0,
            "interpolation": transforms.InterpolationMode.BILINEAR,
        }
```

```
        gaussian_blur_kwargs = {"kernel_size": (3, 3), "sigma": (0.1, 1.0)}
        sharpness_kwargs = {"sharpness_factor": 2, "p": 0.5}

        self.transforms = transforms.Compose(
            [
                transforms.RandomCrop(
                    size=(256, 256), padding=None, pad_if_needed=True,
fill=1.0, padding_mode="constant"
                ),
                # transforms.RandomAffine(**random_affine_kwargs),
                transforms.RandomPerspective(**random_perspective_kwargs),
                transforms.ColorJitter(**color_jitter_kwargs),
                transforms.GaussianBlur(**gaussian_blur_kwargs),
                transforms.RandomAdjustSharpness(**sharpness_kwargs),
            ]
        )


from text_recognizer.data.base_data_module import BaseDataModule


class EnacbeDataModule(BaseDataModule):
    def __init__(self, args):
        super().__init__(args)
        augment = self.args.get("augment_data", "true").lower() == "true"
        listpotk = [
            '../data/downloaded/enacbe/data/groundtruth_train.tsv',
            '../data/downloaded/enacbe/data/groundtruth_2013.tsv',
            '../data/downloaded/enacbe/data/groundtruth_2014.tsv',
        ]
        for potko in listpotk:
            if "2013" in potko:
                dataset_enacb_validacija = DatasetEnacb(potko)
            elif "2014" in potko:
                dataset_enacb_testiranje = DatasetEnacb(potko)
            else:
                dataset_enacb_trening = DatasetEnacb(potko, augment)

        dataset_enacb_trening.nalozi()
        dataset_enacb_validacija.nalozi()
        dataset_enacb_testiranje.nalozi()
        slika, enacba_vektor = dataset_enacb_trening[0]

        self.input_dims = slika.shape
        self.output_dims = enacba_vektor.shape + (1,)

        razred_v_crka = dataset_enacb_trening.razred_v_crka
        self.mapping = [
            razred_v_crka[i] for i in range(len(razred_v_crka))]

        self.data_train = dataset_enacb_trening
        self.data_val = dataset_enacb_validacija
        self.data_test = dataset_enacb_testiranje

    @staticmethod
    def add_to_argparse(parser):
        BaseDataModule.add_to_argparse(parser)
        parser.add_argument("--augment_data", type=str, default="false")
        return parser
```