Inteligenta Artificiala:Monochrome Dreams Classification

Tonica Marius-Luis,341

1.Introducere

Aceasta documentatie este facuta asupra proiectului de laborator din cadrul curslui Inteligenta Artificiala, partea de Machine Learning. Proiectul a fost sustinut sub cadrul unei competitii unde notele au fost acordate depinzand de ce loc au ocupat. Mai multe detalii despre reguli se pot gasi pe Kaggle: https://www.kaggle.com/c/ai-unibuc-24-22-2021/overview .

2.Tehnologii folosite

Pentru creearea retelele neuronale am folosit libraria TensorFlow, o librarie free open-sourced care poate calcula gradientul oricarei functii diferentiabile si respectiv Keras, un deep-learning API integrat in TensorFlow care ne permite sa creem si sa antrenam cu usurinta modele.

Pe langa acestea, am folosit libraria Numpy si Pandas pentru formatarea usoara a datelor. Numpy consista in matrici multidimensionale formate din orice tip de obiect cu functii spre manipularea usoara a datelor. Impreuna cu libraria matplotlib pot inlocui software-ul numit MatLab. Pandas ne ofera posibilitatea de a creea DataFrame-uri in care, precum Numpy, putem manipula foare usor datele folosind functii specifice librariei.

Alte librarii folosite care sunt deja integrate in python(trebuiesc doar exportate) sunt CSV, OS si shutil.

3.Algoritmul

Datele de training constau in 30001 de poze, ele fiind caracterizate in una din cele 9 clase. Peste poze s-a aplicat un filtru alb-negru iar pozele nu pot fi caracterizate cu ochiul liber. De asemenea, am mai primit si 5000 de poze de validare cu aceeasi caracteristica ca si cele de training. In final, dataset-ul pentru test consta in 5000 de poze care, din nou, au acelasi caracter ca si cele de mai sus.

Dupa ce am extractat pozele si le-am pus intr-un folder numit data, am creeat o functie “setup” in care fac 2 lucruri:

1.Pentru folderul de training si validation creez 9 foldere(cu nume de la 0 la 8) si separ pozele in functie de labeluri (label 0= folder 0 si tot asa..) iar pentru folderul test mai creez un subfolder cu numele de “unknown”. Lucrul asta va veni in folos in cateva momente.

2.Functia returneaza un DataFrame prin intermediul lui Pandas. Acesta este format din 2 coloane, prima coloana continand numele pozelor iar a 2 coloana continand labelurile.

```
def setup(generic_path, label_path, is_test):
    col_names = ["id", "label"]
    labels_list = pd.read_csv(label_path, names=col_names)
    os.chdir(generic_path)
    if not is_test:
        if os.path.isdir('0') is False:
            for i in range(9):
                if os.path.isdir(f'{i}') is False:
                    os.makedirs(f'{i}')
            for i, row in labels_list.iterrows():
                photo = labels_list.iloc[i, 0]
                label = labels_list.iloc[i, 1]
                shutil.copy(photo, f'{label}')
    else:
        if os.path.isdir('unknown') is False:
            os.makedirs('unknown')
            for i, row in labels_list.iterrows():
                shutil.copy(labels_list.iloc[i, 0], 'unknown')
        del labels_list["label"]
    os.chdir('../../')
    return labels_list
```

Dupa sortarea imaginilor am decis sa creez batch-uri care vor fi folosite de masina. M-am gandit din timp la performanta modelului si la reducerea overfittingului, asa ca m-am folosit de o clasa integrata in Keras care poate sa faca ambele task-uri deodata: ImageDataGenerator.

```
train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(
    directory=train_path,
    classes=['0', '1', '2', '3', '4', '5', '6', '7', '8'],
    target_size=(32, 32),
    batch_size=100,
    shuffle=True)
validation_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(
    directory=validation_path,
    classes=['0', '1', '2', '3', '4', '5', '6', '7', '8'],
    target_size=(32, 32),
    batch_size=100,
    shuffle=True)
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(
    directory=test_path,
    classes=['unknown'],
    target_size=(32, 32),
    batch_size=10,
    shuffle=False)
```

Aceasta clasa ia pozele din din fiecare folder, le asociaza labeluri (folosinta creeri mai multor foldere), le schimba rezolutila in caz ca imaginile au rezolutii diferite si are posibilitatea de a le amesteca, ca la urma sa returneze un batch. Dar inainte de a le returna, acesta aplica un filtru de post-proccesing, mai exact filtrul aplicat de VGG16, un model folsit pentru fine-tuning.

In final, pozele arata in modul acesta:

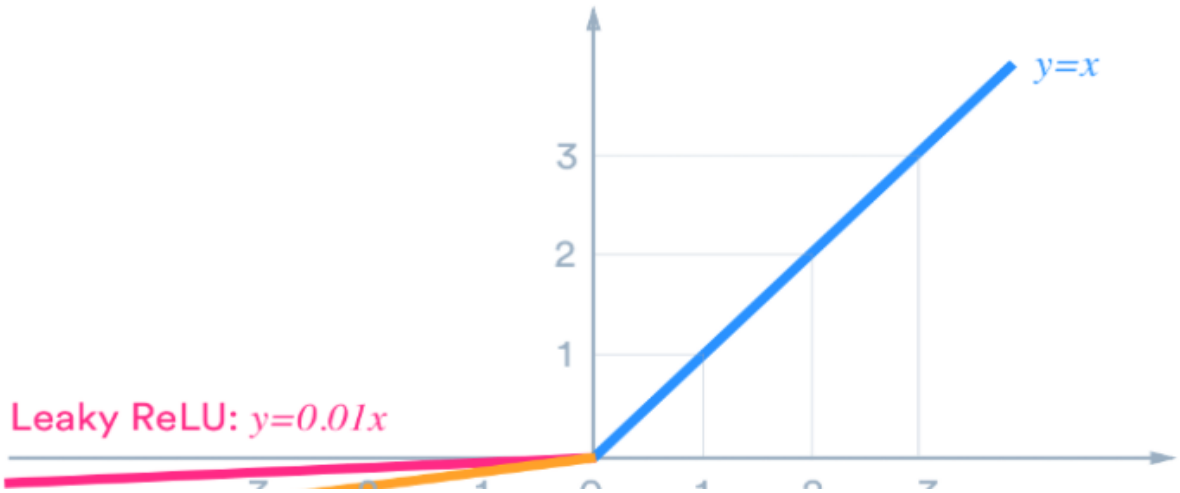
Pentru ochiul liber, pozele arata mult mai rau ca inainte, dar pentru un model acest lucru ajuta foarte mult. Aceasta ofera sansa de a gasi asemanari intre poze.

In continuare am construit modelul:

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.1), padding='same', input_shape=(32, 32, 3)),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.1), padding='same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(units=9, activation='softmax')
])
```

Acesta este format din urmatoarele layere:

1. Primul Layer este un Conv2D cu 32 de filtere si un kernel size de 3x3. In Conv2D, o matrice de dimensiunea kernelului se “plimba” prin matricea pixelilor si face calcule asupra acestora, in final sa adune tot intr-un singur pixel nou. Procesul se repeta pana se ajunge la finalul pozei. Dar asta ar insemna ca poza pierde pixeli de pe margine, creeand posibilitatea de a pierde informatii utile. Aici vine in folos “padding”. Padding adauga un border de 0-uri pe matricea pixelilor, aducand imaginea la rezolutia initiala. Ultima functie ramasa este LeakyReLU, o functie de activare si modificarea functiei ReLU(Rectified Linear Unit), care transforma input-ul intre maximul dintre 0 si el insusi. LeakyReLU fixeaza problema de “dyingReLU” unde un neuron are sansa sa “moara”(ramane blocat la valori negative, aducand nimic retelei), creeand o mica “rampa” pentru valorile negative(alpha).



2. MaxPool2D reduce marimea pozei respectand in mare parte detaliile. Aceasta este o matrice de 2x2(pool_size) care lafel ca la Conv2D face calcule asupra matricei de pixeli. Valoarea “strides” semnifica cati pixeli se va plima matricea dupa fiecare calcul.MaxPool2D reduce timpul si overfitting-ul.

3.Flatten() transforma forma imputului dintr-o inmultire intr-un numar intreg.

4.Ultimul layer de output este Dense. Layerul dense are conexiuni cu toti neuronii dinaintea lui. Metoda de activare este softmax transforma un vector intr-un alt vector de probabilitati categorice, fiind de folos in ultimul layer deoarece rezultatul este interpretat ca o probabilitate.

Urmeaza compilarea modelului:

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```

Pentru optimizator am folosit Adam(Adaptive moment estimation) si este o varianta mai buna a “stochastic gradient descent” pentru modificarea weight-urilor. Adam a preluat avantajele optimizatorilor AdaGrad si RMSProp. Learning rate-ul este setat la 0.0001, acesta facand pasi foarte mici spre probabilitatea cea mai buna. In acest mod ne asiguram ca optimizer-ul nu va trece peste valorile perfecte.

Catergorical crossentropy calculeaza pierderile intre label-uri si predictii.

Accuracy calculeaza cat de frecventa este egalitatea intre predictie si label.

Pentru antrenare vom folosi functia fit:

```
model.fit(x=train_batches,
          steps_per_epoch=len(train_batches),
          validation_data=validation_batches,
          validation_steps=len(validation_batches),
          epochs=20,
          verbose=2)
```

Unde:

- X = training batches, adica batch-urile formate in urma aplicarii functiei ImageDataGenerator descris mai sus.
- Validation data = validation batches, lafel ca la training
- Steps per epoch/validation steps = cati pasi face pe epoca, fiind egal cu marimea batch-urilor.
- Epochs = de cate ori se repeta procesul
- Vebrose = informatii imprimate

Iar in final, am aplicat functia de predictie:

```
predictions = model.predict(x=test_batches, steps=len(test_batches), verbose=1)
```

- Test batches = lafel ca training si validation batches, doar ca fara labeluri.
- Steps = lafel ca validation steps
- Vebrose = informatii imprimate.

Functia predictions retuneaza mai multi vectori de tip one-hot(multi de 0 si un 1, acel 1 fiind pe pozitia numarului indicat). Exemplu:[0, 0, 0, 1, 0] este 3(numerotarea incepe de la 0). Pentru a transforma un one-hot encoder intr-un numar am aplicat functia argmax() din numpy.

```
predictions2 = (np.argmax(predictions, axis=1))
```

Iar ca ultima am pus rezultatele intr-un csv folosindu-ma de numpy-ul predictions2 si dataframe-ul facut pentru folderul test.

```
with open('sample_submission.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["id", "label"])
    i = 0
    for x in predictions2:
        writer.writerow([test_dataframe.iloc[i, 0], x])
        i = i+1
```

Cateva incercari:

1.Varianta cea mai simpla,a oferit si cele mai bune rezultate, chiar daca cu ea a aparut si overfitting:

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.1), padding='same', input_shape=(32, 32, 3)),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Conv2D(filters=64, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.1), padding='same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(units=9, activation='softmax')
])
```

```
Epoch 20/20
301/301 - 15s - loss: 0.4311 - accuracy: 0.8532 - val_loss: 0.7727 - val_accuracy: 0.7642
500/500 [==============================] - 2s 3ms/step
```

2.Am adaugat functia Dropout().Aceasta functie are sansa sa “inghete” anumiti neuroni, oferind o posibilitate la o performanta mai mare. Am pastrat aceeasi hiperparametrii. A avut o mica pierdere in performata.

```
model = Sequential([
    Conv2D(filters=16, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.1), padding='same', input_shape=(32, 32, 3)),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Dropout(0.5),
    Conv2D(filters=32, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.1), padding='same'),
    MaxPool2D(pool_size=(2, 2), strides=2),
    Flatten(),
    Dense(units=9, activation='softmax')
])
```

```
301/301 - 19s - loss: 0.8332 - accuracy: 0.7117 - val_loss: 0.7627 - val_accuracy: 0.7370
500/500 [==============================] - 2s 3ms/step
```

Bibliografie:

https://keras.io/ : toate informatiile despre functii au fost luate de aici;

https://www.tensorflow.org/ ;

https://www.tensorflow.org/tutorials/images/classification#visualize_training_images : link-ul de unde am luat functia de plotare;

https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e : graficul pentru LeakyReLU;