Grigore Mihai-Cătălin - Grupa 242

Approach 1 - CNN

Rețelele neuronale convoluționale (CNN) sunt un tip specializat de rețele neuronale artificiale, proiectate pentru a procesa și a învăța din date structurate într-un mod grilă, cum ar fi imaginile. CNN-urile sunt foarte eficiente în problemele de clasificare și recunoaștere a imaginilor datorită capacității lor de a învăța și a extrage caracteristici relevante din datele de intrare în mod automat.

1. Preprocesarea datelor și augumentare

a. Preprocesare

Pentru a încărca imaginile, funcția load_images citește și redimensionează fiecare imagine la dimensiunea de IMG_SIZE (224x224 pixeli). Redimensionarea este importantă pentru a asigura că toate imaginile de intrare au aceeași dimensiune, independent de dimensiunea originală a imaginilor. Redimensionarea se realizează folosind funcția cv2.resize:

```
img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
```

Imaginile sunt apoi normalizate, împărțind valorile pixelilor la 255, pentru a aduce valorile în intervalul [0, 1]:

```
return np.array(images) / 255.0
```

Aceste etape de preprocesare asigură că valorile de intrare ale modelului sunt într-un interval adecvat și au dimensiuni consistente.

b. Augumentarea Datelor

Pentru a crește diversitatea datelor de antrenament și a îmbunătăți capacitatea modelului de a generaliza, este aplicată augmentarea datelor. Folosim ImageDataGenerator pentru a realiza transformări aleatorii pe imaginile de antrenament, cum ar fi rotația, deplasarea orizontală și verticală, zoom și flip horizontal și vertical:

```python
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True
)
datagen.fit(train_images)
```

Aceste transformări simulează variații reale ale datelor și ajută la prevenirea supraînvățării. Augmentarea datelor îmbunătățește capacitatea modelului de a învăța caracteristici relevante din diverse exemple și reduce impactul unor exemple neobișnuite sau zgomotoase.

2. Reprezentarea caracteristicilor

Modelul CNN este alcătuit din patru blocuri convoluționale, fiecare având o structură similară: un strat convoluțional, un strat de normalizare în lot și un strat de max-pooling. Această arhitectură permite extragerea de caracteristici ierarhice din imaginile de intrare și combină informația locală și globală pentru a obține o reprezentare înaltă a datelor.

După blocurile convoluționale, urmează straturile fully connected (dens) și dropout pentru a realiza clasificarea. Straturile dens transformă reprezentarea înaltă a caracteristicilor într-o predicție de clasă, iar straturile de dropout ajută la prevenirea supraînvățării.

```python
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(IMG_SIZE, IMG_SIZE, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation="relu"),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation="relu"),
    Dropout(0.5),
    Dense(256, activation="relu"),
    Dropout(0.5),
    Dense(1, activation="sigmoid")
])
```

3. Ajustarea hiperparametrilor

Ajustarea hiperparametrilor este esențială pentru a îmbunătăți performanța modelului și a evita probleme precum supraînvățarea sau convergența lentă. Vom explora diferite încercări de ajustare a hiperparametrilor și vom observa impactul acestora asupra performanței modelului.

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

a. Rata de învățare

Rata de învățare controlează cât de rapid modelul învață din date. O rată de învățare prea mare poate cauza oscilații în performanță și convergență lentă, în timp ce o rată de învățare prea mică poate face ca modelul să învețe foarte încet sau să rămână blocat în minime locale. În acest proiect, folosim un optimizator Adam cu o rată de învățare inițială de 0.0001.

Alte încercări:

- rata de învățare = 0.001. Dezavantaje: oscilații: modelul **sare peste minimul funcției de cost** în timpul optimizării și **oscilează** în jurul acestuia, nu învață corect, performanță slabă.
- rata de învățare = 0.00001. Dezavantaje: **convergență lentă**, necesită foarte mult timp și resurse de calcul, **blocare în minime locale**. Minimele locale sunt puncte în care funcția de cost este mai mică decât în vecinătatea imediată, dar nu sunt punctele minime ale întregii funcții de cost - performanță suboptimală

b. Funcția de performanță

Funcția de performanță măsoară cât de bine predicțiile modelului se potrivesc cu etichetele adevărate. În acest proiect, folosim funcția de pierdere "binary_crossentropy".

Alte încercări:

- hinge
- squared_hinge

c. Regularizare

Straturile de Dropout sunt folosite pentru a preveni overfitting și pentru a îmbunătăți capacitatea de generalizare a modelului. Principiul din spatele Dropout constă în dezactivarea aleatorie a unei proporții fixe de neuroni dintr-un strat în timpul antrenării. Astfel, în timpul propagării înainte și a propagării înapoi, acești neuroni dezactivați nu vor participa la procesul

de învățare. Aceasta înseamnă că, în timpul antrenării, un subset diferit de neuroni este folosit pentru a învăța modelul la fiecare pas.

rate = proporția de neuroni care vor fi dezactivați la fiecare pas de antrenare. Valorea optimă 0.5 - înseamnă că aproximativ 50% din neuroni vor fi dezactivați aleatoriu la fiecare pas de antrenare.

Alte încercări:

- rate = 0.25
- rate = 0.75

d. Epoci

Strategia pe care am folosit-o în antrenarea modelului a fost antrenarea pe un număr mare de epoci (e.g. 1000) și salvarea modelului cu cel mai bun f1_score într-un fișier separat. De asemenea, după un anumit număr de epoci (de exemplu când se observă overfitting sau o creștere foarte mare în validation_loss) se poate opri antrenarea.

```
f1 = f1_score(self.y_val, y_pred)
if f1 > self.best_f1:
    self.best_f1 = f1
    self.model.save_weights("best_model.h5")
    print(f"Epoch {epoch + 1} - F1 score improved: {f1:.4f}")
else:
    print(f"Epoch {epoch + 1} - F1 score not improved: {f1:.4f}, Best F1 score: {self.best_f1:.4f}")
```

e. Batch size

Dimensiunea batch-ului (batch size) se referă la numărul de exemple de antrenament utilizate într-o singură iterație / pas de antrenare. Alegerea dimensiunii potrivite pentru batch poate avea un impact semnificativ asupra performanței și a vitezei de antrenare a modelului.

Valoare optimă găsită: 32.

Alte încercări:

- batch_size = 16
- batch_size = 64

4. Evaluarea modelului

Evaluăm performanța modelului pe setul de validare, calculând precizia, recall-ul și matricile de confuzie pentru fiecare clasă. În acest scop, vom utiliza classification_report și confusion_matrix din biblioteca sklearn.metrics:

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
y_pred = model.predict(valid_images)
y_pred_classes = (y_pred > 0.5).astype(int)

report = classification_report(valid_labels["class"], y_pred_classes)
conf_matrix = confusion_matrix(valid_labels["class"], y_pred_classes)

print("Classification Report:")
print(report)
print("Confusion Matrix:")
print(conf_matrix)
```

```
63/63 [==============================] - 1s 8ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96      1712
           1       0.75      0.73      0.74       288

    accuracy                           0.93      2000
   macro avg       0.85      0.84      0.85      2000
weighted avg       0.93      0.93      0.93      2000

Confusion Matrix:
[[1644   68]
 [  79  209]]
```

Approach 2 - KNN

K-nearest neighbors (KNN) este un algoritm de învățare supervizată, simplu și neparametric, utilizat în probleme de clasificare și regresie. KNN funcționează pe baza distanțelor dintre exemplele de date și se bazează pe premisa că exemplele similare se găsesc în apropiere una de cealaltă în spațiul caracteristic.

Am încercat un model simplu de KNN refactorizând codul de la CNN cu următoarele modificări:

- ștergerea tuturor “importurilor” legate de CNN (TensorFlow, ImageDataGenerator etc.)
- import KneighborsClassifier din sklearn.neighbors
- ștergerea clasei F1Score și a generatorului de augumentare
- reshape pt imaginile de train, validation, test să fie 2D înainte de a antrena modelul
- am instanțiat KneighborsClassifier cu 5 vecini și am apelat fit pe datele de train

Rezultatul este următorul (evident, performanță mult mai scăzută decât CNN).

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      1712
           1       0.49      0.09      0.16       288

    accuracy                           0.86      2000
   macro avg       0.68      0.54      0.54      2000
weighted avg       0.81      0.86      0.81      2000

Confusion Matrix:
[[1684   28]
 [ 261   27]]
```