Deep Hallucination Classification

Cuprins

1 Introducere

1.1 Context

Scopul proiectului a fost să ne însușim cunoștințele de Inteligență Artificială acumulate la curs și laborator, precum și documentarea asupra lucrurilor noi. În acest document voi descrie procesul prin care am trecut pentru a crea un model capabil să clasifice imagini ce reprezintă halucinații, punându-le în una din cele 7 categorii date. Datele de antrenare sunt reprezentate de 8000 de imagini, cele de validare de 1173 imagini, iar cele de testare de 2819 imagini. Etichetele pentru cele 7 categorii nu au un nume sugestiv, ci reprezintă cifre de la 0 la 6.

1.2 Resurse utile

Pentru a mă documenta în timp ce am făcut proiectul, m-am folosit atât de resursele oferite la curs și laborator, cât și de câteva materiale online, printre care se numără și acest playlist. Am citit, de asemenea, articole online, precum acesta sau această serie de articole.

1.3 Abordări de învățare automată

Pentru a realiza task-ul, am folosit atât un model SVM(Support Vector Machine), cât și două CNN(Convolutional Neural Network) - unul pre-antrenat, iar celălalt antrenat de mine. Rezultatele cele mai bune au fost obținute folosind modelul CNN pre-antrenat (scor de 0.64914 pe Kaggle).

2 Prima abordare - SVM

2.1 Mod de implementare

Așa cum am menționat anterior, primul model pe care l-am făcut a fost un SVM, cu care am obținut scorul de 0.56534 pe Kaggle.

Pentru a-l implementa m-am folosit de laboratorul 4.

Inițial, am separat datele în 2 vectori pentru fiecare set de date, astfel:

- un vector în care stochez imaginile de antrenare (numit iduri_antrenare)
- un vector în care stochez imaginile de testare (numit iduri_testare)

- un vector în care stochez imaginile de validare (numit iduri_validare)
- un vector în care stochez label-urile imaginilor de antrenare (numit categorii_antrenare)
- un vector în care stochez label-urile imaginilor de validare (numit categorii_validare)

În continuare, am transformat imaginile (adică vectorii iduri_antrenare, iduri_testare, iduri_validare) în **numpy arrays** și i-am redimensionat folosind funcțiile **flatten()** și **reshape()** pentru a-i aduce din formă 4-dimensională în formă bidimensională.

Apoi, am definit **functia de normalizare** a datelor(folosindu-mă de ce am făcut la laborator), care se va comporta diferit în funcție de tipul de normă primit ca parametru. Astfel, în cazul standardizării, vectorii caracteristici sunt transformați astfel încât fiecare să aibă media 0 și deviația standard 1, pe când în cazul normalizării L1 sau L2 are loc scalarea individuală a vectorilor caracteristici corespunzători fiecărui caz astdel încât norma să devină 1.

Mai departe, am normalizat datele de antrenare, am creat modelul SVM, l-am antrenat și am generat predicțiile. Am făcut, de asemenea, și un Grid-search pentru a găsi mai ușor o combinație bună de parametri.

2.2 Testare cu diferiți parametri

Pentru a găsi o acuratețe cât mai bună, am făcut diferite teste, modificând pe rând valoarea parametrului C, tipul de normalizare sau tipul de kernel folosit.

- Parametrul C practic îi spune SVM-ului cât de mult să evite clasificarea greșită a fiecărui exemplu de antrenare.
- Parametrul tip_de _normalizare poate lua valoarea "standard", "l1" sau "l2" în funcție de tipul de normalizare pe care dorim să îl facem.
- Funcțiile kernel sunt folosite atunci când datele nu sunt liniar separabile. Ele funcționează prin faptul că datele sunt scufundate într-un

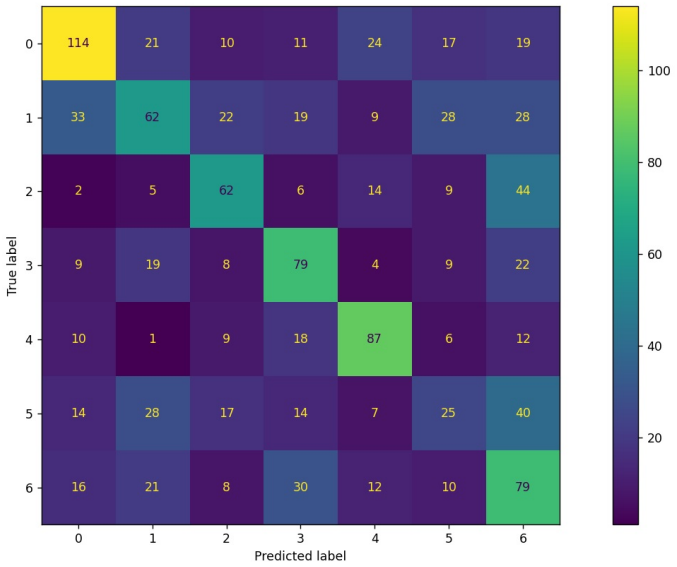
spațiu cu mai multe dimensiuni, iar apoi relațiile liniare sunt căutate în acest spațiu. Formulele pentru tipurile de kernel "rbf" și "linear" pot fi găsite în laboratorul 4

Mai jos voi atașa câteva dintre acuratețile obținute, precum și parametrii cu care le-am obținut:

- Pentru C = 1, kernel = "linear" și normalizare de tip "l2", acuratețea obținută a fost 0.43307757885763
- Pentru C = 5.5, kernel = "rbf" și normalizare "standard", acuratețea obținută a fost 0.56692242114237
- Pentru C = 5, kernel = "rbf" și normalizare "standard", acuratețea obținută a fost 0.5686274509803921
- Pentru C = 100, kernel = "rbf" și normalizare "l1", acuratețea obținută a fost 0.5166240409207161. Dacă am schimba doar kernel-ul cu unul "linear", acuratețea obținută ar fi de 0.39215686274509803

2.3 Matricea de confuzie

Aici este reprezentată matricea de confuzie generată pentru datele de validare:



3 A doua abordare - CNN pre-antrenat

3.1 Mod de implementare

După cum am spus, am încercat să clasific imaginile și folosind o rețea neuronală convoluțională pre-antrenată. Am avut mai multe încercări de modele, printre care se numără EfficientNetV2M, EfficientNetB0, ResNet50, MobileNet, VGG, însă scorul cel mai bun l-am obținut cu primul dintre ele. Totuși, aceste modele sunt pre-antrenate pe imagini mai mari față de cele primite în competiție - care au dimensiunea de 16x16, motiv pentru care acuratețea este destul de mică.

În ceea ce privește layerele, le-am luat pe cele până la block3b_add (am încercat să tai și după alt layer, însă nu am obținut o îmbunătățire vizibilă). Am adăugat și layere convoluționale, fiecare având activare ReLU(Rectified Linear Unit). Peste acestea, am adăugat și un layer dens, cu 128 de unități și activare ReLU, un Dropout pentru a minimiza overfittingul, precum și un layer final, dens, cu 7 unități și activare Softmax. Am încercat să adaug și BatchNormalization, dar fără să văd o îmbunătățire vizibilă. Modelul rezultat va avea următoarea structură :

```
[16]: model_transfer_learning.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 16, 16, 3)]       0

 model (Functional)          (None, None, None, 80)    771976

 conv2d (Conv2D)             (None, 2, 2, 32)          10272

 conv2d_1 (Conv2D)           (None, 2, 2, 64)          8256

 conv2d_2 (Conv2D)           (None, 2, 2, 128)         32896

 dense (Dense)               (None, 2, 2, 128)         16512

 dropout (Dropout)           (None, 2, 2, 128)         0

 global_average_pooling2d (G  (None, 128)              0
 lobalAveragePooling2D)

 dense_1 (Dense)             (None, 7)                 903

=================================================================
Total params: 840,815
Trainable params: 837,071
Non-trainable params: 3,744
_________________________________________________________________
```

```
frozen_model = tf.keras.Model(inputs = model_transfer.input,
                              outputs = pretrained_output.
                              output)
inputs = tf.keras.layers.Input(shape=(16,16,3))
x = frozen_model(inputs)
x = tf.keras.layers.Conv2D(32, (2, 2), padding='same',
                              activation='relu',
                              kernel_initializer=tf.keras.
                              initializers.he_normal())(x)
x = tf.keras.layers.Conv2D(64, (2, 2), padding='same',
                              activation='relu',
                              kernel_initializer=tf.keras.
                              initializers.he_normal())(x)
x = tf.keras.layers.Conv2D(128, (2, 2), padding='same',
                              activation='relu',
                              kernel_initializer=tf.keras.
                              initializers.he_normal())(x)
x = Dense(128, activation="relu")(x)
x = tf.keras.layers.Dropout(0.4)(x)
gap = tf.keras.layers.GlobalAveragePooling2D()(x)
outputs = Dense(7, activation="softmax")(gap)
```

Ca optimizator am folosit algoritmul Adam. Mai departe, am definit o clasa pentru gestionarea datelor de antrenare si validare, utilă pentru a face lazy loading, ceea ce îmi permite să folosesc batch size-uri mai mari pentru antrenare și modele mai adânci(deoarece îmi permite să am mai mult loc în memorie pentru tensori mai mari, fără a ocupa memoria cu imaginile propriu-zise). La fel ca la SVM, creez 2 vectori, unul pentru imaginile de antrenare (iduri_antrenare) și unul pentru label-urile acestora(categorii_antrenare), pe care îi voi folosi la funcția de fit.

3.2 Mici îmbunătățiri

Am încercat să folosesc augumentare de date pentru a crește acuratețea, însă nu am observat neapărat schimbări în bine. Am încercat să dau zoom, resize, flip, rotire și să schimb parametrii numerici, însă varianta finală este aceasta:

```
datagen = ImageDataGenerator(
    rotation_range=10,
    horizontal_flip=True)
```

Am folosit și un ModelCheckpoint, pentru a reține modelul în cel mai bun moment al său, ceea ce m-a ajutat puțin la creșterea scorului.

Apoi, am antrenat modelul și am generat predicțiile.

3.3 Testare pe epoci

Modelul încărcat ultima dată pe Kaggle, cel care mi-a dat scorul de 0.64914, mi-a dat următoarele valori pentru 50 de epoci:

```
Epoch 1/50
13/13 [==============================] - 10s 576ms/step - loss: 2.0485 - accuracy: 0.2276 - val_loss: 1.6193 - val_accuracy: 0.4523
Epoch 2/50
13/13 [==============================] - 7s 548ms/step - loss: 1.5608 - accuracy: 0.3999 - val_loss: 1.4030 - val_accuracy: 0.4870
Epoch 3/50
13/13 [==============================] - 7s 538ms/step - loss: 1.3707 - accuracy: 0.4796 - val_loss: 1.2744 - val_accuracy: 0.5330
Epoch 4/50
13/13 [==============================] - 7s 536ms/step - loss: 1.2547 - accuracy: 0.5255 - val_loss: 1.1888 - val_accuracy: 0.5660
Epoch 5/50
13/13 [==============================] - 7s 539ms/step - loss: 1.1553 - accuracy: 0.5732 - val_loss: 1.1169 - val_accuracy: 0.5955
Epoch 6/50
13/13 [==============================] - 7s 538ms/step - loss: 1.0741 - accuracy: 0.6050 - val_loss: 1.0590 - val_accuracy: 0.6172
Epoch 7/50
13/13 [==============================] - 7s 539ms/step - loss: 1.0048 - accuracy: 0.6480 - val_loss: 1.0161 - val_accuracy: 0.6215
Epoch 8/50
13/13 [==============================] - 7s 538ms/step - loss: 0.9324 - accuracy: 0.6601 - val_loss: 1.0606 - val_accuracy: 0.6120
Epoch 9/50
13/13 [==============================] - 7s 539ms/step - loss: 0.8852 - accuracy: 0.6890 - val_loss: 1.0625 - val_accuracy: 0.6285
Epoch 10/50
13/13 [==============================] - 7s 540ms/step - loss: 0.8287 - accuracy: 0.7045 - val_loss: 0.9837 - val_accuracy: 0.6623
Epoch 11/50
13/13 [==============================] - 7s 525ms/step - loss: 0.7927 - accuracy: 0.7247 - val_loss: 1.0283 - val_accuracy: 0.6345
Epoch 12/50
13/13 [==============================] - 7s 533ms/step - loss: 0.7196 - accuracy: 0.7460 - val_loss: 1.1308 - val_accuracy: 0.6354
Epoch 13/50
13/13 [==============================] - 7s 537ms/step - loss: 0.6813 - accuracy: 0.7654 - val_loss: 1.1012 - val_accuracy: 0.6285
Epoch 14/50
13/13 [==============================] - 7s 541ms/step - loss: 0.6316 - accuracy: 0.7834 - val_loss: 1.1661 - val_accuracy: 0.6033
Epoch 15/50
13/13 [==============================] - 7s 526ms/step - loss: 0.5818 - accuracy: 0.7972 - val_loss: 1.1807 - val_accuracy: 0.6276
Epoch 16/50
13/13 [==============================] - 7s 532ms/step - loss: 0.5036 - accuracy: 0.8231 - val_loss: 1.2616 - val_accuracy: 0.6198
Epoch 17/50
13/13 [==============================] - 7s 553ms/step - loss: 0.4737 - accuracy: 0.8404 - val_loss: 1.3424 - val_accuracy: 0.6094
Epoch 18/50
13/13 [==============================] - 7s 528ms/step - loss: 0.4103 - accuracy: 0.8615 - val_loss: 1.5641 - val_accuracy: 0.6068
Epoch 19/50
13/13 [==============================] - 7s 530ms/step - loss: 0.3764 - accuracy: 0.8766 - val_loss: 1.5340 - val_accuracy: 0.6146
Epoch 20/50
13/13 [==============================] - 7s 529ms/step - loss: 0.3504 - accuracy: 0.8876 - val_loss: 1.5958 - val_accuracy: 0.5920
Epoch 21/50
13/13 [==============================] - 7s 529ms/step - loss: 0.3049 - accuracy: 0.8994 - val_loss: 1.6143 - val_accuracy: 0.6285
```

```
Epoch 18/50
13/13 [==============================] - 7s 528ms/step - loss: 0.4103 - accuracy: 0.8615 - val_loss: 1.5641 - val_accuracy: 0.6068
Epoch 19/50
13/13 [==============================] - 7s 530ms/step - loss: 0.3764 - accuracy: 0.8766 - val_loss: 1.5340 - val_accuracy: 0.6146
Epoch 20/50
13/13 [==============================] - 7s 529ms/step - loss: 0.3504 - accuracy: 0.8876 - val_loss: 1.5958 - val_accuracy: 0.5920
Epoch 21/50
13/13 [==============================] - 7s 529ms/step - loss: 0.3049 - accuracy: 0.8994 - val_loss: 1.6143 - val_accuracy: 0.6285
Epoch 22/50
13/13 [==============================] - 7s 527ms/step - loss: 0.2670 - accuracy: 0.9124 - val_loss: 1.6838 - val_accuracy: 0.6189
Epoch 23/50
13/13 [==============================] - 7s 540ms/step - loss: 0.2534 - accuracy: 0.9194 - val_loss: 1.6804 - val_accuracy: 0.6224
Epoch 24/50
13/13 [==============================] - 7s 531ms/step - loss: 0.2207 - accuracy: 0.9302 - val_loss: 1.7496 - val_accuracy: 0.6155
Epoch 25/50
13/13 [==============================] - 7s 532ms/step - loss: 0.2201 - accuracy: 0.9321 - val_loss: 1.8971 - val_accuracy: 0.6241
Epoch 26/50
13/13 [==============================] - 7s 533ms/step - loss: 0.1887 - accuracy: 0.9414 - val_loss: 2.0041 - val_accuracy: 0.6181
Epoch 27/50
13/13 [==============================] - 7s 533ms/step - loss: 0.1650 - accuracy: 0.9492 - val_loss: 2.0561 - val_accuracy: 0.6137
Epoch 28/50
13/13 [==============================] - 7s 532ms/step - loss: 0.1540 - accuracy: 0.9541 - val_loss: 2.2014 - val_accuracy: 0.6259
Epoch 29/50
13/13 [==============================] - 7s 569ms/step - loss: 0.1684 - accuracy: 0.9489 - val_loss: 2.1525 - val_accuracy: 0.6155
Epoch 30/50
13/13 [==============================] - 7s 555ms/step - loss: 0.1475 - accuracy: 0.9540 - val_loss: 1.9704 - val_accuracy: 0.6111
Epoch 31/50
13/13 [==============================] - 7s 555ms/step - loss: 0.1318 - accuracy: 0.9590 - val_loss: 2.3025 - val_accuracy: 0.6068
Epoch 32/50
13/13 [==============================] - 7s 565ms/step - loss: 0.1379 - accuracy: 0.9569 - val_loss: 2.1361 - val_accuracy: 0.6233
Epoch 33/50
13/13 [==============================] - 7s 558ms/step - loss: 0.1111 - accuracy: 0.9653 - val_loss: 2.2287 - val_accuracy: 0.6155
Epoch 34/50
13/13 [==============================] - 8s 576ms/step - loss: 0.1215 - accuracy: 0.9617 - val_loss: 2.2476 - val_accuracy: 0.6181
Epoch 35/50
13/13 [==============================] - 9s 666ms/step - loss: 0.0956 - accuracy: 0.9690 - val_loss: 2.3484 - val_accuracy: 0.6259
Epoch 36/50
13/13 [==============================] - 8s 629ms/step - loss: 0.0838 - accuracy: 0.9758 - val_loss: 2.4276 - val_accuracy: 0.6302
Epoch 37/50
13/13 [==============================] - 8s 575ms/step - loss: 0.0865 - accuracy: 0.9726 - val_loss: 2.6490 - val_accuracy: 0.6128
Epoch 38/50
13/13 [==============================] - 10s 737ms/step - loss: 0.0915 - accuracy: 0.9737 - val_loss: 2.6530 - val_accuracy: 0.6076
```

Schimbând dropout-ul din 0.4 în 0.8, am obținut următoarele rezultate pentru 50 de epoci:

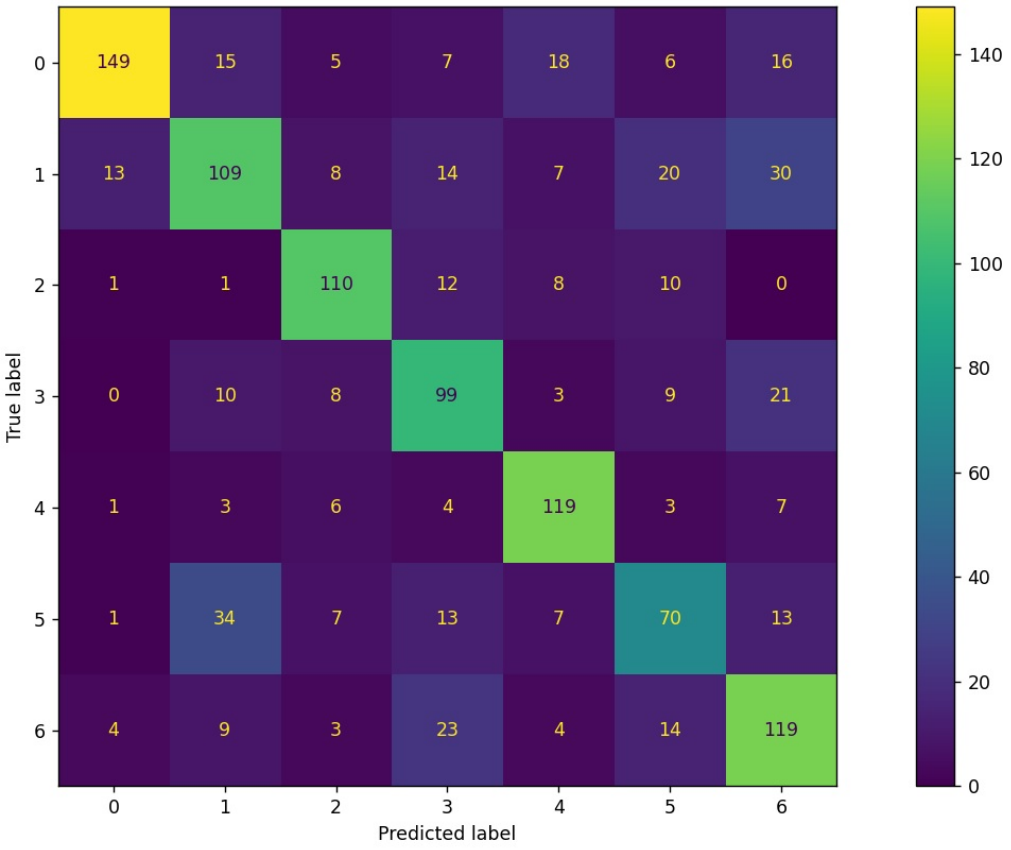
```
Epoch 1/50
16/16 [==============================] - 26s 1s/step - loss: 2.1139 - accuracy: 0.2451 - val_loss: 1.5501 - val_accuracy: 0.4462
Epoch 2/50
16/16 [==============================] - 20s 1s/step - loss: 1.5213 - accuracy: 0.4130 - val_loss: 1.3242 - val_accuracy: 0.5243
Epoch 3/50
16/16 [==============================] - 20s 1s/step - loss: 1.3357 - accuracy: 0.4942 - val_loss: 1.1854 - val_accuracy: 0.5521
Epoch 4/50
16/16 [==============================] - 20s 1s/step - loss: 1.2123 - accuracy: 0.5501 - val_loss: 1.0993 - val_accuracy: 0.6024
Epoch 5/50
16/16 [==============================] - 20s 1s/step - loss: 1.0760 - accuracy: 0.6066 - val_loss: 1.0737 - val_accuracy: 0.6016
Epoch 6/50
16/16 [==============================] - 20s 1s/step - loss: 1.0178 - accuracy: 0.6315 - val_loss: 1.0509 - val_accuracy: 0.6042
Epoch 7/50
16/16 [==============================] - 20s 1s/step - loss: 0.9505 - accuracy: 0.6661 - val_loss: 1.0184 - val_accuracy: 0.6319
Epoch 8/50
16/16 [==============================] - 21s 1s/step - loss: 0.8851 - accuracy: 0.6906 - val_loss: 0.9770 - val_accuracy: 0.6510
Epoch 9/50
16/16 [==============================] - 21s 1s/step - loss: 0.8311 - accuracy: 0.7135 - val_loss: 0.9683 - val_accuracy: 0.6589
Epoch 10/50
16/16 [==============================] - 20s 1s/step - loss: 0.7757 - accuracy: 0.7293 - val_loss: 1.0079 - val_accuracy: 0.6580
Epoch 11/50
16/16 [==============================] - 19s 1s/step - loss: 0.7262 - accuracy: 0.7519 - val_loss: 1.0377 - val_accuracy: 0.6450
Epoch 12/50
16/16 [==============================] - 19s 1s/step - loss: 0.6747 - accuracy: 0.7717 - val_loss: 1.0751 - val_accuracy: 0.6424
Epoch 13/50
16/16 [==============================] - 20s 1s/step - loss: 0.6209 - accuracy: 0.7883 - val_loss: 1.1320 - val_accuracy: 0.6224
Epoch 14/50
16/16 [==============================] - 20s 1s/step - loss: 0.5588 - accuracy: 0.8134 - val_loss: 1.1691 - val_accuracy: 0.6398
Epoch 15/50
16/16 [==============================] - 20s 1s/step - loss: 0.5062 - accuracy: 0.8321 - val_loss: 1.2155 - val_accuracy: 0.6372
Epoch 16/50
16/16 [==============================] - 20s 1s/step - loss: 0.4375 - accuracy: 0.8497 - val_loss: 1.2398 - val_accuracy: 0.6380
Epoch 17/50
16/16 [==============================] - 20s 1s/step - loss: 0.4204 - accuracy: 0.8654 - val_loss: 1.3342 - val_accuracy: 0.6337
```

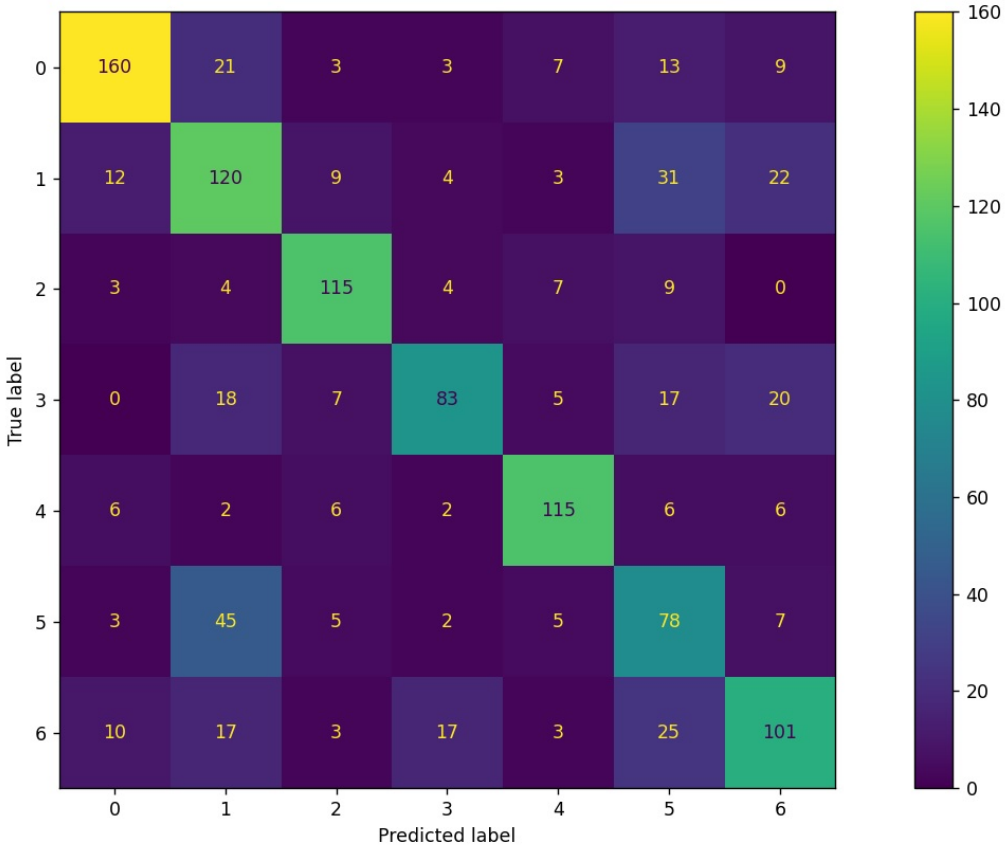
```
Epoch 17/50
16/16 [==============================] - 20s 1s/step - loss: 0.4204 - accuracy: 0.8654 - val_loss: 1.3342 - val_accuracy: 0.6337
Epoch 18/50
16/16 [==============================] - 19s 1s/step - loss: 0.3893 - accuracy: 0.8740 - val_loss: 1.3703 - val_accuracy: 0.6276
Epoch 19/50
16/16 [==============================] - 19s 1s/step - loss: 0.3224 - accuracy: 0.8976 - val_loss: 1.4442 - val_accuracy: 0.6484
Epoch 20/50
16/16 [==============================] - 20s 1s/step - loss: 0.2943 - accuracy: 0.9028 - val_loss: 1.5279 - val_accuracy: 0.6328
Epoch 21/50
16/16 [==============================] - 20s 1s/step - loss: 0.2535 - accuracy: 0.9194 - val_loss: 1.6715 - val_accuracy: 0.6311
Epoch 22/50
16/16 [==============================] - 22s 1s/step - loss: 0.2471 - accuracy: 0.9205 - val_loss: 1.6230 - val_accuracy: 0.6146
Epoch 23/50
16/16 [==============================] - 20s 1s/step - loss: 0.2064 - accuracy: 0.9326 - val_loss: 1.6801 - val_accuracy: 0.6467
Epoch 24/50
16/16 [==============================] - 19s 1s/step - loss: 0.1820 - accuracy: 0.9413 - val_loss: 1.8902 - val_accuracy: 0.6276
Epoch 25/50
16/16 [==============================] - 20s 1s/step - loss: 0.1941 - accuracy: 0.9391 - val_loss: 1.7430 - val_accuracy: 0.6363
Epoch 26/50
16/16 [==============================] - 16s 987ms/step - loss: 0.1843 - accuracy: 0.9414 - val_loss: 1.7687 - val_accuracy: 0.6398
Epoch 27/50
16/16 [==============================] - 12s 731ms/step - loss: 0.1635 - accuracy: 0.9492 - val_loss: 1.7849 - val_accuracy: 0.6285
Epoch 28/50
16/16 [==============================] - 10s 633ms/step - loss: 0.1387 - accuracy: 0.9553 - val_loss: 1.9354 - val_accuracy: 0.6467
Epoch 29/50
16/16 [==============================] - 10s 637ms/step - loss: 0.1226 - accuracy: 0.9629 - val_loss: 2.1090 - val_accuracy: 0.6493
Epoch 30/50
16/16 [==============================] - 10s 634ms/step - loss: 0.1286 - accuracy: 0.9590 - val_loss: 1.9714 - val_accuracy: 0.6302
Epoch 31/50
16/16 [==============================] - 10s 641ms/step - loss: 0.1144 - accuracy: 0.9634 - val_loss: 2.0692 - val_accuracy: 0.6389
Epoch 32/50
16/16 [==============================] - 10s 633ms/step - loss: 0.1040 - accuracy: 0.9685 - val_loss: 2.1448 - val_accuracy: 0.6250
Epoch 33/50
16/16 [==============================] - 11s 667ms/step - loss: 0.0943 - accuracy: 0.9696 - val_loss: 2.1801 - val_accuracy: 0.6354
```

```
Epoch 34/50
16/16 [==============================] - 11s 705ms/step - loss: 0.0885 - accuracy: 0.9711 - val_loss: 2.1826 - val_accuracy: 0.6337
Epoch 35/50
16/16 [==============================] - 11s 670ms/step - loss: 0.0927 - accuracy: 0.9719 - val_loss: 2.1329 - val_accuracy: 0.6259
Epoch 36/50
16/16 [==============================] - 11s 685ms/step - loss: 0.0779 - accuracy: 0.9760 - val_loss: 2.1963 - val_accuracy: 0.6311
Epoch 37/50
16/16 [==============================] - 11s 700ms/step - loss: 0.0822 - accuracy: 0.9755 - val_loss: 2.2888 - val_accuracy: 0.6493
Epoch 38/50
16/16 [==============================] - 11s 692ms/step - loss: 0.0859 - accuracy: 0.9741 - val_loss: 2.3310 - val_accuracy: 0.6293
Epoch 39/50
16/16 [==============================] - 11s 714ms/step - loss: 0.0730 - accuracy: 0.9779 - val_loss: 2.2466 - val_accuracy: 0.6372
Epoch 40/50
16/16 [==============================] - 12s 740ms/step - loss: 0.0605 - accuracy: 0.9806 - val_loss: 2.3848 - val_accuracy: 0.6250
Epoch 41/50
16/16 [==============================] - 11s 699ms/step - loss: 0.0526 - accuracy: 0.9840 - val_loss: 2.5396 - val_accuracy: 0.6207
Epoch 42/50
16/16 [==============================] - 11s 707ms/step - loss: 0.0653 - accuracy: 0.9791 - val_loss: 2.3766 - val_accuracy: 0.6337
Epoch 43/50
16/16 [==============================] - 11s 703ms/step - loss: 0.0662 - accuracy: 0.9791 - val_loss: 2.4356 - val_accuracy: 0.6189
Epoch 44/50
16/16 [==============================] - 11s 682ms/step - loss: 0.0550 - accuracy: 0.9822 - val_loss: 2.4551 - val_accuracy: 0.6224
Epoch 45/50
16/16 [==============================] - 10s 636ms/step - loss: 0.0574 - accuracy: 0.9824 - val_loss: 2.5861 - val_accuracy: 0.6233
Epoch 46/50
16/16 [==============================] - 10s 634ms/step - loss: 0.0667 - accuracy: 0.9809 - val_loss: 2.5224 - val_accuracy: 0.6120
Epoch 47/50
16/16 [==============================] - 10s 632ms/step - loss: 0.0532 - accuracy: 0.9837 - val_loss: 2.4633 - val_accuracy: 0.6302
Epoch 48/50
16/16 [==============================] - 10s 639ms/step - loss: 0.0485 - accuracy: 0.9850 - val_loss: 2.5137 - val_accuracy: 0.6250
Epoch 49/50
16/16 [==============================] - 10s 638ms/step - loss: 0.0491 - accuracy: 0.9834 - val_loss: 2.6280 - val_accuracy: 0.6354
Epoch 50/50
16/16 [==============================] - 10s 650ms/step - loss: 0.0717 - accuracy: 0.9812 - val_loss: 2.5107 - val_accuracy: 0.6372
```

3.4 Matrice de confuzie

Aici este reprezentată matricea de confuzie generată pentru datele de validare, cu 50 de epoci și dropout-ul 0.4, respectiv 0.8:


0
1
2
3
4
5
6
True label
149 15 5 7 18 6 16
13 109 8 14 7 20 30
1 1 110 12 8 10 0
0 10 8 99 3 9 21
1 3 6 4 119 3 7
1 34 7 13 7 70 13
4 9 3 23 4 14 119
0 1 2 3 4 5 6
Predicted label
140
120
100
80
60
40
20
0



3.5 Altă variantă de alegere a layerelor

Cu modelul astfel definit am obținut un scor pe Kaggle de 0.62784.

```
frozen_model = tf.keras.Model(inputs=model_transfer.input,
                              outputs=pretrained_output.
                              output)
inputs = tf.keras.layers.Input(shape=(16,16,3))
x = frozen_model(inputs)
x = tf.keras.layers.Conv2D(64, (2, 2), padding='same',
                              activation='relu',
                              kernel_initializer=tf.keras.
                              initializers.he_normal())(x)
x = tf.keras.layers.Conv2D(32, (2, 2), padding='same',
                              activation='relu',
```

```
                                    kernel_initializer=tf.keras.
                                    initializers.he_normal())(x)
x = tf.keras.layers.Conv2D(7, (2, 2), padding='same',
                                    activation='relu',
                                    kernel_initializer = tf.keras.
                                    initializers.he_normal())(x)
gap = tf.keras.layers.GlobalAveragePooling2D()(x)
outputs = Dense(7, activation="softmax")(gap)
```

4 A treia abordare - CNN nepreantrenat

O ultimă abordare pe care am încercat-o a fost să îmi creez și antrenez propriul model, însă nu am obținut un scor mai mare decât în cazul modelului pre-antrenat. Am construit un model bazat pe arhitectura modelului ResNet, ce are 3 blocuri cu skip connections din 2 în 2 layere, iar layerele de la final sunt de tip dense. De această dată, am folosit și activarea sigmoid și regularizare.

Astfel, structura modelului este următoarea:

```
model.summary()
```

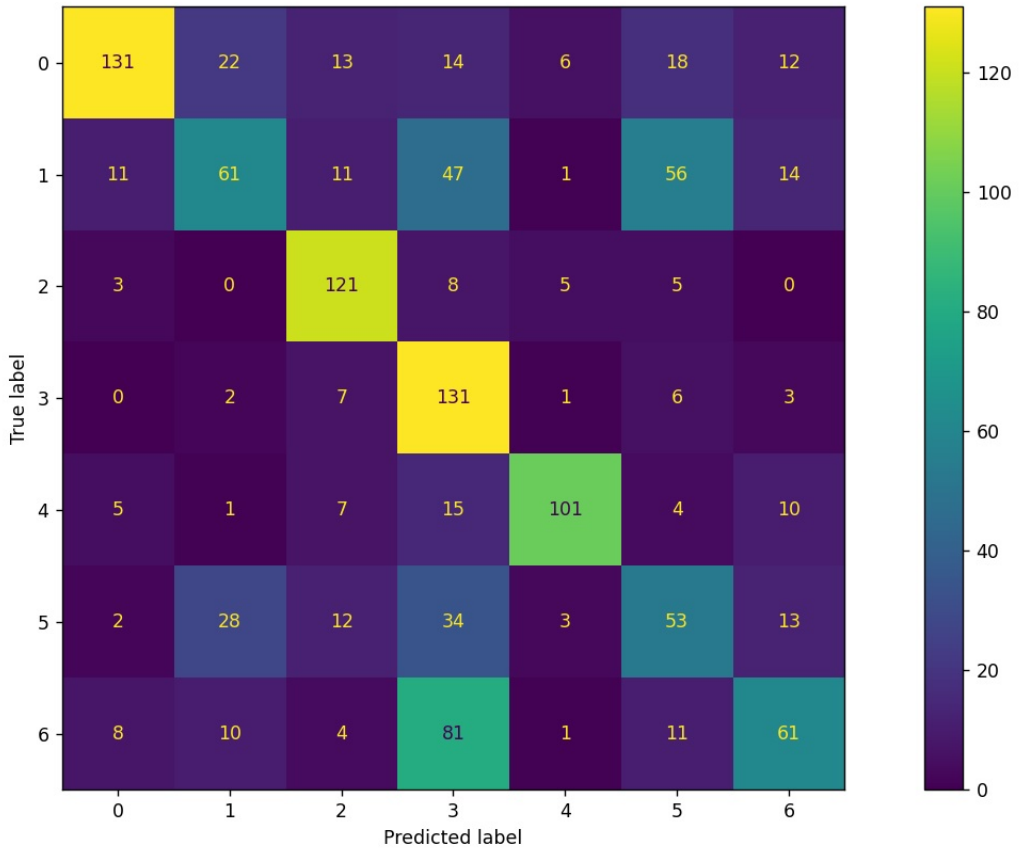
```
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 16, 16, 3)]	0	[]
conv2d (Conv2D)	(None, 16, 16, 64)	1792	['input_1[0][0]']
dropout (Dropout)	(None, 16, 16, 64)	0	['conv2d[0][0]']
conv2d_1 (Conv2D)	(None, 16, 16, 64)	36928	['dropout[0][0]']
dropout_1 (Dropout)	(None, 16, 16, 64)	0	['conv2d_1[0][0]']
batch_normalization (BatchNormalization)	(None, 16, 16, 64)	256	['dropout_1[0][0]']
conv2d_2 (Conv2D)	(None, 16, 16, 64)	36928	['batch_normalization[0][0]']
dropout_2 (Dropout)	(None, 16, 16, 64)	0	['conv2d_2[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 64)	256	['dropout_2[0][0]']
tf.__operators__.add (TFOpLambda)	(None, 16, 16, 64)	0	['batch_normalization_1[0][0]', 'dropout[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256	['tf.__operators__.add[0][0]']
max_pooling2d (MaxPooling2D)	(None, 8, 8, 64)	0	['batch_normalization_2[0][0]']
conv2d_3 (Conv2D)	(None, 8, 8, 128)	73856	['max_pooling2d[0][0]']
dropout_3 (Dropout)	(None, 8, 8, 128)	0	['conv2d_3[0][0]']
conv2d_4 (Conv2D)	(None, 8, 8, 128)	147584	['dropout_3[0][0]']
dropout_4 (Dropout)	(None, 8, 8, 128)	0	['conv2d_4[0][0]']

```
dropout_4 (Dropout)            (None, 8, 8, 128)    0           ['conv2d_4[0][0]']

batch_normalization_3 (BatchNo  (None, 8, 8, 128)   512         ['dropout_4[0][0]']
rmalization)

conv2d_5 (Conv2D)              (None, 8, 8, 128)    147584      ['batch_normalization_3[0][0]']

dropout_5 (Dropout)            (None, 8, 8, 128)    0           ['conv2d_5[0][0]']

batch_normalization_4 (BatchNo  (None, 8, 8, 128)   512         ['dropout_5[0][0]']
rmalization)

tf.__operators__.add_1 (TFOpLa  (None, 8, 8, 128)   0           ['batch_normalization_4[0][0]',
mbda)                                                            'dropout_3[0][0]']

batch_normalization_5 (BatchNo  (None, 8, 8, 128)   512         ['tf.__operators__.add_1[0][0]']
rmalization)

max_pooling2d_1 (MaxPooling2D)  (None, 4, 4, 128)   0           ['batch_normalization_5[0][0]']

conv2d_6 (Conv2D)              (None, 4, 4, 256)    295168      ['max_pooling2d_1[0][0]']

dropout_6 (Dropout)            (None, 4, 4, 256)    0           ['conv2d_6[0][0]']

conv2d_7 (Conv2D)              (None, 4, 4, 256)    590080      ['dropout_6[0][0]']

dropout_7 (Dropout)            (None, 4, 4, 256)    0           ['conv2d_7[0][0]']

batch_normalization_6 (BatchNo  (None, 4, 4, 256)   1024        ['dropout_7[0][0]']
rmalization)

conv2d_8 (Conv2D)              (None, 4, 4, 256)    590080      ['batch_normalization_6[0][0]']

dropout_8 (Dropout)            (None, 4, 4, 256)    0           ['conv2d_8[0][0]']

batch_normalization_7 (BatchNo  (None, 4, 4, 256)   1024        ['dropout_8[0][0]']
rmalization)

tf.__operators__.add_2 (TFOpLa  (None, 4, 4, 256)   0           ['batch_normalization_7[0][0]',
mbda)                                                            'dropout_6[0][0]']
```

```
tf.__operators__.add_2 (TFOpLa  (None, 4, 4, 256)   0           ['batch_normalization_7[0][0]',
mbda)                                                            'dropout_6[0][0]']

flatten (Flatten)              (None, 4096)         0           ['tf.__operators__.add_2[0][0]']

dense (Dense)                  (None, 512)          2097664     ['flatten[0][0]']

dropout_9 (Dropout)            (None, 512)          0           ['dense[0][0]']

dense_1 (Dense)                (None, 128)          65664       ['dropout_9[0][0]']

dropout_10 (Dropout)           (None, 128)          0           ['dense_1[0][0]']

dense_2 (Dense)                (None, 32)           4128        ['dropout_10[0][0]']

dropout_11 (Dropout)           (None, 32)           0           ['dense_2[0][0]']

dense_3 (Dense)                (None, 7)            231         ['dropout_11[0][0]']

==================================================================================================
Total params: 4,092,039
Trainable params: 4,089,863
Non-trainable params: 2,176
```

Iar matricea de confuzie este următoarea:



Rezultatele testării pentru 50 de epoci au dat o acuratețe destul de mică și un loss destul de mare, așa cum ilustrează următoarele print-screen-uri:

```
Epoch 1/50
31/31 [==============================] - 61s 2s/step - loss: 4.2651 - accuracy: 0.2383 - val_loss: 2.7233 - val_accuracy: 0.3030 - lr: 0.0010
Epoch 2/50
31/31 [==============================] - 60s 2s/step - loss: 2.2739 - accuracy: 0.3775 - val_loss: 2.2036 - val_accuracy: 0.2960 - lr: 9.8000e-04
Epoch 3/50
31/31 [==============================] - 57s 2s/step - loss: 1.9644 - accuracy: 0.4134 - val_loss: 1.8686 - val_accuracy: 0.4549 - lr: 9.6040e-04
Epoch 4/50
31/31 [==============================] - 61s 2s/step - loss: 1.8390 - accuracy: 0.4491 - val_loss: 1.9162 - val_accuracy: 0.4028 - lr: 9.4119e-04
Epoch 5/50
31/31 [==============================] - 70s 2s/step - loss: 1.7538 - accuracy: 0.4679 - val_loss: 1.7562 - val_accuracy: 0.4575 - lr: 9.2237e-04
Epoch 6/50
31/31 [==============================] - 65s 2s/step - loss: 1.6907 - accuracy: 0.4877 - val_loss: 1.8674 - val_accuracy: 0.4062 - lr: 9.0392e-04
Epoch 7/50
31/31 [==============================] - 59s 2s/step - loss: 1.6538 - accuracy: 0.4927 - val_loss: 1.8695 - val_accuracy: 0.3967 - lr: 8.8584e-04
Epoch 8/50
31/31 [==============================] - 58s 2s/step - loss: 1.6180 - accuracy: 0.5106 - val_loss: 1.7592 - val_accuracy: 0.4340 - lr: 8.6813e-04
Epoch 9/50
31/31 [==============================] - 59s 2s/step - loss: 1.6009 - accuracy: 0.5159 - val_loss: 1.6157 - val_accuracy: 0.5234 - lr: 8.5076e-04
Epoch 10/50
31/31 [==============================] - 59s 2s/step - loss: 1.5736 - accuracy: 0.5239 - val_loss: 1.7109 - val_accuracy: 0.4931 - lr: 8.3375e-04
Epoch 11/50
31/31 [==============================] - 60s 2s/step - loss: 1.5573 - accuracy: 0.5326 - val_loss: 1.6916 - val_accuracy: 0.4714 - lr: 8.1707e-04
```

```
Epoch 11/50
31/31 [==============================] - 60s 2s/step - loss: 1.5573 - accuracy: 0.5326 - val_loss: 1.6916 - val_accuracy: 0.4714 - lr: 8.1707e-04
Epoch 12/50
31/31 [==============================] - 60s 2s/step - loss: 1.5210 - accuracy: 0.5372 - val_loss: 1.7006 - val_accuracy: 0.4696 - lr: 8.0073e-04
Epoch 13/50
31/31 [==============================] - 59s 2s/step - loss: 1.5157 - accuracy: 0.5473 - val_loss: 1.7827 - val_accuracy: 0.4288 - lr: 7.8472e-04
Epoch 14/50
31/31 [==============================] - 61s 2s/step - loss: 1.4897 - accuracy: 0.5651 - val_loss: 1.6957 - val_accuracy: 0.4740 - lr: 7.6902e-04
Epoch 15/50
31/31 [==============================] - 58s 2s/step - loss: 1.4982 - accuracy: 0.5635 - val_loss: 1.6306 - val_accuracy: 0.5035 - lr: 7.5364e-04
Epoch 16/50
31/31 [==============================] - 58s 2s/step - loss: 1.4837 - accuracy: 0.5727 - val_loss: 1.7021 - val_accuracy: 0.4635 - lr: 7.3857e-04
Epoch 17/50
31/31 [==============================] - 840s 28s/step - loss: 1.4473 - accuracy: 0.5893 - val_loss: 1.8558 - val_accuracy: 0.4497 - lr: 7.2380e-04
Epoch 18/50
31/31 [==============================] - 48s 2s/step - loss: 1.4572 - accuracy: 0.5877 - val_loss: 1.8772 - val_accuracy: 0.4097 - lr: 7.0932e-04
Epoch 19/50
31/31 [==============================] - 48s 2s/step - loss: 1.4255 - accuracy: 0.5989 - val_loss: 1.7813 - val_accuracy: 0.4410 - lr: 6.9514e-04
Epoch 20/50
31/31 [==============================] - 49s 2s/step - loss: 1.4073 - accuracy: 0.6071 - val_loss: 1.7619 - val_accuracy: 0.4557 - lr: 6.8123e-04
Epoch 21/50
31/31 [==============================] - 57s 2s/step - loss: 1.4124 - accuracy: 0.6057 - val_loss: 1.6303 - val_accuracy: 0.5226 - lr: 6.6761e-04
Epoch 22/50
31/31 [==============================] - 53s 2s/step - loss: 1.3713 - accuracy: 0.6256 - val_loss: 1.5688 - val_accuracy: 0.5408 - lr: 6.5426e-04
Epoch 23/50
31/31 [==============================] - 53s 2s/step - loss: 1.3538 - accuracy: 0.6362 - val_loss: 1.6182 - val_accuracy: 0.5156 - lr: 6.4117e-04
Epoch 24/50
31/31 [==============================] - 53s 2s/step - loss: 1.3428 - accuracy: 0.6381 - val_loss: 1.6157 - val_accuracy: 0.5182 - lr: 6.2835e-04
Epoch 25/50
31/31 [==============================] - 54s 2s/step - loss: 1.3374 - accuracy: 0.6442 - val_loss: 1.7679 - val_accuracy: 0.4870 - lr: 6.1578e-04
Epoch 26/50
31/31 [==============================] - 53s 2s/step - loss: 1.3312 - accuracy: 0.6404 - val_loss: 1.4852 - val_accuracy: 0.5799 - lr: 6.0346e-04
```

```
Epoch 25/50
31/31 [==============================] - 54s 2s/step - loss: 1.3374 - accuracy: 0.6442 - val_loss: 1.7679 - val_accuracy: 0.4870 - lr: 6.1578e-04
Epoch 26/50
31/31 [==============================] - 53s 2s/step - loss: 1.3312 - accuracy: 0.6404 - val_loss: 1.4852 - val_accuracy: 0.5799 - lr: 6.0346e-04
Epoch 27/50
31/31 [==============================] - 59s 2s/step - loss: 1.3083 - accuracy: 0.6561 - val_loss: 1.7572 - val_accuracy: 0.4818 - lr: 5.9140e-04
Epoch 28/50
31/31 [==============================] - 65s 2s/step - loss: 1.2966 - accuracy: 0.6579 - val_loss: 1.5966 - val_accuracy: 0.5391 - lr: 5.7957e-04
Epoch 29/50
31/31 [==============================] - 59s 2s/step - loss: 1.2971 - accuracy: 0.6602 - val_loss: 1.4905 - val_accuracy: 0.5851 - lr: 5.6798e-04
Epoch 30/50
31/31 [==============================] - 57s 2s/step - loss: 1.2884 - accuracy: 0.6607 - val_loss: 1.7168 - val_accuracy: 0.4983 - lr: 5.5662e-04
Epoch 31/50
31/31 [==============================] - 65s 2s/step - loss: 1.2962 - accuracy: 0.6610 - val_loss: 1.5469 - val_accuracy: 0.5868 - lr: 5.4548e-04
Epoch 32/50
31/31 [==============================] - 71s 2s/step - loss: 1.2555 - accuracy: 0.6740 - val_loss: 1.5541 - val_accuracy: 0.5790 - lr: 5.3457e-04
Epoch 33/50
31/31 [==============================] - 69s 2s/step - loss: 1.2559 - accuracy: 0.6755 - val_loss: 1.5929 - val_accuracy: 0.5469 - lr: 5.2388e-04
Epoch 34/50
31/31 [==============================] - 67s 2s/step - loss: 1.2191 - accuracy: 0.6928 - val_loss: 1.7355 - val_accuracy: 0.5078 - lr: 5.1341e-04
Epoch 35/50
31/31 [==============================] - 73s 2s/step - loss: 1.2459 - accuracy: 0.6775 - val_loss: 1.5675 - val_accuracy: 0.5694 - lr: 5.0314e-04
Epoch 36/50
31/31 [==============================] - 69s 2s/step - loss: 1.2281 - accuracy: 0.6907 - val_loss: 1.6429 - val_accuracy: 0.5521 - lr: 4.9307e-04
Epoch 37/50
31/31 [==============================] - 70s 2s/step - loss: 1.2001 - accuracy: 0.7025 - val_loss: 1.5616 - val_accuracy: 0.5686 - lr: 4.8321e-04
Epoch 38/50
31/31 [==============================] - 68s 2s/step - loss: 1.1885 - accuracy: 0.7007 - val_loss: 1.6531 - val_accuracy: 0.5321 - lr: 4.7355e-04
Epoch 39/50
31/31 [==============================] - 79s 3s/step - loss: 1.1921 - accuracy: 0.7070 - val_loss: 1.7449 - val_accuracy: 0.5104 - lr: 4.6408e-04
Epoch 40/50
31/31 [==============================] - 82s 3s/step - loss: 1.1685 - accuracy: 0.7152 - val_loss: 1.6340 - val_accuracy: 0.5486 - lr: 4.5480e-04
Epoch 41/50
31/31 [==============================] - 70s 2s/step - loss: 1.1657 - accuracy: 0.7130 - val_loss: 1.6897 - val_accuracy: 0.5469 - lr: 4.4570e-04
```

```
Epoch 40/50
31/31 [==============================] - 82s 3s/step - loss: 1.1685 - accuracy: 0.7152 - val_loss: 1.6340 - val_accuracy: 0.5486 - lr: 4.5480e-04
Epoch 41/50
31/31 [==============================] - 70s 2s/step - loss: 1.1657 - accuracy: 0.7130 - val_loss: 1.6897 - val_accuracy: 0.5469 - lr: 4.4570e-04
Epoch 42/50
31/31 [==============================] - 74s 2s/step - loss: 1.1805 - accuracy: 0.7106 - val_loss: 1.7581 - val_accuracy: 0.5130 - lr: 4.3679e-04
Epoch 43/50
31/31 [==============================] - 80s 3s/step - loss: 1.1510 - accuracy: 0.7174 - val_loss: 1.7949 - val_accuracy: 0.5148 - lr: 4.2805e-04
Epoch 44/50
31/31 [==============================] - 73s 2s/step - loss: 1.1397 - accuracy: 0.7200 - val_loss: 1.6236 - val_accuracy: 0.5477 - lr: 4.1949e-04
Epoch 45/50
31/31 [==============================] - 83s 3s/step - loss: 1.1304 - accuracy: 0.7224 - val_loss: 1.5148 - val_accuracy: 0.5799 - lr: 4.1110e-04
Epoch 46/50
31/31 [==============================] - 68s 2s/step - loss: 1.1266 - accuracy: 0.7269 - val_loss: 1.5453 - val_accuracy: 0.5833 - lr: 4.0288e-04
Epoch 47/50
31/31 [==============================] - 70s 2s/step - loss: 1.1075 - accuracy: 0.7329 - val_loss: 1.5327 - val_accuracy: 0.5911 - lr: 3.9482e-04
Epoch 48/50
31/31 [==============================] - 67s 2s/step - loss: 1.0976 - accuracy: 0.7405 - val_loss: 1.6446 - val_accuracy: 0.5608 - lr: 3.8692e-04
Epoch 49/50
31/31 [==============================] - 67s 2s/step - loss: 1.0986 - accuracy: 0.7326 - val_loss: 1.7087 - val_accuracy: 0.5382 - lr: 3.7919e-04
Epoch 50/50
31/31 [==============================] - 70s 2s/step - loss: 1.0858 - accuracy: 0.7408 - val_loss: 1.6067 - val_accuracy: 0.5582 - lr: 3.7160e-04
```

5 Concluzie

Așadar, se poate observa cu ușurință că varianta alegerii unui model preantrenat a fost cea mai bună, reușind să ajungă la o anumită epocă la un loss sub 1 și la o acuratețe mai mare de 0.66.