

Deep Hallucination Classification

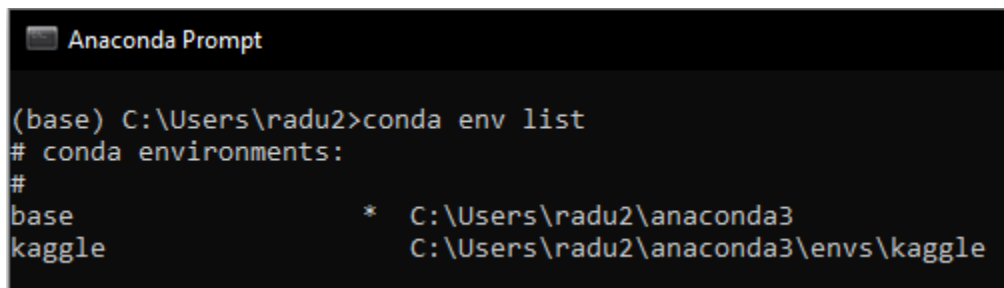
Gheorghe Radu-Mihai, grupa 251

I. Introducere și descrierea cerinței

A doua parte a laboratorului de Inteligență Artificială a constatat în competiția “Deep Hallucination Classification”, unde a trebuit să clasificăm un set de date sub formă de imagini pe categoriile din care fac parte, numerotate de la 0 la 95. Pentru setul de date de antrenare, am avut la dispoziție 12000 de imagini și label-urile acestora, pentru datele de validare 1000 de imagini, iar pentru setul de date de test, 5000 de imagini pentru care trebuia să facem predicții.

II. Configurarea mediului de lucru

Pentru a face ca procesul de antrenare să fie mai rapid și eficient, am folosit resursele GPU ale PC-ului, cu ajutorul programului Anaconda. Astfel, mi-am creat un *virtual environment* de Python care să ruleze pe GPU:



```
Anaconda Prompt

(base) C:\Users\radu2>conda env list
# conda environments:
#
base                * C:\Users\radu2\anaconda3
kaggle              C:\Users\radu2\anaconda3\envs\kaggle
```

III. Citirea și preluarea datelor de lucru

Atât datele de antrenament, cât și cele de validare, au fost sub forma unui fișier .csv, unde primeam pentru fiecare poză și label-ul asociat acesteia:

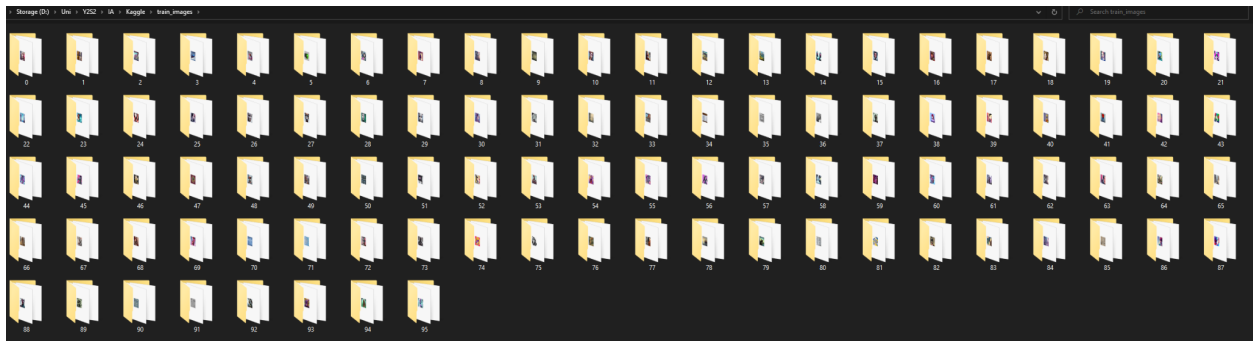
1	Image, Class
2	12953.png, 38
3	02978.png, 76
4	10718.png, 18
5	01916.png, 16
6	15254.png, 53
7	10871.png, 69
8	13745.png, 45
9	03218.png, 18
10	13367.png, 65
11	16226.png, 26
12	08099.png, 95
13	14800.png, 0
14	12381.png, 79
15	13986.png, 84
16	06972.png, 70
17	14769.png, 67

Pentru preluarea datelor, am folosit funcția *image_dataset_from_directory* din Tensorflow, organizând pozele pe batch-uri de câte 32:

```
##### TAKE DATA #####

imgHeight = 64
imgWidth = 64
trainDataset = tf.keras.preprocessing.image_dataset_from_directory(
    'train_images/',
    seed=123,
    image_size=(imgHeight, imgWidth),
    label_mode='categorical')
validationDataset = tf.keras.preprocessing.image_dataset_from_directory(
    'val_images/',
    seed=123,
    image_size=(imgHeight, imgWidth),
    label_mode='categorical')
```

Pentru a putea prelua datele în acest mod, a trebuit să reorganizez structura folderelor *train_images* și *val_images*, astfel ca fiecare poză să se regăsească în subfolderul clasei din care face parte:



IV. Abordări folosite

a. SVM (Support Vector Machine)

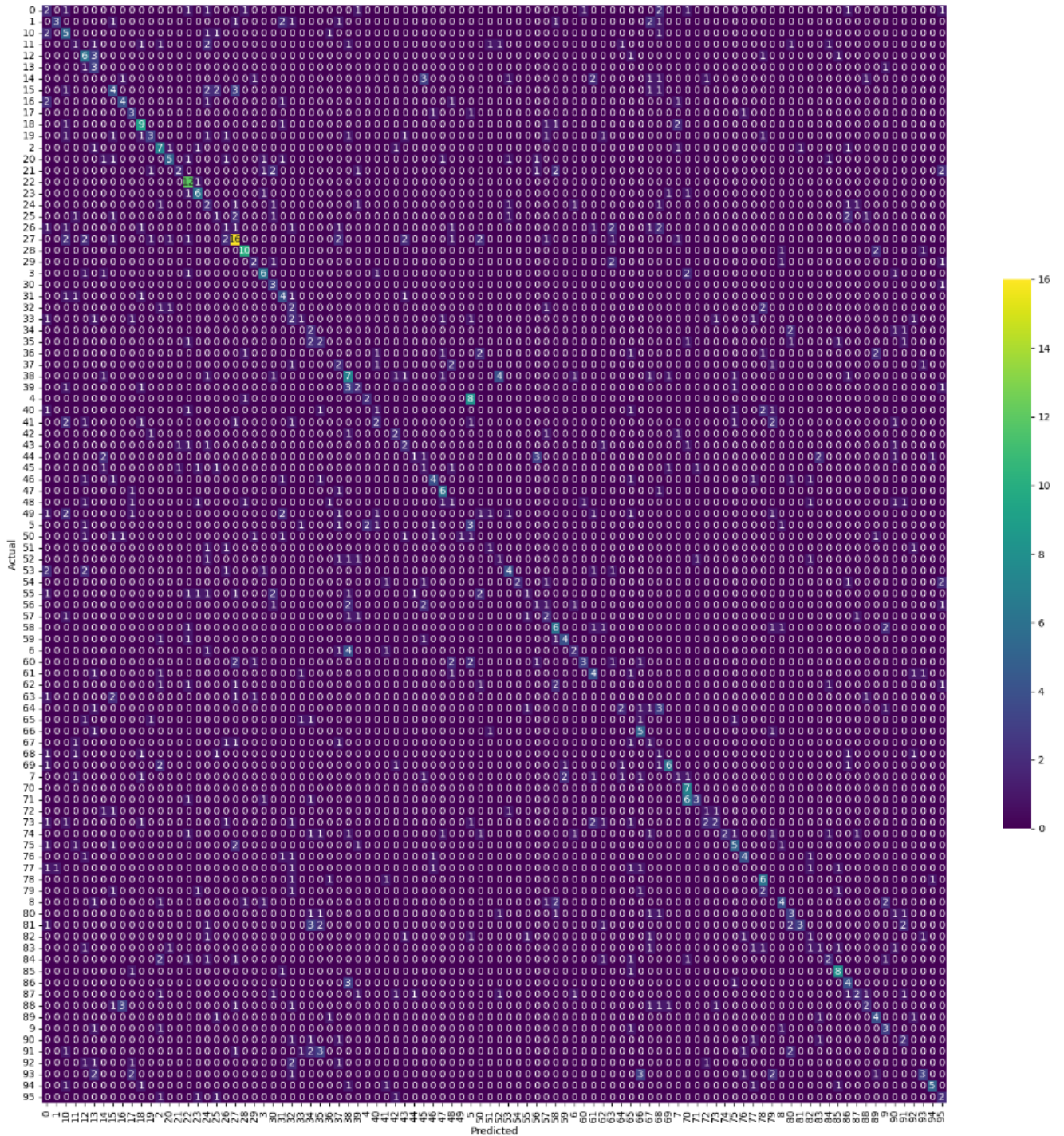
Am convertit imaginile de instruire într-o matrice *Numpy*, apoi le-am normalizat pentru a se potrivi formatului de input necesar pentru SVM. În loc să le păstrez forma tridimensională, am "întins" fiecare imagine într-un vector unidimensional. În mod similar au fost procesate și datele de validare. Fiecare imagine este remodelată de la o formă matriceală $(64, 64, 3)$ la un vector unidimensional cu $64 \cdot 64 \cdot 3 = 12288$ elemente.

Metoda implicită utilizată de *sklearn* pentru SVM-urile multiclase este *una-vs-rest (OvR)*. În această abordare, pentru fiecare dintre cele 96 de clase, se construiește un SVM separat care distinge acea clasă de toate celelalte. Prin urmare, în total, se vor construi 96 de SVM-uri. Pentru a face o predicție pentru o imagine nouă, toate cele 96 de SVM-uri vor fi folosite. Clasa predicată va fi aceea pentru care SVM-ul corespunzător dă cea mai mare "încredere".

În cele din urmă, am antrenat modelul pe aceste date și am făcut predicțiile pe datele de test, de asemenea normalizându-le și pe acestea în prealabil.

Acuratețea obținută cu acest model a fost una mică, aproximativ **0.29**.

Matricea de confuzie:



b. CNN (Convolutional Neural Network)

Pentru a obține o acuratețe sporită, am folosit un model de tip CNN, cu ajutorul bibliotecii TensorFlow și API-ul Keras. Voi prezenta în continuare diferite încercări și abordări folosite.

- Primul model CNN implementat

După preluarea datelor, am început prin a rescala imaginile astfel încât valorile pixelilor să fie între 0 și 1, în loc de 0 și 255, pentru a stabili mai mult procesul de învățare.

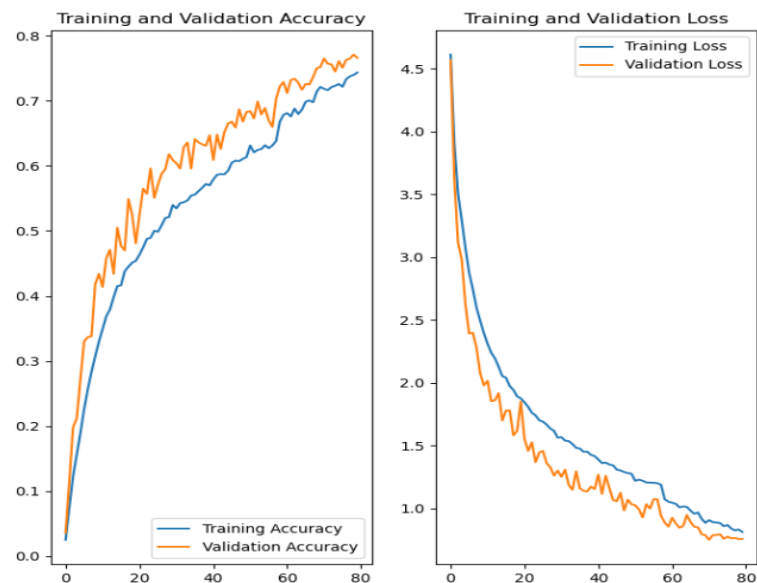
Apoi, pentru început, am folosit 4 straturi convoluționale *Conv2D*, fiecare dintre acestea urmat de către un strat de *MaxPooling2D*, pentru a reduce dimensiunea spațială a datelor, păstrând informațiile importante, pentru ca modelul să fie nevoit să învețe mai puțini parametri. Filtrele aplicate stratelor convoluționale au fost următoarele, progresiv: 64, 128, 256, 512, iar funcția de activare pe care am folosit-o a fost *ReLU* (dacă intrarea este negativă, întoarce 0, altfel întoarce chiar ceea ce primește la intrare).

După aceste straturi, am făcut un *Dropout* de 0.8, pentru a "dezactiva" aleatoriu 0.8 din neuroni, ca să previn ca modelul să fie overfit. Ulterior, am folosit un strat *Flatten* pentru a transforma matricea 2D de la ultimul strat convoluțional într-un vector 1D, pentru a putea fi fully connected la următorul strat. În cele din urmă, am folosit un strat *Dense*, fully connected, unde fiecare neuron este conectat la fiecare neuron din stratul anterior, iar pentru acesta am folosit *Softmax* ca funcție de activare. De asemenea, am folosit optimizatorul *adam* și funcția de pierdere *categorical_crossentropy*.

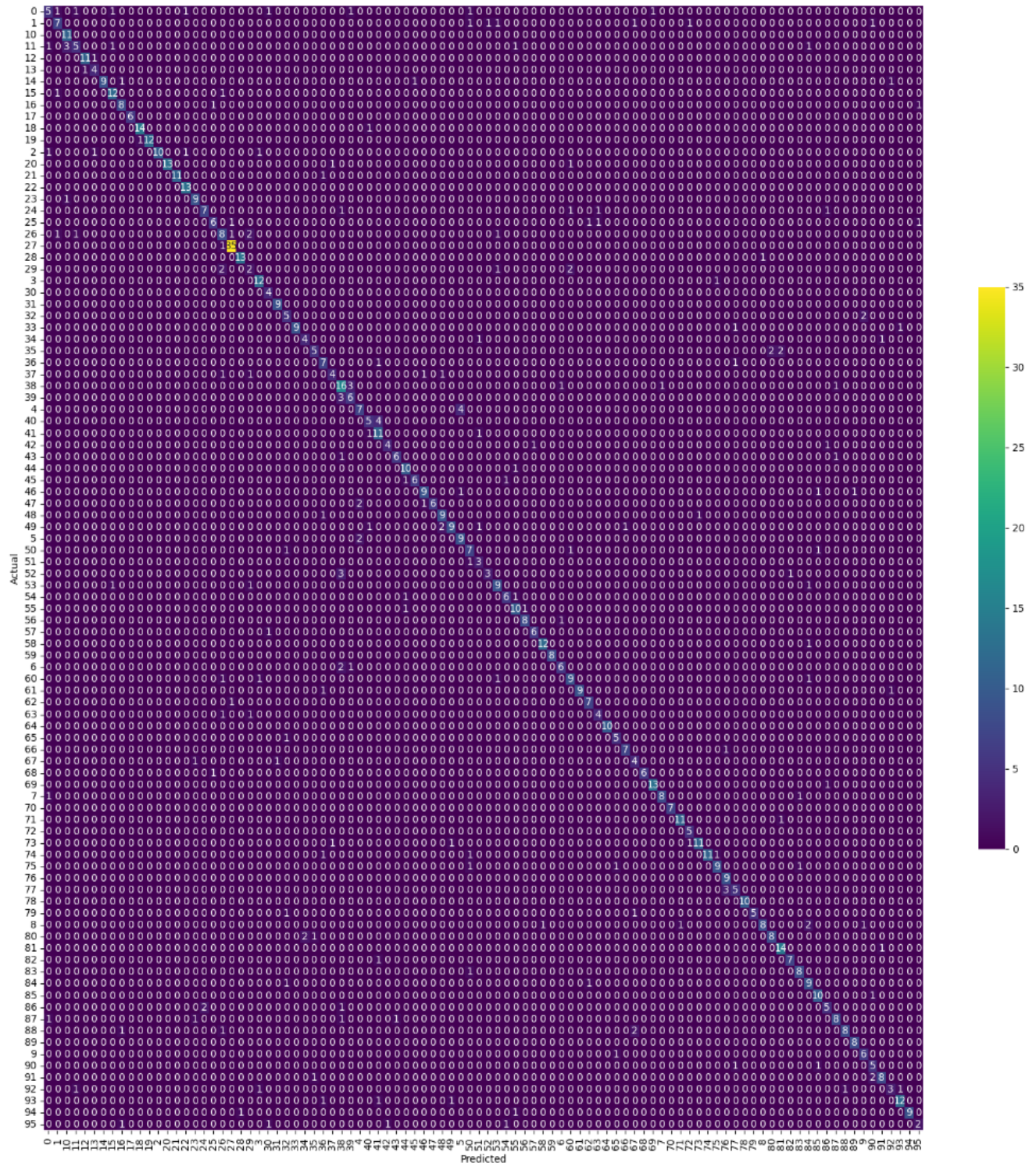
Așa arată construcția modelului inițial de la care am plecat:

```
57 ##### TRAIN MODEL #####
58
59 model079 = tf.keras.models.Sequential([
60     tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=(imgHeight, imgWidth, 3)),
61     tf.keras.layers.Conv2D(64, 3, activation='relu'),
62     tf.keras.layers.MaxPooling2D(),
63
64     tf.keras.layers.Conv2D(128, 3, activation='relu'),
65     tf.keras.layers.MaxPooling2D(),
66
67     tf.keras.layers.Conv2D(256, 3, activation='relu'),
68     tf.keras.layers.MaxPooling2D(),
69
70     tf.keras.layers.Conv2D(512, 3, activation='relu'),
71     tf.keras.layers.MaxPooling2D(),
72
73     tf.keras.layers.Dropout(0.8),
74     tf.keras.layers.Flatten(),
75     tf.keras.layers.Dense(256, activation='relu'),
76     tf.keras.layers.Dense(96, activation='softmax')
77 ])
78 model079.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
79 epochs = 80
80 history = model079.fit(trainDataset, validation_data=validationDataset, callbacks=callbacksSetup('model_079.keras'), epochs=epochs)
```

Acuratețea acestuia, deși l-am antrenat și un număr mai mare de epoci, stagnează pe la aproximativ 0.79.



Matricea de confuzie pentru acesta:



- Îmbunătățiri

Observând apogeul modelului precedent, am încercat să-l îmbunătățesc adăugând *BatchNormalization* înainte de fiecare layer convoluțional, pentru a normaliza output-ul fiecărui strat anterior astfel încât să aibă o medie de 0 și o deviație standard de 1.

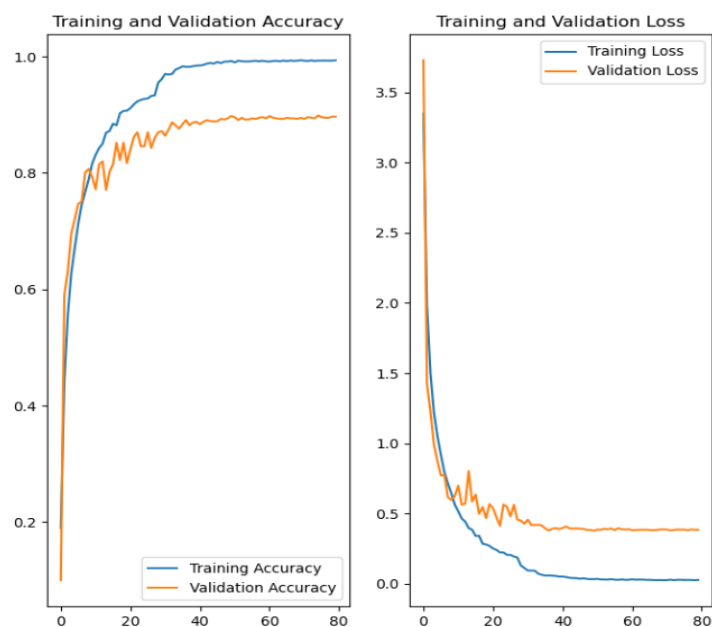
De asemenea, la fiecare layer convoluțional am adăugat *padding='same'*, pentru ca dimensiunea ieșirii convoluționale să rămână la fel ca intrarea. Practic adaug zero-uri în jurul imaginii pentru a putea ca filtrul de 3x3 să fie aplicat pe toată imaginea. Cum imaginea de intrare are dimensiunile 64x64 și aplic un filtru de 3x3 cu *padding='same'*, atunci ieșirea va avea, de asemenea, dimensiunile de 64x64. Fără *padding='same'*, dimensiunile ieșirii ar fi fost de 62x62.

În plus, am adăugat un *Dropout* de 0.2 după fiecare *MaxPooling2D*, pentru a mai preveni din overfitting.

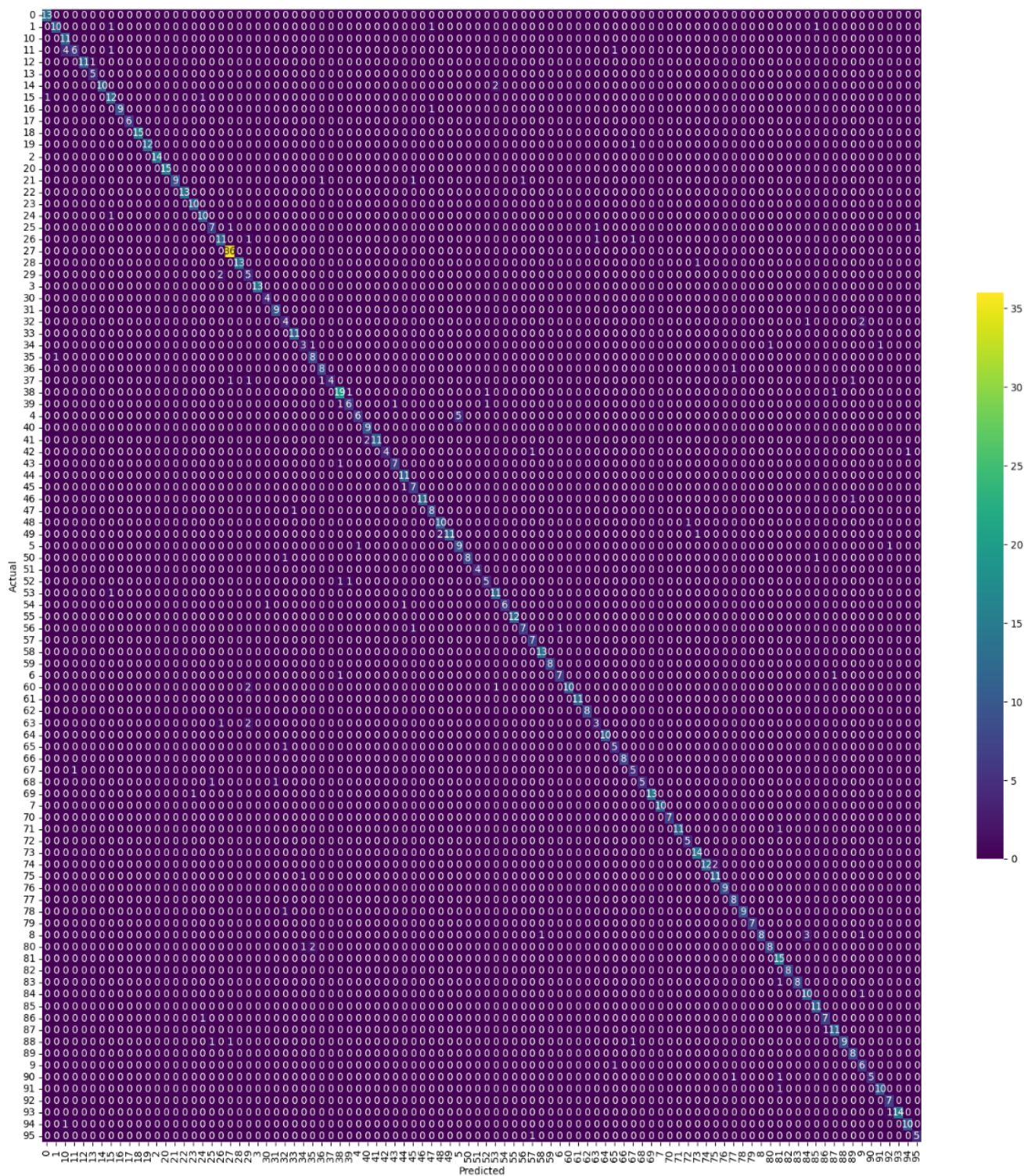
```
59 model090 = tf.keras.models.Sequential([
60     tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=(imgHeight, imgWidth, 3)),
61     tf.keras.layers.BatchNormalization(),
62     tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
63     tf.keras.layers.MaxPooling2D(),
64     tf.keras.layers.Dropout(0.2),
65
66     tf.keras.layers.BatchNormalization(),
67     tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
68     tf.keras.layers.MaxPooling2D(),
69     tf.keras.layers.Dropout(0.2),
70
71     tf.keras.layers.BatchNormalization(),
72     tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'),
73     tf.keras.layers.MaxPooling2D(),
74     tf.keras.layers.Dropout(0.2),
75
76     tf.keras.layers.BatchNormalization(),
77     tf.keras.layers.Conv2D(512, 3, padding='same', activation='relu'),
78     tf.keras.layers.MaxPooling2D(),
79     tf.keras.layers.Dropout(0.2),
80
81     tf.keras.layers.Dropout(0.8),
82     tf.keras.layers.Flatten(),
83     tf.keras.layers.Dense(256, activation='relu'),
84     tf.keras.layers.BatchNormalization(),
85     tf.keras.layers.Dense(96, activation='softmax')
86 ])
```


Observăm din graficul de mai jos că după un număr de epoci de aproximativ 50, acuratețea modelului începe să stagneze și duce în overfitting. De asemenea am mai observat că modelul a învățat foarte mult în primele epoci, reușind să ajungă la o acuratețe pe setul de date de validare de aproximativ 0.8 după numai 15 epoci.

Cu acest model am reușit să obțin o acuratețe de aproximativ **0.90**.



Matricea de confuzie pentru acesta:



- Optimizarea finală

Deși am experimentat cu configurația straturilor și cu numărul de epoci, nu am obținut îmbunătățiri substanțiale ale acurateții, în general aceasta oscilând între **0.885** și **0.905**.

Ce a fost însă un “game-changer” a fost să folosesc *Data Augmentation*, și anume:

- am dublat setul de date de train în felul următor: la cele 12000 de poze originale am adăugat încă 12000, adică varianta “flip”-uită pe orizontală a fiecărei poze. Astfel, modelul învața nu doar pe pozele originale, dar și pe pozele întoarse pe orizontală, având astfel abilitatea de a recunoaște mai bine pozele care reprezentau halucinații cu fețe și nu numai.
- la acest nou set de date (numit *final_dataset* în codul de mai jos) am adăugat o nouă augmentare, și anume un *RandomZoom* pe cele 24000 de imagini. Parametrul *-0.5* indică faptul că imaginea va fi mărită aleator cu un factor cuprins între 0% (nicio modificare) și 50% (mărire cu 50%). Dacă numărul ar fi fost pozitiv, ar fi indicat un zoom în afara imaginii (adică imaginea ar fi fost micșorată).

Pe lângă acest *Data Augmentation*, am adăugat și un regularizator L2 care adaugă cost suplimentar la funcția de pierdere, proporțional cu mărimea pătrată a ponderilor. Acest cost suplimentar are efectul de a încuraja modelul să aibă ponderi mai mici, ceea ce face modelul mai robust la zgomot și ajută la prevenirea overfitting-ului. În plus, am mai pus și un layer *Conv2D* de 512, pe lângă cele existente deja.

Așa am dublat setul de date aplicând *Data Augmentation-ul* cu flip și zoom:

```

56 def deterministicFlipDataAugmentation(image, label):
57     imageFlipped = tf.image.flip_left_right(image)
58     return imageFlipped, label
59
60 augmentedDataset = trainDataset.map(deterministicFlipDataAugmentation)
61 finalDataset = trainDataset.concatenate(augmentedDataset)

```

```

50
51 data_augmentation = tf.keras.Sequential(
52     [
53         tf.keras.layers.RandomZoom(-.5, input_shape=(img_height, img_width, 3), fill_mode="wrap"),
54     ]
55 )
56

```

Și așa arată construcția modelului final:

```

model092 = tf.keras.models.Sequential([
    data_augmentation,
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=(imgHeight, imgWidth, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(256, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(512, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Dropout(0.2),

    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(512, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),

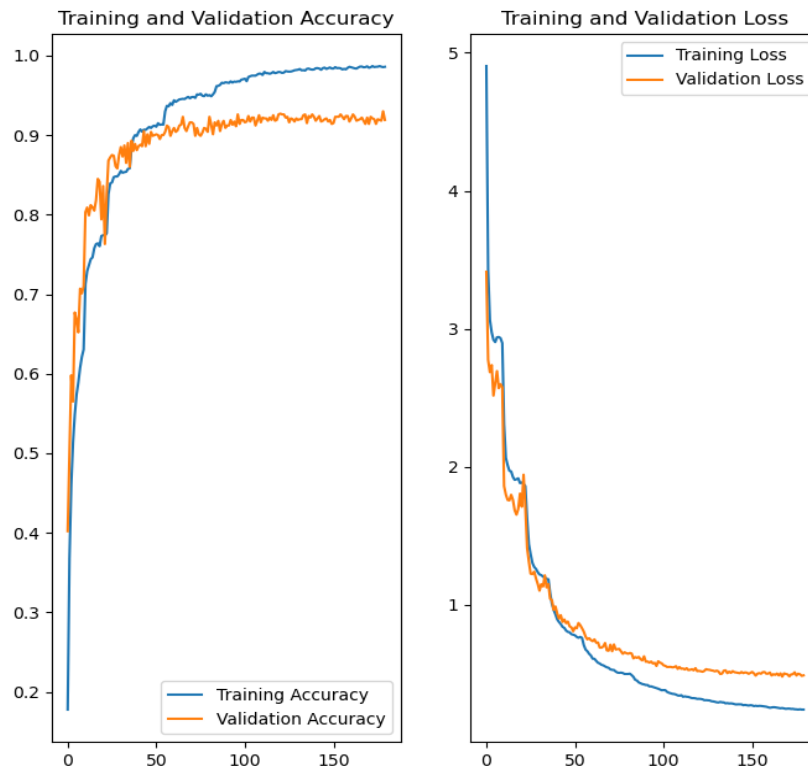
    tf.keras.layers.Dropout(0.8),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(96, activation='softmax')
])

# compilam modelul, folosind optimizatorul adam si ca functie de pierdere, categorical_crossentropy
model092.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

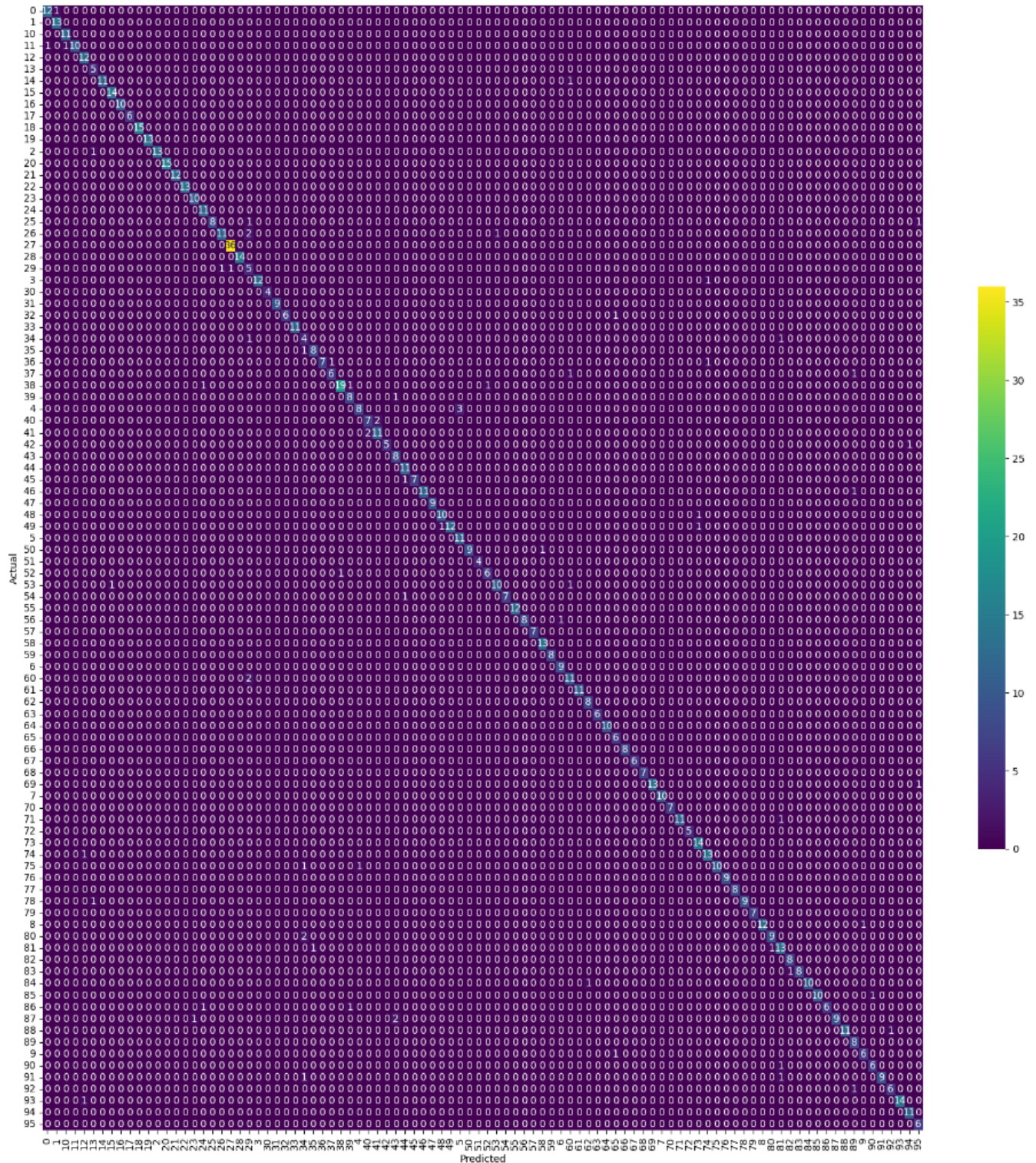
```


Numărul de epoci l-am variat destul de mult până am ajuns la o acuratețe dorită, însă, spre deosebire de modelele precedente, a fost nevoie de un număr mai mare, între 200 și 300.

Spre deosebire de modelul precedent, overfitting-ul s-a produs mult mai târziu, deoarece și învățarea modelului pe datele de antrenament a fost mai liniară și mai uniformă. Observăm acum că dacă îl lăsam numai aproximativ 75 de epoci, nu ar fi atins capacitatea maximă de învățare pe datele de train, așa cum se vede în graficul de mai jos creșterea acurateții la antrenare de la aproximativ 0.95.



Matricea de confuzie pentru acesta:



Așadar, acest model mi-a adus o acuratețe de **0.92628** în clasamentul final al competiției, deși pe clasamentul public (unde evaluarea se făcea cu 30% din numărul total de imagini de testare) acuratețea era de *0.93600*.

V. Funcții *callback* folosite pentru o antrenare mai eficientă

Pentru a economisi timp în antrenarea modelului, am folosit funcții *callback* care:

- reduc rata de învățare atunci când *val_loss* nu se îmbunătățește timp de 5 epoci:

```
reduceLearningRate = ReduceLROnPlateau(monitor='val_loss',  
patience=10, verbose=1, mode='min', factor=0.5, min_lr=1e-5)
```

- opresc antrenarea modelului dacă *val_accuracy* nu se îmbunătățește timp de 100 de epoci:

```
earlyStopping = EarlyStopping(monitor='val_accuracy', patience=75,  
verbose=1, mode='max', restore_best_weights=True)
```

- salvează modelul cu cel mai bun *val_accuracy* pentru a putea fi încărcat ulterior:

```
checkpointSave = ModelCheckpoint(checkpoint,  
monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
```

```
def callbacksSetup(checkpoint):  
    reduceLearningRate = ReduceLROnPlateau(monitor='val_loss', patience=10, verbose=1, mode='min', factor=0.5, min_lr=1e-5)  
    earlyStopping = EarlyStopping(monitor='val_accuracy', patience=75, verbose=1, mode='max', restore_best_weights=True)  
    checkpointSave = ModelCheckpoint(checkpoint, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')  
    return [reduceLearningRate, earlyStopping, checkpointSave]
```

VI. Concluzie

Fiind nevoie de multe încercări și abordări pentru a obține o construcție de model care să ofere o acuratețe cât mai bună, pot spune că, în urma acestor experimente, cea mai mare acuratețe (**0.92628**) am obținut-o folosind un CNN, pe care l-am și folosit ca model principal.