Introducere

Resurse

- https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
- https://www.kaggle.com/datasets/imsparsh/gesture-recognition
- https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9

❕ Convolutional neural networks (CNN) → Model matematic de predictie, folosit in computer vision pentru actiuni precum: `object-detection`, `touchless-control`, si multe altele...

📏 “Invatarea” modelului este o actiune care se bazeaza ca principiu pe algebra lineara, in specific pe linear regression.

$$y = ax + b$$

Retea Neuronala

- Conceptul de retea neuronala este preluat din viata reala. O retea neuronala este o serie de “neuroni”, impartiti in mai multe straturi. “Neuronii” actioneaza

in propriile lor arii vizuale. Un strat de neuroni mai aproape de input, va creea `weight-uri` care sunt folosite pentru a retine o combinatie de parametrii (care pot fi asimilati cu `a` si `b`) de mai sus, pentru a obtine rezultatul `y` (spre exemplul, a pune un label `GESTURE_0` pentru o valoare primita `x`)

- Ecuatia poate fi perceputa ca urmatoarea:

$$y = A * X, X \in R^n$$

unde `n` este numarul de dimensiuni al retelei neuronale.

Structura

Dupa cum este precizat mai sus, neuronii sunt impartiti in `layere` de trei tipuri:
1. Input layer() → input
`x`
2. Hidden layer(s) → operations
`x*a`
3. Output layer → output / label
`y`

Layers

- Layer-urile folosite pentru a “invata” parametrii se numesc, `Convolutional Layers`

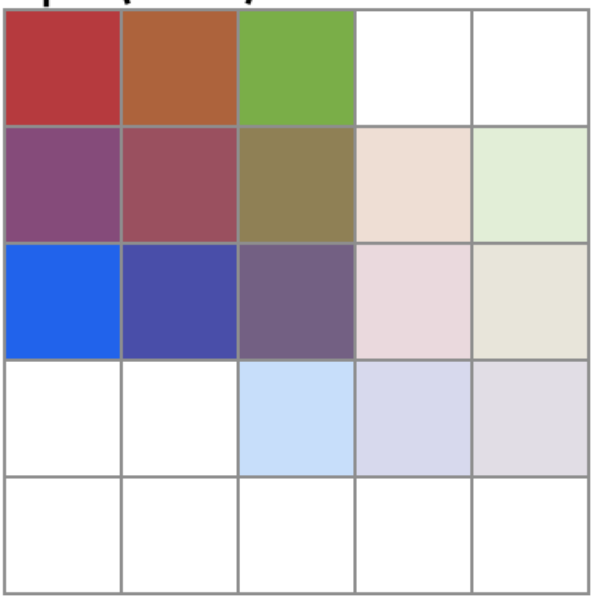
  Aici, layer-urile iau informatie (preprocesata) a pixelilor, aleg formatiuni de forma `R^n` (in cazul citirii videoclipurilor, `n = 3` , o dimensiune fiind reprezentata de pixelii de `height` , o dimensiune fiind reprezentata de pixelii de `width` , alta de pixeli de `height` iar ultima de numarul de `frames` pe care il avem. Dupa ce aleg aceste formatiuni, inmultesc elemente dintr-o matrice generata la inceputul procesarii, le aduna intr-un rezultat, si obtin o matrice rezultat, care este retinuta pentru a invata modelul paramterii `A` din ecuatie.

Intr-un context 2D, am avea o matrice de dimensiune D^2, unde D este de obicei o valoare impara.

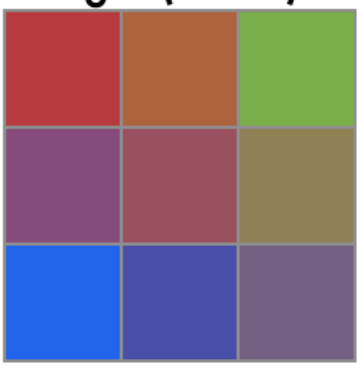
```
n = 2
d = 3

[5, 15, 3]
[45, 22, 1] * X (de dimensiune 2) => y
[1, 2, 3]
```

▼ Vizualizare operatie de convolutie

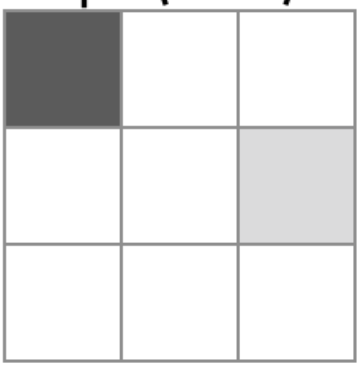
Input (5 × 5):



Weight (3 × 3):



Output (3 × 3):



Matrice Rezultat (Feature Map)

- Din operatiile de inmultire cu `weight` uri, se aplica un process numit “activare”, care adauga non-linearitati pentru a computa rezultatul convolutiei.
- Pentru a obtine un feature map de aceeasi dimensiune ca cea folosita ca input (sau informatiile primite din alte layere), se va mari `padding` ul. Matricea de input va avea cate un rand sau o coloana de 0 in fiecare directie (`sus`, `jos`, `stanga`, `dreapta`)

```
input[5x5] *  0 0 0 0 0
                          0 2 1 5 0
                          0 2 6 3 0
                          0 6 4 1 0
                          0 0 0 0 0
```

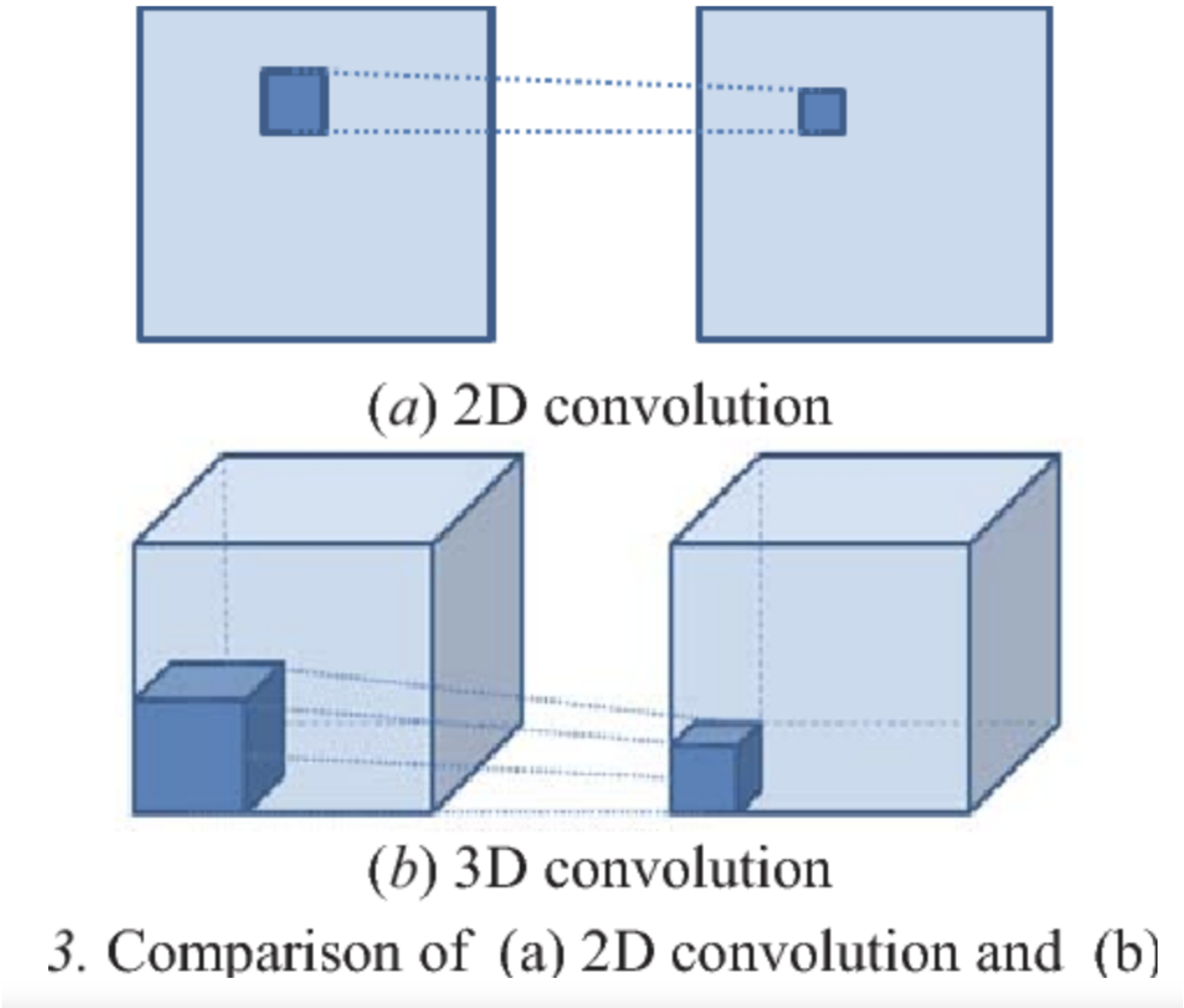
  - Pentru a reduce complexitate, se va folosi un `stride` , care determina numarul de linii si coloane pe care matricea (filtrul) il sare in timpul operatiei de convolutie.

- Layer-urile care optimizeaza rezultatele din Feature Map-uri se numesc `Pooling Layers`. In cazul nostru, se vor folosi `MaxPooling` layers, care optin rezultate maxime prin scanarea matricii printr-o dimensiune aleasa (2 × 2 spre exemplu)
- Layer-urile care se ocupa de folosirea tuturor informatiilor pentru a computa un rezultat (pentru normalizare sau finalizare) se numesc Layere de normalizare, sau, in cazul operatiei de output, se numesc Dense Layers.

📏 Aceste layer-uri (Dense Layers) primesc o configuratie de neuroni, o putere a lui doi, asezate descrescator in functie de numarul de neuroni ales per layer.

3D Convolution

- In cazul procesarii videoclipurilor, avem 3 dimenisuni (`H x W x T`)
  - `H` → height
  - `W` → width
  - `T` → time



3. Comparison of (a) 2D convolution and (b)

Diferenta intre practicarea operatiilor de convolutie, este ca se va adauga o dimensiune in plus, iar filtrul si feature map-urile in loc sa fie o matrice, devin un cub.

3DCNN - Pasi

1. Formatare imagini
2. Design neural network
3. Train neural network (pe un `x` obiect de interes si un label `y`)

4. Salvare network

Formatare imagini

- Fie un input x de o anumita rezolutie. Noi trebuie sa aducem acest x (care este o imagine) intr-un format pe care modelul de `tensorflow` sa-l inteleaga.

```
# Pentru un dataset salvat local, se vor obtine
# path-urile, label-urile si index-urile fiecarei
# imagin din dataset.

# Se va folosi pandas pentru a citi csv-ul, si a retine infor
# intr-un DataFrame pandas.

def get_set_frames():
    train_df = pd.read_csv("./dataset/train.csv", sep=";", he
    train_df.rename(columns={0: "Image", 1: "Label", 2: "Imag
    train_df.info(memory_usage="deep")
    test_df = pd.read_csv("./dataset/val.csv", sep=";", heade
    test_df.rename(columns={0: "Image", 1: "Label", 2: "Image
    test_df.info(memory_usage="deep")

    print(f"Total videos for testing: {len(test_df)}")
    print(f"Total videos for training: {len(train_df)}")
```

- Din path-urile luate mai sus se vor citi imaginile prin `cv2` , care ne va returna o matrice de pixeli a imaginii. Pentru a omite detalii irelevante din imagine (de obicei detalii de margine), se va cropa imaginea in functie de rezolutia initiala:

```
def crop_center_square(frame):
    y, x = frame.shape[0: 2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
```

```
    return frame[start_y : start_y + min_dim, start_x : start
```

- Se vor aduce imaginile la o rezolutie standard pentru a fi luate ca input de model (in cazul nostru `224 x 224`)

```
def load_image(df, path):
    labels = df["Label"].values.tolist()
    img = cv2.imread(path)

    frames = crop_center_square(img)

    frames = cv2.resize(frames, (224, 224))
    return frames, labels
```

Dupa acest pas, sunt salvate si label-urile din fisier-ul `.csv`, si se obtine un `train_set` si un `validation_set`

Design Neural Network

Se iau o serie de poze cu label-urile lor atribuite. Pozele sunt primite ca serii de 30 de imagini (unde 30 ar fi numarul de frame-uri al videoclipurilor), unde fiecare pixel e reprezentat pe 3 canale (RGB).

Input Shape = `(30, 224, 224, 3)`

```
# conv_filters -> numarurl de filtre aplicate pe input
# dense_nodes -> numarul de noduri folosite in dense_layers
# activation='relu' -> functia de activare (rectified linear uni
# activation='softmax' -> functia de activare catre output layer

def conv3D(conv_filters=(16, 32, 64, 128), dense_nodes=(256, 128
    model = Sequential()

    model.add(Conv3D(conv_filters[0], (3, 3, 3), activation='rel
    model.add(MaxPooling3D((2, 2, 2)))
```

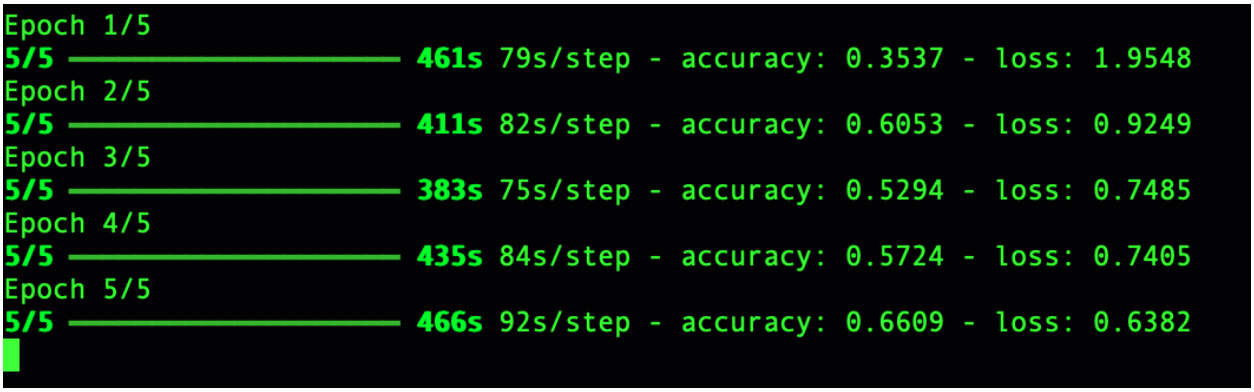
```
    model.add(Conv3D(conv_filters[1], (3, 3, 3), activation='rel
    model.add(MaxPooling3D((2, 2, 2)))

    model.add(Conv3D(conv_filters[2], (3, 3, 3), activation='rel
    model.add(MaxPooling3D((2, 2, 2)))

    model.add(Conv3D(conv_filters[3], (3, 3, 3), activation='rel
    model.add(MaxPooling3D((2, 2, 2)))

    model.add(Flatten())
    model.add(Dense(dense_nodes[0], activation="relu"))

    model.add(Dense(dense_nodes[1], activation="relu"))

    model.add(Dense(5, activation='softmax'))
    model.compile(optimizer="adam", loss='categorical_crossentro
    return model
```

Train Neural Network + Salvare Model

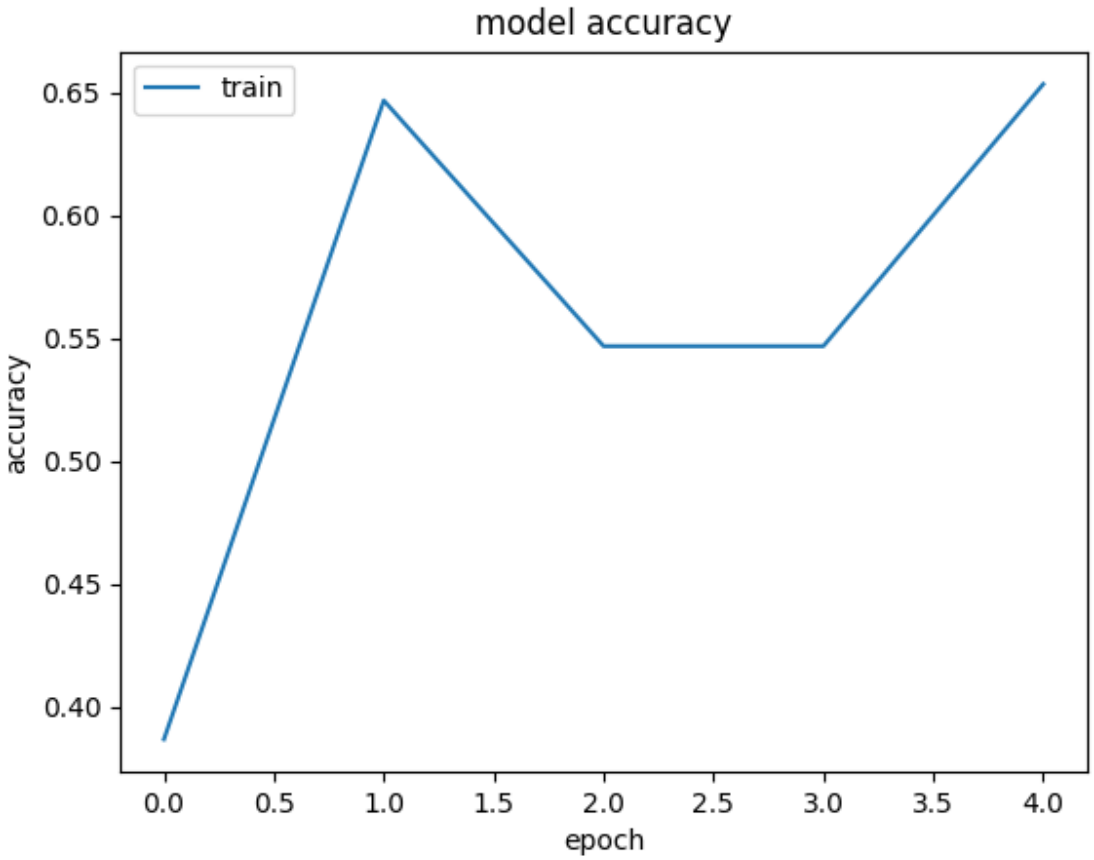
Din dataset se servesc 150 de imagini ca train_set si 30 de imagini ca validation set.

```
# epochs -> numar de iteratii ale procesului de invatat al neuro

def train_3dcnn(train_data, train_gestures):
    train_data = tf.convert_to_tensor(train_data)
    model = conv3D()
    history = model.fit(train_data, train_gestures[:150], epochs
    plt.plot(history.history["accuracy"])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
```

```
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    model.save("3dcnn.h5")

def main():
    trainset = get_final_model_trainset()
    train_3dcnn(trainset[2][:150], trainset[3][:150])
```



- `accuracy` → masura a cat de bun este modelul la a prezice parametrii
- `loss` → masura pentru a determina diferenta dintre rezultatul prezis si rezultatul la care utilizatorul s-ar fi asteptat (invers proportional cu `accuracy`)


model accuracy
train
accuracy
epoch
0.65
0.60
0.55
0.50
0.45
0.40
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0