FINAL PRESENTATION

AI-BASED GREEN VIDEO ENCODING FOR SOCIAL MEDIA

ELLIOT COLE & MATEO ZOUGHEBI
MIHAI MITREA, MOHAMED ALLOOUCHE, CARL DE SOUSA

OVERVIEW

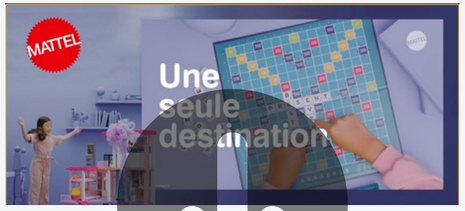
I) CONTEXT

Objectif : Discover how videos are re-encoding on social networks

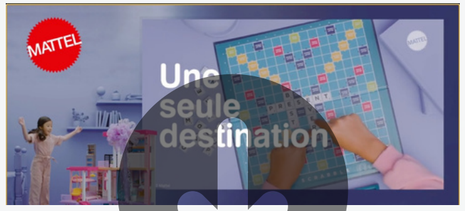
Last time : Attempted to create a database with API without good results. The same video from Vidmizer was uploaded on YouTube and TikTok and downloaded without compression manually.

II) DATASET/DATALOADER 3D

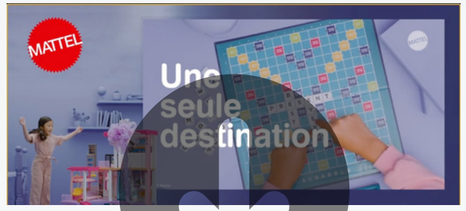
Objectives: Train a 3D classifier with inputs as mp4 videos and assign labels between YouTube Vidmizer and TikTok.



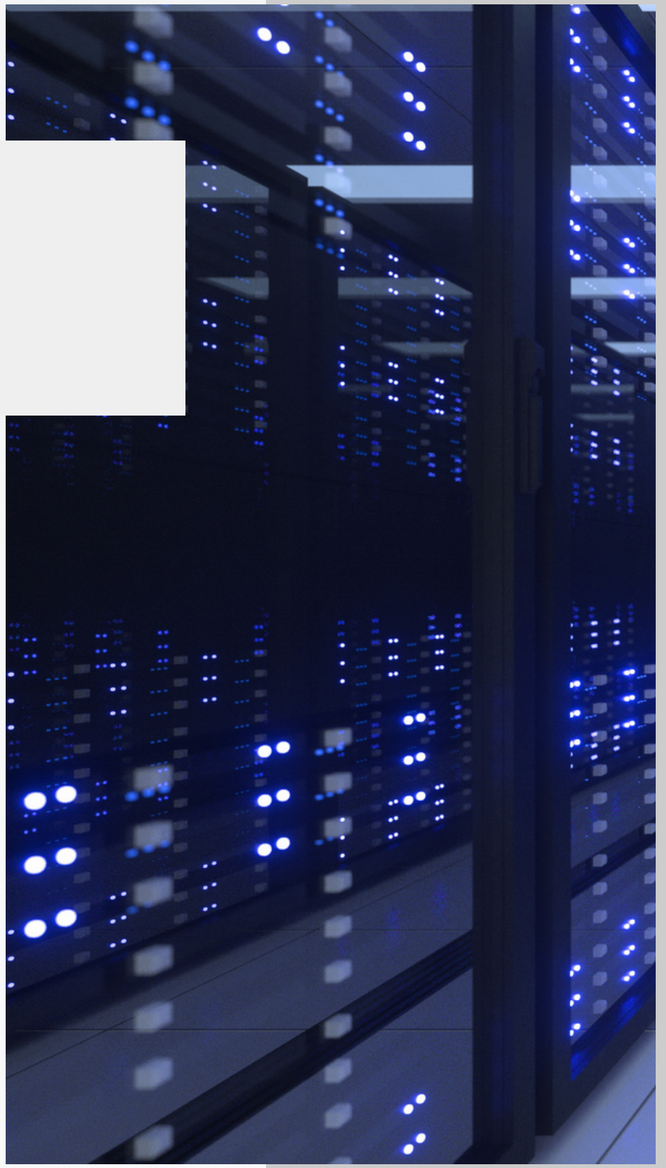
Upload & Download
from **YouTube**



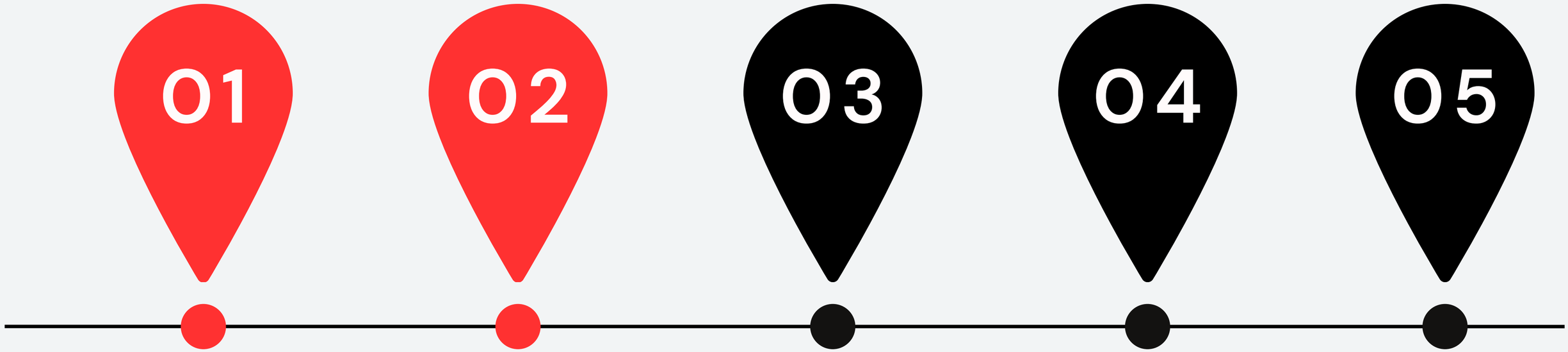
Download
from **Vidmizer**



Upload & Download
from **TikTok**



III) OUR STRATEGY



01 — 2D MULTI-CLASS CLASSIFIER

hand crafted classifier

02 — 2D BINARY CLASSIFIER

ResNET-18

03 — 2D BINARY CLASSIFIER X3

ResNET-18

04 — CASCADE

05 — 2D MULTI-CLASS CLASSIFIER

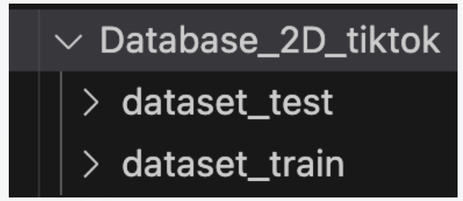
pixel domain

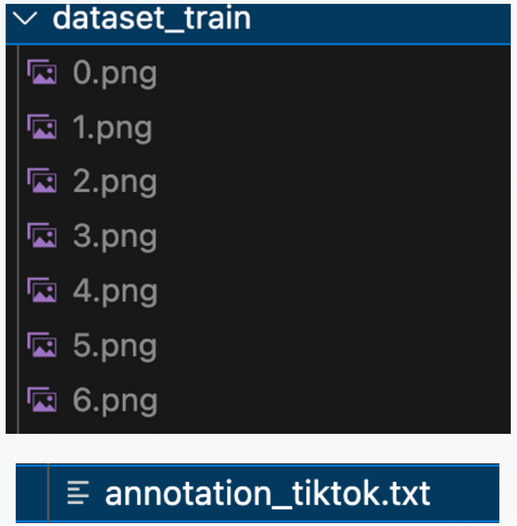
compressed domain

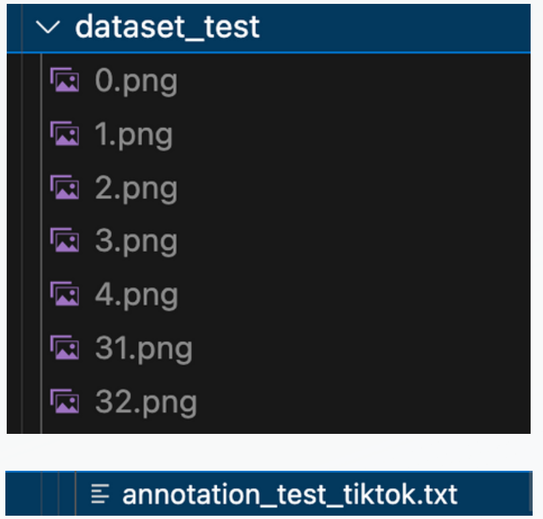
IV) DATASET 2D BINARY, PIXEL CASE

Dataset 3D → Dataset 2D tiktok case | Dataset 2D ytb case | Dataset 2D vidmizer case

Dataset 2D tiktok case =







Each folder is balanced: 1/2 TikTok, 1/4 Vidmizer, 1/4 YouTube for this case

IV) DATA 2D BINARY CASE : CUSTOM LOADER PIXEL

```python
############ Custom Dataset #############
class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file, header=None, sep=' ')
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        # Ici nos indices de lignes étaient confondus avec notre nom d'image
        # Ce n'est plus le cas dans la version binaire ce qui impose la modification suivante :
        #img_path = self.img_dir + '/' + str(idx) +'.png'
        # print(idx)
        img_path = self.img_dir + '/' + str(self.img_labels.iloc[idx,0]) +'.png'
        # print(img_path)
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
###############################################
```

```python
training_dataset = CustomImageDataset(
    annotations_file = annotation,
    img_dir = dataset_2D,
    transform=preprocess
)
```

Annotation.txt

DataSet_tiktok

Batch :
[batch_size, 3 , 576, 576]

2 Labels: Tiktok and others

Get_parameters : Parameters to normalize the batch

```
mean: tensor([0.4098, 0.3934, 0.3934])
std: tensor([0.3189, 0.3121, 0.3026])
```

V) FIRST NETWORK: ORIGIN

```
##### Image preprocessing #####
from torchvision import transforms
preprocess = transforms.Compose([
        transforms.Resize(576),  # image batch, resize smaller edge to 576
        transforms.CenterCrop(576),  # image batch, center crop to square 576x576
        transforms.ToPILImage(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.4098, 0.3934, 0.3934], std=[0.3189, 0.3121, 0.3026])
])
```

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc1 = nn.Linear(318096, 120) # 120 -> 100 000
        self.fc2 = nn.Linear(120, 84) #
        self.fc3 = nn.Linear(84, 3)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

```
Accuracy for class: tiktok is 0.0 %
Accuracy for class: vimizer is 100.0 %
Accuracy for class: youtube is 0.0 %
```

Our first test was on three classes with an unbalanced dataset, and our network predicted only the class with the larger number of I frames.
Overall accuracy : 40%

V) FIRST NETWORK: BACK TO BINARY AND CLASSIC NETWORK

For computation time, we switched to Google Colab to use GPU.

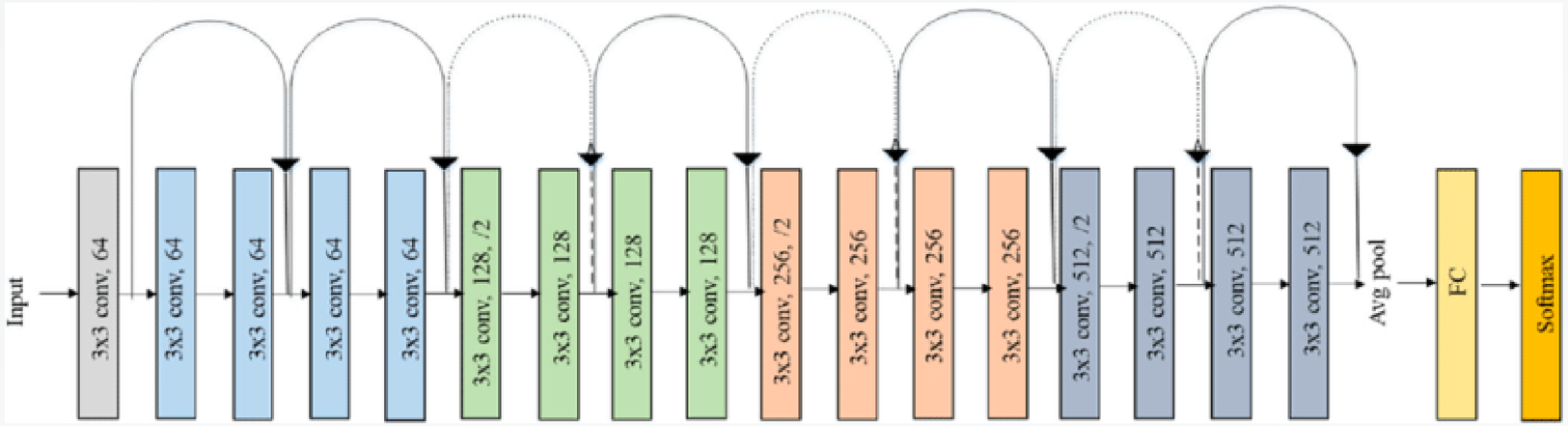
Data ⟶

```
net = torchvision.models.resnet18(weights=None)
num_ftrs = net.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to ``nn.Linear(num_ftrs, len(class_names))``.
net.fc = nn.Linear(num_ftrs, 2)
```

⟶ **0 : tiktok**

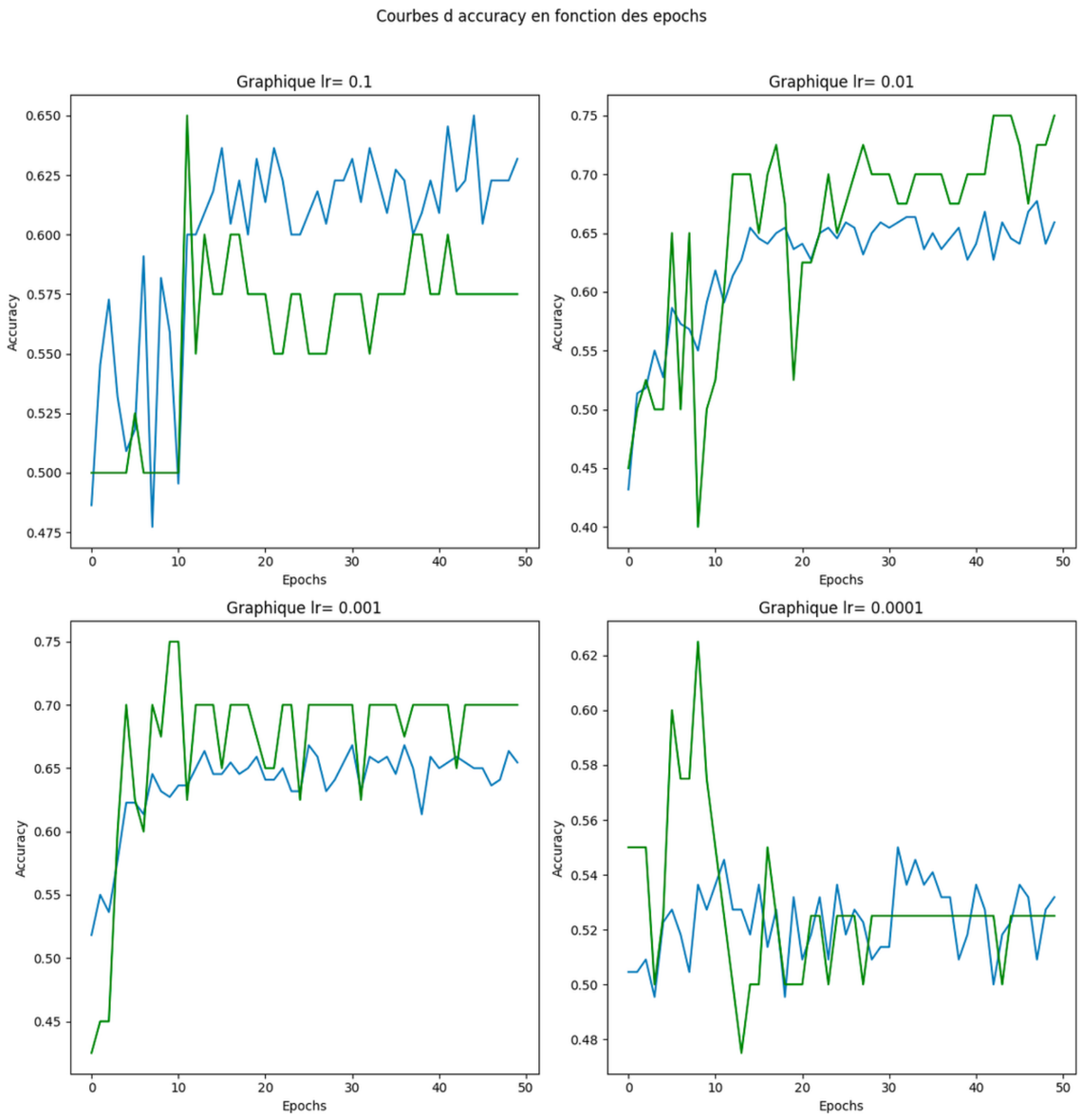
⟶ **1 : others**

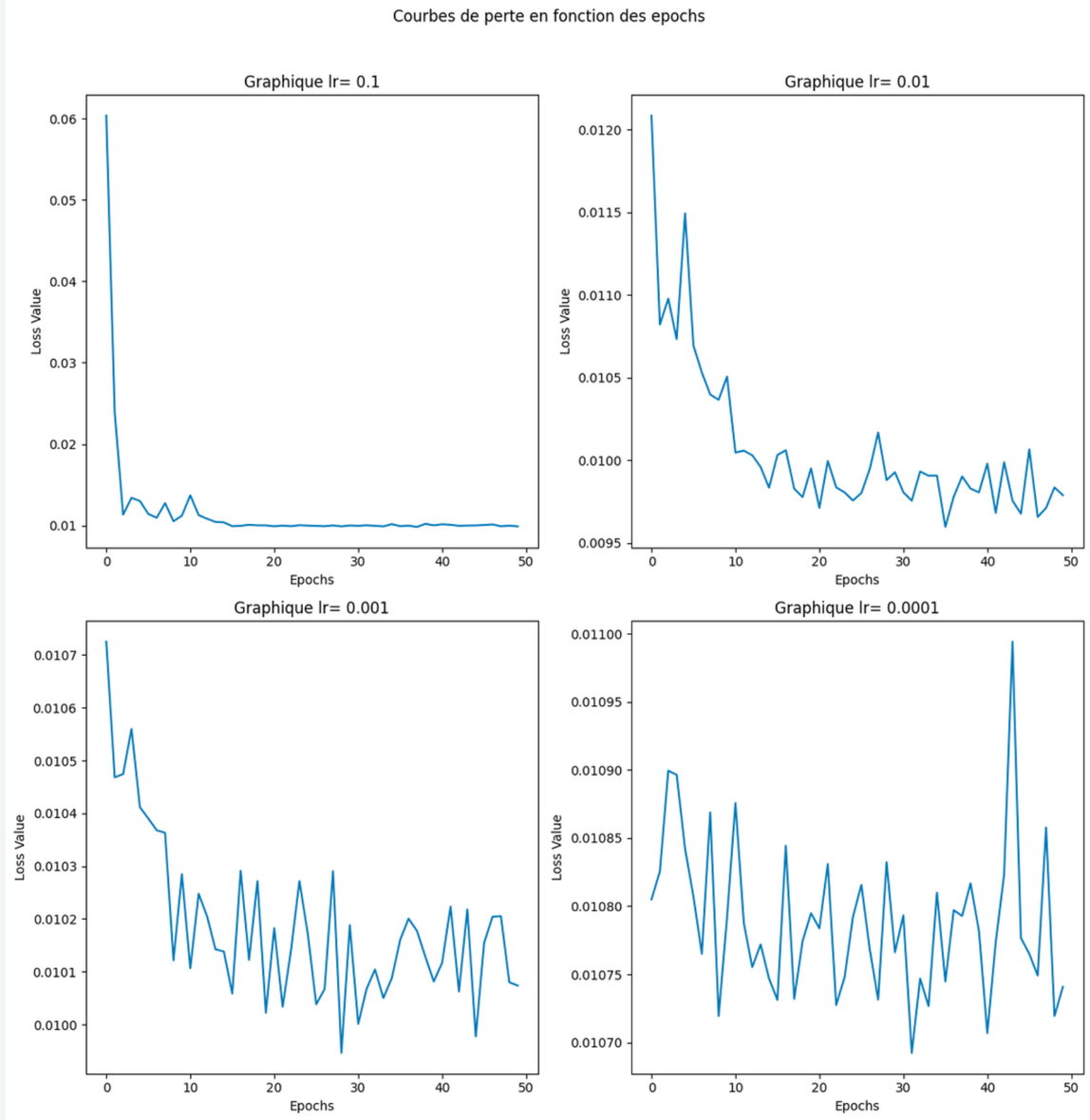
ResNET18



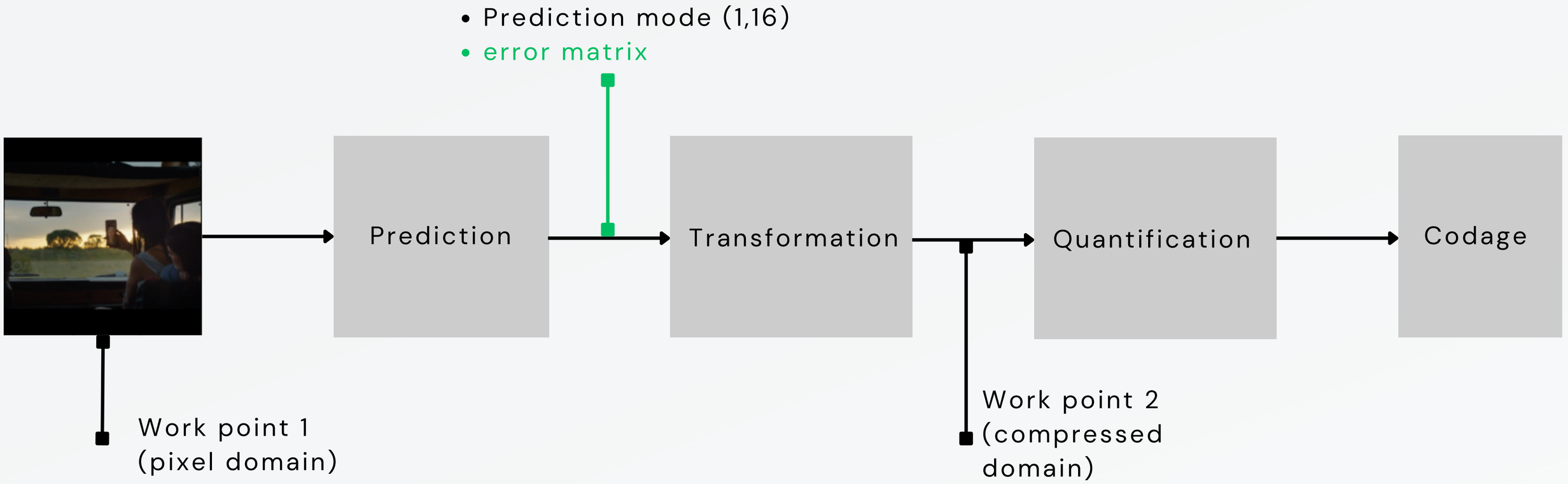
V) FIRST NETWORK: RESULTS

Accuracy & Loss





VI) NEW STRATEGY



Objectives:

- Change our project vision and identify two areas of focus
- Bad results in the pixel domain, try to classify in the compressed domain
- Assumption : we will get more information on the codding parameter by working in the compressed domain

VI) COMPRESSED DOMAIN - DATASET

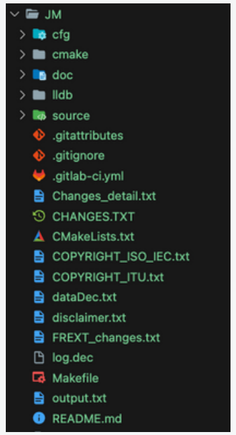
Conversion pipeline (pixel to compressed)

video.mp4

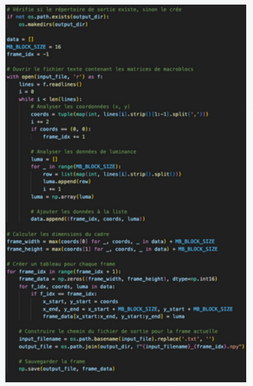
```
ffmpeg -i "video.mp4" -c:v copy "video.264"
```

video.264

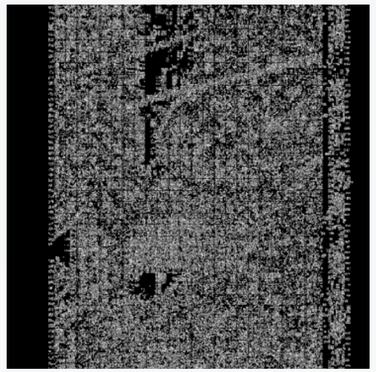
```
./JM/bin/umake/gcc-10.2/x86_64/release/ldecod -i "video.264"
```



video.txt



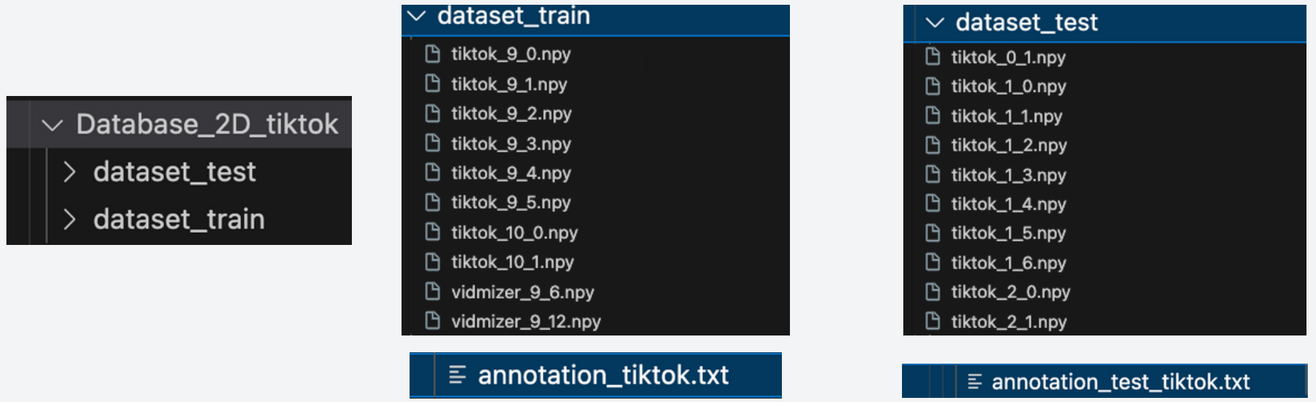
```
python3 decode_txt.py "video.txt" "video.npy"
```



video.npy
(array)

VI) COMPRESSED DOMAIN - DATASET

The dataset



	Vidmizer vs other	tiktok vs other	youtube vs other
Training	vidmizer 88	tiktok 55	youtube 77
	other 44/44	other 27/27	other 36/36
Testing	vidmizer 18	tiktok 8	youtube 10
	other 8/10	other 4/4	other 5/5

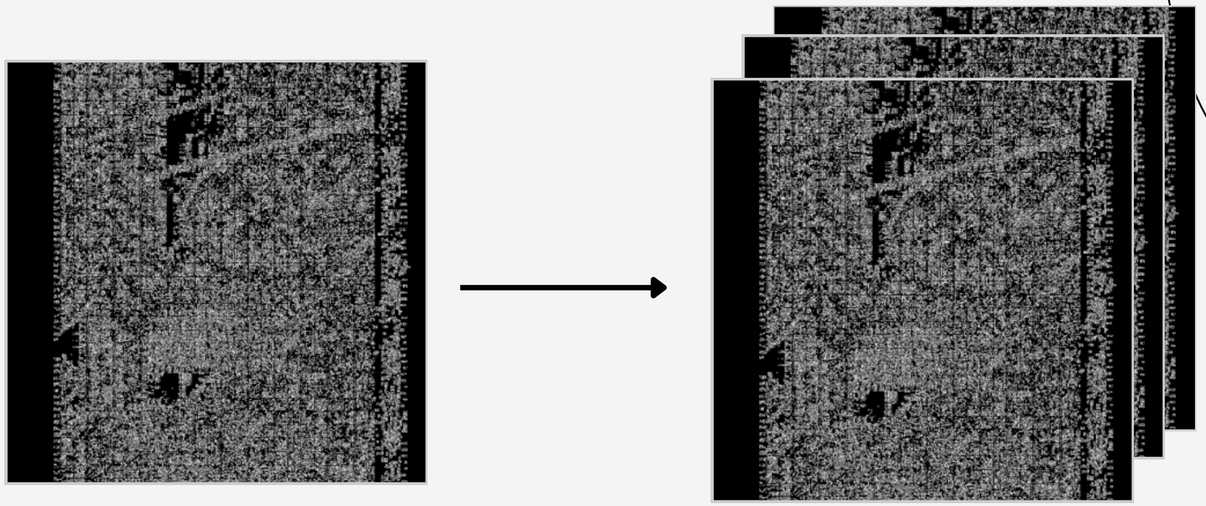
VI) COMPRESSED DOMAIN - DATALOADER

```python
class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, npy_dir, transform=None, target_transform=None):
        self.npy_labels = pd.read_csv(annotations_file, header=None, sep=' ')
        self.npy_dir = npy_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.npy_labels)

    def __getitem__(self, idx):
        # Ici nos indices de lignes étaient confondus avec notre nom d'image
        # Ce n'est plus le cas dans la version binaire ce qui impose la modification suiva
        #npy_path = self.npy_dir + '/' + str(idx) +'.png'
        # print(idx)
        npy_path = self.npy_dir + '/' + str(self.npy_labels.iloc[idx,0]) +'.npy'
        # print(npy_path)
        array = np.load(npy_path).astype(np.float32)
        array3ch = np.stack((array,)*3, axis=-1)
        label = self.npy_labels.iloc[idx, 1]
        if self.transform:
            array3ch = self.transform(array3ch)
        if self.target_transform:
            label = self.target_transform(label)
        return array3ch, label
```

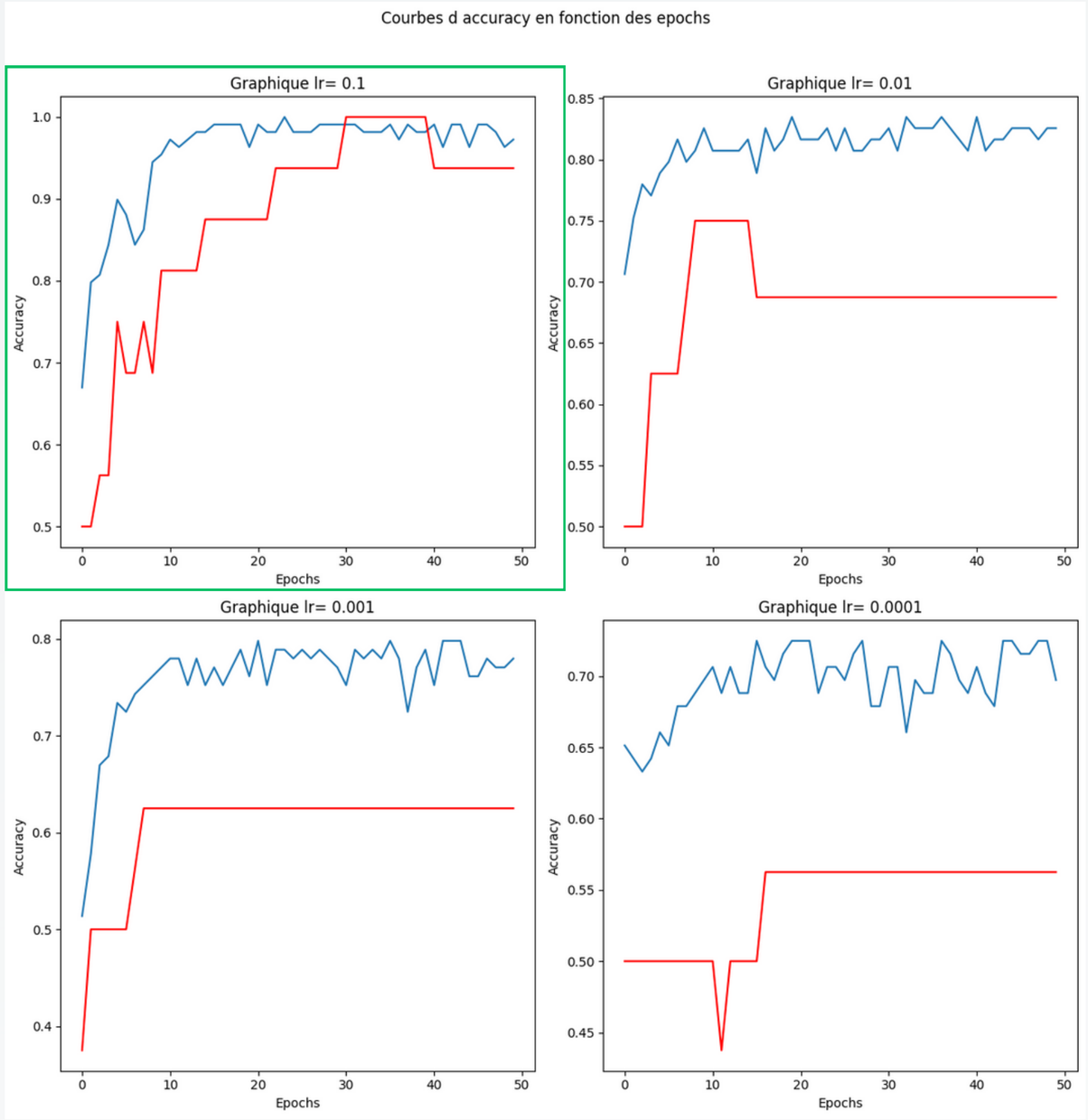
- Duplicated the (**y**,u,v) component

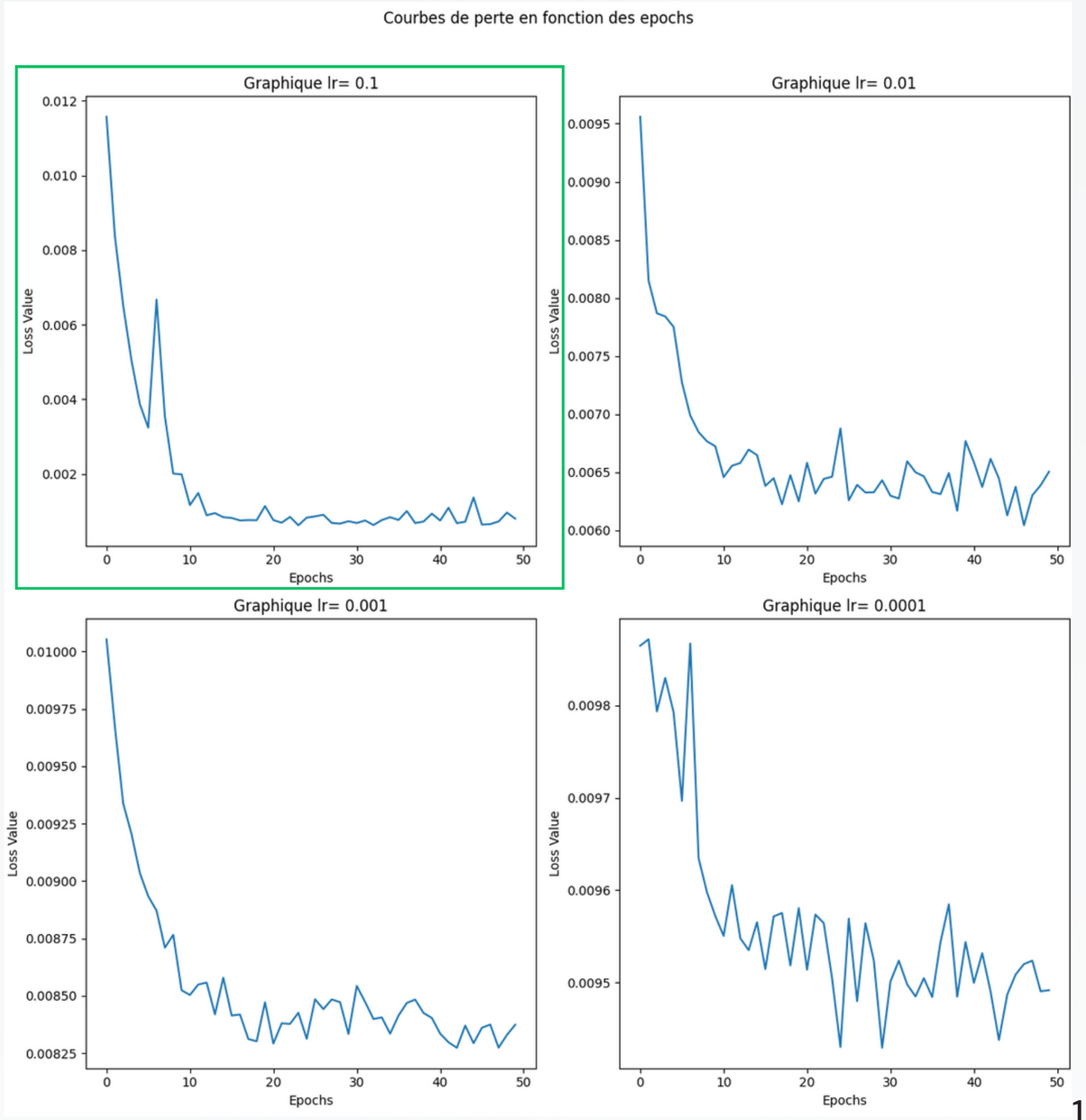


y **(y,y,y)**

V) RESULTS: BINARY TIKTOK CLASSIFIER

Accuracy & Loss





V) RESULTS: BINARY TIKTOK CLASSIFIER

Confusion Matrices

	Prédit Positif	Prédit Négatif
réel Positif	8	1
réel Négatif	0	7

lr = 0,1

	Prédit Positif	Prédit Négatif
réel Positif	5	2
réel Négatif	3	6

lr = 0,01

	Prédit Positif	Prédit Négatif
réel Positif	6	4
réel Négatif	2	4

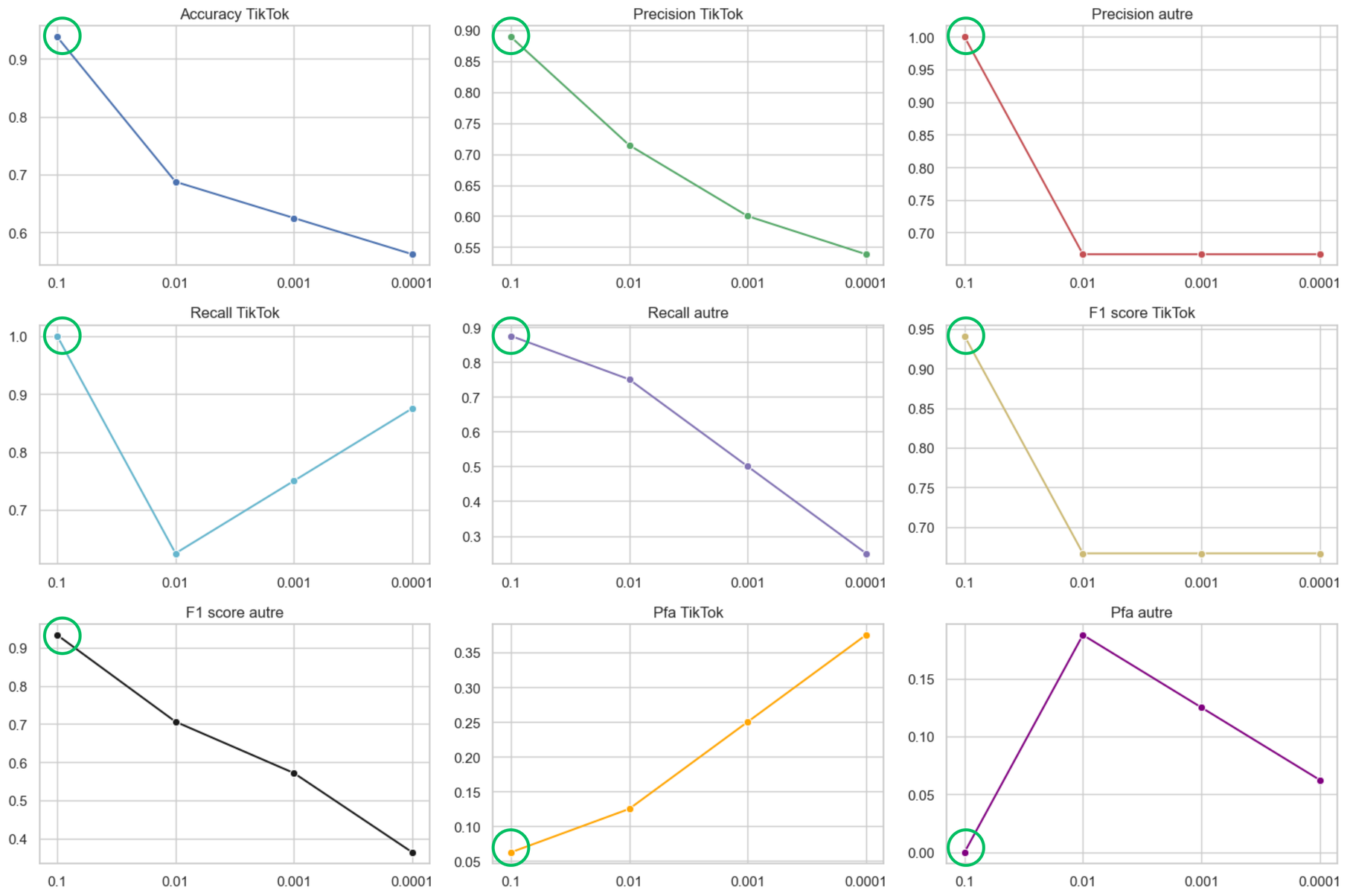
lr = 0,001

	Prédit Positif	Prédit Négatif
réel Positif	7	6
réel Négatif	1	2

lr = 0,0001

V) RESULTS: BINARY TIKTOK CLASSIFIER

Metrics



Recall = TP / (TP + FN)
(TP = True Positives, FN = False Negatives)

Precision = TP / (TP + FP)
(FP = False Positives)

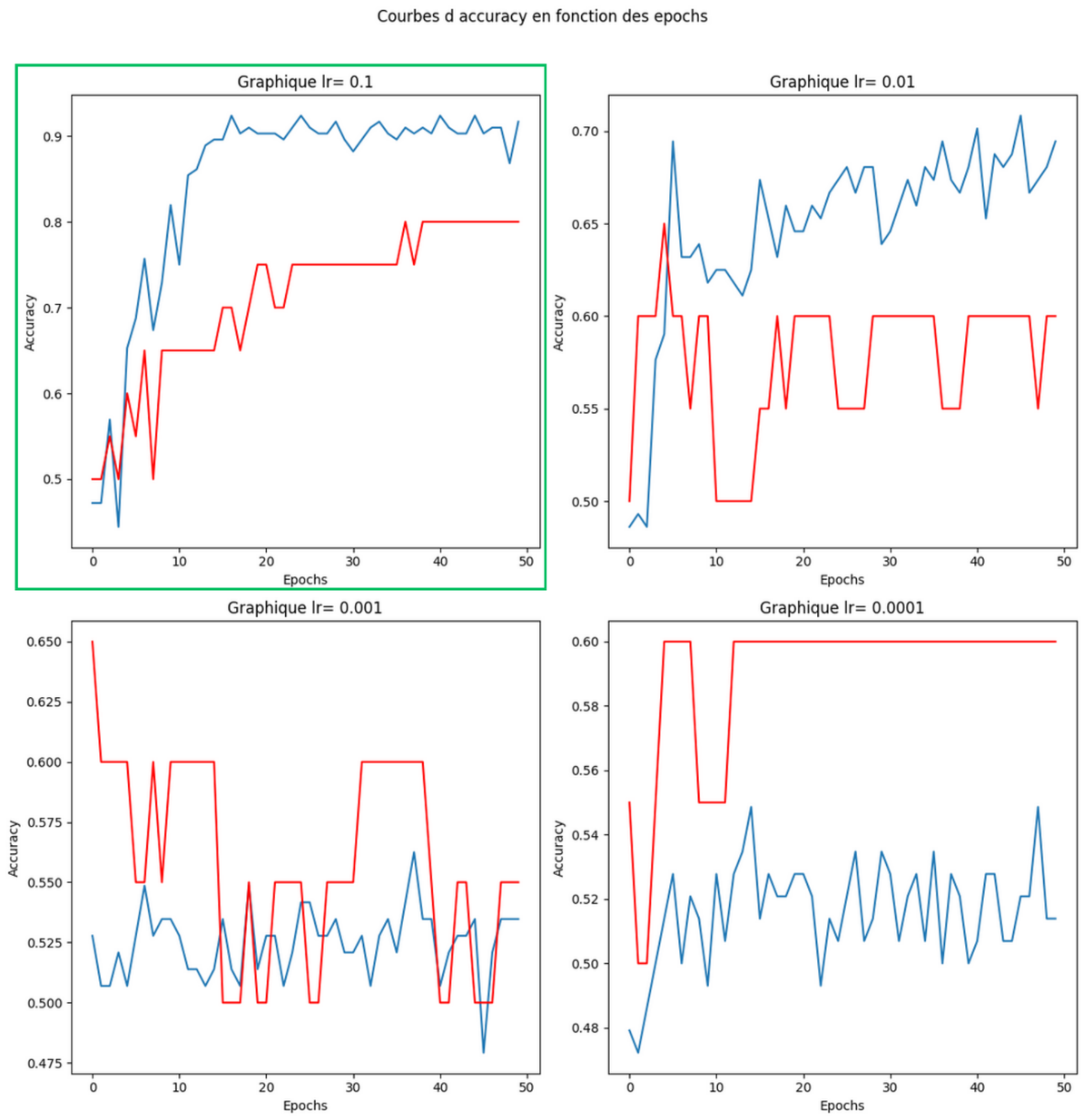
F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

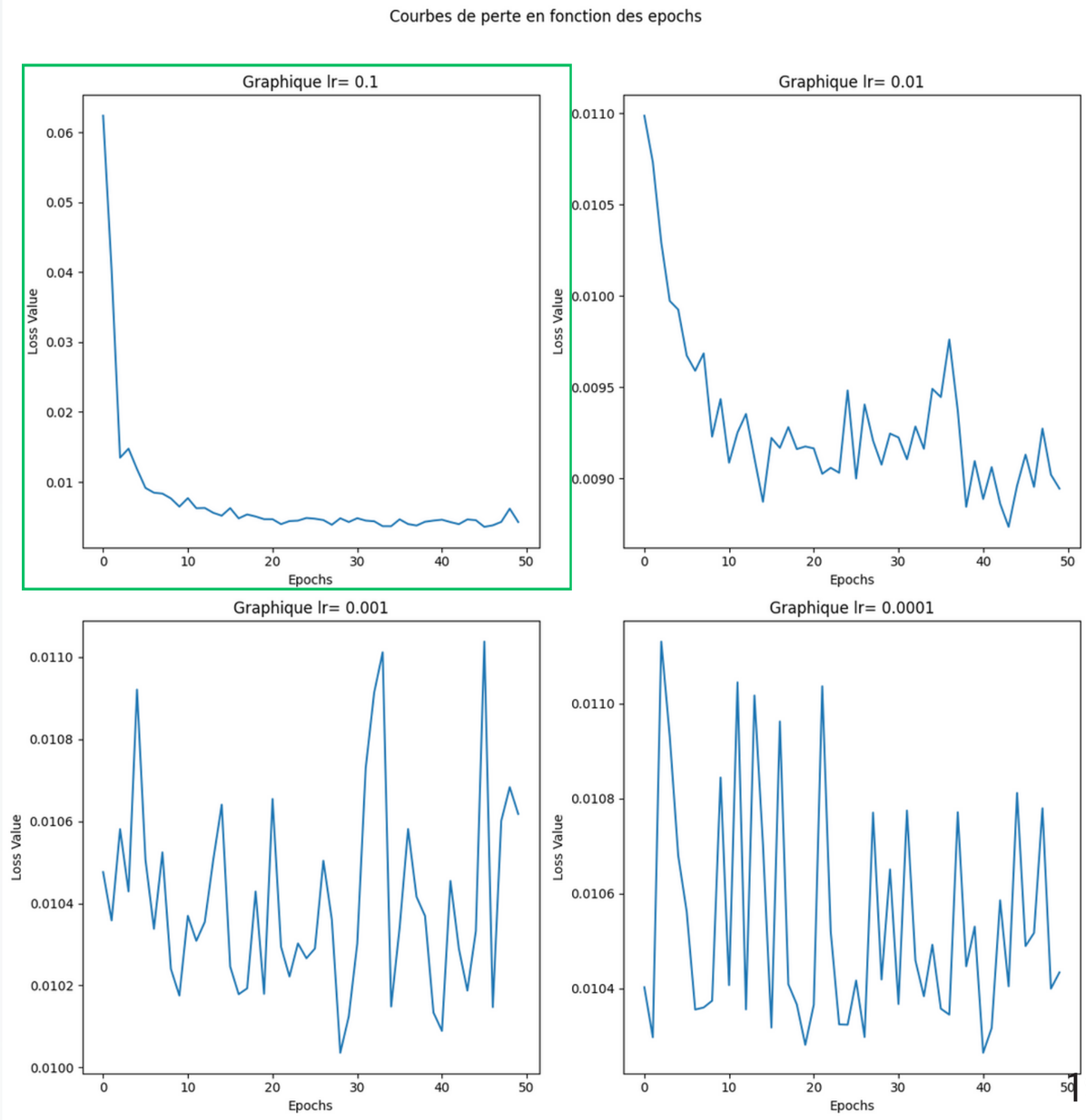
Accuracy=
TP+TN+FP+FN
TP+TN

PFA=
FN/total

V) RESULTS: BINARY YTB CLASSIFIER

Accuracy & Loss





V) RESULTS: BINARY YTB CLASSIFIER

Confusion Matrices

	Prédit Positif	Prédit Négatif
réel Positif	10	4
réel Négatif	0	6

lr = 0,1

	Prédit Positif	Prédit Négatif
réel Positif	8	6
réel Négatif	2	4

lr = 0,01

	Prédit Positif	Prédit Négatif
réel Positif	6	5
réel Négatif	4	5

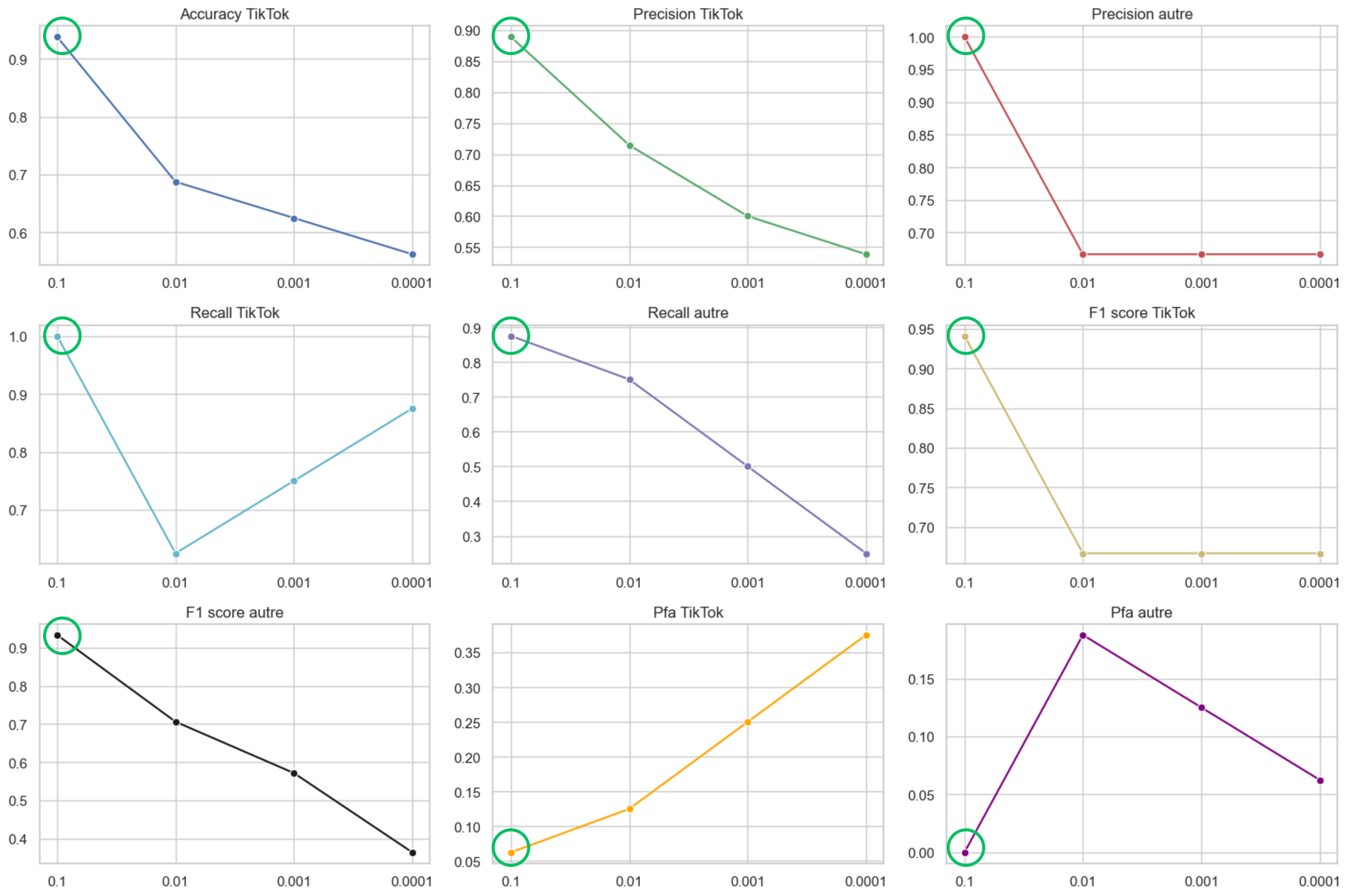
lr = 0,001

	Prédit Positif	Prédit Négatif
réel Positif	6	4
réel Négatif	4	6

lr = 0,0001

V) RESULTS: BINARY TIKTOK CLASSIFIER

Metrics



Recall = TP / (TP + FN)
(TP = True Positives, FN = False Negatives)

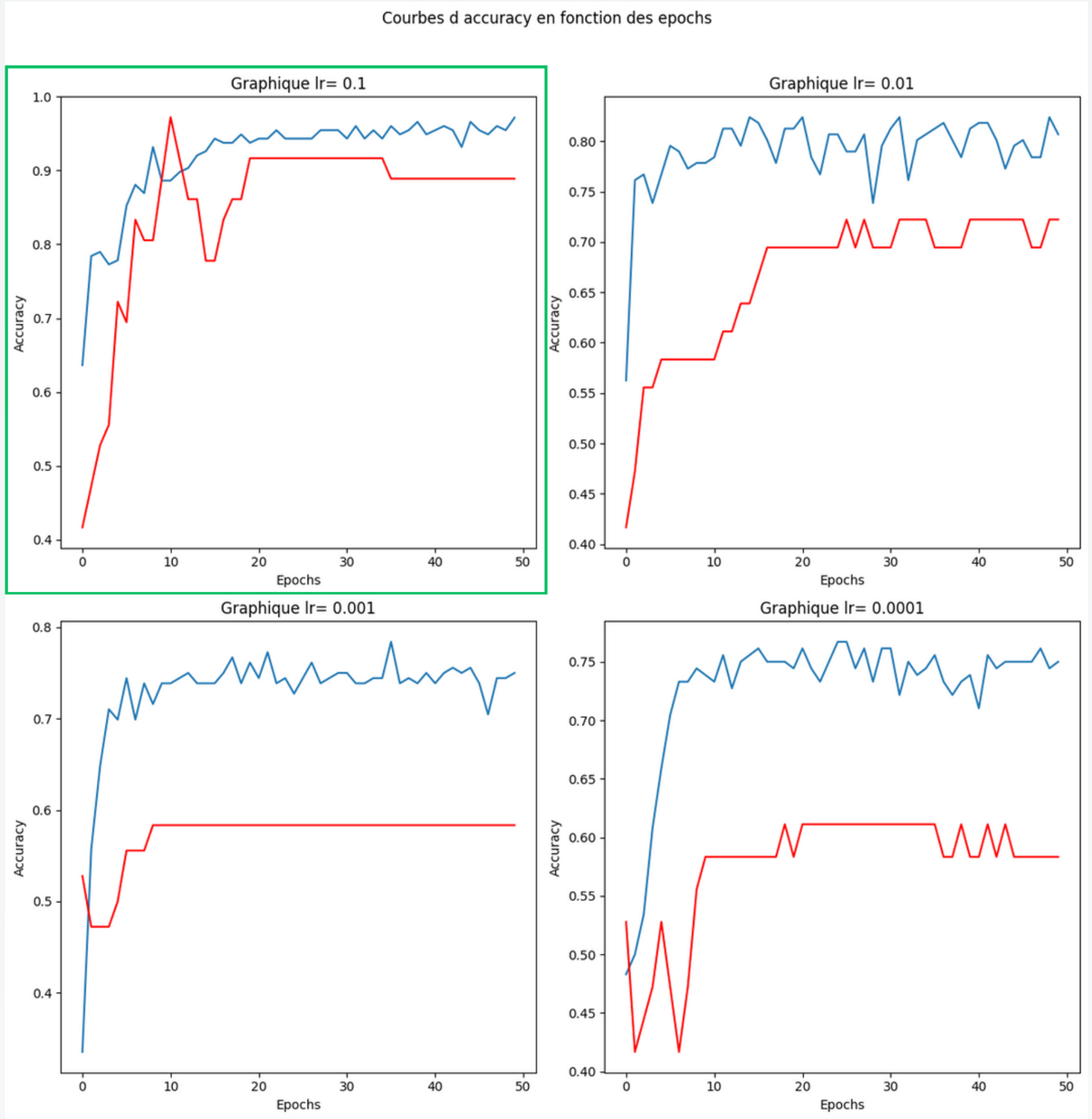
Precision = TP / (TP + FP)
(FP = False Positives)

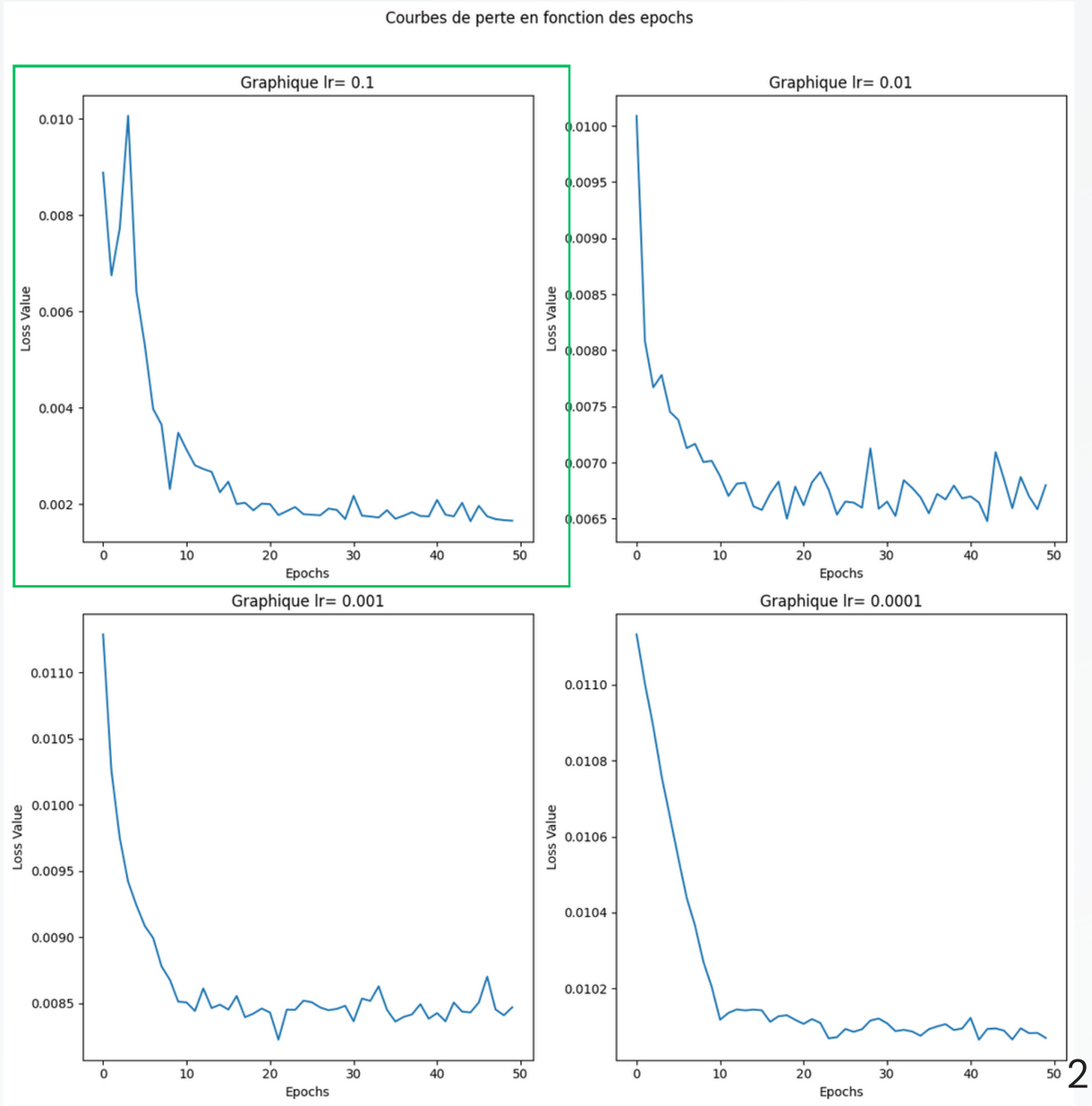
F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

Accuracy=
TP+TN+FP+FN
TP+TN

PFA=
FN/total

V) RESULTS: BINARY VIDMIZER CLASSIFIER Accuracy & Loss





V) RESULTS: BINARY VIDMIZER CLASSIFIER

Confusion Matrices

	Prédit Positif	Prédit Négatif
réel Positif	18	4
réel Négatif	0	14

lr = 0,1

	Prédit Positif	Prédit Négatif
réel Positif	15	7
réel Négatif	3	11

lr = 0,01

	Prédit Positif	Prédit Négatif
réel Positif	10	7
réel Négatif	8	11

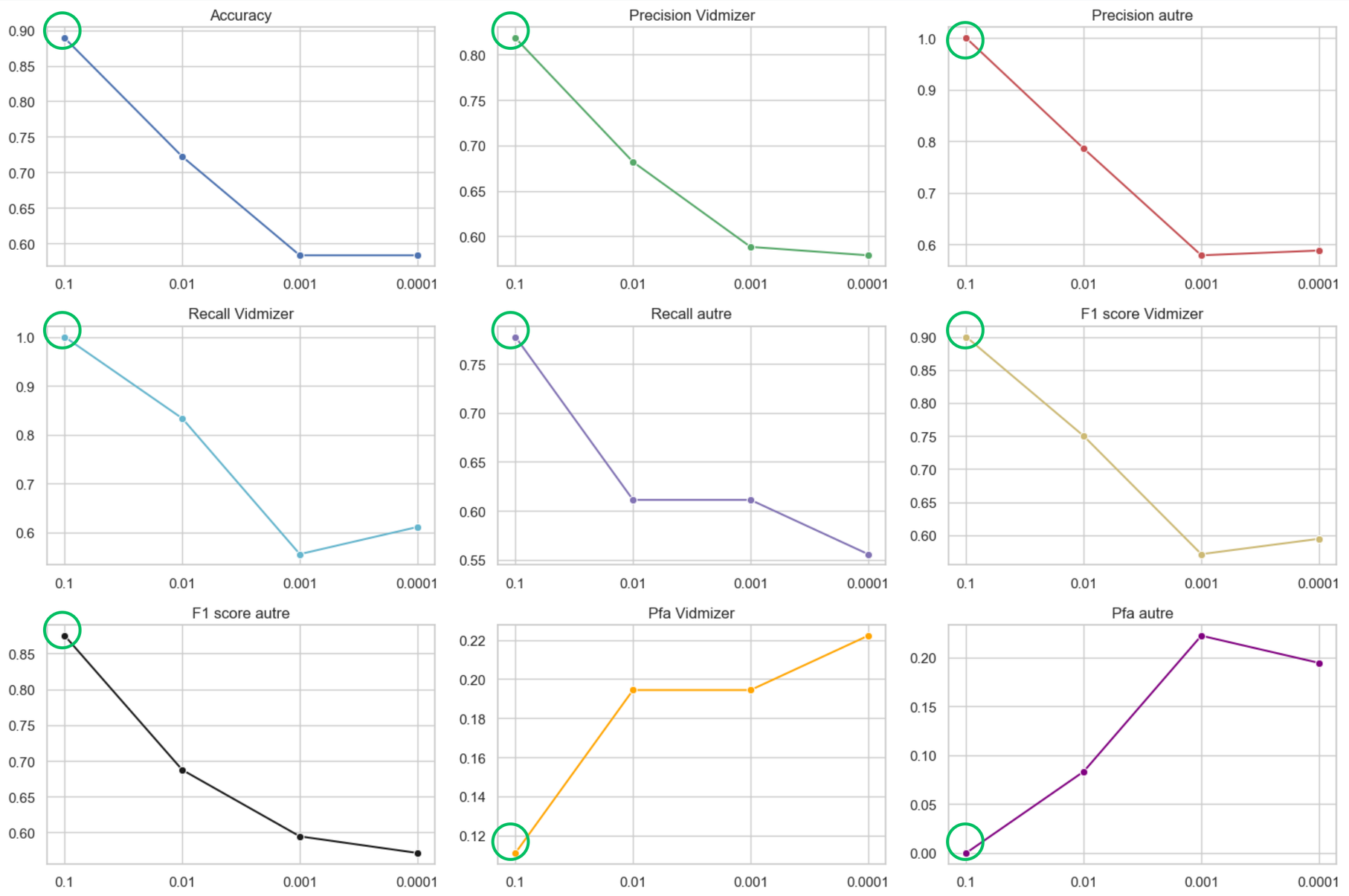
lr = 0,001

	Prédit Positif	Prédit Négatif
réel Positif	11	8
réel Négatif	7	10

lr = 0,0001

V) RESULTS: BINARY YTB CLASSIFIER

Metrics



Recall = TP / (TP + FN)
(TP = True Positives, FN = False Negatives)

Precision = TP / (TP + FP)
(FP = False Positives)

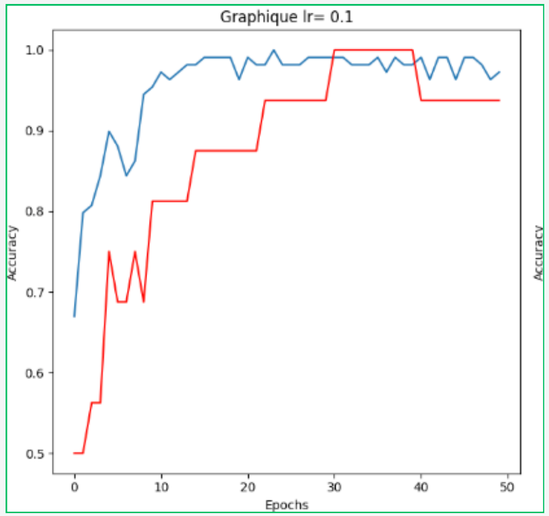
F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

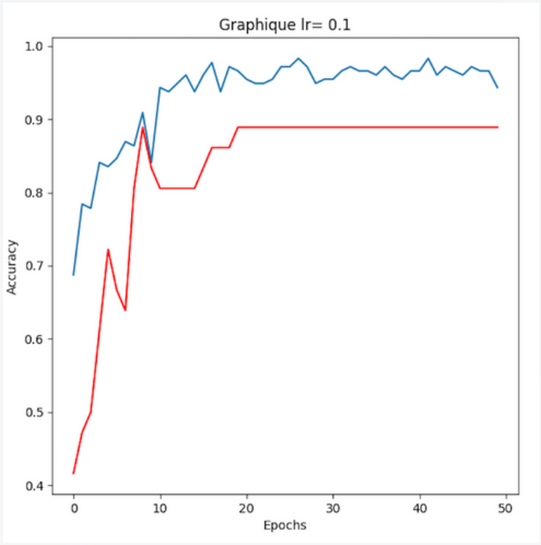
Accuracy=
TP+TN/ FP+FN
TP+TN

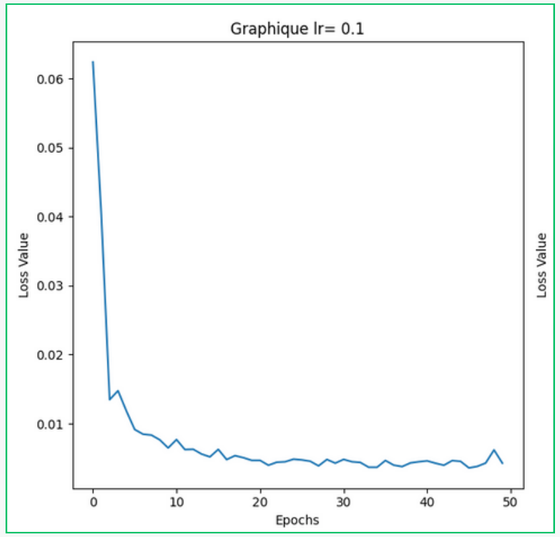
PFA=
FN/total

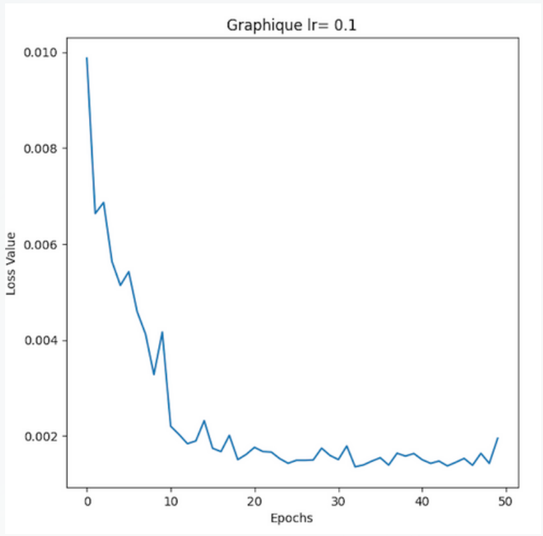
V) RESULTS: BINARY VIDMIZER CLASSIFIER

Normalized or not









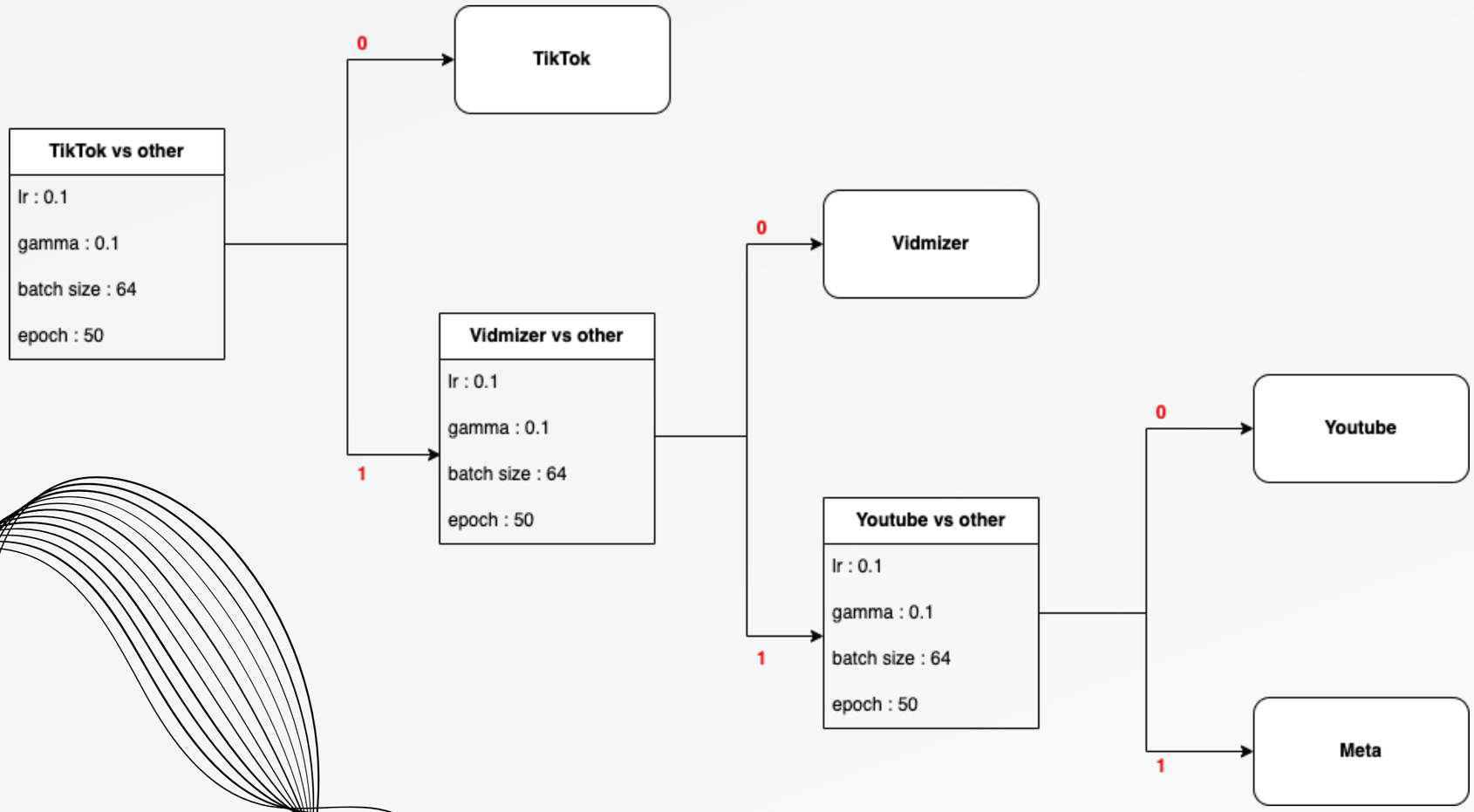
Normalized

Not normalized

V) RESULTS: BINARY CLASSIFIER SELECTION

	Vidmizer	Tiktok	Youtube
LR	0,1	0,1	0,1
Accuracy	0.89	0.93	0.80
Loss	0.0017	0.0008	0.004

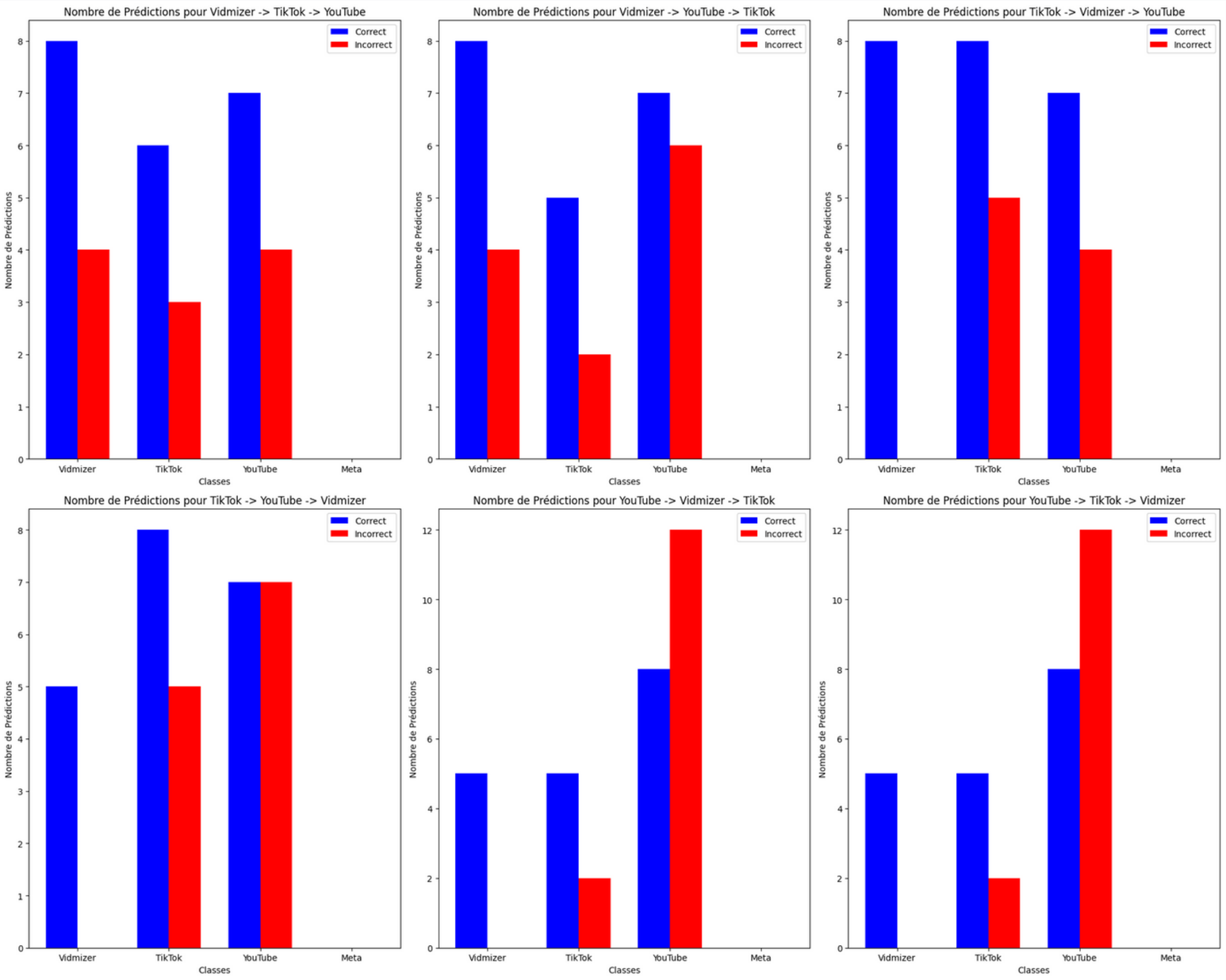
V) RESULTS: CASCADE SCHEME



4 Classes :

- TikTok
- YouTube
- Vidmizer
- Meta (other)

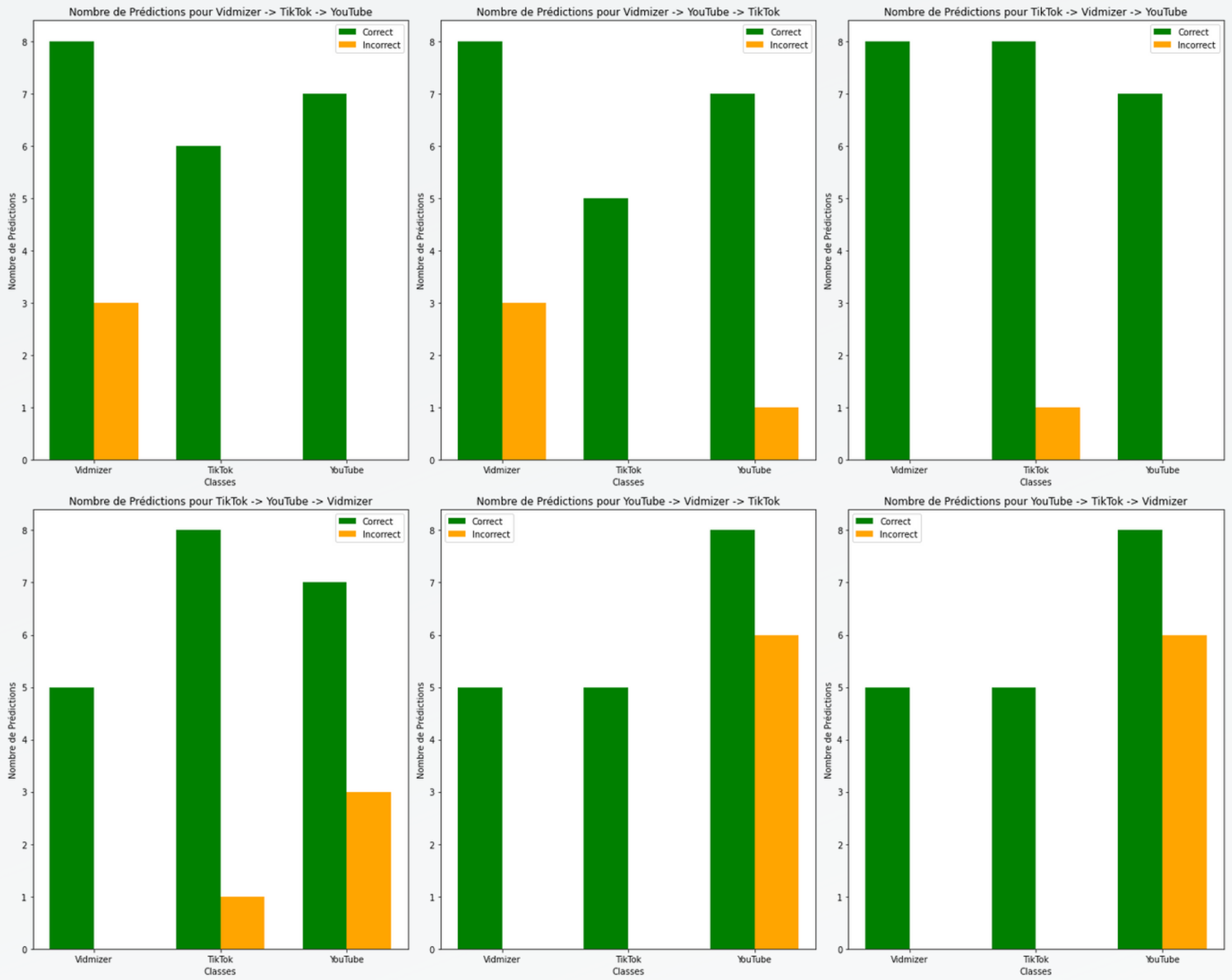
V) RESULTS: CASCADES



With Meta

- The cascade classifiers show better results when arranged in a decreasing enchainment in terms of binary accuracy.
- Conversely, the inverse arrangement also yields remarkable outcomes.
- Moreover, it is noteworthy that the Meta class is never "predicted".
- Accuracy:
  With Meta = 71 %

V) RESULTS: CASCADES

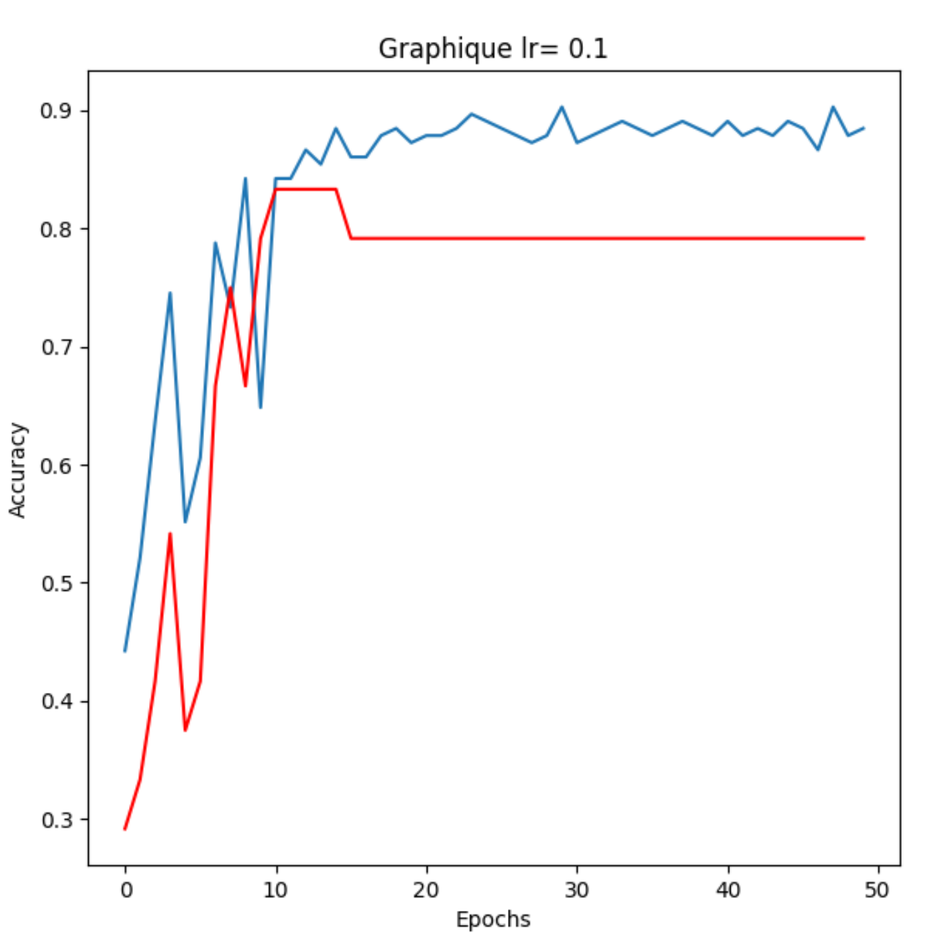


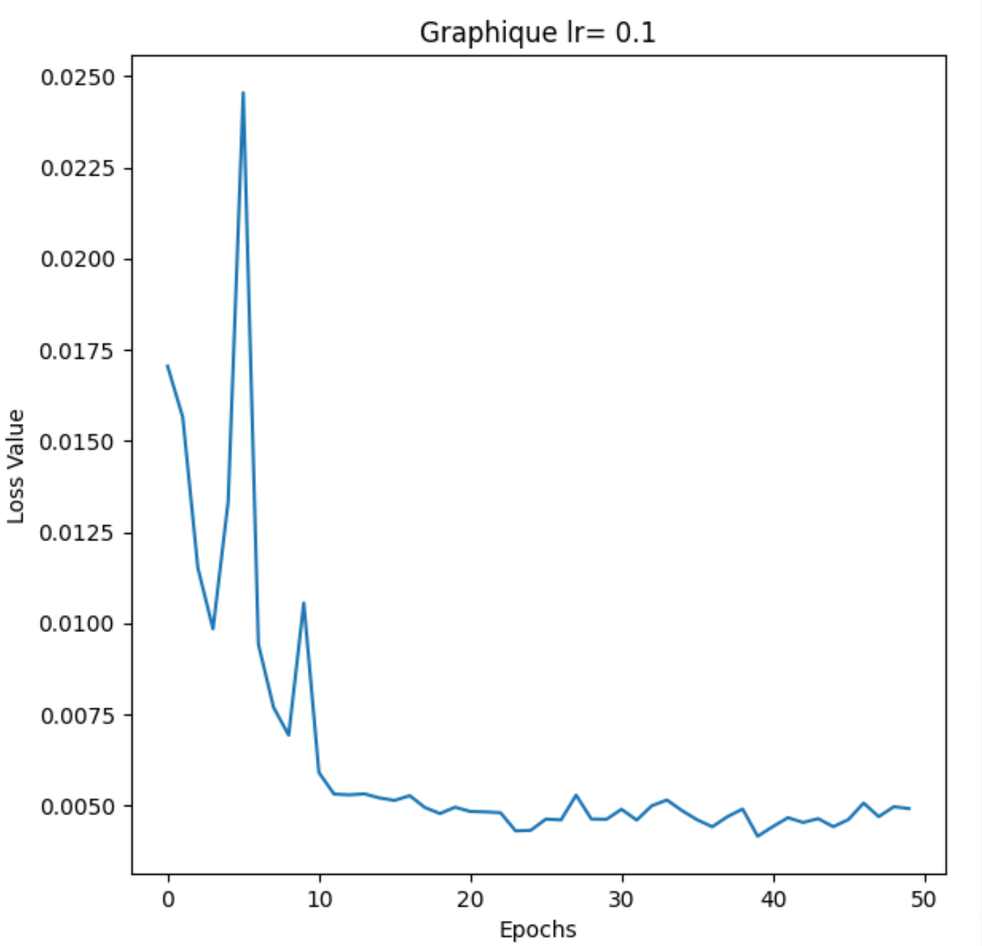
Without Meta

- The cascade classifiers show better results when arranged in a decreasing enchainment in terms of binary accuracy.
- Conversely, the inverse arrangement also yields remarkable outcomes.
- Moreover, it is noteworthy that the Meta class is never "predicted.
- Accuracy:
  Without Meta = 97 %
  With Meta = 71 %
- At this level of compression, the network seems to be able to identify the differences between certain patterns of compression.

V) RESULTS: MULTICLASS - TEST

Accuracy and Loss



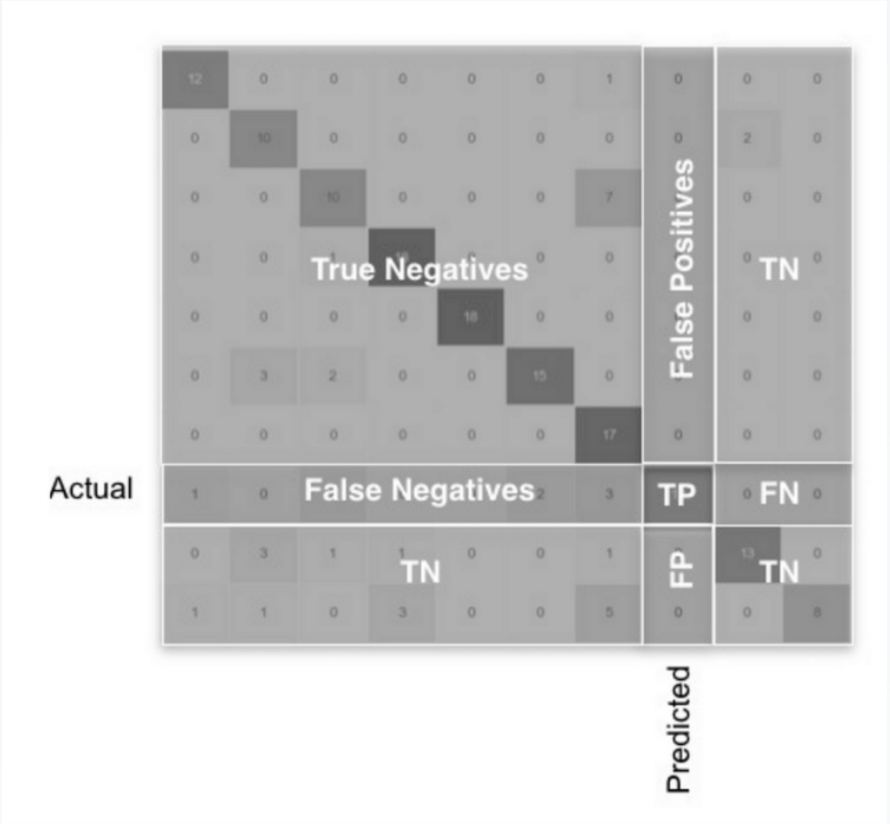


Accuracy = 88%

V) RESULTS: MULTICLASS - TEST

Accuracy and Loss

Confusion Matrix	tiktok predicted	vidmizer predicted	youtube predicted
tiktok actual	6	0	0
vidmizer actual	0	5	0
youtube actual	2	3	8



Accuracy for class: tiktok is 75.0 %
Accuracy for class: vidmizer is 75.0 %
Accuracy for class: youtube is 100.0 %

CONCLUSION

IN A NUT SHELL

- The compressed domain is better to classify videos based on encoding

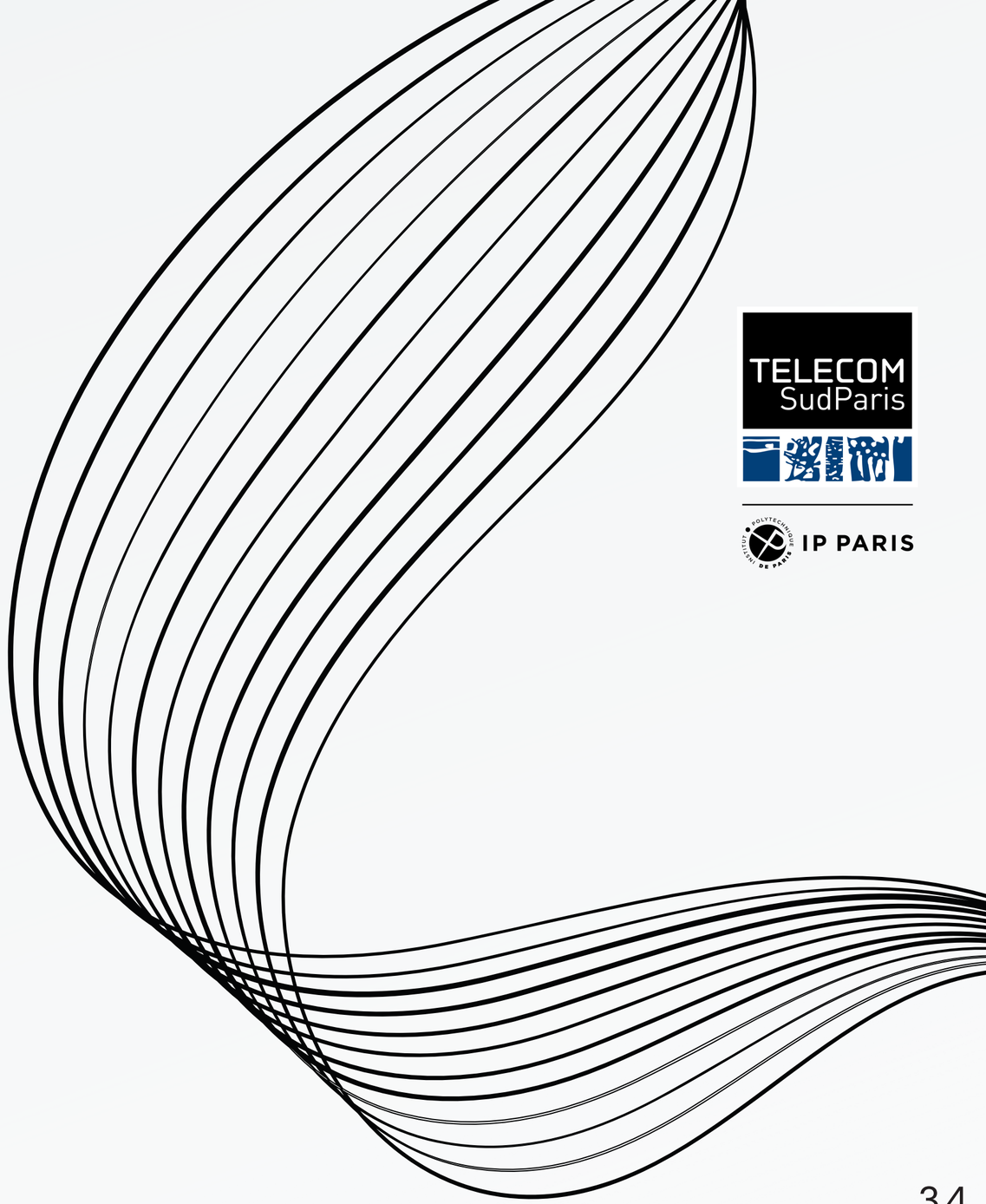
REMARKS

- We have quite good results for our small dataset
- The cascade is better than the multi-class classifier but does not generalize well

ROOM FOR IMPROVMENT

- Try it on a bigger dataset with more classes
- We did not manage to discover the coding parameters of the different platforms

THANK'S FOR WATCHING



TELECOM SudParis

IP PARIS

ESPECIALLY TO MIHAI MITREA, MOHAMED ALLOOUCHE, CARL DE SOUSA AND VIDMIZER