Projet IA 321
Synthetic data augmentation for classification of time series

Asma KHALIL
Florine LEFER
Paul POIRMEUR
André WERNECK
Lorenzo PERRIER

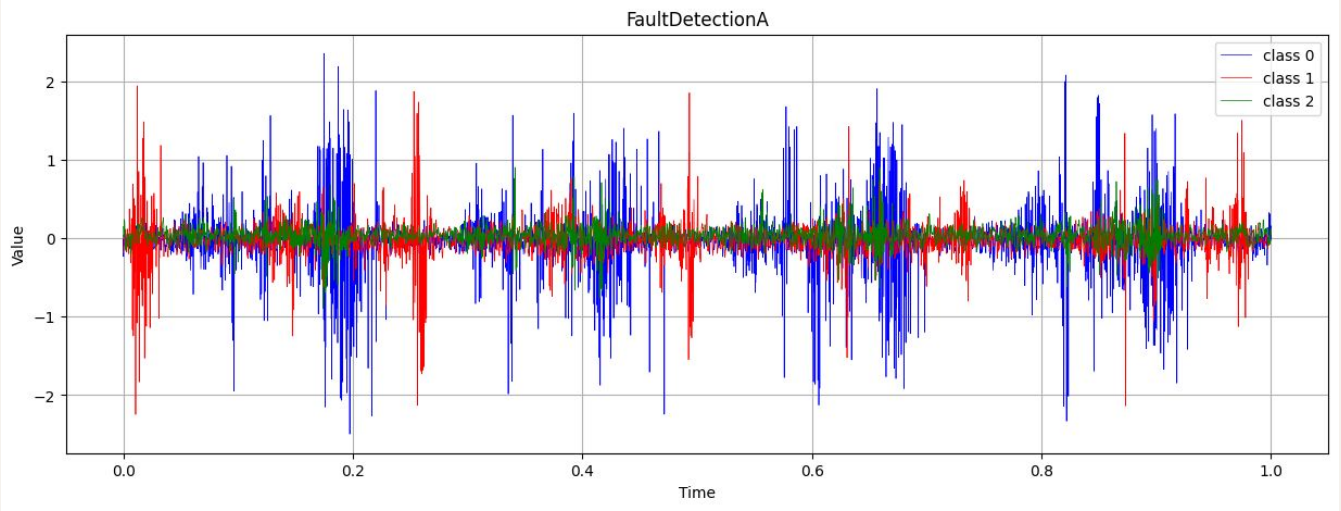
28 mars 2024

Sommaire

I. Le sujet : Synthetic Data Augmentation for Classification of time series

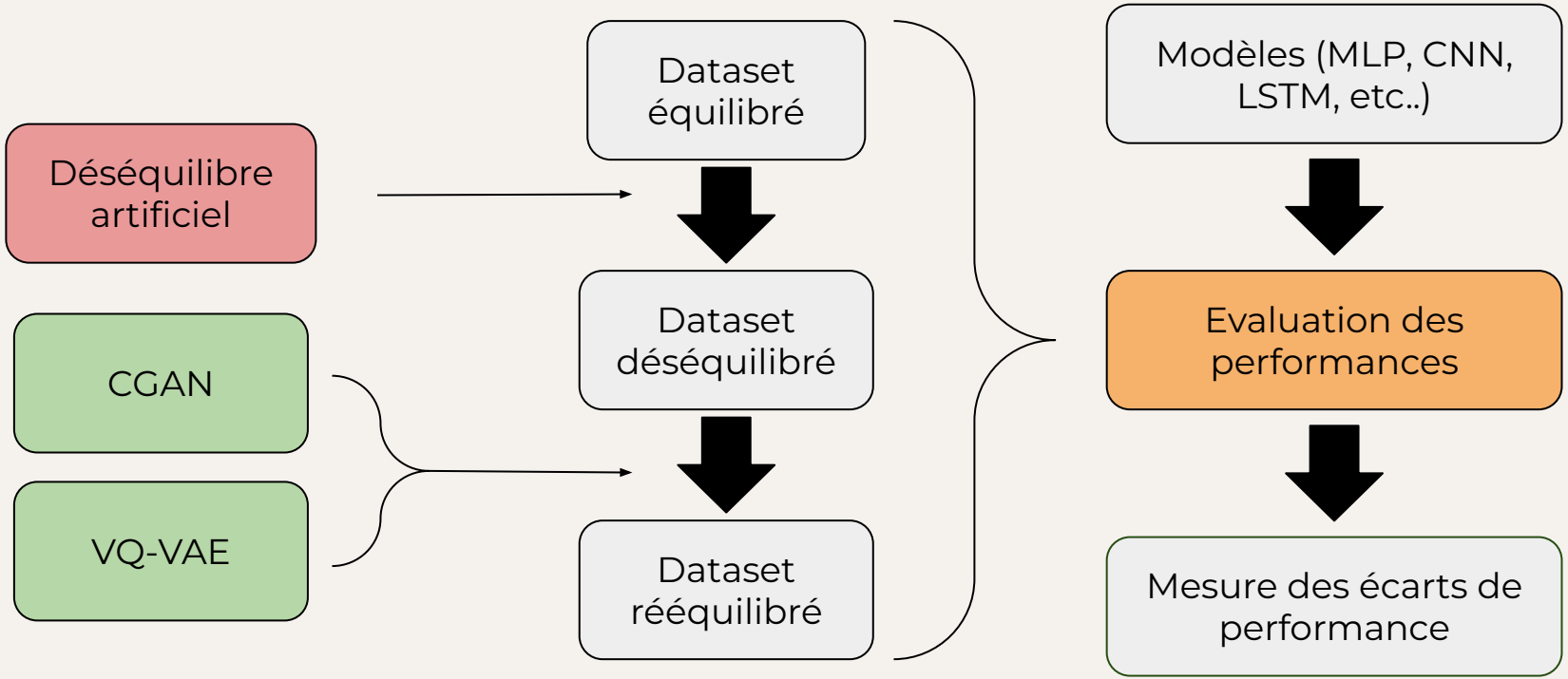
I.2 Présentation du problème

Comment rééquilibrer un dataset de séries temporelles dans lequel une classe est sous représentée grâce à des méthodes de génération?



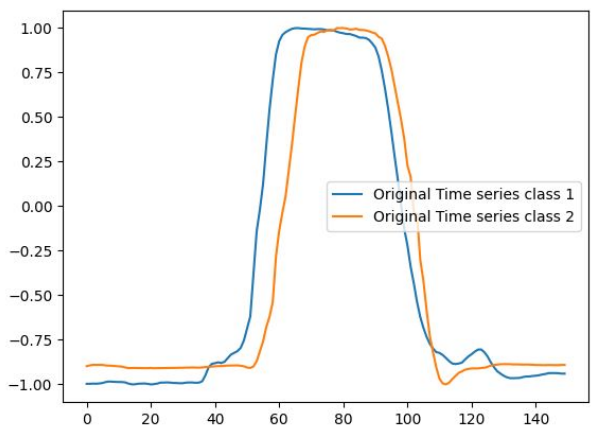
II. Approche méthodologique

II.1 Approche méthodologique



II.2 Dataset Gunpoint

- Taille entraînement : 160
- Taille test : 40
- Longueur d'une série : 150
- 2 classes :
  - armé
  - non armé
- Mouvements, uniaxial



II.2 Dataset FaultDetectionA

- Taille entraînement : 10 912
- Taille test : 2 728
- Longueur d’une série : 5 120
- 3 classes :
  - non endommagées
  - endommagées à l'intérieur
  - endommagées à l'extérieur
- Capteurs (SENSOR), uniaxial

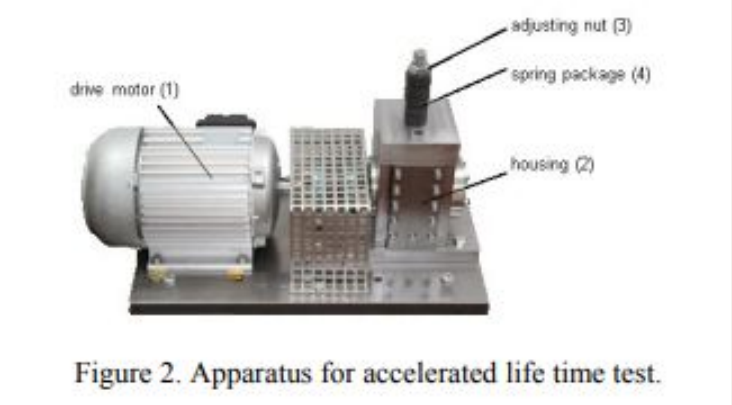


Figure 2. Apparatus for accelerated life time test.

II.3 Le déséquilibre de dataset

Réduction manuelle des données de la classe 0 (pour “GUNPOINT") : application d’un facteur de réduction de 0.1, classe 1 inchangée.

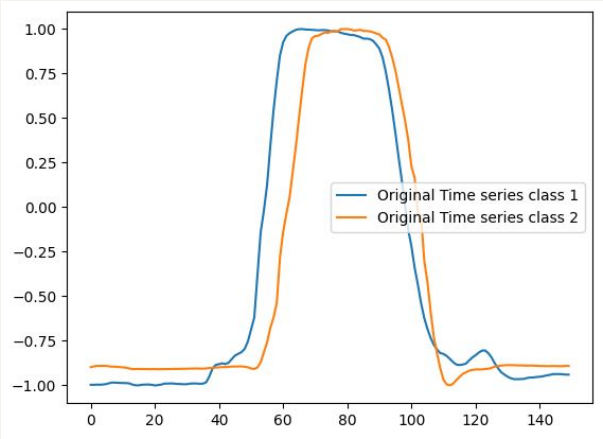
Approches de rééquilibrage :

- utilisation de techniques comme SMOTE (Synthetic Minority Over-sampling Technique)
- génération de données via des modèles génératifs (VAE, CGAN)

Problème : la génération ne fonctionne pas suffisamment bien sur les séries temporelles

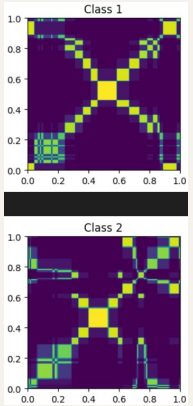
⟹ transformation des séries en **images**

II.4 Des séries temporelles aux images

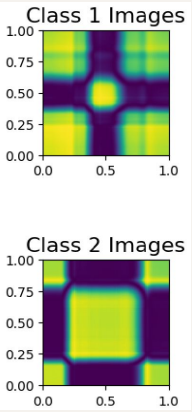


La transformation en image facilite la tâche de génération et permet d'utiliser les outils de traitement d'image

Markov Transition Field →



Gramian Angular Field →



III. Modèles de génération

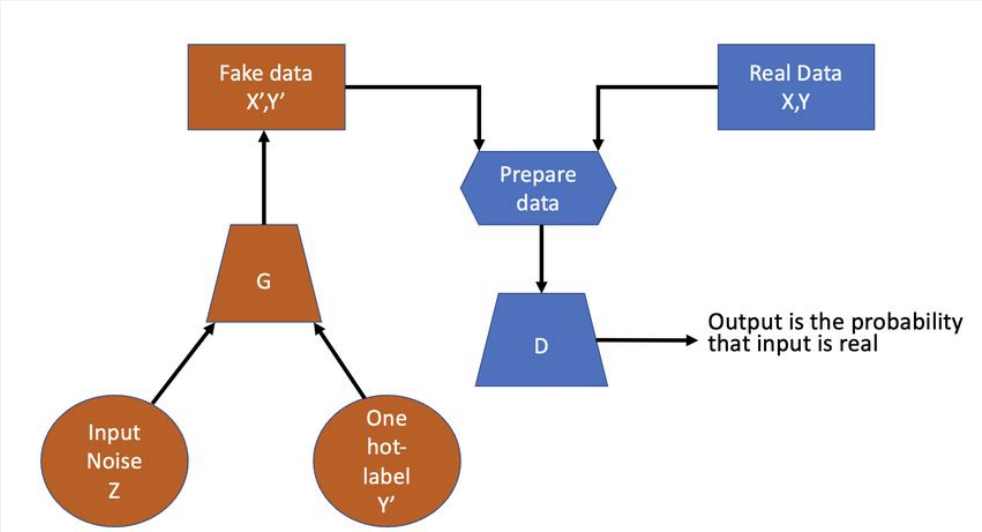
IV.1 Conditional GAN

Avantages :

- Permet d'apprendre un conditionnement pour la génération
- Une architecture flexible

Inconvénients :

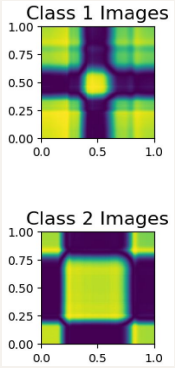
- Difficile à entraîner
- Quand l'arrêter ?

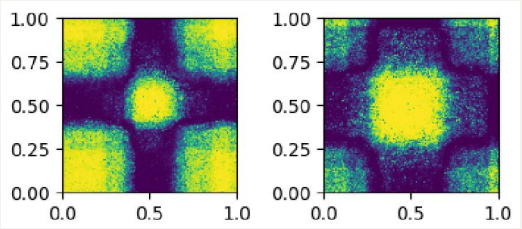


Conditional GAN : Gunpoint

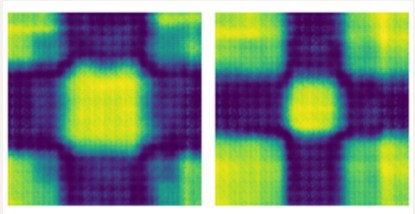
➔ 3 architectures :

Images d'origines :

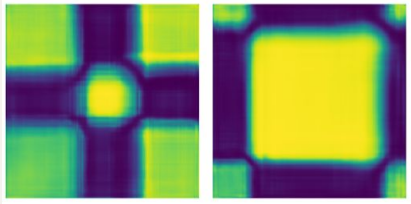




MLP



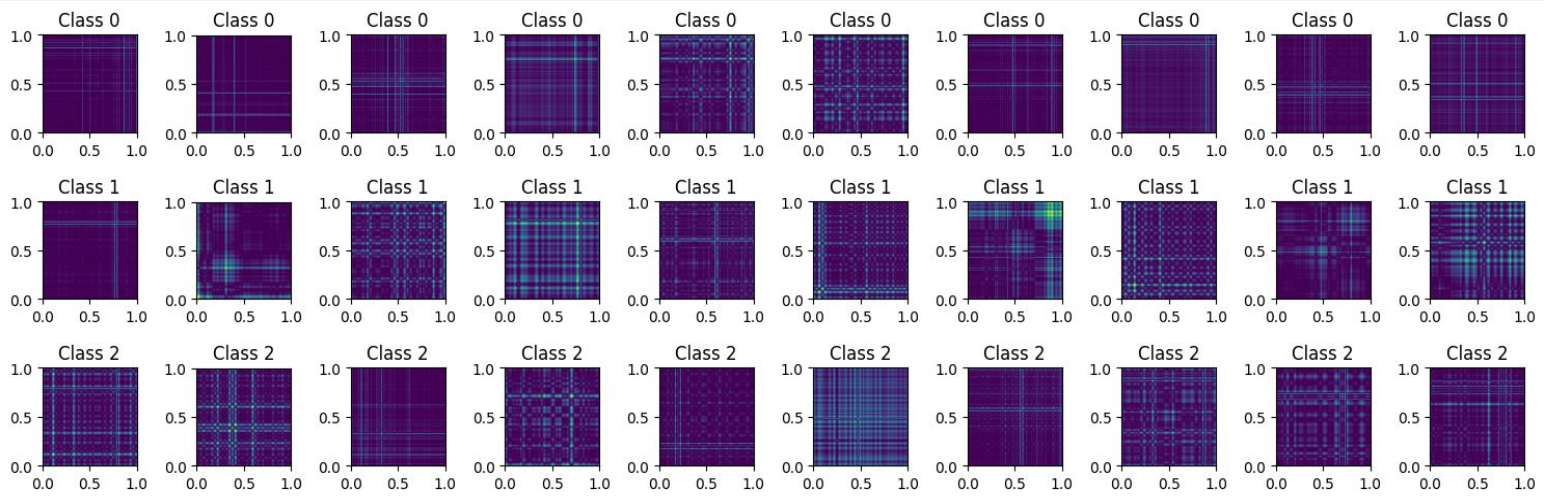
ConvTranspose



Upsample + conv

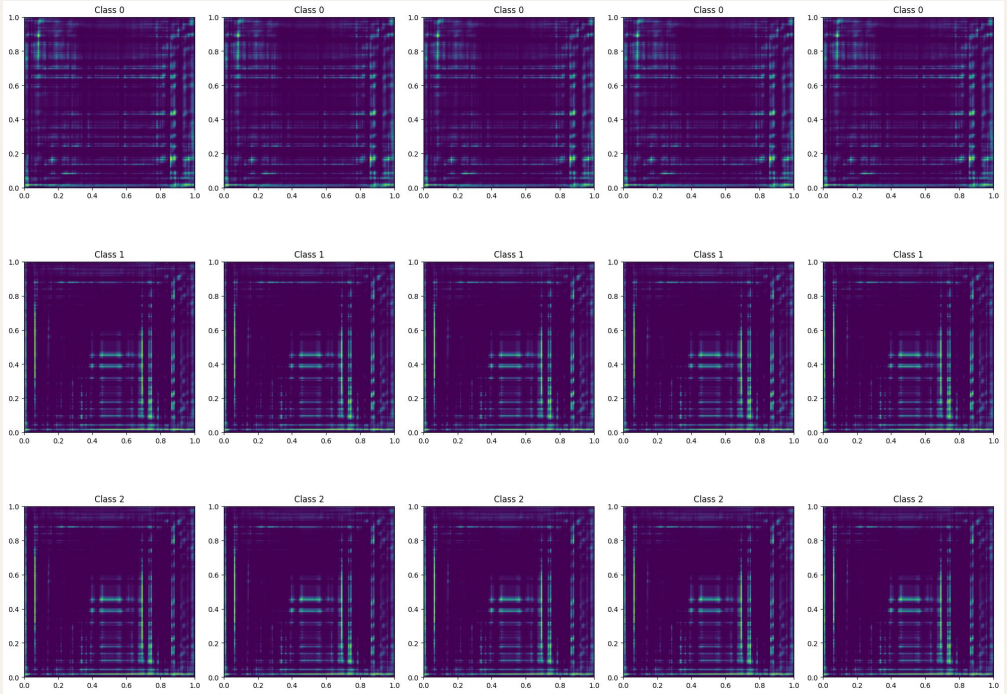
CGAN : Fault Detection A

Utilisation des 150 premières valeurs

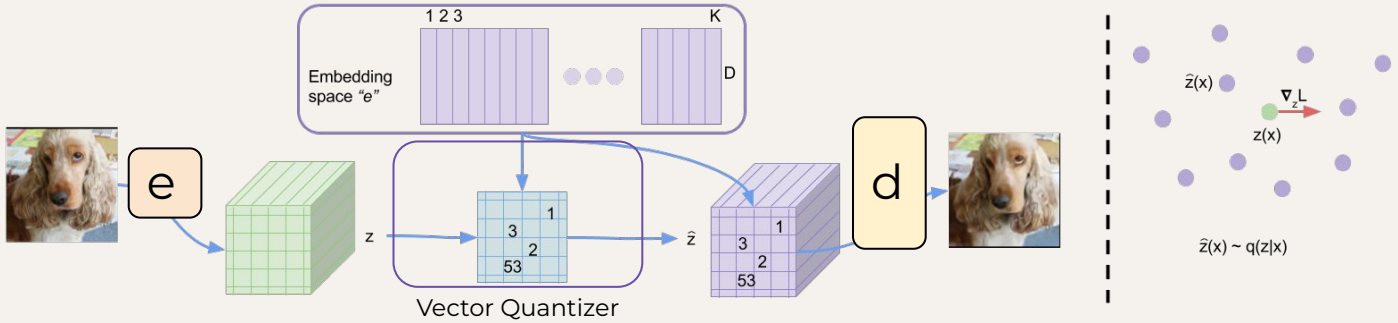


CGAN : Fault Detection A

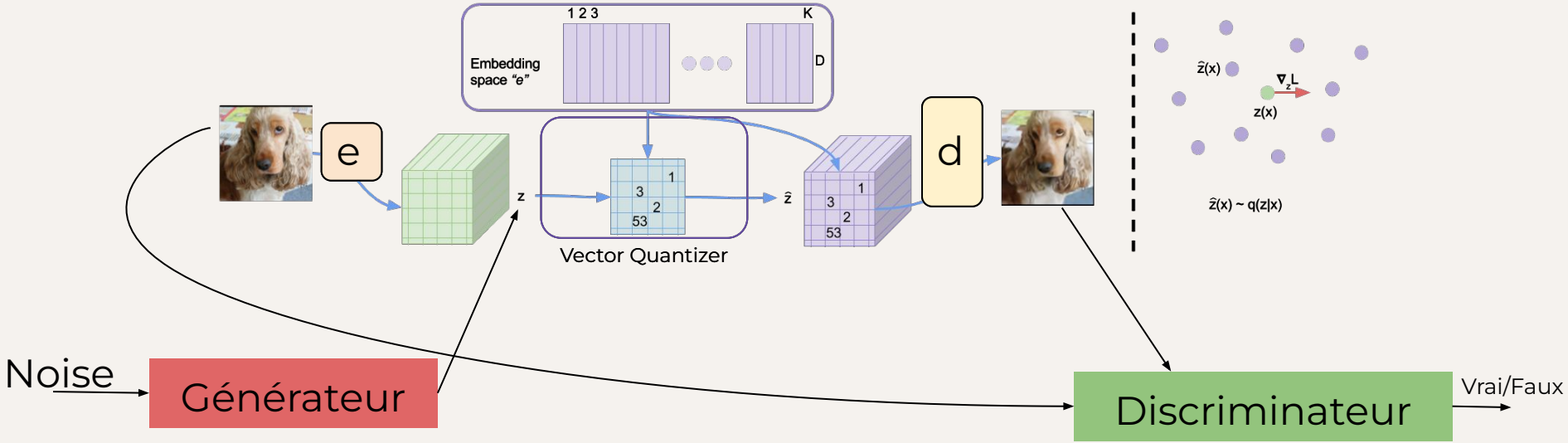
Exemple de génération pendant l’entraînement



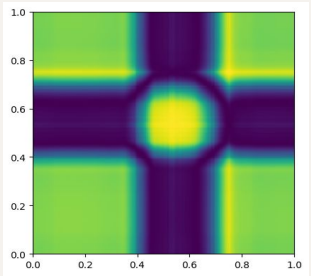
IV.2 VQ-VAE


1 2 3
K
Embedding space "e"
D
e
z
3
1
2
53
Vector Quantizer
ẑ
d
1
3
2
53
ẑ(x)
∇zL
z(x)
ẑ(x) ~ q(z|x)

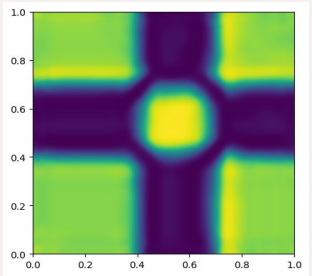
IV.3 CGAN pour la génération avec VQ-VAE



IV.3 CGAN pour la génération avec VQ-VAE

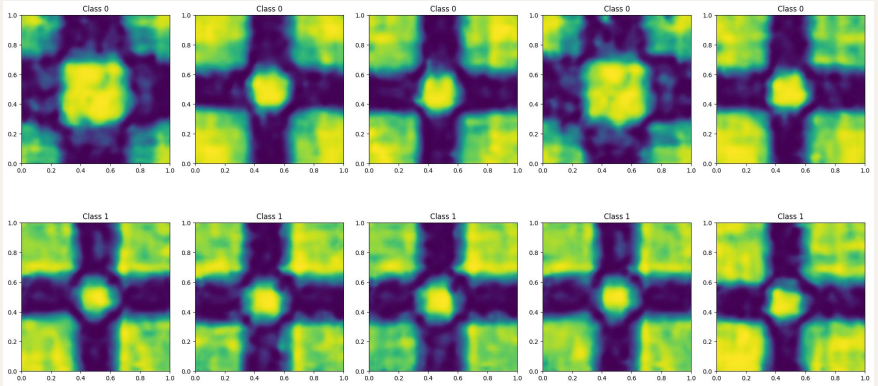


Original



Reconstruit

Génération avec CGAN + décoder du VQ VAE :

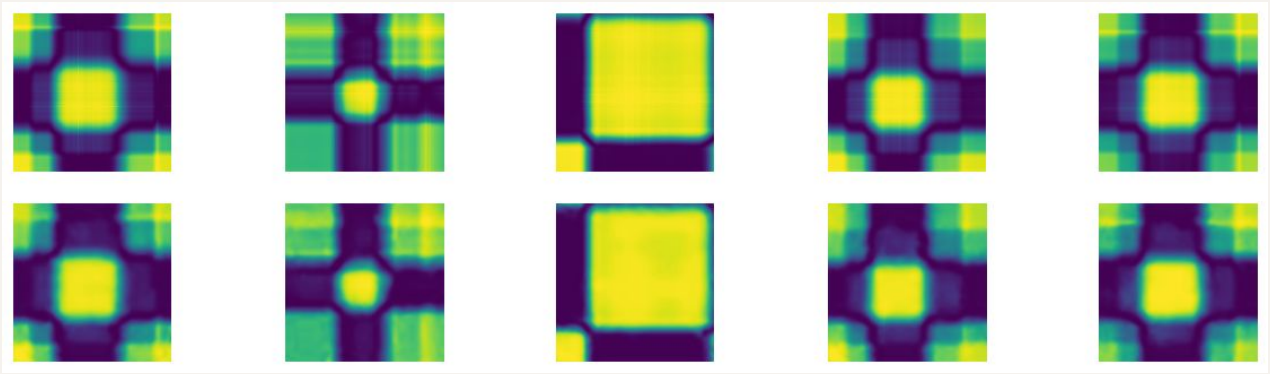


IV.3 CGAN pour la génération avec VQ-VAE

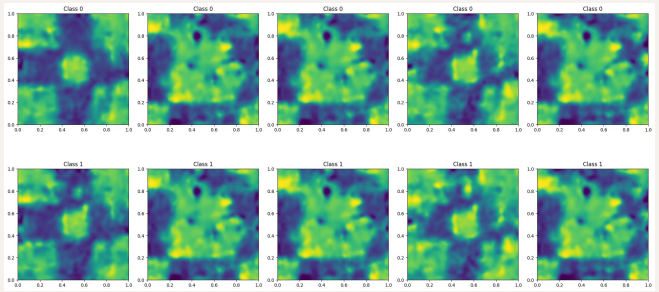
Entraînement du VQ VAE pour une meilleure reconstruction :

Original

Reconstruit



Le GAN n’arrive pas à utiliser le codebook :



IV.3 CGAN pour la génération avec VQ-VAE

Entraînement plus compliqué, gradient vanishing ?

```python
def forward(self, noise, labels):
    # Concatenate label embedding and image to produce input
    emb = self.label_emb(labels)
    emb = emb.view(len(emb),-1)
    gen_input = torch.cat((emb, noise), -1)
    img = self.model(gen_input)

    img = img.view(-1,16,18,18)
    img,_ = self.vector_quantization(img)
    img = self.decoder(img)

    return img
```

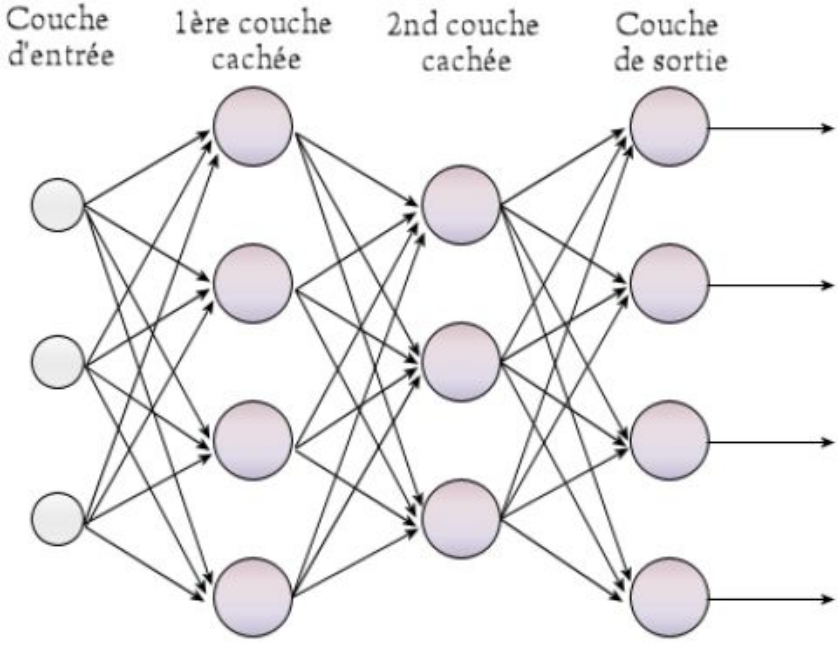
IV. Modèles de classification

III.1 MLP

Paramètres retenus:

- une à deux couches:
  - taille (32) pour time series
  - taille (64,32) pour images

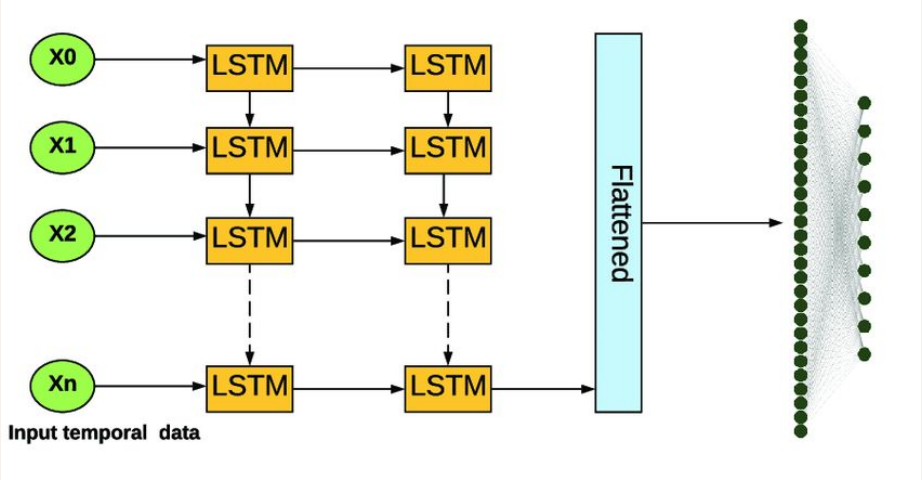
- Nombre de paramètres:
  - 4898 pour (32)
  - 1442210 pour (64,32)



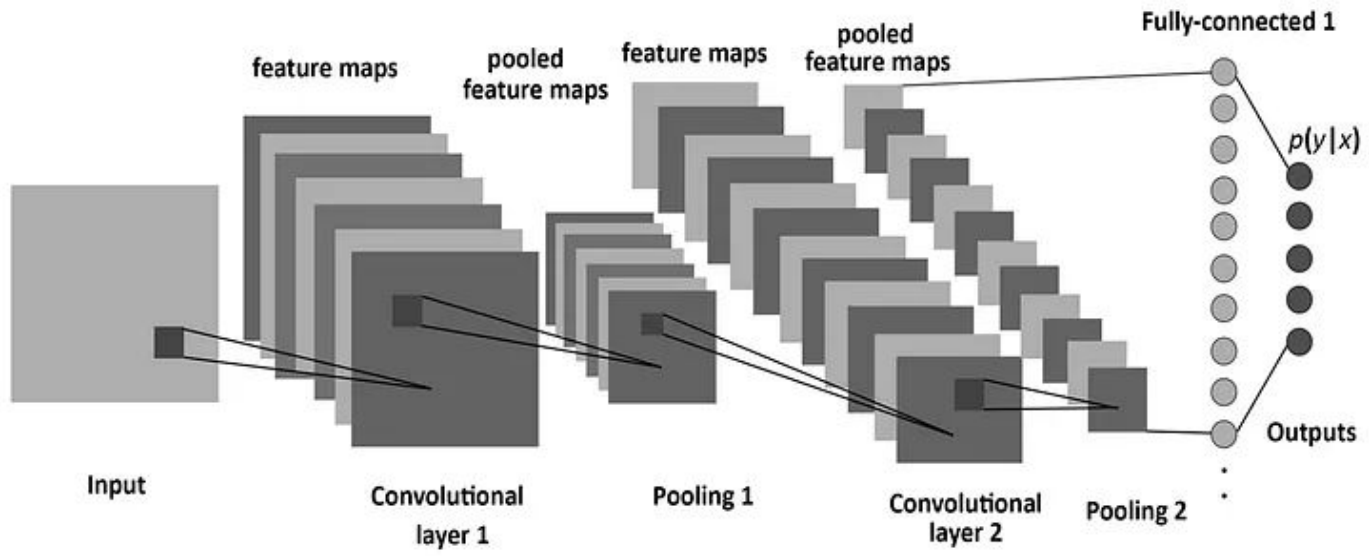
Classification : LSTM

Setup :

- Nous avons testé plusieurs architectures.
- L”earlystop”, le "checkpoint" et validation croisée ont été utilisés.
- L’architecture la plus performante : deux couches LSTM suivi de Dropout à 0.2.
- 211203 paramètres.



Classification : CNN

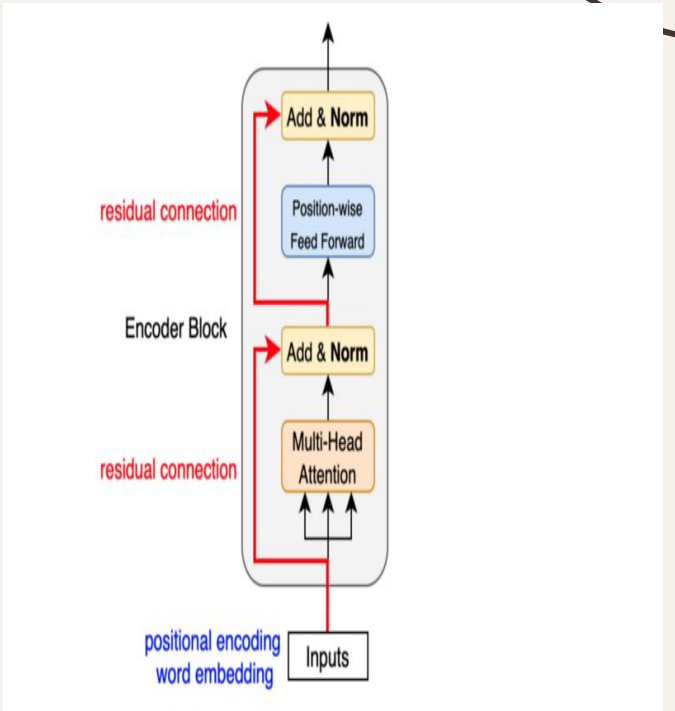


Transformers

```python
import keras
from keras import layers

def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Normalization and Attention
    x = layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs  # Skip connection

    # Feed Forward Part
    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation='relu')(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res  # Skip connection
```



ViT (Vision Transformer)

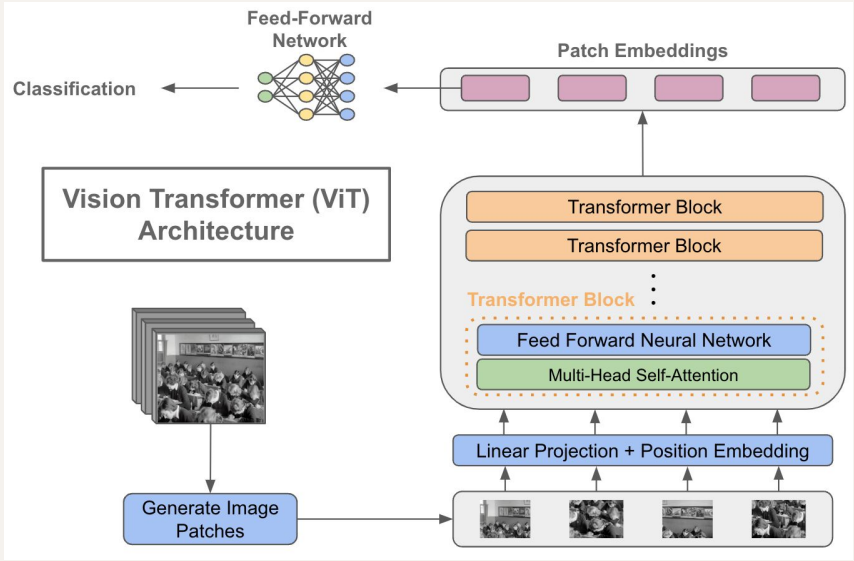
```python
def build_vit_model(image_size, patch_size, num_patches, num_layers, d_model, num_heads, mlp_dim, channels=3, num_classes=10):
    inputs = keras.Input(shape=(image_size, image_size, channels))
    # Prétraitement et découpage en patchs + projection
    patches = PatchEncoder(patch_size, num_patches, d_model)(inputs)

    # Ajout de Position Embedding
    sequence = PositionEmbedding(num_patches, d_model)(patches)

    # Transformer Blocks
    for _ in range(num_layers):
        sequence = TransformerBlock(d_model, num_heads, mlp_dim)(sequence)

    # Token de classification ou moyenne pour la classification
    representation = layers.GlobalAveragePooling1D()(sequence)
    # Couche de sortie
    outputs = layers.Dense(num_classes, activation="softmax")(representation)

    return keras.Model(inputs=inputs, outputs=outputs)
```



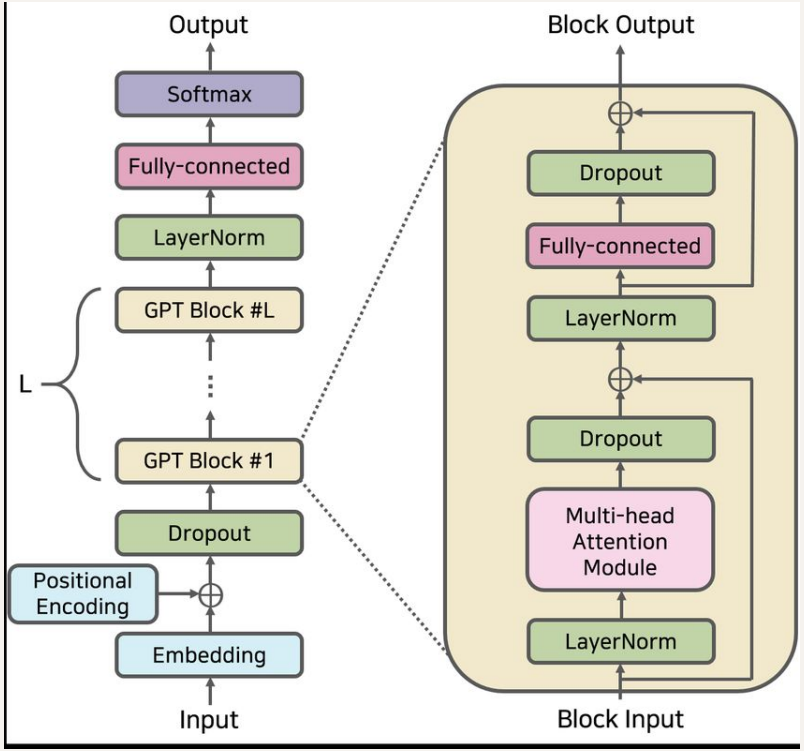
III.2 minGPT sur séries temporelles

Modifications ajoutées:

- Fully connected à la place de la couche embeddings
- Adaptation pour la tâche de classification

Nombres de paramètres:

- 2 690 560



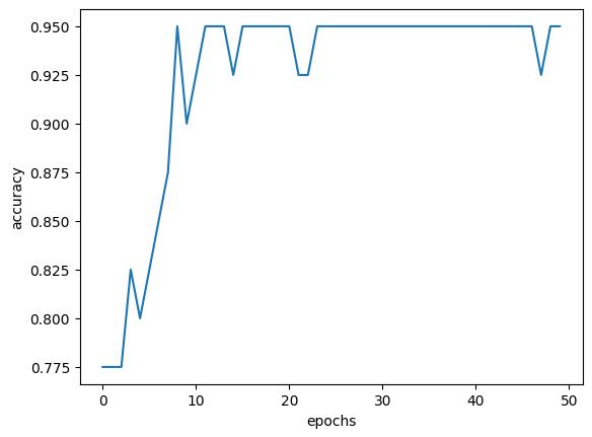
V. Résultats

V.I Classification: comparaison des modèles

	MLP	CNN	LSTM	VIT	minGPT
Dataset initial	0.95	0.925	0.95	0.99	0.90
Une classe réduite à 10%	0.875	0.95	0.5	0.90	0.85
Rééquilibré avec SMOTE	0.95	0.7	0.85	0.90	0.85
Dataset complet en image (GAF)	0.95	0.975	0.975	0.90	x
Une classe réduite à 10% en image	0.90	0.9	0.725	0.85	x
Dataset rééquilibré en image	0.975	0.95	0.975	0.70	x

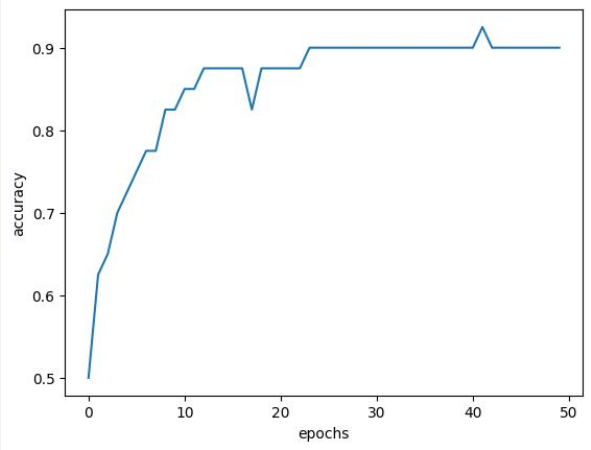
V.2 Apport des données générées

L'évolution de la précision sur le test set pour le MLP, pour trois dataset différents



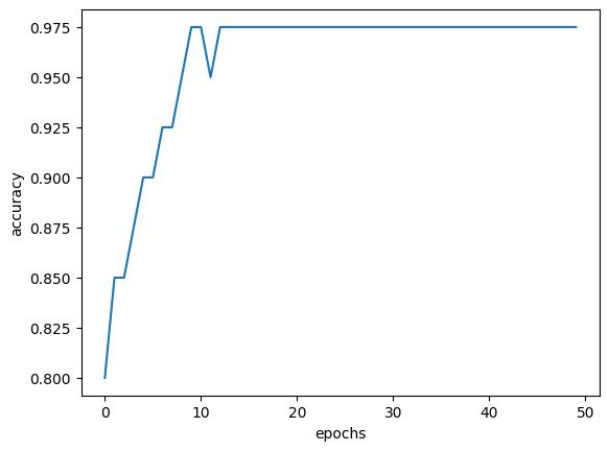
Jeu de donné équilibré

Accuracy: 95%



Jeu de donné déséquilibré : classe 1 à 10%

Accuracy: 90%

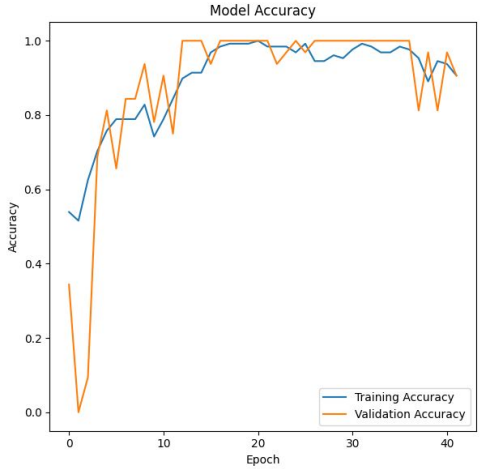


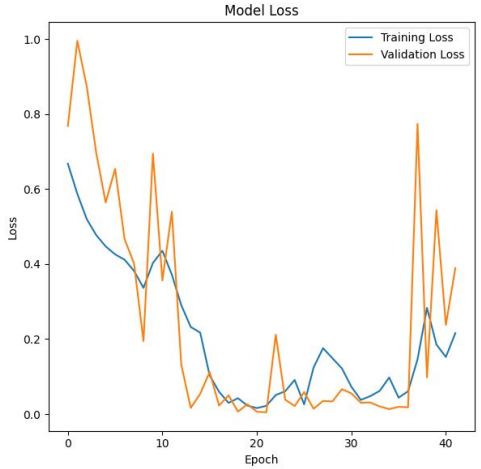
Jeu de donné rééquilibré avec CGAN

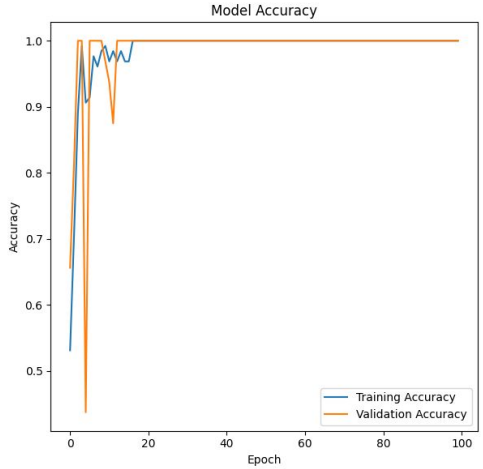
Accuracy: 97.5%

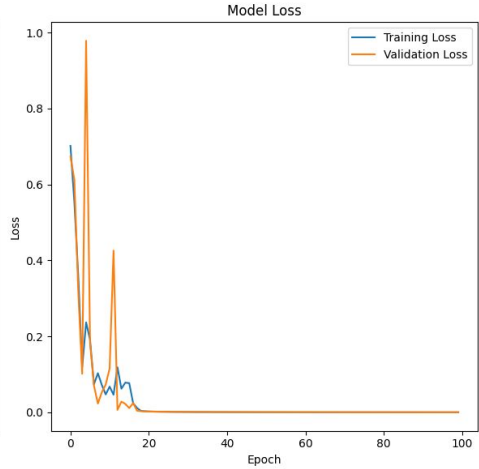
V.II Classification: comparaison des modèles

Comparaison SMOTE/ Génération image









SMOTE

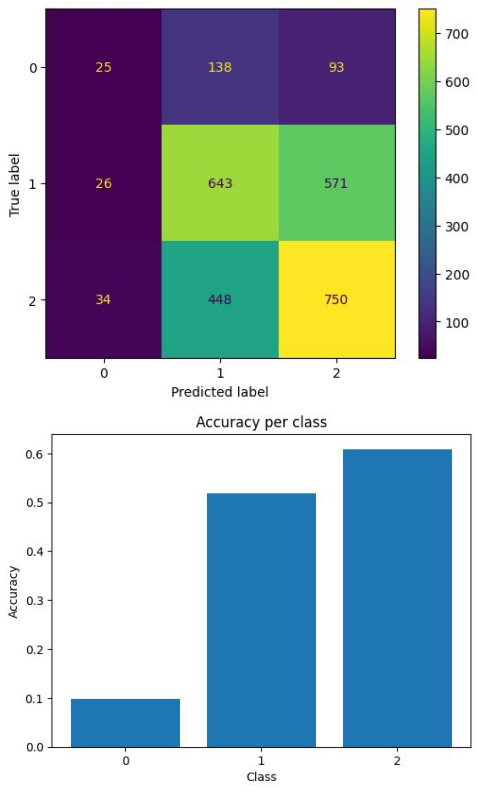
```
Test Accuracy: 85.00%
Errors = 6
```

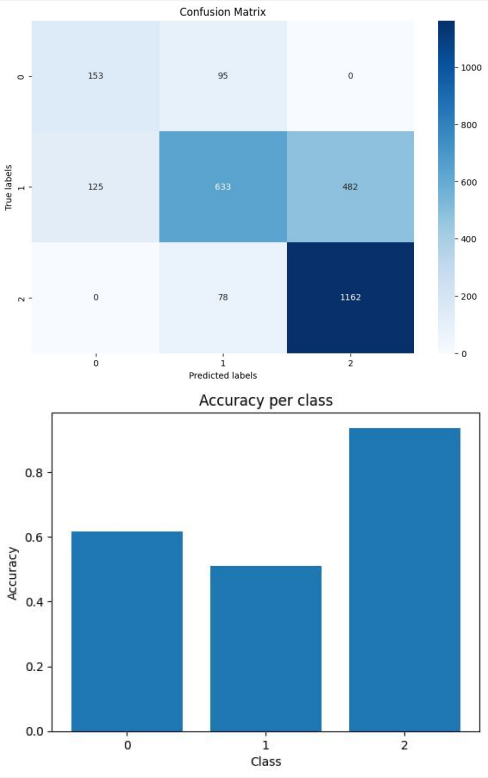
DCGAN

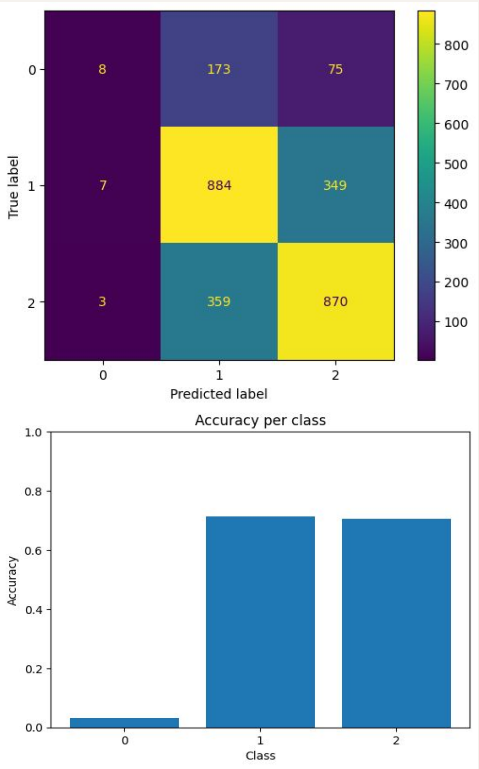
```
Test Accuracy: 97.50%
Errors = 1
```

V.2 Apport des données générées

Comparaison des performances des modèles sur FaultDetectionA







VI. Recherches complémentaires et difficultés

VII. Conclusion

VII. Annexes

VI.1 Taille des échantillons trop élevée

Sur FaultdetectionA, La taille des échantillons étant trop élevé rendait la tâche de génération trop complexe une fois les données transformer en images

VI.2 Transformation des séries temporelles en images:

- La transformation en images a facilité la tâche de génération en permettant l'utilisation d'outils de traitement d'image, une approche innovante face aux difficultés rencontrées avec la génération directe de séries temporelles.

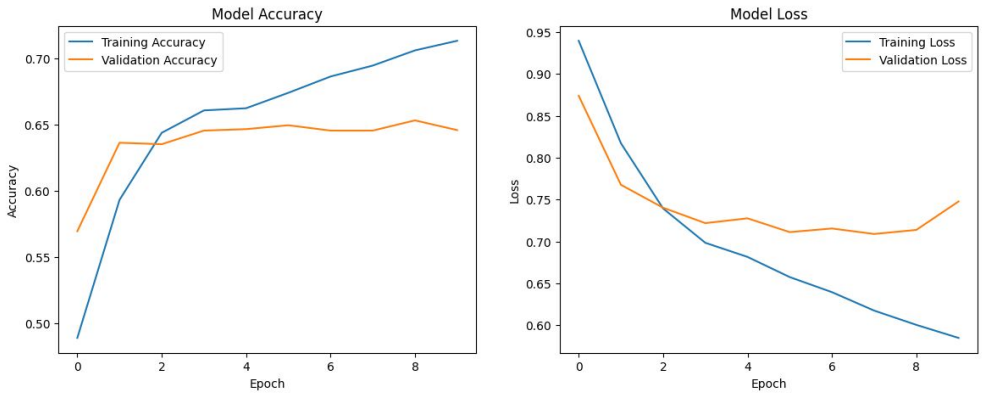
VI.3 Défis de l'apprentissage des GANs:

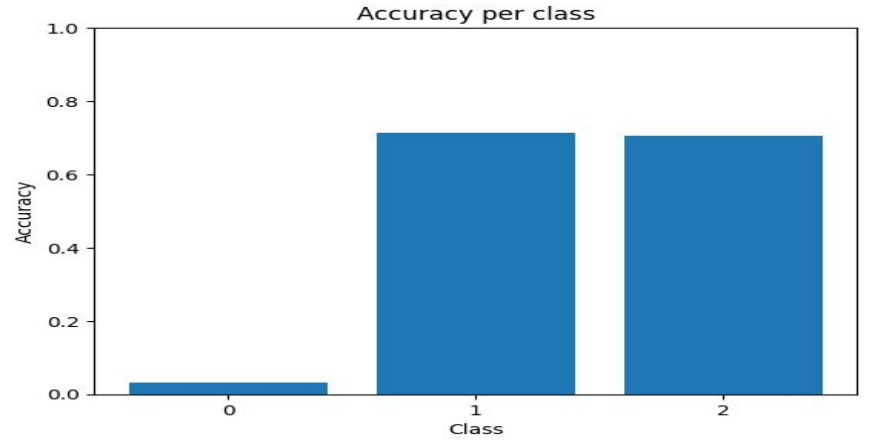
- L'entraînement des GANs, en particulier des CGANs, a présenté des défis significatifs, notamment la détermination du moment optimal pour arrêter l'entraînement et le choix des hyperparamètres et architectures appropriées pour le générateur et le discriminateur.

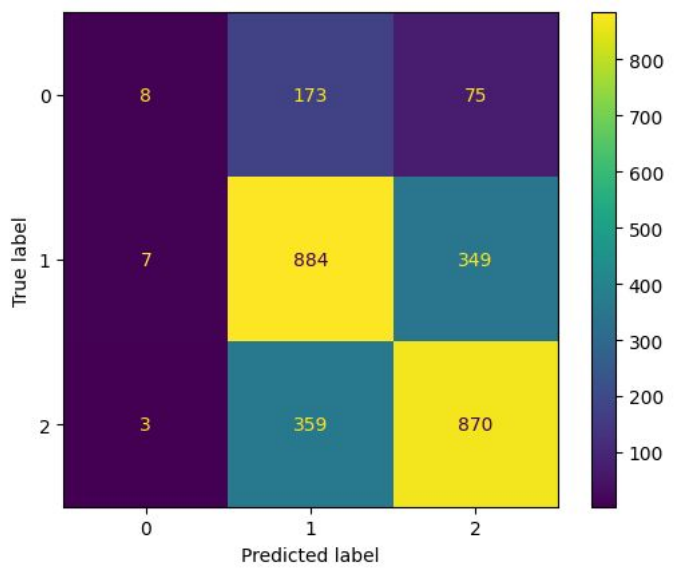
VI.4 Limites de la méthode SMOTE:

- Bien que SMOTE ait été utilisé pour le rééquilibrage, des limitations ont été observées, notamment dans la distinction fine entre des classes très similaires, ce qui souligne l'importance de choisir la méthode de génération adaptée à la spécificité du dataset.

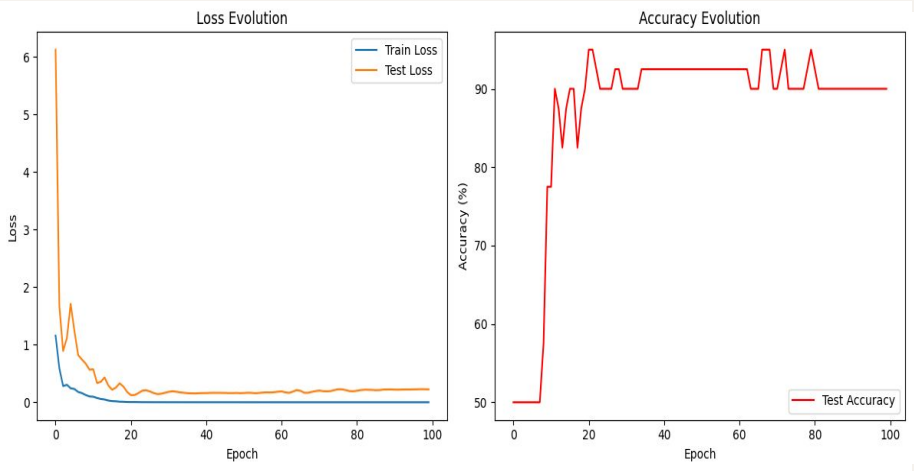
Résultats avec CNN “FaultDetectionA” avec les images



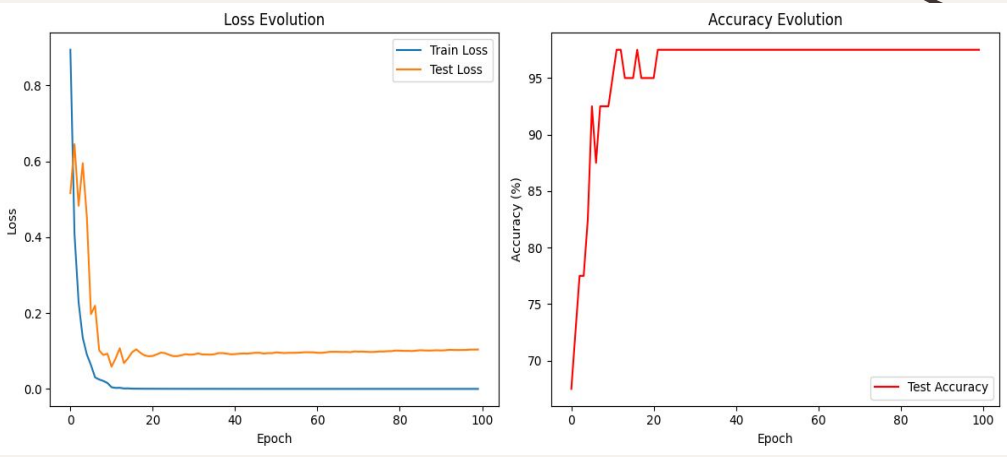




Résultats avec CNN “GUNPOINT”



dataset déséquilibré



rééquilibre avec les images générées

- **Innovations et apprentissages :** L’ouverture de nouvelles voies en appliquant la génération de données synthétiques aux séries temporelles, révélant des améliorations significatives dans la classification.

- **Impact sur la classification de séries temporelles :** La génération de données synthétiques a prouvé son efficacité pour enrichir les datasets déséquilibrés et améliorer la précision des classifications.

- **Perspectives futures :** Les défis identifiés invitent à une exploration plus poussée des techniques de génération de données et des architectures de modèles pour les séries temporelles.

- **Contribution au domaine :** Ce travail enrichit le champ de l'augmentation de données synthétiques pour les séries temporelles, posant les bases pour des avancées futures dans l'analyse prédictive.

Merci de votre attention
Avez-vous des questions ?

Classification : LSTM

Exemple d'Architecture :

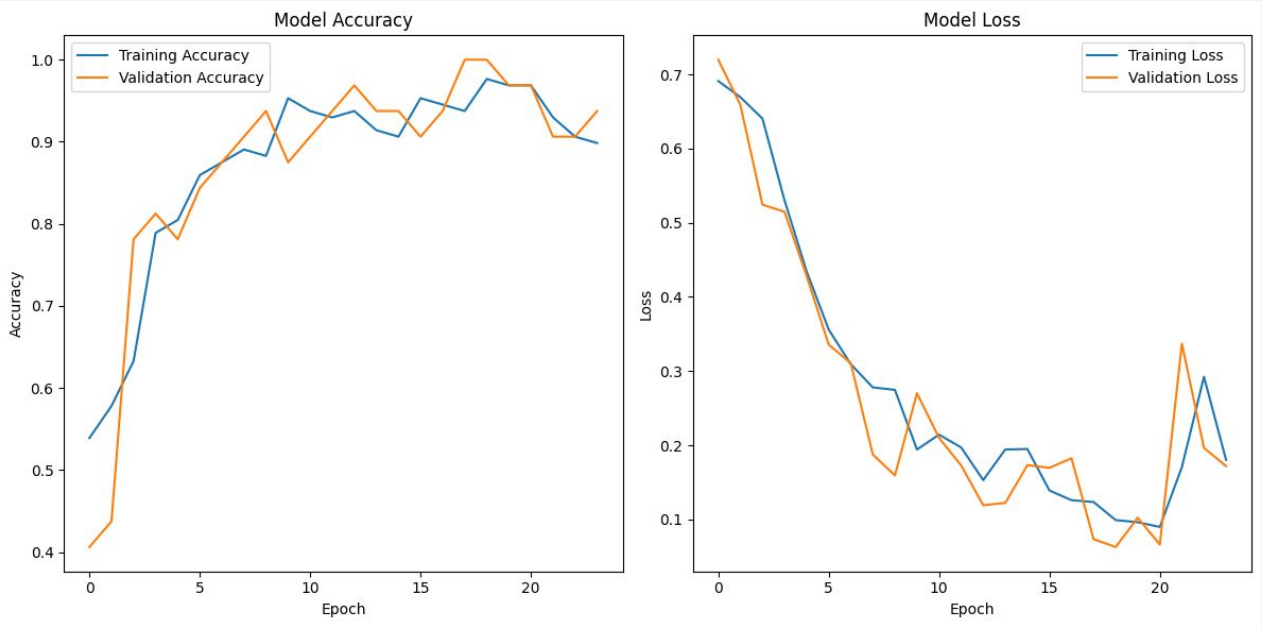
```python
def build_model(input_shape,num_classes,lstm_units=50,dropout=0.2):
    model = Sequential()
    model.add(LSTM(lstm_units, activation='tanh', recurrent_activation='sigmoid', input_shape=input_shape, return_sequences=True))
    model.add(Dropout(dropout))
    model.add(LSTM(lstm_units, activation='tanh', recurrent_activation='sigmoid'))
    model.add(Dropout(dropout))
    model.add(Dense(num_classes, activation='sigmoid'))  # Assuming binary classification

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Create a ModelCheckpoint callback and Early stop
early_stop = EarlyStopping(monitor='val_loss',patience = 5, verbose = 1, restore_best_weights=True)
checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', verbose=1, save_best_only=True, mode='min')
```

Classification : LSTM

Résultats : Gunpoint



Équilibré

```
Test Accuracy: 95.00%
Errors = 2
```

Déséquilibré

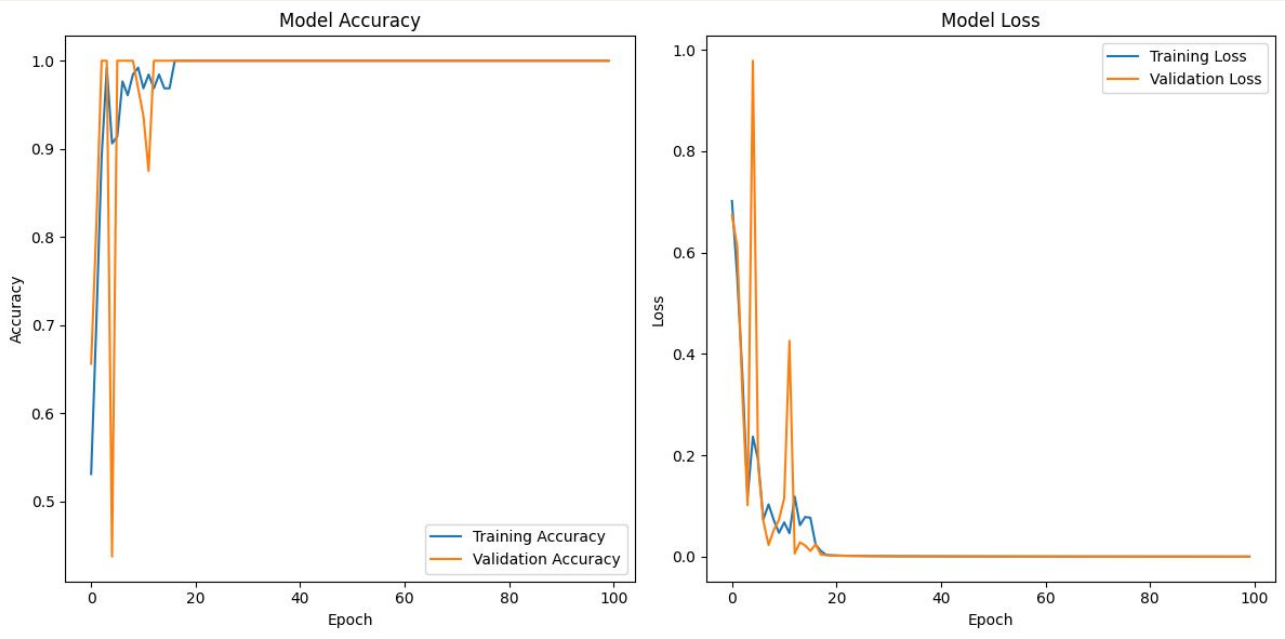
```
Test Accuracy: 50.00%
Errors = 20
```

Équilibré avec SMOTE

```
Test Accuracy: 85.00%
Errors = 6
```

Classification : LSTM

Résultats : Gunpoint (en Images)



Équilibré

```
Test Accuracy: 95.00%
Errors = 2
```

Déséquilibré

```
Test Accuracy: 72.50%
Errors = 11
```

Équilibré avec GAN

```
Test Accuracy: 95.00%
Errors = 2
```

Équilibré avec DCGAN

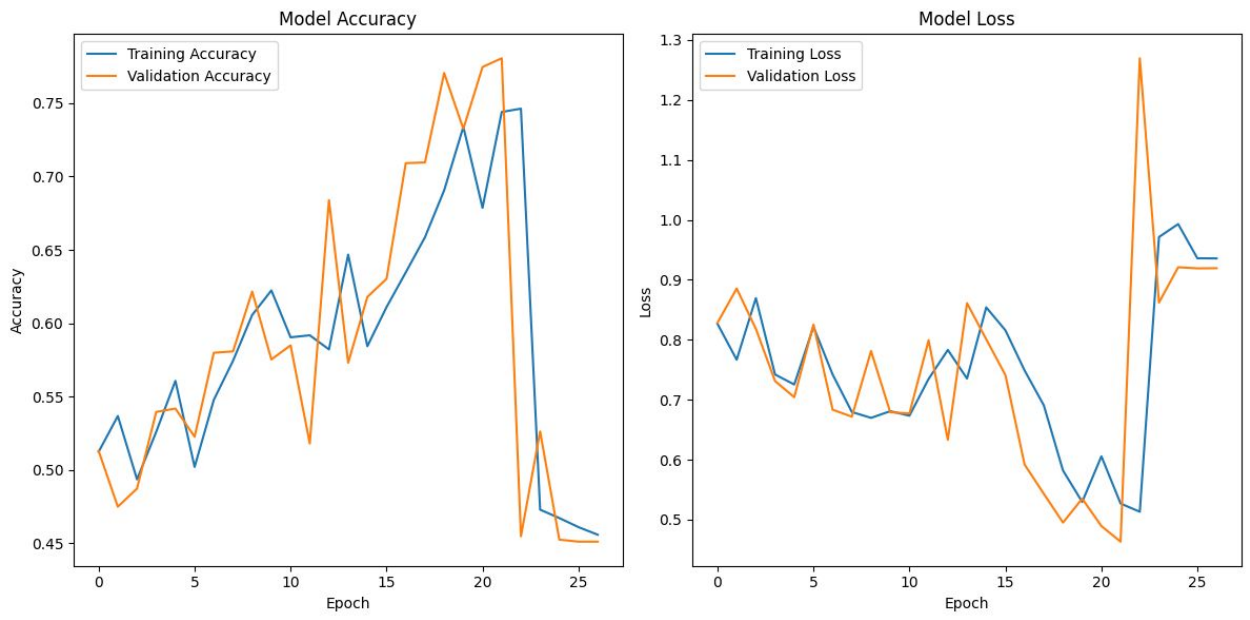
```
Test Accuracy: 97.50%
Errors = 1
```

Équilibré avec VQVAE + GAN

```
Test Accuracy: 87.50%
Errors = 5
```

Classification : LSTM

Résultats : FaultDetectionA



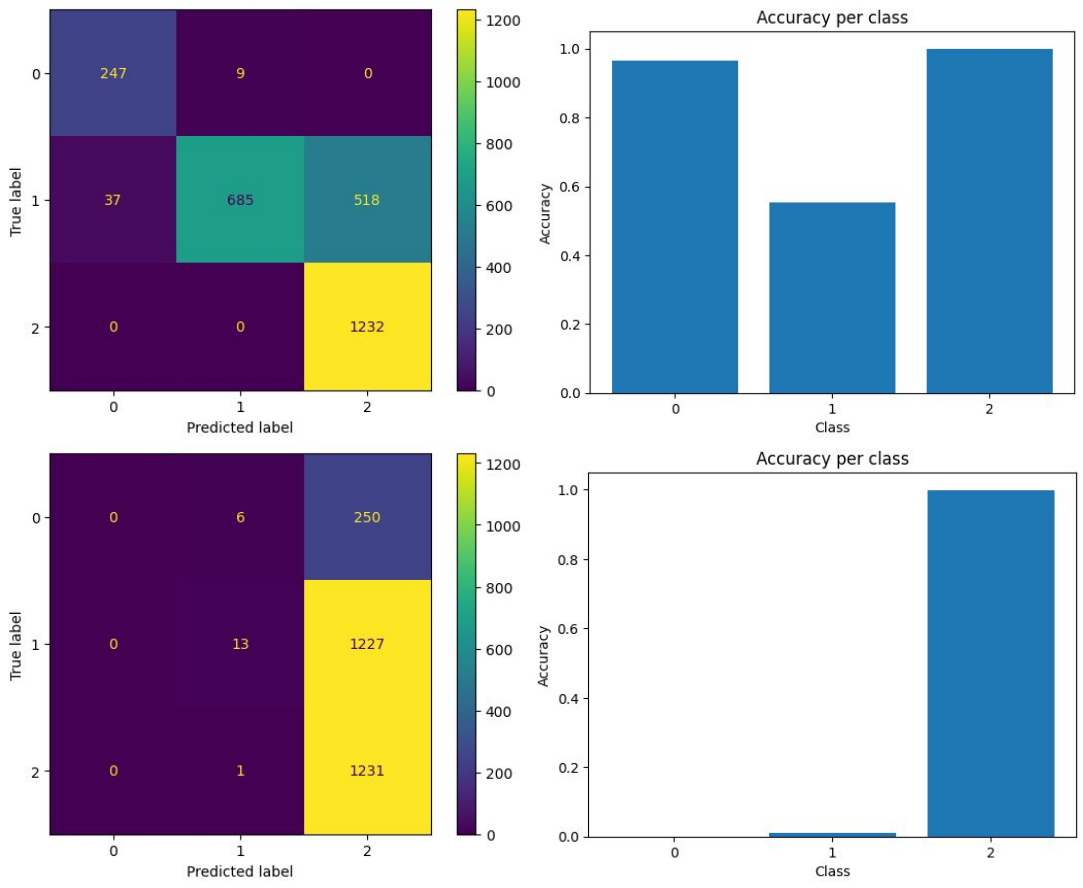
Déséquilibré

```
Test Accuracy: 79.33%
Test Avg F1-Score: 81.65%
Errors = 564
```

Équilibré avec SMOTE

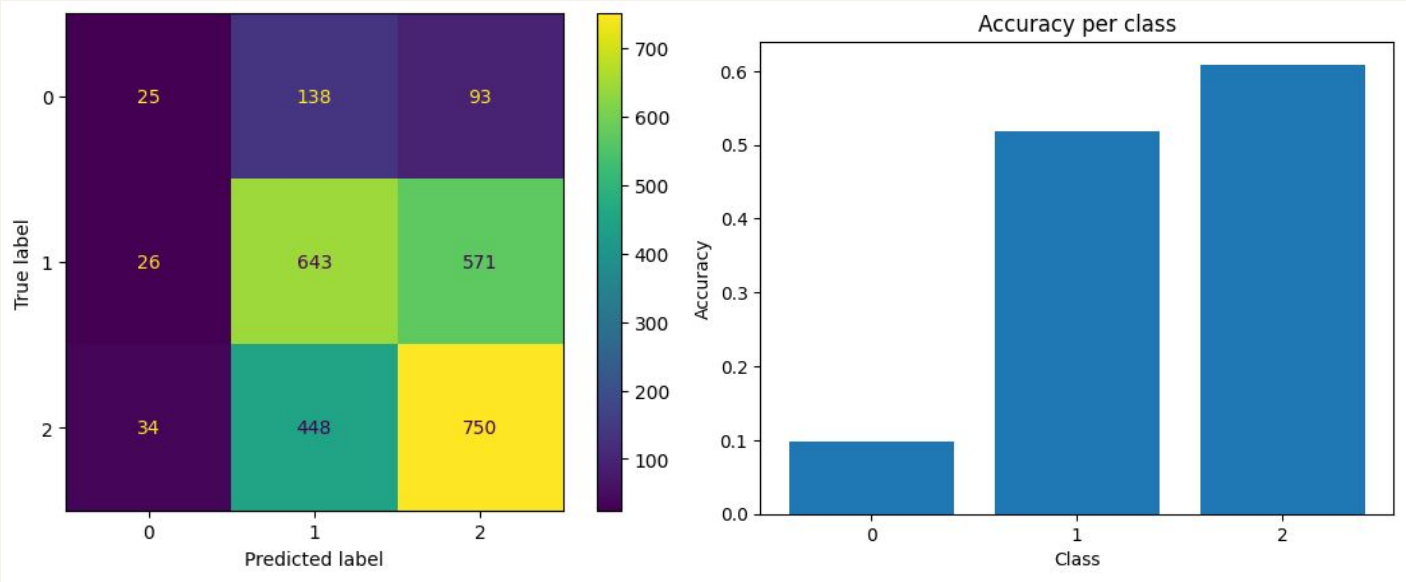
```
Test Accuracy: 45.60%
Errors = 1484
```

Résultats : FaultDetectionA



Classification : LSTM

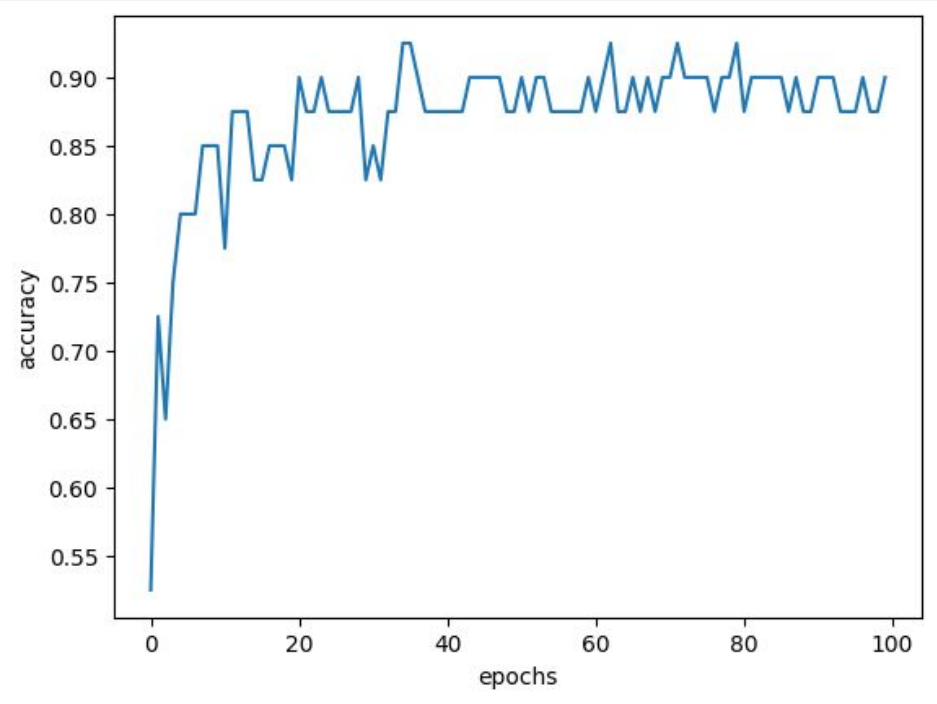
Résultats : FaultDetectionA (en Images)

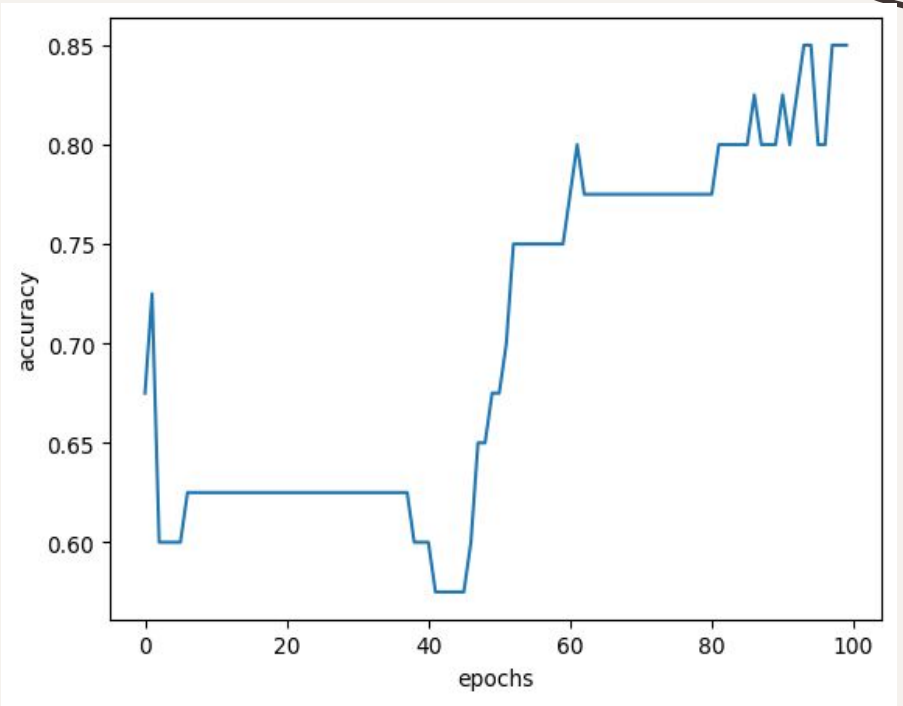


Déséquilibré

```
Test Accuracy: 51.98%
Test Avg F1-Score: 41.15%
Errors = 1310
```

II.3 Le déséquilibre de dataset





Précision sur l'ensemble test initial (sur dataset complet à gauche et dataset déséquilibré à droite