

TP - Traitement d'images

Séance 1 : Prise en main d'un CNN

Metuarea Herearii - LARIS, Université d'Angers

Master 2 Data Science, Université d'Angers

27 Janvier 2025

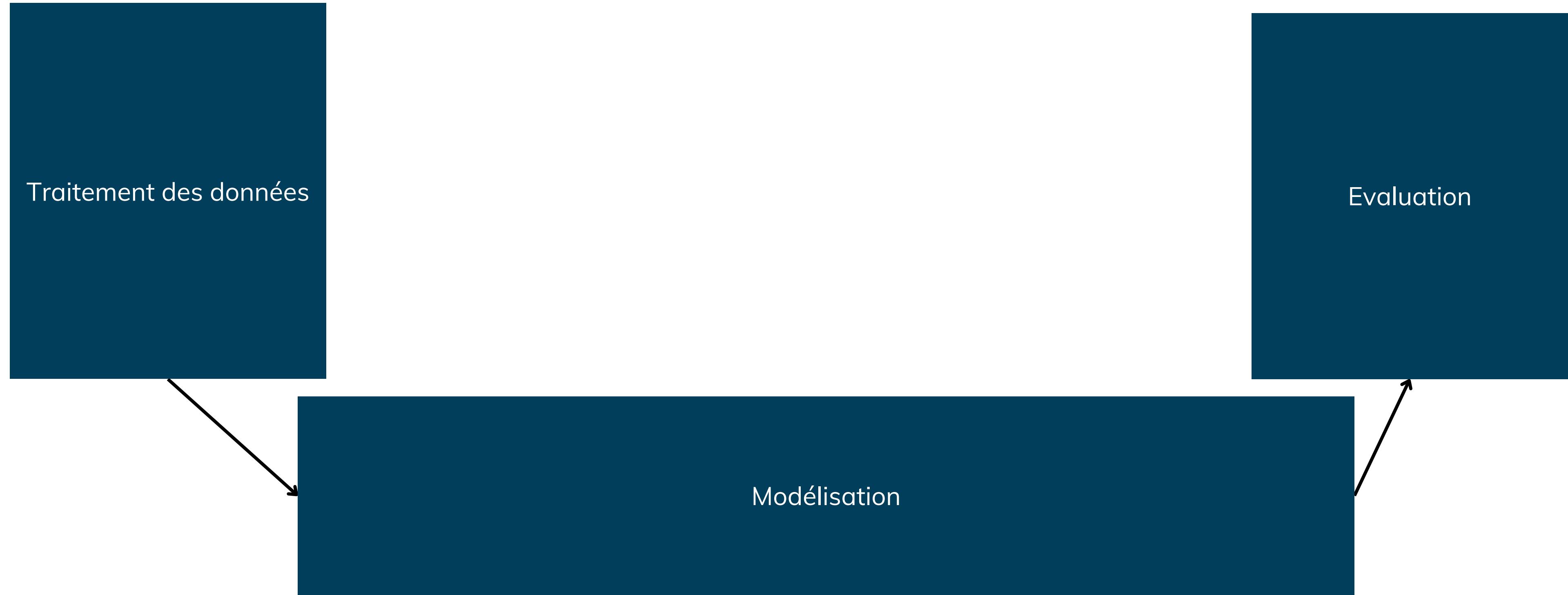


Plan de la séance

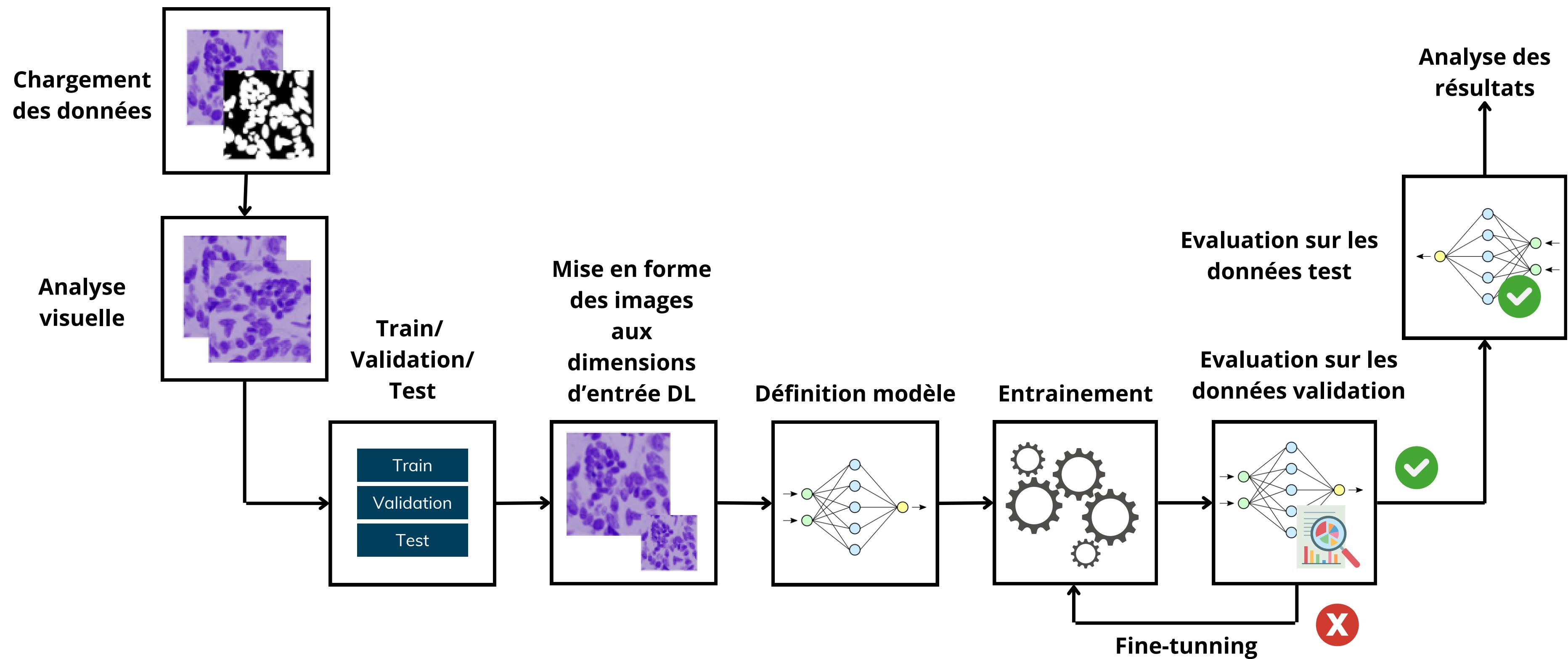
Guide pour concevoir un modèle DL de traitement d'images

TP

Guide pour concevoir un modèle de traitement d'images

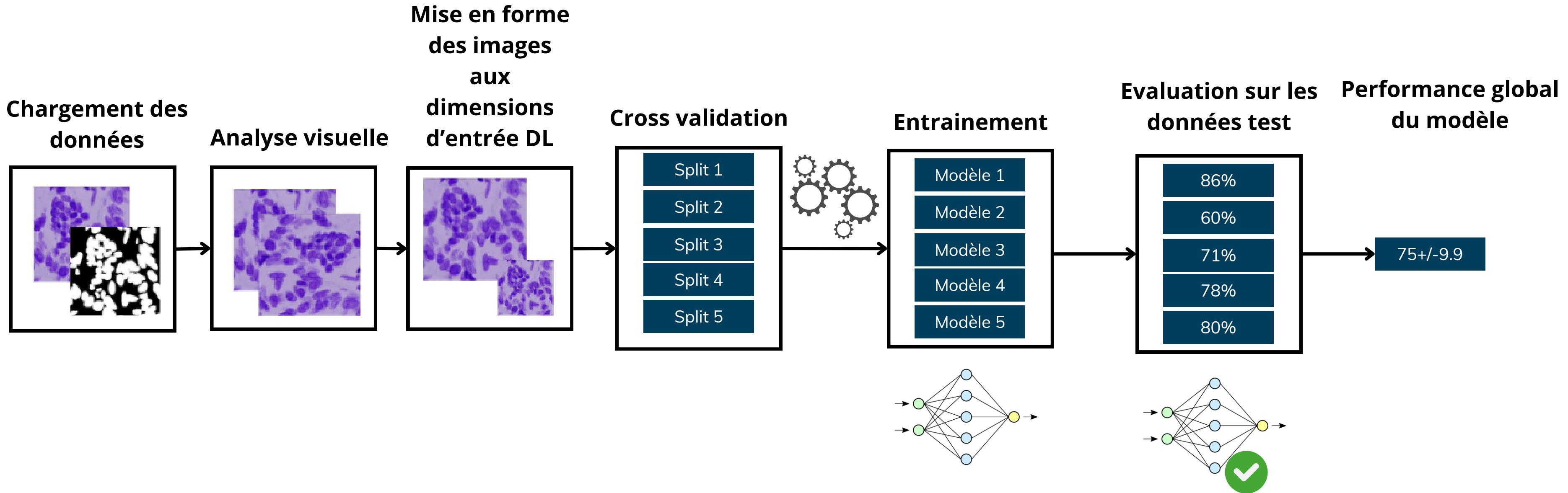


Guide pour concevoir un modèle de traitement d'images



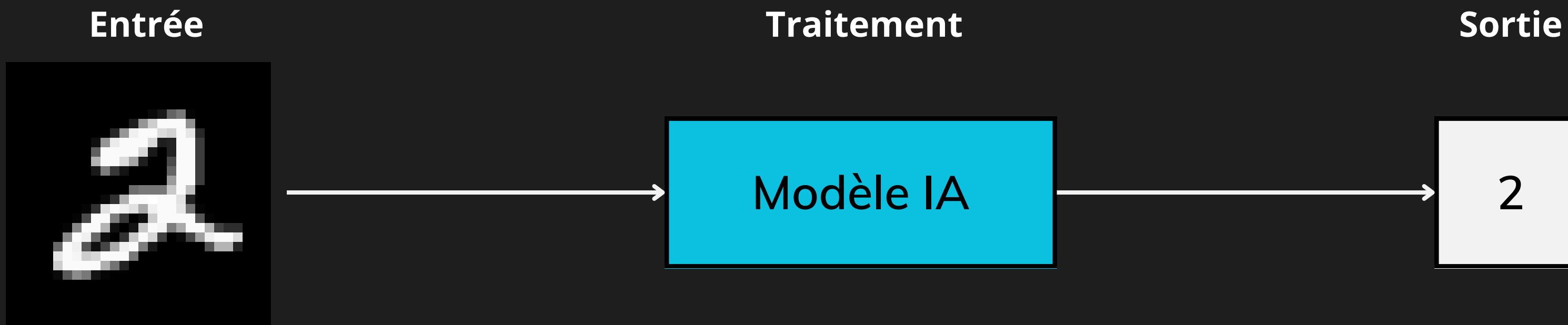
Guide pour concevoir un modèle de traitement d'images

Extra - Validation croisée



TP CN

Objectif de ce TP : Concevoir un modèle de classification d'images



TP CN

Prérequis : Importation des librairies et définition de fonctions utiles

```
# Visualisation display

def show_sample(X_train, Y_train):
    plt.figure(figsize=(5,5))
    plt.subplot(1,4,1)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,2)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,3)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,4)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])

def show_fx_negatif(X_test, Y_pred, Y_test):

    Y_pred_classes = np.argmax(Y_pred, axis=1)
    Y_true_classes = np.argmax(Y_test, axis=1)

    FX_NEGATIF = X_test[(Y_pred_classes!=Y_true_classes)]
    Pred_fx_negatif = Y_pred_classes[(Y_pred_classes!=Y_true_classes)]
    GT_fx_negatif = Y_true_classes[(Y_pred_classes!=Y_true_classes)]
```

TP CN

Prérequis : Importation des librairies et définition de fonctions utiles

```
# Visualisation display

def show_sample(X_train, Y_train):
    plt.figure(figsize=(5,5))
    plt.subplot(1,4,1)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,2)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,3)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,4)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])

def show_fx_negatif(X_test, Y_pred, Y_test):

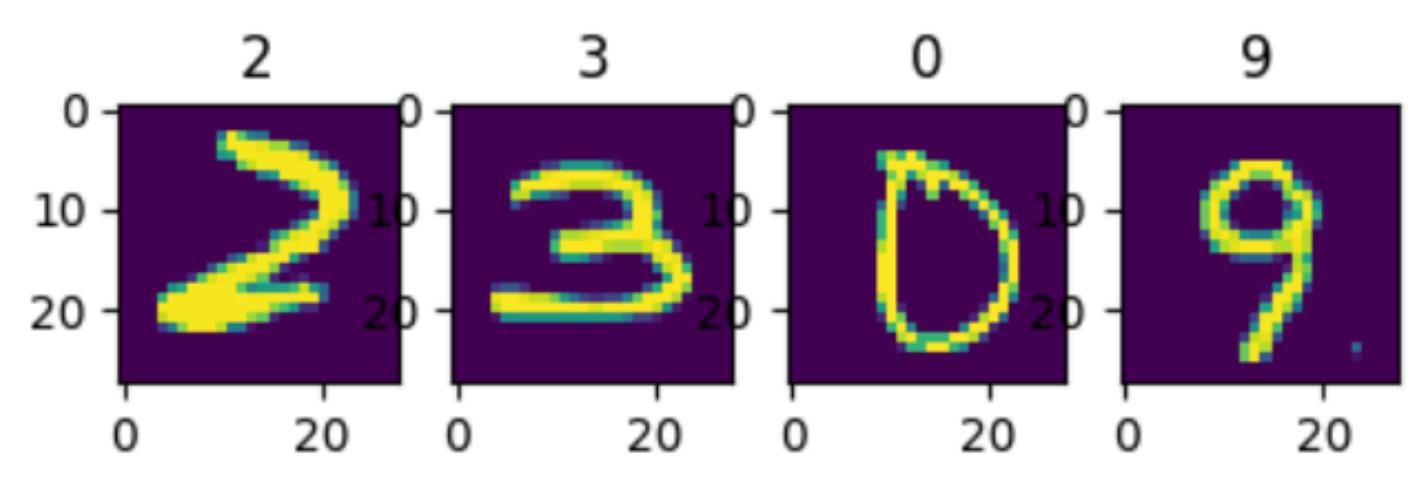
    Y_pred_classes = np.argmax(Y_pred, axis=1)
    Y_true_classes = np.argmax(Y_test, axis=1)

    FX_NEGATIF = X_test[(Y_pred_classes!=Y_true_classes)]
    Pred_fx_negatif = Y_pred_classes[(Y_pred_classes!=Y_true_classes)]
    GT_fx_negatif = Y_true_classes[(Y_pred_classes!=Y_true_classes)]
```

TP CN

I- Importation et visualisation des données

```
MNIST Dataset Shape (images, hauteur, largeur):  
X_train: (60000, 28, 28)  
Y_train: (60000,)  
X_test: (10000, 28, 28)  
Y_test: (10000,)
```



Train	Test
-------	------



TP CN

II - Division de la base de données en train, validation et test

```
Shape of X_train: (48000, 28, 28)
Shape of Y_train: (48000,)
Shape of X_val: (12000, 28, 28)
Shape of Y_val: (12000,)
Shape of X_test: (10000, 28, 28)
Shape of Y_test: (10000,)
```

TP CN

III - Pré-traitement des données

a) Changement des dimensions pour adapter à l'entrée du FCN et mise en format float32

```
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32')  
X_val = X_val.reshape(X_val.shape[0], 784).astype('float32') #add .  
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32')
```

b) Normalisation des images pour adapter à l'entrée du CNN

```
X_train /= 255 # noi  
X_val /= 255 # norm  
X_test /= 255 # noi
```

c) Changement dans le codage de chaque label de sortie pour adapter à la sortie du CNN

```
# one hot encode outputs  
Y_train = to_categorical(Y_train)  
Y_val = to_categorical(Y_val)  
Y_test = to_categorical(Y_test)
```

Après	100000000	1	Avant
	010000000	2	
	001000000←	3	
	
	000000001	9	

TP CN

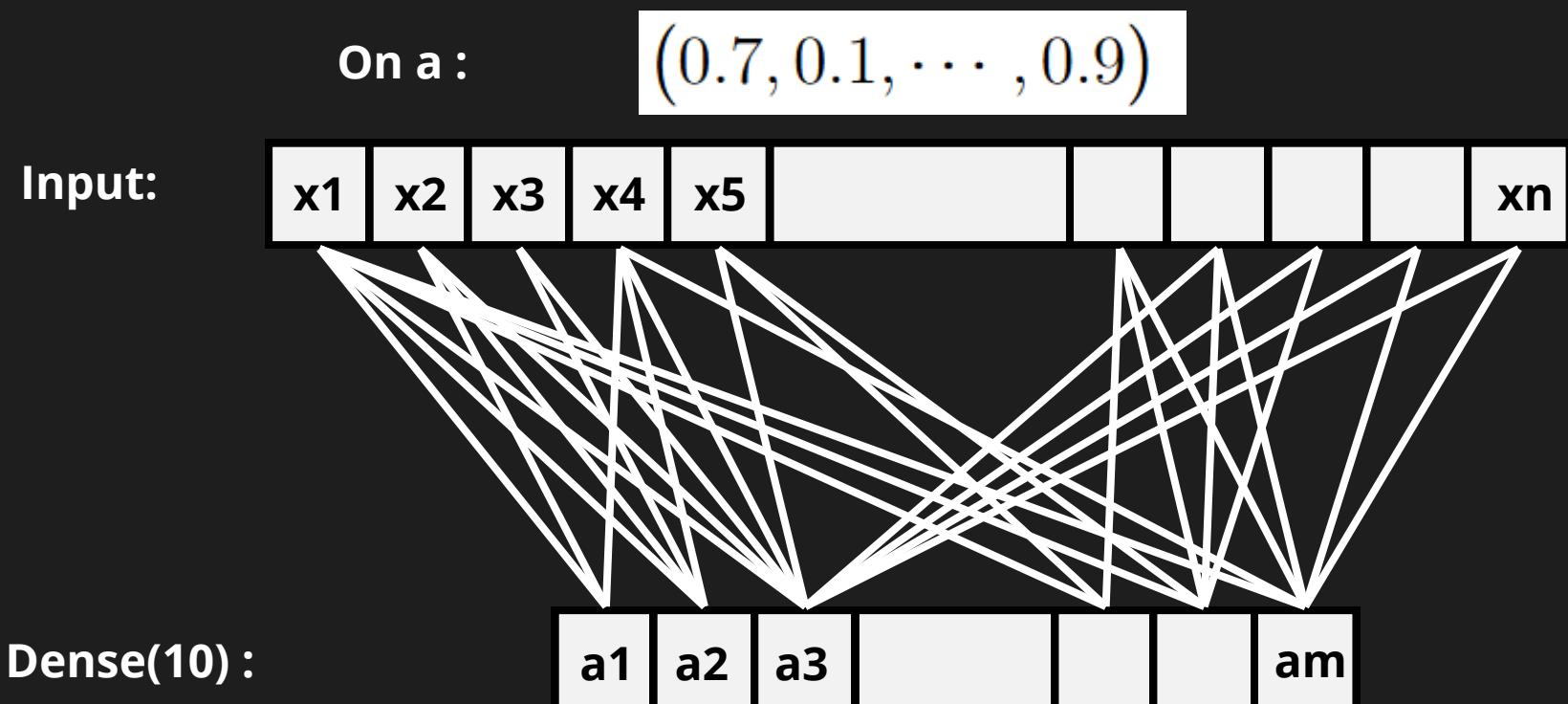
V- Définition du modèle

```
model = Sequential([
    Dense(10, input_shape=(784,)),
    Activation('softmax'),
])
```

TP CN

V- Définition du modèle

```
model = Sequential([  
    Dense(10,input_shape=(784,)),  
    Activation('softmax'),  
])
```



Calcul de la somme pondérée

$$\forall k \in \{1, \dots, m\}, a_k = \sum_{i=1}^n x_i w_i + b$$

avec $n = 784$ et $m = 10$

TP CN

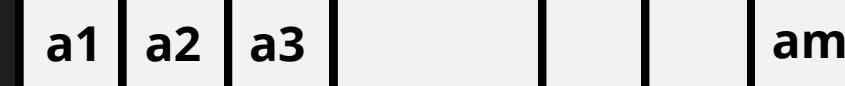
V- Définition du modèle

```
model = Sequential([  
    Dense(10,input_shape=(784,)),  
    Activation('softmax'),  
])
```

Input:



Dense(10):



Calcul de la somme pondérée

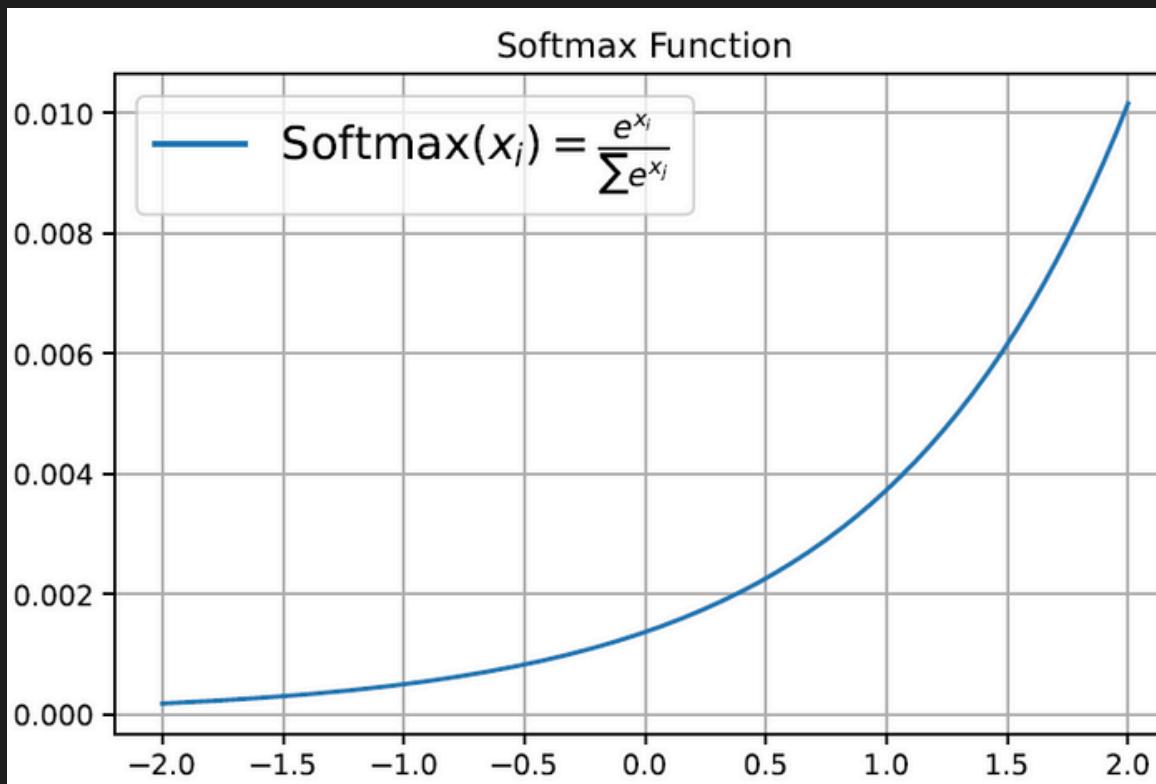
$$\forall k \in \{1, \dots, m\}, a_k = \sum_{i=1}^n x_i w_i + b$$

avec $n = 784$ et $m = 10$

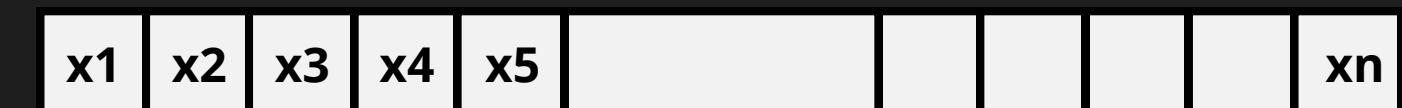
TP CN

V- Définition du modèle

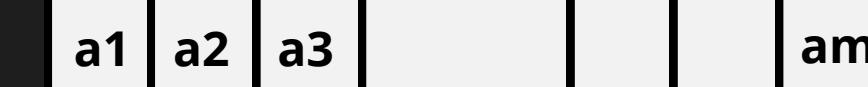
```
model = Sequential([
    Dense(10,input_shape=(784,)),
    Activation('softmax'),
])
```



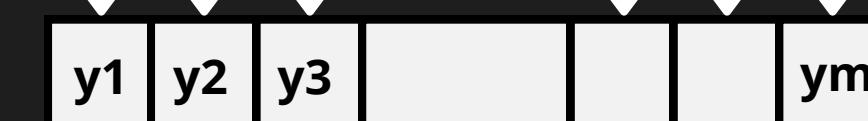
Input:



Dense(10):



Fonction
d'Activation SoftMax:



Calcul de la fonction d'activation Softmax f

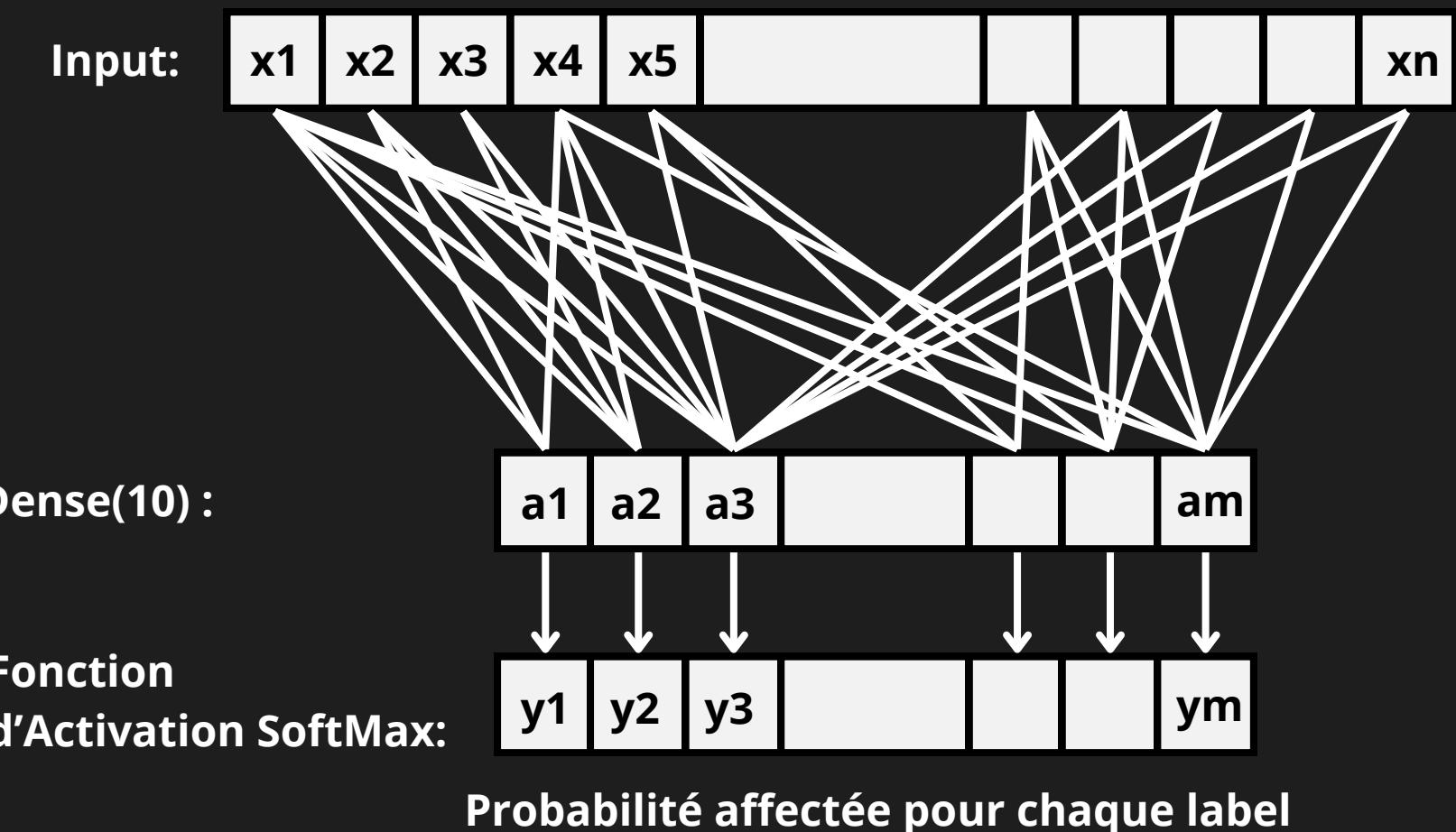
$$\forall k \in \{1, \dots, m\}, y_k = f \left(\sum_{i=1}^n x_i w_i + b \right)$$

avec $n = 784$ et $m = 10$

TP CN

V- Définition du modèle

```
model = Sequential([  
    Dense(10,input_shape=(784,)),  
    Activation('softmax'),  
])
```



TP CN

V- a) Hyperparamètres de l'entraînement

TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

TP CN

V- a) Hyperparamètres de l'entraînement

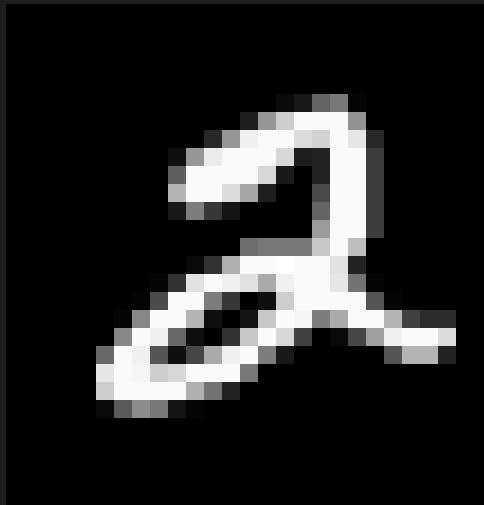
Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Fonction de perte

Entrée



Modèle IA

Sortie

0.1
0.2
...
0.05
0.05

TP CN

V- a) Hyperparamètres de l'entraînement

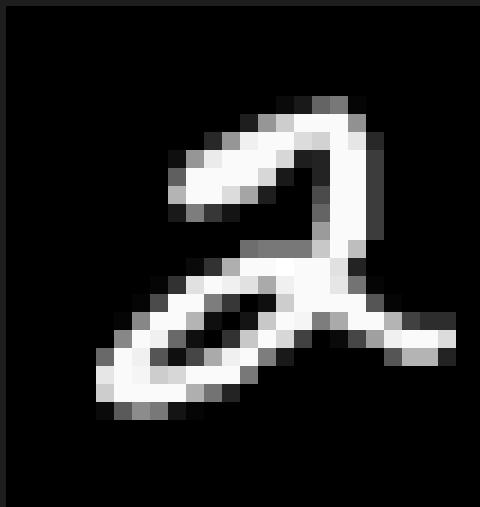
Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Fonction de perte

Entrée



Modèle IA

Sortie

0.1
0.2
...
0.05
0.05

Vérité terrain

0
1
...
0
0

TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Prédit

Vérité terrain

0.1

0

0.2

1

...

...

0.05

0

0.05

0

Fonction de perte
Categorical cross entropy

$$CE = - \sum_{i=1}^N y_i \log (\hat{y}_i)$$

avec :

- y_i le label vérité terrain i : 0 ou 1
- \hat{y}_i le label prédit i : une probabilité entre 0 et 1.

CE = 1.609

TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Métrique Accuracy

$$Accuracy = \frac{TP}{TP + TN + FN + FP}$$

TP CN

V- a) Hyperparamètres de l'entraînement

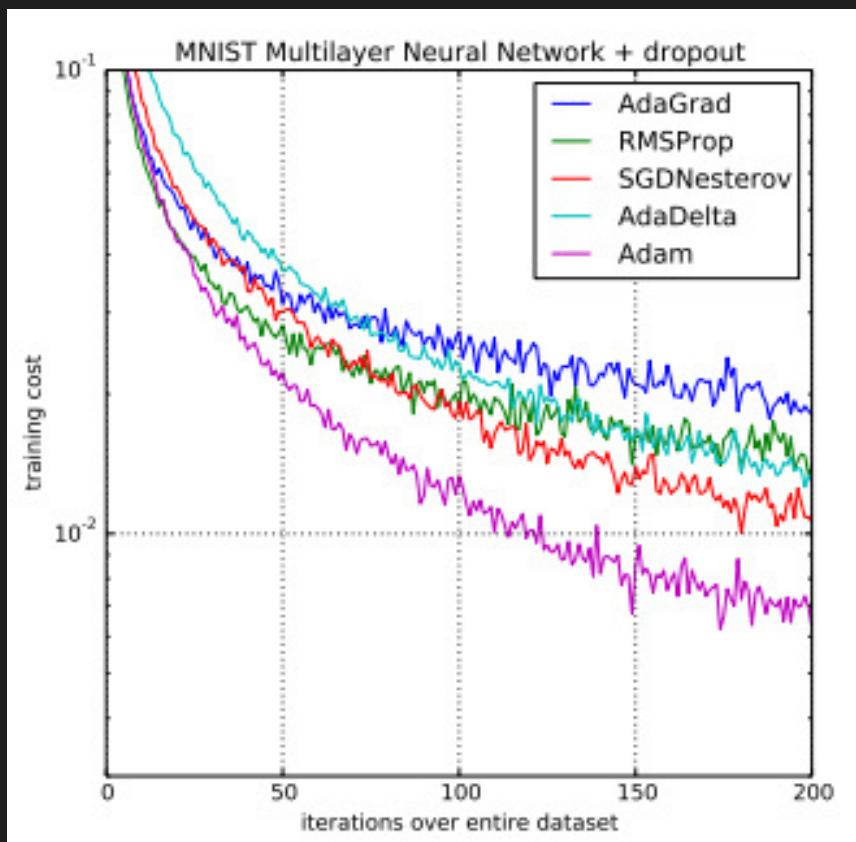
Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Algorithme d'optimisation

Adam



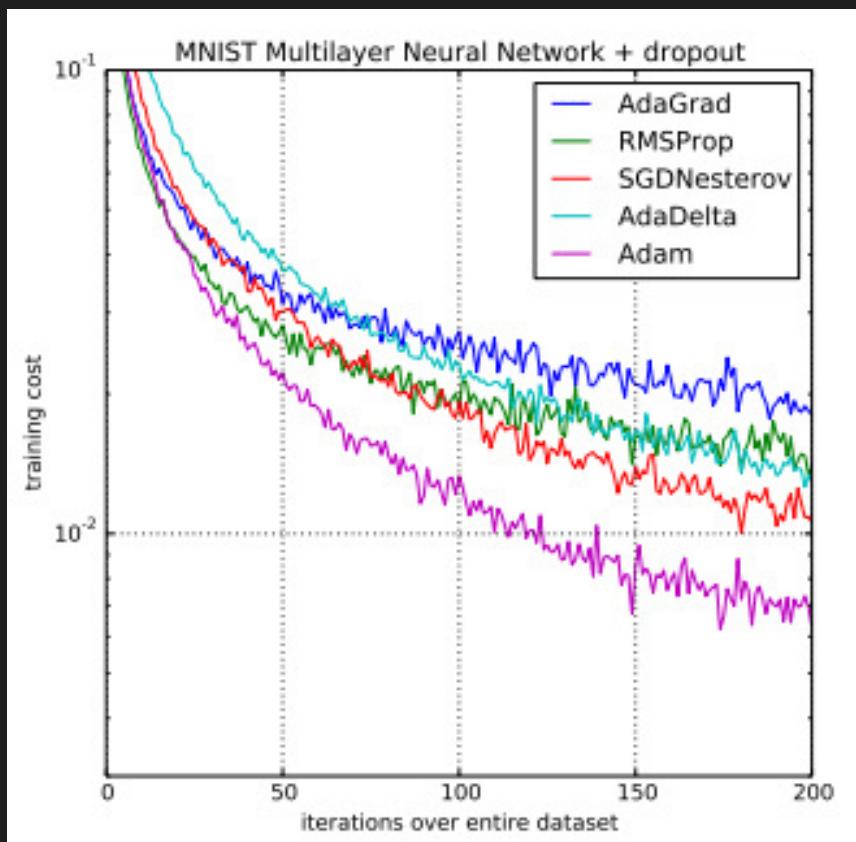
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```



Algorithme d'optimisation

Adam

Hyperparamètre intéressant : Learning rate

TP FCN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement :

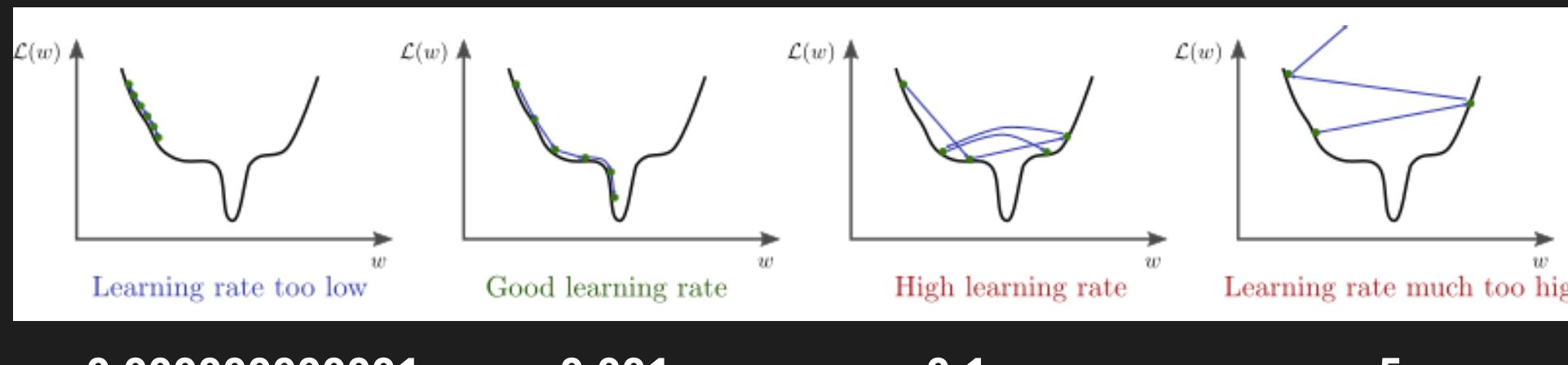
- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

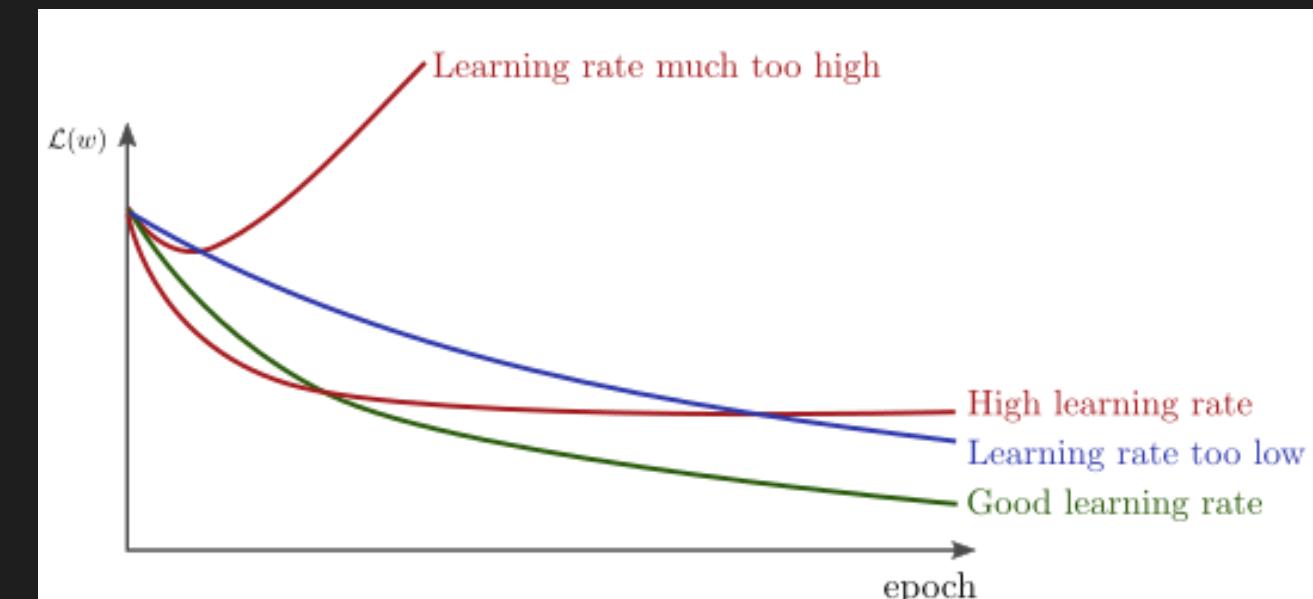
Algorithme d'optimisation

Adam

Hyperparamètre intéressant : Learning rate



Exemple en général



TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressant pour l'entraînement:

- Fonction de perte
 - Métrique
 - Algorithme d'optimisation
-
- EarlyStopping
 - Save best model

```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose=  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

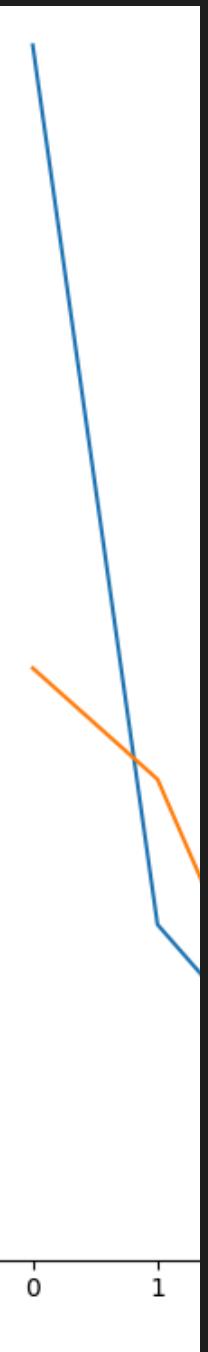
Sauvegarde du meilleur modèle détenant la plus petite valeur
d'erreur

Early stopping

Arrêt de l'entraînement si aucune amélioration de la fonction de
perte sur les données de validation après 5 epoch

TP CN

V- a) Hyperparamètres de l'entraînement



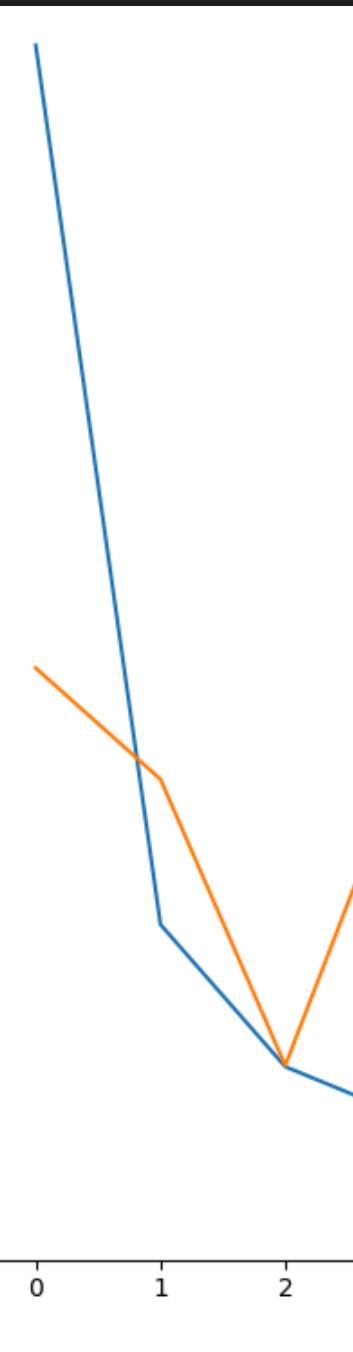
```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:  
  
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose:  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

Early stopping

TP CN

V- a) Hyperparamètres de l'entraînement



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:  
  
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose:  
    save_weights_only=False, mode='auto') #Callback
```

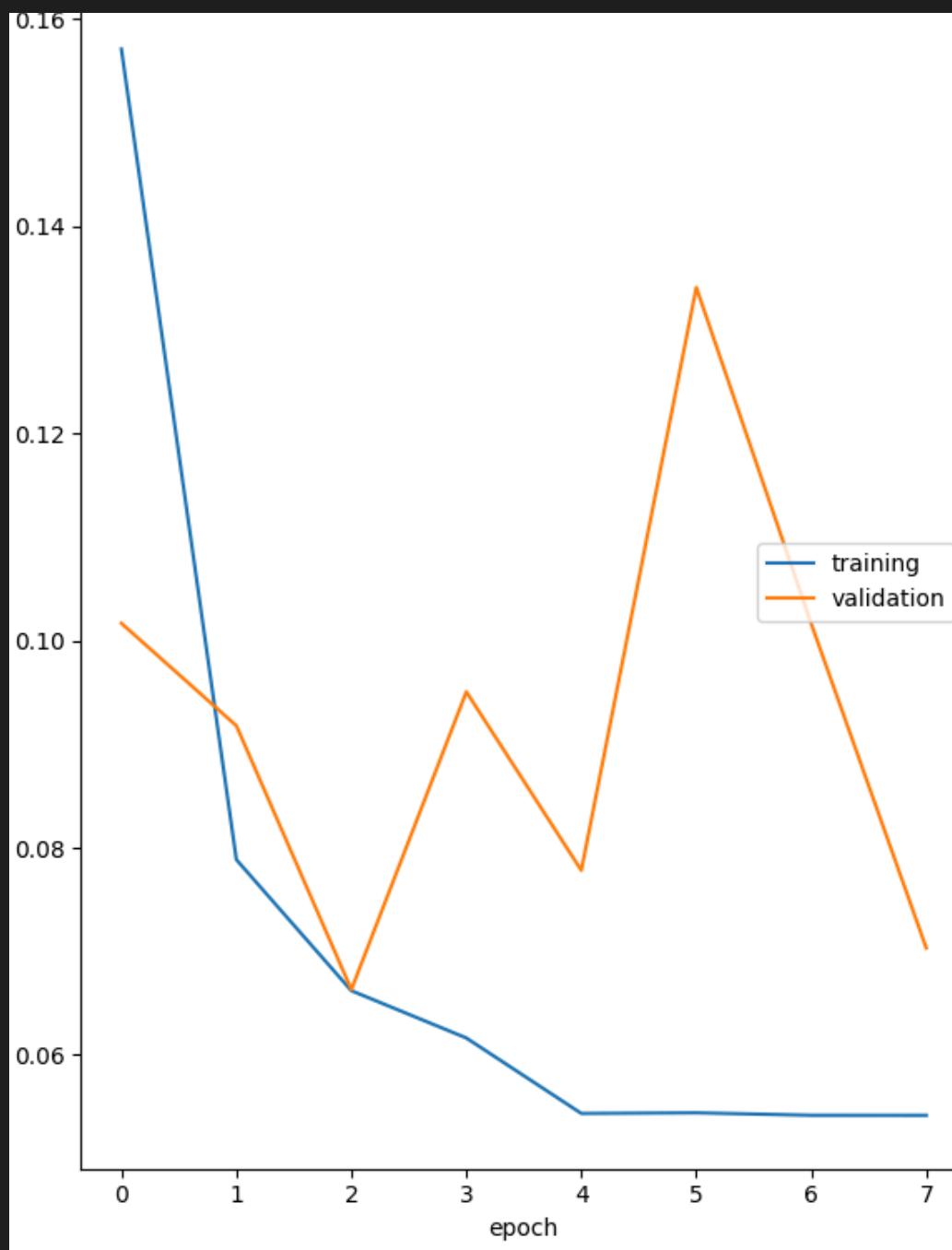
Save best model

Epoch 2 : Plus petite erreur -> Sauvegarde du modèle

Early stopping

TP FCN

V- a) Hyperparamètres de l'entraînement



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose=  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

Epoch 2 : Plus petite erreur -> Sauvegarde du modèle

Early stopping

Epoch 2 : Plus petite erreur

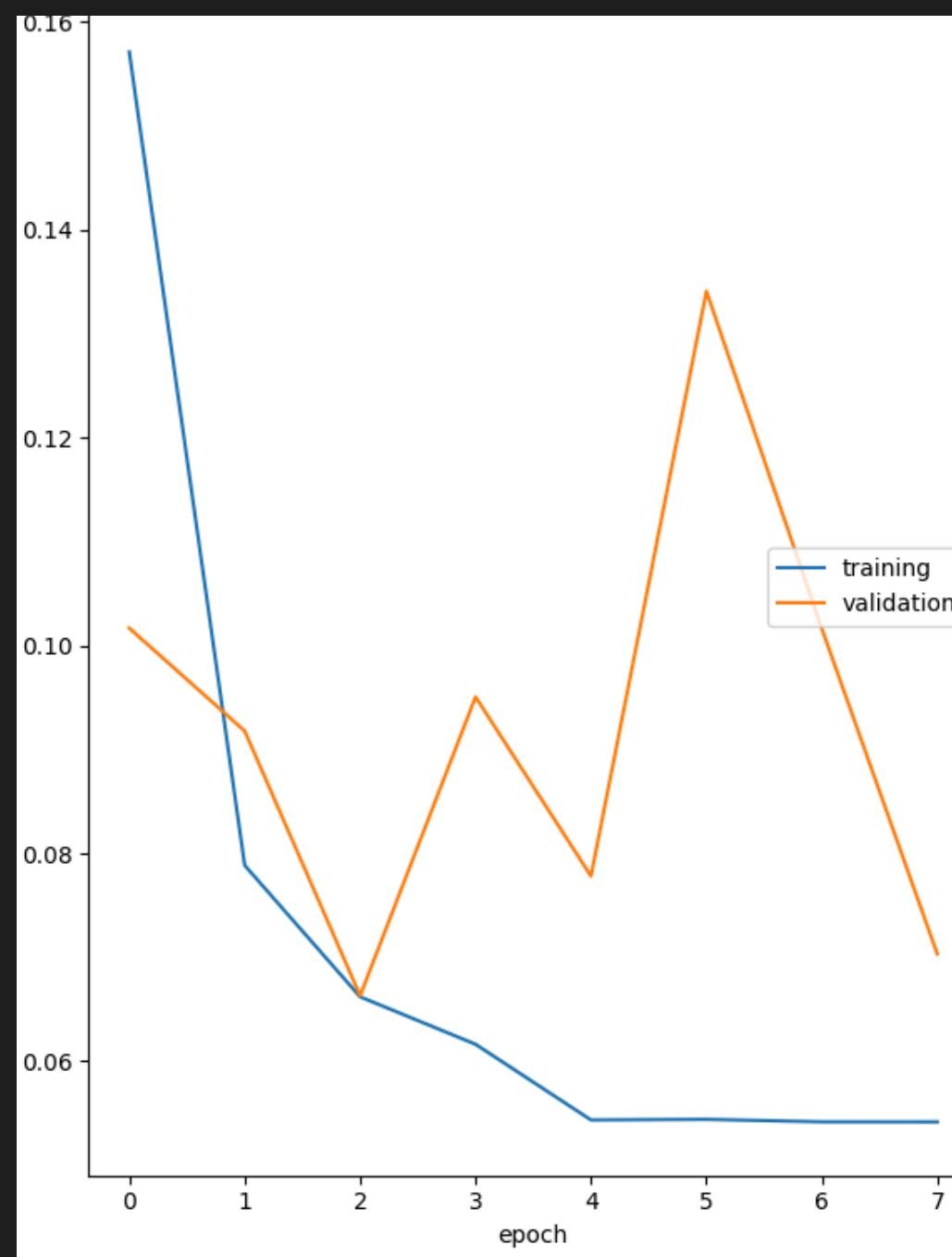
Epoch 3-4-5-6-7 : Pas d'erreur plus petite que l'epoch 2



Début Overfitting

TP CN

V- a) Hyperparamètres de l'entraînement



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose=  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

Epoch 2 : Plus petite erreur -> Sauvegarde du modèle

Early stopping

Epoch 2 : Plus petite erreur

Epoch 3-4-5-6-7 : Pas d'erreur plus petite que l'epoch 2

Entrainement : Stop

TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- **Algorithme d'optimisation**
- **Fonction de perte**
- **Métrique**
- **EarlyStopping**
- **Save best model**

- **Batch size**
- **Epoch**

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```

TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique
- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```

Dataset

- Batch size
- Epoch

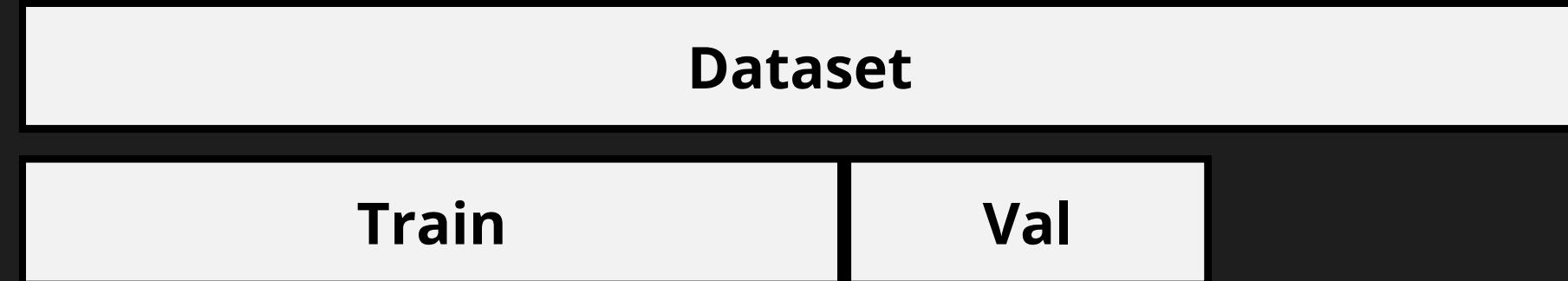
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique
- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



- Batch size
- Epoch

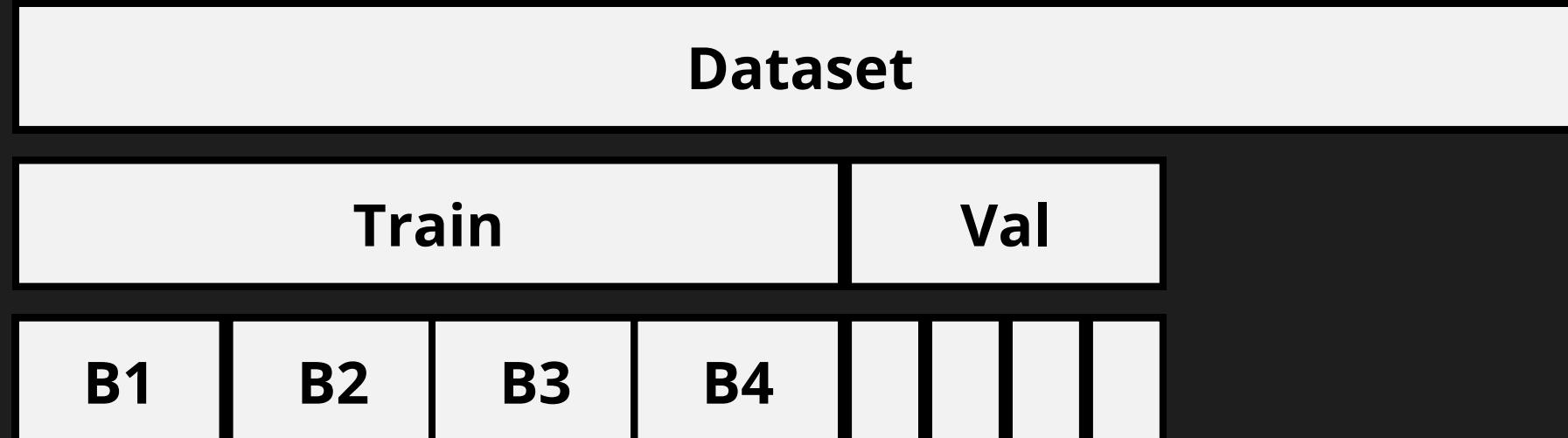
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
 - Fonction de perte
 - Métrique
-
- EarlyStopping
 - Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

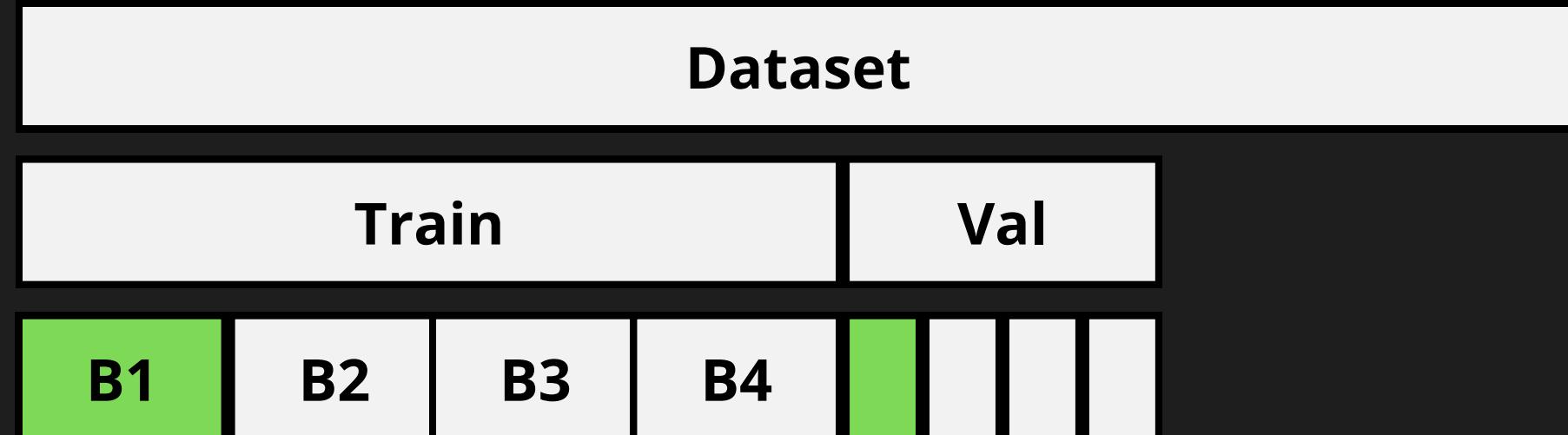
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
 - Fonction de perte
 - Métrique
-
- EarlyStopping
 - Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

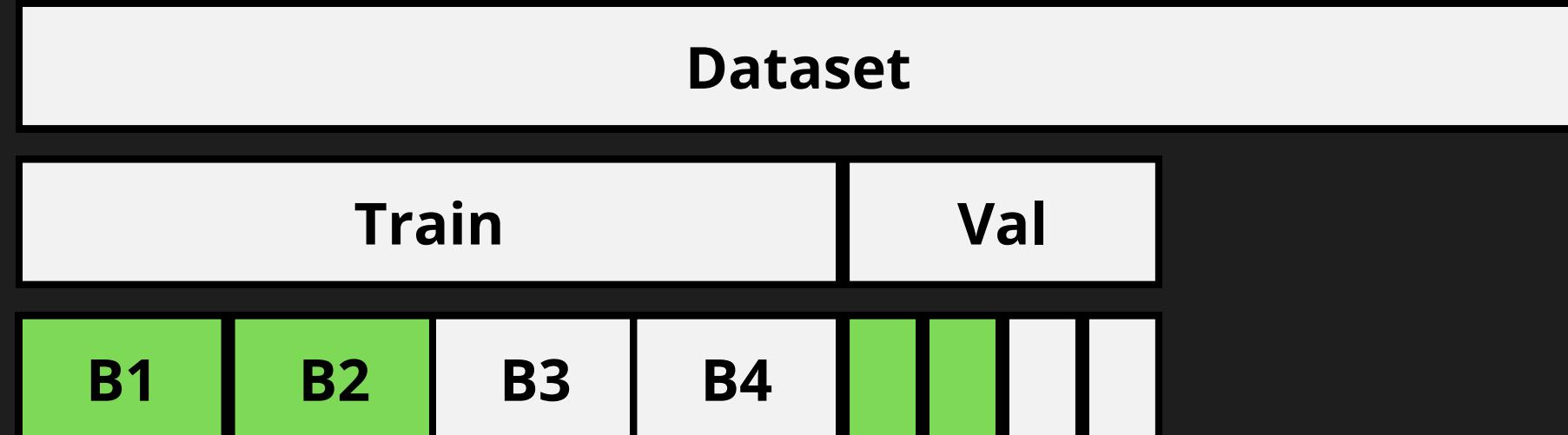
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique
- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



- Batch size
- Epoch

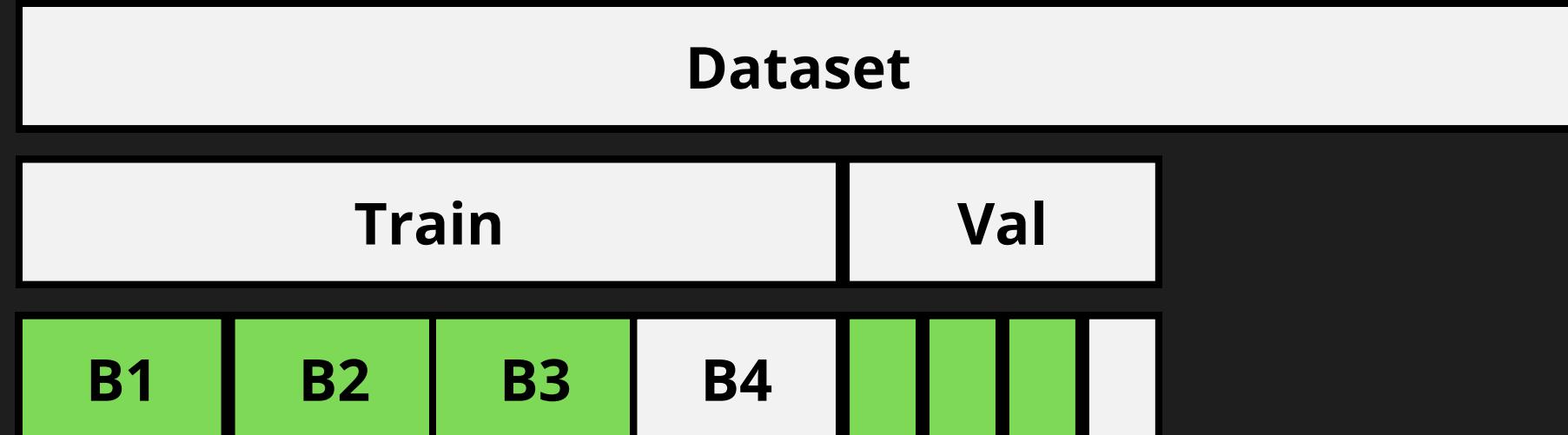
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
 - Fonction de perte
 - Métrique
-
- EarlyStopping
 - Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

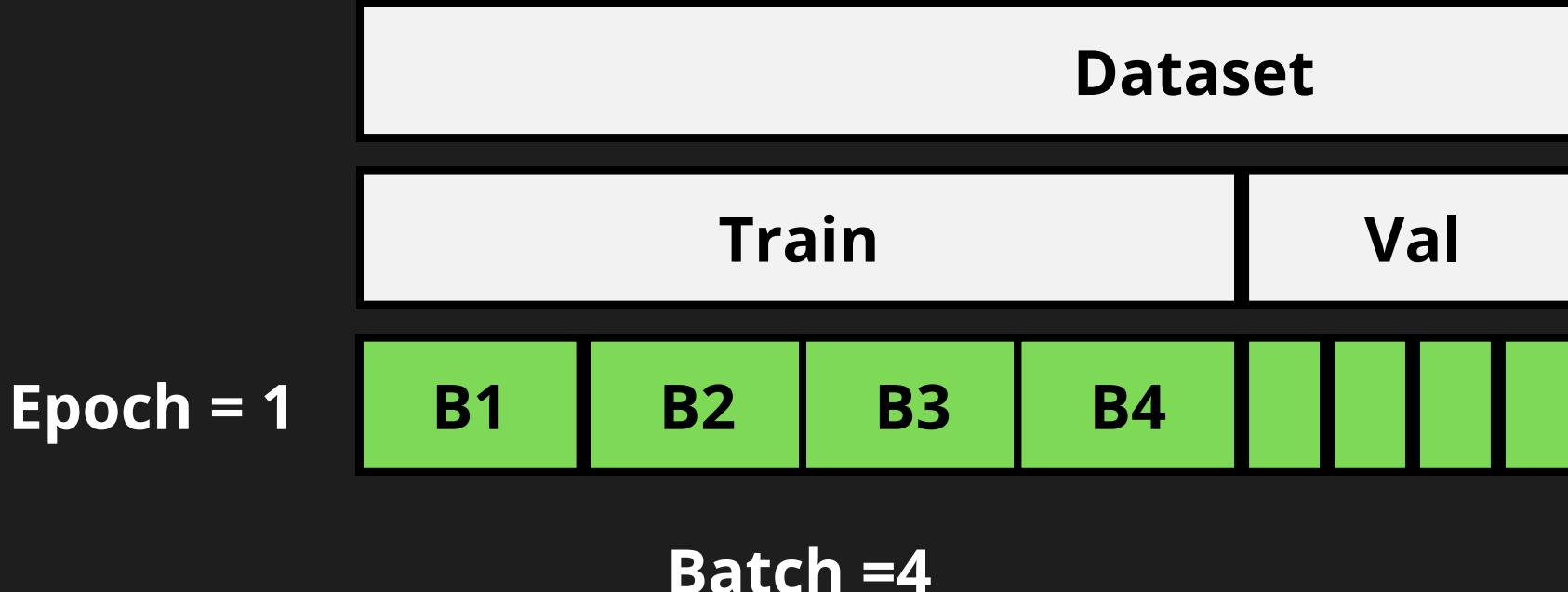
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
 - Fonction de perte
 - Métrique
-
- EarlyStopping
 - Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



- Batch size
- Epoch

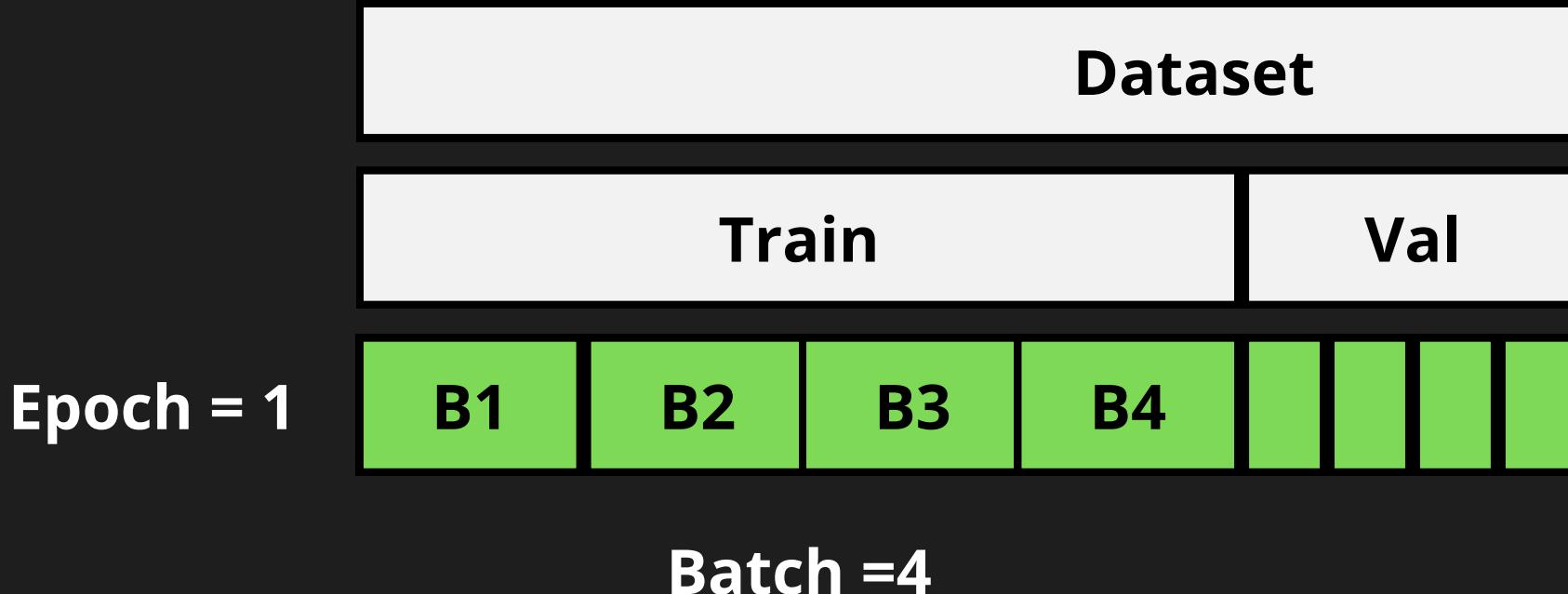
TP CN

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressants pour l'entraînement:

- Algorithme d'optimisation
 - Fonction de perte
 - Métrique
-
- EarlyStopping
 - Save best model

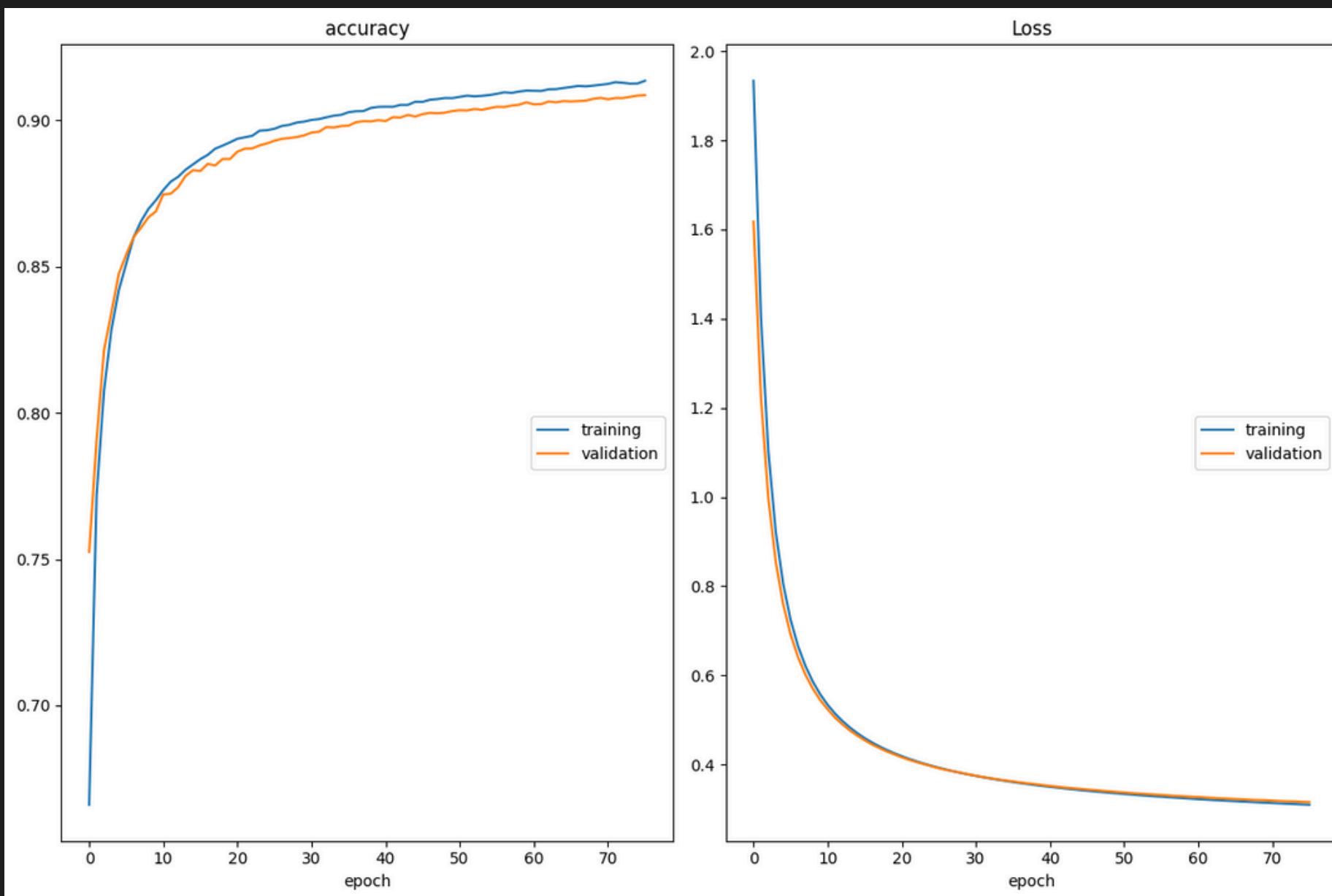
```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



- Batch size
- Epoch

TP CN

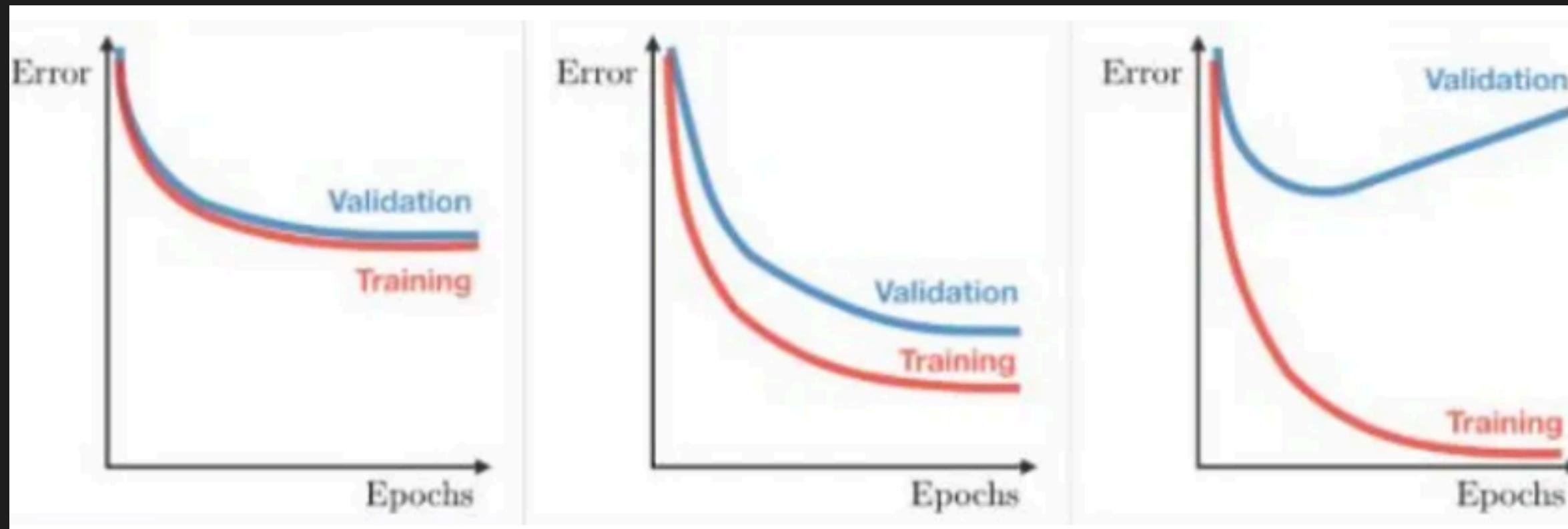
V- b) Entrainement du modèle et évaluation sur les données de validation



TP CN

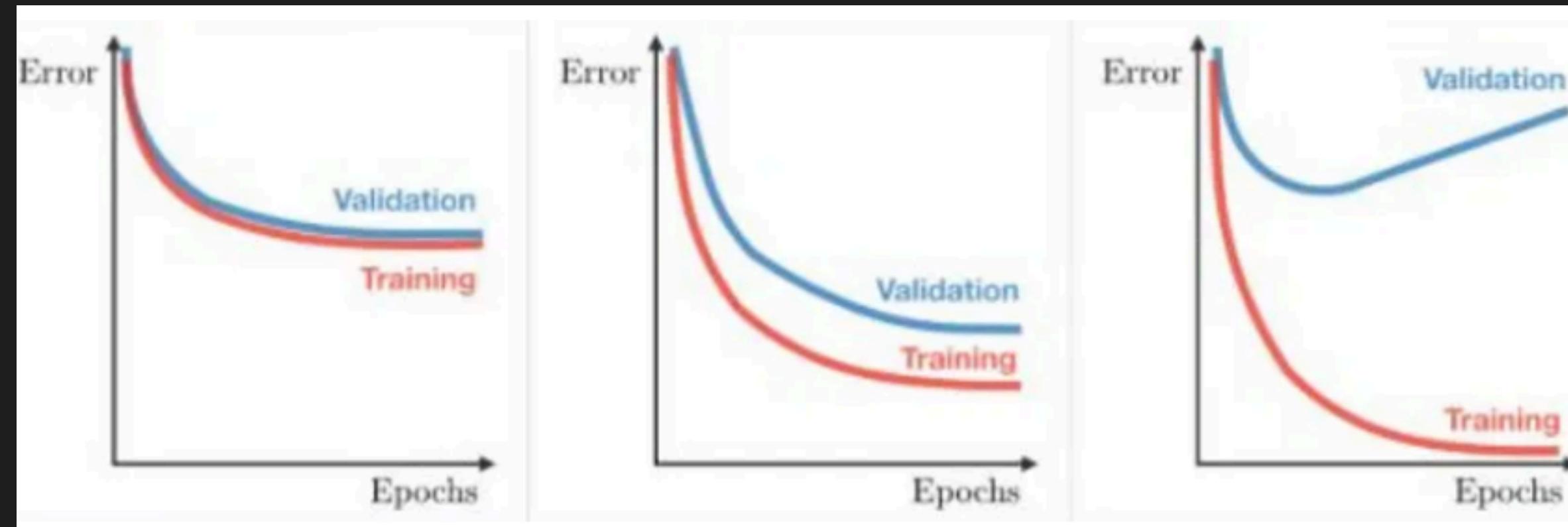
V- b) Entrainement du modèle et évaluation sur les données de validation

Analyse des courbes d'entraînement



V- b) Entrainement du modèle et évaluation sur les données de validation

Analyse des courbes d'entraînement



Underfitting

Good fitting

Overfitting

Si overfitting ou underfitting:

- 1- Appliquer un fine tuning
- 2- Réinitialiser le modèle
- 3- Lancer un nouveau entraînement

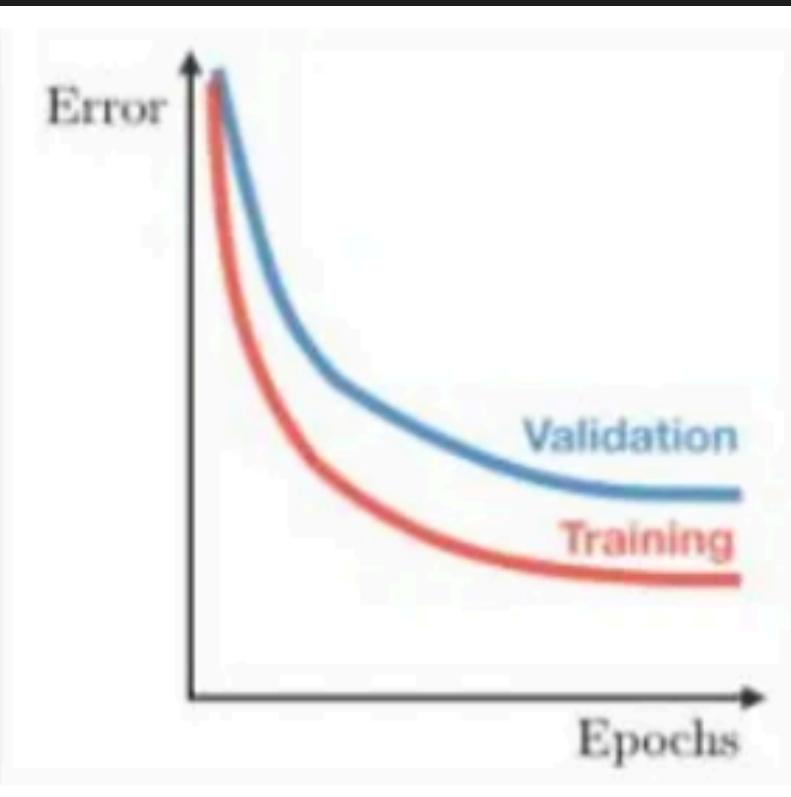
Fine tuning (ou réglage sans fin)

Ajustement des hyperparamètres
du modèle ou de l'entraînement

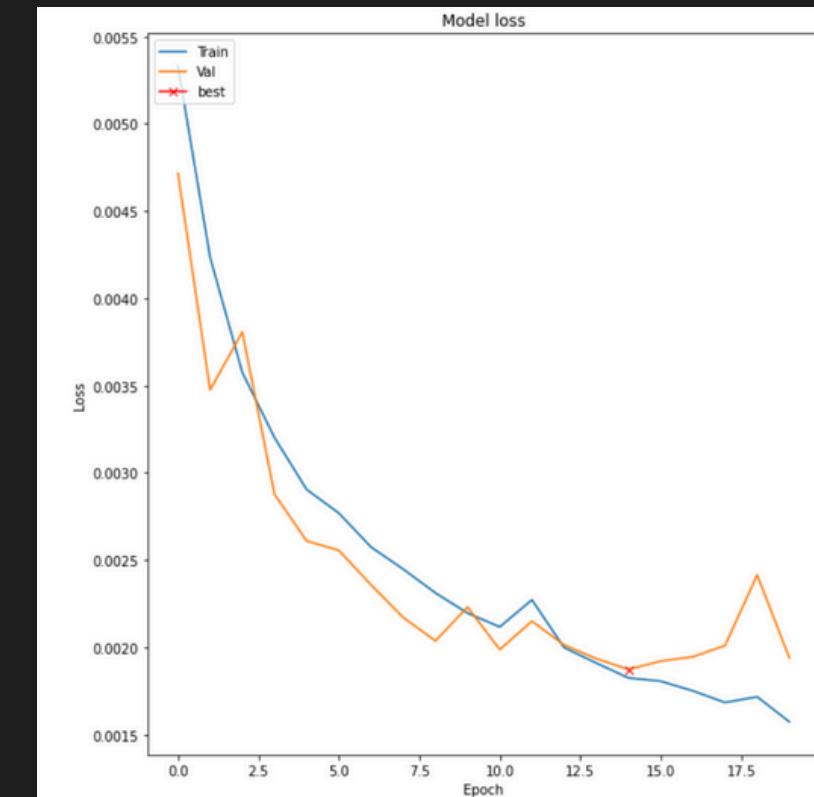
TP CN

V- b) Entrainement du modèle et évaluation sur les données de validation

Analyse des courbes d'entraînement



Good fitting



Exemple de Good fitting

Si overfitting ou underfitting:

- 1- Appliquer un fine tuning
- 2- Réinitialiser le modèle
- 3- Lancer un nouveau entraînement

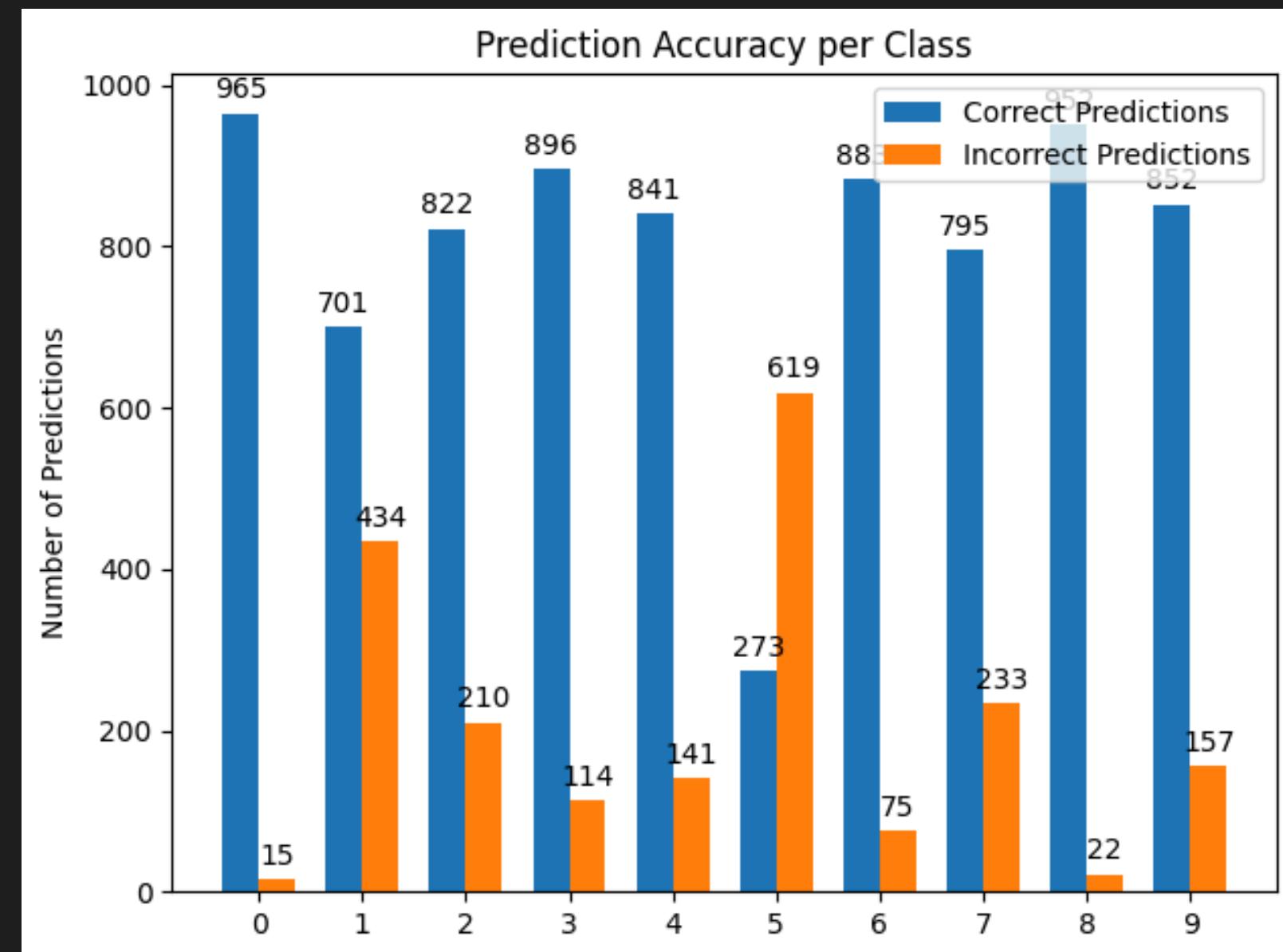
Fine tuning (ou réglage sans fin)

Ajustement des hyperparamètres
du modèle ou de l'entraînement

TP CN

VI - Evaluation sur les données test

```
313/313 ————— 5s 13ms/step - accuracy: 0.9817 - loss: 0.0780
{'accuracy': 0.9847999811172485, 'loss': 0.0603303462266922}
```



TP CN+DropOut

```
Pkeep=0.25
model = Sequential([
    Dense(200, input_shape=(784,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(100, input_shape=(200,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(60, input_shape=(100,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(30, input_shape=(60,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(10),
    Activation('softmax'),
])
```

TP CN+DropOut

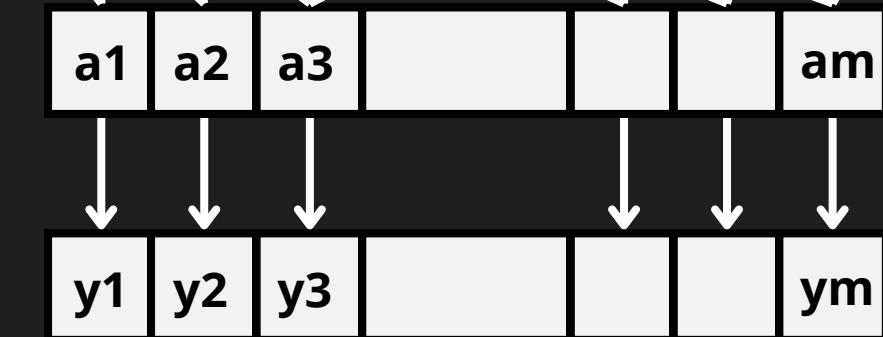
```
Pkeep=0.25  
model = Sequential([  
    Dense(200, input_shape=(784,)),  
    Activation('relu'),  
    Dropout(Pkeep),  
    Dense(100, input_shape=(200,)),  
    Activation('relu'),  
    Dropout(Pkeep),  
    Dense(60, input_shape=(100,)),  
    Activation('relu'),  
    Dropout(Pkeep),  
    Dense(30, input_shape=(60,)),  
    Activation('relu'),  
    Dropout(Pkeep),  
    Dense(10),  
    Activation('softmax'),  
])
```

Input:



Dense(200) :

Fonction
d'Activation ReLU:



Calcul de la fonction d'activation ReLU f

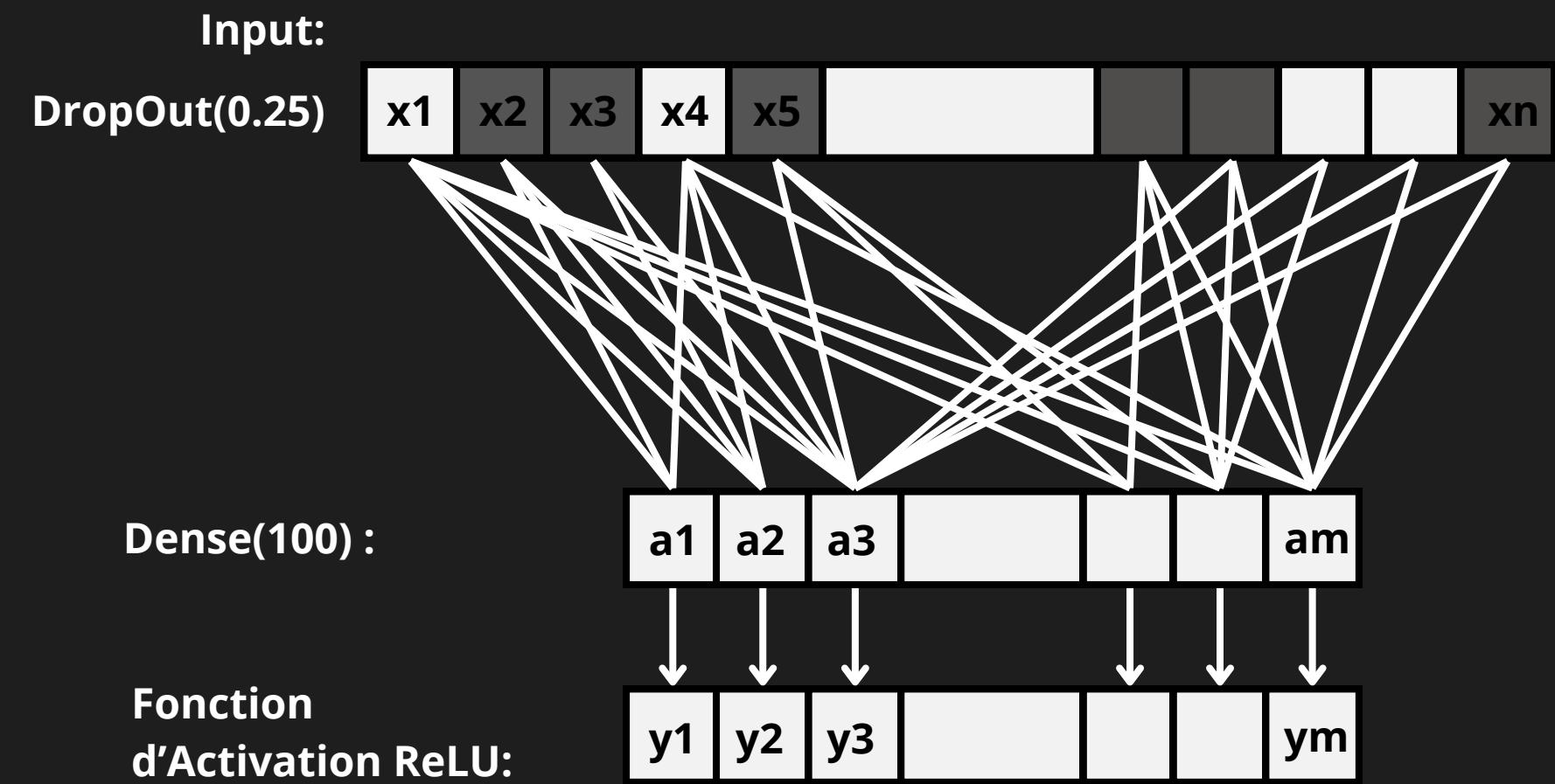
$$\forall k \in \{1, \dots, m\}, y_k = f \left(\sum_{i=1}^n x_i w_i + b \right)$$

avec $n = 784$ et $m = 200$

TP CN+DropOut

Pkeep=0.25

```
model = Sequential([
    Dense(200, input_shape=(784,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(100, input_shape=(200,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(60, input_shape=(100,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(30, input_shape=(60,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(10),
    Activation('softmax'),
])
```



$$\forall k \in \{1, \dots, m\}, y_k = f \left(\sum_{i=1}^n x_i w_i + b \right)$$

avec $n = 200$ et $m = 100$

TP CN+DropOut

```
Pkeep=0.25
model = Sequential([
    Dense(200, input_shape=(784,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(100, input_shape=(200,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(60, input_shape=(100,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(30, input_shape=(60,)),
    Activation('relu'),
    Dropout(Pkeep),
    Dense(10),
    Activation('softmax'),
])
```

TP CN+DropOut

V- a) Hyperparamètres de l'entraînement

Hyperparamètres intéressant du modèle:

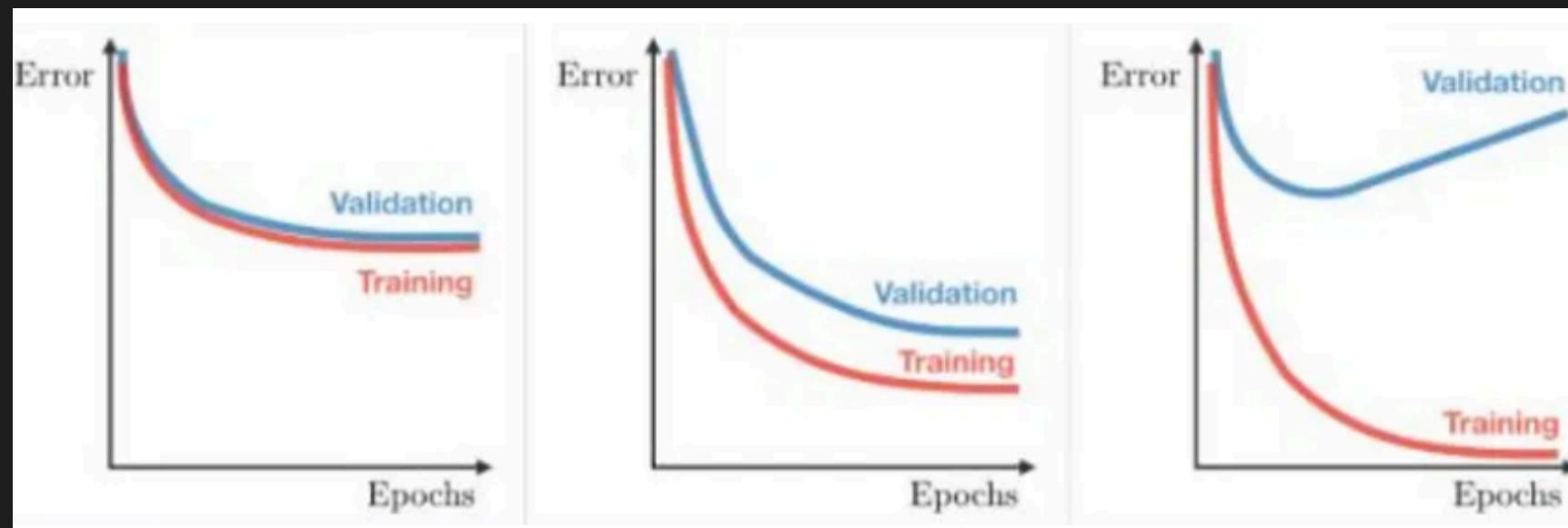
- DropOut

Hyperparamètres intéressant pour l'entraînement:

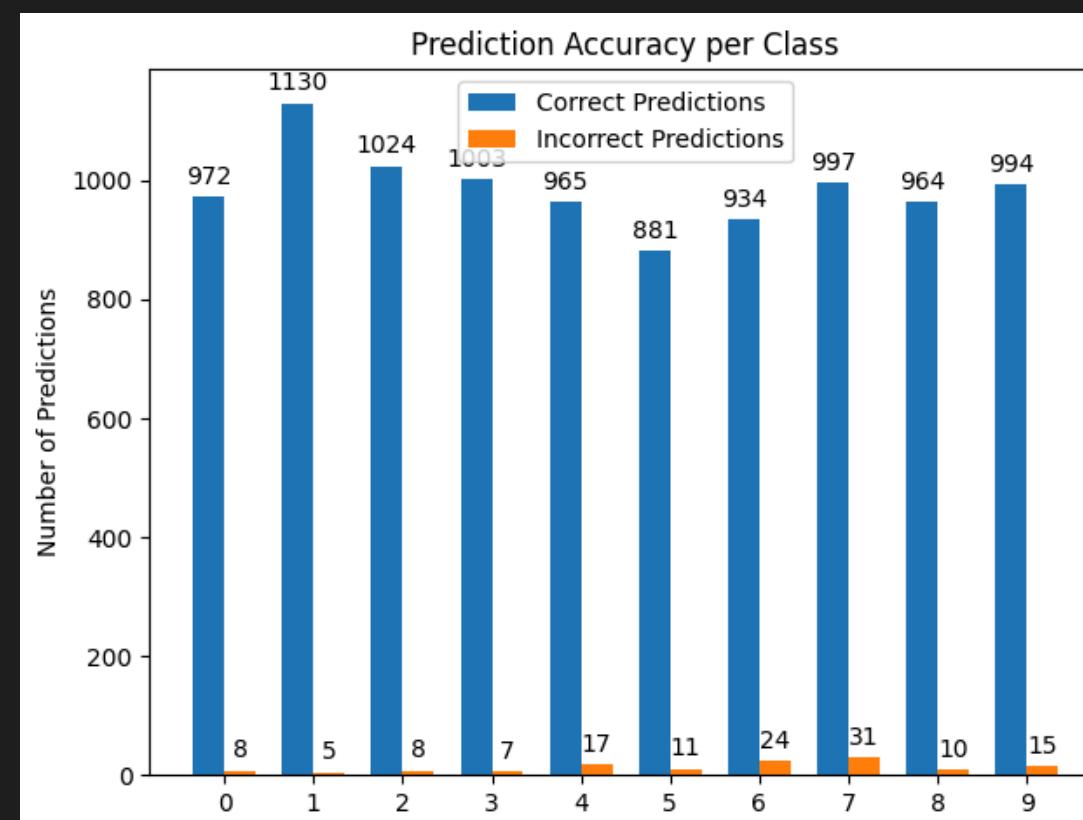
- Algorithme d'optimisation
- Fonction de perte
- Métrique
- EarlyStopping
- Save best model
- Batch size
- Epoch

TP CN+DropOut

V- b) Entrainement du modèle et évaluation sur les données de validation

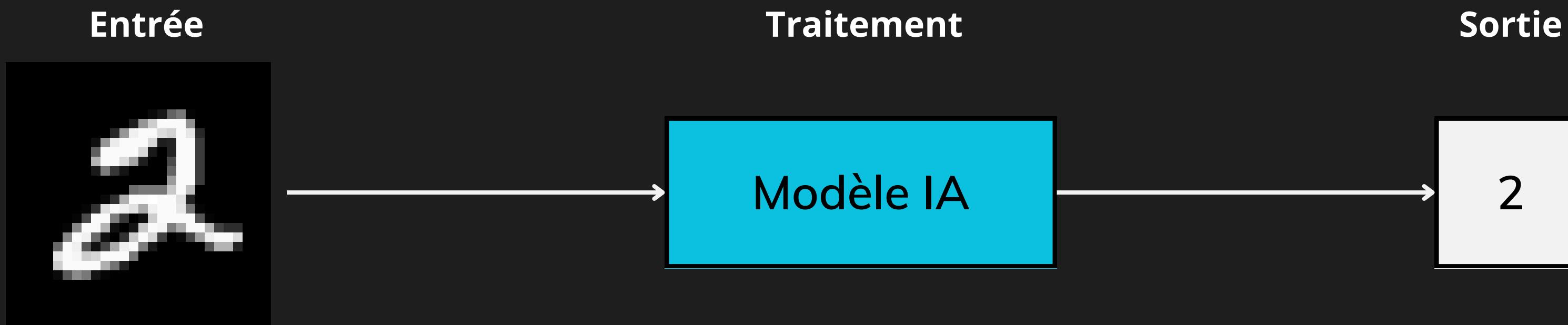


VI - Evaluation sur les données test



TP CNN

Objectif de ce TP : Concevoir un modèle de classification d'images



TP CNN

Prérequis : Importation des librairies et définition de fonctions utiles

```
# Visualisation display

def show_sample(X_train, Y_train):
    plt.figure(figsize=(5,5))
    plt.subplot(1,4,1)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,2)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,3)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])
    plt.subplot(1,4,4)
    index = random.randint(0, X_train.shape[0])
    plt.title(Y_train[index])
    plt.imshow(X_train[index])

def show_fx_negatif(X_test, Y_pred, Y_test):

    Y_pred_classes = np.argmax(Y_pred, axis=1)
    Y_true_classes = np.argmax(Y_test, axis=1)

    FX_NEGATIF = X_test[(Y_pred_classes!=Y_true_classes)]
    Pred_fx_negatif = Y_pred_classes[(Y_pred_classes!=Y_true_classes)]
    GT_fx_negatif = Y_true_classes[(Y_pred_classes!=Y_true_classes)]
```

TP CNN

I- Importation et visualisation des données

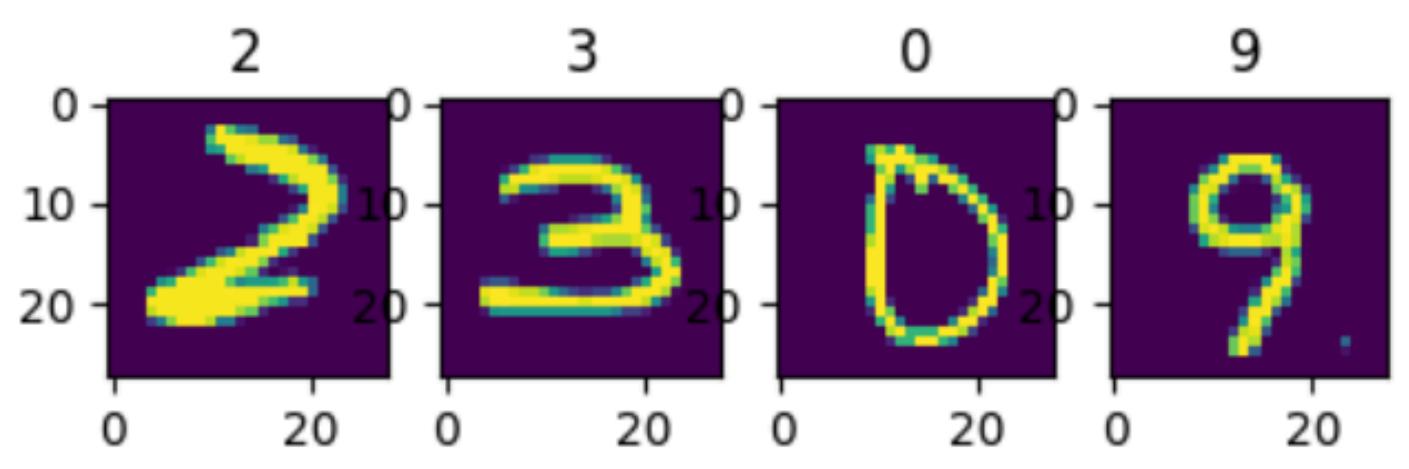
```
MNIST Dataset Shape (images, hauteur, largeur):
```

```
X_train: (60000, 28, 28)
```

```
Y_train: (60000,)
```

```
X_test: (10000, 28, 28)
```

```
Y_test: (10000,)
```



Train

Test



TP CNN

II - Division de la base de données en train, validation et test

```
Shape of X_train: (48000, 28, 28)
Shape of Y_train: (48000,)
Shape of X_val: (12000, 28, 28)
Shape of Y_val: (12000,)
Shape of X_test: (10000, 28, 28)
Shape of Y_test: (10000,)
```

TP CNN

III - Pré-traitement des données

a) Changement des dimensions pour adapter à l'entrée du CNN et mise en format float32

```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_val = X_val.reshape(X_val.shape[0], 28, 28, 1).astype('float32') #add :
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

b) Normalisation des images pour adapter à l'entrée du CNN

```
X_train /= 255 # nois
X_val /= 255 # normalis
X_test /= 255 # noise
```

c) Changement dans le codage de chaque label de sortie pour adapter à la sortie du CNN

```
# one hot encode outputs
Y_train = to_categorical(Y_train)
Y_val = to_categorical(Y_val)
Y_test = to_categorical(Y_test)
```

Après	1	Avant
100000000		2
010000000		3
001000000		3
...
000000001		9

TP CNN

V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)), # 32 different 3x3 kernels -- so 32 feature maps  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2 kernel  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernels  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kernels  
  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-length vector  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement pendant l'entraînement pour éviter le surapprentissage  
    Dense(10),  
    Activation('softmax'),  
])
```

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

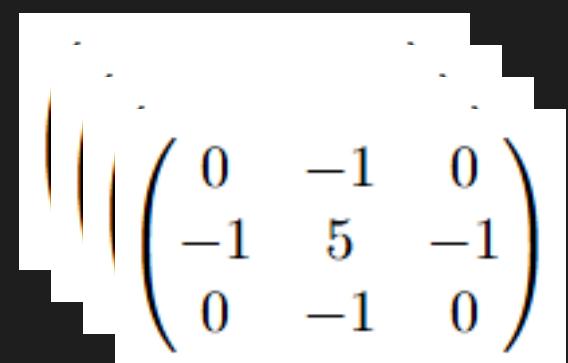
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

On a:

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Une matrice de taille 5x5 (pour l'exemple)



32 filtres

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

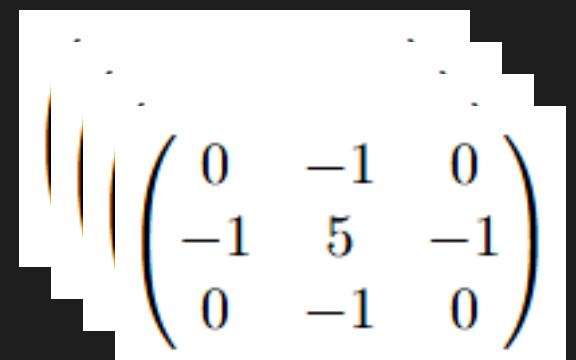
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

On a:

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Une matrice de taille 5x5 (pour l'exemple)



32 filtres

Explication du calcul du réseau pour un filtre et la matrice d'entrée

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ \boxed{-0.5} & -0.3 & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & \boxed{-0.3} & -0.2 \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & \boxed{-0.2} \end{pmatrix}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \boxed{\begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}}$$

$$= I \qquad \qquad \qquad = K \qquad \qquad \qquad = O$$

$$O(i, j) = \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} I(i + m, j + n) \cdot K(m, n)$$

Calcul exécuté :

- Déplacement du filtre avec un pas de 1 : `stride = 1`
- Calcul sans tenir compte des bords de la matrice : `padding = 'valid'`

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
```

```
# Convolution Layer 2
Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
MaxPooling2D(pool_size=(2, 2)),
# Convolution Layer 3
Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
```

```
# Fully Connected Layer 4
Dense(512), # 512 FCN nodes
Activation('relu'),
Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
Dense(10),
Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}$$

stride = 1 & padding = 'valid'

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -0.8 & 2.5 & 1.2 & 0 \\ 0 & 3.3 & 1.2 & -1.2 & 0 \\ 0 & -0.5 & -0.3 & -0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

stride = 1 & padding = 'same'

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K$$

1- Convolution entre la matrice d'entrée et le filtre

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = \boxed{\begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}}$$

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K + b$$

2- Ajout du biais

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Ici, le biais égale à 0 par défaut initialement

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$I * K + b$$

2- Ajout du biais

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 = \boxed{\begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix}}$$

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kernel
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

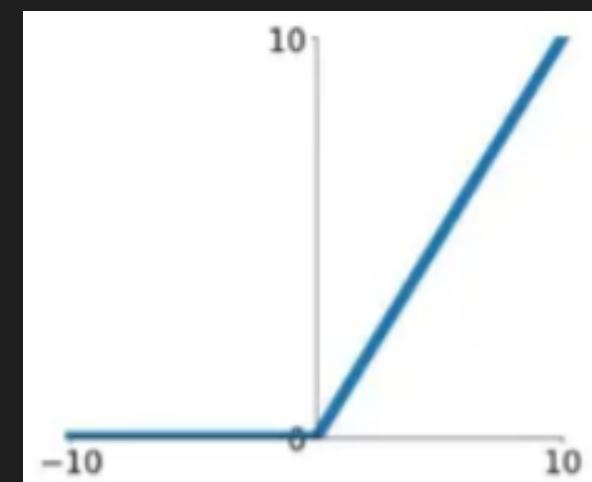
Calcul du réseau de neurone

$$f(I * K + b)$$

3- Fonction d'activation

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = f \left(\begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix} \right)$$

$$f(x) = \max(0, x)$$



TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kernel
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

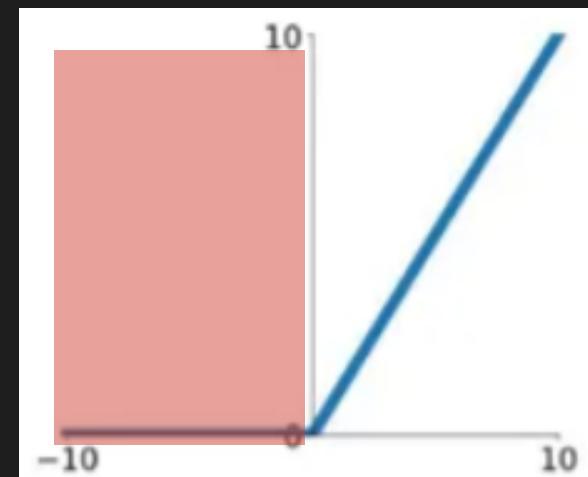
Calcul du réseau de neurone

$$f(I * K + b)$$

3- Fonction d'activation

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = f \left(\begin{pmatrix} -0.8 & 2.5 & 1.2 \\ 3.3 & 1.2 & -1.2 \\ -0.5 & -0.3 & -0.2 \end{pmatrix} \right)$$

$$f(x) = \max(0, x)$$



TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kernel
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
```

Fully Connected Layer 4

```
Dense(512), # 512 FCN nodes
Activation('relu'),
Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
Dense(10),
Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

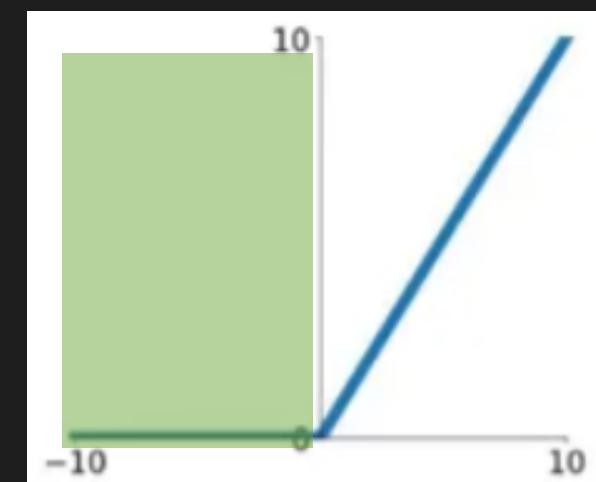
Calcul du réseau de neurone

$$y = f(I * K + b)$$

3- Fonction d'activation

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$f(x) = \max(0, x)$$



TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kernel
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

3- Fonction d'activation

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \boxed{\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}}$$

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$Y[i, j] = \max(X[m, n] \text{ pour } m \in [i, i+2], n \in [j, j+2])$$

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} =$$

$$Y =$$

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \left(\quad \right)$$



Prendre le Max dans cette fenêtre

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3.3 & & \\ & & \end{pmatrix}$$



Prendre le Max dans cette fenêtre

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3.3 & 2.5 \end{pmatrix}$$



Prendre le Max dans cette fenêtre

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3.3 & 2.5 \\ 3.3 & 0 \end{pmatrix}$$



Prendre le Max dans cette fenêtre

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3.3 & 2.5 \\ 3.3 & 1.2 \end{pmatrix}$$



Prendre le Max dans cette fenêtre

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2
    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),
    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Calcul du réseau de neurone

$$y = f(I * K + b)$$

$$f \left(\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix} * \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} + O_3 \right) = \begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Application du MaxPooling sur une fenêtre 2 x 2

$$\begin{pmatrix} 0 & 2.5 & 1.2 \\ 3.3 & 1.2 & 0 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 3.3 & 2.5 \\ 3.3 & 1.2 \end{pmatrix}$$



Prendre le Max dans cette fenêtre

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

Résumé du calcul du réseau

Input:

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$
$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

1- Calcul du réseau de neurone

$$y = f(I * K + b)$$

2-Application du MaxPooling sur une fenêtre 2 x 2

Output:

$$\begin{pmatrix} 3.3 & 2.5 \\ 3.3 & 1.2 \end{pmatrix}$$

Carte de caractéristique (ou feature map)

TP CNN

V- Définition du modèle

```
model = Sequential([
    # Convolution Layer 1
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

    # Convolution Layer 2
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
    MaxPooling2D(pool_size=(2, 2)),

    # Convolution Layer 3
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern

    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len

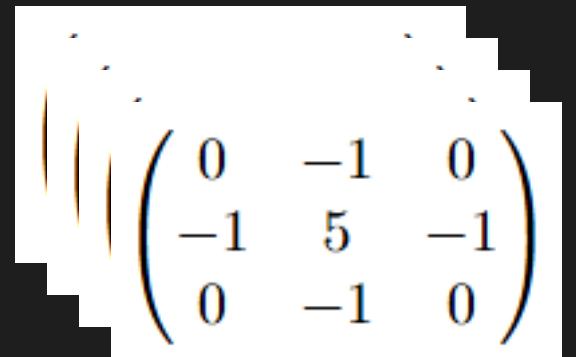
    # Fully Connected Layer 4
    Dense(512), # 512 FCN nodes
    Activation('relu'),
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement
    Dense(10),
    Activation('softmax'),
```

Conclusion pour le calcul d'une couche

Input:

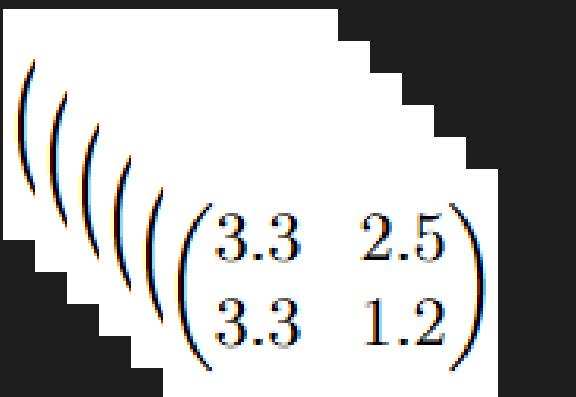
$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.2 & 0.2 \\ 0.2 & 0.3 & 0.8 & 0.5 & 0.3 \\ 0.1 & 0.9 & 0.6 & 0 & 0.1 \\ 0.4 & 0.2 & 0.1 & 0 & 0 \\ 0.5 & 0.1 & 0 & 0.1 & 0.3 \end{pmatrix}$$

Une matrice de taille 5x5 (pour l'exemple)



32 filtres

Output:



32 cartes caractéristiques

TP CNN

V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),
```

Dans la couche 2 et 3 :

- Augmentation du nombre de filtres pour extraire des caractéristiques plus complexes

Sortie de la couche 3:

- Matrice de cartes caractéristiques de taille 7x7x128

TP CNN

V- Définition du modèle

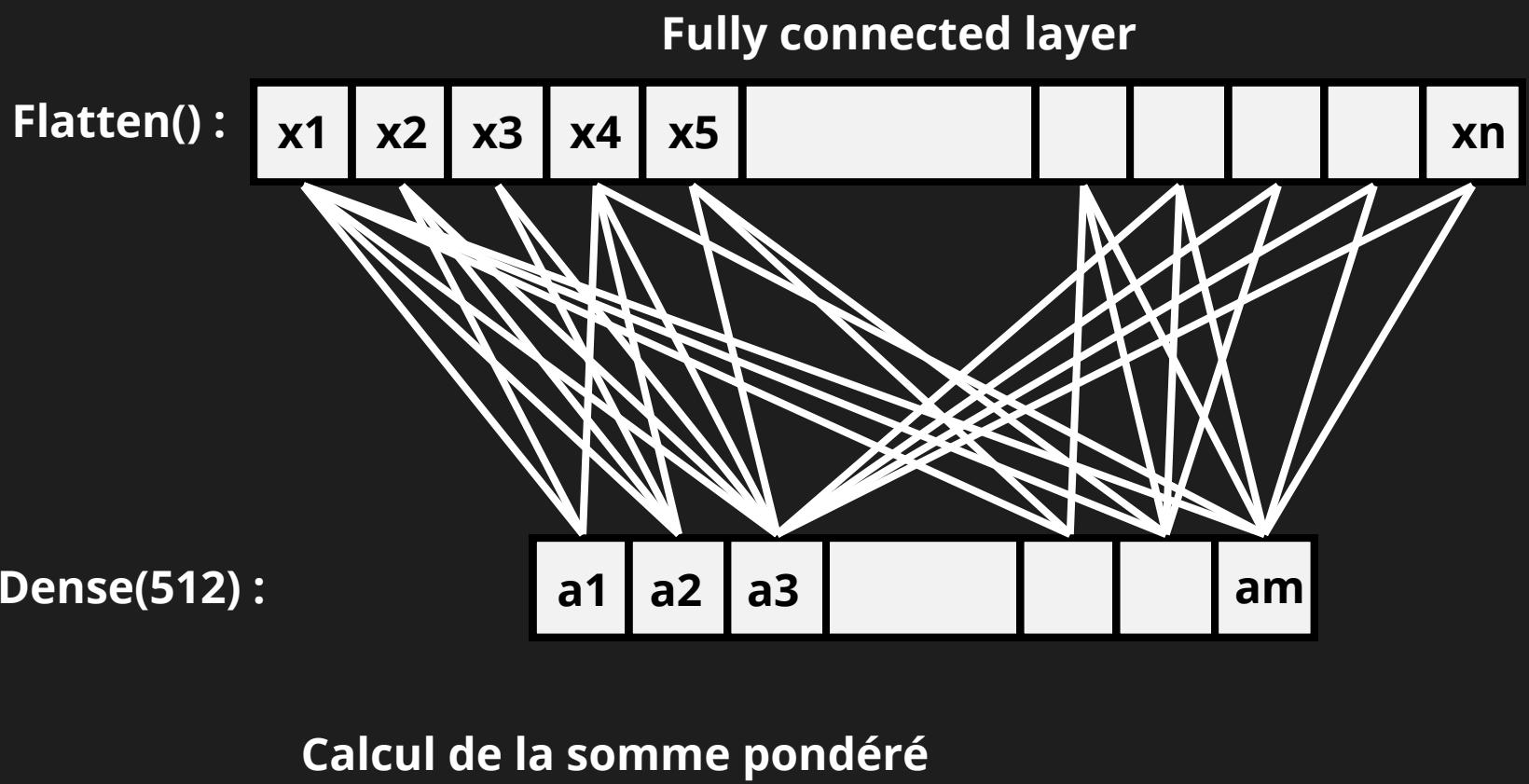
```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),
```

Rendre la matrice 7x7x128 en un vecteur de taille 1x6272

TP CNN

V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),  
])
```



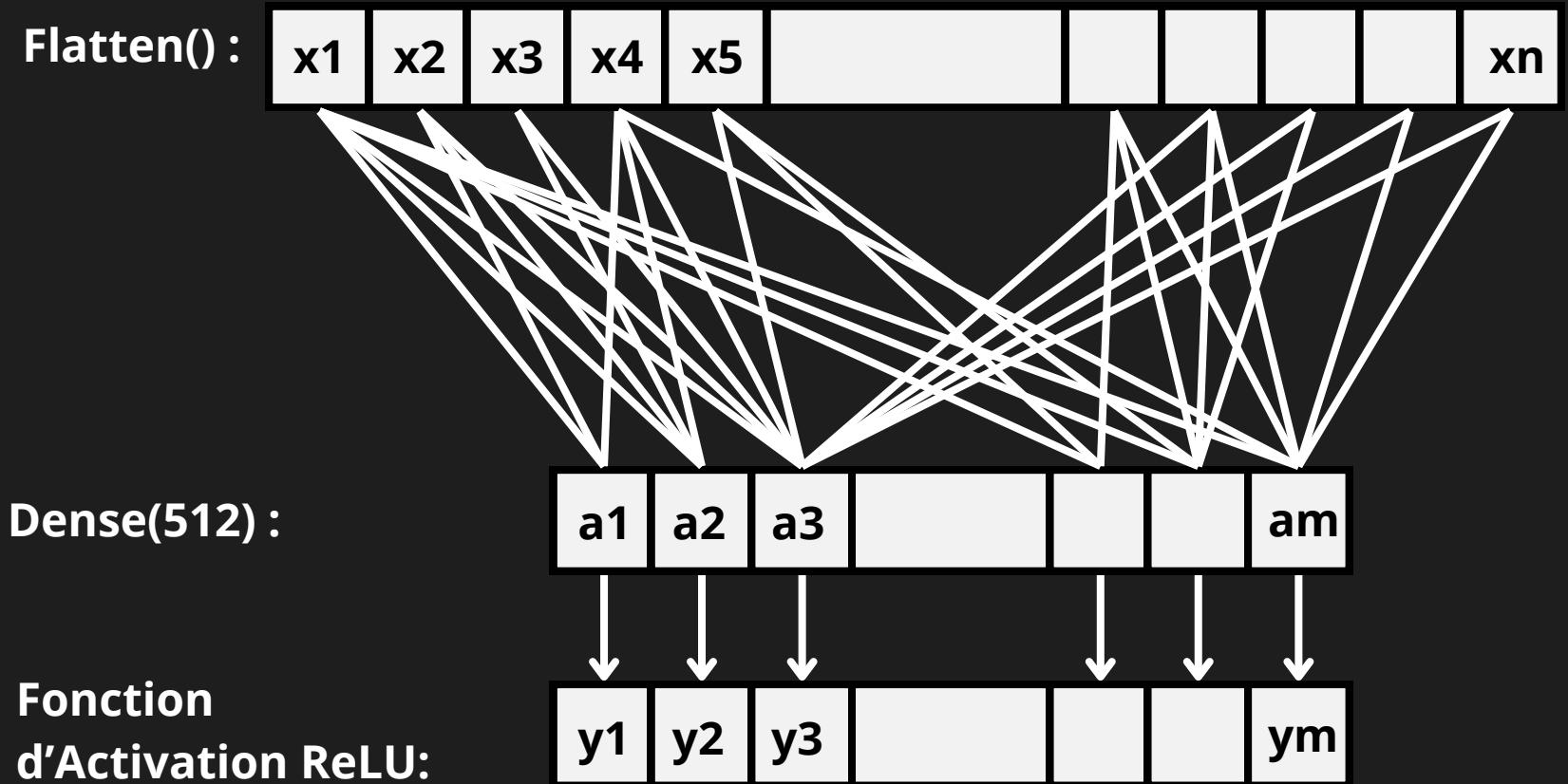
$$\forall k \in \{1, \dots, m\}, a_k = \sum_{i=1}^n x_i w_i + b$$

avec $m = 512$ et $n = 6272$

TP CNN

V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),  
])
```



Calcul de la fonction d'activation ReLU f

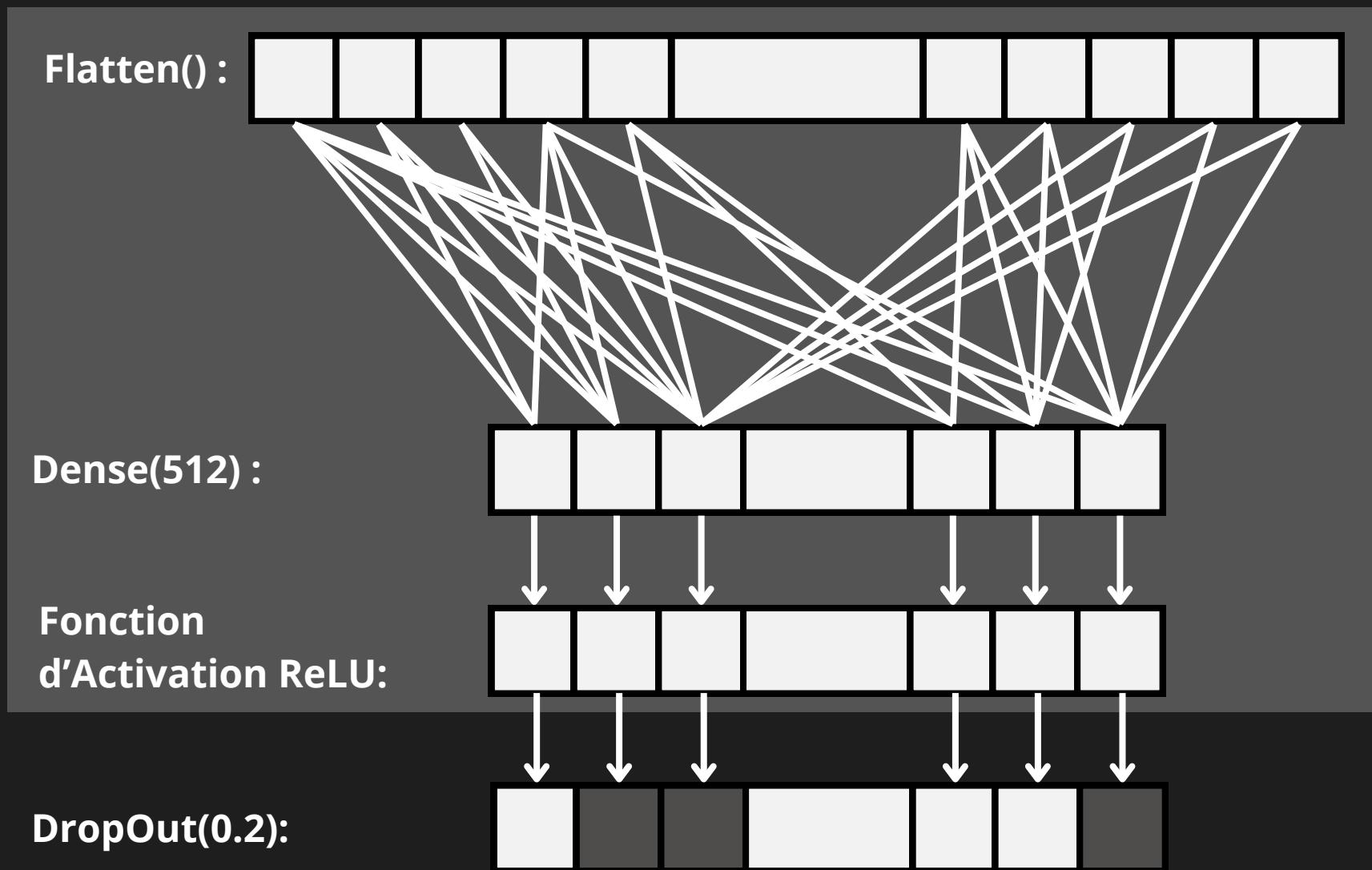
$$\forall k \in \{1, \dots, m\}, y_k = f \left(\sum_{i=1}^n x_i w_i + b \right)$$

avec $m = 512$ et $n = 6272$

TP CNN

V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),  
])
```

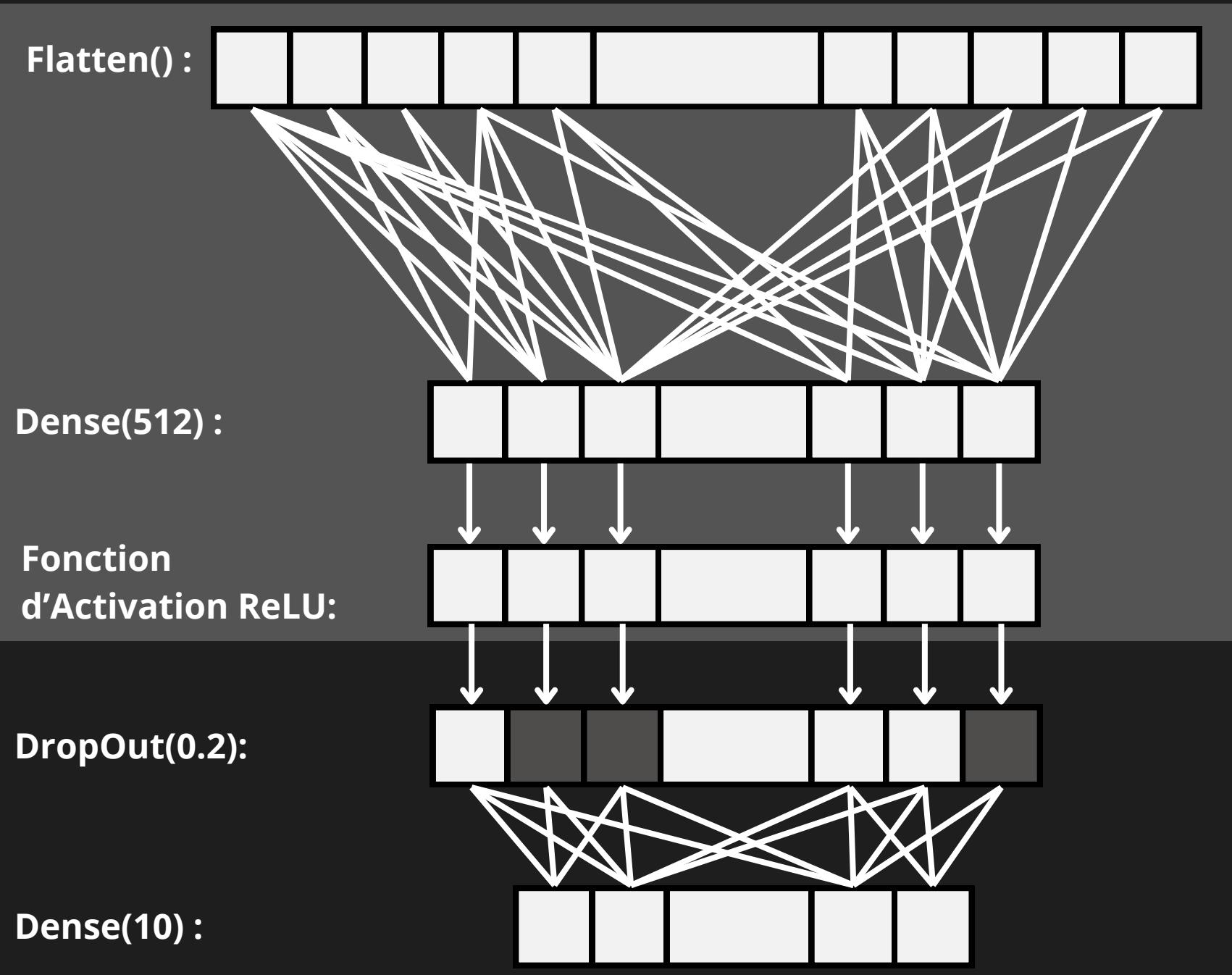


Réduction du risque de surapprentissage en gélant 20% des neurones aléatoirement

TP CNN

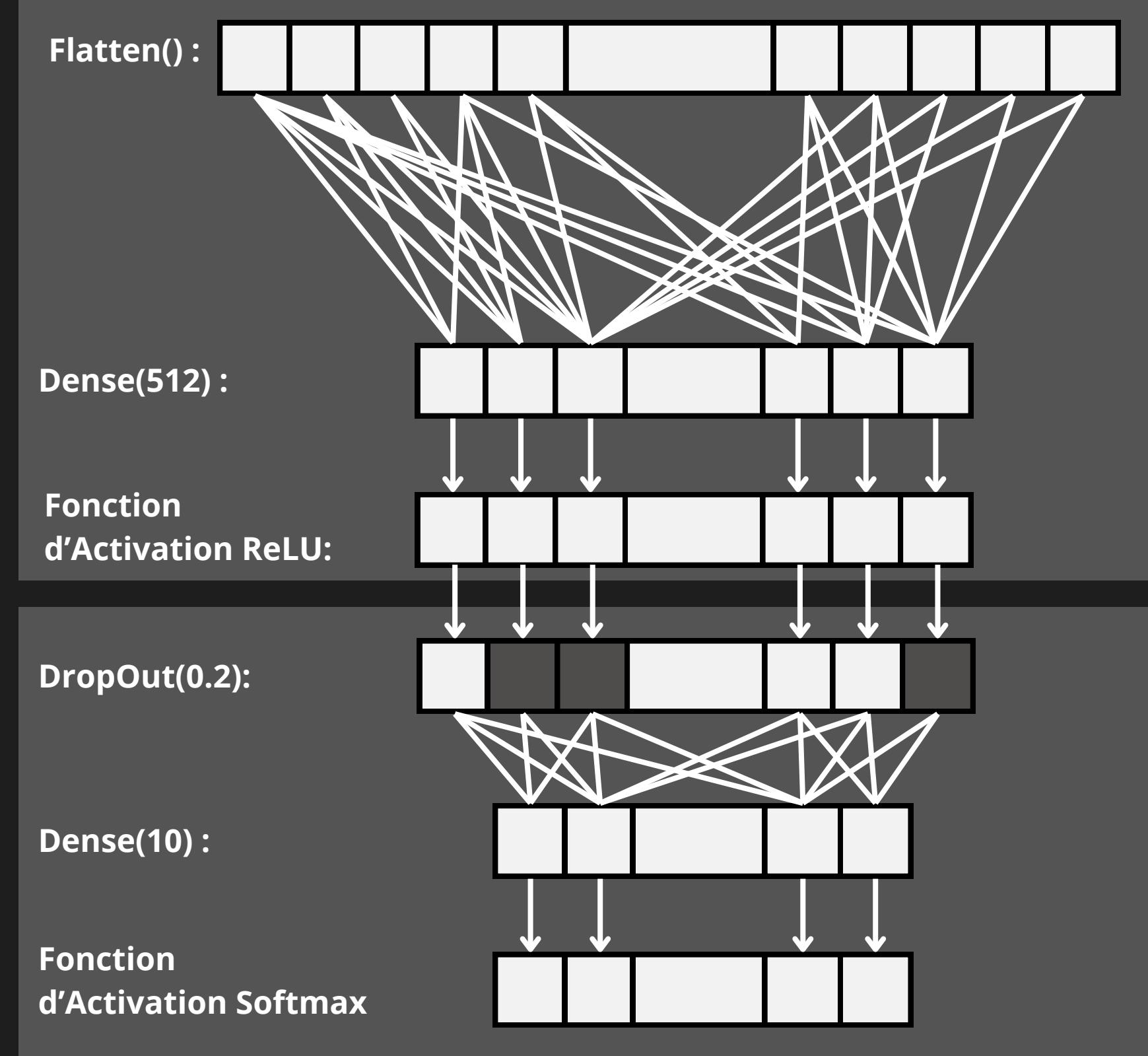
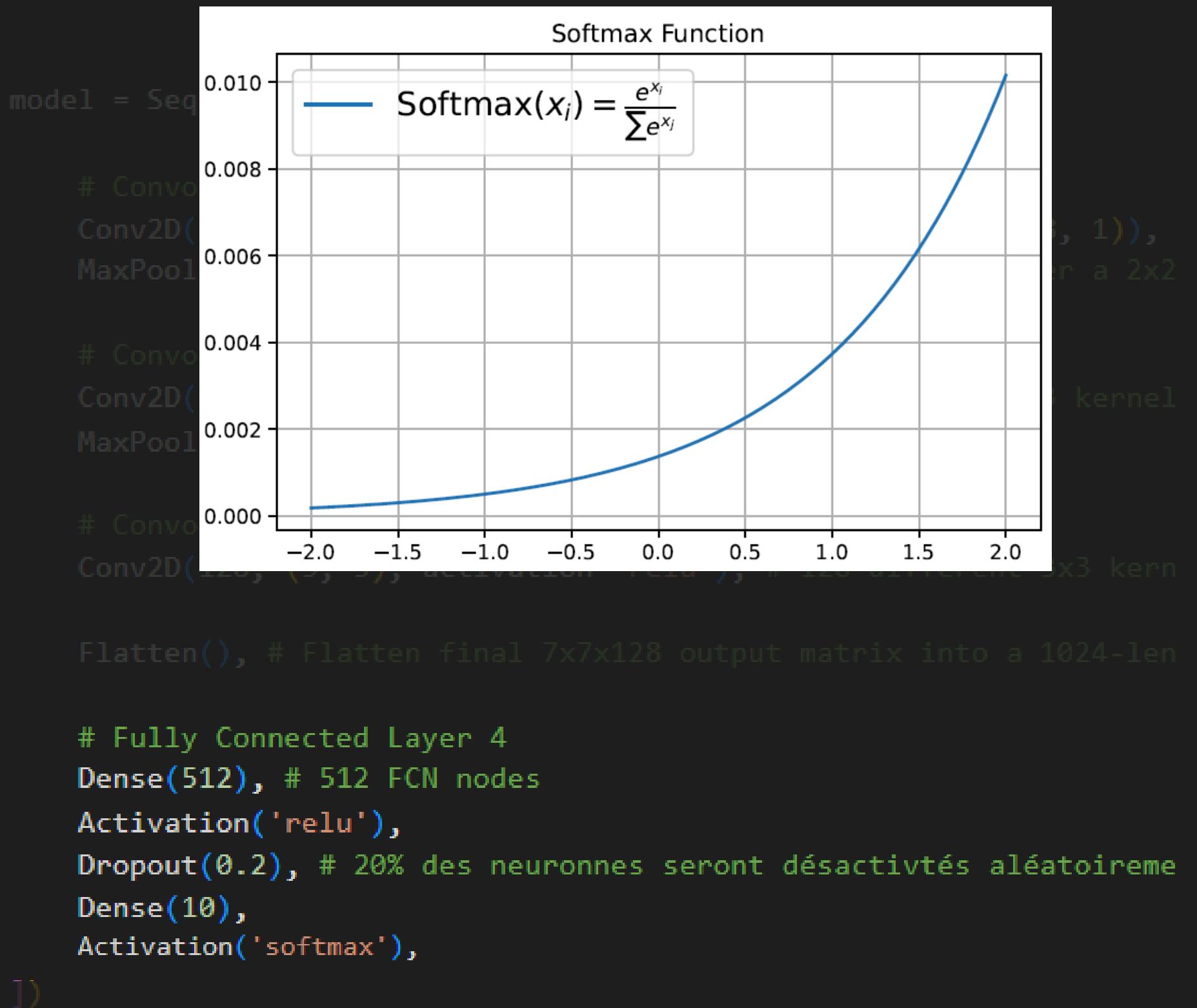
V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),  
])
```



TP CNN

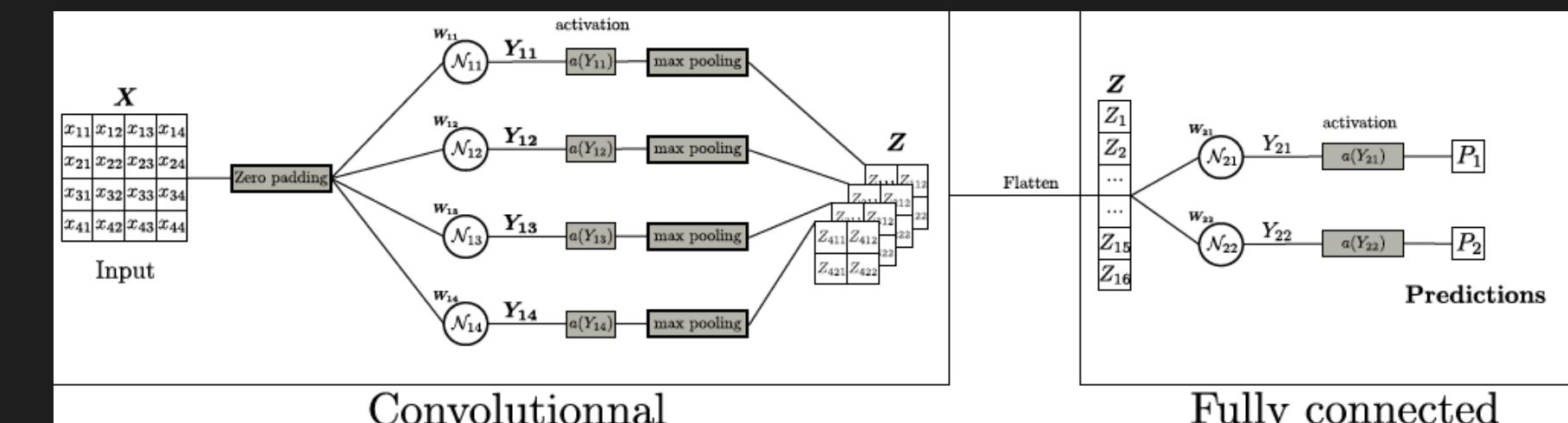
V- Définition du modèle



TP CNN

V- Définition du modèle

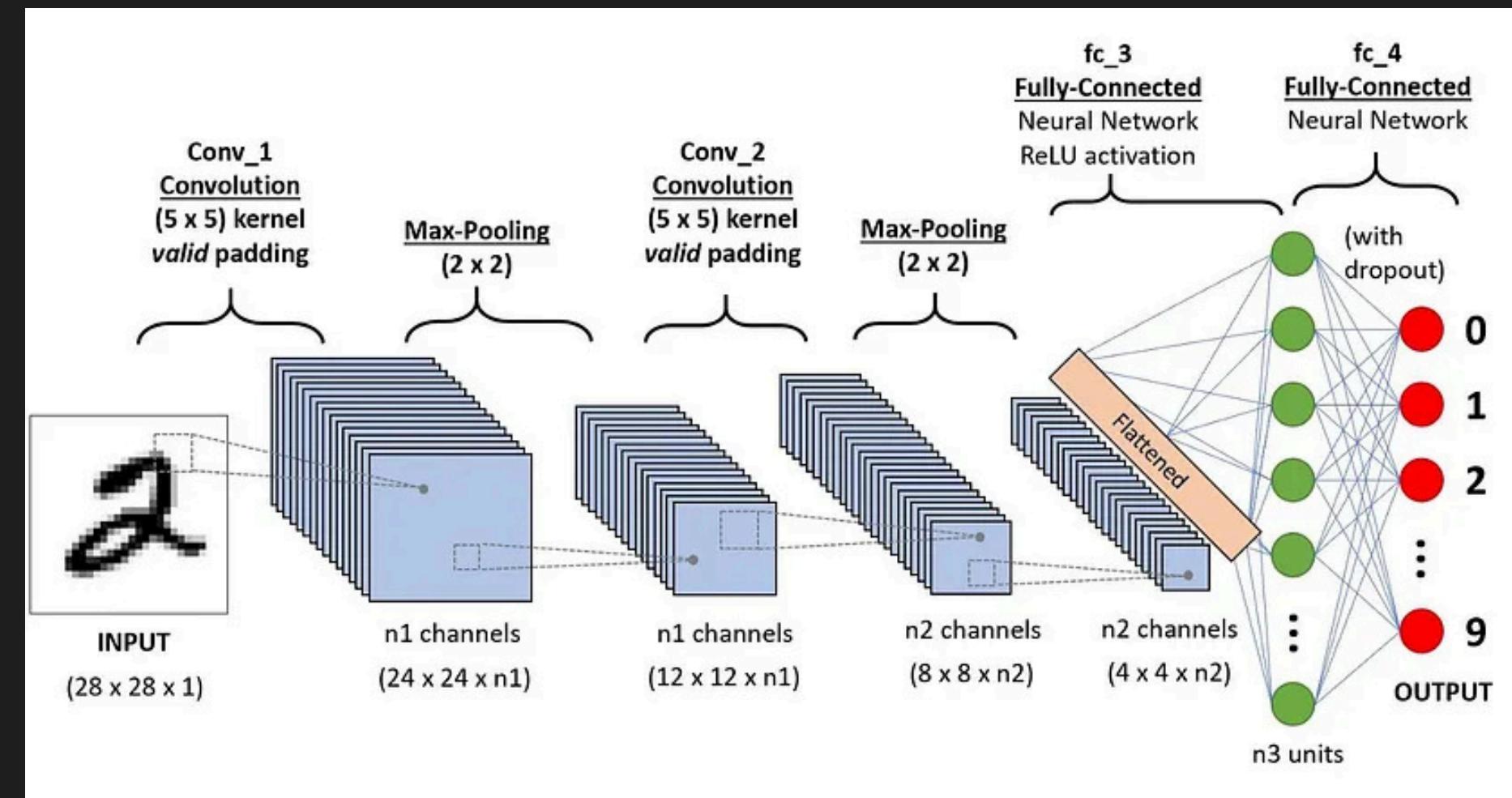
```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),  
])
```



TP CNN

V- Définition du modèle

```
model = Sequential([  
  
    # Convolution Layer 1  
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),  
    MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2  
  
    # Convolution Layer 2  
    Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel  
    MaxPooling2D(pool_size=(2, 2)),  
  
    # Convolution Layer 3  
    Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern  
    Flatten(), # Flatten final 7x7x128 output matrix into a 1024-len  
  
    # Fully Connected Layer 4  
    Dense(512), # 512 FCN nodes  
    Activation('relu'),  
    Dropout(0.2), # 20% des neurones seront désactivés aléatoirement  
    Dense(10),  
    Activation('softmax'),  
])
```



TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- **Nombre de filtres**

```
# Convolution Layer 1
Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
MaxPooling2D(pool_size=(2, 2)), # Pool the max values over a 2x2

# Convolution Layer 2
Conv2D(64, (3, 3), activation='relu'), # 64 different 3x3 kernel
MaxPooling2D(pool_size=(2, 2)),

# Convolution Layer 3
Conv2D(128, (3, 3), activation='relu'), # 128 different 3x3 kern
```

Nombre de filtres:

- **Couche 1 : 32**
- **Couche 2 : 64**
- **Couche 3 : 128**

Si vous voulez multiplier par x filtres:

- **Couche 1 : 32 * x**
- **Couche 2 : 64 * x**
- **Couche 3 : 128 * x**

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

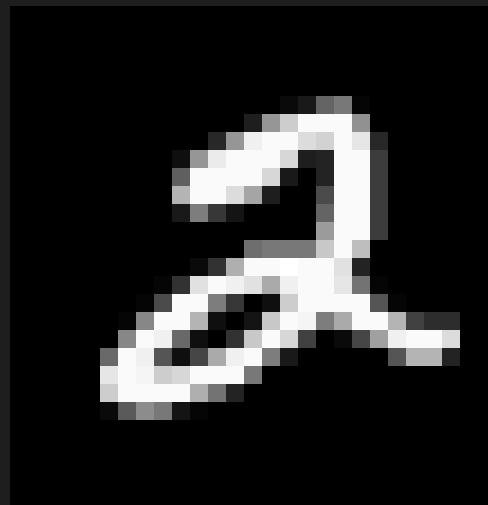
Hyperparamètres intéressant pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Fonction de perte

Entrée



Modèle IA

Sortie

0.1
0.2
...
0.05
0.05

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

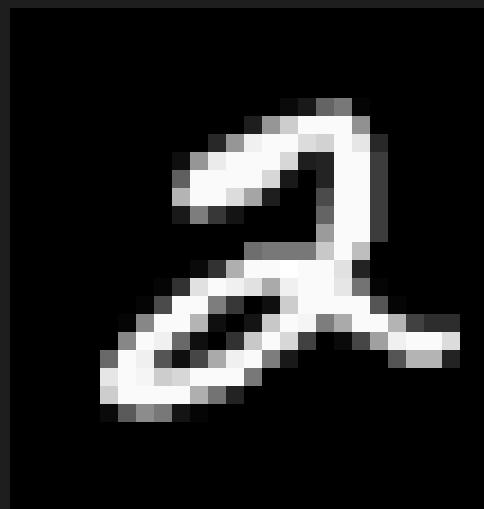
Hyperparamètres intéressant pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Fonction de perte

Entrée



Modèle IA

Sortie

0.1
0.2
...
0.05
0.05

Vérité terrain

0
1
...
0
0

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressants du modèle:

- Nombre de filtres

Hyperparamètres intéressants pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Prédit	Vérité terrain
0.1	0
0.2	1
...	...
0.05	0
0.05	0

Fonction de perte Categorical cross entropy

$$CE = - \sum_{i=1}^N y_i \log (\hat{y}_i)$$

avec :

- y_i le label vérité terrain $i : 0$ ou 1
- \hat{y}_i le label prédit $i : une probabilité entre 0 et 1.$

CE = 1.609

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Métrique
Accuracy

$$Accuracy = \frac{TP}{TP + TN + FN + FP}$$

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

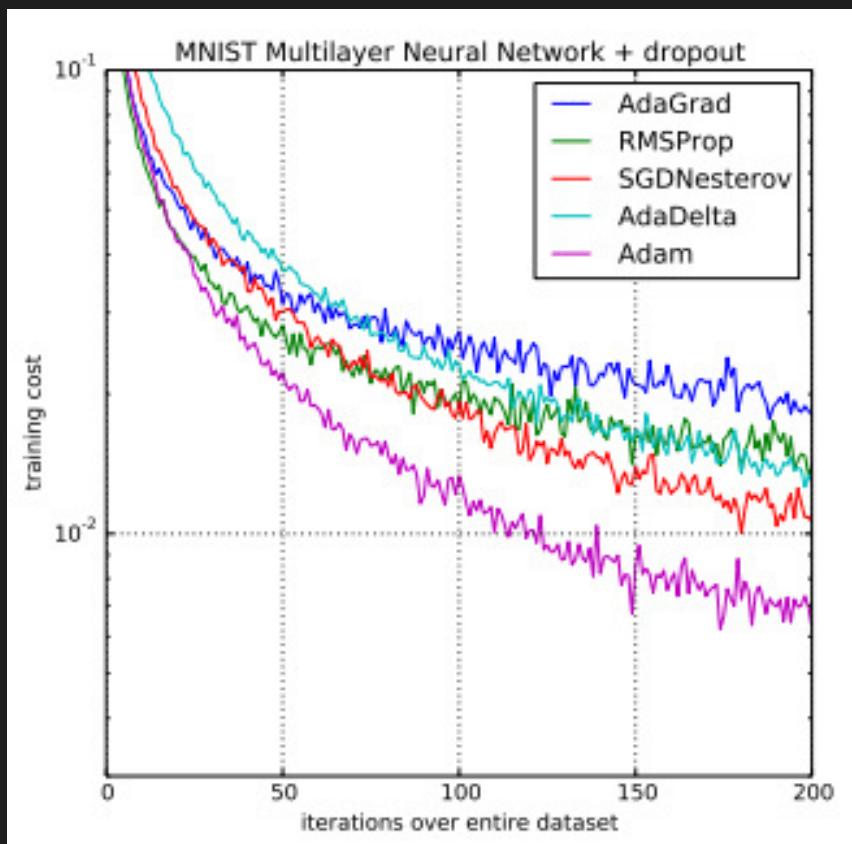
Hyperparamètres intéressant pour l'entraînement :

- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Algorithme d'optimisation

Adam



TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement :

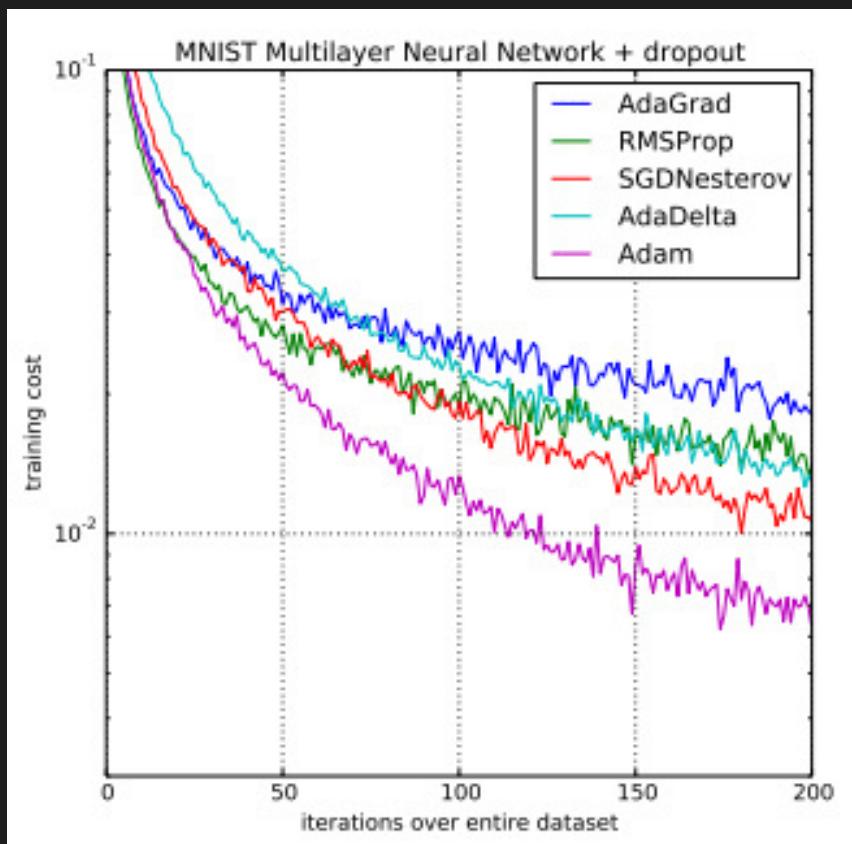
- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Algorithme d'optimisation

Adam

Hyperparamètre intéressant : Learning rate



TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressants du modèle:

- Nombre de filtres

Hyperparamètres intéressants pour l'entraînement :

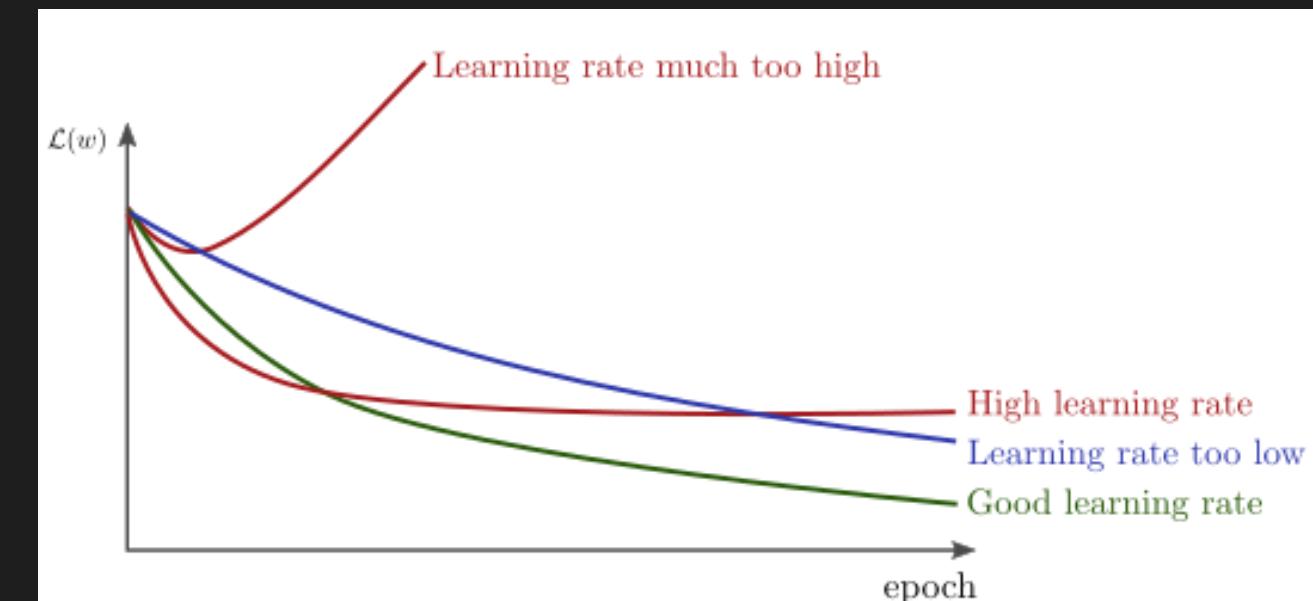
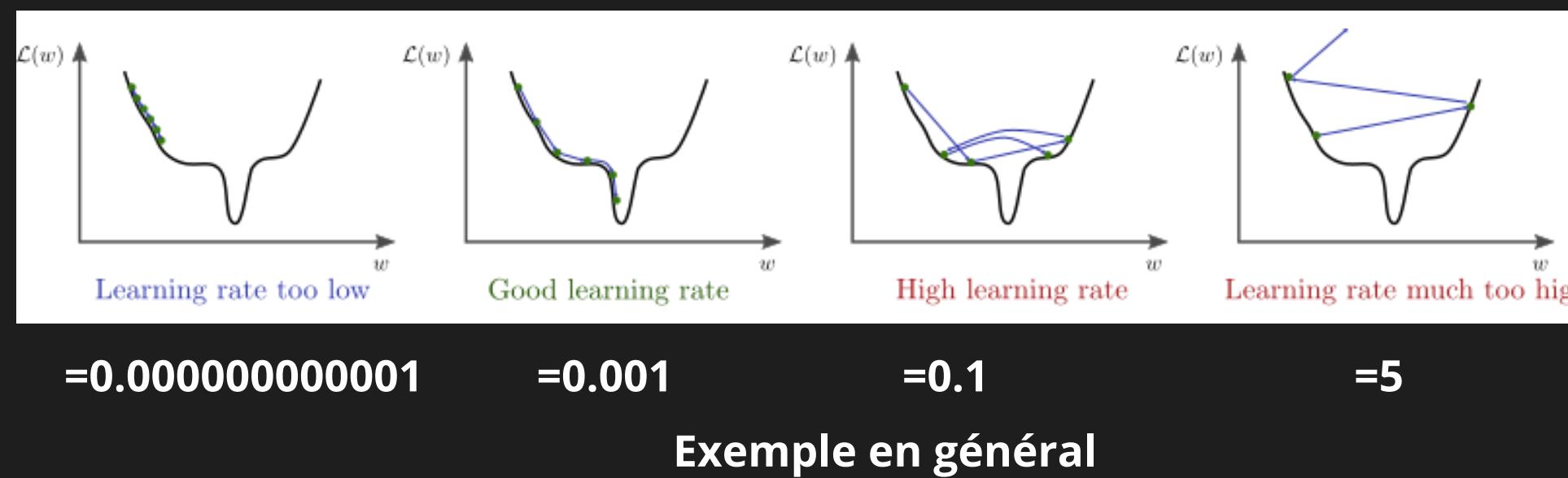
- Fonction de perte
- Métrique
- Algorithme d'optimisation

```
learning_rate = 0.001 #@param {type:"number"}  
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)  
model.compile(optimizer=optimizer,  
              loss='categorical_crossentropy',  
              metrics=['accuracy'])
```

Algorithme d'optimisation

Adam

Hyperparamètre intéressant : Learning rate



TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Fonction de perte
- Métrique
- Algorithme d'optimisation
- EarlyStopping
- Save best model

```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose=  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

Sauvegarde du meilleur modèle détenant la plus petite valeur d'erreur

Early stopping

Arrêt de l'entraînement si aucune amélioration de la fonction de perte sur les données de validation après 5 epoch

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique
- EarlyStopping
- Save best model

```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose=  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

Sauvegarde du meilleur modèle détenant la plus petite valeur d'erreur

Early stopping

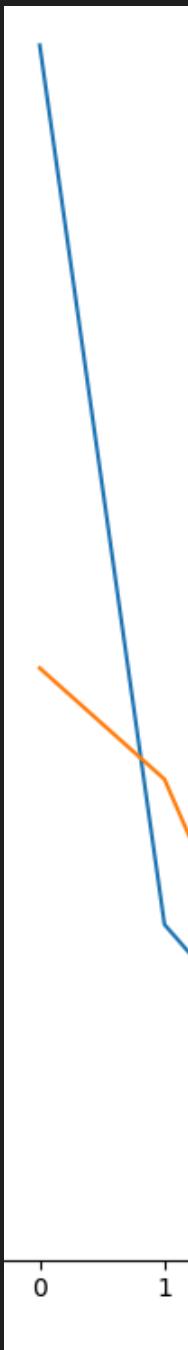
Arrêt de l'entraînement si aucune amélioration après 5 epoch où se situe la plus petite valeur d'erreur

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressants du modèle:

- Nombre de filtres



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:  
  
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose:  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

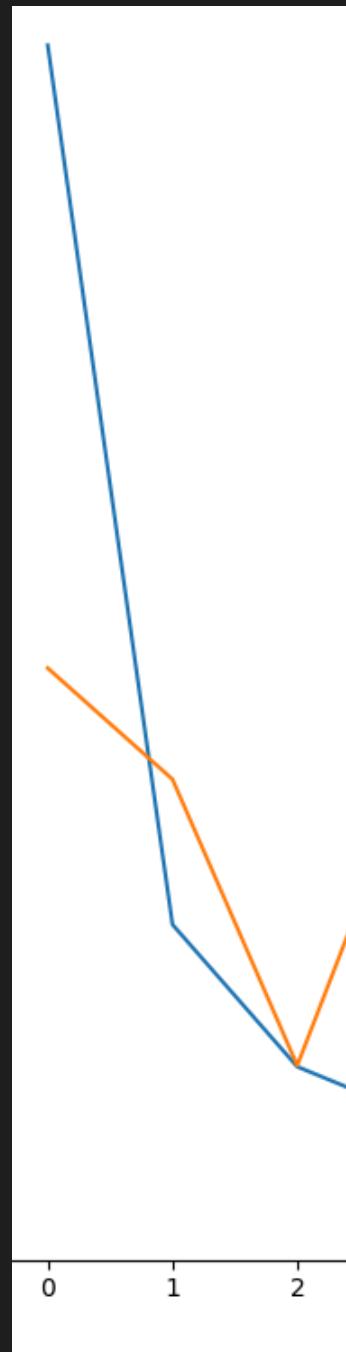
Early stopping

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressants du modèle:

- Nombre de filtres



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:  
  
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose:  
    save_weights_only=False, mode='auto') #Callback
```

Save best model

Epoch 2 : Plus petite erreur -> Sauvegarde du modèle

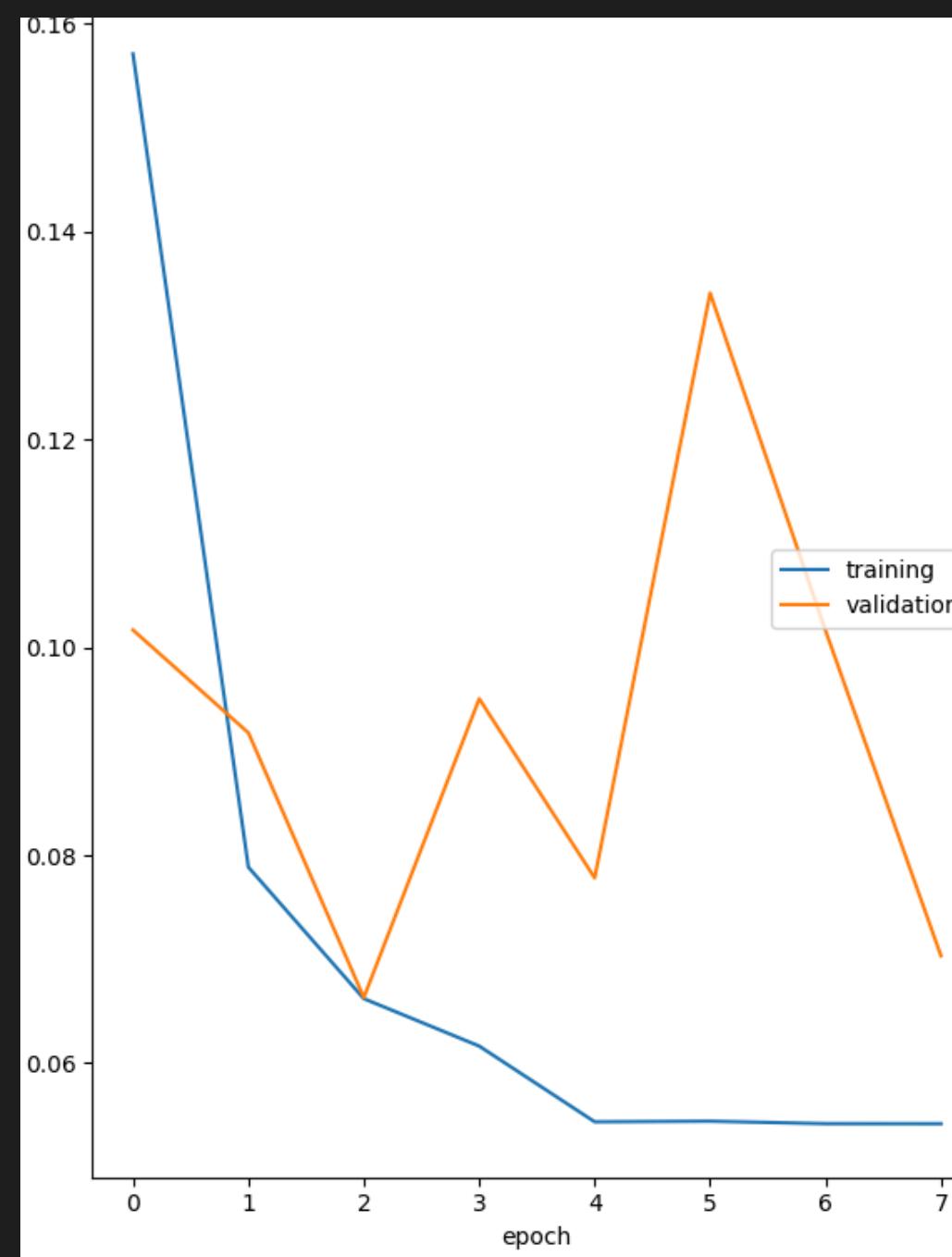
Early stopping

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressants du modèle:

- Nombre de filtres



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:  
    'auto')
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose:  
    1, save_weights_only=False, mode='auto') #Callback
```

Save best model

Epoch 2 : Plus petite erreur -> Sauvegarde du modèle

Early stopping

Epoch 2 : Plus petite erreur

Epoch 3-4-5-6-7 : Pas d'erreur plus petite que l'epoch 2



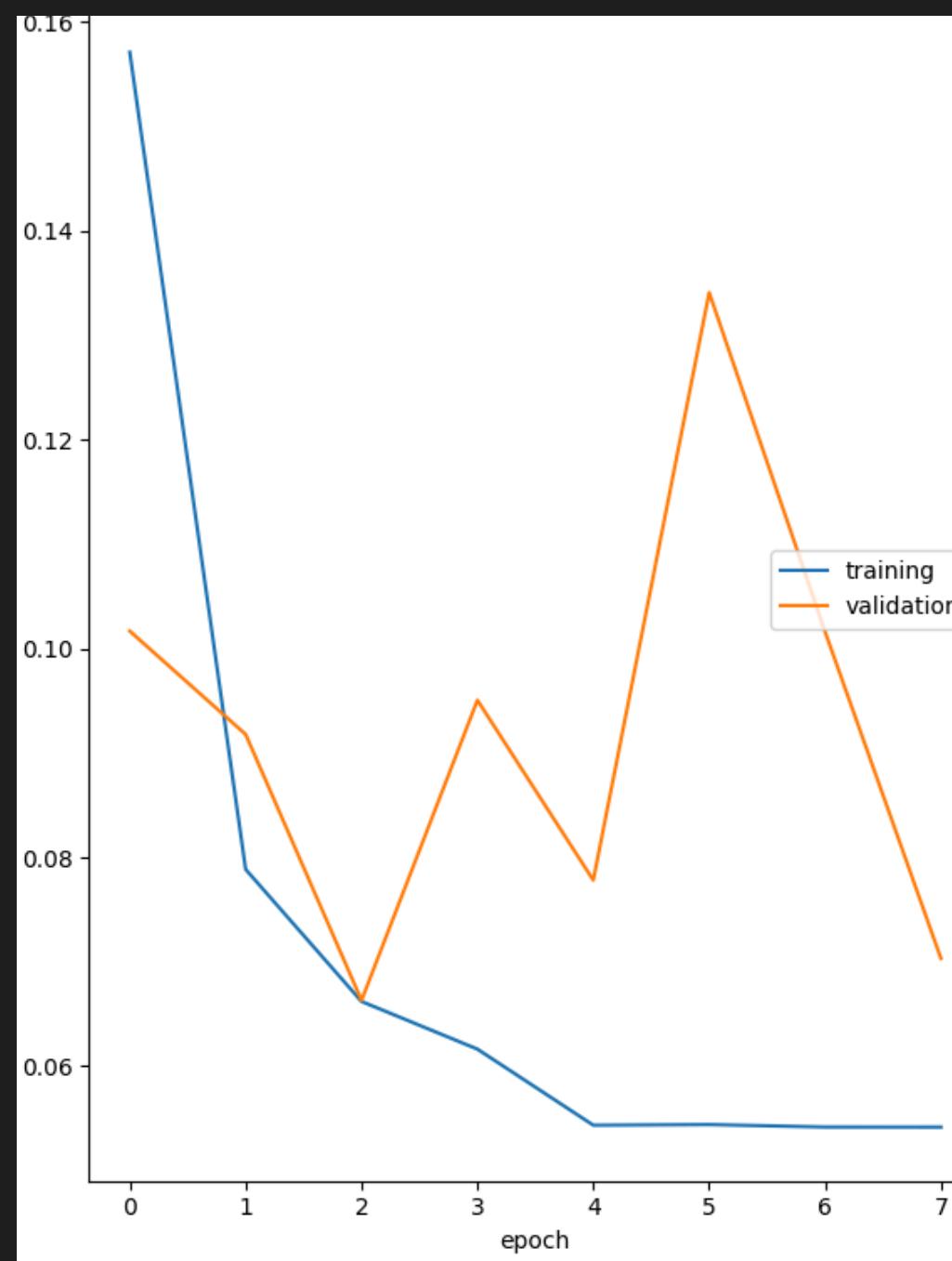
Début Overfitting

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressants du modèle:

- Nombre de filtres



```
early_stop = tf.keras.callbacks.EarlyStopping(  
    monitor='val_loss', patience=5, verbose=1, mode:  
    'auto')
```

```
checkpoint_filepath = '/content/best_model.keras'  
Model_check = tf.keras.callbacks.ModelCheckpoint(  
    checkpoint_filepath, monitor='val_loss', verbose:  
    1, save_weights_only=False, mode='auto') #Callback
```

Save best model

Epoch 2 : Plus petite erreur -> Sauvegarde du modèle

Early stopping

Epoch 2 : Plus petite erreur

Epoch 3-4-5-6-7 : Pas d'erreur plus petite que l'epoch 2

Entrainement : Stop

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique
- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,
           batch_size=2, epochs=1000,
           validation_split=0.3,
           verbose=1,
           callbacks=[plotlossesdeeper,early_stop,Model_check])
```

- Batch size
- Epoch

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

- Batch size
- Epoch

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```

Dataset

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

- Nombre de filtres

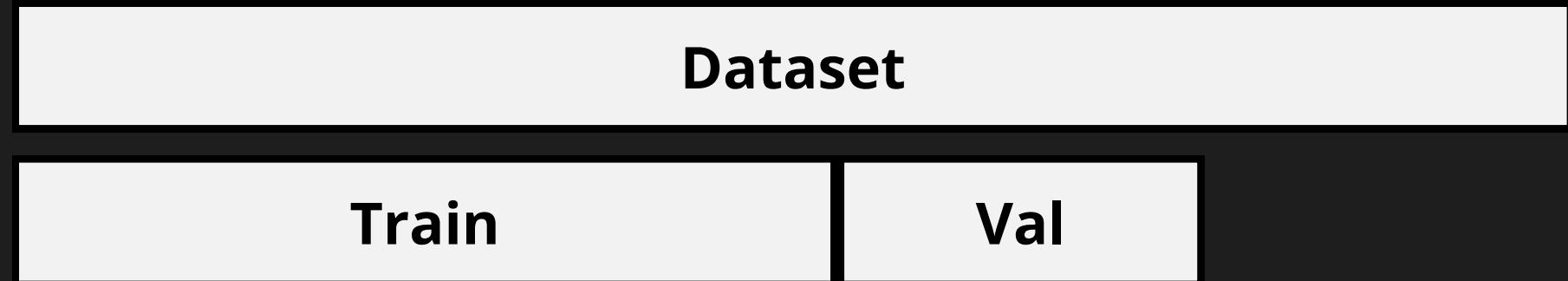
Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

- Batch size
- Epoch

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

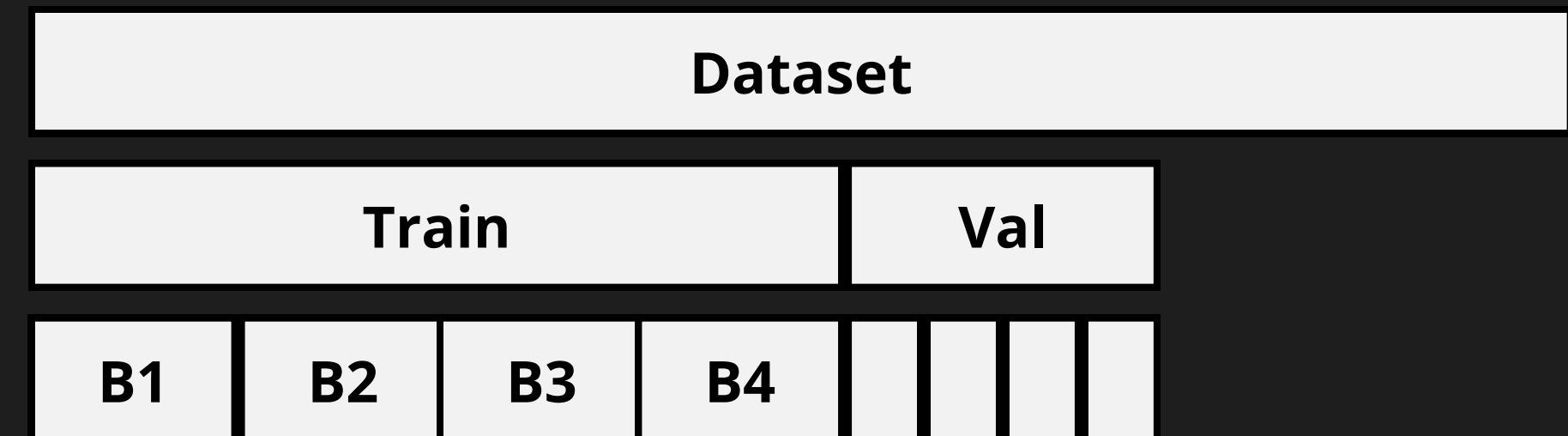
- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

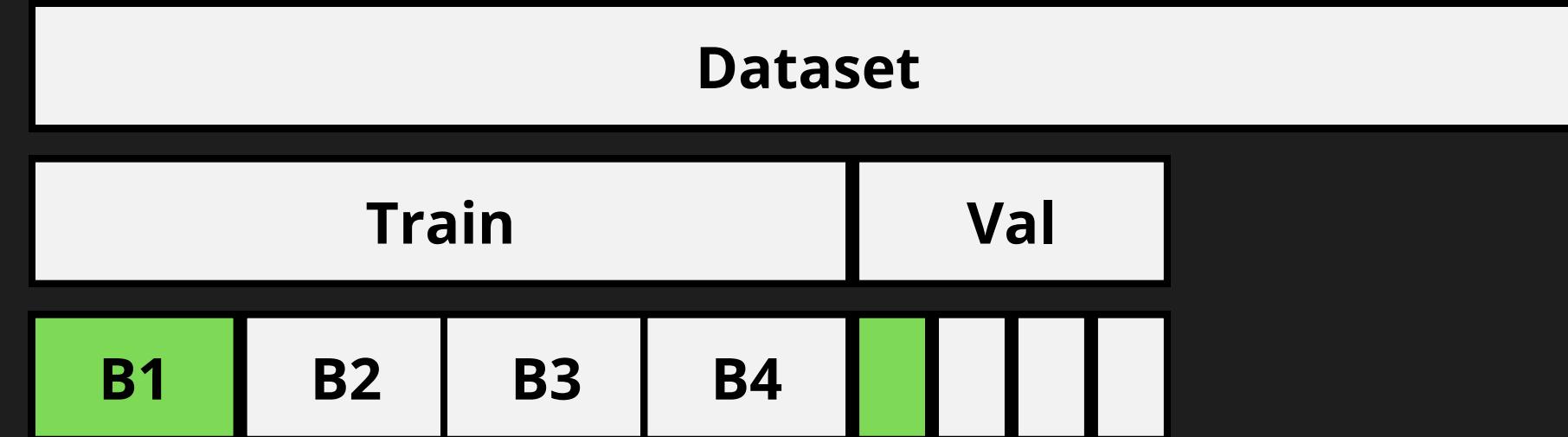
- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

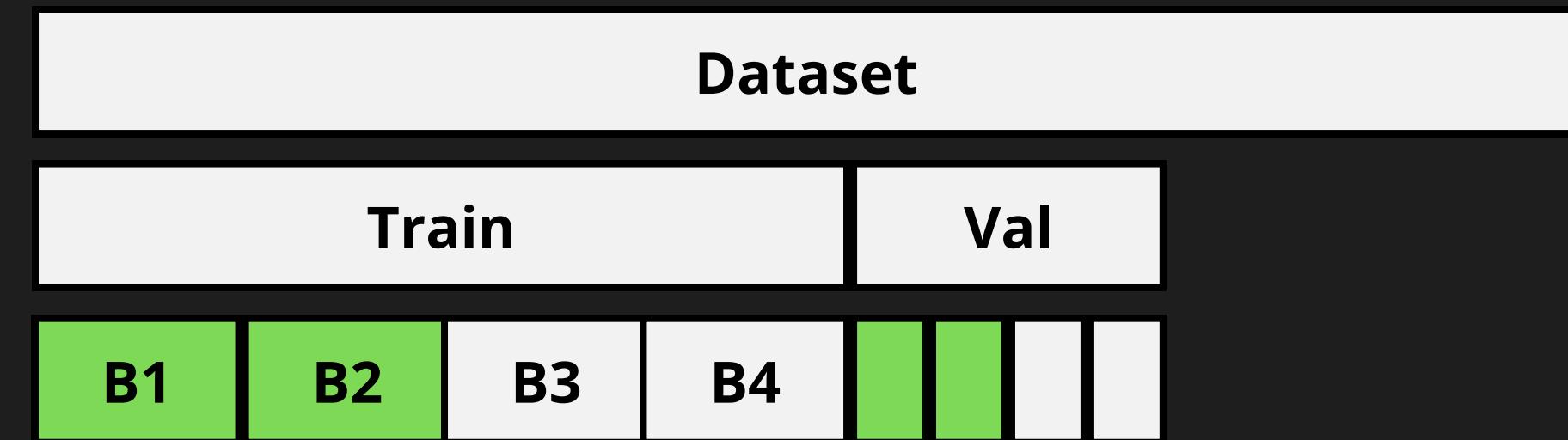
- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

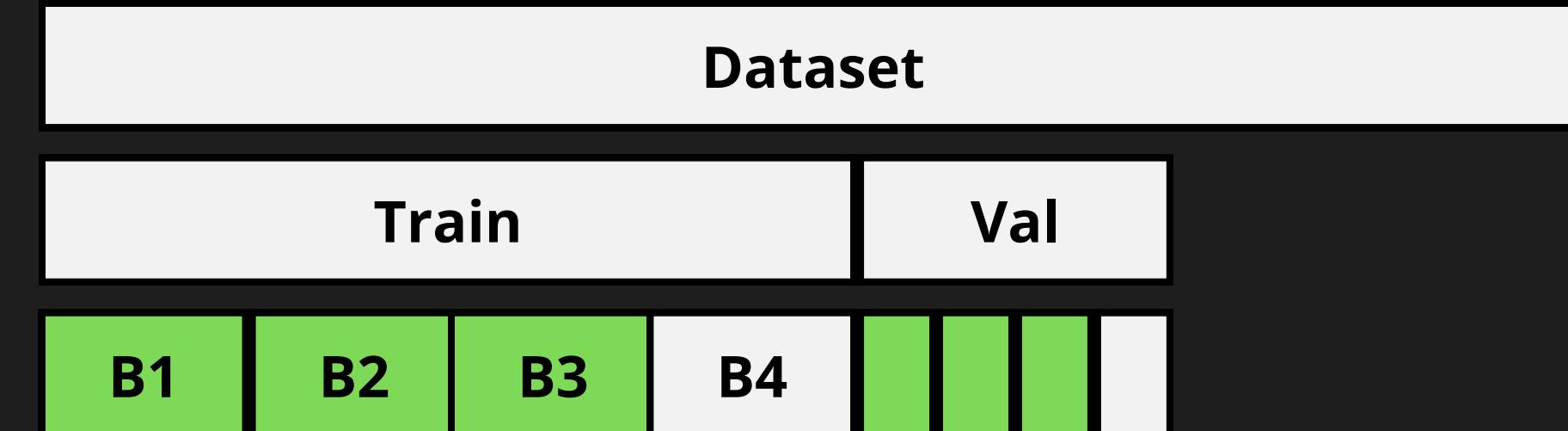
- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



Batch =4

- Batch size
- Epoch

TP CNN

V- a) Hyperparamètres du modèle et de l'entraînement

Hyperparamètres intéressant du modèle:

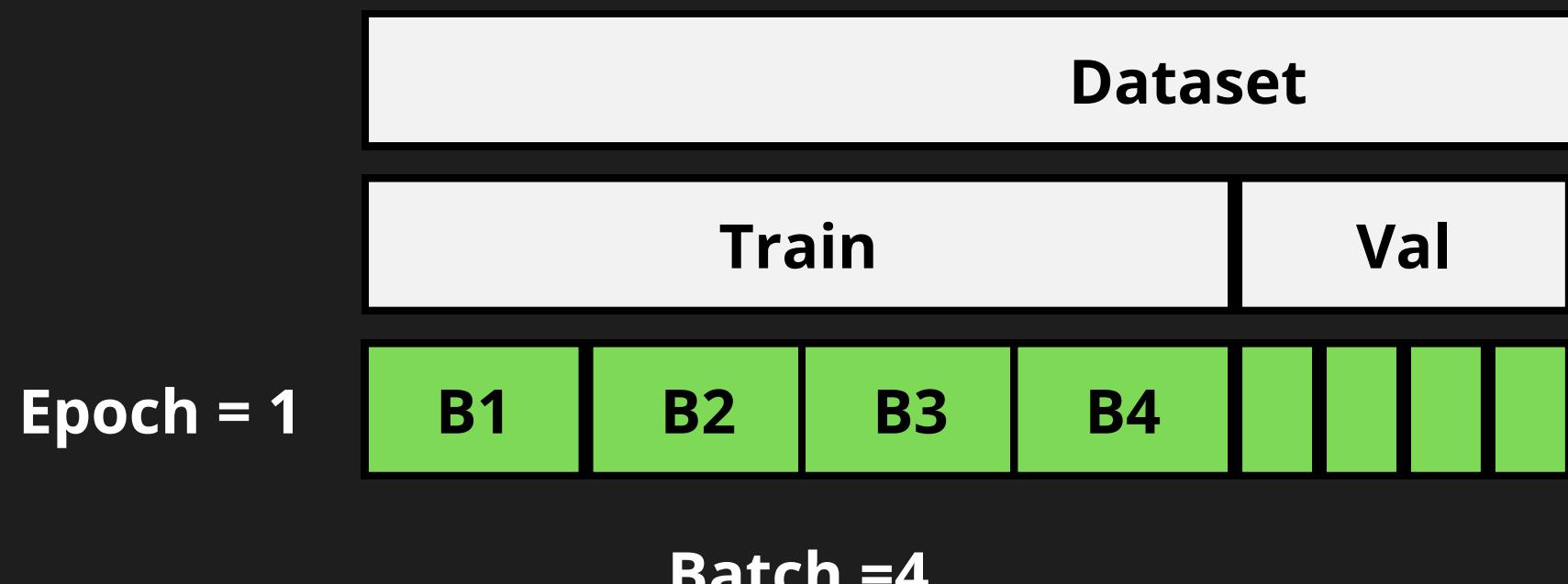
- Nombre de filtres

Hyperparamètres intéressant pour l'entraînement:

- Algorithme d'optimisation
- Fonction de perte
- Métrique

- EarlyStopping
- Save best model

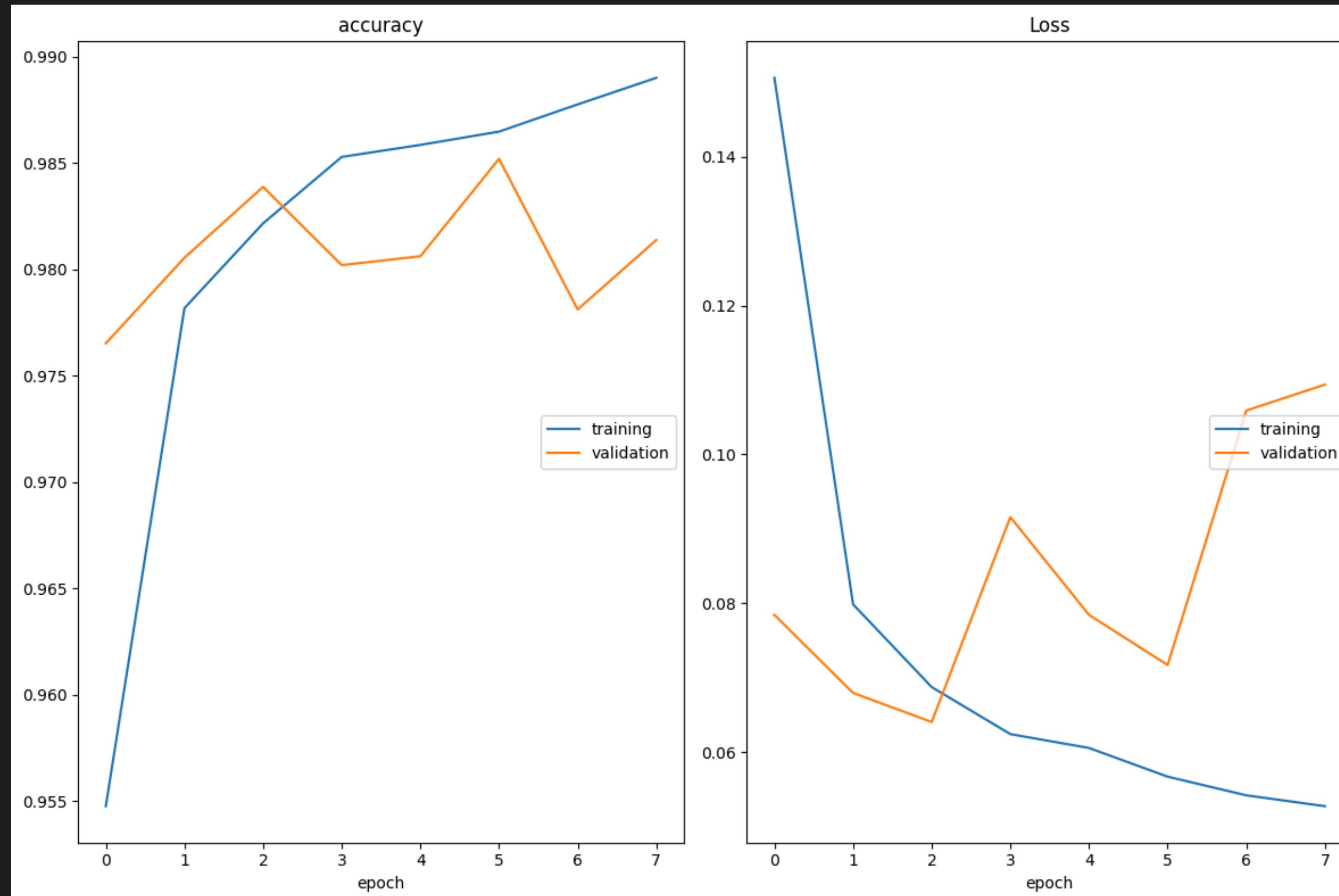
```
model.fit(X_train, Y_train,  
          batch_size=2, epochs=1000,  
          validation_split=0.3,  
          verbose=1,  
          callbacks=[plotlossesdeeper,early_stop,Model_check])
```



- Batch size
- Epoch

TP CNN

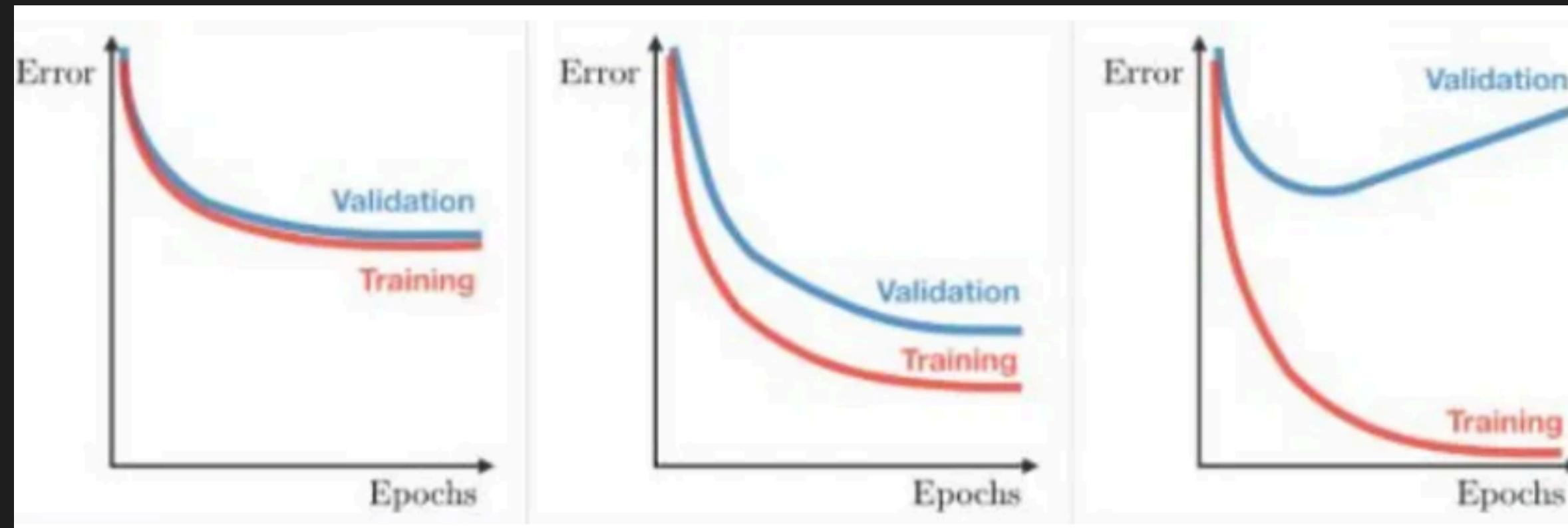
V- b) Entrainement du modèle et évaluation sur les données de validation



TP CNN

V- b) Entrainement du modèle et évaluation sur les données de validation

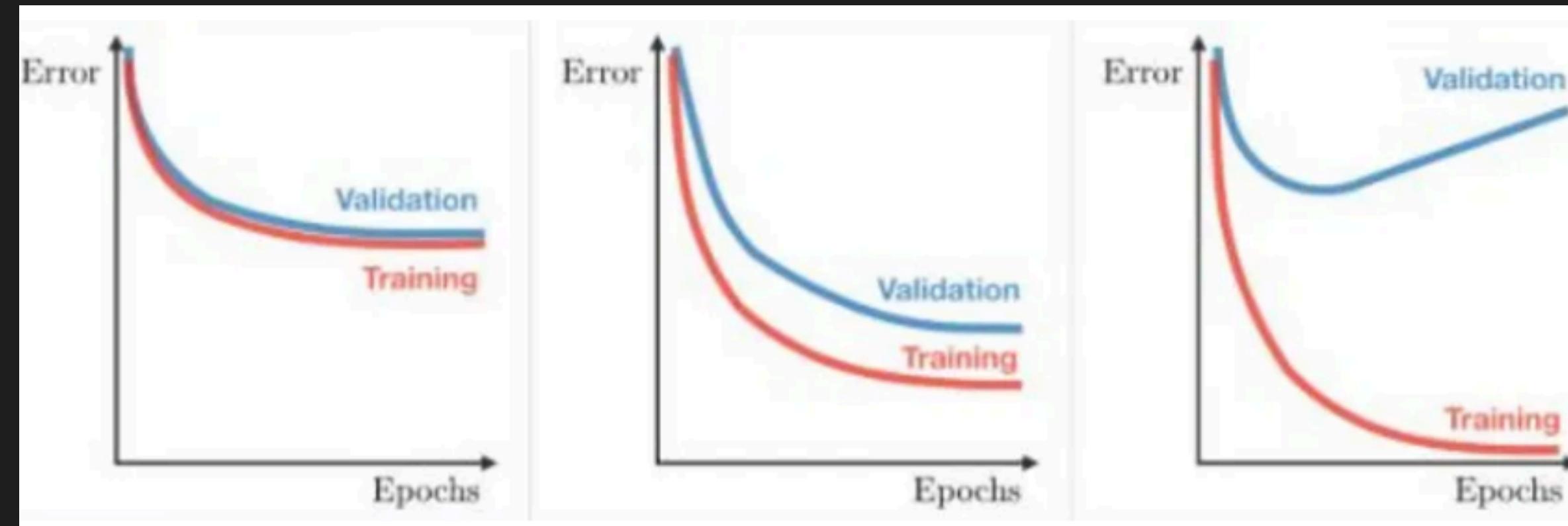
Analyse des courbes d'entraînement



TP CNN

V- b) Entrainement du modèle et évaluation sur les données de validation

Analyse des courbes d'entraînement



Underfitting

Good fitting

Overfitting

Si overfitting ou underfitting:

- 1- Appliquer un fine tuning
- 2- Réinitialiser le modèle
- 3- Lancer un nouveau entraînement

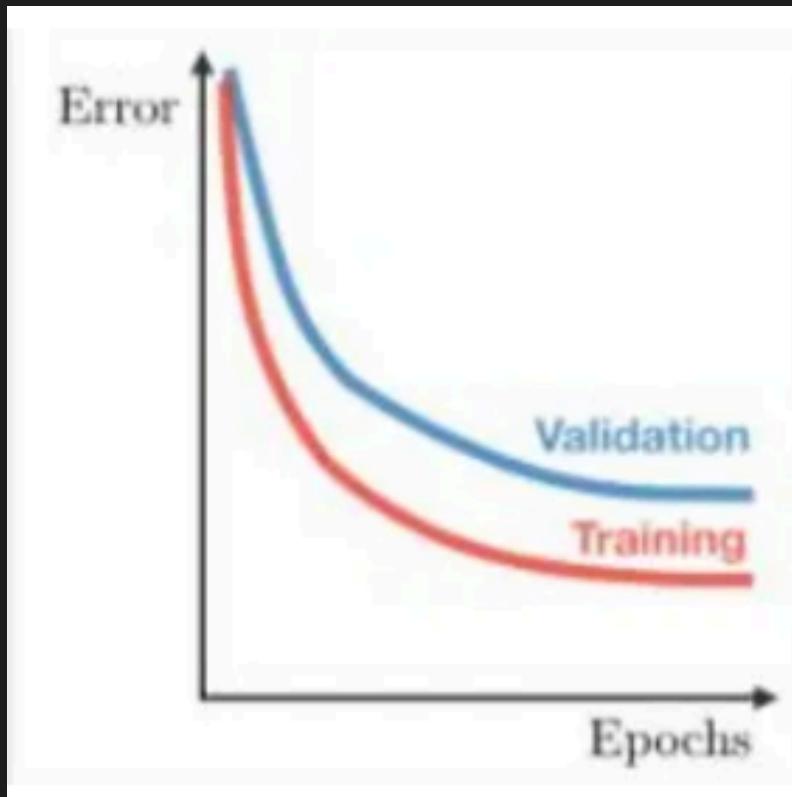
Fine tuning (ou réglage sans fin)

Ajustement des hyperparamètres
du modèle ou de l'entraînement

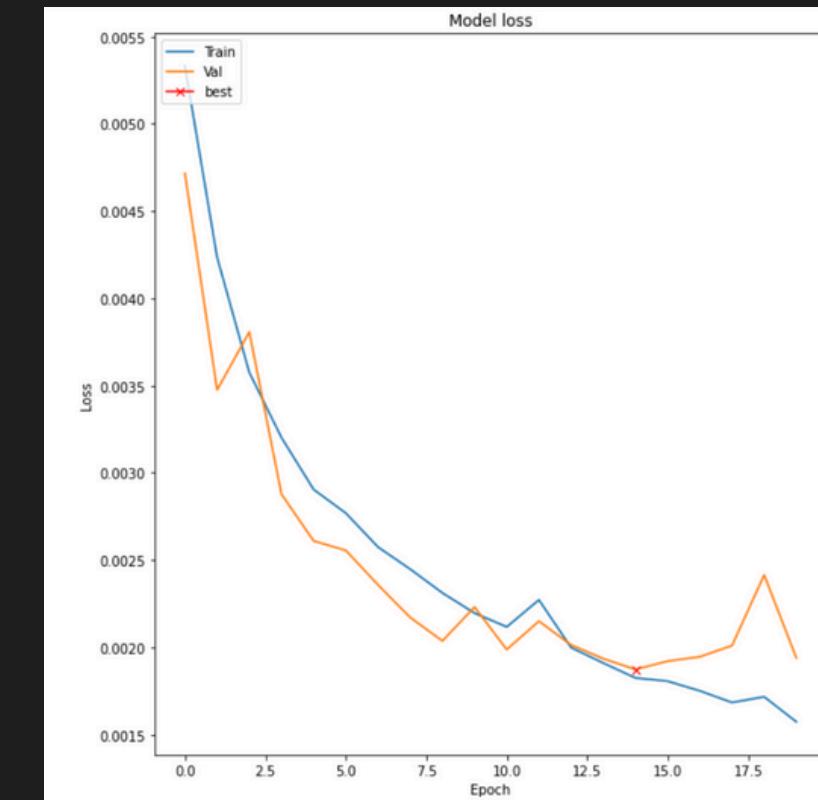
TP CNN

V- b) Entrainement du modèle et évaluation sur les données de validation

Analyse des courbes d'entraînement



Good fitting



Exemple de Good fitting

Si overfitting ou underfitting:

- 1- Appliquer un fine tuning
- 2- Réinitialiser le modèle
- 3- Lancer un nouveau entraînement

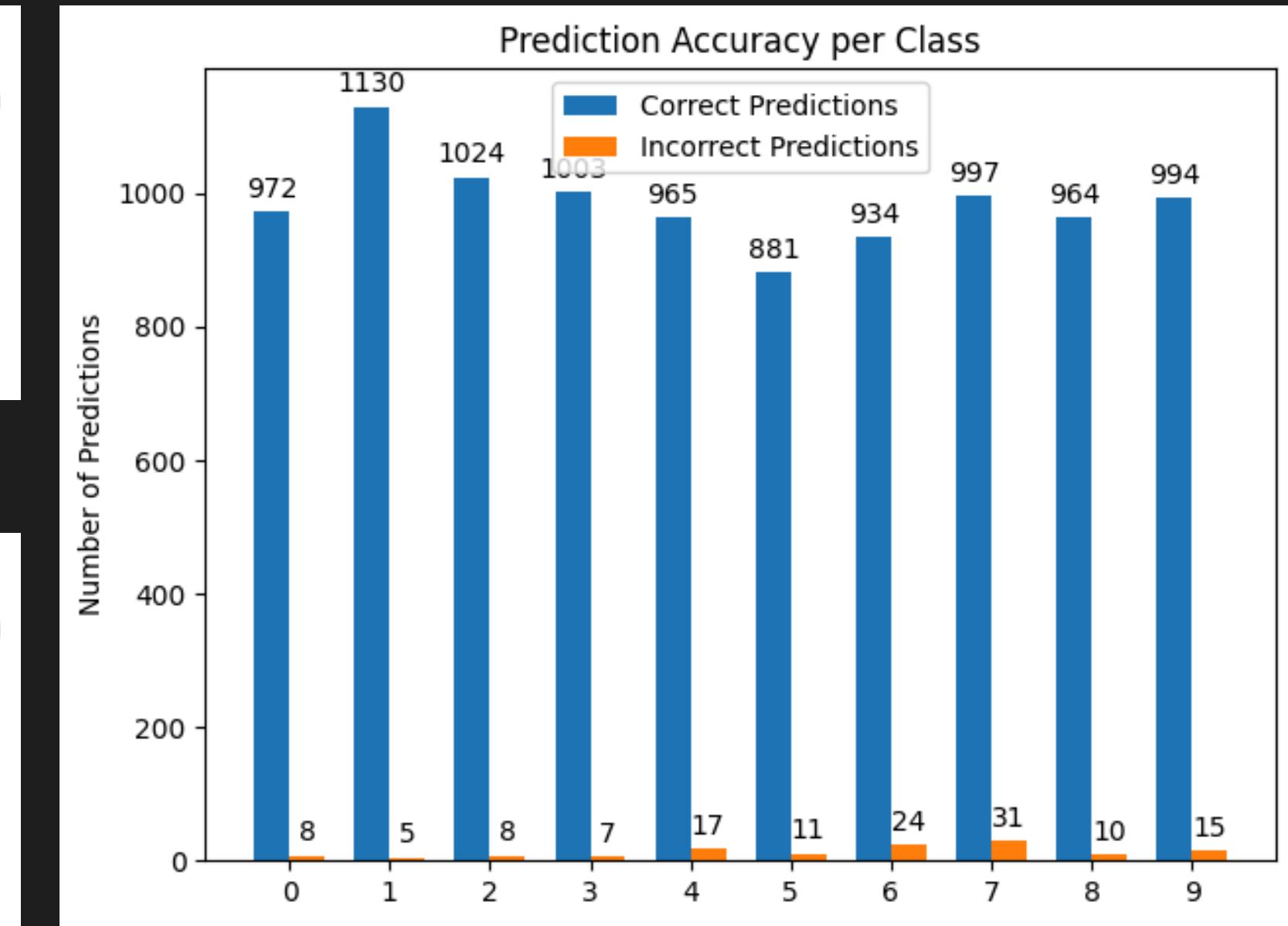
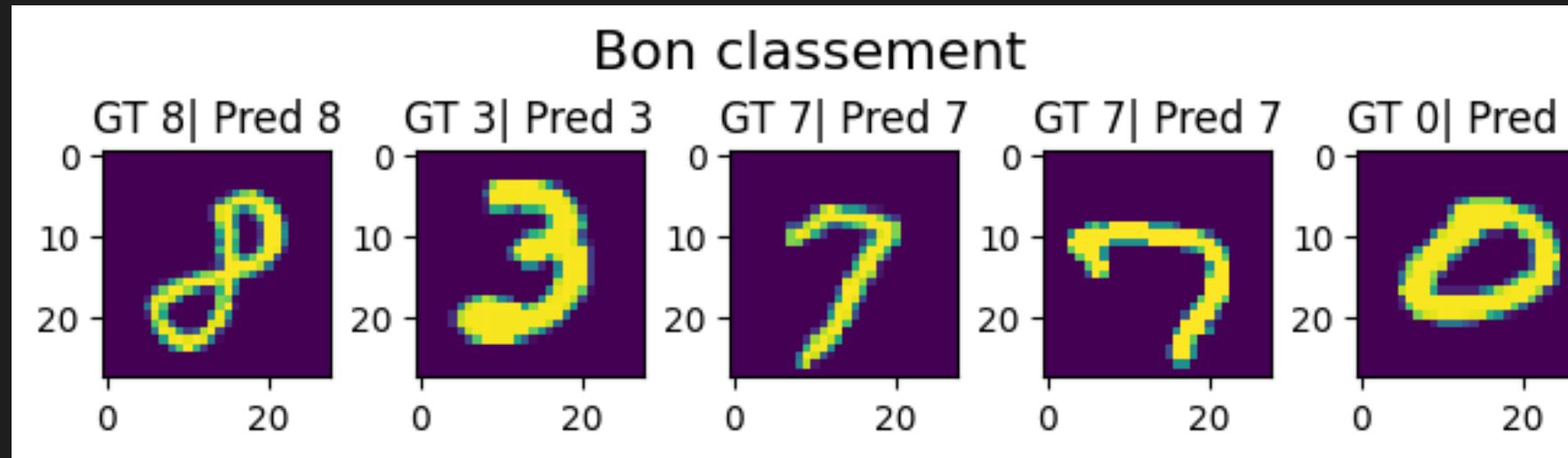
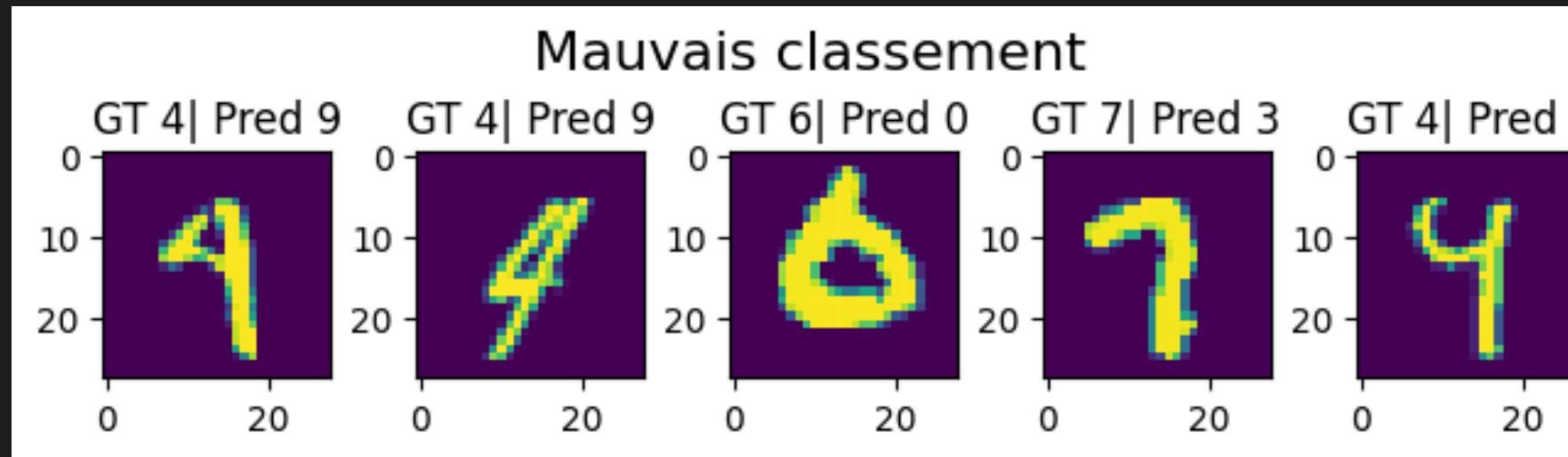
Fine tuning (ou réglage sans fin)

Ajustement des hyperparamètres
du modèle ou de l'entraînement

TP CNN

VI - Evaluation sur les données test

```
313/313 ━━━━━━━━ 5s 13ms/step - accuracy: 0.9817 - loss: 0.0780
{'accuracy': 0.9847999811172485, 'loss': 0.0603303462266922}
```

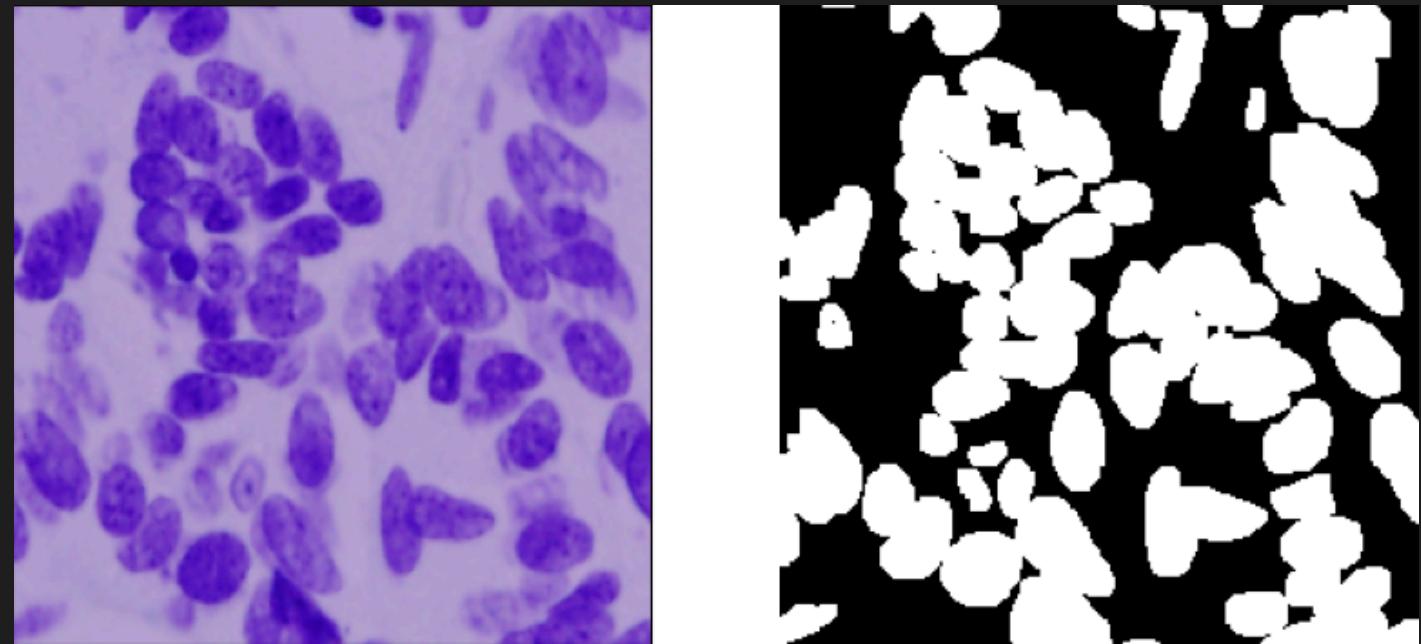


Prochaines séances

Séance 2 -

i) Prise en main d'un code de segmentation d'images UNet

ii) Techniques classiques pour améliorer un modèle : focus sur la Data Augmentation et le transfert learning



Séance 3 - Prise en main de Napari

Séance 4 - Évaluation