The Describer

Describe your image

Ecosystème d'aide aux personnes malvoyantes

Un projet innovant pour un monde compris de tous !

01.

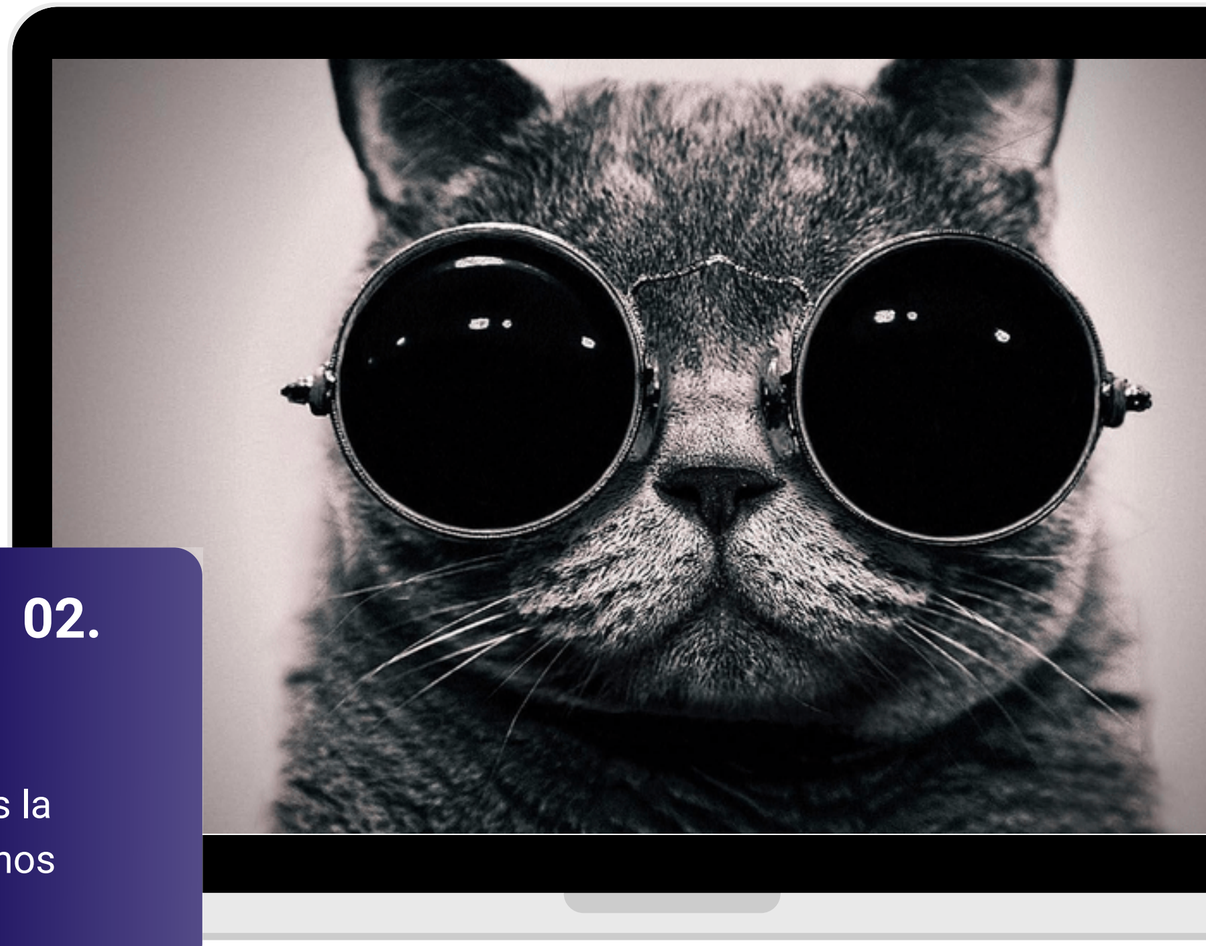
Projet

Création d'une intelligence artificielle capable de décrire une image et son contetxe

02.

Objectif

Rendre accessible à tous la description d'image sur nos outils numériques



L'équipe

LOIC FOURNIER

Administrateur Système

FABIEN COURTOIS

Développeur Frontend

LUCAS BILLARD

Développeur Backend

EWEN BOUQUET

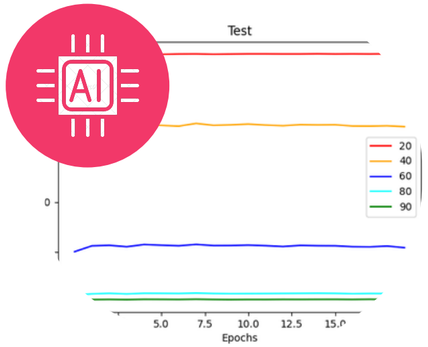
Data Scientist

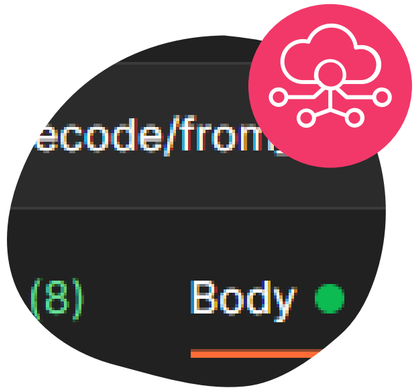
L'écosystème The Describer

Un écosystème riche, développé étape par étape
L'intelligence artificilee

Intelligence articielle

Intelligence articielle capable de décrire des images





API de prédicitons

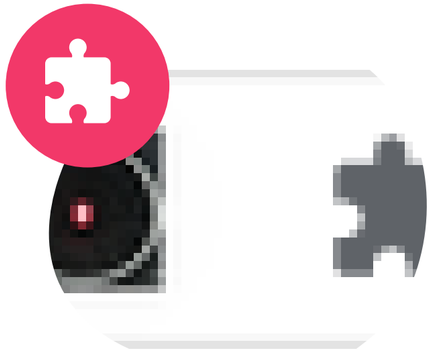
API permettant d'obtenir les prédictions de notre IA

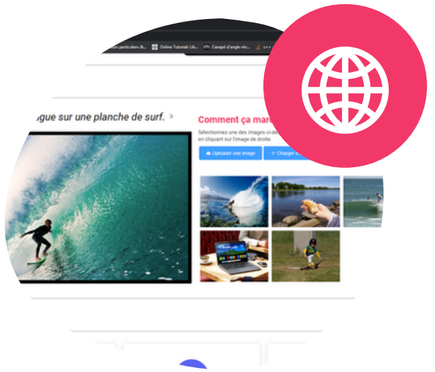
L'écosystème The Describer

Un écosystème riche, développé étape par étape
Accessible depuis le web

Extension de navigateur

Extension de navigateur pour décrire les images des sites





Site web

Site de démonstration et de description de services

L'écosystème The Describer

Un écosystème riche, développé étape par étape
Accessible depuis votre téléphone

Application Mobile

Application mobile pour nous aider dans notre quotidien

Bot Discord

Bot Discord décrivant automatiquement les images dans les conversations

Démonstration

Describe your image

La topologie du réseau

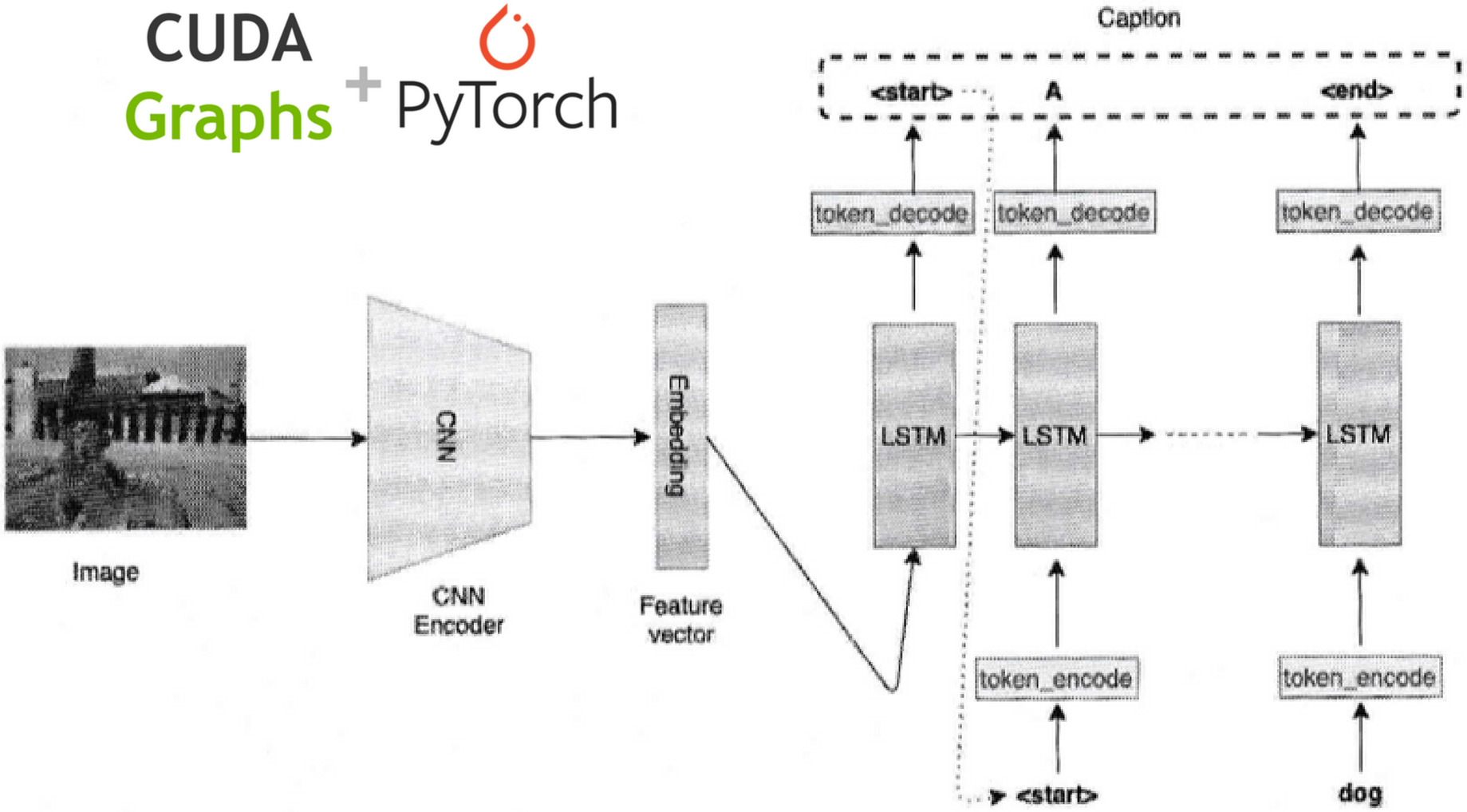
Encodeur

Transforme l'image en une liste de catégories.

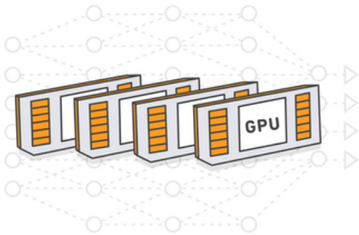
Décodeur

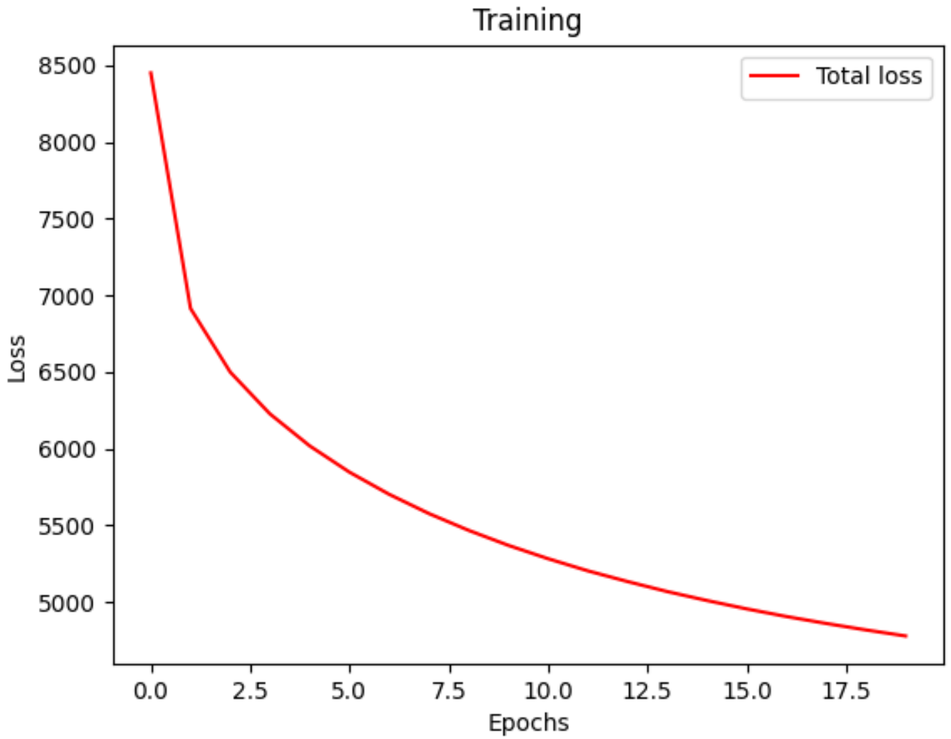
Transforme les catégories en une phrase complète.

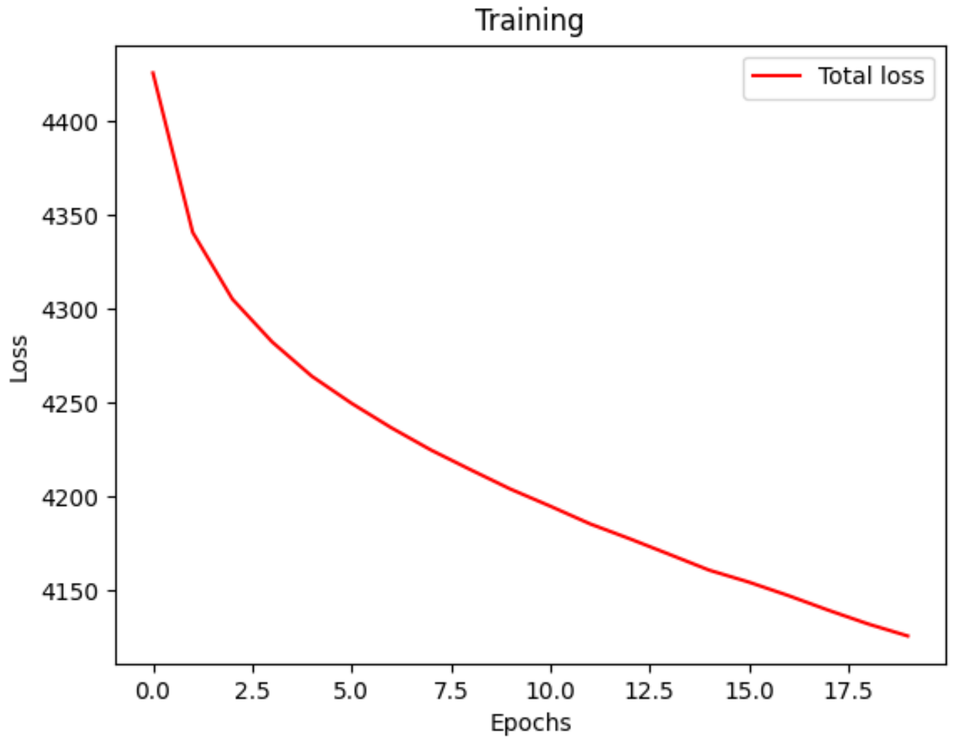
CUDA Graphs + PyTorch



L'entraînement du réseau

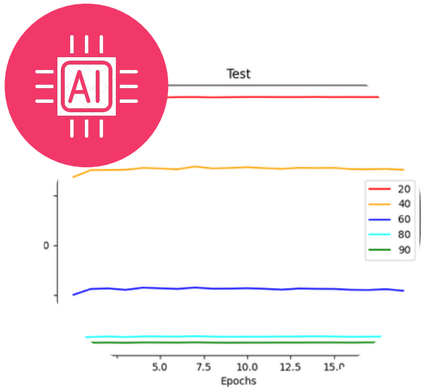






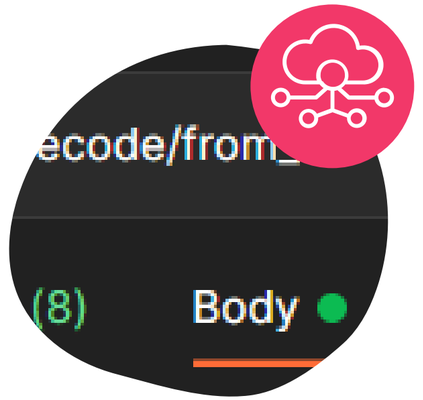
L'organisation de l'écosystème

Intelligence articielle

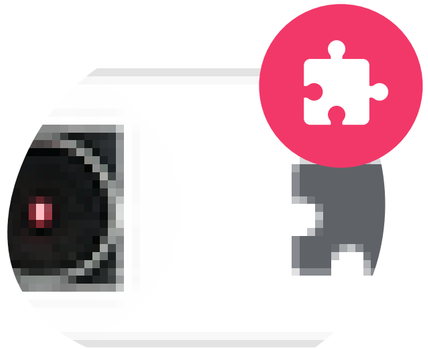


Modèle

- "encoder.cpkt"
- "decoder.cpkt"



API de prédicitons



Extension de navigateur

Site web

Application Mobile

Bot Discord

Merci pour votre attention ! 😊

Describe your image

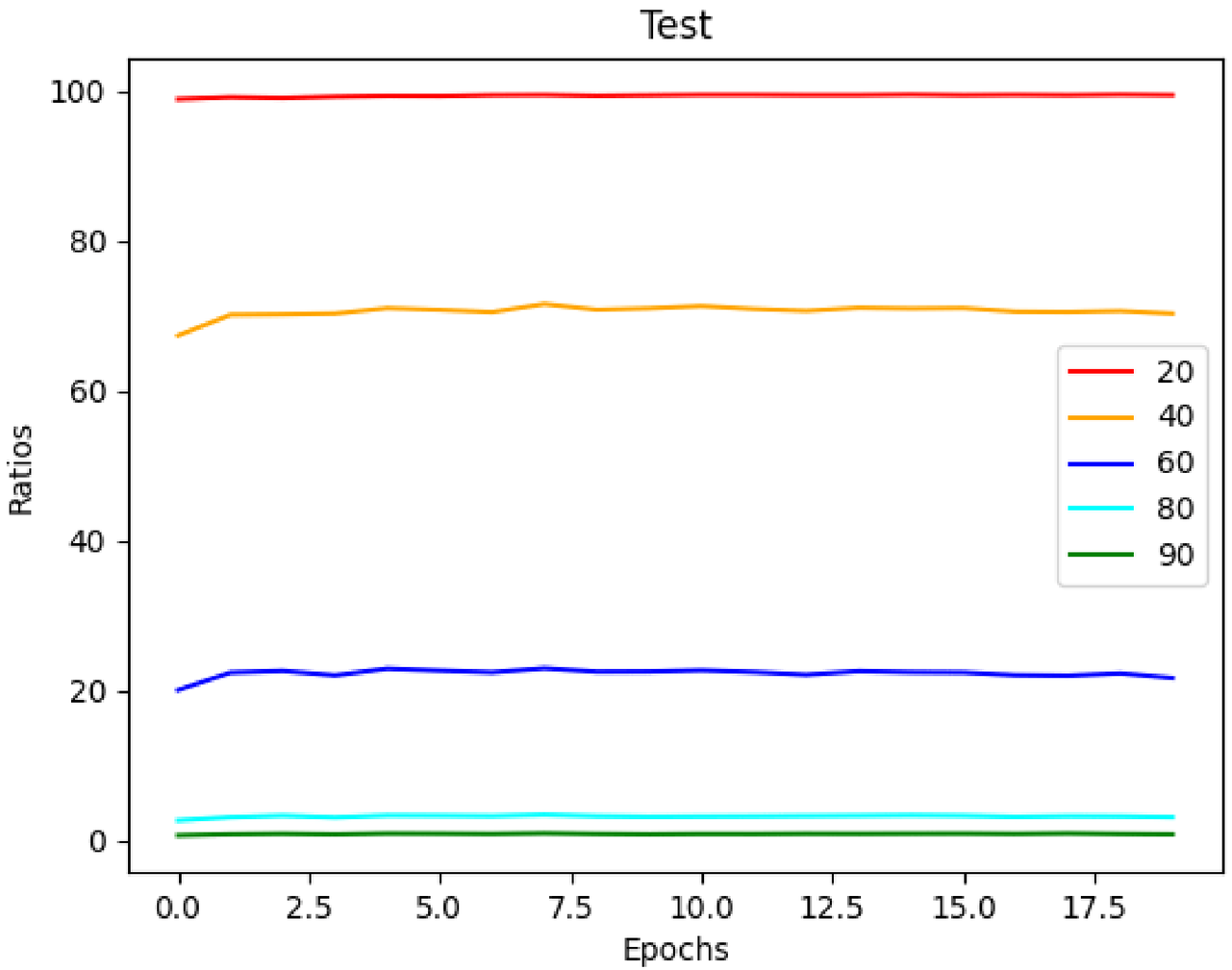
https://the-describer.netlify.app/

Annexes

Describe your image

https://the-describer.netlify.app/

Les scores du réseau



Le réseau "encodeur"

```python
class CNNModel(nn.Module):

    def __init__(self, embedding_size):
        super(CNNModel, self).__init__()

        # We use a pre-trained CNN model available under the PyTorch models repository: the ResNet 152 architecture
        # We remove the last layer of this pre-trained ResNet model
        resnet = models.resnet152(pretrained=True)
        module_list = list(resnet.children())[:-1]
        self.__resnet_module = nn.Sequential(*module_list)

        # Replace it with a fully connected layer
        self.__linear_layer = nn.Linear(resnet.fc.in_features, embedding_size)

        # Followed by a batch normalization layer
        self.__batch_norm = nn.BatchNorm1d(embedding_size, momentum=0.01)

    def forward(self, input_images):
        """Extract feature vectors from input images."""

        # We don't train the ResNet model because it is pretrained
        #   The output of the ResNet model is K x 1000-dimensional, assuming K number of neurons in the penultimate la

        with torch.no_grad():
            resnet_features = self.__resnet_module(input_images)

        resnet_features = resnet_features.reshape(resnet_features.size(0), -1)
        linear_features = self.__linear_layer(resnet_features)
        final_features = self.__batch_norm(linear_features)

        return final_features
```

Le réseau "décodeur"

```python
class LSTMModel(nn.Module):

    def __init__(self, embedding_size, hidden_layer_size, vocabulary_size, num_layers, max_seq_len=20):
        super(LSTMModel, self).__init__()

        # Embedding layer
        self.__embedding_layer = nn.Embedding(vocabulary_size, embedding_size)

        # LSTM layer
        self.__lstm_layer = nn.LSTM(embedding_size, hidden_layer_size, num_layers, batch_first=True)
        # self.__lstm_layer = nn.GRU(embedding_size, hidden_layer_size, num_layers=2, dropout=0.8, bidirectional=True)

        # Fully connected linear layer
        self.__linear_layer = nn.Linear(hidden_layer_size, vocabulary_size)

        # Max length of the predited caption
        self.__max_seq_len = max_seq_len

    def forward(self, input_features, captions, lens):
        """Decode image feature vectors and generates captions."""

        # We apply the embedding layer
        embeddings = self.__embedding_layer(captions)
        embeddings = torch.cat((input_features.unsqueeze(1), embeddings), 1)
        lstm_input = pack_padded_sequence(embeddings, lens, batch_first=True)

        # We apply the LSTM layer
        hidden_variables, _ = self.__lstm_layer(lstm_input)

        # We apply the fully connected layer
        model_outputs = self.__linear_layer(hidden_variables[0])

        return model_outputs
```

Le réseau principal

```python
class FullModel(nn.Module):

    def __init__(self, device, image_shape, vocabulary):
        """Combine the CNN and LSTM models."""
        super(FullModel, self).__init__()

        # Build the models
        self.__encoder_model = CNNModel(image_shape[0]).to(device)
        self.__decoder_model = LSTMModel(image_shape[0], image_shape[0] + image_shape[1], len(vocabulary), 1).to(device)

        # Loss and optimizer
        self.__loss_criterion = nn.CrossEntropyLoss()

    def forward(self, images, captions, lens):

        feats   = self.__encoder_model(images)
        outputs = self.__decoder_model(feats, captions, lens)

        return outputs

    def sample(self, image):

        feats = self.__encoder_model(image)
        return self.__decoder_model.sample(feats)
```