FACULTE DES SCIENCES BEN M'SICK
UNIVERSITÉ HASSAN II DE CASABLANCA

Projet de fin de module : Deep Learning

Classification d'Images de Bâtiments Historiques vs Modernes

Réalisé par : Khadija KHARIF

Zaineb RHOUNI

Youssef SARRAF

Encadré par : Oussama KAICH

Année Universitaire : 2024-2025

Remerciements

Nous tenons à exprimer notre sincère gratitude à M. El Habib Ben Lahmar, professeur du module de Deep Learning, pour la qualité de son enseignement, sa disponibilité et ses précieux conseils tout au long de ce projet. Son expertise et sa rigueur ont grandement contribué à notre compréhension du domaine.
Nous remercions chaleureusement notre encadrant, M. Oussama Kaich, pour son accompagnement attentif, ses orientations pertinentes et sa bienveillance. Son implication a été essentielle au bon déroulement de notre travail.
Un grand merci à nos collègues et partenaires pour les échanges constructifs et les contributions précieuses qui ont enrichi notre analyse.
Nous adressons également nos remerciements à nos familles et amis pour leur soutien moral et leur compréhension tout au long de cette période de travail intense.
Enfin, nous remercions toutes les personnes ayant, de près ou de loin, contribué à la réussite de ce projet.

Résumé

Ce rapport présente un projet de deep learning visant à classer des images de bâtiments en deux catégories : historiques et modernes. Nous explorons toutes les étapes du processus de machine learning, de la collecte et du prétraitement des données à l'entraînement d'un réseau de neurones convolutifs (CNN) et à l'évaluation de ses performances.
Notre approche met l'accent sur l'analyse des caractéristiques architecturales distinctives qui permettent de différencier les styles de construction à travers les époques.

Table des matières

Table des figures

1 Introduction

L'architecture est une manifestation visuelle de l'histoire humaine, reflétant l'évolution des technologies de construction, des valeurs esthétiques et des contextes socio-économiques. La distinction entre les bâtiments historiques et modernes représente non seulement un défi intéressant pour la vision par ordinateur, mais aussi une application pratique potentielle dans des domaines tels que l'urbanisme, la préservation du patrimoine, et l'analyse immobilière.

Ce projet vise à développer un modèle de classification d'images capable de distinguer les bâtiments historiques des constructions modernes. En utilisant des techniques de deep learning, particulièrement les réseaux de neurones convolutifs (CNN), nous exploitons la puissance de l'apprentissage automatique pour identifier les caractéristiques visuelles qui définissent chaque catégorie.

1.1 Objectifs du projet

— Constituer un ensemble de données équilibré d'images de bâtiments historiques et modernes

— Prétraiter les images pour optimiser l'apprentissage du réseau neuronal

— Concevoir et entraîner un CNN adapté à cette tâche de classification binaire

— Évaluer les performances du modèle selon diverses métriques

— Analyser les caractéristiques visuelles sur lesquelles le modèle base ses décisions

1.2 Définition du problème

La classification d'images architecturales présente plusieurs défis spécifiques :

— La grande diversité stylistique au sein de chaque catégorie

— Les variations d'échelle, d'angle et de conditions d'éclairage

— La subjectivité potentielle dans la définition de "historique" vs "moderne"

— Les bâtiments contemporains incorporant des éléments de styles historiques

Notre définition de travail considère comme "historiques" les bâtiments reflétant les styles architecturaux traditionnels, tandis que les bâtiments "modernes" englobent les constructions contemporaines caractérisées par des matériaux actuels et des lignes épurées.

2 Collecte de données

Ce jeu de données contient plus de 12 300 images d'architecture organisées en 9 catégories différentes, basées sur des périodes historiques. Il s'agit d'une extension et d'une réorganisation du jeu de données Architectural styles créé par dumitrux. Des images supplémentaires ont été collectées sur Google Images. Un total de 21 nouveaux styles architecturaux ont été ajoutés à partir de cette collecte. C'est un mélange entre des images collectées depuis Google Images et celles issues de l'article scientifique "Architectural Style Classification using Multinomial Latent Logistic Regression" (ECCV2014), réalisé par Zhe Xu.

Période architecturale	**Nombre d'images**
21st_Century (21e siècle)	1228
Neoclassicism (Néoclassicisme)	1225
Eclecticism (Éclectisme)	1314
Revivalism (Révivalisme)	1923
Baroque (Baroque)	1086
Modernism (Modernisme)	1858
Renaissance_and_Colonialism (Renaissance et colonialisme)	1288
Classical (Classique)	1020
Early_Christian_Medieval (Chrétien primitif et médiéval)	1335

TABLE 1 – Distribution des images dans le jeu de données Architectural Styles Periods Dataset

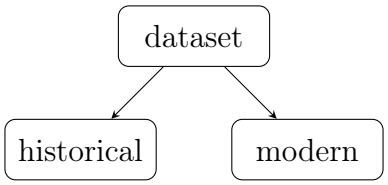


FIGURE 1 – Arborescence du jeu de données

Explication :
L'arborescence ci-dessus représente la structure hiérarchique des données utilisées pour ce projet. Le répertoire principal `dataset` contient deux sous-dossiers :

- **historical** : regroupe les images de bâtiments historiques appartenant aux périodes architecturales listées dans le tableau précédent (Néoclassicisme, Baroque, etc.)

- **modern** : contient les images de constructions contemporaines caractéristiques du 21e siècle

Cette organisation permet de séparer clairement les deux classes cibles pour la tâche de classification binaire.

3 Augmentation de données

Avant de procéder au prétraitement, nous avons effectué une augmentation des données, car la classe `modern` contenait seulement 4 404 images, tandis que la classe `historical` en comptait 7 887.

Nous avons donc utilisé le code suivant pour augmenter les données de la classe `modern`, afin d'équilibrer le dataset et d'obtenir environ 7 887 images pour chaque classe.

Augmentation de la classe Modern

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, array_to_img
import os
import numpy as np
import random

# Répertoires
dataset_dir = "dataset"
modern_dir = os.path.join(dataset_dir, "modern")
historical_dir = os.path.join(dataset_dir, "historical")

# Nombre d'images par classe
modern_images = [f for f in os.listdir(modern_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png')) and not f.startswith("aug_")]
nb_modern = len(modern_images)
nb_historical = len([f for f in os.listdir(historical_dir) if f.lower().endswith(('.jpg', '.jpeg', '.png'))])

# Nombre d'images à generer
diff = nb_historical - nb_modern

# Generateur de donnees pour l'augmentation
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Generation et sauvegarde des images augmentees
augmented_images = []
for img_name in modern_images:
    img_path = os.path.join(modern_dir, img_name)
    img = load_img(img_path)
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    aug_img = next(datagen.flow(x, batch_size=1))[0].astype('uint8')
    new_name = f"aug_{img_name}"
    save_path = os.path.join(modern_dir, new_name)
    array_to_img(aug_img).save(save_path)
    augmented_images.append(save_path)

# Reduction si il ya beaucoup d'images generees
if len(augmented_images) > diff:
    to_delete = random.sample(augmented_images, len(augmented_images) - diff)
    for path in to_delete:
        os.remove(path)
```

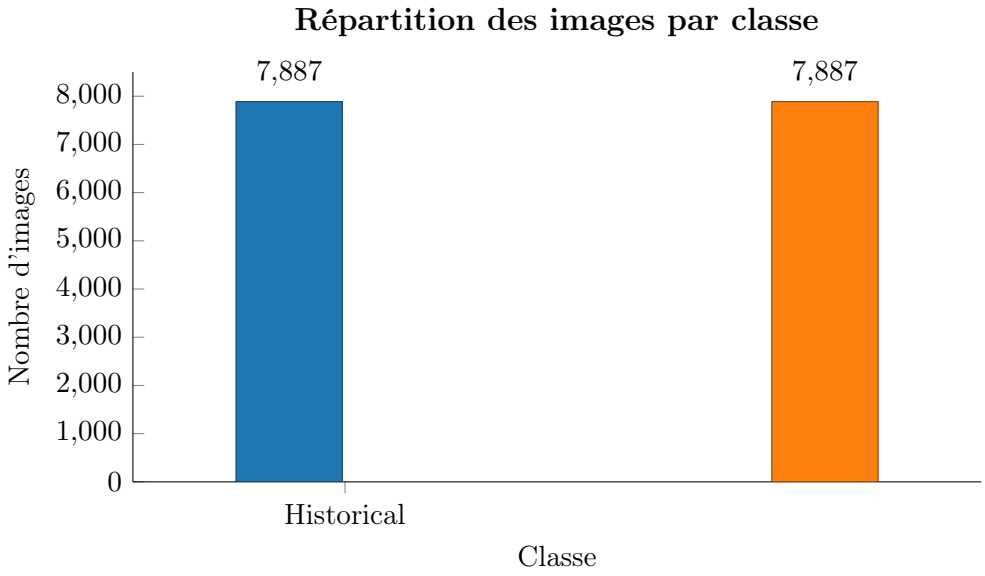


FIGURE 2 – Résultat après l'augmentation de données

4 Prétraitement

Le prétraitement des images est une étape cruciale pour préparer les données d'entraînement du modèle. Les opérations principales incluent :

- Redimensionnement des images à 224x224 pixels
- Augmentation de données avec rotation (20°), décalage (10%), et retournement horizontal
- Égalisation d'histogramme sur le canal de luminance (YCrCb) pour améliorer le contraste
- Normalisation des valeurs de pixels entre 0 et 1
- Séparation des données en ensembles d'entraînement (80%), validation (20%) et test

Prétraitement des images avec amélioration et augmentation de données

```python
import os
import numpy as np
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tqdm import tqdm

# -----------------------
# CONFIGURATION
# -----------------------
DATASET_DIR = "dataset_split"
IMG_SIZE = (128, 128) # Note: Text says 224x224, code says 128x128
BATCH_SIZE = 32
SAVE_DIR = "preprocessed_data"
AUGMENTED = False # Note: This is set to False here, but augmentation is discussed before.

# Fonction d'amélioration : égalisation
def enhance_image(img):
    img = (img * 255).astype(np.uint8)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YCrCb)
    img[:, :, 0] = cv2.equalizeHist(img[:, :, 0])
    img = cv2.cvtColor(img, cv2.COLOR_YCrCb2RGB)
    return img.astype(np.float32) / 255.0

# Extraction des données à partir d'un générateur
def extract_all_data(generator, mode_name, test_mode=False):
    images, labels = [], []
    total = 0
    print(f" Extraction des images {mode_name}...")
    for i in tqdm(range(len(generator))):
        try:
            x_batch, y_batch = generator[i]
            for x, y in zip(x_batch, y_batch):
                try:
                    enhanced = enhance_image(x) # Enhancement is applied here
                    images.append(enhanced)
                    labels.append(y)
                    total += 1
                    if test_mode and total >= 10:
                        return np.array(images), np.array(labels)
                except Exception as e:
                    print(f" Image ignorée : {e}")
        except Exception as e:
            print(f" Batch ignoré : {e}")
    return np.array(images), np.array(labels)
```

PRÉTRAITEMENT PRINCIPAL

```python
# -----------------------
# PRÉTRAITEMENT PRINCIPAL
# -----------------------
def preprocess_images(test_mode=False):
    print(" Prétraitement des images...")

    datagen = ImageDataGenerator(
        rescale=1./255,
        validation_split=0.2, # Applied on train_dir for val_gen
        rotation_range=20 if AUGMENTED else 0,
        width_shift_range=0.1 if AUGMENTED else 0,
        height_shift_range=0.1 if AUGMENTED else 0,
        horizontal_flip=AUGMENTED # AUGMENTED is False by default in this script )

    # Générateurs train/val
    train_gen = datagen.flow_from_directory(
        os.path.join(DATASET_DIR, "train"),
        target_size=IMG_SIZE, # Uses (128,128)
        batch_size=BATCH_SIZE,
        class_mode='binary',
        subset='training',
        shuffle=True )

    val_gen = datagen.flow_from_directory(
        os.path.join(DATASET_DIR, "train"), # Validation from train directory
        target_size=IMG_SIZE, # Uses (128,128)
        batch_size=BATCH_SIZE,
        class_mode='binary',
        subset='validation',
        shuffle=True )

    # Générateur test
    test_gen = ImageDataGenerator(rescale=1./255).flow_from_directory(
        os.path.join(DATASET_DIR, "test"),
        target_size=IMG_SIZE, # Uses (128,128)
        batch_size=BATCH_SIZE,
        class_mode='binary',
        shuffle=False )

    x_train, y_train = extract_all_data(train_gen, "entraînement", test_mode)
    x_val, y_val = extract_all_data(val_gen, "validation", test_mode)
    x_test, y_test = extract_all_data(test_gen, "test", test_mode) # Typo corrected
    ↪ here

    os.makedirs(SAVE_DIR, exist_ok=True)
    suffix = "_test" if test_mode else ""
    np.save(os.path.join(SAVE_DIR, f"x_train{suffix}.npy"), x_train)
    np.save(os.path.join(SAVE_DIR, f"y_train{suffix}.npy"), y_train)
    np.save(os.path.join(SAVE_DIR, f"x_val{suffix}.npy"), x_val)
    np.save(os.path.join(SAVE_DIR, f"y_val{suffix}.npy"), y_val)
    np.save(os.path.join(SAVE_DIR, f"x_test{suffix}.npy"), x_test)
    np.save(os.path.join(SAVE_DIR, f"y_test{suffix}.npy"), y_test)

    print(" Données sauvegardées !")
    print(f" train : {len(x_train)}, val : {len(x_val)}, test : {len(x_test)}")

if __name__ == "__main__":
    preprocess_images(test_mode=False)
```

5 Modélisation CNN

Dans cette section, nous présentons le code Python utilisé pour construire et entraîner le modèle CNN pour la classification des images de bâtiments historiques et modernes. Le modèle repose sur une approche de transfert d'apprentissage avec MobileNetV2, suivi d'un fine-tuning pour optimiser les performances.

Code du modèle CNN

```python
import os
import shutil
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split

# === Étape 1 : Organisation du dataset ===
original_data_dir = 'dataset'  # contient modern/ et historical/
base_dir = 'data_split' # Note: This is DATASET_DIR in preprocessing script
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'val')
test_dir = os.path.join(base_dir, 'test')

# Nettoyage
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)

# Création des dossiers
for split_dir in [train_dir, val_dir, test_dir]:
    os.makedirs(os.path.join(split_dir, 'modern'))
    os.makedirs(os.path.join(split_dir, 'historical'))

# Fonction de split
def split_and_copy(class_name):
    src = os.path.join(original_data_dir, class_name)
    images = os.listdir(src)
    # Splitting: 80% train, 10% val, 10% test from original_data_dir
    train_imgs, temp_imgs = train_test_split(images, test_size=0.2,
    ↪  random_state=42)
    val_imgs, test_imgs = train_test_split(temp_imgs, test_size=0.5,
    ↪  random_state=42) # 0.5 of 0.2 is 0.1

    for img in train_imgs:
        shutil.copy(os.path.join(src, img), os.path.join(train_dir, class_name))
    for img in val_imgs:
        shutil.copy(os.path.join(src, img), os.path.join(val_dir, class_name))
    for img in test_imgs:
        shutil.copy(os.path.join(src, img), os.path.join(test_dir, class_name))

# Appliquer le split
split_and_copy('modern')
split_and_copy('historical')
```

Suite du code : Modélisation CNN

```python
# === Étape 2 : Prétraitement ===
img_size = (224, 224) # Note: This is 224x224, different from IMG_SIZE in preprocessing script
batch_size = 32

datagen = ImageDataGenerator(rescale=1./255) # No augmentation here for training generators

train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=img_size, # Uses (224,224)
    batch_size=batch_size,
    class_mode='binary'
)

val_generator = datagen.flow_from_directory(
    val_dir,
    target_size=img_size, # Uses (224,224)
    batch_size=batch_size,
    class_mode='binary'
)

test_generator = datagen.flow_from_directory(
    test_dir,
    target_size=img_size, # Uses (224,224)
    batch_size=batch_size,
    class_mode='binary',
    shuffle=False
)

# === Étape 3 : Transfert learning avec MobileNetV2 ===
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

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

# === Entraînement initial (tête du modèle uniquement) ===
early_stop = EarlyStopping(patience=3, monitor='val_loss', restore_best_weights=True)

history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop]
)
```

Suite du code : Modélisation CNN

```python
# === Étape 4 : Fine-tuning ===
base_model.trainable = True
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
              loss='binary_crossentropy',
              metrics=['accuracy'])

history_ft = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop]
)

# === Étape 5 : Affichage des courbes ===
def plot_history(histories, titles): # titles parameter seems unused
    plt.figure(figsize=(12, 5))

    # Loss
    plt.subplot(1, 2, 1)
    for h, label in histories:
        plt.plot(h.history['loss'], label=f'{label} Train')
        plt.plot(h.history['val_loss'], '--', label=f'{label} Val')
    plt.title("Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()

    # Accuracy
    plt.subplot(1, 2, 2)
    for h, label in histories:
        plt.plot(h.history['accuracy'], label=f'{label} Train')
        plt.plot(h.history['val_accuracy'], '--', label=f'{label} Val')
    plt.title("Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()

    plt.tight_layout()
    plt.show()

plot_history([(history, 'Base'), (history_ft, 'Fine-tune')], ['Base', 'Fine-tune'])
↪  # titles passed here

# === Étape 6 : Évaluation finale ===
loss, accuracy = model.evaluate(test_generator)
print(f"\n Accuracy sur test : {accuracy*100:.2f}%")
```

5.1 Résultats de l'entraînement et du Fine-Tuning

Pour évaluer les performances du modèle CNN lors des phases de Transfer Learning (TL) et de Fine-Tuning (FT), nous avons analysé les courbes de perte (*loss*) et de précision (*accuracy*) sur les ensembles d'entraînement et de validation. Ces courbes sont illustrées dans la Figure 3.

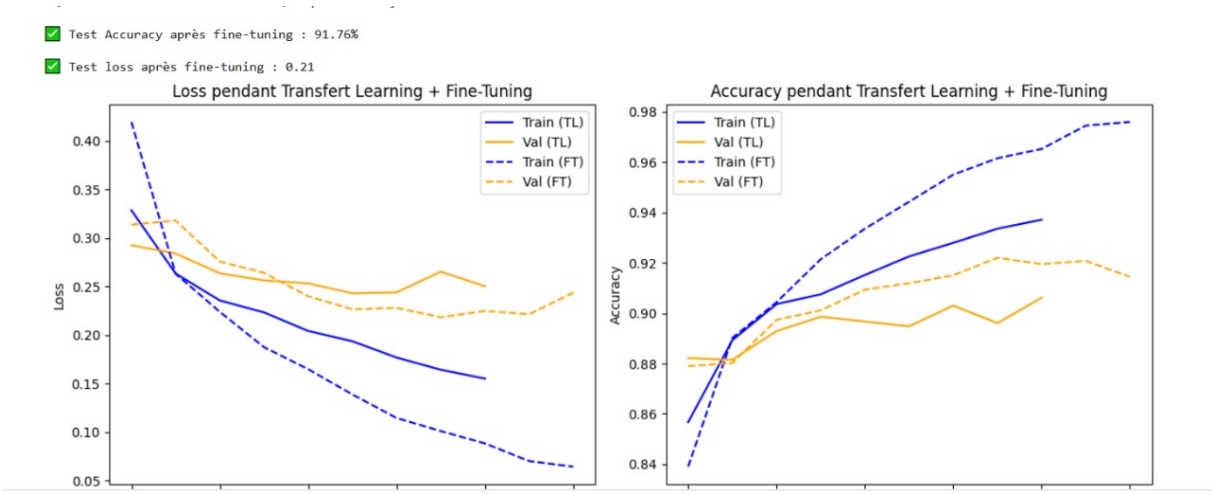


FIGURE 3 – Courbes de perte et de précision pendant le Transfer Learning et le Fine-Tuning

La Figure 3 présente deux graphiques :

— **Gauche : Perte pendant le Transfer Learning et le Fine-Tuning**

— **Droite : Précision pendant le Transfer Learning et le Fine-Tuning**

Dans le graphique de gauche, la perte d'entraînement (Train TL) commence à environ 0,40 et diminue jusqu'à 0,10, tandis que la perte de validation (Val TL) fluctue entre 0,25 et 0,30, avec une légère hausse vers la fin. Pendant le Fine-Tuning, la perte d'entraînement (Train FT) débute à 0,20 et chute sous 0,10, et la perte de validation (Val FT) se stabilise autour de 0,25.

Dans le graphique de droite, la précision d'entraînement (Train TL) passe de 0,86 à 0,97, et la précision de validation (Val TL) atteint 0,92. Pendant le Fine-Tuning, la précision d'entraînement (Train FT) grimpe de 0,90 à près de 0,98, et la précision de validation (Val FT) atteint environ 0,94.

Ces courbes démontrent que le Transfer Learning initialise efficacement le modèle avec des poids pré-entraînés, tandis que le Fine-Tuning optimise ces poids pour le jeu de données spécifique. Les métriques finales sur l'ensemble de test (précision de 91,76 % et perte de 0,21) confirment l'efficacité de cette approche. L'instabilité de la perte de validation pendant le Transfer Learning suggère un léger surajustement ou une variabilité des données, atténuée par le Fine-Tuning.

6 Déploiement

Dans cette section, nous présentons le code Python utilisé pour créer l'application Streamlit qui permet de prédire si une image de bâtiment est historique ou moderne.

Code de l'application Streamlit

```python
import streamlit as st
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

st.markdown(
    """<style>.stApp { background-color: #EEDDC8; /* marron bébé clair */}</style>
    """, unsafe_allow_html=True)

# Charger le modèle
model = load_model("mon_modele_fine_tune.h5")

# Fonction de prédiction
def predict_single_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224)) # Uses (224,224)
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_array)[0][0]
    label = "Moderne" if prediction >= 0.5 else "Historique"
    return label, prediction

# Interface Streamlit
st.title(" Prédiction : Bâtiment Moderne ou Historique")

uploaded_file = st.file_uploader(" Téléchargez une image", type=["jpg", "jpeg",
↪  "png"])

if uploaded_file is not None:
    # Afficher l'image
    img = image.load_img(uploaded_file, target_size=(224, 224))
    st.image(img, caption="Image téléchargée", width=300)

    # Prédiction
    label, prediction = predict_single_image(uploaded_file)
    # Résultat : Afficher label + probabilité associée à cette classe
    if label == "Moderne":
        proba = prediction
    else:
        proba = 1 - prediction

    # Résultat : Afficher label + probabilité
    st.markdown(f"### Prédiction : *{label}*")
    st.markdown(f"Probabilité : *{proba:.2f}*")

    # Visualisation graphique
    fig, ax = plt.subplots()
    ax.barh(["Historique", "Moderne"], [1 - prediction, prediction],
    ↪  color=["#DDA853", "#183B4E"])
    ax.set_xlim(0, 1)
    ax.set_xlabel("Probabilité")
    ax.set_title("Probabilité par classe")
    st.pyplot(fig)
```

6.1 Utilisation de Ngrok pour le déploiement

Pour faciliter l'accès à notre application Streamlit depuis Internet sans hébergement sur un serveur distant, nous avons utilisé Ngrok, un outil qui crée un tunnel sécurisé entre notre environnement local et une URL publique temporaire. Voici les étapes pour utiliser Ngrok avec Streamlit :

1. Installation de Ngrok : Téléchargez et installez Ngrok sur votre machine.

2. Lancement de l'application Streamlit : Exécutez votre application Streamlit avec la commande suivante :

```
streamlit run streamlit.py
```

3. Création du tunnel Ngrok : Ouvrez un terminal et exécutez la commande suivante pour créer un tunnel vers le port 8501 (port par défaut de Streamlit) :

```
ngrok http 8501
```

4. Accès à l'application : Ngrok vous fournira une URL publique que vous pouvez utiliser pour accéder à votre application Streamlit depuis n'importe quel navigateur.

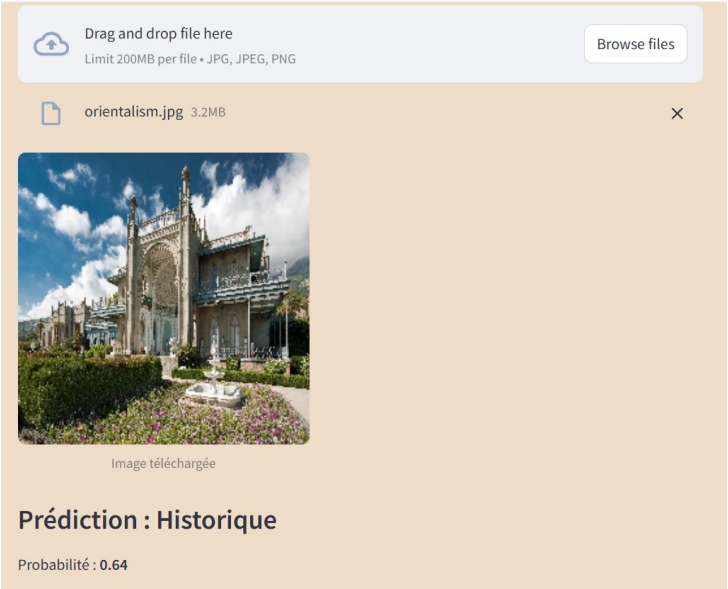
L'application est accessible via le lien suivant :
`https://51bf-34-125-123-187.ngrok-free.app/`

Cette méthode permet de partager facilement votre application avec d'autres personnes sans avoir à déployer votre code sur un serveur distant.

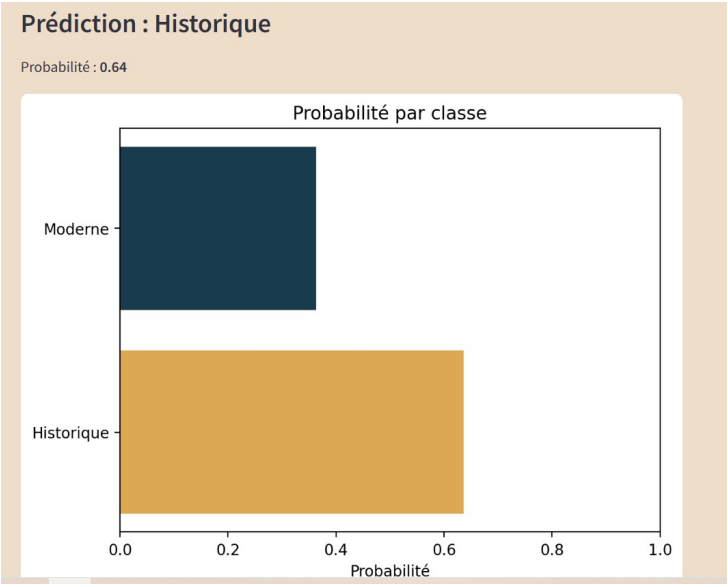
7 Interface Utilisateur

La capture d'écran suivante illustre le fonctionnement de l'application Streamlit à travers trois étapes clés :

(a) Interface de téléchargement



(b) Résultat de prédiction (Historique)



(c) Graphique des probabilités par classe

FIGURE 4 – Démonstration de l'application Streamlit

Explication :

— **Figure 4a** : Interface initiale permettant de téléverser une image (formats JPG/PNG) avec indication des limites techniques

— **Figure 4b** : Résultat affiché après analyse d'un bâtiment historique avec probabilité de 64

— **Figure 4c** : Visualisation interactive des scores de confiance pour les deux classes

Cette interface minimaliste permet une interaction intuitive tout en fournissant des informations détaillées sur le processus de classification.

8 Limites

— La classification binaire simplifie excessivement la richesse des styles architecturaux

— Le modèle reste sensible aux variations importantes d'angle et d'éclairage

— Les bâtiments hybrides ou rénovés posent un défi de classification

— L'ensemble de données pourrait bénéficier d'une plus grande diversité géographique

9 Applications potentielles

Ce projet ouvre la voie à plusieurs applications pratiques :

— Cartographie urbaine automatisée pour l'analyse de quartiers

— Outils d'aide à la préservation du patrimoine

— Systèmes de recommandation touristique basés sur les préférences architecturales

— Assistance aux évaluations immobilières

10 Conclusion et perspectives

Notre projet a démontré qu'un CNN relativement simple peut efficacement classer les bâtiments en catégories historiques et modernes avec une précision élevée. Les résultats confirment la pertinence des techniques de deep learning pour l'analyse architecturale.

10.1 Travaux futurs

— Affiner la classification en sous-catégories de styles architecturaux

— Développer une approche multi-modalité combinant analyse d'images et métadonnées (date de construction, localisation)

— Explorer des techniques d'apprentissage semi-supervisé pour exploiter des données non étiquetées

— Créer une application mobile permettant l'identification en temps réel du style architectural

10.2 Conclusion générale

Ce projet illustre la puissance des réseaux de neurones convolutifs pour la compréhension et l'analyse du patrimoine architectural. Au-delà de l'aspect technique, notre travail contribue à la préservation et à la valorisation du patrimoine bâti en facilitant son identification et sa catégorisation automatisées.

La distinction entre bâtiments historiques et modernes représente une première étape vers des systèmes plus sophistiqués capables d'analyser finement les caractéristiques architecturales et leur évolution à travers le temps et l'espace.

Webographie

— **Architectural Styles Dataset (dumitrux sur Kaggle) :** Base initiale du jeu de données utilisé.
https://www.kaggle.com/datasets/dumitrux/architectural-styles-dataset

— **Google Images :** Utilisé pour la collecte d'images supplémentaires. https://images.google.com/

— **TensorFlow Documentation :** Documentation officielle de la bibliothèque TensorFlow, utilisée pour le deep learning. https://www.tensorflow.org/api_docs

— **Keras Documentation :** Documentation de Keras, l'API de haut niveau pour TensorFlow. https://keras.io/

— **Streamlit Documentation :** Guide pour la création d'applications web interactives en Python. https://docs.streamlit.io/

— **Ngrok Official Website :** Service pour exposer des serveurs locaux sur Internet. https://ngrok.com/

— **Matplotlib Documentation :** Pour la création de graphiques et visualisations en Python. https://matplotlib.org/stable/contents.html

— **OpenCV Documentation :** Bibliothèque pour le traitement d'images, utilisée pour l'égalisation d'histogramme. https://docs.opencv.org/

— **Python Programming Language :** Site officiel du langage Python. https://www.python.org/

— **Stack Overflow :** Communauté pour développeurs, utile pour la résolution de problèmes techniques. https://stackoverflow.com/