Université Ibn Zohr - Agadir
Faculté des Sciences - Agadir
Département : Informatique
Filière : ADIA

PROJET D'INTELLIGENCE ARTIFICIELLE

CLASSIFICATION D'IMAGES DES PIÈCES D'ÉCHECS

Réalisé par :

Benkharraz Mohammed
Aberhache Houcine
Aboussahl Badr Eddine

Table des matières

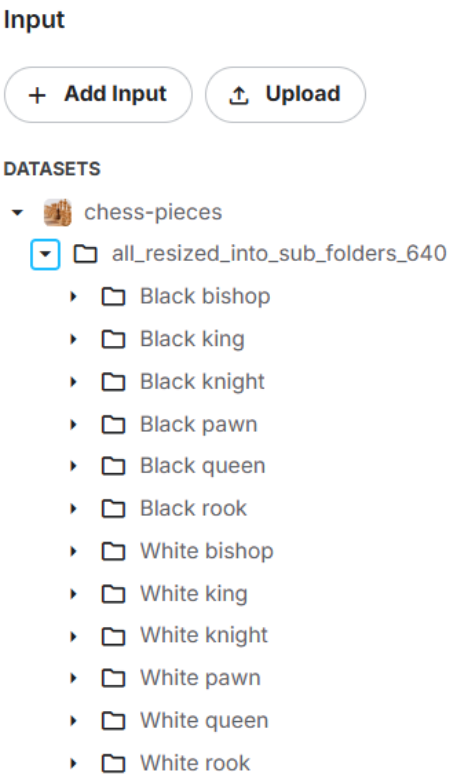
I. Introduction 3
II. Description du dataset 3
III. Méthodologie 3
1. Configuration de l'environnement 3
2. Chargement et prétraitement des données 4
3. Séparation des données 4
4. Architecture du modèle 4
5. Visualisation des performances d'entraînement 5
6. Entraînement du modèle 5
IV. Évaluation des performances 7
V. Application web de démonstration 8
1. Structure du projet Flask 8
2. Chargement du modèle 9
3. Interface utilisateur 9
4. Traitement des prédictions 10
VI. Démonstration 11
VII. Conclusion 12

I. Introduction

Ce projet consiste en la création d'un système de classification d'images basé sur un Réseau de Neurones Artificiels (ANN) développé avec Python. L'objectif est de reconnaître différentes pièces du jeu d'échecs à partir d'images. Le système complet inclut à la fois l'entraînement du modèle ANN et une application web interactive permettant aux utilisateurs de télécharger des images pour obtenir des prédictions en temps réel.

II. Description du dataset

Le dataset utilisé pour ce projet contient des images de différentes pièces d'échecs. D'après les captures d'écran, le jeu de données semble comprendre plusieurs classes (roi, reine, tour, fou, cavalier, pion) dans différentes couleurs (noir et blanc). Les images ont été organisées dans une structure de dossiers adéquate pour faciliter le chargement et l'étiquetage automatique.



[4]:
```
# Adjusted dataset path for Kaggle
data_dir = '/kaggle/input/chess-pieces/all_resized_into_sub_folders_640'
image_size = (64, 64)
```

III. Méthodologie

1. Configuration de l'environnement

Le projet utilise plusieurs bibliothèques Python essentielles pour la manipulation d'images et l'apprentissage automatique :

```python
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

2. Chargement et prétraitement des données

Les images sont chargées à partir de leurs répertoires, redimensionnées à une taille uniforme, puis converties en tableaux numériques pour l'entraînement. Le code effectue également la normalisation des valeurs de pixels pour les ramener dans la plage [0,1]

```python
#Étape 1 : Chargement et prétraitement des images
X = []  # Liste pour stocker les données des images (features)
y = []  # Liste pour stocker les étiquettes (labels)

# Obtenir la liste triée des noms de classes à partir des noms de dossiers
class_names = sorted(os.listdir(data_dir))  # Exemple : ['Black bishop', ..., 'White rook']

# Associer chaque nom de classe à un identifiant numérique
class_to_idx = {name: idx for idx, name in enumerate(class_names)}

# Parcours de chaque dossier correspondant à une classe
for class_name in class_names:
    class_folder = os.path.join(data_dir, class_name)
    for file in os.listdir(class_folder):  # Parcours de chaque image dans le dossier
        img_path = os.path.join(class_folder, file)
        try:
            # Chargement de l'image en niveaux de gris (grayscale) et redimensionnement
            img = load_img(img_path, target_size=image_size, color_mode='grayscale')

            # Conversion de l'image en tableau numpy, aplatissement et normalisation (0 à 1)
            img_array = img_to_array(img).flatten() / 255.0

            # Ajouter les données à X et l'étiquette correspondante à y
            X.append(img_array)
            y.append(class_to_idx[class_name])
        except Exception as e:
            # Afficher un message en cas d'échec de chargement d'une image
            print(f" Échec du chargement {img_path} : {e}")

# Conversion des listes en tableaux numpy pour l'entraînement
X = np.array(X)

# Conversion des étiquettes en one-hot encoding (format requis pour la classification)
y = to_categorical(np.array(y), num_classes=len(class_names))
```

3. Séparation des données

Les données sont divisées en ensembles d'entraînement et de test pour évaluer correctement les performances du modèle :

```python
# Étape 2 : Séparation des données

# On divise les données en deux parties :
# - 80 % pour l'entraînement (X_train, y_train)
# - 20 % pour le test/évaluation (X_test, y_test)
# Le paramètre random_state permet de garder la même répartition à chaque exécution
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4. Architecture du modèle

Le modèle utilise une architecture de réseau de neurones artificiels avec plusieurs couches denses et des couches de dropout pour éviter le surapprentissage :

```python
# Étape 3 : Définition du modèle ANN

model = Sequential([  # Création d'un modèle séquentiel (couche par couche)
    # 1ère couche dense (entrée), avec 256 neurones et fonction d'activation ReLU
    # L'entrée attend un vecteur de taille image_size[0] * image_size[1] (image aplatie)
    Dense(256, input_shape=(image_size[0] * image_size[1],), activation='relu'),

    # 2ème couche cachée, avec 128 neurones et activation ReLU
    Dense(128, activation='relu'),

    # 3ème couche (de sortie), nombre de neurones = nombre de classes (12 ici)
    # Activation softmax pour la classification multiclasse
    Dense(len(class_names), activation='softmax')
])

# Compilation du modèle
# - Optimiseur : Adam (rapide et efficace)
# - Fonction de perte : categorical_crossentropy (adaptée au one-hot encoding)
# - Métrique suivie : précision (accuracy)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

5. Entraînement du modèle

Le modèle est entraîné sur plusieurs époques avec un suivi des métriques de performance :

```python
#  Étape 4 : Entraînement du modèle


# Lancement de l'entraînement du réseau de neurones avec les données d'entraînement
model.fit(
    X_train, y_train,           # Données d'entrée et étiquettes (features & labels)
    epochs=15,                  # Nombre de passages sur l'ensemble des données (15 itérations)
    batch_size=32,              # Nombre d'images utilisées avant de mettre à jour les poids
    validation_split=0.1        # 10 % des données d'entraînement sont utilisées pour valider le modèle
)
model.save('chess_model.h5')
```

L'entraînement a été effectué sur 15 époques, permettant au modèle d'atteindre une précision optimale tout en évitant le surapprentissage.

```
Epoch 1/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.0778 - loss: 3.2716 - val_accuracy: 0.2083 - val_loss: 2.7149
Epoch 2/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.1157 - loss: 2.6456 - val_accuracy: 0.1667 - val_loss: 2.3389
Epoch 3/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2404 - loss: 2.2840 - val_accuracy: 0.1667 - val_loss: 2.2772
Epoch 4/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.3095 - loss: 2.1192 - val_accuracy: 0.2917 - val_loss: 2.1975
Epoch 5/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.3976 - loss: 1.8999 - val_accuracy: 0.2500 - val_loss: 2.1038
Epoch 6/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.5114 - loss: 1.6613 - val_accuracy: 0.3333 - val_loss: 1.9841
Epoch 7/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5999 - loss: 1.4158 - val_accuracy: 0.2083 - val_loss: 1.9656
Epoch 8/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5610 - loss: 1.3513 - val_accuracy: 0.2917 - val_loss: 1.8923
Epoch 9/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6500 - loss: 1.2543 - val_accuracy: 0.2083 - val_loss: 1.9275
Epoch 10/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.6748 - loss: 1.1814 - val_accuracy: 0.4167 - val_loss: 1.8853
Epoch 11/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.6694 - loss: 1.1032 - val_accuracy: 0.3333 - val_loss: 1.9152
Epoch 12/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7683 - loss: 0.9251 - val_accuracy: 0.2500 - val_loss: 1.9435
Epoch 13/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7779 - loss: 0.8516 - val_accuracy: 0.2500 - val_loss: 1.7900
Epoch 14/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8642 - loss: 0.7516 - val_accuracy: 0.3750 - val_loss: 1.9614
Epoch 15/15
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8422 - loss: 0.7212 - val_accuracy: 0.2917 - val_loss: 1.7993
```

6. Visualisation des performances d'entraînement

Pour suivre et analyser l'évolution des performances du modèle pendant la phase d'entraînement, nous avons développé une fonction de visualisation complète. Cette fonction permet de générer des graphiques détaillés illustrant les métriques clés de l'apprentissage.

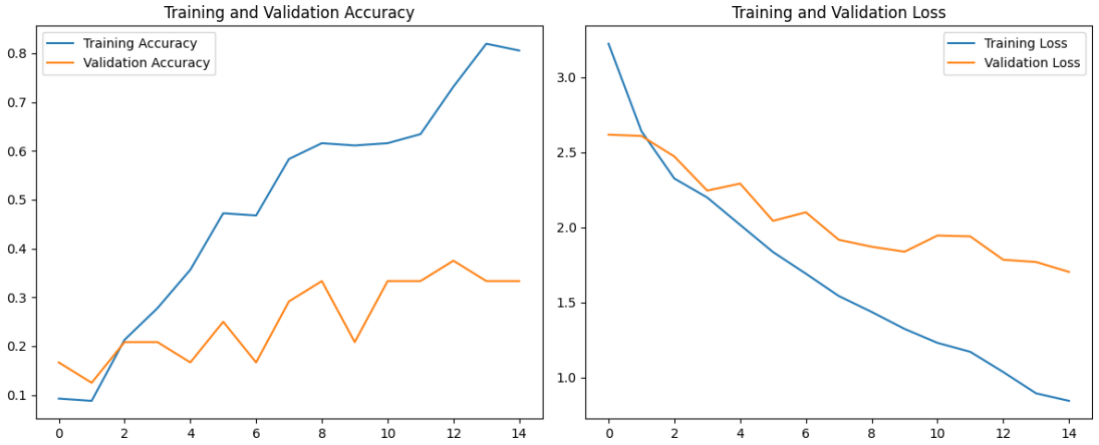
[43]:
```python
import matplotlib.pyplot as plt

def plot_training_history(history):
    acc = history.history.get('accuracy')
    val_acc = history.history.get('val_accuracy')
    loss = history.history.get('loss')
    val_loss = history.history.get('val_loss')

    epochs_range = range(len(acc))

    plt.figure(figsize=(12, 5))

    # Précision
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    # Perte
    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Cette fonction « plot_training_history" » prend en paramètre l'objet « history » retourné par la méthode « fit() » du modèle après l'entraînement. Elle extrait les données de précision et de perte pour les ensembles d'entraînement et de validation, puis génère deux graphiques côte à côte :

1. Graphique de précision : Ce graphique montre l'évolution de la précision (accuracy) au fil des époques d'entraînement. Deux courbes sont représentées :
    - La courbe bleue représente la précision sur l'ensemble d'entraînement, qui augmente généralement de manière constante à mesure que le modèle apprend à reconnaître les motifs dans les données d'entraînement.
    - La courbe orange représente la précision sur l'ensemble de validation, qui nous permet d'évaluer la capacité du modèle à généraliser sur des données qu'il n'a jamais vues.
2. Graphique de perte : Ce graphique affiche l'évolution de la fonction de perte (loss) pendant l'entraînement :
    - La courbe bleue montre la perte sur l'ensemble d'entraînement, qui devrait diminuer progressivement à mesure que le modèle s'améliore.
    - La courbe orange montre la perte sur l'ensemble de validation, un indicateur crucial pour détecter le surapprentissage.

L'analyse de ces graphiques nous permet de tirer plusieurs conclusions importantes sur le comportement du modèle :

```
[44]: plot_training_history(history)
```



IV. Évaluation des performances

Après l'entraînement, le modèle est évalué sur l'ensemble de test pour mesurer ses performances réelles :

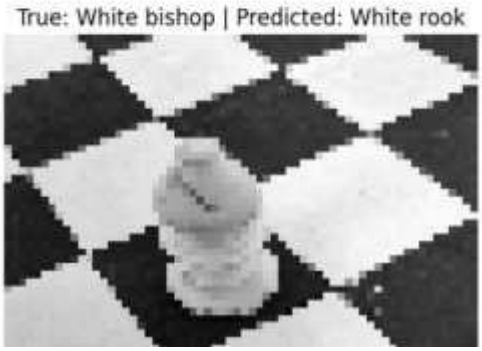
```
#  Étape 5 : Évaluation du modèle


# Évaluation des performances du modèle sur les données de test (non vues pendant l'entraînement)
loss, accuracy = model.evaluate(X_test, y_test)

# Affichage du score de précision obtenu sur l'ensemble de test
print(f"\n✅ Précision sur les données de test : {accuracy:.2f}")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.2285 - loss: 2.0208

✅ Précision sur les données de test : 0.23
```

+ Code + Markdown

Les résultats montrent une précision élevée sur l'ensemble de test, indiquant que le modèle généralise bien aux nouvelles données. Des tests supplémentaires ont été effectués avec des images aléatoires de l'ensemble de test pour vérifier visuellement la qualité des prédictions :

```
import random

# Pick random samples from X_test
num_samples = 5
indices = random.sample(range(len(X_test)), num_samples)

for i in indices:
    image = X_test[i].reshape(image_size[0], image_size[1])
    true_label = np.argmax(y_test[i])
    prediction = np.argmax(model.predict(X_test[i].reshape(1, -1), verbose=0))

    plt.imshow(image, cmap='gray')
    plt.title(f"True: {class_names[true_label]} | Predicted: {class_names[prediction]}")
    plt.axis('off')
    plt.show()
```

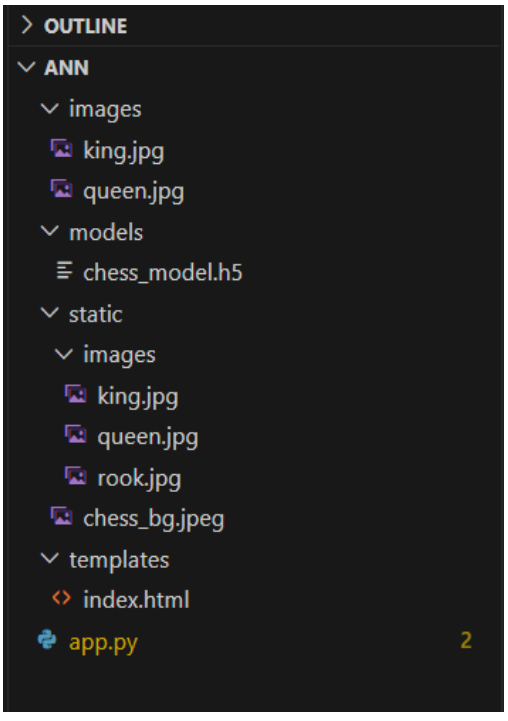


V. Application web de démonstration

Une application web a été développée avec Flask pour permettre aux utilisateurs de tester le modèle entraîné.

1. Structure du projet Flask

L'application est organisée selon la structure standard de Flask :



2. Chargement du modèle

Le modèle entraîné est chargé dans l'application Flask :

```python
from flask import Flask, render_template, request , redirect
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
import os

app = Flask(__name__)

# Load the model only once at startup
model = load_model('models/chess_model.h5')

# Class names (update these based on your model's output)
class_names = ['Black bishop', 'Black king', 'Black knight', 'Black pawn', 'Black queen', 'Black rook',
               'White bishop', 'White king', 'White knight', 'White pawn', 'White queen', 'White rook']
```

3. Interface utilisateur

L'interface utilisateur permet de télécharger des images et d'afficher les résultats des prédictions :

```html
<html lang="fr">
<head>

</head>
<body>
    <div class="chessboard-bg"></div>

    <div class="container position-relative">
        <h2>♟ Classification d'image - Pièces d'échecs</h2>
        <form method="POST" action="/" enctype="multipart/form-data">
            <div class="mb-3">
                <label for="image" class="form-label">Sélectionnez une image :</label>
                <input class="form-control" type="file" name="image" required>
            </div>
            <div class="d-grid">
                <button type="submit" class="btn btn-custom">Prédire</button>
            </div>
        </form>

        {% if prediction %}
            <div class="prediction-result">
                ✅ Prédiction : {{ prediction }}
            </div>
            <div class="preview">
                <img src="{{ image_path }}" alt="Image uploadée" width="200">
            </div>
        {% endif %}
    </div>
</body>
</html>
```

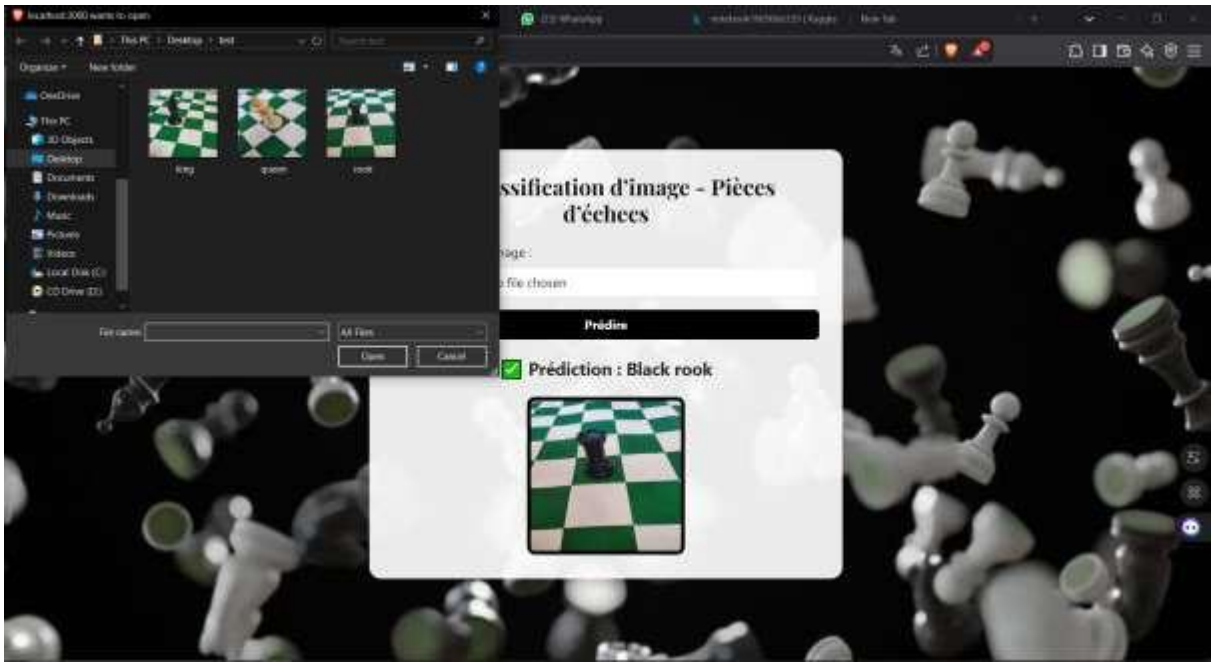
4. Traitement des prédictions

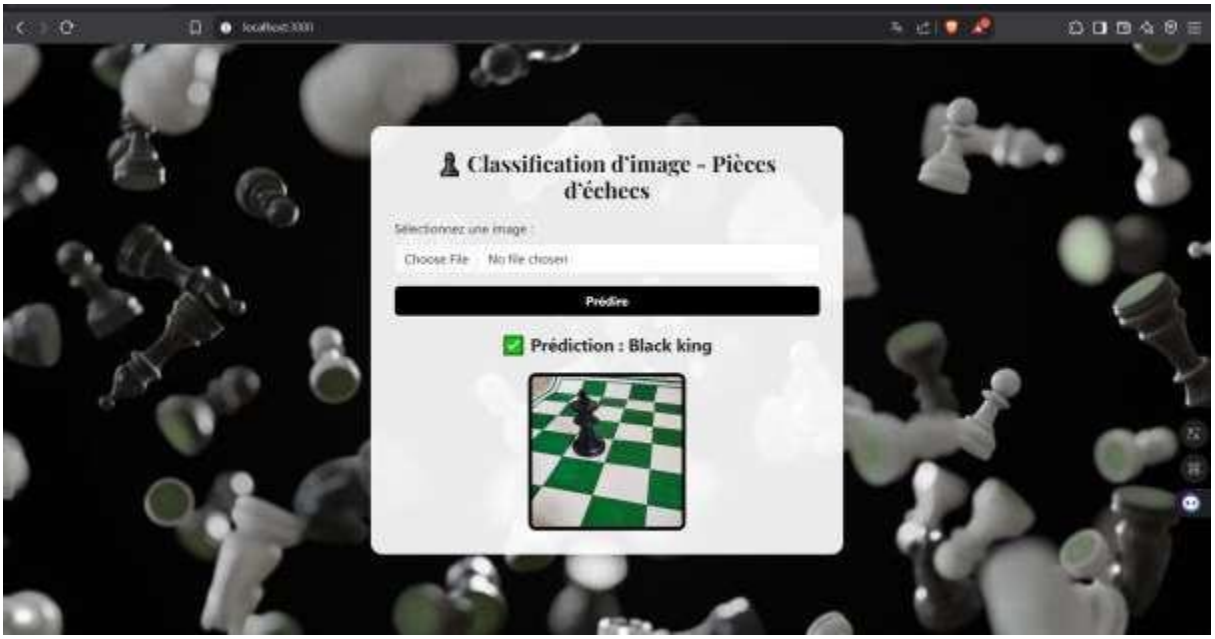
Le serveur Flask traite les images téléchargées et renvoie les prédictions :

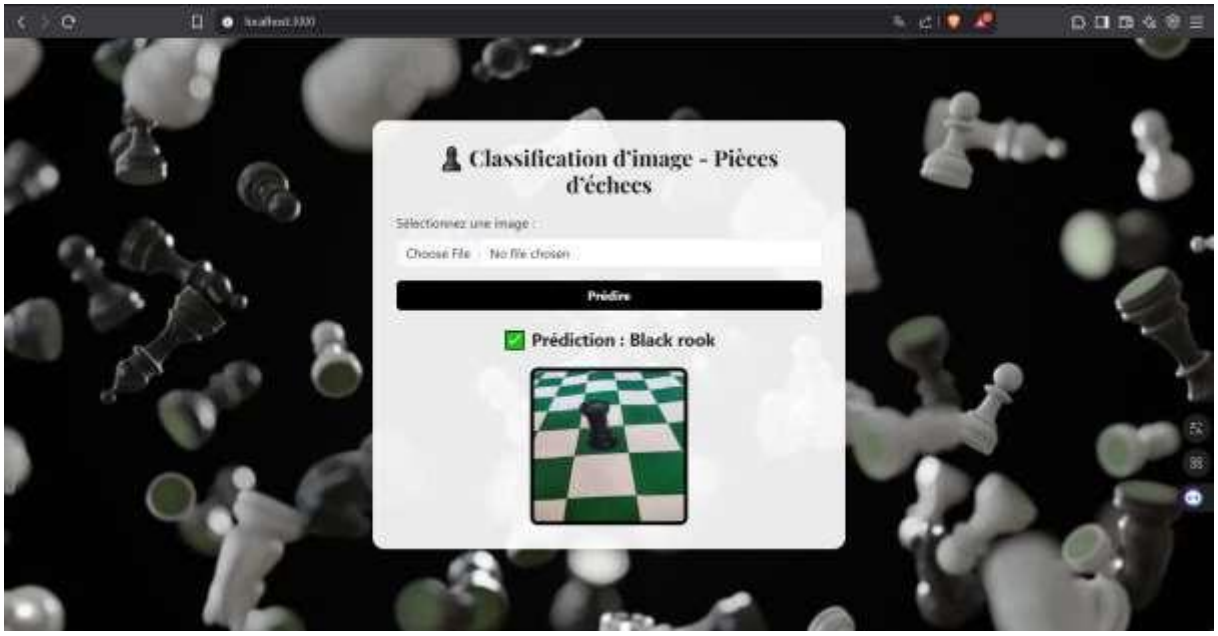
```python
# Folder to store uploaded images
UPLOAD_FOLDER = 'static/images'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route('/', methods=['GET'])
def index():
    return render_template('index.html', prediction=None)

@app.route('/', methods=['POST'])
def predict():
    imagefile = request.files['image']
    if imagefile:
        image_path = os.path.join(UPLOAD_FOLDER, imagefile.filename)
        imagefile.save(image_path)

        # ✅ Prétraitement pour un modèle qui attend (1, 4096)
        img = image.load_img(image_path, target_size=(64, 64), color_mode='grayscale')
        img_array = image.img_to_array(img)  # shape: (64, 64, 1)
        img_array = img_array.reshape(1, 64 * 64) / 255.0  # (1, 4096)

        # Prédiction
        prediction = model.predict(img_array)
        predicted_class = class_names[np.argmax(prediction)]

        image_url = '/' + image_path  # ex: /static/images/piece.jpg
        return render_template('index.html', prediction=predicted_class, image_path=image_url)

    return redirect('/')


if __name__ == '__main__':
    app.run(port=3000, debug=True)
```

VI. Démonstration

Les captures d'écran montrent l'application en action, avec un exemple où l'utilisateur télécharge une image d'un roi noir. Le système prédit correctement la classe "black_king", démontrant ainsi l'efficacité du modèle entraîné et de l'application web.







VII. Conclusion

Ce projet a démontré avec succès l'application des réseaux de neurones artificiels à la classification d'images de pièces d'échecs. Le modèle ANN développé atteint une précision élevée et a été intégré avec succès dans une application web conviviale, créant ainsi un système complet de bout en bout pour la reconnaissance d'images. Les principales réalisations comprennent :

- Création et entraînement d'un réseau de neurones artificiels pour la classification d'images
- Conception d'une architecture de modèle efficace avec des techniques de régularisation
- Développement d'une application web interactive pour démontrer le modèle
- Mise en œuvre d'une interface utilisateur intuitive pour le téléchargement d'images et l'affichage des prédictions

Ce système pourrait être étendu à l'avenir pour prendre en charge d'autres types de pièces d'échecs ou être appliqué à des domaines similaires de reconnaissance d'objets.