CIA-2 : NEURAL NETWORKS AND DEEP LEARNING

Submitted by, Riswi Ayub (2382445)

Problem: Implement Image processing in you own dataset, save the model after trainning. use the saved model to create a simple UI using Grado/ streamlet/ Flask.

Minimum 5 class, each class 30 orginal image argument the 30 into more.

Fabric Pattern Classification Project: Introduction

This project focuses on developing a machine learning model to classify fabric patterns into five distinct categories: **Animal Print**, **Polka Dots**, **Floral**, **Checkered**, and **Stripes**. The goal was to build an end-to-end pipeline for image classification, leveraging deep learning techniques and evaluating the model's performance to ensure its effectiveness. Below is an overview of the steps undertaken in this project:

---

1. **Data Collection and Organization**

- A custom dataset was prepared with fabric pattern images categorized into the five mentioned classes.
- The dataset was organized into **Train**, **Validation**, and **Test** folders, following a structured hierarchy for efficient model training and evaluation.
- Each folder contained subfolders representing the classes, with the images distributed as follows:
  - **Train**: 20 images per class
  - **Validation**: 5 images per class
  - **Test**: 5 images per class

---

2. **Data Augmentation and Preprocessing**

- To increase the diversity and robustness of the model, data augmentation techniques such as **rotation**, **scaling**, **shifting**, and **flipping** were applied to the training images.
- All images were resized to a uniform dimension of 150x150 pixels and normalized to scale pixel values between 0 and 1 for better model performance.

---

3. **Model Development**

- A convolutional neural network (CNN) was designed using TensorFlow and Keras. The architecture included:
  - Three convolutional layers with increasing filters to extract hierarchical image features.
  - MaxPooling layers to reduce spatial dimensions and computational complexity.
  - A fully connected layer for feature integration and a dropout layer for regularization.
  - A softmax layer to output probabilities for the five fabric pattern classes.
- The model was compiled with the Adam optimizer and categorical cross-entropy loss function, suitable for multi-class classification tasks.

---

4. **Model Training**

- The model was trained on the augmented training data for 10 epochs while monitoring its performance on the validation set.
- Training accuracy and loss trends were observed to ensure that the model was learning effectively without overfitting.

---

5. **Model Evaluation**

- The model's performance was assessed on the training, validation, and test datasets. Key metrics, including accuracy, were calculated to measure its ability to generalize to unseen data.

---

6. **Deployment with Gradio**

- A user-friendly interface was built using Gradio, allowing users to upload fabric images and obtain real-time predictions.
- The interface provided a confidence score for each pattern, highlighting the model's decision-making process.

---

7. **Testing and Debugging**

- The model was tested with various sample images to verify its predictive capabilities.
- Issues such as potential biases or incorrect predictions were analyzed, and adjustments were made to improve accuracy and reliability.

Outcome

The project successfully created a fabric pattern classification model, demonstrating the application of deep learning techniques in a practical domain. The deployment interface further ensured ease of use, making the model accessible for real-world applications like fabric design categorization or retail inventory management.

In [2]:
```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
import numpy as np
import gradio as gr
```

In [3]:
```
# Directories for dataset

data_dir = "/Users/riswee/Desktop/Fabric_patterns"
train_dir = os.path.join(data_dir, "Train")
validation_dir = os.path.join(data_dir, "Validation")
test_dir = os.path.join(data_dir, "Test")
```

In [4]:
```
# Data augmentation and generators

train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

In [5]:
```
validation_datagen = ImageDataGenerator(rescale=1.0 / 255)
test_datagen = ImageDataGenerator(rescale=1.0 / 255)
```

In [6]:
```
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 21 images belonging to 5 classes.
```

In [7]:
```
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 10 images belonging to 5 classes.
```

In [8]:
```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 6 images belonging to 5 classes.
```

In [9]:
```
# Model architecture

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(5, activation='softmax')
])
```

```
/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/keras/src/layers/c
onvolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
input_dim` argument to a layer. When using Sequential models, prefer using
an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [10]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [11]:
```
# Model training

history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

```
/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/keras/src/trainer
s/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
lass should call `super().__init__(**kwargs)` in its constructor. `**kwarg
s` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not
pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
```

```
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.2381 - loss: 1.5603
/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.2381 - loss: 1.5603 - val_accuracy: 0.5000 - val_loss: 6.0034
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.7632 - val_accuracy: 0.5000 - val_loss: 5.3237
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.5301 - val_accuracy: 0.5000 - val_loss: 3.2006
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.3644 - val_accuracy: 0.5000 - val_loss: 1.7333
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.3383 - val_accuracy: 0.5000 - val_loss: 1.6405
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.3888 - val_accuracy: 0.5000 - val_loss: 2.0689
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.3287 - val_accuracy: 0.5000 - val_loss: 2.4812
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.3062 - val_accuracy: 0.5000 - val_loss: 2.4464
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.3620 - val_accuracy: 0.5000 - val_loss: 1.9142
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.9048 - loss: 0.2903 - val_accuracy: 0.5000 - val_loss: 1.5114
```

In [12]:
```python
# Evaluate the model

train_loss, train_acc = model.evaluate(train_generator, verbose=0)
val_loss, val_acc = model.evaluate(validation_generator, verbose=0)
test_loss, test_acc = model.evaluate(test_generator, verbose=0)

print(f"Training Accuracy: {train_acc:.2f}")
print(f"Validation Accuracy: {val_acc:.2f}")
print(f"Test Accuracy: {test_acc:.2f}")
```

```
Training Accuracy: 0.90
Validation Accuracy: 0.50
Test Accuracy: 0.67
```

In [13]:
```python
# Save the model
```

```
model.save("fabric_pattern_classifier.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

In [14]:
```
# Load model and test prediction

loaded_model = tf.keras.models.load_model("fabric_pattern_classifier.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

In [15]:
```
# Function to predict pattern and print result

def predict_pattern(image_path):
    print(f"Loading image from: {image_path}")
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(
    img_array = tf.keras.preprocessing.image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    print("Image preprocessed. Predicting...")
    predictions = loaded_model.predict(img_array)
    class_names = list(train_generator.class_indices.keys())
    predicted_class = class_names[np.argmax(predictions)]
    confidence = np.max(predictions)
    print(f"Predicted Pattern: {predicted_class}, Confidence: {confidence
    return predicted_class, confidence
```

In [16]:
```
# Example usage of prediction

def example_usage():
    example_image_path = "/Users/riswee/Desktop/polkaa.jpeg"  # Replace w
    if not os.path.exists(example_image_path):
        print(f"Error: File not found at {example_image_path}")
        return
    predict_pattern(example_image_path)
```

In [17]:
```
# Gradio Interface
def gradio_predict(image):
    img = tf.image.resize(image, (150, 150)) / 255.0
    img = np.expand_dims(img, axis=0)
    predictions = loaded_model.predict(img)
    class_names = list(train_generator.class_indices.keys())
    return {class_names[i]: float(predictions[0][i]) for i in range(5)}
```

In [18]:
```
interface = gr.Interface(
    fn=gradio_predict,
    inputs=gr.Image(type="pil"),
    outputs=gr.Label(num_top_classes=5),
    title="Fabric Pattern Classifier",
    description="Upload a fabric image to classify its pattern: Animal Pr
```

```
)
```

In [19]:
```
if __name__ == "__main__":
    example_usage()  # Uncomment to test with an example image
    interface.launch()
```

```
Loading image from: /Users/riswee/Desktop/polkaa.jpeg
Image preprocessed. Predicting...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
Predicted Pattern: Polka_dots, Confidence: 0.99
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
```

```
Traceback (most recent call last):
  File "/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/gradio/que
ueing.py", line 536, in process_events
    response = await route_utils.call_process_api(
  File "/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/gradio/rou
te_utils.py", line 322, in call_process_api
    output = await app.get_blocks().process_api(
  File "/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/gradio/blo
cks.py", line 1931, in process_api
    inputs = await self.preprocess_data(
  File "/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/gradio/blo
cks.py", line 1662, in preprocess_data
    processed_input.append(block.preprocess(inputs_cached))
  File "/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/gradio/com
ponents/image.py", line 190, in preprocess
    im = PIL.Image.open(file_path)
  File "/Users/riswee/opt/anaconda3/lib/python3.9/site-packages/PIL/Image.
py", line 3498, in open
    raise UnidentifiedImageError(msg)
PIL.UnidentifiedImageError: cannot identify image file '/private/var/folde
rs/5h/_4my78qx1sg1gr1gwsnpk83w0000gn/T/gradio/12a8f2e1844910f8715bff99b101
b0d4cd719bf32fc99f84e1fd242f5079cdd0/stripes.avif'
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
```

Analysis of Model Performance

Based on the results:

- **Training Accuracy: 0.90 (90%)**
  - The model performs well on the training data, achieving high accuracy. This indicates that the model has learned the patterns from the training dataset effectively.
- **Validation Accuracy: 0.50 (50%)**
  - A significant drop compared to the training accuracy suggests that the model is overfitting. It performs well on the training data but struggles to generalize to unseen data in the validation set.
- **Test Accuracy: 0.67 (67%)**
  - The test accuracy is better than the validation accuracy, which is unusual. However, this could be due to the small size of the test dataset (only 5 images per class), which may not fully represent the real-world distribution.

Possible Issues and Recommendations:

1. **Overfitting**

- The gap between training accuracy (90%) and validation accuracy (50%) suggests overfitting. The model is too specialized for the training data.
- **Recommendations**:
  - Increase the size of the training dataset with more diverse images for each class.
  - Use stronger regularization techniques such as higher dropout rates or L2 regularization.
  - Experiment with reducing the model complexity by decreasing the number of layers or neurons.

2. **Class Imbalance**

   - If certain classes dominate the dataset, the model might focus more on those classes, leading to biased predictions.
   - **Recommendations**:
     - Ensure that the dataset is balanced with equal representation of each class.
     - Apply class weights during model training to account for imbalances.

3. **Data Augmentation**

   - The data augmentation techniques used might not fully capture the variations in fabric patterns.
   - **Recommendations**:
     - Experiment with additional augmentation techniques, such as random cropping, brightness adjustments, and contrast variations.

4. **Validation and Test Dataset Size**

   - The validation and test datasets are small, with only 5 images per class. This can lead to unreliable accuracy metrics.
   - **Recommendations**:
     - Increase the size of both validation and test datasets for better evaluation.
     - Perform k-fold cross-validation to get more reliable performance metrics.

5. **Learning Rate and Epochs**

   - The model was trained for only 10 epochs. It's possible that the model did not converge properly.
   - **Recommendations**:
     - Experiment with a learning rate schedule or train for more epochs while monitoring the validation accuracy.

6. **Feature Learning**

   - The convolutional layers may not be capturing sufficient feature diversity

due to suboptimal filter configurations or pooling strategies.

- **Recommendations**:
  - Add more convolutional layers or increase the number of filters.
  - Use pre-trained models like VGG16 or MobileNet for transfer learning to leverage existing knowledge.

In [ ]:

In [ ]: