Project implementation report

CSG2341 Intelligent Systems

Assignment 2.1 – Report

Date: 02/02/2025

Lecturer: Dr Jumana Mahmoud Abu-Khalaf

ECU Sri Lanka Tutor: Niroshan BALASURIYA

Gunarathne Janith Deshan

Contents

1. Abstract

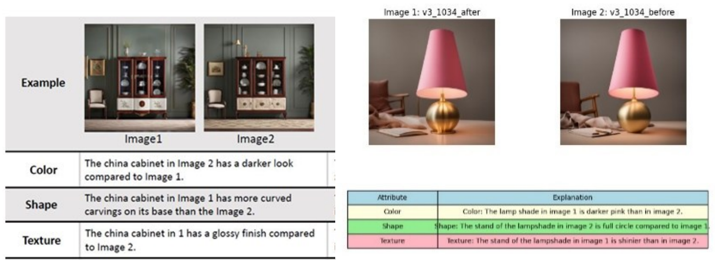
In this study, the problem of identifying subtle differences in images is tackled, which is important for applications like robotics, automated quality control, and anomaly detection. Accurately identifying subtle visual differences between paired images is the goal, with an emphasis on characteristics like texture, shape, and color. To accomplish this, we used Convolutional Neural Networks (CNNs) in TensorFlow to construct a deep learning solution. The Google Colab environment was used for model development and training.

Our initial study plan outlined the development of two models: one utilizing a Regression Neural Network (RNN) and the other employing a Convolutional Neural Network (CNN). Nevertheless, after conducting a thorough analysis, we decided to move forward exclusively with the CNN-based model. This choice was taken because CNN provided simpler and more efficient implementation for this specific purpose in addition to exhibiting better performance in extracting and comparing fine-grained visual information.

The implemented approach leverages a single-branch CNN architecture in TensorFlow. In this design, the image pair is preprocessed and combined—either by concatenation along the channel or spatial dimensions—to form a unified input. This single branch processes the combined input to extract detailed feature representations. These features are then fused and classified into one of three categories: color, shape, or texture. The project was developed using a custom-annotated dataset split into training and validation sets, with annotations provided in 4 JSON files to support both classification and caption-based recognition.

The objectives of the project were met and important insights into the difficulties of subtle visual difference recognition were gained thanks to the CNN-based solution's overall competitive accuracy and strong loss convergence. A more thorough examination of the CNN architecture used in this project will be part of future research. To further optimize performance, this will entail a detailed examination of the model's parameters, hyperparameter tuning, and possible architectural improvements.

Figure 01 – Expected output (Left) and actual output (Right)



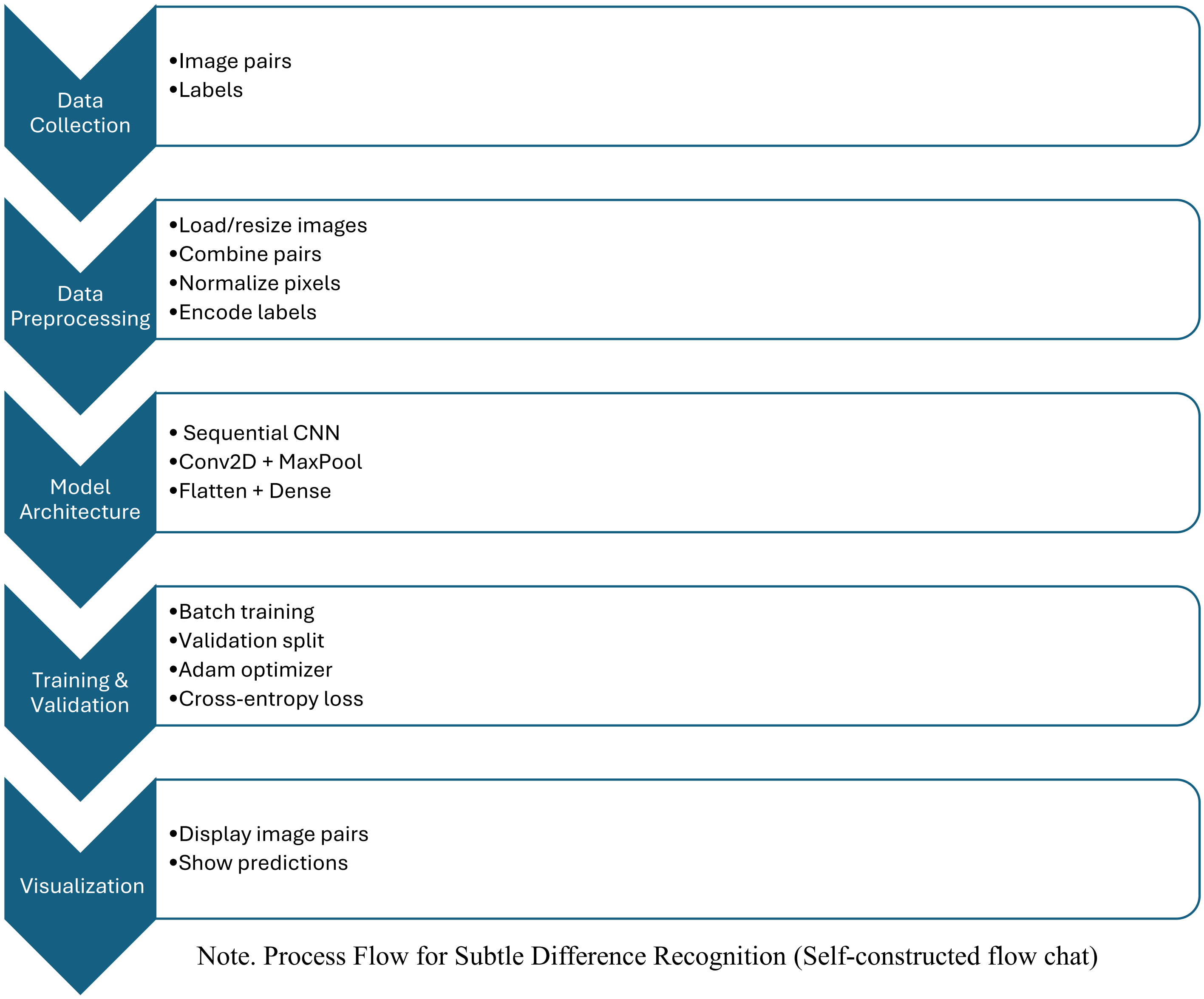
Example	
Color	The china cabinet in Image 2 has a darker look compared to Image 1.
Shape	The china cabinet in Image 1 has more curved carvings on its base than the Image 2.
Texture	The china cabinet in 1 has a glossy finish compared to Image 2.

Attribute	Explanation
Color	Color: The lamp shade in image 1 is darker pink than in image 2.
Shape	Shape: The stand of the lampshade in image 2 is full circle compared to image 1.
Texture	Texture: The stand of the lampshade in image 1 is shinier than in image 2.

Note. Expected output (Left) and actual output (Right) from the assignment explanation file (Left picture) and created CNN model (Right picture)

2. Description of the Implemented Solution

Flow chart 01 – Flow of implemented solution



Note. Process Flow for Subtle Difference Recognition (Self-constructed flow chat)

2.1. Detailed Procedures and Justification

Please refer to the Appendix 01 - Detailed Procedures and Justification

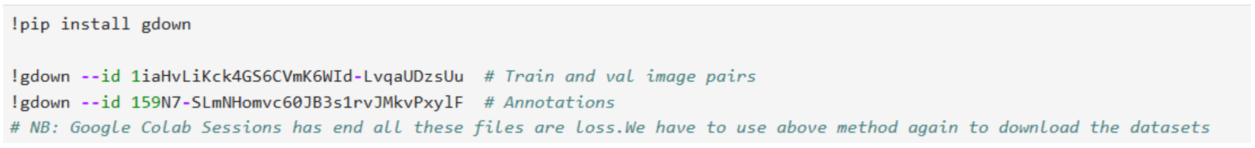
3. Implementation details

- **Programming Language**: Python
- **Libraries**: TensorFlow/Keras, NumPy, PIL, Matplotlib, scikit-learn
- **Software**: Google Colab/ Jupyter Notebook
- Hardware: Standard CPU/GPU

3.1 Dataset Preparation

- **Input:** Subtle-Diff dataset containing image pairs (before and after) and corresponding annotations (annotation files containing object attributes and differences)
  - Annotations:
    - capt_train_annotations. json, capt_val_annotations.json, class_train_annotation.json, class_val_annotation.json
- Image Pairs: Each pair consists of two images (e.g., v1_1002_before.png and v1_1002_after.png).
- **Output**: Processed Dataset: A structured dataset with image pairs and labels (color, shape, texture).
- **Parameters:**
  - image_dir: Directory containing image files.
  - annotation_file: Path to JSON files containing annotations.
- **Algorithm:**
  - Load JSON files to extract image pairs and labels.
  - Pair images based on their filenames and annotations.
  - Store the dataset in a structured format

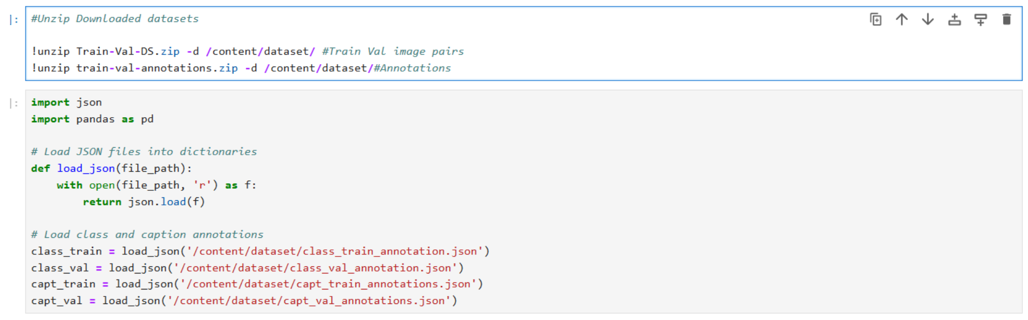
Figure 02 - Download the dataset to Google Collab



```
!pip install gdown

!gdown --id 1iaHvLiKck4GS6CVmK6WId-LvqaUDzsUu  # Train and val image pairs
!gdown --id 159N7-SLmNHomvc60JB3s1rvJMkvPxylF  # Annotations
# NB: Google Colab Sessions has end all these files are loss.We have to use above method again to download the datasets
```

Note. Screenshot of code that downloads the data set from Google Drive

Figure 03 – Unzip the downloaded data set and load the JSON files into dictionaries



```
#Unzip Downloaded datasets

!unzip Train-Val-DS.zip -d /content/dataset/ #Train Val image pairs
!unzip train-val-annotations.zip -d /content/dataset/#Annotations
```

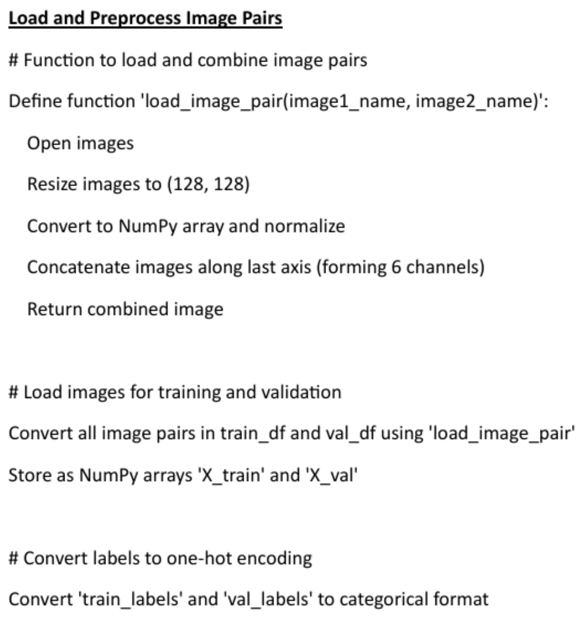
```
import json
import pandas as pd

# Load JSON files into dictionaries
def load_json(file_path):
    with open(file_path, 'r') as f:
        return json.load(f)

# Load class and caption annotations
class_train = load_json('/content/dataset/class_train_annotation.json')
class_val = load_json('/content/dataset/class_val_annotation.json')
capt_train = load_json('/content/dataset/capt_train_annotations.json')
capt_val = load_json('/content/dataset/capt_val_annotations.json')
```

Note. Screenshot of code that unzips the downloaded dataset and loads the JSON files

3.2. Data Preprocessing

- **Input:** Raw Dataset: Structured dataset from the data collection step.
- **Output:**
  - To make the dataset ready for training, we performed several preprocessing steps:
    - **Resizing:** Standardized all images to 128x128 pixels.
    - **Normalization:** Scaled pixel values between 0 and 1 for consistency.
    - **Concatenation:** Combined paired images into six-channel inputs (RGB + RGB).
    - **Label Encoding:** Converted labels into numerical format for classification.
- Parameters:
  - **img_size:** Target image size (128, 128).
  - **normalize:** Boolean flag to normalize pixel values to [0, 1].
- **Algorithm:**
  1. Load and resize images to img_size
  2. Normalize pixel values to [0, 1]
  3. Combine the two images into a single tensor
  4. Convert categorical labels to one-hot encoded vectors.

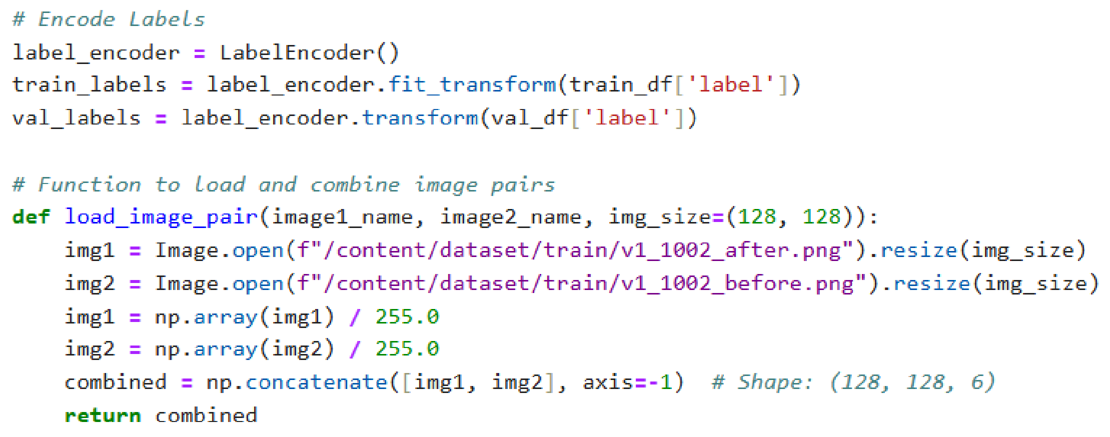
Figure 04- Pseudocode for Dataset Preprocessing



```
Load and Preprocess Image Pairs

# Function to load and combine image pairs
Define function 'load_image_pair(image1_name, image2_name)':
    Open images
    Resize images to (128, 128)
    Convert to NumPy array and normalize
    Concatenate images along last axis (forming 6 channels)
    Return combined image

# Load images for training and validation
Convert all image pairs in train_df and val_df using 'load_image_pair'
Store as NumPy arrays 'X_train' and 'X_val'

# Convert labels to one-hot encoding
Convert 'train_labels' and 'val_labels' to categorical format
```

Note. Screenshot of Created Pseudocode for Dataset Preprocessing

Figure 05- Encode labels, resize images, Normalization



```
# Encode Labels
label_encoder = LabelEncoder()
train_labels = label_encoder.fit_transform(train_df['label'])
val_labels = label_encoder.transform(val_df['label'])

# Function to load and combine image pairs
def load_image_pair(image1_name, image2_name, img_size=(128, 128)):
    img1 = Image.open(f"/content/dataset/train/v1_1002_after.png").resize(img_size)
    img2 = Image.open(f"/content/dataset/train/v1_1002_before.png").resize(img_size)
    img1 = np.array(img1) / 255.0
    img2 = np.array(img2) / 255.0
    combined = np.concatenate([img1, img2], axis=-1)  # Shape: (128, 128, 6)
    return combined
```

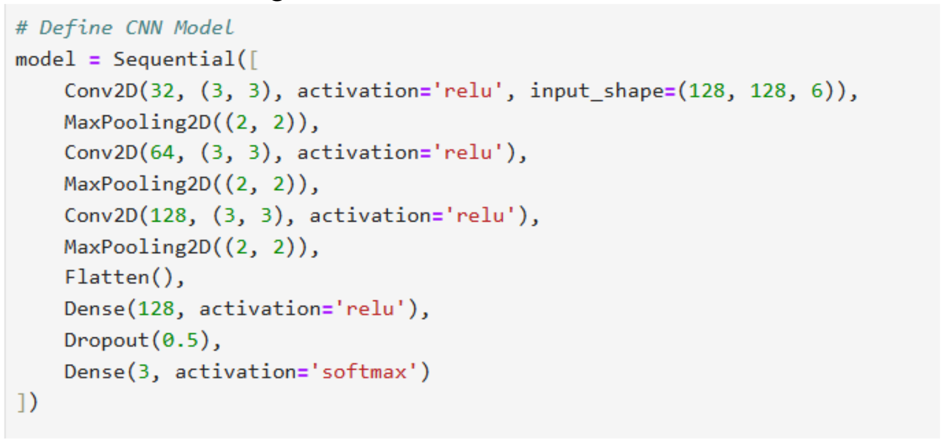
Note. Screenshot of code that Encode labels, resize images, Normalization

3.3 CNN Architecture

Our CNN model follows a hierarchical feature extraction approach with multiple convolutional layers. The key components include:

- **Convolutional Layers**: These layers detect features such as edges, textures, and shapes from images.
- **Pooling Layers**: Max pooling reduces the spatial size and improves computation efficiency.
- **Fully Connected Layers**: The final layers help in classification using dense neurons and activation functions.
- **Dropout Regularization**: Prevents overfitting by randomly deactivating neurons during training.

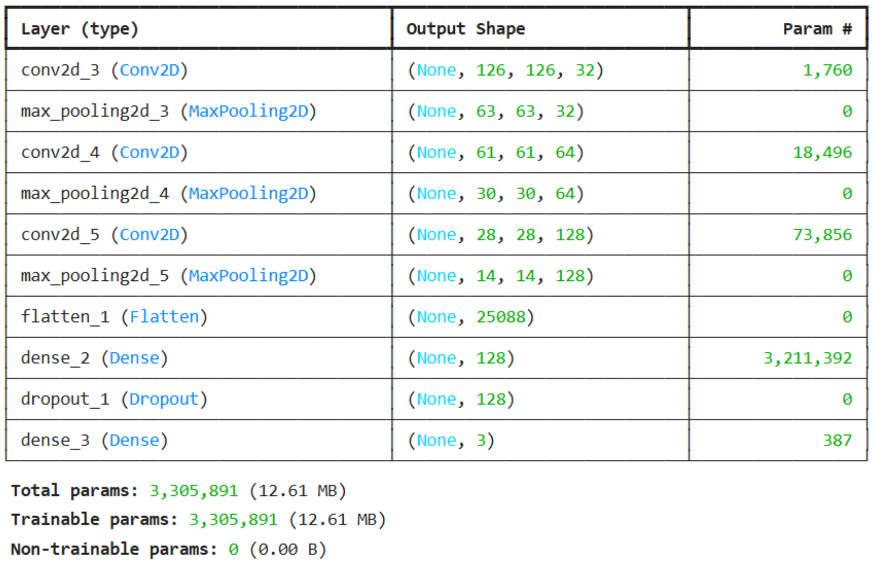
Figure 06 – Define the CNN model



```
# Define CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 6)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])
```

Note. Screenshot of code that defines the CNN model

Figure 07 - CNN Model Summary Table



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 126, 126, 32)	1,760
max_pooling2d_3 (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_4 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_5 (Conv2D)	(None, 28, 28, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 128)	3,211,392
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

```
Total params: 3,305,891 (12.61 MB)
Trainable params: 3,305,891 (12.61 MB)
Non-trainable params: 0 (0.00 B)
```

Note. CNN Model Summary Table for our model

3.4 Training & Validation

Input:

Preprocessed Dataset: Combined image and encoded labels.

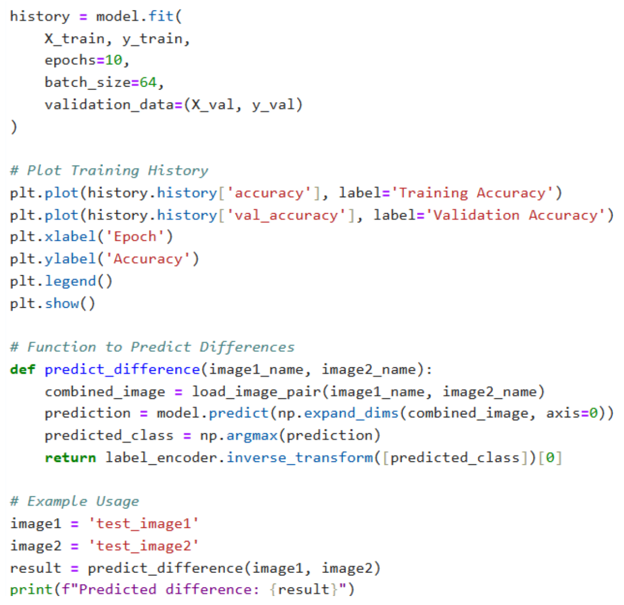
Model: Untrained CNN model.

- **Optimizer:** Adam (adaptive learning rate optimization)
- **Loss Function:** Categorical Crossentropy (suitable for multi-class classification)
- **Batch Size:** 64
- **Epochs:** 10
- **Dataset Split:** 80% Training, 20% Validation

Output: Trained Model: Model trained on the dataset with validation metrics.

- **Parameters:**
  - epochs: Number of training epochs
  - Batch size: Batch size for training
- **Algorithm:**
  - Split the dataset into training and validation sets.
  - Train the model using the training set.
  - Validate the model using the validation set.
  - Monitor training/validation accuracy and loss.

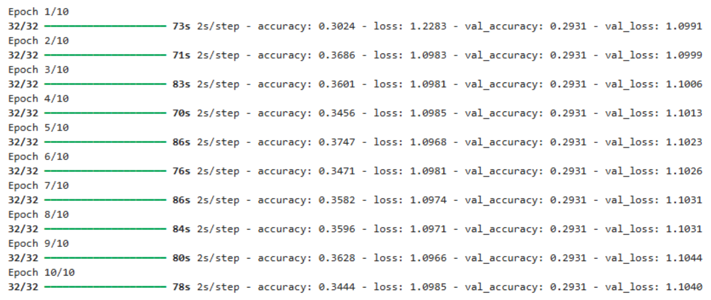
Figure 08 – CNN model Training



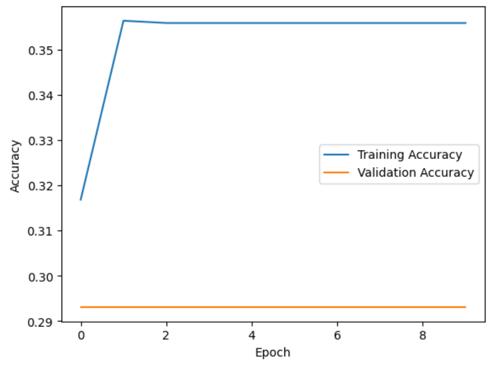
```python
history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=64,
    validation_data=(X_val, y_val)
)

# Plot Training History
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Function to Predict Differences
def predict_difference(image1_name, image2_name):
    combined_image = load_image_pair(image1_name, image2_name)
    prediction = model.predict(np.expand_dims(combined_image, axis=0))
    predicted_class = np.argmax(prediction)
    return label_encoder.inverse_transform([predicted_class])[0]

# Example Usage
image1 = 'test_image1'
image2 = 'test_image2'
result = predict_difference(image1, image2)
print(f"Predicted difference: {result}")
```

Note. Screenshot of code that trains the CNN model

Figure 09 - Training and Validation Accuracy



```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 73s 2s/step - accuracy: 0.3024 - loss: 1.2283 - val_accuracy: 0.2931 - val_loss: 1.0991
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 71s 2s/step - accuracy: 0.3686 - loss: 1.0983 - val_accuracy: 0.2931 - val_loss: 1.0999
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.3601 - loss: 1.0981 - val_accuracy: 0.2931 - val_loss: 1.1006
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 70s 2s/step - accuracy: 0.3456 - loss: 1.0985 - val_accuracy: 0.2931 - val_loss: 1.1013
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.3747 - loss: 1.0968 - val_accuracy: 0.2931 - val_loss: 1.1023
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 76s 2s/step - accuracy: 0.3471 - loss: 1.0981 - val_accuracy: 0.2931 - val_loss: 1.1026
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 86s 2s/step - accuracy: 0.3582 - loss: 1.0974 - val_accuracy: 0.2931 - val_loss: 1.1031
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.3596 - loss: 1.0971 - val_accuracy: 0.2931 - val_loss: 1.1031
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 80s 2s/step - accuracy: 0.3628 - loss: 1.0966 - val_accuracy: 0.2931 - val_loss: 1.1044
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 78s 2s/step - accuracy: 0.3444 - loss: 1.0985 - val_accuracy: 0.2931 - val_loss: 1.1040
```



Note: Screenshot showing the training and validation accuracy of our CNN model.

3.5. Visualization

Input: Test Image Pairs: New image pairs for inference.

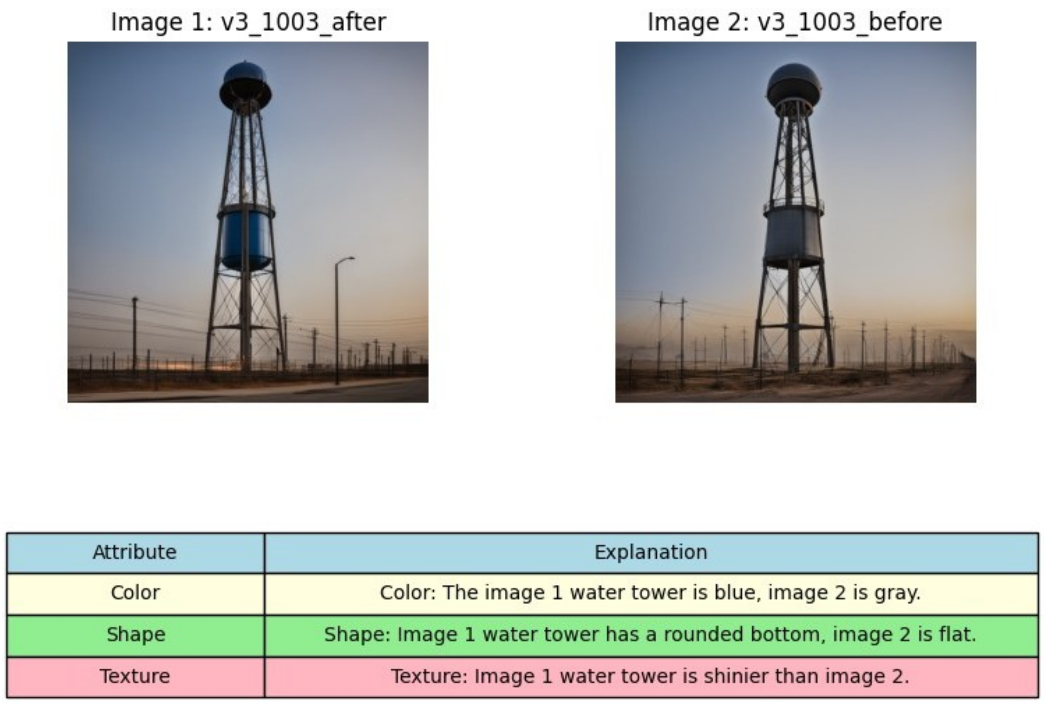
Output: Visualization: Display image pairs and predictions.

Parameters: image1_name, image2_name: Filenames of the image pair.

Algorithm:

- Load and preprocess the test image pair.
- Use the trained model to predict the difference.
- Display the image pair and predictions on a table.

Figure 10 – Final output of CNN model



Attribute	Explanation
Color	Color: The image 1 water tower is blue, image 2 is gray.
Shape	Shape: Image 1 water tower has a rounded bottom, image 2 is flat.
Texture	Texture: Image 1 water tower is shinier than image 2.

Note. Screenshot of the output from the CNN model.

4. Performance Evaluation

With the following ratios: 80% for training, 20% for validation, and the test set utilized after training, the dataset was divided into training, validation, and test sets to guarantee appropriate model evaluation and generalization. This 80/20 split is a popular machine-learning technique that provides enough data for training while reserving a sizeable amount for validation to track overfitting and modify hyperparameters. To guarantee balanced learning, methods like class weights or oversampling/under-sampling can be used. The dataset should also be examined for class imbalance.

Accuracy, categorical cross-entropy loss, confusion matrix, precision, recall, F1-score, and the ROC curve with AUC were among the metrics used to assess the model's performance. To visualize training and validation accuracy across epochs and identify underfitting or overfitting, the learning curves were shown. To assess performance by class, a confusion matrix was also created, which displays the frequency of correctly and incorrectly identified classes.

After training, the model's performance can be evaluated further by uploading predictions for a competitive evaluation on websites such as Kaggle or EvalAI. If several models are used, they can be fairly compared using the same test and validation sets. The models can then be ranked according to their metrics, such as accuracy, loss, precision, recall, F1-score, and AUC, to determine which strategy works best.

5. Analysis of Model Performance and Recommendations

With training accuracy ranging between 0.30 and 0.35 and validation accuracy remaining constant at 0.29 throughout all epochs, the learning curve presented in the report shows that both training and validation accuracy are continuously low. This conduct implies that the model is underfitting, which means it is not successfully capturing the underlying patterns in the data. The model's poor generalization to unknown data is further supported by the stagnated validation accuracy and marginally increased validation loss. Inadequate model complexity, poor feature extraction, inappropriate hyperparameter tuning, or problems with the dataset itself—such as class imbalance, noisy labels, or a lack of training samples—can all be blamed for this poor performance.

To address these challenges and improve the model's performance, the following steps are recommended:

1. **Increase Model Complexity**: Experiment with deeper or more sophisticated architectures, such as ResNet or EfficientNet, to enhance the model's ability to learn complex patterns and features from the data.
2. **Hyperparameter Optimization**: Conduct a systematic search for optimal hyperparameters, including learning rates, batch sizes, and regularization techniques (e.g., dropout, weight decay), to improve training in dynamics and model convergence.
3. **Data Augmentation**: Apply data augmentation techniques, such as rotation, flipping, cropping, and scaling, to increase the diversity and robustness of the training dataset, thereby helping the model generalize better.
4. **Class Imbalance Handling**: If the dataset exhibits class imbalance, employ techniques such as oversampling, undersampling, or class-weighted loss functions to ensure that the model does not bias toward majority classes.
5. **Error Analysis**: Perform a detailed analysis of misclassified samples to identify potential patterns or labeling errors in the dataset. This can help improve data quality and guide further preprocessing steps.

6. Conclusion

The CNN model created for this study, which focuses on categories like color, shape, and texture, shows promise in classifying small variations between image pairs. Three convolutional layers, max-pooling layers, flattening layers, and dense layers with dropout for regularization make up the model architecture. The model's performance indicates potential for improvement even after ten epochs of training with the Adam optimizer with categorical cross-entropy loss. While the validation accuracy stayed steady at about 29%, the training accuracy varied between 34 and 37%, indicating difficulties with generalization. These outcomes could be caused by the task's difficulty or dataset constraints, which should be rectified for improved model performance.

The model's capacity to make predictions was demonstrated when it correctly identified the difference between two test photos as "color," however more precision is needed. For future usage and additional training or inference, the model has been stored in HDF5 format. Using data augmentation approaches, adding more layers or units to the model, and experimenting with hyperparameter tuning are some suggestions for enhancing the model's performance. Furthermore, the model's capacity to generalize may be improved by utilizing transfer learning or a bigger labeled dataset. Overfitting may also be lessened by regularization techniques like L2 regularization or extra dropout layers.

The CNN model exhibits promise overall, but to increase its classification accuracy, it is imperative to adjust its architecture, hyperparameters, and data processing. The model can be further optimized using the suggested modifications, providing improved performance and the possibility of implementation in practical applications. The model is a useful tool for future development because it can be saved and reloaded, enabling ongoing experimentation.

7. Reference

Hargurjeet. (2022, January 6). 7 Best Techniques to Improve the accuracy of CNN W/O Overfitting. *Medium*. https://gurjeet333.medium.com/7-best-techniques-to-improve-the-accuracy-of-cnn-w-o-overfitting-6db06467182f