Notebook Explanation: Detection-of-Manipulated-and-Authentic-Images

This document provides a detailed explanation of the Jupyter Notebook `detection-of-manipulated-and-authentic-images.ipynb`. The notebook focuses on building and training a Convolutional Neural Network (CNN) to detect manipulated (fake) versus authentic (real) images using TensorFlow and Keras.

1. Overview

The project aims to perform binary classification (Real vs. Fake) on image forensics datasets. It utilizes four distinct datasets, running experiments on them individually and then combining them into a single large dataset for a comprehensive model training session.

2. Libraries and Setup

The notebook uses standard Python libraries for data science and deep learning:

- **Core:** `os`, `numpy`, `pandas`
- **Visualization:** `matplotlib.pyplot`, `seaborn`
- **Deep Learning:** `tensorflow` (Keras API)
- **Metrics:** `sklearn.metrics` (classification_report, confusion_matrix)

3. Global Configuration

Key hyperparameters defined at the start:

- **Image Size:** `(256, 256)` - Input resolution for the model.
- **Batch Size:** `32` - Number of samples per gradient update.
- **Epochs:** `20` - Maximum iterations over the dataset (subject to early stopping).
- **Autotune:** `tf.data.AUTOTUNE` - For dynamic performance optimization.

4. Data Loading Pipeline

The notebook implements two data loading strategies: **Individual** and **Combined**.

Single Dataset Loading (`load_datasets`)

- Uses `tf.keras.utils.image_dataset_from_directory`.
- **Structure:** Expects `train`, `validation`, and `test` subdirectories.
- **Labeling:** `labels='inferred'` (uses folder names) and `label_mode='binary'`.
- **Optimization:**
  - `cache()`: Keeps data in memory to speed up training.
  - `prefetch(buffer_size=AUTOTUNE)`: Prepares the next batch while the GPU works on the current one.

Combined Dataset Loading (`load_combined_datasets`)

- Iterates through a list of dataset paths.

- Loads each split (train/val/test) independently.
- Uses `ds.concatenate(new_ds)` to merge datasets.
- **Memory Management:** Explicitly **removes** `.cache()` for the combined dataset to prevent RAM overflow (`OOM`). It only uses `.prefetch()`.
- **Shuffling:** Reduces shuffle buffer size to 50 to manage memory load.

5. Model Architecture (`build_model`)

The model is a custom Convolutional Neural Network (CNN).

1. **Input Layer:** Shape `(256, 256, 3)`.
2. **Data Augmentation & Preprocessing:**
   - `RandomFlip("horizontal")`
   - `RandomRotation(0.1)`
   - `RandomZoom(0.1)`
   - `Rescaling(1./255)`: Normalizes pixel values to [0, 1].
3. **Convolutional Blocks (4 Blocks):**
   - Each block consists of:
     - `Conv2D`: Extract features (Filters: 32 -> 64 -> 128 -> 256).
     - `BatchNormalization`: Stabilizes learning.
     - `MaxPooling2D`: Reduces spatial dimensions.
     - `Dropout`: Regularization (Rates: 0.2 -> 0.2 -> 0.3 -> 0.4).
4. **Classifier Head:**
   - `Flatten`: Converts 2D feature maps to 1D vector.
   - `Dense(512, activation='relu')`: Fully connected layer.
   - `BatchNormalization`
   - `Dropout(0.5)`: Aggressive regularization.
   - `Dense(1, activation='sigmoid')`: Output layer for binary classification (probability 0-1).

Compilation:

- **Optimizer:** `adam`
- **Loss:** `binary_crossentropy`
- **Metrics:** `accuracy`

6. Training Strategy (`train_model`)

The training process includes callbacks to prevent overfitting and optimize convergence:

- **EarlyStopping:** Stops training if `val_loss` doesn't improve for 5 epochs. Restores best weights.
- **ReduceLROnPlateau:** Reduces learning rate by a factor of 0.2 if `val_loss` plateaus for 3 epochs.

7. Evaluation & Visualization

Plotting (`plot_history`)

Generates side-by-side plots for:

- Training vs. Validation Accuracy
- Training vs. Validation Loss

Evaluation (`evaluate_model`)

1. **Basic Metrics:** Prints final Test Loss and Test Accuracy.
2. **Classification Report:** Precision, Recall, F1-Score for both classes.
3. **Confusion Matrix:** Heatmap visualization of True Positives, False Positives, True Negatives, and False Negatives.

8. Execution Flow

The notebook runs the experiment pipeline in 5 distinct stages:

1. **Dataset 1:** Load -> Train -> Evaluate.
2. **Dataset 2:** Load -> Train -> Evaluate.
3. **Dataset 3:** Load -> Train -> Evaluate.
4. **Dataset 4:** Load -> Train -> Evaluate.
5. **Combined Experiment:**
   - Merges all 4 datasets.
   - Trains a new model on the massive combined dataset.
   - Evaluates performance to see if more data improves robustness.

9. Code Summary

The core logic is encapsulated in the `run_experiment` function, which abstracts the complexity:

```python
def run_experiment(data_path, dataset_name="Dataset"):
    # 1. Load Data
    train_ds, val_ds, test_ds = load_datasets(data_path)

    # 2. Build Model
    model = build_model(...)

    # 3. Train
    history = train_model(model, train_ds, val_ds)

    # 4. Visualize
    plot_history(history, dataset_name)

    # 5. Evaluate
    evaluate_model(model, test_ds)

    return model, history
```

This modular design allows for easy scalability and testing of multiple datasets.