Wasif Waheed

REPORT: ANALYSIS OF CNN_ON_CAT_VS_DOG_DATASET.IPYNB

Prepared: August 29, 2025

---

1. Executive Summary

This notebook is a hands- on educational assignment that guides a student to build a convolutional neural network (CNN) to classify images of cats vs dogs. The notebook contains clear structure and instructions: environment setup, data preparation using `ImageDataGenerator`, model building (for a CNN), compilation and training, and evaluation (plots, confusion matrix). However, many cells are left as `TODO` placeholders (underscores and hints), so the notebook appears to be a template rather than a fully executed experiment.

This report documents what the notebook intends to do, what is present, what is missing, and provides concrete recommendations and an actionable checklist to finish the experiment and obtain reproducible results.

---

2. Objective & Scope

Objective: Implement, train, and evaluate a CNN that classifies images into two classes: `cat` and `dog`.

Scope (as implied by the notebook): - Use `ImageDataGenerator` for preprocessing and augmentation. - Build a sequential CNN with convolutional, pooling, dropout, and dense layers. - Compile the model with an appropriate optimizer and loss for binary classification. - Train the model with callbacks (EarlyStopping, ModelCheckpoint) and visualize learning curves (loss/accuracy) and a confusion matrix on a test/validation split.

---

3. Dataset and preprocessing (notebook contents)

Dataset: The notebook refers to a Cat vs Dog dataset arranged into directories (train/validation/test) and uses Keras image utilities to load it via `ImageDataGenerator.flow_from_directory(...)`.

Preprocessing / augmentation (intended): - Rescale pixel values from `[0,255]` to `[0,1]` (using `rescale=1./255`). - Typical augmentations the notebook suggests (but leaves for TODO) include: `rotation_range`, `width_shift_range`, `height_shift_range`, `shear_range`, `zoom_range`, `horizontal_flip`. - Target image size, batch size and class

mode are called out but left blank in the template; these must be filled in (common choices shown in Recommendations).

Data checks present: The notebook includes cells to display sample images and verify class labels, which is good practice.

---

4. Model architecture (notebook state)

The notebook contains a placeholder for building the CNN and triggers `model.summary()`. The actual architecture is not fully implemented in the provided file — it contains `TODO` markers. Typical, recommended architectures for this dataset (and what you could complete in those cells) are shown below.

Recommended model (example to fill in the TODO):

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

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

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

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

This is a compact architecture which is fast to train on small GPUs and often good enough for binary cat/dog classification from moderate-resolution images.

---

5. Compilation, loss & metrics

The notebook leaves placeholders for `compile()` arguments. For binary cat vs dog classification use: - **Loss:** `binary_crossentropy` - **Optimizer:** `adam` (or `RMSprop(lr=1e-4)` for more stable training) - **Metrics:** `accuracy` (and optionally `AUC`)

Example:

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

---

6. Training setup & callbacks

The notebook indicates (but has placeholders for) key training settings and callbacks. It includes references to `EarlyStopping` and `ModelCheckpoint`.

Recommended settings:

- **Image size:** 150×150 (or 224×224 if using transfer learning)
- **Batch size:** 32
- **Epochs:** 20–30 (use EarlyStopping to avoid over- fitting)
- **Callbacks:**
    - `EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=True)`
    - `ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)`

Example `fit` call:

```
history = model.fit(
    train_gen,
    validation_data=val_gen,
    epochs=25,
    callbacks=[early_stop, checkpoint]
)
```

The notebook references plotting `history.history['loss']`, `history.history['val_loss']`, and corresponding accuracy curves — these should be executed after training.

---

7. Evaluation & Visualization

The notebook includes code (some placeholders) to: - Plot training/validation loss and accuracy curves. - Compute a confusion matrix using `sklearn.metrics.confusion_matrix` on test labels vs predictions. - Show sample test images with predicted vs actual labels.

These are appropriate evaluation steps. Since the notebook was a template, no `history` or trained weights were included; therefore the report cannot show numeric results. After training, the notebook's evaluation cells should be run to produce: - Final accuracy on validation/test sets - Loss curves showing under- or over- fitting - Confusion matrix with counts of true/false positives/negatives.

---

8. Findings from the provided notebook file

- The notebook is well structured and pedagogically sound: it walks the user through environment setup, data creation, visual verification, model building, compilation, training with callbacks, and evaluation.
- **But it is not executed:** many critical cells contain placeholders (`____`, `# TODO`) and most code cells were not filled in or run. The notebook contains about **21 markdown cells** and **25 code cells**, many of which require the student to provide concrete values and code.
- There are skeletons for: imports, data generators, model definition, callbacks, training, plotting, and confusion matrix, but **no finished experiment outputs** (no `history` plots, no saved model file, no evaluation metrics present in the file).

---

9. Recommendations & next steps (actionable checklist)

1. **Fill in data generator args**
   - Choose `target_size=(150,150)`, `batch_size=32`, `class_mode='binary'` (or `categorical` if using one-hot labels). Use `rescale=1./255`. Add augmentation parameters for the training generator.
2. **Complete model architecture**
   - Implement the CNN (example provided above) or use a pretrained backbone (MobileNetV2, EfficientNetB0) for transfer learning if higher accuracy is required.
3. **Compile the model**
   - Use `binary_crossentropy` + `Adam`.
4. **Train with callbacks**
   - Add `EarlyStopping` and `ModelCheckpoint`. Set `epochs=25` and rely on EarlyStopping.
5. **Run training and plot learning curves**
   - After training, plot `loss/val_loss` and `accuracy/val_accuracy` to check for overfitting/underfitting.
6. **Evaluate on test set**
   - Use `model.evaluate()` and compute confusion matrix + classification report (`precision`, `recall`, `f1-score`).
7. **Optional improvements**
   - Use data augmentation more aggressively if overfitting.
   - Try transfer learning with fine- tuning for better accuracy with fewer epochs.
   - Experiment with image size (224×224) and different optimizers/learning rates.
8. **Reproducibility**
   - Set random seeds (`numpy`, `tf`, `random`) and document environment (TF version, python version). Save the final best model and training `history` as a pickle or JSON for reporting.

---

10. Suggested completed notebook structure (what to add)

1. Title & objective (already present)
2. Environment setup + package versions (add `!pip freeze` or print versions)
3. Data directory overview (counts per class)
4. Data generators: training (with augmentation), validation, test
5. Visualize sample images (already present)
6. Build model (concrete code cell)
7. Compile model
8. Define callbacks
9. Train model (`model.fit`) and save `history`
10. Plot curves and save images
11. Evaluate on test data (confusion matrix & classification report)
12. Discussion of results, limitations, and improvements

---

11. Appendix — mapping notebook placeholders to concrete values

- `target_size=(150,150)`
- `batch_size=32`
- `class_mode='binary'`
- `optimizer='adam'`
- `loss='binary_crossentropy'`
- `metrics=['accuracy']`
- `epochs=25`
- `callbacks=[EarlyStopping(...), ModelCheckpoint(...)]`

---

12. Conclusion

The notebook is an excellent template for learning how to build a CNN for a binary image classification task. It contains the right educational steps but remains incomplete. To produce a fully reproducible experiment and numeric results, the `TODO` cells need to be filled with concrete parameters and code, training must be executed, and evaluation cells run. The recommended model and parameter choices in this report provide a safe, practical starting point to finish the assignment and obtain interpretable results.
