




Below is a side-by-side comparison of your “local” script vs. the Colab/Drive script, followed by my recommendation.






Aspect	My Script	Your - GoogleDrive Script
Environment setup	Pure Python/TensorFlow, no external mounts.	Mounts Google Drive, mixed <code>df</code> vs. <code>data</code> naming.
Data loading	Single, clean <code>pd.read_csv(csv_path)</code> at top.	Reads CSV three times in different places; uses both <code>df</code> and <code>data</code> .
Label encoding	Encoded once, stored in <code>df</code> , clear naming.	Encoded twice (two separate <code>LabelEncoder()</code> blocks), redundant.
Train/Val/Test split	Simple 60/20/20 split (no stratify) or stratified option.	Stratified on brand, then random split—removes small classes.
tf.data pipeline	Fully modular <code>create_dataset()</code> with clear one-hots.	Similar pipeline, but computes depths via <code>train_df.nunique()</code> each time.
Model architecture	MobileNetV2 base + two 128-unit branches + dropout.	MobileNetV2 base + two 128-unit branches + dropout (almost identical).
Training schedule	20 epochs initial + 10 fine-tune	100 epochs initial + 10 fine-tune
Metrics & evaluation	Jaccard, Dice, confusion matrices, smart <code>classification_report</code> fix.	Jaccard & Dice only (no per-class report fix), no confusion matrices.
Visualization	History plots + sample predictions grid.	History plots + full-dataset grid + seaborn import (unused).
Code cleanliness	Single flow, minimal repetition.	Several copy-paste blocks, inconsistent variable names.

Pros & Cons

Your Local Script

-  **Clean & DRY**: Everything lives in one place, no repeated CSV loads.
-  **Robust evaluation**: Handles missing classes in reports, shows confusion matrices.
-  **Easily reproducible**: No external dependencies beyond your local filesystem.

Colab/Drive Script

-  **Google Drive integration**: Handy if your data lives in Drive.
-  **Longer training**: Might converge more if you actually have enough data.
-  **Redundancy**: Multiple `pd.read_csv`, double encoding, unused imports (`seaborn`).
-  **Variable confusion**: Switches between `df`, `data`, and `data.sample` vs. `df.sample`.
-  **Evaluation gaps**: No per-class reports or confusion matrices to diagnose issues.

Recommendation

I'd lean **strongly** toward your **local script** as the base—it's more maintainable, easier to debug, and already has the fixes for classification reporting. If you need to run in Colab:

1. **Port the Drive mount** into your local script (just once, at the top).
2. **Remove all redundant CSV reads** and `LabelEncoder` calls—do it exactly once.
3. **Keep the smart `classification_report`** with `labels=` & `target_names=` so you never hit that mismatch.
4. Optionally **increase epochs** (e.g. to 100) but monitor over-fitting with early stopping.

That way you get the best of both worlds: a clean, battle-tested pipeline plus Drive integration.

Problems With My code:

Your model is dramatically over-fitting and struggling with class imbalance, which shows up in every metric:

1. Train vs. Validation Performance

- **Final training accuracy** was around **66 % (brand)** and **79 % (task)**, but your **validation accuracy** sits at only **9 % (brand)** and **18 % (task)**.
 - That gap tells you the network memorized the training set but failed to generalize to unseen data.
-

2. Jaccard (Intersection-over-Union) & Dice (F1) on Test Set

Metric	Brand	Task
Jaccard (IoU)	0.0476	0.1061
Dice (F1-score)	0.0714	0.1333

- **Both are near zero**—your predicted and true label sets barely overlap on average.
 - A Dice/F1 of 0.07 for brand means your harmonic mean of precision & recall across brands is only 7 %.
-

3. Confusion Matrices

Brand

[[0 1 1 1 0 0 0]

```
[0 0 0 1 0 0 1]
[0 0 0 0 0 0 0]
[0 2 0 0 0 0 0]
[0 1 0 0 0 0 0]
[0 1 0 0 0 0 0]
[0 1 0 0 0 0 1]]
```

- Many rows (true classes) have **all zeros**—the model never predicted those brands.
- It's lopsided toward a handful of brands (e.g. the last row “V. Mueller” gets some predictions).

Task

```
[[0 1 0 0 1 ...]
 [0 1 0 0 0 ...]
 [ ...      ]
 ...
 ]
```

- Same story: most task-types are never predicted, a couple get all the mass.

4. Classification Reports

You can see from the per-class precision/recall:

- **Support** (number of true samples) is often 0 or 1–2.
- Almost every class has **0 precision** and **0 recall** except for the few classes with slightly more samples (e.g. “V. Mueller” or “Dissection (Tissue)”).

5. What’s Going Wrong?

1. **Severe class imbalance**

- Some brands/tasks have only 1–2 examples. The network can't learn those.

2. Small dataset + many classes

- With 11 brands and 11 task types but only ~50 images, the per-class data is tiny.

3. Random split

- Your 60/20/20 split may have left some classes completely out of val/test (or train), making evaluation and learning impossible.

4. Over-fitting

- A frozen MobileNetV2 + small Dense head is too flexible on a tiny dataset, so it memorizes.

6. Recommendations to Improve

1. Gather more data

- You need dozens (ideally hundreds) of images per brand and task.

2. Stratified splitting

- Use `stratify=` in `train_test_split` on both brand and task (or do a multi-label stratification) to ensure every class appears in each split.

3. Class weights or focal loss

- In `model.compile(...)`, pass `class_weight=` for brands/tasks to up-weight rare classes, or switch to a focal-loss variant.

4. Reduce number of classes

- Merge very rare brands/tasks into an “Other” category until you have enough examples.

5. Stronger regularization & augmentation

- Increase **Dropout**, add random rotations/zoom, or use `tf.image` augmentations to artificially enlarge your dataset.

6. Fine-tune more of the backbone

- Unfreeze more layers of MobileNetV2 (or use a larger model) once you have more data.

7. Evaluate with cross-validation

- With very little data, a 5-fold CV can give you a more reliable estimate than one random split.

Bottom Line

Right now, your model is memorizing the training images but cannot generalize to new ones. The extremely low IoU and F1 scores confirm that almost none of your predictions match the true labels. To fix this, you'll need more balanced data, smarter splitting, and techniques to handle the imbalance and over-fitting.