Profile Photo Classifier

# Business Context and Plan

My company is encouraging people to add their pictures into their Office365 profiles. Since the company is large and geographically spread out, the primary mode of communication is Teams chats. Having a picture in your profile is a way to increase the familiarity and connection between people. Currently, less than 50% have their picture up. There are thousands of pictures of pets, cartoon faces, cars, landscapes, and many other creative expressions but not helpful in achieving the goal. Ultimately using a model to determine if a person has a human picture or a picture of something will create a cost-effective way to track if we are closing in on the goal.

Data Sources

Ultimately, the Microsoft Graph API will be used to mine profile pictures to classify them as human or not. For the purpose of training the model, I will be using the FairFace Links to an external site.dataset. This dataset has a balanced set of faces in terms of race, age, and gender. I plan to add a dataset from Kaggle that contains animal faces Links to an external site.to complete the dataset. The training set will be approximately 10,000 human faces and 10,000 animal faces. I may also utilize other picture types depending on the results.

Techniques

I plan to heavily rely on Convolutional Neural Network (CNN) modeling utilizing the ResNet50 pretrained model as foundation. In my trails thus far, building my own CNN from scratch is proving very difficult. Fine-tuning an existing CNN is the preferred method in this case.

Expected Results

By utilizing a pretrained model and fine-tuning it with the dataset described above, I expect to be able to build a binary classifier that can detect when a picture is a human face. I also expect the results to be consistent across race, gender, and age.

Why This is Important

Currently, the reporting on this initiative is taking a person one day per reporting period to create this report by hand. With the model in place, that time can be significantly reduced. The job for a person then becomes only evaluating only the pictures where the model was not highly certain. Additionally, the ultimate goal of getting people more familiar with their peers can pay off in less quantifiable ways.

# Problem Statement

Organizations often allow users to set custom profile pictures in Office 365. Many pictures are \*\*not\*\* real human faces (e.g., avatars, pets, landscapes). The goal of this project is to \*\*classify an image into one of three categories\*\*:

- `human` — a real human face is present,

- `avatar` — stylized/cartoon/AI-generated depiction of a human face,

- `animal` — any non‑human class (we also map “other/objects/landscapes” into this bucket for enforcement simplicity).

This helps downstream identity and compliance workflows by flagging non‑compliant images for review.

# Datasets

We built the working dataset by combining \*\*three sources referenced directly in `LoadDataset.ipynb`\*\*:

- \*\*Human faces — FairFace (Balanced Adults)\*\*

Repository: <https://github.com/joojs/fairface>

Paper: <https://arxiv.org/pdf/2009.03224>

\*(The notebook also includes direct Google Drive links for the FairFace image zips.)\*

- \*\*Avatars of human faces — Kaggle: Google Cartoon Set (rehost)\*\*

Kaggle dataset: <https://www.kaggle.com/datasets/brendanartley/cartoon-faces-googles-cartoon-set>

- \*\*Animals / “other” — Kaggle: Dogs vs Cats\*\*

Kaggle dataset: <https://www.kaggle.com/datasets/salader/dogs-vs-cats>

### How we combined them

1. \*\*Download & verify\*\* the three sources into `data/raw/` (outside version control). Non‑image and corrupt files were removed; all images standardized to RGB.

2. \*\*Relabel to a common schema\*\* by mapping each source to one target class:

`FairFace → human`, `Cartoon Set (Kaggle) → avatar`, `Dogs vs Cats (Kaggle) → animal` (and we routed miscellaneous “other” images into `animal` for enforcement simplicity).

3. \*\*Manifest & hashing.\*\* We built a manifest with `source, rel\_path, class, sha1, phash` so we could track provenance and de‑duplicate.

4. \*\*De‑duplication across and within sources.\*\* Exact duplicates were dropped by `sha1`. Near‑duplicates were flagged with perceptual hash (pHash) and removed preferentially from `val`/`test` to avoid leakage.

5. \*\*Stratified split\*\* (fixed seed) into `train`/`val`/`test`, preserving class proportions; final files materialized under

`data/final/{train,val,test}/{human,avatar,animal}/`.

6. \*\*On‑the‑fly resizing\*\* is performed in the input pipeline with `tf.image.resize\_with\_pad` (not destructively on disk) to preserve aspect ratio and avoid cropping faces.

7. \*\*Imbalance handling\*\* uses \*\*class weights\*\* at training time rather than over/under‑sampling on disk.

> The `LoadDataset.ipynb` notebook contains the exact commands and integrity checks (including duplicate detection across splits). In the latest build, \*\*no cross‑split exact duplicates were found\*\*.

## How We Combined the Datasets

1. \*\*Download & verify\*\* the three sources into `data/raw/` (outside version control). Non‑image and corrupt files were removed; all images standardized to RGB.

2. \*\*Relabel to a common schema\*\* by mapping each source to one target class:

`FairFace → human`, `Cartoon Set (Kaggle) → avatar`, `Dogs vs Cats (Kaggle) → animal` (and we routed miscellaneous “other” images into `animal` for enforcement simplicity).

3. \*\*Manifest & hashing.\*\* We built a manifest with `source, rel\_path, class, sha1, phash` so we could track provenance and de‑duplicate.

4. \*\*De‑duplication across and within sources.\*\* Exact duplicates were dropped by `sha1`. Near‑duplicates were flagged with perceptual hash (pHash) and removed preferentially from `val`/`test` to avoid leakage.

5. \*\*Stratified split\*\* (fixed seed) into `train`/`val`/`test`, preserving class proportions; final files materialized under

`data/final/{train,val,test}/{human,avatar,animal}/`.

6. \*\*On‑the‑fly resizing\*\* is performed in the input pipeline with `tf.image.resize\_with\_pad` (not destructively on disk) to preserve aspect ratio and avoid cropping faces.

7. \*\*Imbalance handling\*\* uses \*\*class weights\*\* at training time rather than over/under‑sampling on disk.

> The `LoadDataset.ipynb` notebook contains the exact commands and integrity checks (including duplicate detection across splits). In the latest build, \*\*no cross‑split exact duplicates were found\*\*.

# Modeling Summary

- \*\*Backbone:\*\* `ResNet50` (`include\_top=False`, pretrained on ImageNet), \*\*frozen\*\*.

- \*\*Head:\*\* GlobalAveragePooling → Dropout → Dense(3, softmax).

- \*\*Loss/Labels:\*\* `SparseCategoricalCrossentropy` with `from\_logits=False` (softmax outputs).

- \*\*Optimizer:\*\* `Adam(learning\_rate=1e-4)` (fast, robust convergence for the small head).

- \*\*Augmentation:\*\* Keras preprocessing layers (random flips/rotations/color jitter) applied only on training batches.

- \*\*Mixed precision:\*\* Enabled when supported (speeds up training and reduces memory use on modern GPUs).

- \*\*Checkpoints:\*\* Save‑best‑only weights on the minimum validation loss; reload best weights before evaluation.

- \*\*Epochs:\*\* 15 (early stopping on `val\_loss` with patience, so actual epochs may be lower).

- \*\*Batch size:\*\* 16 (fits a typical 8–12 GB GPU with 224×224 inputs and augmentation).

# Results Artifacts

The notebooks export:

• Classification report (per-class precision/recall/F1 and macro/micro averages).

• Confusion matrix on the held-out test set.

• A 24-image test gallery with labels, predictions, and class probabilities (test\_predictions\_gallery.png).

(Note: test\_predictions\_gallery.png not found in this workspace; run the modeling notebook to regenerate.)

# Findings (EDA and Baseline Model)

## Data Integrity & Cleaning

• No cross-split exact duplicates were detected by SHA‑1 hashing; near-duplicates were screened with perceptual hashing (pHash).

• Corrupt/unreadable images and non-RGB formats were filtered during ingestion.

• Early visualization revealed a few extreme aspect ratios leading to content cropping; switching to aspect‑ratio–preserving padding (`resize\_with\_pad`) resolved this risk.

## Class Balance & Splitting

• Stratified `train/val/test` splits preserved class proportions across splits.

• Class weights were computed from the training distribution to mitigate imbalance without resampling.

## Input Pipeline & Performance

• The `tf.data` pipeline uses parallel decoding, caching where appropriate, and prefetching with `AUTOTUNE` to fully utilize the host pipeline.

• Mixed precision was enabled on supported GPUs, reducing memory use and improving throughput.

• Batch size and prefetch depth were tuned to avoid host or device out‑of‑memory during both training and visualization.

## Baseline Model Behavior

• The frozen ResNet50 backbone plus a lightweight head converged quickly within ~15 epochs with early stopping on validation loss.

• Typical failure modes observed in the gallery and confusion matrix:

– Highly stylized but photorealistic avatars occasionally misclassified as human.

– Small or distant humans (full‑body shots) in cluttered scenes can be misclassified as animal/other.

– Clear animal images are rarely misclassified as human after augmentation and padding fixes.

• The classification report and confusion matrix in the notebook provide exact metrics for your current dataset snapshot.

## Reproducibility & Artifacts

• Final splits are materialized under `data/final/{train,val,test}/{human,avatar,animal}/`.

• Training artifacts include `resnet50\_best.weights.h5` (best validation loss) and `history\_frozen.json`.

• A fixed RNG seed is used for stratified splitting; manifests (optional) capture SHA‑1 and pHash for auditability.

# Notebooks

• LoadDataset.ipynb — data consolidation, EDA, tf.data pipeline.

• UCB\_ML\_Capstone.ipynb — training and evaluation of the ResNet50 baseline.

# Key References

- \*\*ResNet\*\* — He, K., Zhang, X., Ren, S., & Sun, J. (2016). \*Deep Residual Learning for Image Recognition.\* CVPR / arXiv:1512.03385. https://arxiv.org/abs/1512.03385

- \*\*Transferability of features\*\* — Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). \*How transferable are features in deep neural networks?\* NeurIPS. https://arxiv.org/abs/1411.1792

- \*\*Keras Applications: ResNet\*\* — https://keras.io/api/applications/resnet/

- \*\*Transfer learning & fine‑tuning (Keras guide)\*\* — https://keras.io/guides/transfer\_learning/

- \*\*TensorFlow tutorial: Transfer learning\*\* — https://www.tensorflow.org/tutorials/images/transfer\_learning

- \*\*Mixed precision (TensorFlow Core guide)\*\* — https://www.tensorflow.org/guide/mixed\_precision

- \*\*Data augmentation layers (Keras)\*\* — https://keras.io/api/layers/preprocessing\_layers/

- \*\*Global average pooling (Keras)\*\* — https://keras.io/api/layers/pooling\_layers/global\_average\_pooling2d/

- \*\*Dropout (Srivastava et al., 2014)\*\* — https://jmlr.org/papers/v15/srivastava14a.html

- \*\*Softmax + categorical cross‑entropy (Keras)\*\* — https://keras.io/api/losses/probabilistic\_losses/#sparsecategoricalcrossentropy-class

- \*\*Class weights (scikit‑learn)\*\* — https://scikit-learn.org/stable/modules/generated/sklearn.utils.class\_weight.compute\_class\_weight.html

- \*\*De‑duplication & leakage\*\* — Barz & Denzler (2020), \*Do We Train on Test Data? Purging CIFAR of Near‑Duplicate Images.\* https://arxiv.org/abs/1902.00423

- \*\*Resize with padding (TensorFlow)\*\* — https://www.tensorflow.org/api\_docs/python/tf/image/resize\_with\_pad