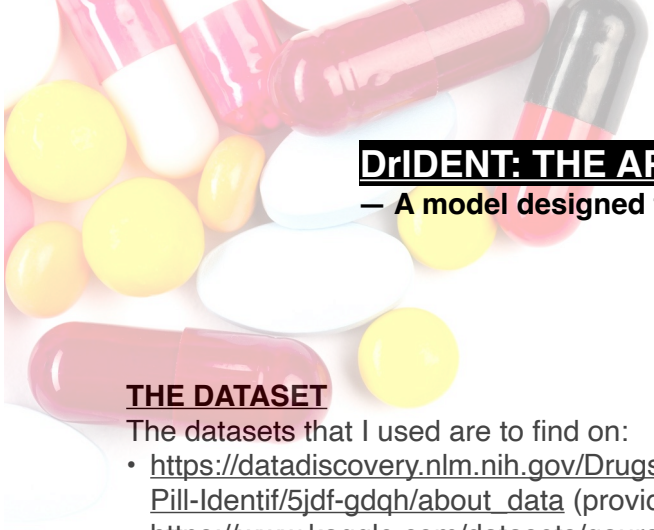

DrIDENT: THE APP FOR PILL IDENTIFICATION

— A model designed to recognize solid pharmaceuticals —

THE DATASET

The datasets that I used are to find on:

- https://datadiscovery.nlm.nih.gov/Drugs-and-Chemicals/Computational-Photography-Project-for-Pill-Identif/5jdf-gdqh/about_data (provided by the National Library of Medicine)
- https://www.kaggle.com/datasets/gauravduttakiit/pharmaceutical-drug-recognition (provided by Gaurav Dutta on Kaggle)

→ concatenated the two datasets and then took a sample subset of (50%) because the dataset was far too big and the kernel always crashed.

→ Problem: The dataset was very imbalanced, with some classes having significantly fewer samples. This can prevent the model from learning properly and cause it to focus on the most frequent classes.

→ Solution: Used class weights to inform the model of the imbalance, giving less common classes a higher weight during training.

IMAGE PREPROCESSING

- Created a function load_and_preprocess_image to:
- Load each image using OpenCV
- Resize each image to 224x224 pixels (the input size for the model).
- Normalize pixel values to the range [0, 1].
- If an image is missing or cannot be loaded, a placeholder array of zeros (an empty image) is created.
- Applied this function to all image paths in combined_data_sample, storing the processed images in a NumPy array.
- Filtering Out Placeholder Images:
- Filtered out images that were placeholders (all-zero arrays) to remove missing or problematic images.
- Filtered out their corresponding labels as well.
- Filtering Classes with Sufficient Samples:
- Counted the occurrences of each class label.
- Retained only classes with at least 20 samples to ensure sufficient data for each category.
- Refiltered the images and labels to include only valid classes.
- Re-encoding Labels:
- Re-encoded the filtered labels to have contiguous values (0, 1, 2, etc.) by creating a mapping dictionary.
- Applied this mapping to the filtered_labels array to finalize the labels for model training.

MODEL TRAINING

Model Architecture Setup:

- **Transfer Learning** with MobileNet as the base model and **pre-trained weights from ImageNet**
- Removed the top layer to adapt MobileNet for a custom classification task.
- Added new layers on top:
- Global Average Pooling: Reduces the feature map dimensions.
- Batch Normalization: Normalizes activations to speed up training.
- Dense Layer with Mish Activation: A fully connected layer with 512 units and a Mish activation function.
- Dropout Layer: Adds dropout with a rate of 0.3 to reduce overfitting.
- Output Layer: A final dense layer with softmax activation for multi-class classification based on the number of valid classes.

Freezing Base Model Layers:

- Initially froze all layers in the MobileNet base model to retain its pre-trained features and focused training on the new layers added.

Model Compilation:

Compiled the model using:

- Adam Optimizer with an initial learning rate of 10^{-4}.
- Sparse Categorical Crossentropy as the loss function (suitable for integer labels)

Data Augmentation:

- Created an ImageDataGenerator with various transformations, including, e.g. Rotation, Shifts, Zoom, Brightness Adjustment, and Horizontal Flip for better generalization

Train-Validation Split:

- Split the preprocessed images and labels into training (80%) and validation (20%) sets.
- Converted these datasets into tf.data.Dataset format, with batches of 32 and prefetching for faster training.

Class Weighting:

- Calculated class weights to handle imbalanced classes.
- Provided these weights during training to give more importance to underrepresented classes.

Learning Rate Scheduler:

- Implemented Cosine Annealing with Gradual Warm Restarts: reduces the learning rate following a cosine curve and resets periodically, helping the model avoid local minima.

Callbacks for Training:

- ReduceLROnPlateau: Reduces the learning rate if validation loss plateaus.
- EarlyStopping: Stops training if no improvement in validation loss for 12 epochs and restores the best weights.
- ModelCheckpoint: Saves the model's weights only when validation accuracy improves.
- LearningRateScheduler: Adjusts the learning rate using the custom cosine annealing function.

Initial Training:

- Trained the model for 150 epochs with frozen MobileNet layers and the defined class weights and callbacks.
- Gradual Unfreezing for Fine-Tuning:
- Gradually unfroze layers in the MobileNet base model in steps (30 and 60 layers at a time) to allow fine-tuning.
- After each unfreeze step, recompiled the model with a lower learning rate (10^{-5}) and continued training for 100 additional epochs.

MODEL EVALUATION

- Loaded the best model weights saved during training.
- Evaluated the model on the validation set to obtain the final validation accuracy.

MODEL DEPLOYMENT

- Used Streamlit to create a local browser app that allows users to upload pill images for identication
- Model and Label Mapping: Loaded a cached machine learning model and a label mapping file to convert model outputs to class names.
- Image Preprocessing: Resized and converted images to grayscale, normalized values, and ensured compatibility with model input requirements.
- Prediction and Display: Processed images with the model to predict the pill class, displaying both the predicted class and confidence scores for each potential class.
- User-Friendly Output: Showed the uploaded and preprocessed images along with detailed prediction results, making the app informative and easy to use.