TEAM 11 – VINYL RECORD AND BOOK CLASSIFIER

PROJECT 3: MARK WIREMAN

APRIL 10, 2025

PROBLEM STATEMENT

With a growing collection of books and vinyl albums it is a challenge to keep track of what I have, the details of the item, and current value.

Alas, a solution to the problem has been found! What a better way to use my newly acquired knowledge to begin a journey along the AI deep learning path to assist me in creating an inventory of my collection.

The first step is to take my idea and build an….

IMAGE CLASSIFICATION OVERVIEW

AGENTIC AI SYSTEM

DOWNLOAD IMAGES
Uses the DuckDuckGo Python library to download random images of books and vinyl records.

TRAIN THE MODEL
Designed to train and validate a deep learning model using either MobileNetV2 or ResNet50 as the base model, XLA or Adam optimization

PREPROCESS IMAGES
Perform a cleaning of the images using the TensorFlow and PIC libraries to convert or remove problem images, if required.

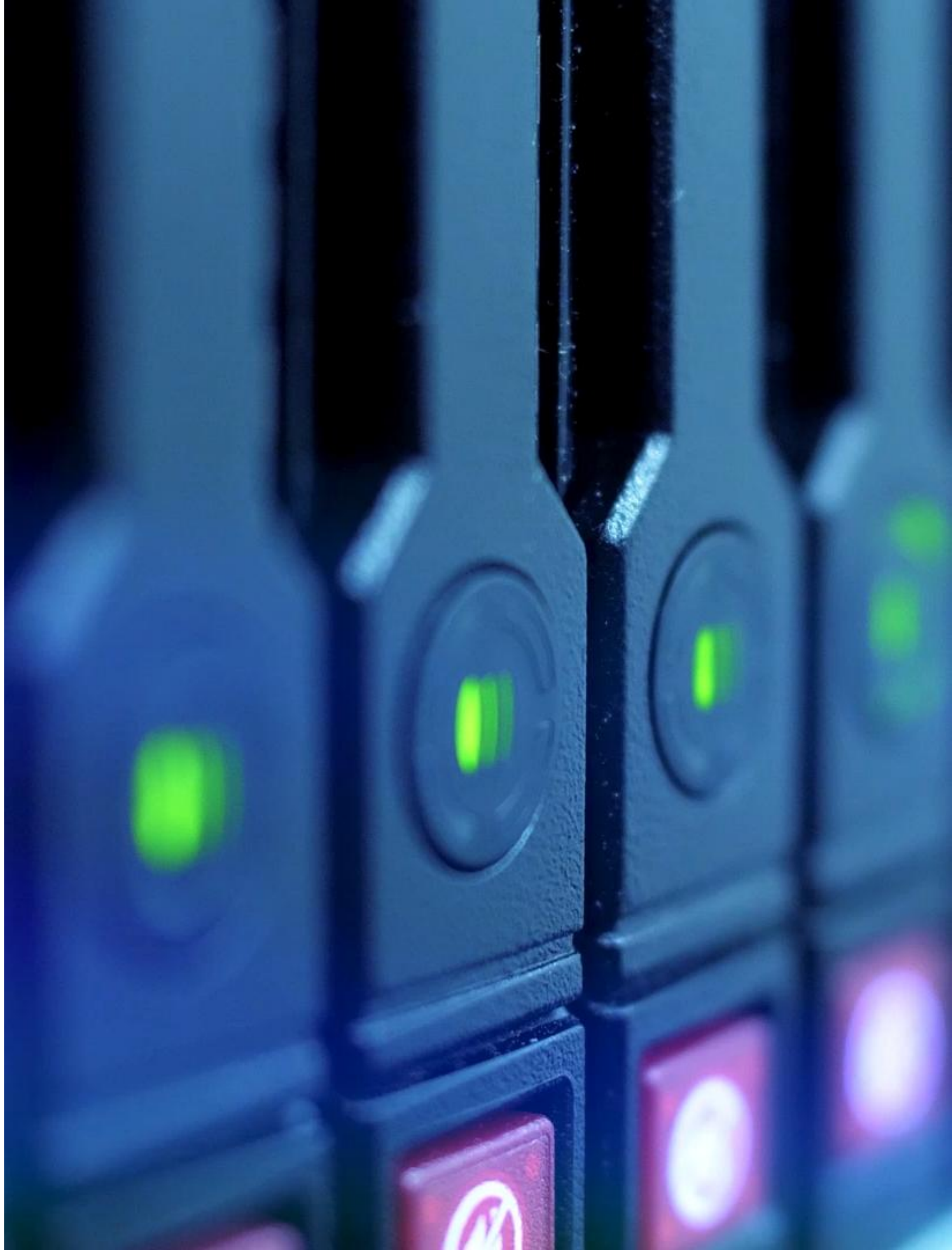
PREDICT THE CATEGORY
The model is intended to classify images into two categories: vinyl records and books.

CHOOSE THE PARAMETERS
A Flask application to allow the user to configure the agents and select what agents to run.

VIEW RESULTS
A built-in system to view progress in real-time and visualize the results.

KEY FEATURES

- **Model Selection:** The code allows users to choose between MobileNetV2 and ResNet50 as the base model.
- **Custom Preprocessing Layer:** A custom preprocessing layer is implemented to handle image preprocessing for both MobileNetV2 and ResNet50.GPU
- **Optimization:** The code is optimized for GPU usage with shared memory optimization and auto-clustering enabled.
- **XLA Compilation:** The code uses XLA (Accelerated Linear Algebra) compilation to optimize computations.
- **K-Fold Cross-Validation:** The code implements K-Fold cross-validation to evaluate the model's performance.
- **Data Augmentation:** The code applies data augmentation techniques to the training data.
- **Model Evaluation:** The code evaluates the model's performance using metrics such as accuracy, precision, and recall.

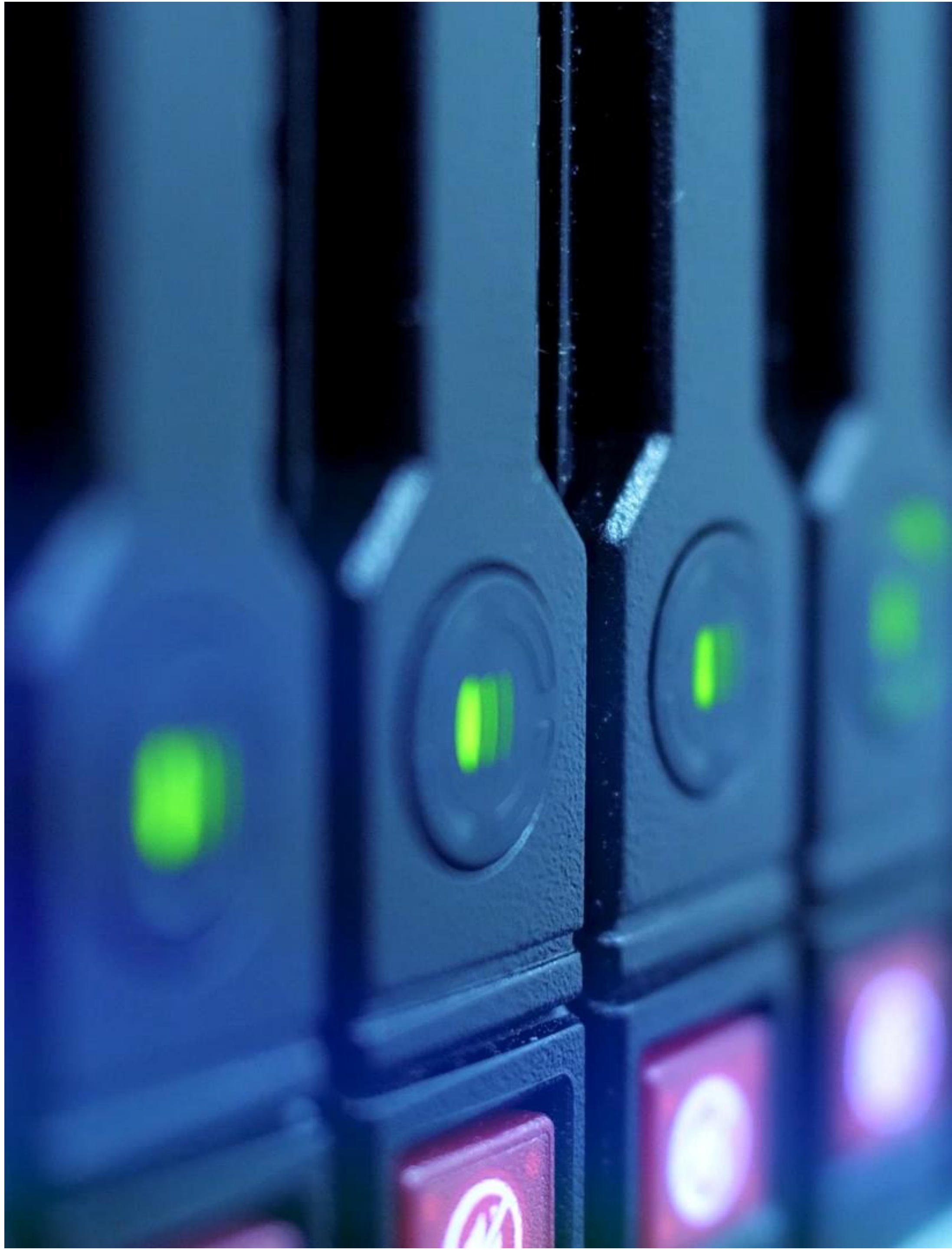


PERFORMANCE AND BENCHMARKING FEATURES

LET THE AGENTS DO THE WORK!

AUTOMATED PERFORMANCE AND BENCHMARKING

- **GPU Performance Benchmarking:** The code includes a benchmarking function to measure the performance of the GPU with different batch sizes.
- **Batch Size Optimization:** The code adjusts the batch size based on the GPU's performance to optimize shared memory utilization.
- **Speedup Calculation:** The code calculates the speedup achieved by using batch processing with different batch sizes.
- **Custom Preprocessing Layer:** A custom preprocessing layer to handle image preprocessing
- **Global Average Pooling Layer:** A global average pooling layer to reduce the spatial dimensions of the feature maps
- **Dense Layers:** One or more dense layers with ReLU activation and dropout
- **Final Classification Layer:** A final classification layer with softmax activation



AUTOMATED PERFORMANCE AND BENCHMARKING

```
Cross-Validation Results:
Average Validation Loss: 0.2789
Average Validation Accuracy: 0.9164
Best model from fold 12 with validation loss: 0.1443

Training final model on all training data...
Found 226 images belonging to 2 classes.
```

Metrics from folds used to determine “best model” to use

```python
# Create a new model for this fold
fold_model = create_model(base_model_name=model,no_of_layers=layers,use_xla=use_xla)

# Setup callbacks
fold_callbacks = []

# Add early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
fold_callbacks.append(early_stopping)

# Add learning rate scheduler
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-6,
    verbose=1
)
fold_callbacks.append(reduce_lr)
```

Configuring each fold for early stopping and learning rate

```python
# Benchmark function to measure performance improvements with shared memory
def benchmark_gpu_performance(model, img_paths, runs=5, batch_sizes=[1, 4, 8, 16, 32, 64]):
    """Benchmark GPU performance with different batch sizes to show shared memory benefits"""
    if not img_paths:
        print("No images provided for benchmarking")
        return

    # Limit to first 100 images for benchmarking
    test_paths = img_paths[:100] if len(img_paths) > 100 else img_paths

    print("\n=== GPU SHARED MEMORY PERFORMANCE BENCHMARK ===")
    print(
        f"Testing with {len(test_paths)} images,

    results = {}

    # Test individual image processing (no shared memory benefit)
    print("\nBenchmarking individual image processing (baseline)...")
    start_time = time.time()
    for _ in range(runs):
        for path in test_paths[:10]:  # Limit to 10 images for individual processing
            _ = predict_image(model, path)
    individual_time = (time.time() - start_time) / (10 * runs)
    print(
        f"Average time per image (individual): {individual_time*1000:.2f} ms")
    results['individual'] = individual_time * 1000

    # Test batch processing with different batch sizes
    for batch_size in batch_sizes:
        print(f"\nBenchmarking batch size {batch_size}...")
        start_time = time.time()
        for _ in range(runs):
            _ = predict_batch(model, test_paths, batch_size=batch_size)
        batch_time = (time.time() - start_time) / (len(test_paths) * runs)
        print(
            f"Average time per image (batch size {batch_size}): {batch_time*1000:.2f} ms")
        results[f'batch_{batch_size}'] = batch_time * 1000

        # Calculate speedup
        speedup = individual_time / batch_time
        print(f"Speedup with batch size {batch_size}: {speedup:.2f}x")
        results[f'speedup_{batch_size}'] = speedup

    # Find the best batch size
    best_batch_size = batch_sizes[0]
```

Use GPU to calculate best batch size to use

```
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 520ms/step - accuracy: 0.8950 - loss: 0.3043 - precision: 0.8950 - recall: 0.8950
Epoch 33: val_loss did not improve from 0.33897
Epoch 33: loss=0.3371, acc=0.8792
26/26 ━━━━━━━━━━━━━━━━━━━━ 15s 579ms/step - accuracy: 0.8944 - loss: 0.3055 - precision: 0.8944 - recall: 0.8944 - val_accuracy: 0.8947 - val_loss: 0.3884 - val_precision: 0.8947 - val_rec
all: 0.8947 - learning_rate: 2.5000e-05
Epoch 34/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 520ms/step - accuracy: 0.8024 - loss: 0.3996 - precision: 0.8024 - recall: 0.8024
Epoch 34: val_loss did not improve from 0.33897
Epoch 34: loss=0.3537, acc=0.8309
26/26 ━━━━━━━━━━━━━━━━━━━━ 16s 609ms/step - accuracy: 0.8035 - loss: 0.3979 - precision: 0.8035 - recall: 0.8035 - val_accuracy: 0.8421 - val_loss: 0.4027 - val_precision: 0.8421 - val_rec
all: 0.8421 - learning_rate: 2.5000e-05
Epoch 35/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 521ms/step - accuracy: 0.8872 - loss: 0.3658 - precision: 0.8872 - recall: 0.8872
Epoch 35: ReduceLROnPlateau reducing learning rate to 1.249999968422344e-05.

Epoch 35: val_loss did not improve from 0.33897
Epoch 35: loss=0.3507, acc=0.8792
26/26 ━━━━━━━━━━━━━━━━━━━━ 16s 581ms/step - accuracy: 0.8869 - loss: 0.3652 - precision: 0.8869 - recall: 0.8869 - val_accuracy: 0.8947 - val_loss: 0.3929 - val_precision: 0.8947 - val_rec
all: 0.8947 - learning_rate: 2.5000e-05
Fold 7 - Validation Loss: 0.3390, Validation Accuracy: 0.8947

Training fold 8/12
Found 207 images belonging to 2 classes.
Found 19 images belonging to 2 classes.
Created Layer 0 with units of 1024.
Created Layer 1 with units of 512.
Created Layer 2 with units of 256.
Created Layer 3 with units of 256.
Created Layer 4 with units of 1024.
Created Layer 5 with units of 512.
Created Layer 6 with units of 256.
Created Layer 7 with units of 256.
Created Layer 8 with units of 1024.
Created Layer 9 with units of 512.
Created Layer 10 with units of 256.
Created Layer 11 with units of 256.
Created Layer 12 with units of 1024.
Created Layer 13 with units of 512.
Created Layer 14 with units of 256.
resnet_xla_True
Epoch 1/100
26/26 ━━━━━━━━━━━━━━━━━━━━ 0s 564ms/step - accuracy: 0.4910 - loss: 0.6957 - precision: 0.4910 - recall: 0.4910
Epoch 1: val_loss improved from inf to 0.69340, saving model to tmp/kfold_20250408-071034/fold_8/checkpoint/best_model.keras
Epoch 1: loss=0.6942, acc=0.5072
26/26 ━━━━━━━━━━━━━━━━━━━━ 31s 841ms/step - accuracy: 0.4916 - loss: 0.6956 - precision: 0.4916 - recall: 0.4916 - val_accuracy: 0.4737 - val_loss: 0.6934 - val_precision: 0.4737 - val_rec
all: 0.4737 - learning_rate: 1.0000e-04
Epoch 2/100
```

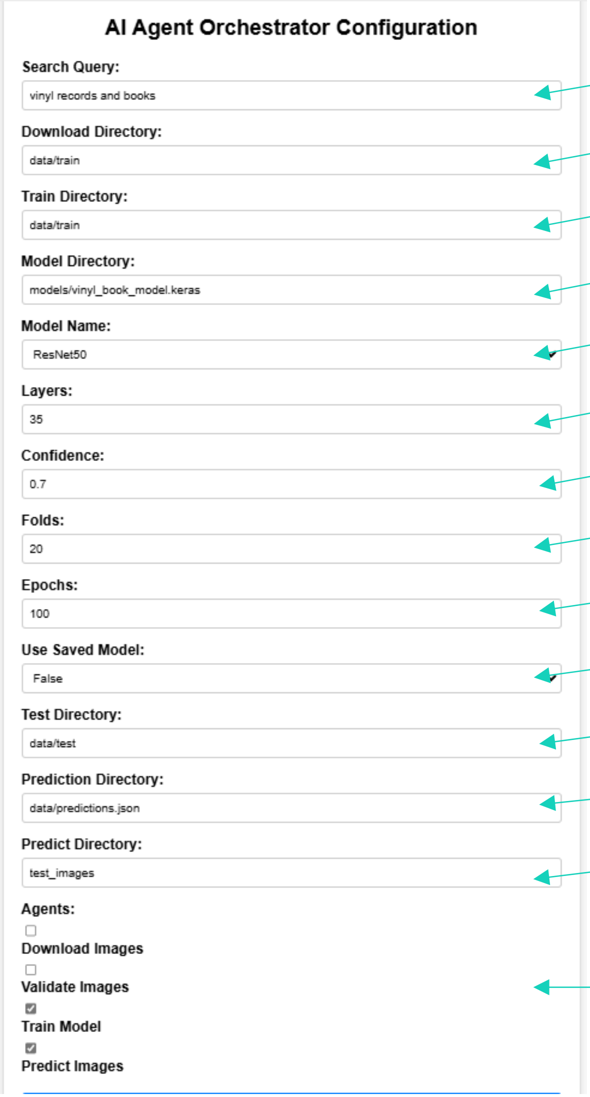
ReduceLROnPlateau callback monitors validation loss and the training uses other metrics to improve model performance and validation.

K-Fold cross-validation to evaluate the model's performance and automated assignment of units to the layers to improve the deep learning with the training set

USAGE

CONFIGURATIONS

USER INTERFACE AND CLI CONFIGURATION SETTINGS



Random images to download

Directory to save images

Directory of images to use for training

Directory location and name of trained model

Pre-trained model to use for training the custom image classification

Number of training layers per fold

Confidence to use for prediction

Number of K-folds to use

Number of epochs per layer

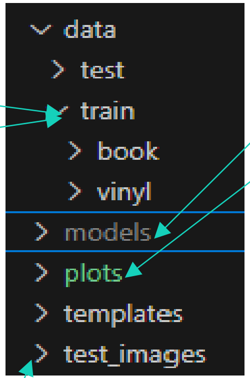
Use the saved model for prediction – otherwise a new model will be trained

If any additional testing is required, put directory here

Directory location and name of prediction results

Directory where the images to predict/classify are stored

Select Agents to run:
1. Download – uses the Download Directory
2. Validate – uses the Download Directory
3. Train – uses the Model Directory, Train Directory, Model Name, Layers, Confidence, Folds, and Epochs
4. Predict – uses the Model Directory, Prediction Directory, and Predict Directory



Created by the application

NOTE: If cloning, the code will run with the default settings

CLI: test_orchestrator.py

```python
from orchestrator import Orchestrator

orchestrator_config = {
'search_query': 'vinyl records and books',
'download_dir': 'data/train',
'train_dir':'data/train',
'model_dir':'models/vinyl_book_model.keras',
'test_dir':'data/test',
'prediction_dir':'data/predictions.json',
'model_name':'mobilenet',
'layers':15,
'download_images':False,
'validate_images':False,
'train_model':True,
'predict_images':True,
'confidence' : 0.7,
'folds':12,
'epocs':100,
'use_saved_model':False,
'predict_dir':'test_images'
}

orchestrator_obj = Orchestrator(orchestrator_config)
orchestrator_obj.orchestrate_pipeline()
```

DEMO

UI AND CLI

THE FUTURE

BACKLOG OF ENHANCEMENTS

FUTURE ENHANCEMENTS

1. Allow user to enter a learning rate
2. Allow user to enter a patience value
3. Allow user to enter the number of training images to fetch
4. Create a CSV or database inventory of the predicted images
5. Refactor the code to reduce code complexity and support dynamic classes
6. Use a GUID to align output with model runs
7. Improve the async messaging by using Kafka or Redis
8. Bug fixes

GITHUB REPOSITORY

QUESTIONS