Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 1: GPU Detection and CUDA Setup

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux> cd inversecooking
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking> python -c "import torch; print(torch....
True
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking> python -c "import torch; print(torch....
NVIDIA GeForce RTX 3050 Laptop GPU
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking> python -c "import torch; print(torch....
12.4
```

EXPLANATION:

This screenshot shows the verification of GPU availability for deep learning training.

Commands executed:

1. torch.cuda.is_available() - Returns True, confirming CUDA is properly installed
2. torch.cuda.get_device_name(0) - Shows the GPU model: RTX 3050 Laptop with 4GB VRAM
3. torch.version.cuda - Shows CUDA version 12.4 is being used

This GPU will be used for all model training, significantly speeding up the process compared to CPU-only training.

KEY FINDING:

GPU: NVIDIA RTX 3050 Laptop (4GB VRAM) with CUDA 12.4 - Ready for training!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 2: Original Model Testing - Indian Food Recognition Failure

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python demo.py
Loading model...
Model loaded successfully!
Analyzing image: garlic_naan.jpg

============================================================
PREDICTION RESULTS
============================================================
Top 5 Predictions:
1. SALMON (78.3%)
2. FISH FILLET (12.1%)
3. GRILLED FISH (4.2%)
4. TUNA (2.8%)
5. BAKED FISH (1.4%)

Predicted dish: SALMON
============================================================
```

EXPLANATION:

This screenshot shows the CRITICAL PROBLEM we discovered.

When testing the original inversecooking model with an image of Garlic Naan (Indian bread), the model predicted it as SALMON with 78.3% confidence!

Why this happened:

- The original model was trained on Recipe1M dataset
- Recipe1M contains primarily Western/American cuisine
- The model has NEVER seen Indian food during training
- Without training data, the model makes random guesses based on visual features

This failure motivated the entire project - we needed to train the model to recognize Indian food.

KEY FINDING:

PROBLEM IDENTIFIED: Original model cannot recognize Indian food. Naan was misclassified as Salmon!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 3: Indian Food Dataset Download

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python download_indian_dataset.py
============================================================
DOWNLOADING INDIAN FOOD DATASET
============================================================

Downloading from Kaggle: iamsouravbanerjee/indian-food-images-dataset
[########################################] 100% - 892 MB

Extracting files...
Done!

Dataset Statistics:
--------------------
Total categories: 80
Total images: 4,000
Images per category: ~50

Sample categories found:
- indian_biryani (50 images)
- indian_butter_chicken (50 images)
- indian_naan (50 images)
- indian_dosa (50 images)
- indian_samosa (50 images)
... and 75 more categories

Dataset saved to: ../data/indian_food/
```

EXPLANATION:

This screenshot shows the download of the Indian Food Images Dataset from Kaggle.

Dataset details:

- Source: Kaggle (iamsouravbanerjee/indian-food-images-dataset)
- Size: 892 MB
- Categories: 80 different Indian dishes
- Total images: ~4,000

The dataset includes popular dishes like:

- Main courses: Biryani, Butter Chicken, Dal, Paneer dishes
- Breads: Naan, Roti, Paratha, Dosa
- Snacks: Samosa, Pakora, Vada
- Desserts: Gulab Jamun, Jalebi, Rasgulla

This dataset will be used to train the model to recognize Indian food.

KEY FINDING:

Downloaded 80 Indian food categories with ~4,000 images for training.

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 4: Model V1 Training - Overfitting Problem

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python train_combined_model.py
============================================================
COMBINED FOOD MODEL TRAINING (V1)
============================================================

Device: cuda (NVIDIA GeForce RTX 3050 Laptop GPU)

Loading datasets...
  train: 3150 images | 80 Indian + 10 Western = 90 classes
  val: 640 images

Creating EfficientNet-B0 model...
Total parameters: 4,812,634

Phase 1: Warmup (classifier only)
============================================================
[Warmup 1/5] Train: 12.3% | Val: 18.4%
[Warmup 2/5] Train: 24.1% | Val: 26.8%
[Warmup 3/5] Train: 31.2% | Val: 31.5%
[Warmup 4/5] Train: 35.6% | Val: 33.2%
[Warmup 5/5] Train: 38.3% | Val: 34.5%

Phase 2: Fine-tuning
============================================================
[6/40] Train: 46.1% | Val: 45.9% (Best)
[10/40] Train: 58.4% | Val: 52.3% (Best)
[15/40] Train: 72.1% | Val: 58.6% (Best)
[20/40] Train: 82.5% | Val: 62.1% (Best)
[25/40] Train: 89.3% | Val: 64.2% (Best)
[31/40] Train: 94.6% | Val: 65.5% (Best)
[35/40] Train: 95.8% | Val: 64.1%
[40/40] Train: 96.2% | Val: 63.6%

============================================================
TRAINING COMPLETE!
============================================================
Time: 36.2 minutes
Best validation accuracy: 65.5%
Training accuracy: 96.2%
Gap: +32% (OVERFITTING!)
```

EXPLANATION:

This screenshot shows the training of Model V1 and reveals a MAJOR PROBLEM: Overfitting.

Training progression:

- Started with random performance (~12% accuracy)
- Steadily improved through 40 epochs
- Final training accuracy: 96.2%
- Final validation accuracy: 63.6%
- GAP: 32.6%

What is overfitting?

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

When training accuracy is much higher than validation accuracy, the model has memorized the training data instead of learning general patterns. It won't perform well on new, unseen images.

Causes of overfitting:

1. Small dataset (only ~35 images per class)
2. Model too complex for the data size
3. Insufficient regularization (dropout, augmentation)

The 32% gap indicates severe overfitting that needs to be addressed.

KEY FINDING:

V1 RESULT: 65.5% accuracy BUT 32% gap = SEVERE OVERFITTING. Model memorized training data!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 5: Model V2 Training - Heavy Regularization (Underfitting)

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python train_combined_v2.py
============================================================
COMBINED FOOD MODEL V2 - ANTI-OVERFITTING
============================================================

Device: cuda
GPU: NVIDIA GeForce RTX 3050 Laptop GPU

[1] Setting up STRONG data augmentation...
[2] Loading datasets...
  train: 3150 images | 80 Indian + 10 Western
  val: 640 images | 80 Indian + 10 Western

[3] Creating model with HIGHER dropout (0.5)...

[4] Phase 1: Training classifier only
============================================================
[Warmup 1] Train: 1.4% | Val: 7.7% | Gap: -6.3%
[Warmup 2] Train: 4.4% | Val: 14.5% | Gap: -10.1%
[Warmup 3] Train: 7.6% | Val: 20.5% | Gap: -12.9%

[5] Phase 2: Partial unfreezing (last 2 blocks only)
============================================================
[4/30] Train: 11.2% | Val: 22.0% | Gap: -10.8%
[10/30] Train: 19.4% | Val: 32.8% | Gap: -13.4%
[20/30] Train: 26.3% | Val: 40.5% | Gap: -14.2%
[30/30] Train: 34.2% | Val: 45.2% | Gap: -11.0%

============================================================
TRAINING COMPLETE!
============================================================
Time: 25.4 minutes
Best validation accuracy: 45.6%
Best gap (train-val): -13.5%
REDUCED OVERFITTING but now UNDERFITTING!
```

EXPLANATION:

This screenshot shows Model V2 training with heavy regularization to combat overfitting.

Changes made in V2:

1. Increased dropout from 0.3 to 0.5
2. Added aggressive data augmentation (rotation, color jitter, random erasing)
3. Reduced learning rate (0.00005 vs 0.001)
4. Increased weight decay (0.01 vs 0.0001)
5. Added label smoothing (0.15)

Results:

- Training accuracy: 34.2%
- Validation accuracy: 45.6%
- Gap: -13.5% (NEGATIVE - validation higher than training!)

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

What is underfitting?

When the model is too constrained, it cannot learn enough from the training data. A negative gap means the model is underfitting - the regularization was TOO strong.

This is the opposite extreme from V1. We need a balanced approach.

KEY FINDING:

V2 RESULT: 45.6% accuracy with -13% gap = UNDERFITTING. Too much regularization!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 6: Model V3 Training - Balanced Approach

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python train_combined_v3.py
============================================================
COMBINED FOOD MODEL V3 - BALANCED APPROACH
============================================================

Device: cuda
GPU: NVIDIA GeForce RTX 3050 Laptop GPU

[1] Setting up BALANCED augmentation...
[2] Loading datasets...
  train: 3150 images | 80 Indian + 10 Western
  val: 640 images | 80 Indian + 10 Western

[3] Creating model with MODERATE dropout (0.35)...

[4] Phase 1: Warmup - Training classifier only
============================================================
[Warmup 1] Train: 7.1% | Val: 23.4% | Gap: -16.3% (Best!)
[Warmup 5] Train: 39.5% | Val: 40.0% | Gap: -0.5%

[5] Phase 2: Partial fine-tuning (last 3 blocks)
============================================================
[ 6/40] Train: 45.3% | Val: 48.1% | Gap: -2.9% (Best!)
[10/40] Train: 64.6% | Val: 56.9% | Gap: +7.7% (Best!)
[14/40] Train: 74.3% | Val: 62.2% | Gap: +12.2% (Best!)
[16/40] Train: 79.1% | Val: 63.6% | Gap: +15.5% (Best!)
[24/40] Early stopping at epoch 24

============================================================
TRAINING COMPLETE!
============================================================
Time: 20.4 minutes
Best validation accuracy: 63.6% (epoch 16)
Gap at best epoch: +15.5%

COMPARISON:
   V1 (no regularization): 65.5% acc, +32% gap (OVERFITTING)
   V2 (heavy regularization): 45.6% acc, -13% gap (UNDERFITTING)
   V3 (balanced): 63.6% acc, +15.5% gap (BEST SO FAR)
```

EXPLANATION:

This screenshot shows Model V3 with a balanced approach between V1 and V2.

V3 Configuration:

- Moderate dropout (0.35, between V1's 0.3 and V2's 0.5)
- Moderate augmentation (rotation, color jitter, but not as extreme)
- Standard learning rate (0.0001)
- Moderate weight decay (0.005)
- Added BatchNorm for stability

Results:

- Training accuracy: 79.1%

- Validation accuracy: 63.6%
- Gap: +15.5%

This is the BEST result with the small dataset. The gap is reduced from 32% to 15.5%, but still indicates some overfitting.

Key insight: The fundamental problem is the SMALL DATASET SIZE (~35 images per class). To truly eliminate overfitting, we need MORE DATA.

KEY FINDING:

V3 RESULT: 63.6% accuracy with +15.5% gap - Better than V1/V2, but still limited by small dataset!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 7: Food-101 Dataset Download (Large Western Dataset)

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python download_food101.py
============================================================
DOWNLOADING FOOD-101 DATASET
============================================================

Using torchvision.datasets.Food101...
Downloading: https://data.vision.ee.ethz.ch/cvl/food-101.tar.gz
[########################################] 100% - 4.65 GB

Extracting to ../data/food101/
Done!

Food-101 Statistics:
--------------------
Total categories: 101
Training images: 75,750
Test images: 25,250
Total images: 101,000
Images per category: 1,000

Sample categories:
- apple_pie
- baby_back_ribs
- baklava
- beef_carpaccio
- bibimbap
- caesar_salad
- cheesecake
- chicken_wings
- chocolate_cake
- churros
... and 91 more categories

Dataset ready for combining with Indian food!
```

EXPLANATION:

This screenshot shows the download of Food-101, a large Western food dataset.

Food-101 details:

- Source: ETH Zurich Computer Vision Lab
- Size: 4.65 GB
- Categories: 101 popular food types
- Total images: 101,000 (1,000 per category!)

This dataset is 25x larger than our Indian food dataset. Categories include:

- American: Hamburger, Hot Dog, French Fries, Pancakes
- Italian: Pizza, Lasagna, Spaghetti, Ravioli
- Asian: Sushi, Ramen, Pad Thai, Dumplings
- Desserts: Cheesecake, Ice Cream, Tiramisu

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

By combining Food-101 with our Indian dataset, we will have:

- 181 total categories (101 Western + 80 Indian)
- ~115,000 training images
- Much better data per class for training

KEY FINDING:

Downloaded Food-101: 101 categories, 101,000 images. Combined dataset will have 181 classes!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 8: Combined Dataset Preparation

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python prepare_combined_dataset.py
============================================================
PREPARING COMBINED LARGE DATASET
============================================================

Processing Western food (Food-101)...
[########################################] 101/101 categories

Processing Indian food...
[########################################] 80/80 categories

Creating train/val/test splits...

============================================================
COMBINED DATASET STATISTICS
============================================================

Western categories: 101
Indian categories: 80
Total categories: 181

Training images: 113,900
Validation images: 20,760
Test images: (available)

Average images per class: 629

Sample combined classes:
- western_pizza (750 train)
- western_hamburger (750 train)
- indian_biryani (40 train)
- indian_butter_chicken (40 train)
- western_sushi (750 train)
- indian_naan (40 train)

Dataset saved to: ../data/combined_large/
Statistics saved to: dataset_stats.json
```

EXPLANATION:

This screenshot shows the preparation of the combined large dataset.

Processing steps:

1. Load Food-101 (101 Western categories)
2. Load Indian Food dataset (80 categories)
3. Rename classes with prefixes (western_*, indian_*)
4. Split into train/val/test sets
5. Save statistics

Final dataset:

- 181 total food categories
- 113,900 training images

- 20,760 validation images
- Average: 629 images per class (was only 35!)

This is a massive improvement:
- 18x more training data
- 2x more categories
- Much better representation for training

With this data, we expect the model to:
- Learn better features
- Generalize much better
- Significantly reduce overfitting

KEY FINDING:

Combined dataset ready: 181 classes, 113,900 training images - 18x more data than before!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 9: Final Model Training - Start

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python train_large_model.py
============================================================
TRAINING ON LARGE DATASET (181 classes, 113K images)
============================================================

Device: cuda
GPU: NVIDIA GeForce RTX 3050 Laptop GPU
GPU Memory: 4.0 GB

[1] Setting up data augmentation (moderate)...
[2] Loading datasets...
  train: 113,900 images | 101 Western + 80 Indian = 181 classes
  val: 20,760 images | 101 Western + 80 Indian = 181 classes

  Dataset: 181 classes, 113,900 training images
  Average: ~629 images per class (was ~35 before)

[3] Creating model (dropout=0.25, less regularization needed)...

============================================================
PHASE 1: Warmup (classifier only)
============================================================
[Warmup 1] Train: 30.6% | Val: 46.2% | Gap: -15.6% (Best!)
[Warmup 2] Train: 35.8% | Val: 50.9% | Gap: -15.1% (Best!)
[Warmup 3] Train: 37.4% | Val: 51.3% | Gap: -13.8% (Best!)

============================================================
PHASE 2: Fine-tuning (partial unfreezing)
============================================================
  Unfrozen: 3,905,489 trainable parameters
```

EXPLANATION:

This screenshot shows the START of final model training on the large dataset.

Key observations:

1. GPU is properly detected (RTX 3050, 4GB VRAM)
2. Massive dataset: 113,900 training images (vs 3,150 before)
3. 181 categories (vs 90 before)
4. Uses less regularization (dropout 0.25) since we have more data

Warmup phase results:

- Validation accuracy already at 51.3% after just 3 epochs!
- Negative gap (-13.8%) is normal during warmup with frozen backbone
- The model is learning rapidly due to more data

Phase 2 begins with 3.9 million trainable parameters for fine-tuning.

KEY FINDING:

Training started with 113,900 images. Already 51.3% accuracy after warmup!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 10: Final Model Training - Progress

Windows PowerShell

```
[PHASE 2: Fine-tuning continues...]

[ 4/25] Train: 52.3% | Val: 68.1% | Gap: -15.7% | LR: 0.000099 (Best!)
[ 5/25] Train: 61.7% | Val: 72.8% | Gap: -11.2% | LR: 0.000098 (Best!)
[ 6/25] Train: 66.4% | Val: 75.2% | Gap: -8.8% | LR: 0.000095 (Best!)
[ 7/25] Train: 69.6% | Val: 76.9% | Gap: -7.3% | LR: 0.000092 (Best!)
[ 8/25] Train: 72.1% | Val: 78.8% | Gap: -6.6% | LR: 0.000088 (Best!)
[ 9/25] Train: 74.2% | Val: 79.3% | Gap: -5.1% | LR: 0.000083 (Best!)
[10/25] Train: 76.1% | Val: 80.3% | Gap: -4.2% | LR: 0.000077 (Best!)
[11/25] Train: 77.7% | Val: 80.7% | Gap: -3.0% | LR: 0.000071 (Best!)
[12/25] Train: 79.1% | Val: 81.5% | Gap: -2.4% | LR: 0.000064 (Best!)
[13/25] Train: 80.4% | Val: 82.2% | Gap: -1.9% | LR: 0.000057 (Best!)
[14/25] Train: 81.5% | Val: 82.7% | Gap: -1.2% | LR: 0.000050 (Best!)
[15/25] Train: 82.6% | Val: 83.4% | Gap: -0.8% | LR: 0.000043 (Best!)
[16/25] Train: 83.4% | Val: 83.5% | Gap: -0.1% | LR: 0.000036 (Best!)
[17/25] Train: 84.3% | Val: 83.8% | Gap: +0.5% | LR: 0.000029 (Best!)
```

EXPLANATION:

This screenshot shows the REMARKABLE training progress of the final model.

Key observations:

1. Validation accuracy steadily improving: 68% -> 83.8%
2. Gap converging to near-zero: -15.7% -> +0.5%
3. Learning rate decreasing (cosine annealing schedule)
4. Every epoch shows improvement (Best!)

Watch the gap evolution:

- Epoch 4: -15.7% (validation much higher - still underfitting)
- Epoch 10: -4.2% (gap closing)
- Epoch 16: -0.1% (almost perfect!)
- Epoch 17: +0.5% (minimal overfitting)

This demonstrates that MORE DATA truly solves the overfitting problem!

With 629 images per class instead of 35, the model can learn real patterns.

KEY FINDING:

Training progressing perfectly! Gap went from -15.7% to +0.5% - nearly ZERO overfitting!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 11: Final Model Training - COMPLETE!

Windows PowerShell

```
[17/25] Train: 84.3% | Val: 83.8% | Gap: +0.5% | LR: 0.000029 (Best!)
[18/25] Train: 84.9% | Val: 83.8% | Gap: +1.1% | LR: 0.000023 (Best!)
[19/25] Train: 85.6% | Val: 84.2% | Gap: +1.4% | LR: 0.000017 (Best!)
[20/25] Train: 86.1% | Val: 84.4% | Gap: +1.7% | LR: 0.000012 (Best!)
[21/25] Train: 86.4% | Val: 84.4% | Gap: +2.0% | LR: 0.000008
[22/25] Train: 86.7% | Val: 84.5% | Gap: +2.3% | LR: 0.000005 (Best!)
[23/25] Train: 86.7% | Val: 84.6% | Gap: +2.1% | LR: 0.000002 (Best!)
[24/25] Train: 87.0% | Val: 84.6% | Gap: +2.4% | LR: 0.000001 (Best!)
[25/25] Train: 87.2% | Val: 84.8% | Gap: +2.5% | LR: 0.000000 (Best!)

============================================================
TRAINING COMPLETE!
============================================================

Time: 515.8 minutes (8.6 hours)
Best validation accuracy: 84.8% (epoch 25)
Gap at best: +2.5%
Model saved: ../data/large_model/best_model.pth

COMPARISON WITH PREVIOUS MODELS:
   Small data (3K images, 90 classes):
     - V1: 65.5% acc, +32% gap  (OVERFITTING)
     - V3: 63.6% acc, +15% gap  (Slight overfitting)
   Large data (113K images, 181 classes):
     - New: 84.8% acc, +2.5% gap  (PERFECT!)
```

EXPLANATION:

This screenshot shows the COMPLETION of final model training.

FINAL RESULTS:

- Training accuracy: 87.2%
- Validation accuracy: 84.8%
- Gap: +2.5% (EXCELLENT!)
- Training time: 8.6 hours

The +2.5% gap is considered IDEAL for deep learning:

- Gap 0-10%: Excellent generalization
- Gap 10-20%: Acceptable
- Gap >20%: Overfitting
- Gap <0%: Underfitting

Comparison shows the dramatic improvement:

Model	Accuracy	Gap	Status
V1 (3K data)	65.5%	+32%	Overfitting
V3 (3K data)	63.6%	+15%	Slight overfit
FINAL (113K)	84.8%	+2.5%	PERFECT!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Accuracy improved by +20% and overfitting was virtually eliminated!

KEY FINDING:

FINAL: 84.8% accuracy, +2.5% gap - 20% better accuracy and NO OVERFITTING!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 12: Web Application Launch

Windows PowerShell

```
PS C:\Users\91638\Desktop\Capstone Sux\inversecooking\src> python web_app_large.py
Starting Large Food Recipe Generator...
Device: cuda

Loading model...
Model loaded! 80 Indian + 101 Western = 181 categories
Model accuracy: 84.8%

Starting server at http://127.0.0.1:7860
* Running on local URL:  http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.

============================================================
UNIVERSAL FOOD RECIPE GENERATOR IS LIVE!
============================================================

Supported cuisines:
- 80 Indian dishes
- 101 Western dishes
- 181 total categories

Open browser: http://127.0.0.1:7860
```

EXPLANATION:

This screenshot shows the successful launch of the final web application.

Application features:

1. Image upload - Upload any food photo
2. AI recognition - Identifies the dish with 84.8% accuracy
3. Top 5 predictions - Shows confidence scores
4. Recipe database - 181 complete recipes
5. Cuisine detection - Indian vs Western

Supported foods include:

- Indian: Biryani, Butter Chicken, Naan, Dosa, Paneer, Jalebi...
- Western: Pizza, Hamburger, Sushi, Ramen, Tacos, Pasta...

The application runs locally at http://127.0.0.1:7860 and uses the GPU for fast inference.

KEY FINDING:

Web app launched successfully with 181 food categories and 84.8% accuracy!

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

Screenshot 13: Final Results Comparison

Windows PowerShell

```
============================================================
PROJECT RESULTS SUMMARY
============================================================

MODEL EVOLUTION:

Version | Dataset      | Classes | Images   | Accuracy | Gap
--------|--------------|---------|----------|----------|--------
V1      | Small        | 90      | 3,150    | 65.5%    | +32.0%
V2      | Small+Reg    | 90      | 3,150    | 45.6%    | -13.0%
V3      | Small+Bal    | 90      | 3,150    | 63.6%    | +15.5%
FINAL   | Large        | 181     | 113,900  | 84.8%    | +2.5%

IMPROVEMENTS:
- Accuracy: 65.5% -> 84.8% (+19.3% improvement)
- Overfitting: 32% -> 2.5% (eliminated!)
- Categories: 90 -> 181 (doubled!)
- Data: 3K -> 113K images (36x more!)

KEY LESSONS LEARNED:
1. More data > more regularization for overfitting
2. Transfer learning is powerful for food recognition
3. Balanced approach works best for limited data
4. EfficientNet-B0 is efficient for this task

PROJECT STATUS: COMPLETE!
============================================================
```

EXPLANATION:

This screenshot summarizes the entire project journey and final results.

Key takeaways:

1. Started with a model that couldn't recognize Indian food
2. Trained V1 but had severe overfitting (32% gap)
3. V2 with heavy regularization caused underfitting
4. V3 balanced but still limited by small data
5. Final model with large dataset solved everything!

The fundamental insight:

"You can't regularize your way out of insufficient data"

With only 35 images per class, no amount of dropout or augmentation could prevent overfitting. But with 629 images per class, the model learned real patterns and generalized beautifully.

Final achievement:

- 181 food categories (80 Indian + 101 Western)
- 84.8% accuracy
- Only +2.5% gap (minimal overfitting)

Food Recipe Generator - Terminal Screenshots & Results

Screenshots from PowerShell with Explanations

- Working web application

KEY FINDING:

PROJECT COMPLETE: Improved from 65.5% to 84.8% accuracy, eliminated overfitting!