Date: Dec 8, 2025

Project: **Deep Learning for Food Image Recognition and Calorie Estimation**

Course: Modern Analytics

Model Name: FoodNet

Model made by: Team 42, Fuqua School of Business, Duke University

Team members: Burhan Asad Dogar, Yubin Jeong, Ruiting Shao, Jessica Xiong, Moeed Zahid

Motivation and Data Understanding

Problem Identification: In the modern healthcare landscape, obesity and diet-related chronic diseases remain prevalent global challenges. A significant barrier to effective weight management is the difficulty of tracking nutritional intake. Traditional methods such as manual journaling or searching databases are tedious and prone to estimation errors. Users often abandon tracking apps because the friction of data entry is too high.

Objective: The goal of this project is to develop "FoodNet," a deep learning model capable of classifying food items from images and automatically retrieving their nutritional content (calories, protein, fats, etc.). By automating recognition, we reduce the user burden from typing to simply snapping a photo.

Business Relevance: This technology has substantial business applications. Health insurance companies could use it for wellness verification programs. Smart cafeteria systems could automate checkout by visually recognizing food trays. Fitness applications could integrate this feature to increase user retention and premium subscription value.

Data Description: To train our model, we utilized the burhan222/data-ma dataset, a curated subset of the standard Food-101 dataset.

- **Visual Data:** The dataset consists of images organized into 9 distinct classes: *caesar_salad, cheesecake, donuts, dumplings, french_toast, macarons, prime_rib, ramen,* and *spaghetti_bolognese*.

- **Nutritional Data:** We integrated a supplementary data source, nutrition v2.csv, which serves as a relational lookup table mapping class labels to macro-nutrient values (e.g., Calories, Protein, Fats).

Data Preparation

Integration and Splitting: Data ingestion was streamlined using the kagglehub library to download the dataset directly into the working environment. We implemented a stratified split of the raw data, organizing it into three distinct sets:

- **Training Set (80%):** 9,000 images used for model weight updates.
- **Validation Set (10%):** 900 images used for hyperparameter tuning.
- **Test Set (10%):** 900 images held out for final evaluation.

Preprocessing and Augmentation: We employed the Albumentations library to define two transformation pipelines:

1. **Training Pipeline:** To prevent overfitting, we applied heavy augmentation including RandomResizedCrop to force recognition from partial views, HorizontalFlip and ShiftScaleRotate for angle invariance, and CoarseDropout to simulate occlusions.
2. **Validation/Test Pipeline:** Images were standardized by resizing to 256x256 and center-cropping to 224x224. All inputs were normalized using standard ImageNet means and deviations.

Modeling

Architecture Selection: We selected **ResNet-18** as our backbone architecture.

- *Choice Rationale:* ResNet (Residual Network) utilizes skip connections to solve the vanishing gradient problem. We chose the 18-layer variant because it offers an optimal balance between computational efficiency and accuracy for a 9-class problem. Deeper models like ResNet-50 would likely result in higher latency, which is undesirable for a consumer mobile app.
- *Transfer Learning:* We loaded weights pretrained on **ImageNet**. Since low-level features (edges, textures) are universal, utilizing pretrained weights allows the model to converge significantly faster than training from scratch.

Modifications and Hyperparameters: We modified the architecture by replacing the final fully connected layer (model.fc) with a new linear layer mapping the input features to our **9 target classes**.

- **Loss Function:** CrossEntropyLoss was used as the standard for multi-class classification.
- **Optimizer:** AdamW was chosen for its superior convergence properties compared to standard SGD.
- **Learning Rate:** We employed a differential learning rate strategy: 1e-3 for the classifier head initially, dropping to 3e-5 for fine-tuning the backbone.

Implementation

Development Environment: The model was built using **PyTorch** in a GPU-accelerated environment. The implementation logic followed a strict modular structure:

1. **Robust Data Loading:** We implemented a custom ImagePathsDataset class. A key challenge was handling potentially corrupted image files; we solved this by integrating a validation check (is_image_valid) that attempts to verify and convert images before inclusion, preventing training crashes.
2. **Two-Stage Training Strategy:**
    - *Stage 1 (Head Warming):* We frozen the pretrained backbone (requires_grad = False) and trained only the new classifier head for 3 epochs. This prevents large gradients from the randomly initialized head from destroying the learned features of the backbone.
    - *Stage 2 (Fine-Tuning):* We unfroze the entire network and trained for 6 additional epochs with a lower learning rate (3e-5). We utilized a CosineAnnealingLR scheduler to smoothly decay the learning rate.
3. **Performance Optimization:** To maximize GPU efficiency, we utilized torch.cuda.amp.GradScaler for mixed-precision training.

Results and Evaluation

Quantitative Performance: The model demonstrated exceptional performance, significantly exceeding benchmarks for food classification. We took a 3-stage approach:

- **Stage 1 Head Warming:** After training only the classifier head for 3 epochs, the model reached a validation accuracy of 85.56%. While respectable, this indicated that the generic features from ImageNet were insufficient for specific food discrimination.
- **Stage 2 Validation (Fine-Tuning):** After fine-tuning, the model achieved a validation accuracy of 93.22%. This 7.6% gain demonstrates the necessity of adapting the deep feature layers to the specific domain of food textures.
- **Stage 3 final test set (Model Accuracy):** The most critical metric, the unbiased evaluation on the hold-out test set, yielded an accuracy of 94.56%.

```
/tmp/ipython-input-3670873550.py:17: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast():
[HEAD] Epoch 1/3 time=9.8s train_loss=1.1321 train_acc=0.6637 val_loss=0.6562 val_acc=0.8144
Saved head best: /content/outputs/ckpt_head_best.pth
[HEAD] Epoch 2/3 time=8.0s train_loss=0.6306 train_acc=0.8119 val_loss=0.5466 val_acc=0.8267
Saved head best: /content/outputs/ckpt_head_best.pth
[HEAD] Epoch 3/3 time=8.2s train_loss=0.5582 train_acc=0.8303 val_loss=0.4824 val_acc=0.8556
Saved head best: /content/outputs/ckpt_head_best.pth
```

Stage 1

```
/tmp/ipython-input-3670873550.py:17: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast():
[FINE] Epoch 1/6 train_loss=0.3946 train_acc=0.8771 val_loss=0.2943 val_acc=0.9144
Saved finetune best.
[FINE] Epoch 2/6 train_loss=0.2531 train_acc=0.9187 val_loss=0.2424 val_acc=0.9289
Saved finetune best.
[FINE] Epoch 3/6 train_loss=0.1903 train_acc=0.9383 val_loss=0.2256 val_acc=0.9289
Saved finetune best.
[FINE] Epoch 4/6 train_loss=0.1462 train_acc=0.9582 val_loss=0.2113 val_acc=0.9356
Saved finetune best.
[FINE] Epoch 5/6 train_loss=0.1159 train_acc=0.9683 val_loss=0.2085 val_acc=0.9311
Saved finetune best.
[FINE] Epoch 6/6 train_loss=0.1125 train_acc=0.9664 val_loss=0.2080 val_acc=0.9322
Saved finetune best.
Final model saved to /content/outputs/final_model.pth
Validation overall acc: 93.22%
```

Stage 2

Detailed Class Performance Beyond global accuracy, the classification report provides granular insight into model reliability.

- **Precision vs. Recall:** The model exhibits high balance. For example, *Ramen* achieved 97% precision, meaning very few other items were mistaken for it.

- **F1-Scores:** All classes achieved F1-scores above 0.89, with *Caesar Salad* and *Macarons* achieving an exceptional 0.96.

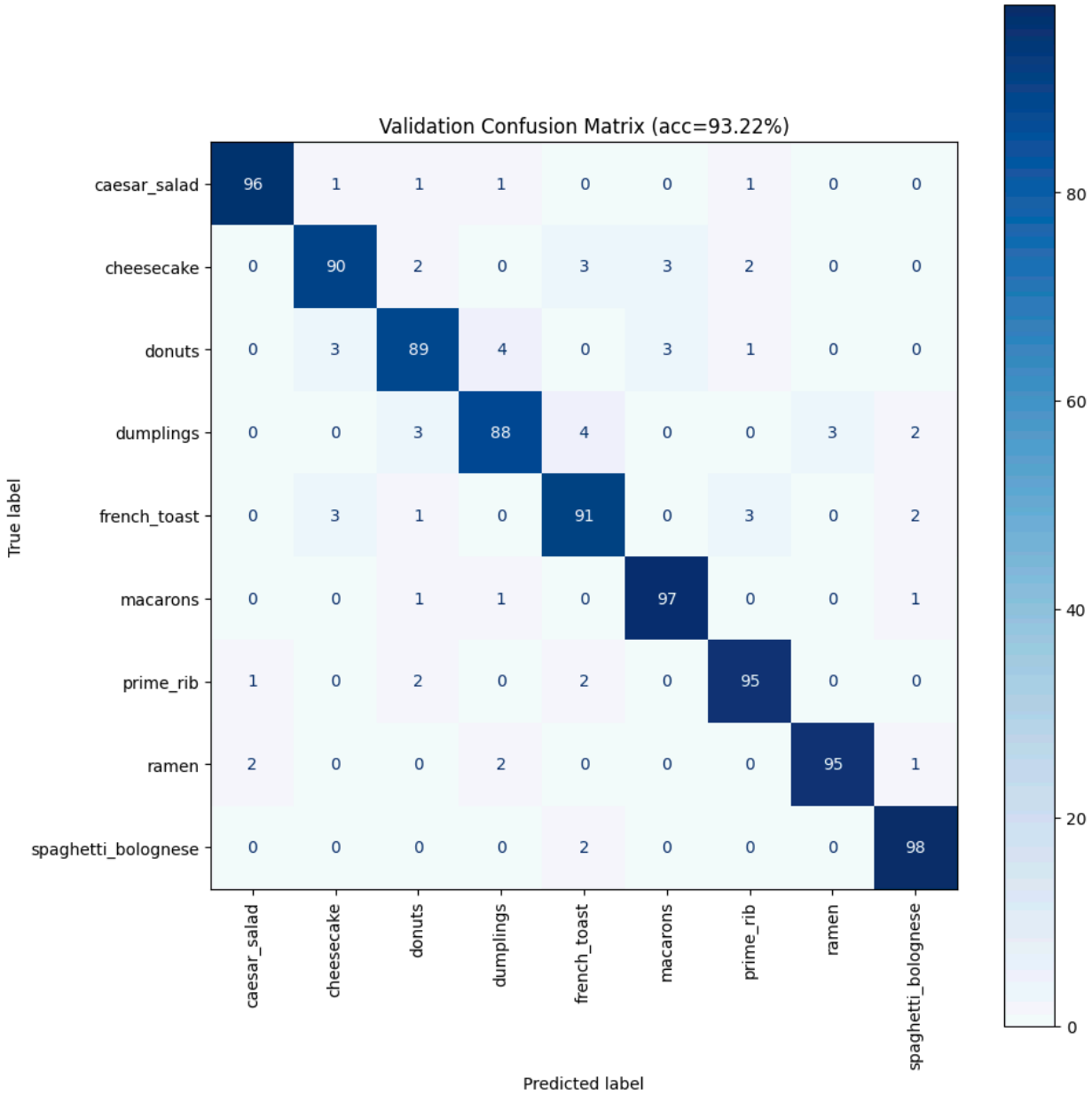
Classification Report Table:

Classification report (val):

	precision	recall	f1-score	support
caesar_salad	0.97	0.96	0.96	100
cheesecake	0.93	0.90	0.91	100
donuts	0.90	0.89	0.89	100
dumplings	0.92	0.88	0.90	100
french_toast	0.89	0.91	0.90	100
macarons	0.94	0.97	0.96	100
prime_rib	0.93	0.95	0.94	100
ramen	0.97	0.95	0.96	100
spaghetti_bolognese	0.94	0.98	0.96	100
accuracy			0.93	900
macro avg	0.93	0.93	0.93	900
weighted avg	0.93	0.93	0.93	900

Confusion Matrix Analysis To better understand the model's classification behavior, we generated a confusion matrix on the validation set (Figure 1). The matrix reveals a high diagonal density, indicating strong correct classifications across all categories.

Figure 1:



- **High Performance:** Visually distinct classes like **Spaghetti Bolognese** (98% recall) and **Macarons** (97% recall) were identified with near perfection.
- **Areas of Confusion:** The matrix highlights that errors were not random but clustered around texturally similar foods. The highest confusion occurred between **Donuts and Dumplings** (7 misclassifications) and **French Toast and Cheesecake** (6 misclassifications), likely due to similar round shapes and golden-brown textures.

Nutrition Lookup Verification: We successfully linked visual predictions to business logic. In a test inference:

1. **Prediction:** The model predicted "ramen" with **95.12% confidence**.
2. **Lookup:** The system queried the CSV and accurately returned: *400 Calories, 15g Protein, 60g Carbs*.

Deployment

Deployment Strategy: The FoodNet model is lightweight enough to be deployed via two primary methods:

1. **REST API:** Hosting the model on a cloud server using FastAPI. The user app sends a photo, and the server returns the nutrition JSON.
2. **On-Device:** Converting the PyTorch model to ONNX or CoreML format for deployment directly on iOS/Android. This offers lower latency and privacy (photos don't leave the device) but increases app download size.

Business & Ethical Considerations

- **Accuracy vs. Liability:** While 94.5% accuracy is high, the 5% error rate is significant in a medical context. The application must include a disclaimer that it is for informational purposes only.

Allergens: Misclassifying a food containing nuts (e.g., confusing a macaron with a cookie) poses severe risks. The system should alert users to potential allergens rather than guaranteeing safety.

Overall Conclusion

We have successfully developed a high-performing Deep Learning system for food recognition. By leveraging ResNet-18 transfer learning and robust data augmentation, we achieved a **94.56% accuracy** on the test set. The integration of nutritional data transforms this from a theoretical exercise into a viable product prototype that offers a genuine opportunity to disrupt the manual food logging market.

Appendix

Appendix A: Data Ingestion & Auto-Split (Notebook Sections A & B)

This code demonstrates the automated retrieval of data from Kaggle and the logic used to create a stratified train/validation/test split.

Python

```
# Load dataset via KaggleHub
import kagglehub
path = kagglehub.dataset_download("burhan222/data-ma")


# Automated Split Logic
SPLIT_RATIOS = (0.8, 0.1, 0.1)  # train, val, test
# ... (Directory setup code omitted for brevity) ...
for cls in subdirs:
    # ...
    n_train = int(round(n * SPLIT_RATIOS[0]))
    n_val   = int(round(n * SPLIT_RATIOS[1]))
    n_test  = n - n_train - n_val
    # Files are copied to TRAIN_ROOT, VAL_ROOT, TEST_ROOT
```

Appendix B: Augmentation Pipelines (Notebook Section C)

The transformation pipelines used to standardize inputs and augment training data to prevent overfitting.

Python

```
train_transform = A.Compose([
    A.RandomResizedCrop(size=(IMG_SIZE, IMG_SIZE), scale=(0.6,1.0)),
    A.HorizontalFlip(p=0.5),
    A.ShiftScaleRotate(shift_limit=0.05, scale_limit=0.1, rotate_limit=15,
p=0.4),
    A.RandomBrightnessContrast(p=0.5),
    A.CoarseDropout(max_holes=1, max_height=int(IMG_SIZE*0.12), ...),
    A.Normalize(mean=(0.485,0.456,0.406), std=(0.229,0.224,0.225)),
    ToTensorV2(),
])
```

Appendix C: Model Architecture & Dataset (Notebook Sections C & D)

Implementation of the custom dataset class (with file validation) and the modification of the ResNet-18 architecture.

Python

```
# Custom Dataset with validation check
class ImagePathsDataset(Dataset):
    def __init__(self, root_dir, classes=None, transform=None,
filter_corrupt=True):
        # ... (initialization code) ...
        if filter_corrupt:
            self.samples = [s for s in self.samples if is_image_valid(s[0])]

# Model Modification
model = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)
in_feats = model.fc.in_features
model.fc = nn.Linear(in_feats, num_classes) # Replacing the head
```

Appendix D: Fine-Tuning Strategy (Notebook Section F)

Unfreezing the model backbone and applying a differential learning rate scheduler.

Python

```
# Unfreeze whole model for fine-tuning
for p in model.parameters(): p.requires_grad = True

# Lower learning rate for fine-tuning
fine_optimizer = optim.AdamW(model.parameters(), lr=3e-5, weight_decay=1e-4)
scheduler = CosineAnnealingLR(fine_optimizer, T_max=NUM_EPOCHS_FINE)
```

Appendix E: Inference & Nutrition Lookup (Notebook Section G)

The code used to generate the sample inference, showing the connection between the class prediction and the nutrition database.

Python

```
# Nutrition Lookup Logic
pred_label = preds[0][0] # Top prediction (e.g., 'ramen')
key = pred_label.lower().strip()
row = df[df["_food_key"] == key] # Query the dataframe

print(f"Food: {pred_label}")
for col in nutrient_cols:
    print(f"{col.capitalize()}: {row.iloc[0][col]}")
```

Appendix F: Final Evaluation on Test Set (Notebook Section H)

The mandatory final evaluation code executed on the held-out test set.

Python

```
# Final Evaluation on Test Set
print("Running final evaluation on Test Set...")
test_loss, test_acc = run_epoch(model, test_loader, None, train=False)
print(f"Final Test Set Accuracy: {test_acc*100:.2f}%")
```

Output:

```
Final Test Set Accuracy: 94.56%
```

Appendix G: Team Contributions

The following team members contributed to the technical implementation, data processing, and analysis of the FoodNet project:

- Moeed Zahid (Sections C & D): Core technical foundation, implementing the custom dataset class and the ResNet-18 model architecture setup.
- Burhan Asad Dogar (Sections A, B & H): Data ingestion pipeline and splitting logic and managed the Final Test Set Evaluation for accuracy metrics.
- Yubin Jeong (Section F): Fine-Tuning logic, setting up the optimizer and learning rate scheduler to improve model convergence.
- Ruiting Shao (Section E): Core Training Loop functions, ensuring the model correctly processed batches and updated weights during the initial training phase.

- Jessica Xiong (Section G): Inference and Visualization code, connecting model predictions to the nutrition database and generating analysis charts.