# Workshop on Deep Learning

## Project Report – Carb Inspector

**Gur Abraham & Hila Ofek**

## (1) Problem setting:

Nutrition tracking is important for health management. People with diabetes must monitor carbohydrate intake to regulate insulin and blood sugar, athletes also track carbs as part of their dietary plans. Nutritional awareness is beneficial for the public for a healthier living. Yet, accurate carb tracking is inconvenient, often requiring food weighing and manual nutritional lookups.

Our project addresses this by offering a web application where users upload an image of a food dish and receive two outputs:

1. A **predicted food label** (the type of food in the image)
2. A **predicted carbohydrate amount** in grams

## (2) Approach and Existing Frameworks:

To face the dual task of food classification and carbohydrate prediction, we used transfer learning in the form of a Convolutional Neural Network (CNN). CNNs excel at extracting hierarchical features from images, making them widespread and our natural choice. Our CNN backbone is EfficientNet-B4. Recall transfer learning is an ML technique in which knowledge is reused to enhance performance in related task, here the backbone's image features are reused to learn the classification and regression tasks.

EfficientNet (Tan M., Le Q. V., 2019, Google Research), is a scalable CNN framework that balances the **width** (number of channels), **depth** (number of layers) and **resolution** (input image size) of the network, using compound scaling method, thus achieving strong accuracy with relatively few parameters. Pretrained on ImageNet (14M images, 20K categories), providing a strong starting point for our model's fine tuning. Among its variants, we used **EfficientNet-B4**, which offered us sufficient capacities.

We also tested other backbones (EfficientNet-B0, B3, B5, NASNetMobile, MobileNet-V3, ResNet50, Inception-V3), but EfficientNet-B4 offered the best trade-off between accuracy and computational feasibility under the GPU limitations.

While the backbone extracts features from the input images, we extended the model into a **multi-headed CNN model:**

1. A **classification head** to predict the food label (e.g. "apple", "pizza", "bread").
2. A **regression head** to predict the carbohydrate content (in grams).

This design allows the model to learn both the categorical and the numerical targets simultaneously, encouraging the shared backbone to capture visual features relevant to recognition as well as fine-grained cues related to carbohydrate content.

**Alternative architecture: two-model split.** We tried training two independent models – one for classification and one for regression. This design underperformed the joint multi-head approach. Details in section (4).

## (3) Datasets:

We trained our model on Nutrition5k Dataset (Thames Q. at al., 2021, Google Research), which pair meals images with metadata such as mass, ingredients and macronutrients (including carbs). To adapt it for our task: We filtered for single-ingredient dishes, ensuring the model's output carb values match the visible food, avoiding ambiguity of mixed dishes. For each included dish, we extracted the **image path**, **meal name**, **total mass (gr)** and **carbohydrates (gr)** from the metadata. Images are accessed from Kaggle and the metadata from Google Cloud Storage (GCS). We discarded meals with missing values.

After filtering, we obtain **6760 images** across **98 Food categories**. We wrapped them in a Hugging Face dataset (containing: image, GCS path, mass, carbs amount and label), and split the data into **training** (80%) and **testing** (20%) sets.

We also experimented with other datasets but did not include them in the final model:

- **FoodPics Extended 2022:** only 568 usable images – too few to meaningfully affect the output; also it's higher image resolution degraded performance at overlapping classes.
- **Food-101:** provides only images and labels, without metadata (carbs). using it for the classification head only on overlapping categories did not improve the accuracy or loss. It is likely due to label-space imbalance (N5K provides both classification and regression info, whereas food-101 provide only classification info, thus unequaly trained featured for the tasks), domain differences in resolution and quality, and bias the feature space towards classification that mismatched the regression-head's feature space.

## (4) Training and Selected Results:

### (4.1) The final model structure:

We fine-tuned EfficientNet-B4 as a shared backbone, replacing the final layer with two heads, while all other EfficientNet layers remain intact.:

- **Classification head**: Dropout(p=0.2) → Linear(in_feats → num_classes)
- **Regression head**: Dropout(p=0.2) → Linear(in_feats → num_classes) → Softplus(β=1) enforcing non-negative carbohydrate predictions.

Output: We end with two vectors with num-class components – the first vector, outputted from the classification head, has the probability the input image belongs to each class. And the second, from the regression head, has the predictions the corresponding carb value in grams had the input image belonged to that class.

Data and Preprocessing: the input uses the EfficientNet-B4 default transform (resize to $380 \times 380$, ImageNet normalization). We wrapped the dataset with a PyTorch dataset, returning {pixel_value, label, carbs_val} and trained with DataLoader (batch size=16).

Training: we train for 10 epochs with the Adam optimizer ($\mathrm{lr} = 1\mathrm{e}-4$), that provides stable fine-tuning. The loss is the weighted sum of the classification's Cross-Entropy and the regression's SmoothL1 loss: $\boldsymbol{\mathcal{L}} = \mathbf{CE}(\hat{\mathbf{y}}_{\mathbf{cls}}, \mathbf{y}) + \boldsymbol{\alpha} \cdot \mathbf{SmoothL1}(\hat{\mathbf{y}}_{\mathbf{carb}}, \mathbf{c}); \boldsymbol{\alpha} = \mathbf{2}$. Giving extra weight to the regression task because regression targets have smaller numeric scale, and empirically $\alpha = 2$ reduced carb error without harming classification accuracy. Validation reports top-1 accuracy (classification) and SmoothL1 loss $[\mathrm{gr}^2]$ (regression).

SmoothL1 Loss: $\mathcal{L}(\hat{y}, y; \beta) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, |y-\hat{y}| \le \beta \\ \beta|y-\hat{y}| - \frac{1}{2}\beta^2, |y-\hat{y}| > \beta \end{cases}$. Behaves like MSE (mean square error) for small errors, and like MAE (mean absolute error) for large errors, combining MSE's sensitivity for small errors with MAE's robustness for large errors. Preventing domination by outliers (like MSE) and retaining a stable convergence (unlike MAE). With $\beta = 1gr$, the loss maintains consistent units of $[gr^2]$.

(4.2) Other variations that we have tried:

- **Freeze/ Unfreeze**: for most variants – we freezed the backbone for the first 5 epochs training only the head then unfreezed it for the last 5, to stabilize optimization. Removing it in the final version improved accuracy.
- **Regression Head Output**: initially the regression head only predicted the carb value for the top-1 prediction, resulting in $88.24\%$ accuracy, $1.7753\,\mathrm{gr}^2$ loss and $2.2328\mathrm{gr}$ average error (MAE) in the test set, with a mostly diagonal confusion matrix (CM). For our application to be able to present top-3 guesses, we changed the output for the vector we discussed before (and removed un/freezing). Thus, our model learns more features in the regression head causing the loss to drop by half, and since the learned features are shared for both heads the CM was diagonalized, and the accuracy improved from the removal of the un/freezing.

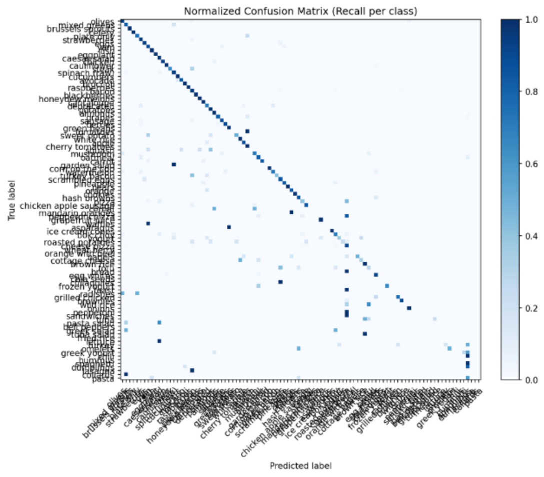
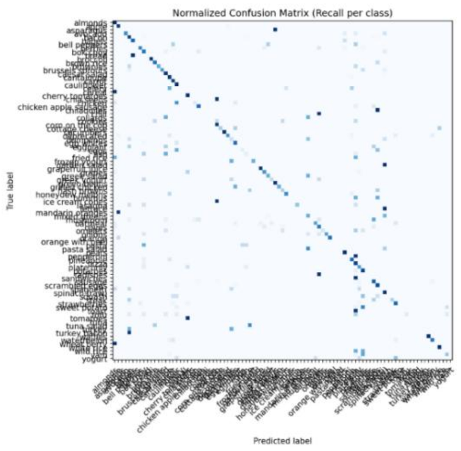


Fig 1: Confusion matrix for single regression output model

- **Other backbones**: we compared the classification task alone on different CNN backbones (EfficientNet B0, B3, B5, NASNetMobile, MobileNet-V3, ResNet-50, Inception-V3) on food-101 and nutrition5k databases (separately and combined) and chose EfficientNet-B4 that run reliably and gave best results under GPU limits (sec 2).
- **Loss weights $\boldsymbol{\alpha}$:** empirically tuning the parameter in $\mathcal{L} = \mathrm{CE} + \alpha \cdot \mathrm{SmoothL1}$ for $\alpha = \{0.5,1,2,3,5\}$ showed best tradeoff in $\alpha = 2$.

* **Augmentations**: MixUp and CutMix achieved $\sim 80\%$ accuracy and $\sim 2.5gr^2$ loss, and AutoAugment degraded even further to $\sim 70\%$ accuracy, so they were excluded. MixUp changes the eigen base of the samples by linear blending of two samples, their labels and their carb estimation with a $\lambda \sim f_\beta(x; \alpha, \alpha)$ such as $x' = \lambda x_i + (1-\lambda) x_j$, $f_\beta(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{Beta}(\alpha,\beta)}$ the beta distribution, $\mathrm{Beta}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$. CutMix similarly changes the base by patching a rectangle around the sample with ratio accordingly $\lambda' = 1 - \frac{\text{patch area}}{\text{image area}}$. AutoAugment also performs transformations such as rotations and translations on the samples.
* **Regression Head MLP**: a Multi Level Preceptron (MLP) layer was applied to the regression head: $\mathrm{Linear(in\_feats} \to 256) \to \mathrm{ReLU} \to \mathrm{Dropout}(0.2) \to \mathrm{Linear}(256 \to 1)$ a MLP is a 2-level NN that should have helped the regression head learn more info for the single carb estimation, by capturing non-linear relationships between different features. Resulting with $84.91\%$ accuracy and $2.2563gr^2$ loss, we removed it to later replace it with a vector output instead of a single output.
* **MSE regression loss:** initially the regression loss was MSE. As discussed in sec 4.1, MSE penalizes large errors, so we replaced it with SmoothL1 that uses MSE's sensitivity in small errors while overriding said penalization.
* **Optimizers**: we used AdamW with differential learning rates (lr) instead of Adam with a constant lr. Because the backbone is pretrained and only fine-tuned so it needs smaller rate than the heads which are trained entirely. However, this caused a ~10% accuracy drop, probably because it affected the binding of the heads to the backbone.
* **Two model split**: as discussed in section 2, we trained also two different EfficientNet-B4 models, one for classification only and the other for regression only. This underperformed or multiheaded architecture, with $82.77\%$ accuracy and $4.4711gr^2$ loss. We tried this split because some of our variants showed a slight tradeoff between the accuracy and the loss. We believe the split underperformed since the multiheaded backbone learns features simultaneously for both tasks, thus each feature is useful for both tasks since the model knows it does 2 tasks, so the classification features now also hold info for the regression and vice versa. Splitting the model removes the shared info, leading to the underperformance.
* **Dataset combinations**: we added samples from food-1o1 to the classification head on overlapping categories:
  * Kept a 70/30 N5K/F101 ratio (to attempt to not ruin the existing N5K distributions): 34.48% accuracy, 1.8527 loss
  * Merging F101 categories into one relevant N5K category (e.g. green olives + black olives → olives). $72.71\%$ N5K accuracy, $70.43\%$ merged accuracy, $3.9437gr^2$ loss.
  * Different optimizer: 79.44% N5K accuracy, 71.44% merged accuracy, 2.1799 loss
  * Family aware label smoothing: by examining the CM, we saw that several labels are consistently confused with one another. So, we decided to change the regression head to first select a label family then a label within it (e.g. "berries" than "raspberry"). 85.85% N5K acc, 80.79% merged acc, 1.6253 loss, CM diagonalized.



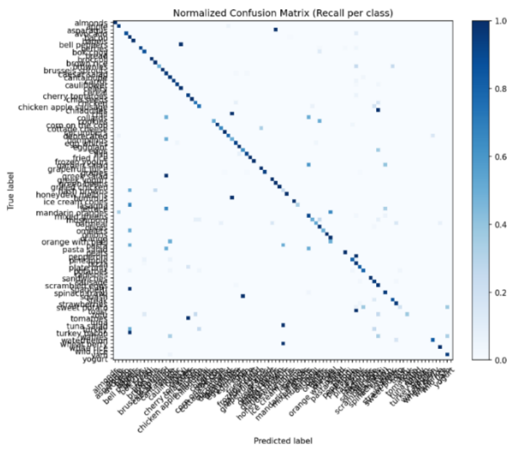


Fig 2a: CM before family-aware label smoothing    Fig 2b: CM after family-aware label smoothing

- <u>CM prior</u>: we added a CM from a previous variant as a prior to the classifier distribution to help classify. The CM was diagonalized but worse metrics overall: 70.74% N5K accuracy, 70.74% merged accuracy, 3.1670 loss.

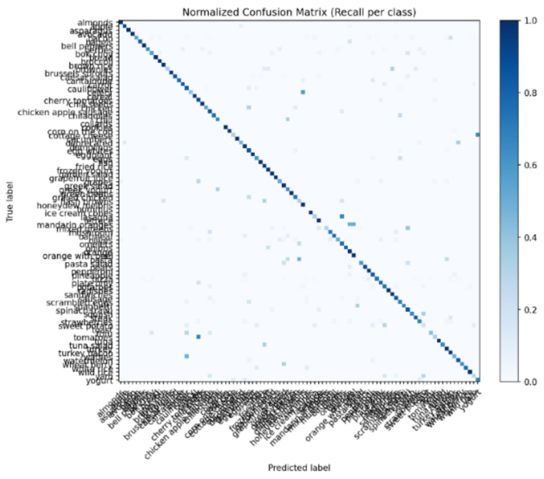


Fig 3: Confusion matrix after adding a CM prior to the model

**<u>Takeaway</u>**: The best variant of the model is: EfficientNet-B4 backbone two-headed model, with num_class outputs for each head, that freezes the backbone for the first 5 epochs then unfreezes the entire model for the rest of the epochs for fine-tuning; with an Adam optimizer and the following loss function $\mathcal{L} = CE + 2.0 \cdot SmoothL1$.

<u>(4.3) selected data outputs from the training and validation:</u>

Our final model runs for ten epochs. At the 10th and final epoch, we reached:

- **Training**: 91.44% classification accuracy and 0.7827 $gr^2$ SmoothL1 loss.
- **Validation**: 91.27% top-1 classification accuracy, 97.17% top-3 classification accuracy and 0.9954 $gr^2$ SmoothL1 loss.
- **Average error in the carb estimation**: $\epsilon_{avg} = 1.3285\ gr$ (calculated with MAE).
- **Confusion matrix**:

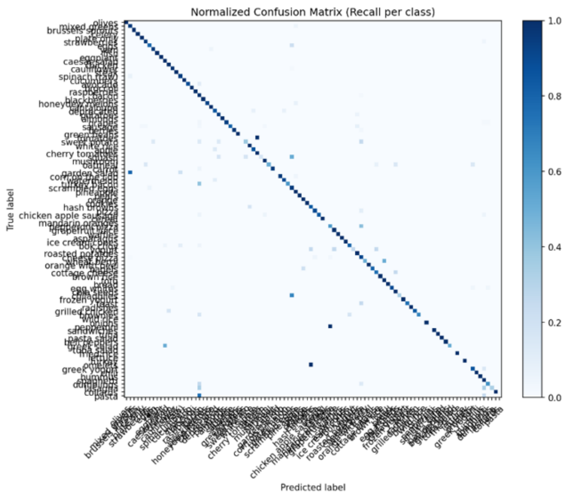


Fig 4: confusion matrix for our final model

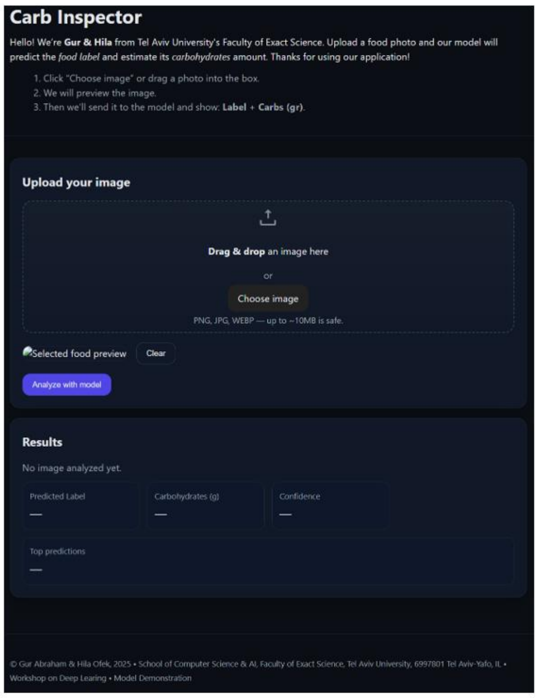
## **<u>(5) Application</u>**:

We created a user-friendly web application for our model: a single-file Python app containing an embedded HTML/CSS/JS UI. Operating instructions for the API/UI are provided in the GitHub README file.

The application contains instructions for the users' usage. The user should (1) upload an image via the "choose image" button or drag-and-drop it into the reserved space. Then (2) the application calls the model and analyses it, and return an output.

<u>The output contains</u>:

- The predicted label
- The carbohydrates amount (in grams)
- The model's confidence of this prediction
- List of the top-3 predictions made by the model.



**Carb Inspector**

Hello! We're **Gur & Hila** from Tel Aviv University's Faculty of Exact Science. Upload a food photo and our model will predict the *food label* and estimate its *carbohydrates* amount. Thanks for using our application!

1. Click "Choose image" or drag a photo into the box.
2. We will preview the image.
3. Then we'll send it to the model and show: **Label** + **Carbs (gr)**.

**Upload your image**

**Drag & drop** an image here

or

PNG, JPG, WEBP — up to ~10MB is safe.

Selected food preview

**Results**

No image analyzed yet.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| — | — | — |

Top predictions

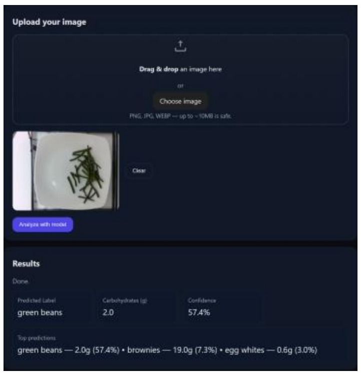
—

© Gur Abraham & Hila Ofek, 2025 • School of Computer Science & AI, Faculty of Exact Science, Tel Aviv University, 6997801 Tel Aviv-Yafo, IL • Workshop on Deep Learing • Model Demonstration

Fig 5: the application interface

<u>(5.1) Application demo:</u>

Here are selected testing results.



**Results**

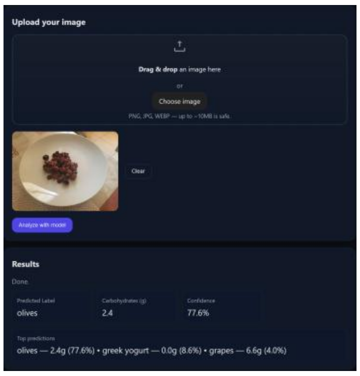
Done.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| green beans | 2.0 | 57.4% |

Top predictions

green beans — 2.0g (57.4%) • brownies — 19.0g (7.3%) • egg whites — 0.6g (3.0%)

(a)



**Results**

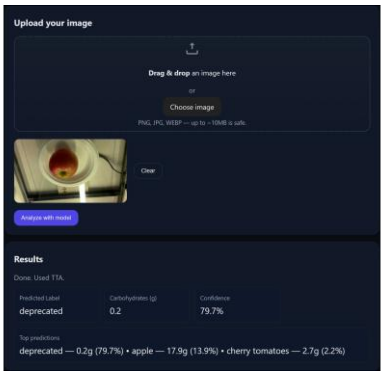
Done.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| olives | 2.4 | 77.6% |

Top predictions

olives — 2.4g (77.6%) • greek yogurt — 0.0g (8.6%) • grapes — 6.6g (4.0%)

(b)



**Results**

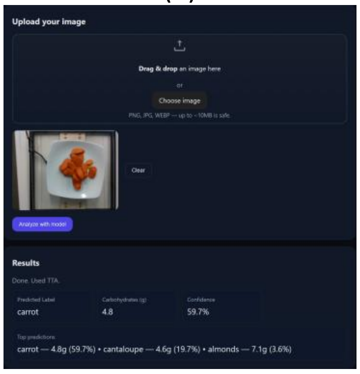
Done. Used TTA.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| deprecated | 0.2 | 79.7% |

Top predictions

deprecated — 0.2g (79.7%) • apple — 17.9g (13.9%) • cherry tomatoes — 2.7g (2.2%)

(c)



**Results**

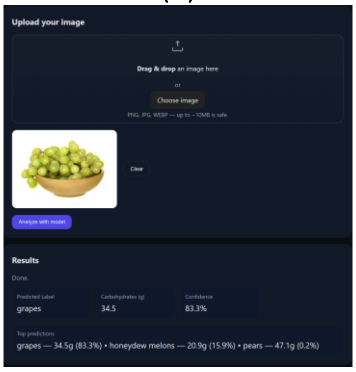
Done. Used TTA.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| carrot | 4.8 | 59.7% |

Top predictions

carrot — 4.8g (59.7%) • cantaloupe — 4.6g (19.7%) • almonds — 7.1g (3.6%)

(d)



**Results**

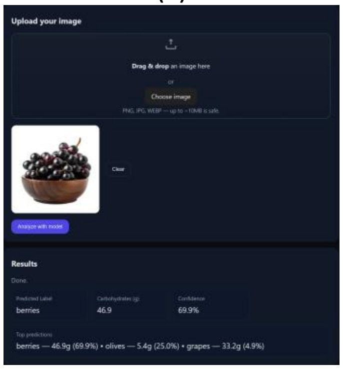
Done.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| grapes | 34.5 | 83.3% |

Top predictions

grapes — 34.5g (83.3%) • honeydew melons — 20.9g (15.9%) • pears — 47.1g (0.2%)

(e)



**Results**

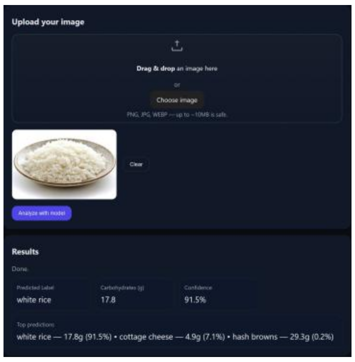
Done.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| berries | 46.9 | 69.9% |

Top predictions

berries — 46.9g (69.9%) • olives — 5.4g (25.0%) • grapes — 33.2g (4.9%)

(f)



Upload your image
Drag & drop an image here
or
Choose image
PNG, JPG, WEBP — up to ~10MB is safe.
Clear
Analyze with model

Results

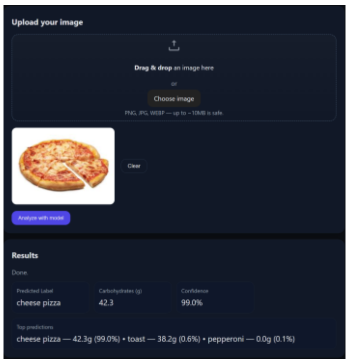
Done.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| white rice | 17.8 | 91.5% |

Top predictions

white rice — 17.8g (91.5%) • cottage cheese — 4.9g (7.1%) • hash browns — 29.3g (0.2%)

(g)



Upload your image
Drag & drop an image here
or
Choose image
PNG, JPG, WEBP — up to ~10MB is safe.
Clear
Analyze with model

Results

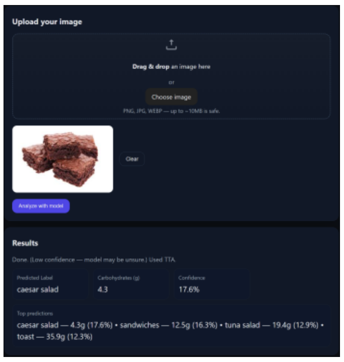
Done.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| cheese pizza | 42.3 | 99.0% |

Top predictions

cheese pizza — 42.3g (99.0%) • toast — 38.2g (0.6%) • pepperoni — 0.0g (0.1%)

(h)



Upload your image
Drag & drop an image here
or
Choose image
PNG, JPG, WEBP — up to ~10MB is safe.
Clear
Analyze with model

Results

Done. (Low confidence — model may be unsure.) Used TTA.

| Predicted Label | Carbohydrates (g) | Confidence |
|---|---|---|
| caesar salad | 4.3 | 17.6% |

Top predictions

caesar salad — 4.3g (17.6%) • sandwiches — 12.5g (16.3%) • tuna salad — 19.4g (12.9%) • toast — 35.9g (12.3%)

(i)

Fig 6: demo usage of the application

<u>An overview of the above examples</u>:

**Dataset images:** figures 6(a), 6(c) and 6(d) are taken from Nutrition5K. the model correctly predicted the label in the top-1st attempt in (a) and (d), while (c) was correct in the 2nd attempt (top-3 accurate). Carbs estimations were also close at the correct labels for all these examples (predicted vs real): $2.0gr$ vs 2.8gr (a), $17.9gr$ vs $19.88gr$ (c) and $4.8gr$ vs $7.5gr$ (d), largest error from our samples.

**Own photo**: fig 6(b) is an image we took of olives, with true carbs amount $2.121gr$, the model predicts close with $2.4gr$ and successfully predicts the label 1st attempt.

**Web images**: figures 6(e-h) taken from google are correct mostly at top-1, with only (f) correct at top-3.

**Failure Case**: in fig 6(i) none of the top-k predictions is correct, but the model flags low confidence ($< 20\%$), warning the user of possible error.

**<u>(6) Result Assessment</u>**:

We compared our model performance to other existing models' performance on our validation set.

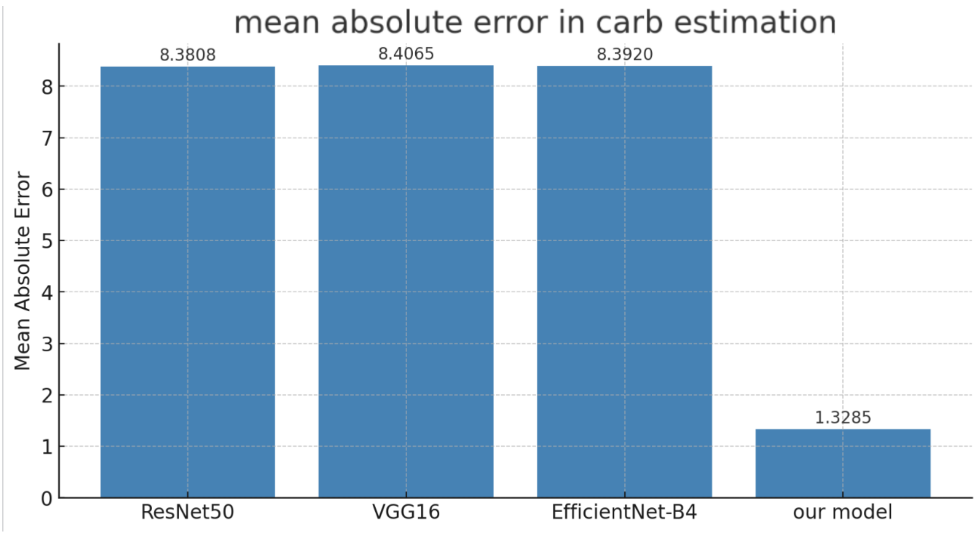


Fig 7: comparison between our model performance and others

As you can see, the error in our model is significantly less than existing CNNs, including what EfficientNetB4 (the backbone for the model) without our fine-tuning.

## (6) GitHub structure:

The attached GitHub repository contains the training notebook (.ipynb) used to build the model, the trained model, a single python file API/UI for Windows and executable build for Linux, other supporting files, required for a clean run of the model. More details in the README.md file inside the repository.

Link to GitHub Repository: https://github.com/HilaHorizon/Carb-Inspector

## (7) Future Works:

There are several directions in which we can extend the project, with the main focus on **introducing more data** to the model. One approach is to add **more databases** beyond Nutrition5K. This would **increase the number of classes** the model can recognize from the current 98 to a larger set, increasing the **diversity** of the label space. Currently, where the model encounters an image from an unknown category, it attempts to match it with one of the known labels, causing misclassification; more classes would resolve this issue. Moreover, adding more databases would **improve the model's generalization and robustness**, making it **more reliable** across different image qualities and resolutions, and ultimately boosting accuracy. Another approach is to make better use of Nutrition5K itself. Currently, the training is restricted to single-ingredient dishes, while the dataset also contains many **multi-ingredient dishes** (as discussed in section 2), that we do not use. Extending the model to handle these cases from Nutrition5K or other similar databases we would find, would make the use of the application more practical and user-friendly, as users wouldn't need to photograph each food item separately.

## (8) Bibliography:

- Tan M., Le Q.V., **EfficientNet**, Google Research, 2019. https://arxiv.org/pdf/1905.11946
- Thames Q. at al, **Nutrition5k**, Google Research, 2021. Kaggle images: https://www.kaggle.com/datasets/zygmuntyt/nutrition5k-dataset-side-angle-images Github repository with metadata: https://github.com/google-research-datasets/Nutrition5k Article: https://arxiv.org/pdf/2103.03375.