Project Report: Airflow Rate Prediction from IR Video and Delta T

1. Dataset Brief Details:

- **Objective:** To predict the airflow rate based on infrared (IR) video recordings and corresponding temperature difference measurements.
- **Target Variable (Label):** 4 distinct, discrete airflow rates (1.6V, 1.8V, 2.1V, 2.4V). This makes the problem a **multi-class classification task**.
- **Input Features:**
  - **Infrared (IR) Video Recordings:** Provided as .mat files containing sequences of thermal frames (key assumed 'TempFrames').
  - **Temperature Difference (ΔT):** A scalar value parsed from the .mat filename.
- **Dataset Size:** Very small. The current run processed **22 samples** in total. The distribution across classes observed in this run was slightly uneven: Class 1.6 (5 samples), 1.8 (5 samples), 2.1 (6 samples), 2.4 (6 samples). This small size and slight imbalance remain the primary challenges.
- **Structure:** Data organized into folders named according to the airflow rate (e.g., FanPower_1.6V).

2. Feature Explanation (Handcrafted Features):

A comprehensive set of **24 statistical features** were extracted from the raw IR video frames for each sample using the **extract_handcrafted_features** function. These aim to capture various aspects of the thermal behavior within the videos:

- **Basic Intensity Stats:** mean_temp, std_temp, max_temp, min_temp, median_temp, q25_temp, q75_temp(summarizing overall temperature levels and spread over time).
- **Distribution Shape (Frame Means):** skew_temp, kurt_temp (describing the shape of the distribution of average frame temperatures).
- **Intra-Frame Variation Stats:** mean_frame_std, std_frame_std, skew_frame_std, kurt_frame_std(describing the average amount and variability of temperature changes *within* individual frames).
- **Temporal Gradient Stats:** mean_temp_grad, std_temp_grad, max_abs_temp_grad, skew_temp_grad, kurt_temp_grad (describing the magnitude, variability, and distribution shape of temperature changes *between*consecutive frames).
- **Spatial Gradient Stats:** mean_spatial_grad, std_spatial_grad, max_spatial_grad, skew_spatial_grad, kurt_spatial_grad (describing the magnitude, variability, and distribution shape of temperature changes *across space* within individual frames).

Summary of All Features Extracted and Final Feature Count

The feature extraction function, **extract_handcrafted_features**, produces a set of statistical features derived from the IR video frames. These features can be grouped as follows:

- **Basic Statistical Features of Frame Intensities:**
  - **mean_temp:** The average pixel intensity (interpreted as temperature) across each frame, then averaged over all frames.
  - **std_temp:** The standard deviation of the pixel intensities across frames, indicating overall temperature variability.
  - **max_temp:** The maximum pixel intensity observed in any frame.
  - **min_temp:** The minimum pixel intensity observed in any frame.
  - **median_temp:** The median pixel intensity across frames.
  - **q25_temp:** The first quartile (25th percentile) value across frames.
  - **q75_temp:** The third quartile (75th percentile) value across frames.

- **Distribution Shape of Frame Means:**
  - **skew_temp:** The skewness of the distribution of per-frame mean intensities. This measures the asymmetry of the distribution.
  - **kurt_temp:** The kurtosis (using Fisher's definition, so a normal distribution has 0) of the distribution of per-frame means, which indicates how heavy-tailed or peaked the distribution is.

- **Intra-Frame Variation (Per-Frame Standard Deviation) Statistics:**
  - **mean_frame_std:** The average of the per-frame standard deviations (capturing how much variation exists within each frame).
  - **std_frame_std:** The standard deviation of the per-frame standard deviations over time.
  - **skew_frame_std:** The skewness of the distribution of per-frame standard deviations.
  - **kurt_frame_std:** The kurtosis of the distribution of per-frame standard deviations.

- **Temporal Gradient Statistics:**
  - **mean_temp_grad:** The average of the mean absolute differences (temporal gradients) calculated between consecutive frames.
  - **std_temp_grad:** The standard deviation of the temporal gradient values, representing how much the rate of change varies over time.
  - **max_abs_temp_grad:** The maximum absolute temporal gradient observed among consecutive frames.
  - **skew_temp_grad:** The skewness of the distribution of temporal gradients.

    - **kurt_temp_grad:** The kurtosis of the distribution of temporal gradients.

- **Spatial Gradient Statistics:**
    - **mean_spatial_grad:** The mean of the spatial gradient magnitudes calculated for each frame (details below).
    - **std_spatial_grad:** The standard deviation of these spatial gradient magnitudes over frames.
    - **max_spatial_grad:** The maximum spatial gradient magnitude observed among all frames.
    - **skew_spatial_grad:** The skewness of the distribution of the per-frame mean spatial gradients.
    - **kurt_spatial_grad:** The kurtosis of the distribution of the per-frame mean spatial gradients.

- **mean_spatial_grad** represents the average value of the spatial gradient magnitudes computed for each frame. The spatial gradient in an image quantifies how quickly pixel intensities change over space. High gradient values indicate sharp changes (edges or boundaries), while low values indicate smooth regions.

How It Was Extracted:

- For each frame:
    - **Gradient Computation:**
      The code uses np.gradient to compute gradients along the x-axis and y-axis separately.
    - **Gradient Magnitude Calculation:**
      The magnitude of the gradient at each pixel is computed as grad_mag = sqrt((gx)^2 + (gy)^2)
    - **Per-Frame Aggregation:**
      The average of all the gradient magnitudes in that frame is calculated. This gives one scalar value per frame—a single **mean spatial gradient** for that frame.
- **Temporal Aggregation:**
  Finally, the mean of these per-frame scalars is computed across all frames (using np.mean), resulting in one final scalar value for **mean_spatial_grad** per video sample.

Clarification:
Yes, for each frame we obtain one scalar representing the mean gradient (the average “edge strength”), and for instance, 150 frames, we would initially have 150 such scalar values. These 150 values are then averaged (or aggregated further by computing standard deviation, skewness, etc.) so that the final feature **mean_spatial_grad** becomes a single representative scalar value for that entire video sample.

— Code is written, but not included in this results —

- **Using CNN Features:** If using the approach with MobileNetV2 (after global average pooling), the base model outputs a feature vector of size 1280 per frame. After temporal averaging, this results in one 1280-dimensional vector per video. Combining this with delta_T gives a final feature vector of **1281 dimensions** per video. The CNN features *replace* the handcrafted statistical video features.

2.5 Pipeline Implementation:

To ensure consistency and reproducibility in data processing and modeling, I have implemented the scikit-learn Pipeline architecture. Feature extraction (whether handcrafted or CNN-based) is performed first to generate the input feature matrix X (including delta_T). This matrix is then fed into the pipeline for subsequent steps.

A typical pipeline configuration in this project includes:

1. **Imputation (SimpleImputer):** Handles any potential missing values (NaNs) that might arise during the feature extraction phase, ensuring the subsequent steps receive complete data. The 'mean' strategy is commonly used as a default.
2. **Scaling (StandardScaler):** Normalizes features by removing the mean and scaling to unit variance. This is crucial for distance-based algorithms like SVC and KNN, can improve the performance of models like Logistic Regression, and is essential for PCA.
3. **Dimensionality Reduction (Optional - PCA):** Principal Component Analysis can be included to reduce the number of features while retaining a specified amount of variance (e.g., n_components=0.95 to keep 95% of variance). This is particularly relevant when using high-dimensional CNN features to potentially mitigate the curse of dimensionality and reduce overfitting risk with the small dataset. This step can be easily included, excluded, or tuned via the configuration.
4. **Classifier (model):** The final step is the chosen classification algorithm (e.g., RandomForestClassifier, SVC).

The key benefit of this pipeline is that all steps (imputation, scaling, PCA) are 'fitted' only on the training data within each cross-validation fold and then used to 'transform' both the training and testing data for that fold. This prevents data leakage from the test set into the training process, leading to more reliable performance estimates.

This modular design also facilitates experimentation. Different classifiers can be easily swapped in as the final step of the pipeline without altering the preceding preprocessing stages. Furthermore, other steps, such as advanced feature selection techniques like RFECV (Recursive Feature Elimination with

Cross-Validation), could potentially be integrated into this pipeline in future work to automatically select the most relevant features before classification.

3. Results and Graph Interpretation:

The evaluation focuses on assessing the performance of various classification models in distinguishing between the four distinct airflow rates. Given the very small dataset size (N=20), a robust cross-validation strategy is essential. The primary method employed is **Leave-One-Out Cross-Validation (LOOCV)**, where the model is trained on 19 samples and tested on the remaining 1 sample, repeating this process 20 times so each sample serves as the test set exactly once.
Alternatively, **Stratified K-Fold Cross-Validation** (with K typically limited to 3-5 due to small class sizes) can be used if LOOCV proves computationally intensive or exhibits high variance, ensuring class proportions are maintained in each fold. (But this can only be done on a larger dataset)
Several standard classification algorithms available in **scikit-learn** are evaluated within this cross-validation framework, including:

- RandomForestClassifier
- GradientBoostingClassifier
- SVC (Support Vector Classifier)
- LogisticRegression (often included as a simpler baseline)

Performance is measured using standard multi-class classification metrics calculated across all LOOCV folds:

- **Accuracy:** The overall percentage of correct predictions.
- **Weighted F1-Score:** The primary metric used for comparing models overall and selecting the "best" model. It calculates the F1-score (harmonic mean of precision and recall) for each class and averages them, weighted by the number of true instances for each class (support). This provides a balanced measure, especially important if performance across classes is uneven.
- **Confusion Matrix:** Visualized to show specific patterns of misclassification between the different airflow rates.
- **Confusion Matrix (and Plot):**
    - **Rows:** Represent the *actual* airflow rates.
    - **Columns:** Represent the *predicted* airflow rates.
    - **Diagonal Elements:** Show the number of samples correctly classified for each class. Higher numbers on the diagonal are better.
    - **Off-Diagonal Elements:** Show misclassifications. For instance, the number in row '1.6V', column '1.8V' indicates how many times a video with an actual rate of 1.6V was incorrectly predicted as 1.8V.
    - **Interpretation:** Look for high numbers on the diagonal and low numbers off-diagonal. Identify which specific classes are

frequently confused with each other (high off-diagonal values). This tells you where the model struggles.

- **Classification Report:** Provides a detailed breakdown per class:
  - **Precision:** For a predicted class (e.g., 1.8V), what proportion of those predictions were actually correct? (High precision means fewer false positives).
  - **Recall:** For an actual class (e.g., 1.8V), what proportion did the model correctly identify? (High recall means fewer false negatives).
  - **F1-Score:** The harmonic mean of precision and recall, providing a single balanced score for each class's performance.
  - **Support:** How many actual samples belong to each class (should be 5 for each in your case initially).
  - **Weighted Avg F1-Score:** Used to select the "best" model overall in the script. It averages the per-class F1-scores, weighted by the number of samples in each class.

3.1 Hyperparameter Tuning Summary

GridSearch identified the following best parameters and corresponding cross-validated weighted F1-scores during the tuning phase:

- **SVC:** Best Score = **0.6818**, Best Params = {'model__C': 5.0, 'model__kernel': 'linear'}
- **LogisticRegression:** Best Score = 0.4545, Best Params = {} (default parameters used as grid was empty)
- **GradientBoosting:** Best Score = 0.4091, Best Params = {'model__learning_rate': 0.01, 'model__max_depth': 3, 'model__n_estimators': 50}
- **RandomForest:** Best Score = 0.3636, Best Params = {'model__max_depth': 3, 'model__min_samples_split': 2, 'model__n_estimators': 50}

SVC with a linear kernel showed the most promise during the tuning phase.

3.2 Final Model Performance (Post-Tuning using LOOCV):

The following results were obtained by running LOOCV again, applying the best hyperparameters found above for each model:

- **Best Performer: SVC (Tuned: C=5.0, kernel='linear')**
  - Accuracy: **0.6818**
  - F1 Score (Weighted): **0.6801**
  - Classification Report: precision recall f1-score support
  - 1.6 0.60 0.60 0.60 5
  - 1.8 0.50 0.40 0.44 5
  - 2.1 0.62 0.83 0.71 6
  - 2.4 1.00 0.83 0.91 6

        -
        - Confusion Matrix:[[3 1 1 0]
        - [2 2 1 0]
        - [0 1 5 0]
        - [0 0 1 5]]

    - **LogisticRegression (Default Params)**
        - Accuracy: 0.4545
        - F1 Score (Weighted): 0.4492
    - **GradientBoosting (Tuned)**
        - Accuracy: 0.4091
        - F1 Score (Weighted): 0.3815
    - **RandomForest (Tuned)**
        - Accuracy: 0.3636
        - F1 Score (Weighted): 0.3636

3.3 Interpretation:

- The **SVC model with a linear kernel (C=5.0)** provided the best performance among the tested models on this dataset with handcrafted features, achieving approximately 68% accuracy and weighted F1-score in the LOOCV evaluation.
- **Analysis of Best Model (SVC):**
    - The model performs very well on the highest airflow rate (2.4V), achieving perfect precision and high recall.
    - It also performs reasonably well on the 1.6V and 2.1V classes.
    - The main difficulty lies in correctly identifying the 1.8V airflow rate (only 40% recall), which it frequently misclassifies as 1.6V (2 instances) or 2.1V (1 instance). There is also some confusion between 1.6V and 1.8V/2.1V.
- The performance of the tree-based methods (RandomForest, GradientBoosting) was notably lower, even after tuning, suggesting that linear separation (as used by the best SVC) might be more effective in the feature space created by these specific handcrafted features, or that the tree models overfit more easily on this small dataset.
- Overall, while SVC shows promise, the ~68% performance indicates significant room for improvement, likely limited by the small dataset size and potential ambiguity between adjacent airflow rates in the feature space.

4. Further Steps and Recommendations to Increase Accuracy and Reliability:

The primary limitation is the N=20 dataset size. Addressing this will yield the biggest potential gains.

1. **Acquire More Data:** This is the single most impactful recommendation. Even increasing to 10-20 samples per class would provide the models with more examples to learn from, improving robustness and generalization.
2. **Data Augmentation:** Implement realistic augmentations for IR video data *before* feature extraction within the training folds of your cross-validation. Examples:
    - Slight random rotations, shifts, zooms.
    - Random horizontal flips (if physically meaningful for your setup).
    - Small adjustments to brightness/contrast (simulating sensor variations).
    - Adding minor thermal noise. This artificially increases training data diversity.
3. **Feature Engineering Comparison:**
    - Systematically compare the performance using **handcrafted features** vs. **CNN features**. With N=20, simpler handcrafted features might sometimes generalize better than high-dimensional CNN features if the latter lead to overfitting the training folds.
    - If using CNNs, try different **pre-trained bases** (ResNet, EfficientNet) or **temporal aggregation** methods (max pooling over time, simple LSTM/GRU on frame features - use very cautiously due to small N).
4. **Dimensionality Reduction/Feature Selection:**
    - If using high-dimensional features (especially CNN), experiment with **PCA** settings (e.g., different n_components values) or explore other techniques like SelectKBest within the pipeline to see if reducing dimensionality helps combat overfitting.
5. **Model Tuning and Selection:**
    - Perform **hyperparameter tuning** (e.g., using GridSearchCV or RandomizedSearchCV with cv=LeaveOneOut() or Stratified-K-Fold(n_splits=...)) for the classifiers (RF: n_estimators, max_depth; SVC: C, gamma; GB: n_estimators, learning_rate, max_depth). *Be mindful* that extensive tuning on N=20 can overfit the cross-validation scores themselves. Keep model complexity relatively constrained.
    - Can include **simpler classifiers** like LogisticRegression in model comparison as a baseline.
6. **Ensemble Methods:** Based on more results can consider to combine the predictions (e.g., averaging probabilities) of the top 2-3 *different* well-performing models (e.g., RF and SVC) identified during cross-validation.

Hyperparameter Tuning (Future Work):
While the initial analysis focused on establishing baseline performance with default or common hyperparameters, optimizing model performance often requires tuning these parameters. A framework for this using scikit-learn's GridSearchCV has been considered, although not extensively applied in the baseline runs presented here due to the extreme data limitation (tuning on N=20 can easily overfit the *cross-validation performance* itself). GridSearchCV systematically works through multiple hyperparameter combinations, using cross-validation (again, likely LOOCV or Stratified K-Fold in this context) to evaluate each combination's performance on unseen data within the folds. This allows for finding the optimal settings for parameters like:

- n_estimators, max_depth for Random Forest and Gradient Boosting.
- C, gamma, kernel for SVC.
- Regularization strength (C) for Logistic Regression.
- Potentially n_components for PCA if treated as a tunable parameter.
- Settings for feature selection methods like RFECV if integrated.

This tuning process, planned for future iterations (especially if more data becomes available), will be critical for maximizing the predictive accuracy and reliability of the models, particularly when dealing with potentially complex feature interactions arising from CNN features or when incorporating automated feature selection.