Weather Image Classification

Aaron Bergfeld, Daniel Costa, Bella Davies, Theo Hui
MIDS w207, Spring 2025

Motivation

Research Question: How accurately can a ML model recognize weather conditions from images?

Use Case: Autonomous vehicles could adjust their driving behavior based on incremental snapshots of weather conditions.

Goals:

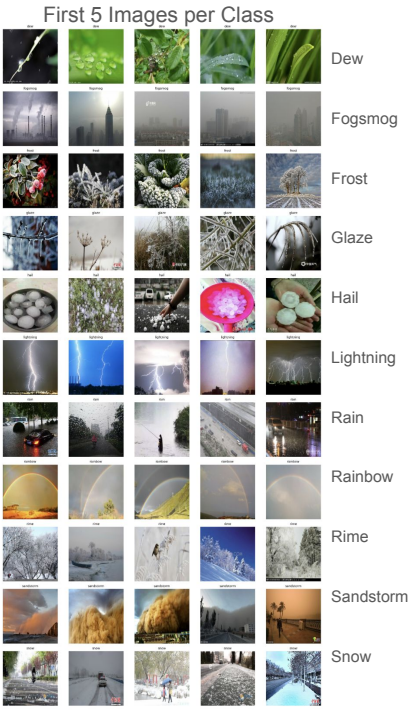
- Image preprocessing for CNNs
- Dimensionality reduction
- Experimentation with new models

What sort of data preprocessing techniques and models will aid in weather classification?

Data

Weather Image Dataset:
https://www.kaggle.com/datasets/jehanbhathena/weather-dataset

- 6862 images belonging to 11 classes
  - Dew
  - Fogsmog
  - Frost
  - Glaze
  - Hail
  - Lightning
  - Rain
  - Rainbow
  - Rime
  - Sandstorm
  - Snow
- 80% Train: 5489 images
- 10% Validation: 686 images
- 10% Test: 687 images



11 classes of almost 7000 images. Varied types of images, sweeping shots vs close ups

Modeling

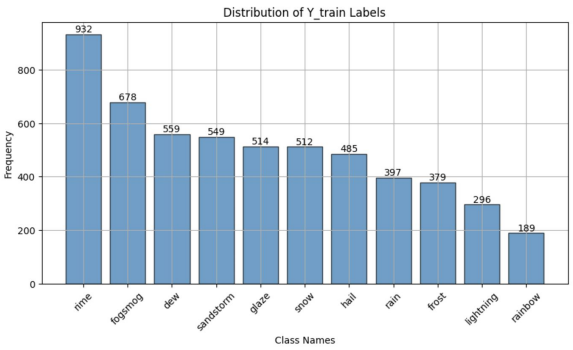
Model	Val Loss	Val Accuracy	*Val F1 Score	Val AUC
1. Baseline Log Reg	**1.50**	**0.51**	**0.46**	**0.88**
2. Log Reg w/ Oversampling	1.48	0.56	0.52	0.88
3. K-Means → CNN	3.85	0.60	0.58	0.93
4. PCA → CNN	1.49	0.59	0.58	0.93
5. Stacked 5+6 → Simple Log Reg	1.00	0.67	0.66	0.82
6. Stacked 5+6 → CNN Tuned	1.01	0.69	0.68	0.94
7. CNN	1.02	0.71	0.73	0.96
*8. CNN Tuned	0.8	0.82	0.83	0.98
*9. Resnet50 + Simple Log Reg	0.45	0.91	0.91	0.99
*10. Resnet50 + CNN Tuned	0.41	0.90	0.90	0.99
*11. SAM + Resnet50 + Log Reg	0.44	0.89	0.90	0.99

Experiments

1. Class Distribution Balancing: Oversampling
2. Data Augmentation
3. Hyperparameter Tuning: Keras Hyperband
4. Dimensionality Reduction: K-means Color Quantization
5. Dimensionality Reduction: Principal Component Analysis
6. Ensemble Learning: Combine Model Predictions for Stacking
7. Transfer Learning: ResNet-50 Feature Extraction
8. Image Segmentation: SAM (Segment Anything Model)

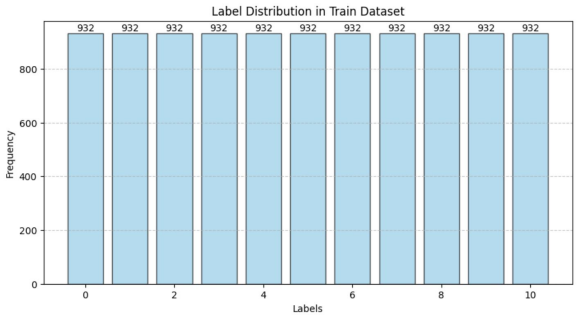
1. Class Distribution Balancing: Oversampling

Before Oversampling:



- X_train.shape = (5490, 224, 224, 3)

After Oversampling:



- X_train.shape = (10252, 224, 224, 3)

Improvement from Baseline Val Accuracy: 5%

we addressed class imbalance by **copying the minority class samples** until they matched the size of the majority class.

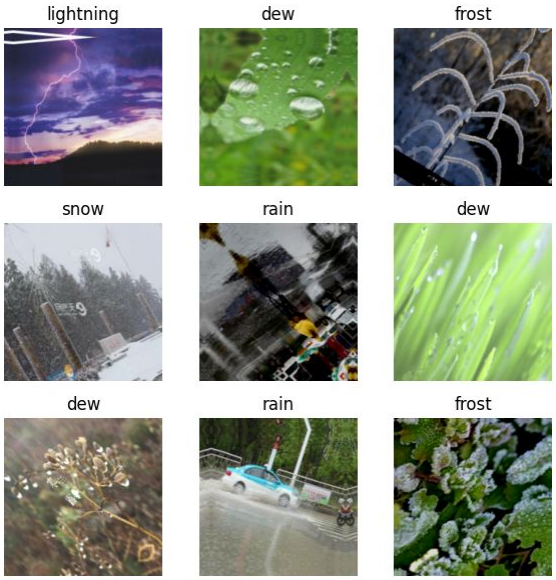
Why did we need to balance the classes?

- **Imbalanced datasets** can lead to biased model performance, where the model becomes biased toward predicting the majority class because it has more examples to learn from.

- By balancing the classes, we ensure the model has **equal exposure** to both classes, leading to better generalization and fairness in predictions. This helps improve the model's ability to correctly identify instances of the minority class.

2. Data Augmentation

- Random Flip: "horizontal_and_vertical"
- Random Rotation: 0.2
- Random Zoom: 0.2
- Random Contrast: 0.2
- Random Brightness: 0.2

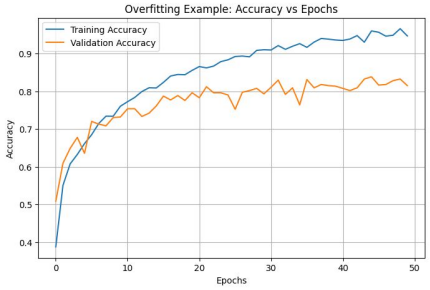


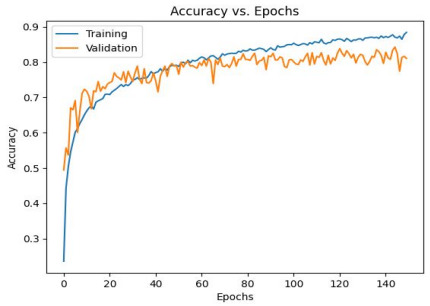
- We used several augmentation techniques to increase data diversity and improve model robustness:
    - **Random Flip (horizontal & vertical)**: Helps the model become invariant to object orientation.
    - **Random Rotation (0.2)**: Simulates rotations, allowing the model to handle varying object angles.
    - **Random Zoom (0.2)**: Adds scale variation, improving the model's ability to recognize objects at different distances.
    - **Random Contrast (0.2)**: Adjusts contrast to simulate different lighting conditions.
    - **Random Brightness (0.2)**: Alters brightness to help the model adapt to various lighting environments.

3. Hyperparameter Tuning: Keras Hyperband

CNN Tuned Parameters:

- conv_1 # of filters: 16
- conv_1 kernel size: 7x7
- conv_2 # of filters: 128
- conv_2 kernel size: 3x3
- conv_3 # of filters: 48
- conv_3 kernel size: 3x3
- dropout: 0.1
- # of dense units: 192
- L2 regularization: 0.001
- Learning_rate: 0.001





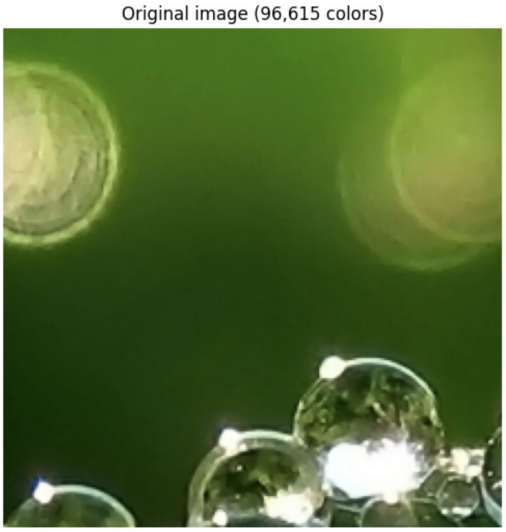
Our first attempt to improve on the **softmax regression baseline** was to implement a **Convolutional Neural Network (CNN)**, which allowed us to capture more complex patterns in the data.

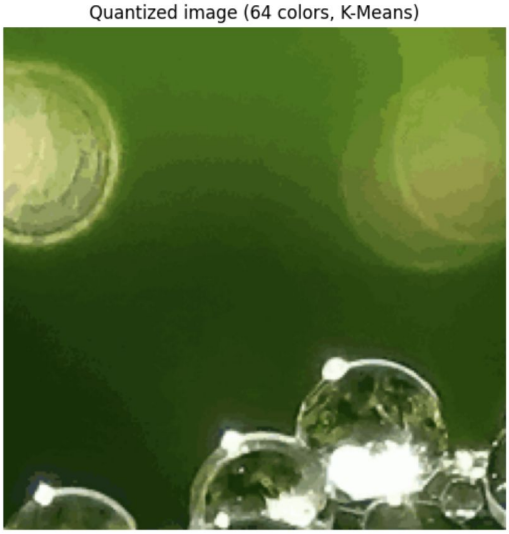
To optimize the model, we used the **Keras Hyperband tuner**. This helped us efficiently search through a wide range of hyperparameters and make better architecture decisions, saving both time and computational resources.

A key finding from the tuning process was that incorporating **dropout** and **L2 regularization** significantly reduced **overfitting**. This led to **slightly better validation accuracy**, although it came at the cost of **increased training time** due to the additional regularization steps.

Overall, the pure CNN model achieved an f1 score of 83%, a 37% improvement over our baseline model.

4. Dimensionality Reduction: K-Means Color Quantization



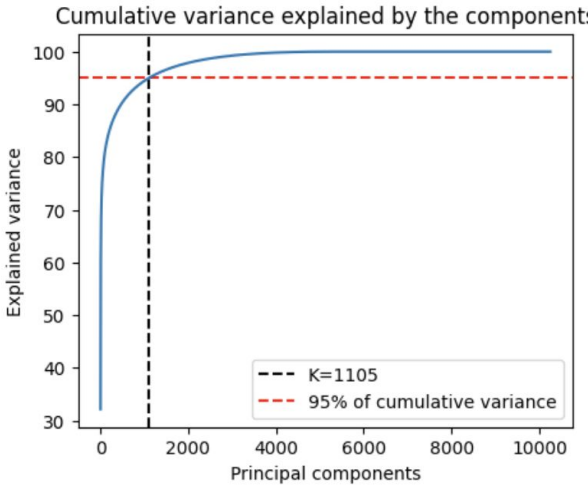


Bella

Next, we wanted to try a couple different approaches to dimensionality reduction, to feed a reduced dataset to a CNN. Reducing the dimension of image will allow us to capture the majority of the information from the image to feed into models. This can help the model to learn the more important information from a noisy dataset..

1. **KMeans Clustering**
    - Choose k = number of colors wanted
    - Algorithm finds k centroids in color space and assigns each pixel to nearest centroid, which represent the new reduced colors.
2. **Color Mapping**
    - Replace each pixel's RGB values with those of nearest centroid to create images with only k colors

5. Dimensionality Reduction: Principal Component Analysis



Choosing k = 1200:	X_train.shape
Before PCA:	(10252, 224, 224, 3)
Flatten data for PCA:	(10252, 150528)
After PCA:	(10252,**1200**)
Reshape data for CNN:	(10252, **20, 20, 3**)

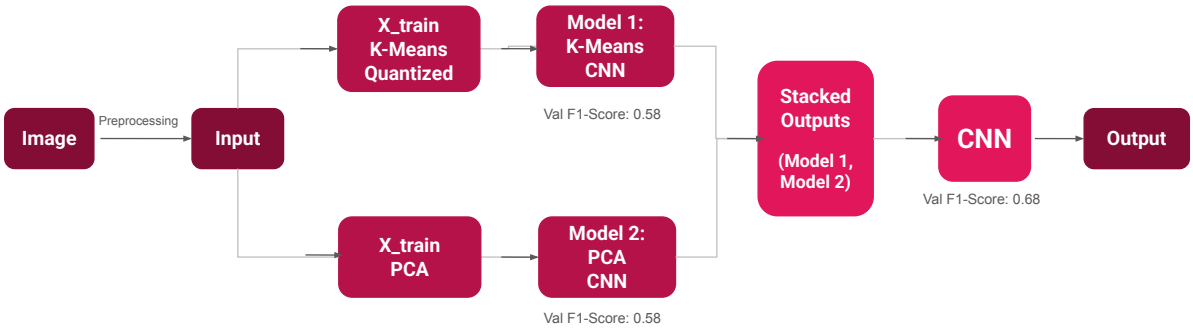
Bella

Another approach to dimensionality reduction that we tried was PCA, which reduces the size or dimension of data by selecting a specified top k principal components which cover a majority of the variance in a dataset.

1. **Data Preparation**
   - Flatten image matrix into 1D vector
2. **Covariance Matrix**
   - PCA computes covariance matrix of the data which shows the relationships between pixel values
3. **Eigenvalues & eigenvectors**
   - Then using the covariance matrix to find eigenvectors, which are principal components or directions of max variance, and eigenvalues which are the amount of variance in each direction
4. **Dimensionality Reduction**
   - Sort eigenvectors by eigenvalues (descending). Image shows 1105 for k, but this does not project easily to 3 dimensional space for RGB, so select top 1200 eigenvectors/principal components dimensions to project the data onto this new lower-dimensional space which captures slightly over 95% of cumulative variance. After pca, we reshape the images back to 3-dimensional → (20,20,3)

Improved speed of training.

6. Ensemble Learning: Stacking Model Outputs

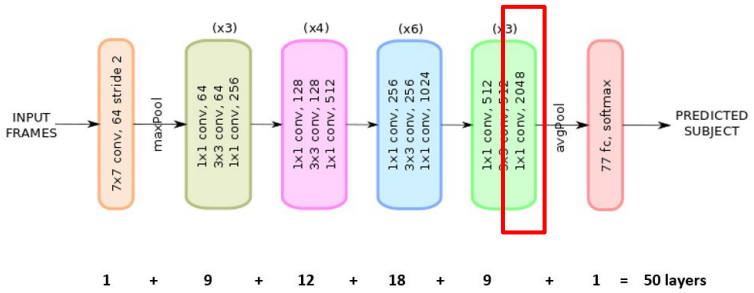


Bella

Then using the 2 CNNs that learned from the kmeans reduced dataset and the pca reduced dataset, we wanted to try ensemble learning to introduce a third model which will learn from the predictions of both weaker models to improve upon their performance. This looks at the probabilities outputted from both models for each image, then decides the most likely outcome from the two probabilities. This resulted in a 10% improvement in f1-score from the k-means cnn and pca-cnn. We used f1 since its a balanced representation of precision and recall.

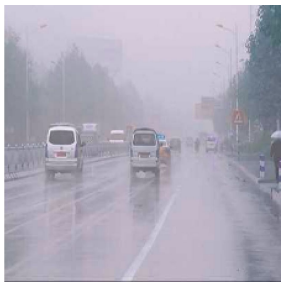
7. Transfer Learning: ResNet-50 Feature Extraction

- Much larger than anything we could field
- 25.6 million parameters, 50 convolutional layers, 1.3 million training set,1000 result categories
- Convert from (224, 224) to (2048,) vector



Theo

SAM (Segment Anything Model)



Original Image

Segmented Image

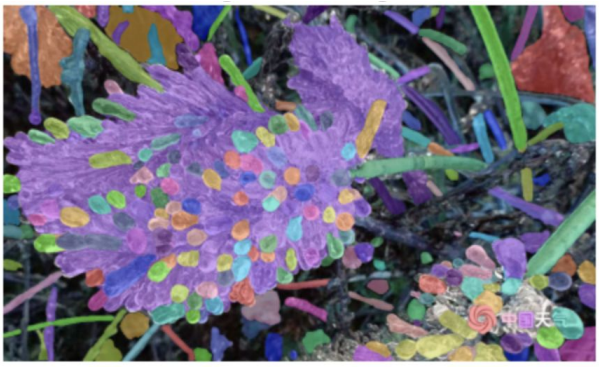
One thing we noticed throughout training our models is that many images contained background noise which could hurt the performance of our classifiers. To see if we could improve upon the performance of our fine-tuned Resnet50 classifier, we used Meta's Segment Anything Model (or SAM) to partition images into segments in order to de-noise the image data.

Here you can see two examples of how SAM detects segments in an image, with different segments highlighted by random colors.

Computational Constraints

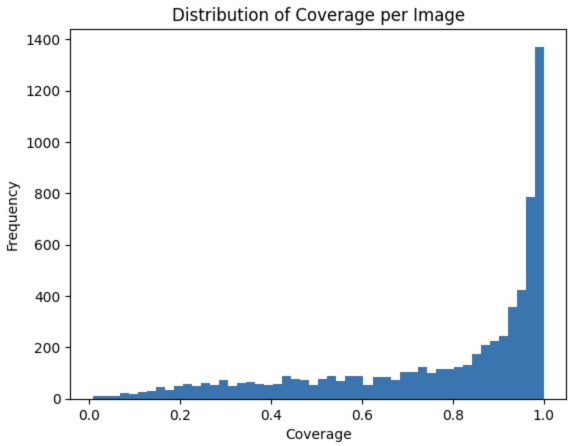
Resized Image: 28% Segmented

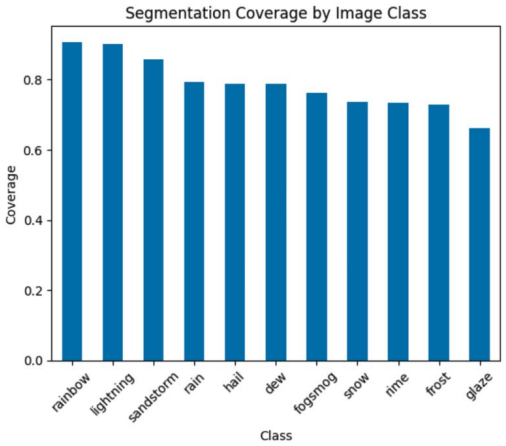
Original Image: 65% Segmented



Unfortunately, due to the computational load of SAM's 636 million parameters, we had to shrink the size of our photos to the Resnet input shape of 224 by 224 pixels before segmenting, which caused some segments to lose detail. For example you can see that the ice glazing on the leaves in the image on the left were not detected by SAM when shrunken down, but on the right, you can see that this glazing in violet is detected by SAM in the original unedited photo.

Segmentation Coverage



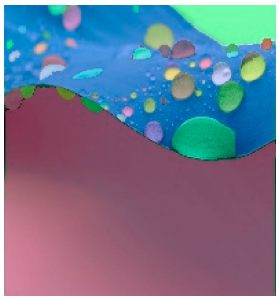


Depending on the granularity of detail in the photo, SAM was able to segment some images better than others. On the left, you can see the distribution in proportion of each image that SAM was able to segment.

This metric also varied by class, with image classes with more background focus like rainbow, lightning, and sandstorm performing better than image classes that have more detailed foregrounds like glaze, frost, and rime. Thus, due to the constraints of SAM's performance on the resized images, SAM's segmentation was not applied to all classes.

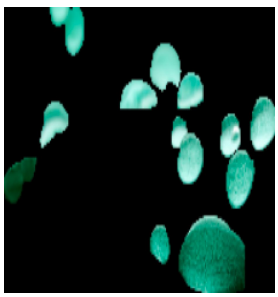
Preprocessing Pipeline

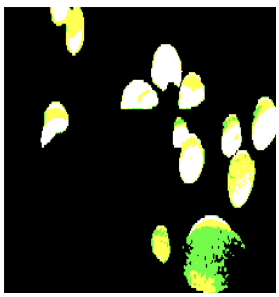
1
Resized
Image



2
Segmented
Image

3
De-Noised
Image

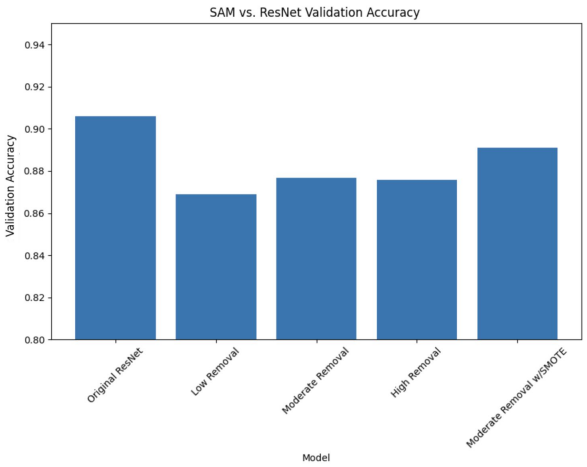


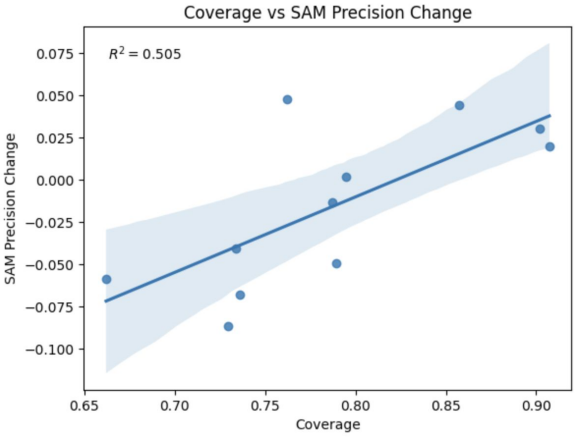


4
ResNet
Normalized
Input

Here is how each image is processed. First, we resized the original image. Second, we input this image to SAM to get a list of segment masks. Third, we de-noised the image by tuning hyperparameters such as the number of large and small segments to remove, then, cropped the image to the composite bounding box of the remaining masks. The last step was to normalize the input's pixel values to the range acceptable by Resnet's feature extractor.

Evaluation





We ran this process with 3 different sets of mask removal hyperparameters corresponding to removing a different proportion of masks from each image class. The model that performed the best was the one in which a moderate proportion of masks were removed. Using SMOTE to balance classes on the moderate model improved its accuracy by nearly 2%. This model slightly underperformed the original Resnet model overall, but there were some interesting findings when analyzing class-specific metrics. In the figure on the right, you can see that the better SAM was able to segment an image class, the better it was able to improve our classifier's precision. This suggests that running this experiment again on images in their original format instead of in their shrunken forms may lead to better accuracy results.

Modeling

Model	Val Loss	Val Accuracy	***Val F1 Score**	Val AUC
1. Baseline Log Reg	1.50	0.51	**0.46**	0.88
2. Log Reg w/ Oversampling	1.48	0.56	**0.52**	0.88
3. K-Means → CNN	3.85	0.60	**0.58**	0.93
4. PCA → CNN	1.49	0.59	**0.58**	0.93
5. Stacked 5+6 → Simple Log Reg	1.00	0.67	**0.66**	0.82
6. Stacked 5+6 → CNN Tuned	1.01	0.69	**0.68**	0.94
7. CNN	1.02	0.71	**0.73**	0.96
***8. CNN Tuned**	**0.8**	**0.82**	**0.83**	**0.98**
***9. Resnet50 + Simple Log Reg**	**0.45**	**0.91**	**0.91**	**0.99**
***10. Resnet50 + CNN Tuned**	**0.41**	**0.90**	**0.90**	**0.99**
***11. SAM + Resnet50 + Log Reg**	**0.44**	**0.89**	**0.90**	**0.99**

Final Model Evaluation Results

Test loss: .44

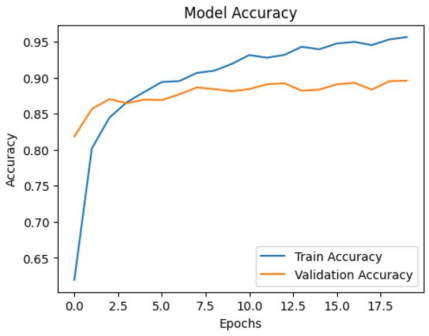
Test accuracy: .90

Test f1: .90

Test auc: .99

Classification Report:

	precision	recall	f1-score	support
dew	1.00	0.95	0.97	129
fogsmog	0.90	0.96	0.93	182
frost	0.89	0.73	0.80	100
glaze	0.77	0.88	0.82	144
hail	0.95	0.97	0.96	129
lightning	0.98	0.97	0.98	63
rain	0.82	0.93	0.87	95
rainbow	0.96	0.98	0.97	47
rime	0.87	0.87	0.87	217
sandstorm	0.95	0.90	0.92	136
snow	0.88	0.79	0.83	130
accuracy			0.90	1372
macro avg	0.91	0.90	0.90	1372
weighted avg	0.90	0.90	0.90	1372



Theo

Conclusions

Improvement from baseline: 44%

Best model: Resnet50 + Simple Log Reg

Most interpretable best model: SAM + Resnet50 + Log Reg

*9. Resnet50 + Simple Log Reg	0.45	0.91	0.91	0.99
*10. Resnet50 + CNN Tuned	0.41	0.90	0.90	0.99
*11. SAM + Resnet50 + Log Reg	0.44	0.89	0.90	0.99

Trade off of computation time vs. human interpretability

Home-baking your own model is generally not as effective


Thank You!
Any Questions?

Git/Contributions

Git Link: https://github.com/djcosta2/207_Final_Project

Aaron
- Class Distribution Balancing: Oversampling
- Data Augmentation
- CNN Hyperparameter Tuning: Keras Hyperband

Bella
- Baseline Logistic Regression, & Logistic Regression with Oversampling
- Dimensionality Reduction: K-Means Color Quantization, Principal Component Analysis with CNN
- Ensemble Learning: Stack Model Predictions from K-Means and PCA for Simple Log Reg, then CNN

Daniel
- Image Segmentation Analysis
- SAM (Segment Anything Model) with ResNet
- Class balancing with SMOTE

Theo
- Data Loading
- CNN
- Transfer Learning: ResNet-50 Feature Extraction with CNN