

# Weather Pictures Classification Using CNNs and PCA (For Dimension Reduction)

By: Jian Hui Mai

## Table of Contents

Abstract Summaries-----	3
Objective of Project-----	3
Tools and Approach-----	3
Results and Outcomes -----	3
Description of Work Accomplished-----	5
Preprocessing -----	5
Normalization of Images and Resizing -----	5
Data Augmentation -----	5
Principal Component Analysis (PCA) -----	6
Convolved Neural Networks (CNN) -----	7
Original Model -----	8
Second Model with Two Convolution Layers -----	10
Third Model with Two Convolution Layers and 20% Dropout -----	11
Final Model -----	12
Descriptions on How to Run and Compile Code -----	14
Preprocessing -----	14
PCA -----	15
CNN -----	15

## Abstract Summaries

A dataset containing images of dew, fog/smog, frost, glaze, hail, lightning, rain, rainbows, rime, sandstorm, and snow was downloaded from [Kaggle](#). The dataset contained images of different quality, sizes, and dimensions. In total, there are 6,916 pictures. Due to the variations in size and color intensity, all pictures were normalized and resized to 100 x 100. Additionally, since the dataset only contained 6,916 pictures, I used data augmentation to generate more pictures. All the pictures were then processed with Principal Component Analysis (PCA) for compression. The compressed images were then trained using Convolutional Neural Networks to classify the type of weather displayed in the images. More details regarding the preprocessing and training processes are provided later in this paper.

## Objective of Project

The objective of this project is to predict the weather given an image. In other words, when a computer is provided an image, it can say that the image is dew or any of the other categories.

## Tools and Approach

The tools and approaches used in this project include Keras, OS, glob, cv2, Sklearn, NumPy, Matplotlib, and PIL. All these tools/libraries are written in Python and within Jupyter Notebook. Additionally, all these tools were used either for preprocessing or training the PCA and CNN models.

## Results and Outcomes

As previously stated in my abstract, I created PCA and CNN models. Since PCA is an unsupervised learning algorithm, I chose a variance of 95% since that is usually the variance we look for. Further details about the variance will be provided later. Additionally, I created four

CNN models that I tuned to achieve the best accuracy. The original model had an 56% accuracy on both the validation and test data. With tuning and the addition of new layers, I was able to increase the accuracy to 85% on the validation data and test data. Further details about the tuning will be provided later in the paper.

## Description of Work Accomplished

### Preprocessing

For preprocessing, I normalized the images and created new images through data augmentation.

### Normalization of Images and Resizing

As previously stated, the images were originally of different dimensions. In order to train the images using CNNs, all images must be the same size. With this in mind, I resized all images to 100 x 100. Additionally, I normalized the pixels within the images which restricted the range of pixel intensity values. This would cause the most intense and least intense pixel values to be restricted. The normalization and resizing of images were accomplished by the cv2 library in Python. I looped through each folder containing the images to apply normalizing and resizing and replaced the original images. I would also delete any image containing errors or corrupted.



Original Image



Resized Image

The above shows a sample of an original image and the resized image. You can see that the resized image looks like it has been “squeezed” to accommodate the reduction in size.

### Data Augmentation

For the last step of my image preprocessing, I created five new images for every one found. Since the original dataset contained only 6,916 images, it would be a good idea to create more to increase the accuracy of our models. The new images were created using the technique

called data augmentation. Data augmentation works by rotating the original images within a certain range to create a new image. In my implementation using the Keras image preprocessing library, I set the rotation range to be thirty degrees with empty pixels filled in by those nearby. All images were first converted into a three-dimensional NumPy array containing the red, green, blue values and then converted back to an image. Additionally, images created by this method would be appended with the name “generated”.



Original (Resized)

Rotated

The pictures above show an example of an original (resized) and rotated image. You can see that the rotated image appears less “slanted” compared to the original one. After data augmentation, I was able to increase the dataset size to 40,496 images.

### Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a linear dimension reduction algorithm that utilizes Singular Value Decomposition. PCA attempts to reduce the dimensions of images/data while keeping as much quality as possible. Since the dimensions of images are rather large and humans cannot visualize images beyond 3 dimensions, I thought it would be a good idea to apply PCA. Additionally, PCA could help speed up the training of Convolutional Neural Networks. Since PCA is an unsupervised learning algorithm, there is no direct metric to measure the quality of the output. As previously stated, PCA attempts to “keep as much of the quality as possible” With this in mind, Sklearn allows the tweaking of the `n_components` parameters which refers to the “number of components to keep” Usually, we would like to keep 95% of the variance. We can

determine the number of components to achieve 95% variance manually or tell Sklearn to set the variance to 95%. I chose the latter method. I implemented PCA by first converting each image to a red, green, and blue (RGB) array using cv2. Each color will then be transformed and inverted to extract 95% variance using Sklearn. The transformed and inverted image array will then be merged to create a new image. Any image that could not be combined or transformed and inverted would be deleted. The final image would be saved into a new folder called PCA\_dataset using the same file structure as the original dataset folder.



Original (Resized)



PCA

Above represents two images, the one on the left is the original resized image and the one on the right represents the image with PCA applied. The image on the right appears blurry and fuzzy compared to the left.

### Convolved Neural Networks (CNN)

To classify each type of image, I utilized CNNs. The CNNs were trained on the dataset with PCA applied. I utilized the ImageDataGenerator from Keras to extract the labels for each image. The labels are the folder name that each image is contained in. In other words, the pictures were grouped in their own folders, and each folder corresponds to specific weather.

ImageDataGenerator assigns the labels based on numerical order. For example, dew would be zero, fog/smog would be one, and so on. These labels would become the y of our array.

Additionally, x would be retrieved from each image. The loop would go through each folder and image, reading it using the cv2 library and creating a NumPy array. Each row of the x and y arrays would consist of an image. These x and y arrays would then be randomly shuffled

otherwise all the testing sets would be images from the last couple of folders. After shuffling, I created a train and test split of 80/20. I would use these test and training sets to create four models.

### **Original Model**

The structure of my original model (implemented in Keras) is a convolution layer, pooling layer, flatten layer, and two dense layers. The convolution layer contains thirty-two filters with a size of 3x3 using the RELU activation function. It takes the NumPy array containing the images as input. Additionally, each convolution layer requires a pooling layer for robustness and speed which I created of size 2x2 and type of Max Pooling. I chose the RELU activation function to mitigate the vanishing gradient problem. The vanishing gradient problem occurs when the networks are unable to update the weights near the input of the model which forces the network to converge to a poor solution. Convolution layers are required for feature extraction of images. After the convolution and max-pooling layer, I added a flatten layer to convert the 2D arrays into a single continuous linear vector. After flattening, I added the first dense layer with the activation function of RELU and 256 units. It retrieves all the neurons from the previous flattened layer. The second dense layer is then added with a Softmax activation function and eleven units. I used the Softmax activation because we are working with multiclass classification. Additionally, there are eleven classes resulting in eleven units.



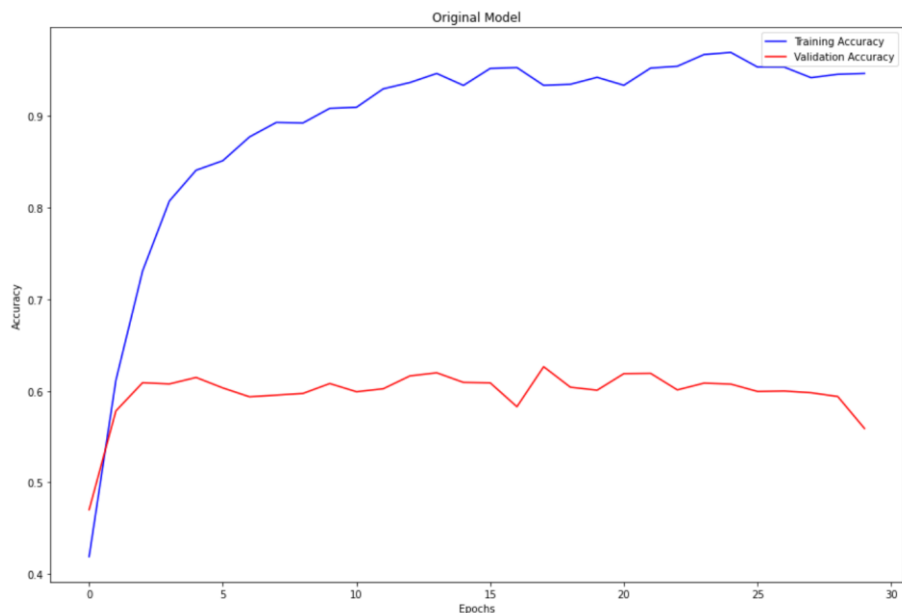
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
flatten (Flatten)	(None, 76832)	0
dense (Dense)	(None, 256)	19669248
dense_1 (Dense)	(None, 11)	2827

=====  
Total params: 19,672,971  
Trainable params: 19,672,971  
Non-trainable params: 0  
=====

Pictorial representation of the original model structure

I then compiled the model with the Adam optimizer and loss of “SparseCategoricalCrossentropy” (because our classes are represented by numerical values and not one-hot encoded). I then fit the model with a batch size of 128, a validation split of 20%, and 30 epochs. The validation split is used to tune the later models. Additionally, each epoch represents the number of times the full training set is used to train the model. I created a line graph representing the training and validation accuracy for each epoch below.



The graph appears to show that this model is overfitting since the validation accuracy appears to hover around 60% accuracy after 10 epochs while the training accuracy increases. The final validation and testing accuracy is 56%.

## Second Model with Two Convolution Layers

Since my original model overfitted, I decided to add another convolution layer after the first convolution layer. The second convolution layer contains sixty-four filters with a size of 4x4 and the activation function of RELU. After the convolution layer, I also included a max-pooling layer of 2x2 to ensure robustness and speed.

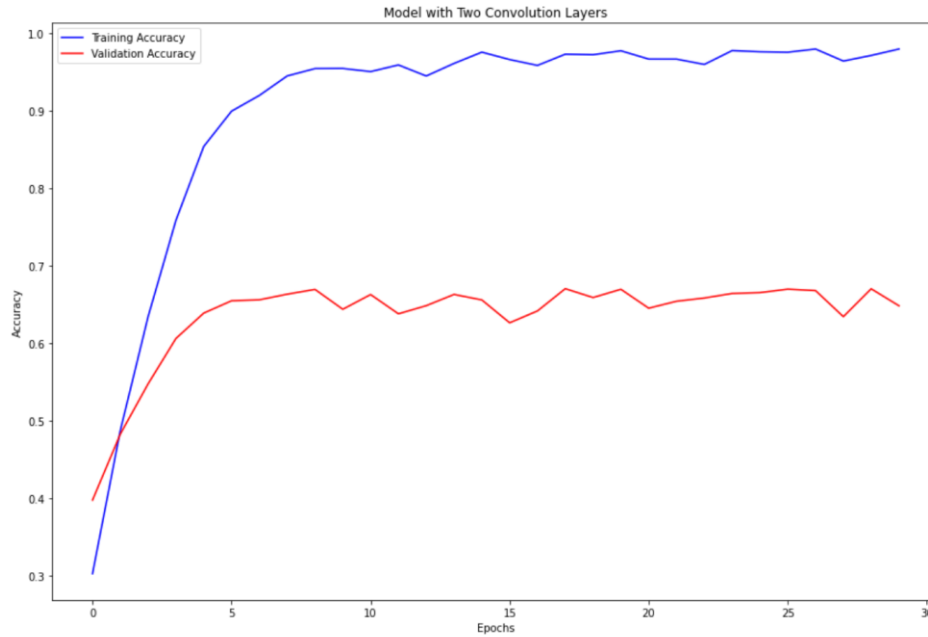
Model: "sequential\_2"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d_3 (MaxPooling 2D)	(None, 49, 49, 32)	0
conv2d_4 (Conv2D)	(None, 46, 46, 64)	32832
max_pooling2d_4 (MaxPooling 2D)	(None, 23, 23, 64)	0
flatten_2 (Flatten)	(None, 33856)	0
dense_4 (Dense)	(None, 256)	8667392
dense_5 (Dense)	(None, 11)	2827

=====  
Total params: 8,703,947  
Trainable params: 8,703,947  
Non-trainable params: 0  
=====

### Structure of Second Model

To allow for easy comparison to the first model, I kept the same optimizer, loss function, batch size, validation split, and the number of epochs.



According to this graph, it appears that the model still overfits. The validation accuracy did increase by 9%. The final validation and testing accuracy is 65%.

### Third Model with Two Convolution Layers and 20% Dropout

I added a dropout layer with a 20% dropout after the flattened layer to alleviate the overfitting. The dropout layer works by randomly setting 20% of the hidden units to 0 during training.

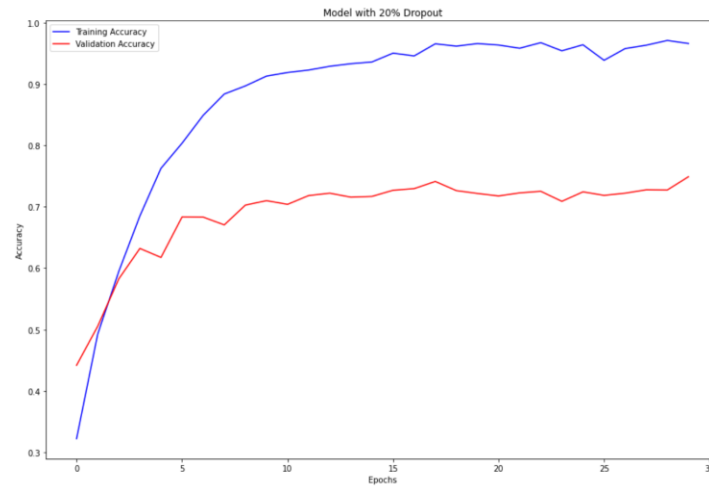
Model: "sequential\_5"

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d_10 (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_11 (Conv2D)	(None, 46, 46, 64)	32832
max_pooling2d_11 (MaxPooling2D)	(None, 23, 23, 64)	0
flatten_5 (Flatten)	(None, 33856)	0
dropout_5 (Dropout)	(None, 33856)	0
dense_10 (Dense)	(None, 256)	8667392
dense_11 (Dense)	(None, 11)	2827

=====  
 Total params: 8,703,947  
 Trainable params: 8,703,947  
 Non-trainable params: 0

### Structure of the Third Model

As previously, the optimizer, loss, batch size, validation split, and the number of epochs is the same.



The model appears to overfit less, but the model could still improve. The validation and testing accuracy both increases to 75%.

## Final Model

The final model builds on top of the last three models and adds a third convolution layer and max-pooling layer. The convolution layer contains 64 filters of size 3x3 and RELU activation function. The max-pooling layer is of size 2x2.

Model: "sequential\_6"

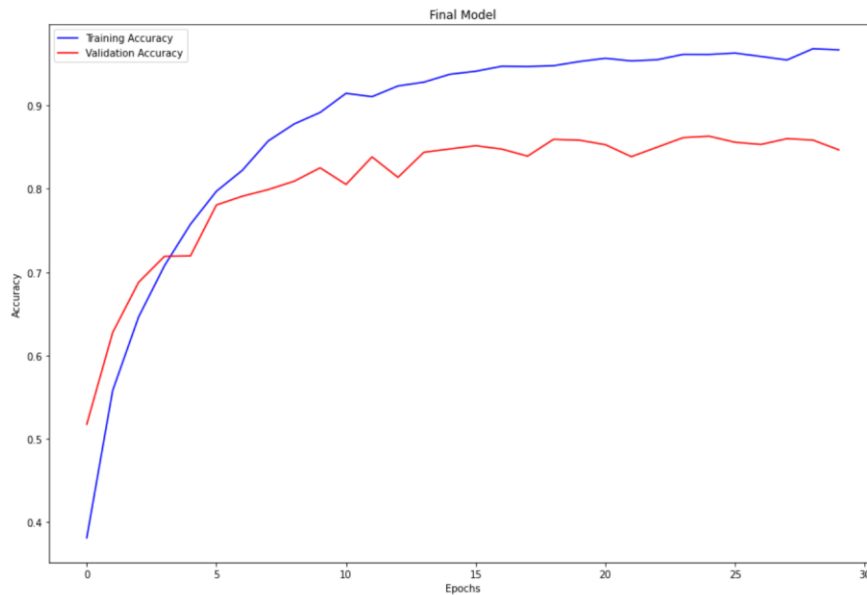
Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_13 (Conv2D)	(None, 46, 46, 64)	32832
max_pooling2d_13 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_14 (Conv2D)	(None, 21, 21, 64)	36928
max_pooling2d_14 (MaxPooling2D)	(None, 10, 10, 64)	0
flatten_6 (Flatten)	(None, 6400)	0
dropout_6 (Dropout)	(None, 6400)	0
dense_12 (Dense)	(None, 256)	1638656
dense_13 (Dense)	(None, 11)	2827

---

Total params: 1,712,139  
Trainable params: 1,712,139  
Non-trainable params: 0

## Structure of Final Model

Like previously in the other three models, the final is trained on the same optimizer, loss, batch size, validation split, and the number of epochs.



Looking at the validation accuracy, the model appears to have a 10% improvement over the previous model. The final validation and testing accuracy is 85%. With a high testing and validation accuracy, this would be the final model to classify any new weather images.

## Descriptions on How to Run and Compile Code

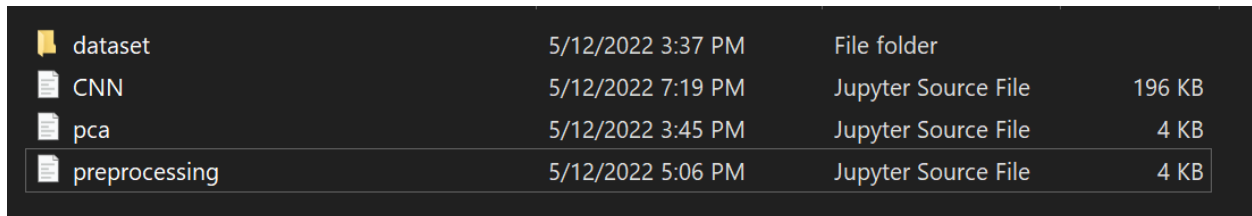
Original Dataset: <https://www.kaggle.com/datasets/jehanbhathena/weather-dataset>

Dataset I used for Training and Testing:

<https://drive.google.com/file/d/1b3fSbYvSlogZ1w3tTa2eOlh8hlFOTnq0/>

- If you use this dataset, you can only run the CNN Jupyter Notebook file since this dataset is created from the preprocessing and PCA files.

Please create a folder containing the dataset downloaded from Kaggle to ensure that the Jupyter notebooks can find the dataset folder. Additionally, my Jupyter notebooks should be in the same root folder. Please see the screenshot below for details.



dataset	5/12/2022 3:37 PM	File folder	
CNN	5/12/2022 7:19 PM	Jupyter Source File	196 KB
pca	5/12/2022 3:45 PM	Jupyter Source File	4 KB
preprocessing	5/12/2022 5:06 PM	Jupyter Source File	4 KB

Files must also be run in the format below or would result in an error.

- Preprocessing
- PCA
- CNN






### Preprocessing

- Run one cell after another
  - Any image errors will be printed under the cell-like this

```
Error at Image Location: fogsmog\4514.jpg
■ Error at Image Location: snow\1187.jpg
```
  - Once the third and fourth cell are completed, it should say “Successfully Completed” after the cell.

## PCA

- No creation of folders is required as Python would automatically create.
- Run one cell after another
  - Any image errors will be printed under the cell
  - Once the last cell is completed, it would say “Successfully Completed” after the cell.
- After completion, the folder format should be this:

○		dataset	5/12/2022 3:37 PM	File folder	
		PCA_dataset	5/12/2022 3:44 PM	File folder	
		CNN	5/12/2022 7:19 PM	Jupyter Source File	196 KB
		pca	5/12/2022 3:45 PM	Jupyter Source File	4 KB
		preprocessing	5/12/2022 5:06 PM	Jupyter Source File	4 KB

## CNN

- Run one cell after the other
- The outputs will automatically populate after each cell