# CMPSC 448 Final Project

## Introduction

The weather has a significant impact on our daily lives; it affects everything from sunshine and rainfall to cloud formations. Using deep learning techniques, this project attempts to tackle the challenge of classifying the weather. The basis for our investigation is provided by our dataset, which is an array of images capturing different weather conditions. The goal is to create a model with the help of a deep learning system that can differentiate between the different kinds of weather.

## The Task

Our initial investigation focused on two intriguing tasks: sentiment analysis of movie reviews and classifying weather images into three categories. These tasks represented distinct domains, encompassing text classification and image classification, respectively. After careful consideration, we chose to pursue the image classification task, specifically classifying images depicting clouds, rain, and sunshine.

To achieve our objective of classifying weather images, we employed two powerful neural network architectures: Transformer and Recurrent Neural Networks (RNNs). Transformer, a relatively recent innovation in the field of natural language processing, has demonstrated remarkable effectiveness in various tasks, including machine translation and text summarization. Its ability to capture long-range dependencies within sequences makes it well-suited for analyzing images, which can be viewed as sequences of pixels.

RNNs, on the other hand, have a long-standing history in sequence modeling, making them particularly adept at processing sequential data such as time series and natural language. Their recurrent nature allows them to incorporate temporal context into their predictions, a crucial aspect of image classification tasks.

## Dataset and Preprocessing

To train our weather classification models, we constructed a dataset composed of images representing three distinct weather conditions: clouds, rain, and sunshine. To ensure consistency and compatibility with the input requirements of the Transformer model, we resize all images to a standardized dimension of 224 x 224 pixels. This step ensured that the model received input data of uniform dimensions, facilitating efficient processing and feature extraction.

Recognizing the potential for overfitting, a common challenge in deep learning, we implemented a comprehensive data augmentation strategy. This strategy involved applying a series of transformations to the original images, effectively creating new training examples without altering the underlying weather category. Specifically, we added some randomness to the images by flipping them, changing the contrast, and more. The initial convolutional layers in our Transformer models play a crucial role in extracting meaningful features from the input images. These layers act as feature detectors, identifying and encoding patterns and relationships within the pixel intensities.

The RNN (Recurrent Neural Network) model was compiled using categorical cross-entropy as the loss function and the Adam optimizer. The training was conducted for 20 epochs with a batch size of 32, leveraging the augmented data generated by the ImageDataGenerator.

Upon completion of training, the model was saved to disk in two components: the model architecture in a file named 'rnn_image_classifier.json' and the model weights in a file named 'rnn_image_classifier_weights.h5'. This allows for easy reconstruction and deployment of the trained model in future use cases.

## Implementation and Architecture

### Transformer-Based Systems

Transformer-based systems do not utilize the same modules and methods as CNN or RNN. This already makes them unique in Deep Learning. Transformer models are also proven to be better than CNN and RNN at their jobs.

For model implementation, we opted for a Vision Transformer (ViT), a pre-trained Transformer model developed by Google Research specifically designed for image classification problems. To accelerate our model development and harness the power of ViT, we opted to retrain the pre-trained ViT model using our specialized dataset of weather images. Retraining a pre-trained model involves fine-tuning its parameters onto a new task, allowing it to adapt to the specific characteristics of our data.

### RNN

For the RNN model implementation, we gathered the image pixels into a sequence that were treated as a timestep. The input layer accepted these preprocessed images which then led to implementing several recurrent layers. They utilized Long Short-Term Memory (LSTM) units which captured the patterns and dependencies along each image. To mitigate the overfitting, dropout layers are interspersed between the recurrent layers which dropped a certain percentage of the neurons connection during training, forcing the network to learn more robust features. Following, the network transitions to a fully connected layer that integrates the learned features. The output layer consists of three neurons, corresponding to the three weather conditions (clouds, rain, sunshine), employing a softmax activation function for multi-class classification.

## Training

To ensure a fair comparison between the algorithms, each model was trained using the same dataset. This approach eliminated any potential bias or advantage arising from data disparities.

### Transformer-Based Systems

The training process involved optimizing the model parameters to minimize the classification error. This optimization process fine-tuned the model's ability to distinguish between different weather conditions based on the visual features extracted from the images.

For the ViT model, we employed an optimizer to drive the learning process and minimize classification errors. The training data was organized into folders corresponding to each weather category (clouds, rain, and sunshine), ensuring that the model received sufficient examples for each class.

To thoroughly train the model, we set the epoch number to 20. This means that the model iterated through the entire training dataset five times, allowing it to learn and generalize effectively.

Within each epoch, the model processed each image individually, utilizing the extracted features to refine its classification capabilities. This iterative process enabled the model to gradually improve its accuracy.

### RNN

The image data was first converted into sequences of pixel values allowing for RNN to work on them. We used categorical cross-entropy as the loss function, similar to the Transformer model, and employed the Adam optimizer for its efficiency in handling sparse gradients and adaptive learning rates.

Training was conducted over 20 epochs with a batch size of 32. Each batch of images is sequentially fed into the RNN, ensuring that the temporal dynamics within the images are effectively captured.

We applied BPTT to update the weights of the network which was crucial for training RNN as it handles the temporal sequences efficiently.

## Challenges and Solutions

While we did not observe concrete evidence of overfitting, we proactively implemented strategies to mitigate its potential occurrence. Overfitting is a common challenge in deep learning, where a model learns the training data too well and fails to generalize to unseen data. The randomness we added to the images definitely helped avoid overfitting.

Initially, the Transformer model failed to execute during training. Despite image transformations, the program encountered errors related to the training set. Upon investigation, we discovered that the issue lay in the way training data was being passed to the model. To address this problem, we employed logits to calculate the probability distribution over the classes for each image. This modification provided the model with the necessary information to process the training data correctly. Additionally, the model failed to recognize the dev set location, even though the images were present in the specified directory. Troubleshooting revealed that a missing folder in the path was causing the error. Resolving this issue involved adding the dev images to a dedicated class folder within the directory. This rectified the path issue and enabled the model to recognize and process the dev data effectively.

A common issue we face with standard RNNs is the vanishing gradient where gradients become too small for effective learning in deep networks. We counter this by using LSTM units which are specifically designed to maintain gradients across many timesteps.

RNNs are traditionally more suited for 1D sequential data, so we had to adapt them for image classification by preprocessing the images into a sequence format without losing spatial information. This involved trials with different sequencing methods and choosing the one that retained essential image features.

## Results and Observation

The resulting models demonstrated promising results, achieving high classification accuracy on a held-out dev set. These findings suggest that our approach, combining Transformer and RNN architectures, has the potential to effectively classify weather images. When running at higher epochs, our code took longer times to run. This is the reason why we had to limit our epoch numbers to usable levels.

We used variable epoch numbers to see how it would affect the accuracy of the models. We used epoch numbers 2 and 20 and evaluated our results. For the transformer, we saw accuracies of 95.4% and 97.2% at epoch numbers 2 and 20 respectively. Similarly, for the RNN, we saw accuracies of 72.9% and 80.61% at epoch numbers 2 and 20 respectively. We also observed that the transformer model is significantly more effective than the RNN. This might make sense as the transformer is known to run more efficiently than CNN and RNN.

The dev set size was 180 images. Generally, the accuracy increases at a diminishing rate as the epoch number rises and this pattern matches our models as well. We also looked at every image in the dev set along with the prediction to ensure that the accuracy was indeed matching. We successfully implemented our transformer and RNN models with acceptable levels of performance.