**Multi-class Weather Classification**

An image classification task

SYSC 5108: Deep Learning

April 12, 2023

| Mahitha Sangam  101212458  *Department of Systems and Computer Engineering*  *Carleton University*  *Ottawa, ON* | Abhishek Ahuja  101218880  *Department of Systems and Computer Engineering*  *Carleton University*  *Ottawa, ON* |
| --- | --- |

# ABSTRACT

The report presents a multi-class image classification study using Convolutional Neural Network (CNN) and Vision Transformer (ViT). The project takes up the study of weather classification using images to conduct the study. The primary objective of this research is to evaluate the performance of these two state-of-the art deep learning techniques in classifying diverse weather conditions. For this purpose, a comprehensive dataset consisting of diverse and accurately labeled images representing various weather conditions is used for training and evaluation.

The report begins with an introduction to the study, including motivation for selecting the application, deep learning and objective. It follows with the background of study, which talks about the reasoning for the solutions picked to solve the problem .Subsequently, dataset, algorithms, training and testing process are discussed under system description. The final results discuss the test loss, test accuracy, classification report and confusion matrices.

The results demonstrate that both architectures perform really well for image classification. However, the dataset size affects the performance of ViT. CNN was able to perform better as our dataset size is smaller.

In conclusion, the study highlights that the greatest and newest technology available doesn’t always fit everyone’s needs. A user should pick the algorithm which aligns with their own use case.

# Table of Acronyms

# Table of Math Notations

# Table of Contents

[**ABSTRACT 2**](#_uy641savop1f)

[**Table of Acronyms 3**](#_t3lesqup0ssr)

[**Table of Math Notations 4**](#_s40g414vuvgy)

[**Table of Contents 5**](#_2oay0yavnr1v)

[**Chapter 1: Introduction 1**](#_5ds7v4e9y7h2)

[**Chapter 2: Background 2**](#_fhawiss8j7b1)

[**Chapter 3: System Description 4**](#_ollxnu6lyois)

[3.1 Dataset 4](#_u06z0tkhcodo)

[3.2 Algorithms 4](#_ln2ms75rqbp4)

[3.2.1. CNN 4](#_bb9nz5knvmzd)

[3.2.2. Vision Transformer 4](#_dgc7egp4da2d)

[3.3 Training 4](#_kvdzaxyt50u6)

[3.3.1. CNN 4](#_i4mqedrce2rb)

[3.3.2. Vision Transformer 5](#_6ig71wgk8zl1)

[3.4 Testing 5](#_pvfswbha0idj)

[**Chapter 4: Results 6**](#_damzhnxg9qzr)

[**CONCLUSION 7**](#_b60sovtsjfv)

[**REFERENCES 8**](#_795svk6fwbs3)

# List of Figures

| List Of Images   * [Figure 2. CNN Model Loss Plot](#tykls428amaj) * [Figure 3. CNN Confusion Matrix](#p7egbd4i8k8s) * [Figure 4. ViT Model Accuracy](#nl7kgty95v65) |
| --- |

# Chapter 1: Introduction

Image classification, a fundamental task in computer vision, has witnessed remarkable advancements in recent years, primarily driven by the development of deep learning techniques. These techniques have been successfully employed in a wide range of applications, including object recognition, scene understanding, and medical image analysis. The ability of deep learning models to automatically learn hierarchical representations from raw data has been a key factor in their success, outperforming traditional methods that rely on handcrafted features and domain-specific knowledge.

Among the deep learning architectures employed for image classification, Convolutional Neural Networks (CNNs) have emerged as the dominant approach, thanks to their capacity to learn local patterns and spatial hierarchies in images. More recently, Vision Transformers (ViTs), which leverage the transformer architecture, have gained significant attention for their ability to model long-range dependencies and global context in images. Both CNNs and ViTs have demonstrated exceptional performance in various image classification tasks.

In this study, we have chosen multi-class weather classification as a representative problem to compare the performance of CNNs and ViTs. Weather classification is a crucial aspect of meteorology, environmental science, and climate studies, with widespread applications in agriculture, disaster management, transportation, and renewable energy. Accurate and timely weather predictions can not only help mitigate the adverse effects of extreme weather events but also enable better decision-making across various sectors.

The primary objective of this project is to investigate the potential of both deep learning architectures in classifying diverse weather conditions using image data. To this end, a comprehensive dataset consisting of diverse and accurately labeled images representing various weather classes is utilized for training and evaluation purposes. The evaluation criteria comprise accuracy, precision, recall, F1-score, and computational efficiency. The results and insights obtained from this study can provide guidance for selecting appropriate models based on specific project requirements and constraints, ultimately contributing to the development of more accurate and efficient weather classification systems and advancing the understanding of image classification techniques in general.

# Chapter 2: Background

Image classification is the task of assigning predefined categories to images based on their visual content. It serves as a foundation for many computer vision applications, such as object recognition, scene understanding, and content-based image retrieval. Over the years, various approaches have been proposed for image classification, ranging from traditional methods, like feature extraction and machine learning classifiers, to more recent deep learning-based techniques.

Weather classification involves categorizing atmospheric conditions into discrete classes, such as clear, cloudy, rainy, snowy, or foggy. Accurate weather classification is essential for numerous applications, including meteorology, agriculture, disaster management, and transportation. With the increasing availability of high-resolution satellite imagery and remote sensing data, the potential for developing more advanced weather classification techniques has grown significantly.

1. Image Classification Techniques

The evolution of image classification techniques can be broadly divided into three phases:

1. Traditional methods: These approaches rely on handcrafted features, such as color histograms, texture descriptors, and shape-based features, followed by the application of machine learning classifiers like Support Vector Machines (SVM), k-Nearest Neighbors (kNN), and Decision Trees. These methods are often limited in their ability to adapt to complex and dynamic patterns in images and can be time-consuming to design and optimize.
2. Shallow learning techniques: These methods involve the use of feature learning techniques, such as Principal Component Analysis (PCA), Independent Component Analysis (ICA), and Sparse Coding, to automatically learn more expressive features from raw data. Despite their ability to adapt to data, shallow learning techniques are generally outperformed by deep learning models, particularly in tasks with large and complex datasets.
3. Deep learning techniques: Deep learning models, such as CNNs and ViTs, have demonstrated remarkable success in image classification tasks due to their ability to learn hierarchical representations and model complex patterns in data. CNNs are designed to capture local patterns and spatial hierarchies using convolutional layers, while ViTs leverage self-attention mechanisms to model long-range dependencies and global context in images.

B. Chosen Techniques

In this study, we chose to investigate deep learning techniques, specifically CNNs and ViTs, for the following reasons:

1. Superior performance: Both CNNs and ViTs have consistently outperformed traditional and shallow learning methods in various image classification tasks, demonstrating their ability to learn more expressive and robust representations from data.
2. Scalability: Deep learning models are highly scalable and can be trained on large and diverse datasets, enabling them to capture more complex and subtle patterns in images.
3. Transferability: Pre-trained deep learning models can be fine-tuned for specific tasks, allowing for the transfer of learned knowledge to new domains and accelerating the training process.
4. Architectural innovations: The continuous development of novel deep learning architectures, such as residual connections, attention mechanisms, and efficient convolutional layers, provides opportunities for further improvements in performance and efficiency.

Considering these factors, the selection of CNNs and ViTs for this study enables a comprehensive comparison of their performance and efficiency in the context of multi-class weather classification, providing valuable insights for practitioners and researchers alike.

# Chapter 3: System Description

This chapter provides a detailed description of the system developed for the multi-class weather classification task, including the dataset, algorithms, the training and testing processes.

## Dataset

Multi-class weather dataset(MWD) for image classification is a valuable dataset used in the research paper entitled “Multi-class weather recognition from the still image using heterogeneous ensemble method”. It was published on Mendeley [1]. However, the dataset is taken from Kaggle [2] for this project as it has an updated folder structure for the dataset folder, to facilitate the data load procedure.

The dataset has a total of 1125 images across 4 classes of Sunrise, Shine, Rain and Cloudy. The distribution is given in Table 1.

Table 2. Dataset Class Distribution

| **Class** | **Number of Images** |
| --- | --- |
| Sunrise | 357 |
| Shine | 253 |
| Rain | 215 |
| Cloudy | 300 |

## Algorithms

### CNN

A CNN is an effective deep learning model for processing grid-like data, such as images, where local spatial relationships between features are important for making predictions.

The CNN consists of several types of layers:

* + - Convolutional layers: These layers perform convolution operations on the input image or feature maps. The convolution operation involves sliding a filter or kernel (of size filter\_size, e.g., (3, 3)) over the input image or feature maps and computing the element-wise product followed by a sum to produce a single value for the output feature map. The purpose of these layers is to learn local features from the input. Each convolutional layer in the model is defined as follows:

model.add(Conv2D(num\_filters, filter\_size, activation='relu'))

* + - Pooling layers: These layers perform downsampling operations, such as max pooling or average pooling, to reduce the spatial dimensions of the feature maps. This helps in reducing the computational complexity of the model and extracting more robust features. In our model, we use max pooling layers:

model.add(MaxPooling2D((2, 2)))

* + - Fully connected (dense) layers: After the convolutional and pooling layers, the feature maps are flattened into a 1D vector, which is then passed through one or more fully connected layers. These layers learn global patterns in the feature maps and perform classification tasks. In our model, we use dropout to prevent overfitting, as shown below:

model.add(Dense(num\_dense\_units, activation='relu'))

model.add(Dropout(dropout\_rate))

* + - Output layer: The final layer in the model is the output layer, which performs the classification using a softmax activation function. The softmax function computes the probability distribution of the input classes, ensuring that the sum of probabilities for all classes equals 1.

model.add(Dense(num\_classes, activation='softmax'))

### Vision Transformer

The Vision Transformer (ViT) is a deep learning architecture that applies the Transformer model, originally designed for natural language processing tasks, to computer vision tasks like image classification. The key insight behind ViT is that an image can be treated as a sequence of patches, similar to a sequence of words in a sentence, allowing the Transformer to learn and capture the relationships among these patches.

The original Transformer model was proposed by Vaswani et al. in the paper "Attention Is All You Need" (2017) [3]. The Vision Transformer was introduced by Dosovitskiy et al. in the paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" (2020) [4].

The main components of the Vision Transformer are as follows:

Patch Embedding: The input image is divided into non-overlapping patches of a fixed size (e.g., 16x16 pixels). Each patch is then flattened into a 1D vector and passed through a linear layer (i.e., fully connected layer) to obtain a patch embedding. The output dimension of the linear layer is the embedding dimension.

Position Embedding: Positional information is crucial in the Transformer architecture to capture the spatial relationships between patches. A position embedding is added to each patch embedding to encode its position in the image. The position embeddings are learnable parameters and are updated during training.

Transformer Blocks: The combined patch and position embeddings are fed into a series of Transformer blocks. Each block consists of a multi-head self-attention layer and a position-wise feed-forward layer, connected through residual connections and layer normalization. The multi-head self-attention layer allows the model to learn the relationships among the patches, while the feed-forward layer captures local information within each patch.

Classification Head: After passing through the Transformer blocks, a global average pooling layer is applied to reduce the dimensionality of the output. Finally, a fully connected layer with a softmax activation function is used to produce the class probabilities for the image classification task.

## Training

### CNN

In this project, we trained a Convolutional Neural Network (CNN) using Keras and TensorFlow to classify images in our dataset. The training process can be divided into the following steps:

* Data Augmentation: To increase the diversity of the training data and improve the model's generalization capability, we employed data augmentation techniques using Keras' ImageDataGenerator. We applied various transformations, such as rotation, shifting, zooming, and flipping, to generate more training samples from the existing data. The validation set was created by setting a validation split of 25%.
* Creating Generators: We created separate generators for the training and validation sets using the flow\_from\_directory method. The generators were configured with the appropriate target size, batch size, and class mode to efficiently feed the model during training.
* Model Architecture: We designed a CNN architecture with a variable number of convolutional and fully connected layers. The CNN included convolutional layers with ReLU activation functions, followed by max-pooling layers to reduce the spatial dimensions. After the convolutional layers, the output was flattened and passed through fully connected layers with dropout for regularization.
* Hyperparameter Tuning: To find the optimal hyperparameters for the model, we employed a grid search with cross-validation. We used scikit-learn's GridSearchCV in combination with KerasClassifier to perform the search. The hyperparameters considered in the search included the number of convolutional layers, filter sizes, number of dense layers, number of dense units, and dropout rate.
* Model Training: The model is trained using the stochastic gradient descent optimization algorithm, specifically the Adam optimizer. The categorical cross-entropy loss function is used to compute the discrepancy between the predicted class probabilities and the true class labels. Mathematically, the categorical cross-entropy loss L for a single data point can be defined as:

L(y, p) = - ∑ [y\_i \* log(p\_i)]

where y is the true class label (one-hot encoded), p is the predicted class probability distribution, and the summation is over all classes i.

The optimization algorithm aims to minimize the loss function by updating the weights of the model. In the case of the Adam optimizer, the weights are updated using a combination of the first and second moments of the gradients, allowing for adaptive learning rates for each weight.

Weight updates are performed using the following formulas:

m\_t = β1 \* m\_{t-1} + (1 - β1) \* g\_t

v\_t = β2 \* v\_{t-1} + (1 - β2) \* g\_t^2

m\_hat\_t = m\_t / (1 - β1^t)

v\_hat\_t = v\_t / (1 - β2^t)

w\_t = w\_{t-1} - α \* m\_hat\_t / (sqrt(v\_hat\_t) + ε)

where m\_t and v\_t are the first and second moments of the gradients, g\_t is the gradient at time step t, β1 and β2 are the exponential decay rates for the moments, α is the learning rate, and ε is a small constant to prevent division by zero. The weights w are updated at each time step.

The model is trained using the training set with data augmentation to increase the effective size of the dataset and improve generalization. The training is performed for a specified number of epochs, where each epoch represents a complete iteration through the training dataset.

During the training process, the optimizer adjusts the weights of the model to minimize the categorical cross-entropy loss. The performance of the model is evaluated on the validation set at the end of each epoch. The model's weights are updated iteratively to optimize the classification performance on the training set while monitoring its generalization capability on the validation set.

### Vision Transformer

The model is trained using the Adam optimization algorithm, which is an extension of the stochastic gradient descent (SGD) algorithm, with an adaptive learning rate. The categorical cross-entropy loss function is used to compute the discrepancy between the predicted class probabilities and the true class labels:

loss = - Σ(y\_true \* log(y\_pred))

The input images are resized to a size of 256x256 pixels, and the training set undergoes data augmentation using an ImageDataGenerator. The data augmentation techniques include rescaling, rotation, width and height shift, shear, zoom, and horizontal flipping. The validation split is set to 25% of the training data. The training and validation sets are created using flow\_from\_directory methods from the ImageDataGenerator.

The Vision Transformer architecture is defined within the create\_vit\_model function. The model consists of an input layer, patch embedding layer, position embedding layer, multiple transformer blocks, a global average pooling layer, and an output layer.

The input layer takes images with dimensions (img\_size, img\_size, 3), where img\_size is set to 256. The patch embedding layer divides the input image into non-overlapping patches and converts them into a sequence of vectors with an embedding dimension of 256. The position embedding layer adds positional information to the patch embeddings. The transformer blocks consist of multi-head self-attention layers and feed-forward layers, with dropout and layer normalization applied after each layer. The multi-head self-attention mechanism is defined as:

Attention(Q, K, V) = Softmax((Q \* K^T) / sqrt(d\_k)) \* V

Here, Q, K, and V are the query, key, and value matrices, and d\_k is the dimension of the key vectors. The attention function is applied to the input matrix in parallel, with the results concatenated and linearly transformed to produce the final output.

In this implementation, there are 4 transformer blocks, and each multi-head self-attention layer has 8 attention heads. The feed-forward network inside the transformer blocks has a dimension of 512. After the transformer blocks, a global average pooling layer is applied to the output of the last transformer block, which reduces the dimensionality of the output and computes the average of each feature map. The resulting output is a 1D feature vector of size equal to the embedding dimension (256 in this case).

Finally, the output layer is a dense layer with a number of units equal to the number of classes (num\_classes, which is set to 4 in this case) and a softmax activation function. The softmax activation function is used to convert the output of the last dense layer into class probabilities:

softmax(x\_i) = exp(x\_i) / Σ(exp(x\_j)), for all j in the output layer

The model is then compiled with the Adam optimizer (with a learning rate of 1e-4), the categorical cross-entropy loss function, and the accuracy metric.

The model is trained for 10 epochs using the fit method, with the training and validation generators as input. The fit method performs forward and backward passes through the model, updating the weights according to the Adam optimization algorithm and minimizing the categorical cross-entropy loss. The training process alternates between updating the weights and evaluating the model on the validation set to monitor its performance.

During the training process, the history object records the training and validation loss and accuracy for each epoch. This information can be plotted to visualize the learning process, assess the model's performance, and identify potential issues such as overfitting or underfitting.

In summary, the training process involves feeding input images through the Vision Transformer model, computing the categorical cross-entropy loss, and updating the model weights using the Adam optimizer. The model is trained for 10 epochs, with performance evaluated on the validation set after each epoch. The goal of the training process is to minimize the loss function and maximize the classification accuracy.

## Testing

Once the model has been trained, it is essential to evaluate its performance on a separate test set, which was not used during training. The purpose of the test set is to provide an unbiased estimate of the model's performance on new, unseen data.

### CNN

The trained model is tested on a separate dataset to evaluate its performance. The test dataset is generated using the ImageDataGenerator with rescaling, and the generator is created with the test\_datagen.flow\_from\_directory() method. The test dataset has the same image size (150 x 150) as the training and validation datasets.

* After training the model with the best hyperparameters found during the grid search, the model's performance is assessed using the test dataset. The model's test loss and accuracy are evaluated using the model.evaluate() method.

test\_loss, test\_acc = model.evaluate(test\_generator)

print("Test loss:", test\_loss)

print("Test accuracy:", test\_acc)

* the model's accuracy and loss are plotted over the training epochs to visualize the learning process.

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

..

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

..

* To further analyze the model's performance, we generate a classification report and confusion matrix. The classification report includes metrics like precision, recall, and F1-score for each class, which provide more insights into the model's performance.

y\_pred\_categorical = np.argmax(y\_pred, axis=1)

y\_true\_categorical = test\_generator.classes

print(classification\_report(y\_true\_categorical, y\_pred\_categorical))

* Lastly, the confusion matrix is created to visualize the model's classification results. It helps in understanding the correct and incorrect predictions for each class, which can be useful in identifying the model's strengths and weaknesses.

cm = confusion\_matrix(y\_true\_categorical, y\_pred\_categorical)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, cmap='Blues',vmin = 0, vmax = 70, xticklabels=test\_generator.class\_indices.keys(), yticklabels=test\_generator.class\_indices.keys())

..

### Vision Transformer

The test set is prepared using the ImageDataGenerator class with only the rescale parameter set to 1./255, ensuring that the pixel values are normalized between 0 and 1. The test generator is then created using the flow\_from\_directory method, specifying the test directory, target image size, batch size, and class mode, and setting the shuffle parameter to False to maintain the order of the test samples.

After training the model, it is evaluated on the test set using the evaluate method, which returns the test loss and test accuracy:

test\_loss, test\_acc = model.evaluate(test\_generator)

print("Test loss:", test\_loss)

print("Test accuracy:", test\_acc)

To further analyze the model's performance, a series of evaluation plots and metrics are generated:

* Training and Validation Accuracy and Loss: These plots show the accuracy and loss values for the training and validation sets during each training epoch. They can help to identify overfitting or underfitting issues, and to determine if the model's performance has converged.

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

...

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

…

* Classification Report: This report displays the precision, recall, F1-score, and support for each class. It provides a detailed view of the model's performance in classifying each category, highlighting potential imbalances or misclassifications.

y\_pred = np.argmax(model.predict(test\_generator), axis=1)

y\_test = test\_generator.classes

print(classification\_report(y\_test, y\_pred))

* Confusion Matrix: The confusion matrix shows the true and predicted labels for each class. It can be used to identify specific cases where the model is struggling to make accurate predictions, or to confirm that the model is performing well across all categories.

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', vmin = 0, vmax = 70)

...

By examining these evaluation metrics and plots, you can assess the model's performance, identify areas for improvement, and determine the suitability of the model for the given task. The results are provided in the next section.

# 

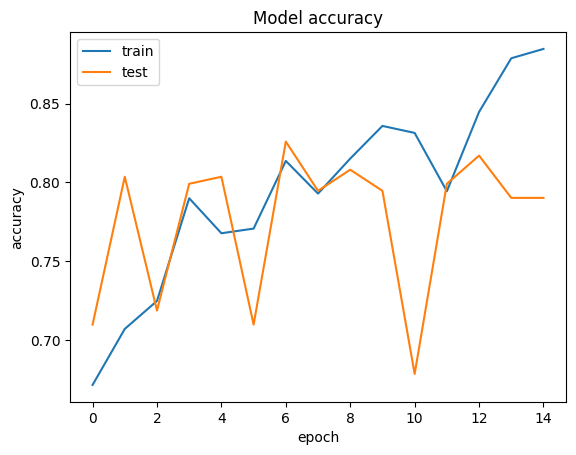
# Chapter 4: Results

In this sections, we will first talk about individual results and then we’ll compare them to draw the conclusions.

## CNN Results

Test loss: 0.44590064883232117

Test accuracy: 0.8844444155693054

  
Figure 1. CNN Model Accuracy Plot

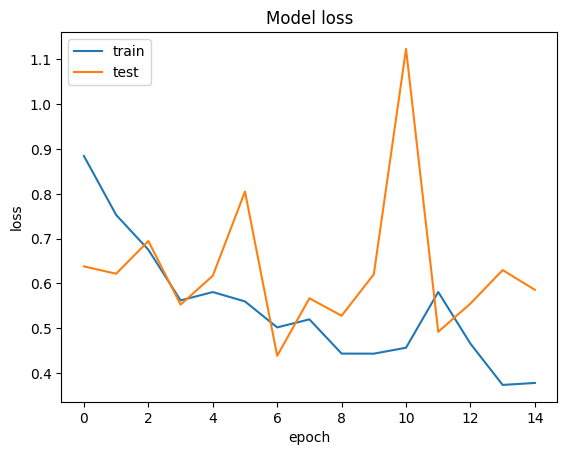


Figure 2. CNN Model Loss Plot

Table 3. CNN Classification Report

|  | precision | recall | f1 score | support |
| --- | --- | --- | --- | --- |
| 0 | 0.91 | 0.87 | 0.89 | 60 |
| 1 | 0.97 | 0.70 | 0.81 | 43 |
| 2 | 0.75 | 0.98 | 0.85 | 51 |
| 3 | 0.96 | 0.94 | 0.95 | 71 |
|  | | | | |
| accuracy |  |  | 0.88 | 225 |
| macro avg | 0.90 | 0.87 | 0.87 | 225 |
| weighted avg | 0.90 | 0.88 | 0.88 | 225 |

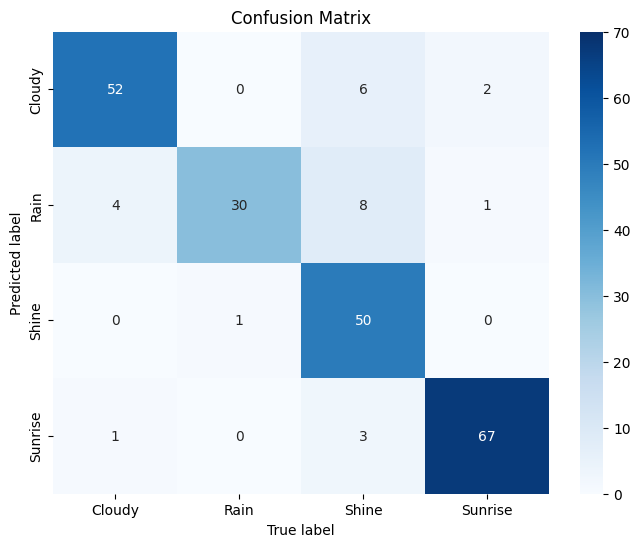


Figure 3. CNN Confusion Matrix

## ViT Results

Test loss: 0.5706254243850708

Test accuracy: 0.7511110901832581

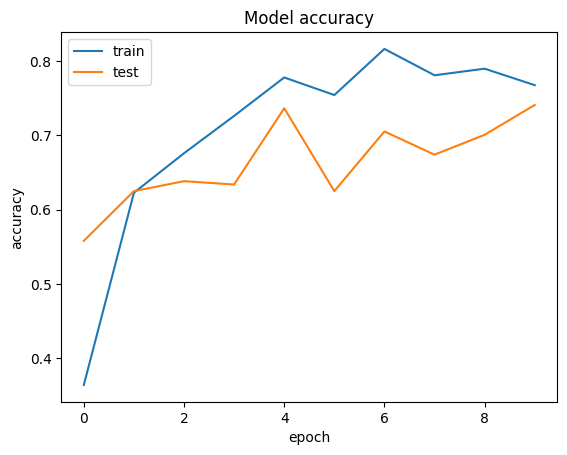


Figure 4. ViT Model Accuracy

## Comparison

# CONCLUSION

# REFERENCES

[1] Ajayi, Gbeminiyi (2018), “Multi-class Weather Dataset for Image Classification”, Mendeley Data, V1, doi: 10.17632/4drtyfjtfy.1

[2] Multi-class Weather Dataset https://www.kaggle.com/datasets/pratik2901/multiclass-weather-dataset

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30, 5998-6008.

[4] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: transformers for image recognition at scale.