# Atmospheric Gravity Wave Detection Using Transfer Learning Techniques

[1]Jorge López González, [2]Theodore Chapman, [3]Kathryn Chen, [4]Hannah Nguyen, [5]Logan Chambers, [4]Seraj A.M. Mostafa, [4]Jianwu Wang, [4]Sanjay Purushotham, [4,6]Chenxi Wang, [6,7]Jia Yue
[1]University of Puerto Rico, Río Piedras, San Juan, PR, USA
[2]University of Rochester, Rochester, NY, USA
[3]University of California, Santa Barbara, Santa Barbara, CA, USA
[4]University of Maryland, Baltimore County, Baltimore, MD, USA
[5]Albany State University, Albany, GA, USA
[6]NASA Goddard Space Flight Center, Greenbelt, MD, USA
[7]Catholic University of America, Washington DC, USA
Emails: [1]jorge.lopez19@upr.edu, [2]tchapma6@u.rochester.edu, [3]kathryn_chen@ucsb.edu, [4]{hannahn2, serajmostafa, jianwu, psanjay}@umbc.edu, [5]lchambe3@students.asurams.edu, [6]{chenxi.wang, jia.yue}@nasa.gov

***Abstract*—Atmospheric gravity waves are produced when gravity attempts to restore disturbances through stable layers in the atmosphere. They have a visible effect on many atmospheric phenomena such as global circulation and air turbulence. Despite their importance, however, little research has been conducted on how to detect gravity waves using machine learning algorithms. We faced two major challenges in our research: our raw data had a lot of noise and the labeled dataset was extremely small. In this study, we explored various methods of preprocessing and transfer learning in order to address those challenges. We pre-trained an autoencoder on unlabeled data before training it to classify labeled data. We also created a custom CNN by combining certain pre-trained layers from the InceptionV3 Model trained on ImageNet with custom layers and a custom learning rate scheduler. Experiments show that our best model outperformed the best performing baseline model by 6.36% in terms of test accuracy.**

***Index Terms*—atmospheric gravity waves, deep learning, image denoising, transfer learning, model customization**

## I. Introduction

Gravity waves (or, to be more precise, buoyancy waves) are air oscillations caused by gravity and buoyancy force. They are common in fluids like the atmosphere and the ocean. Gravitational waves, on the other hand, are spatial and temporal "ripples" caused by energetic processes in the universe. Gravity waves are created when parcels of air are displaced from their equilibrated position and the force of gravity attempts to restore the equilibrium. Various atmospheric disturbances, such as airflow over mountains, jet streams, and thunderstorms, cause atmospheric gravity waves.

These disturbances can occur when air is forced to rise upwards in stable air, creating a wave pattern as the air sinks back down over time, similar to how ripples are formed when a stone is thrown into a still water surface [1]. Gravity waves can sometimes be detected using radar or satellites, and they can be seen inside images as producing patterns resembling ripples or clouds.

When present, gravity waves have a significant impact on many weather phenomena. Simulations of middle atmosphere circulation are more accurate when gravity waves are correctly factored in, indicating that they influence circulation. They may even have an effect on tidal waves [2], and are also capable of causing a type of turbulence known as Clear Air Turbulence, which is responsible for up to 40% of all aviation accidents [3]. Because of a better understanding of the importance of gravity waves in recent years, interest in detecting these waves has skyrocketed. To gain a better understanding of this phenomenon, some researchers are turning to machine learning techniques.

During the research, we focused on machine learning techniques for detecting gravity waves in the atmosphere. Two major challenges in detecting gravity waves from satellite images were addressed. The first major challenge involved working with a noisy dataset. Because the images were scanned from a satellite, they could have been corrupted by a defective sensor, a faulty channel, or other factors that degraded the quality of the scanned images. The problem with using noisy images for training is that the data may not be correctly interpreted by the learning model. The second major challenge was dealing with the only dataset available at the time of research, which was a small dataset provided to use for the gravity waves classification problem. This dataset included 710 images for training, 140 for validation, and 236 for testing. When dealing with a small dataset, the problem arises because the model has fewer examples from which to learn general features. If a model is trained on data that contains too many specific details, it will be unable to learn generalized features and thus will not be able to make correct predictions.

To address the aforementioned two challenges, we investigated various techniques, and our contributions are summarized below. We also made our implementation open source, with the source code available at Big Data REU GitHub repository [4].

- Different techniques were explored to signify the signals and remove noises from the data in our datasets. Because the radiation signals in our raw data were very weak, we transformed the data before saving them as images. We further used Fast Fourier transform (FFT) to reduce noises in satellite images. With these preprocessing techniques, gravity waves were more visible, and both signal to noise ratios (SNR) and pixel distributions improved significantly.
- To overcome the restrictions of training with a small dataset, numerous transfer learning strategies using various types of models were proposed. The first model made use of an autoencoder pre-trained on unlabeled data. The second model is a custom model that combines some aspects of a pre-trained model with a convolutional architecture and a trainable custom classifier. The pre-trained model is utilized for feature extraction and is trained with the ImageNet dataset.
- Several experiments were carried out to assess the performance of all models used for training and testing. The results compare the accuracy scores of all models and demonstrate the ability of deep learning methods to predict the presence of gravity waves on scanned satellite images.

The sections for the remainder of the paper were organized as follows. In Section II, we presented related work on studies that address problems related to this research. Following that, in Section III, we provided details about the overall pipeline from our work, followed by a demonstration of the dataset and the techniques used for image preprocessing in Section IV. Section V provided details on the methods used for the approaches of using an autoencoder and a custom model for training and testing. Section VI contains a detailed discussion of the experiments and their results. This paper's conclusion can be found in Section VII.

## II. Background and Related Works

### A. Gravity Wave Detection

Most existing methods for detecting gravity waves require researchers to take measurements of atmospheric features in the target region. For example, Zink et al. [5] and Colligan et al. [6] identified gravity waves using radiosonde sounding measurements of horizontal wind speed. Linear transformation was applied to make the measurements to more interpretable. To isolate the superimposed gravity waves, the resulting function for local maxima was scanned and the wave packet data was recorded at those points.

That data was recorded and further analyzed with Stokes parameter analysis to determine wave direction, speed, and height- all of which aid in the identification of gravity waves. This general technique produced fairly accurate results, but required access to various weather predicting instruments. Furthermore, it can only detect gravity waves in small areas.

Coïsson et al. determined that it was possible to detect tsunami-induced gravity waves using satellite radio occultation measurements [7]. By analyzing the radio waves' changing amplitude and frequency, they could identify characteristics suggesting that the waves were not from the ionosphere, instead the gravity waves were excited by tsunamis. Koch et al. proposed an automatic mesoscale gravity wave detection system in 1997 [8].

Gravity wave detection can be made more efficient and accessible using the bountiful and publicly available satellite image data. Thus far, very little research has been conducted regarding detection using artificial intelligence. There is a notable study by Lai et al. that developed a convolutional neural network based program, which extracts gravity wave patterns in all-sky airglow images [9]. This could be achieved by using a convolutional neural network to classify images of clear skies, and unwarp them onto geographic maps. The gravity waves were then localized using the Object Detection API from TensorFlow.

Our study seeks to establish machine learning methods for gravity wave detection using satellite imagery. Specifically, we aimed to create a model that could classify a small dataset of satellite images with 95% validation accuracy.

### B. Transfer Learning

Transfer learning involves training a model on two tasks, typically referred to as the source tasks and target tasks, while attempting only to maximize the model's ultimate performance on the target task. This differs from multi-task learning because, in multi-task learning, the researcher aims for good performance in all of the domains where the model is being trained on. [10]. The strategy of transfer learning covers a wide variety of approaches, which are often further subdivided. One popular taxonomy categorizes approaches according to the sort of labels available for the source and target tasks, and how closely the samples in the source task resemble those in the target task [11].

The transfer learning we attempted involves transfer from the source task of categorizing images in the

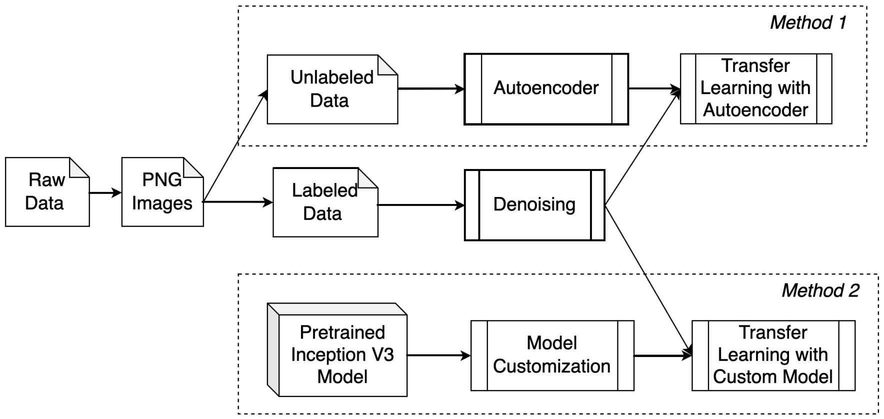


Fig. 1. Diagram of the Overall Pipeline.

Imagenet dataset to categorizing grayscale satellite imagery. This is referred to as “inductive transfer learning” because the training on the source domain was not meant to directly improve performance on the target domain, but rather to improve the model’s ability to learn the target task. More specifically, Pan and Yang categorize our approach of using training on the source domain as an initialization algorithm for our model as “feature-representation-transfer” [10].

Generally, feature-representation transfer learning involves training a model on a domain structured similarly to the target domain. For this training, the features would be important for the understanding of the samples from the source domain. Therefore, this same features would tend to also be informative for the target domain. For instance, Blitzer et al. designed a training method to perform transfer sentiment classification from one domain of text to another. They relied on structural commonalities between documents in the same language but in different domains [12].

Another common approach to feature representation transfer learning would be to perform unsupervised feature extraction on unlabeled images from a domain, very similar to the target domain. Typically such approaches would compress the inputs with the intent of forcing a model to produce abstract representations of the raw inputs, which capture regularities within the input domain. For instance, Glorot et al. used an unsupervised autoencoder model to transform textual reviews into compressed summaries, which was used as input for sentiment classifiers, resulting in significantly improved classification performance [13].

## III. Overall Pipeline

Figure 1 shows the overall pipeline we used. The pipeline starts with raw data in the format of hdf5 files supplied by the NASA Soumi NPP satellite. We transformed the data into PNG images that were comprehensible to humans. Some PNG images were labeled manually, while the rest remained unlabeled. Since our input data consisted of noisy images which can negatively impact performance of the models, the labeled images were also denoised with FFT denoising.

We used two methods for classification. For the first method (more at Section V-A), we leveraged the unlabeled images to train an autoencoder. After that we saved the encoder block and used it as part of a classification model trained on the labeled denoised dataset. For the second method (more at Section V-B), we cut off the Inception V3 model pretrained on ImageNet at the last layer with the larger (14x14) feature map. We added custom layers to this model and used transfer learning to train the model with the labeled dataset.

## IV. Data Preprocessing

In this study we used NASA VIIRS DNB (Day Night Band) images [14] collected from satellite Suomi NPP. The raw data were in Hierarchical Data Format version 5 (HDF5) [15]. Later we processed the data into PNG format for the rest of the experiments.

### A. Image Generation From Raw Data

The raw data consisted of measurements of the radiance of light with wavelengths in the range $(0.5\mu m, 0.9\mu m)$. It was stored in a 1000x1000 matrix, the cells of which corresponded to a spatial mapping over a portion of the Earth’s atmosphere. Images were recorded every six minutes of the region visible to the satellite at that time. Radiance values were in the range $(-10^{-9}, 10^{-9})$. We transformed the raw data files according to Algorithm 1 to produce images that were comprehensible to humans. See Figure 2 for an illustration. The resulting images were classified by hand and then further processed.

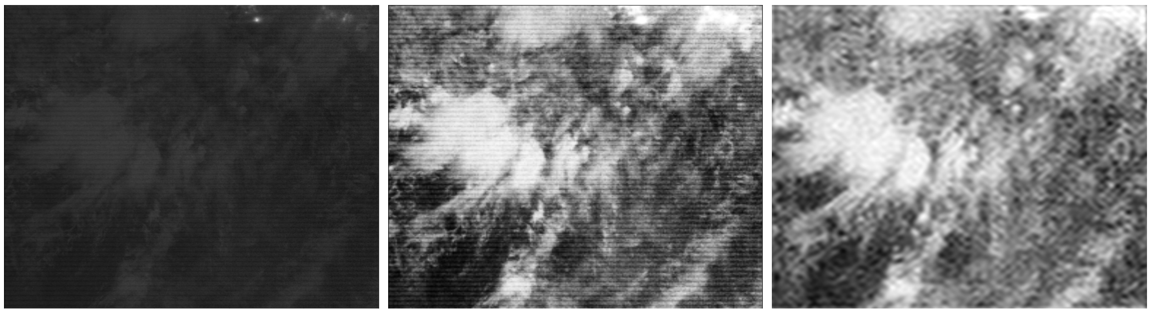


Fig. 2. Normalized raw data (left), preprocessed PNG (middle) and FFT denoised PNG (right).

**Algorithm 1** HDF5 to PNG

```
Require: $arr \in R^{1000x1000}$
1: $F(x) \leftarrow P(Z <= x)$ for $Z \sim$ Normal distribution fitted to the values of arr
2: $arr \leftarrow arr - min(arr)$
3: $arr \leftarrow \frac{arr}{median(arr)} * 0.5$
4: $arr \leftarrow clip(arr, 0, 1)$
5: $arr \leftarrow F(arr)$    ▷ Transform the approximately normally distributed values to uniform ones
6: $arr \leftarrow clip(arr, 0, 1)$
```

## B. Image Denoising using Fourier Transforms

One common technique for image processing is Fourier filtering. In Fourier filtering, one zeroes out a subset of elements of the image's frequency domain representation. This often serves to significantly reduce the complexity of the image significantly with minimal impact on its visual clarity. We did the implementation by taking the 2D Fourier Transform of the image and zeroing out all but the highest and lowest frequencies. Finally we took the inverse Fourier transform of the remaining frequencies to produce a denoised image.

**Algorithm 2** FFT Denoising

```
$INPUT : i \leftarrow image$
$OUTPUT : image \rightarrow I$
1: $i \leftarrow fft2(i)$
2: $i \leftarrow (1 - 2 * fraction) * i$
3: $I \leftarrow ifft2(i)$
```

Fourier transform technique transforms an image into sines and cosines of varying amplitudes and phases to the frequency domain. This, in fact reveals the repeating patterns of amplitudes and phases [16] which is the special case of the "orthogonal functions". Breaking down complicated signals into linear superposition to get the result for the original signal is the main idea of the function.

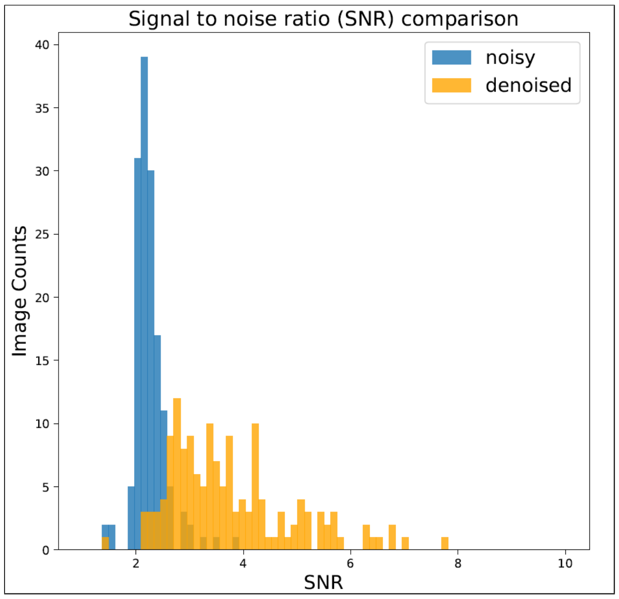


Fig. 3. SNR plot for images before and after denoising.

All the PNG images were denoised using FFT technique as it offered an improvement in model performance (more at Section 5 & 6) compared to other denoising techniques such as image thresholding. A pseudo representation of the fourier filtering used is shown in Algorithm 2 that we used in this work to denoise all the images.

At first, we applied the FFT (*fft2*) algorithm that returns the two-dimensional fourier transform matrix using a fast fourier transformation. The following step was to crop out all the signals except the top and bottom 10%, row and column-wise, so we could remove the unnecessary frequency elements. In the next steps we would reverse the process of the first two steps, where we reconstruct the image from signals. We utilized Python's *SciPy* package in this process. Figures 3 and 4 illustrate the impact of denoising the images from the dataset in different ways. In Figure 3 we plotted 150 images to compare the differences between the noisy images and

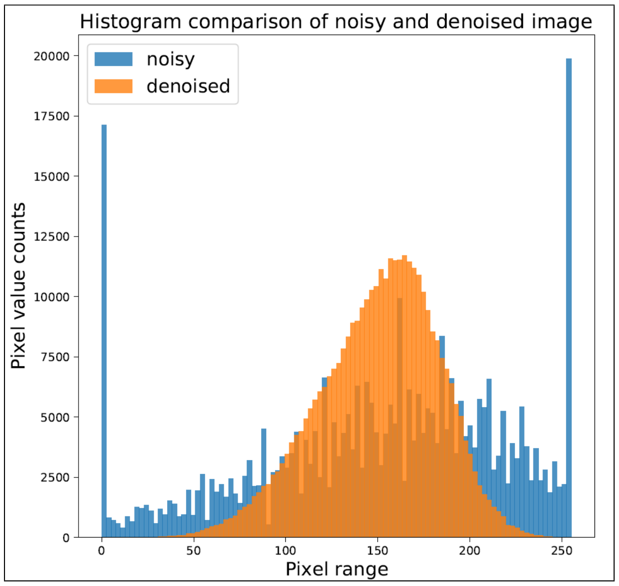


Fig. 4. Histograms of pixel values of an image before and after FFT denoising.

the denoised images at the same scale in terms of the signal to noise ratio (SNR). On the other hand, Figure 4, represents a histogram distribution comparison, of a single image, that shows how the FFT impacted the noise removal within an image.

## V. Transfer Learning based Detection

### A. *Autoencoder based Feature Learning from Unlabeled Data*

An autoencoder is a self-supervised learning model that is trained to output a recreation of its input. They typically comprise of an encoder block, which reduces the input's dimensions, and a decoder block, which reconstructs the input back from the lower-dimensional representation [18]. We used a convolutional autoencoder, which is an autoencoder that uses convolutional layers in the encoder and decoder blocks and is therefore more effective at reconstructing images [17]. Figure 5 shows the typical structure of a convolutional autoencoder. Lu et al. obtained favorable results by training an autoencoder on a larger unlabeled dataset before using the convolutional layers in a classification model [19].

This approach seemed likely to help with our difficulties caused by our limited supply of labeled data and the unusual structure of the images we were modeling. Training an autoencoder allowed us to leverage our relatively large supply of unlabeled images to acquire a model which had been pre-trained on images from our domain rather than attempting to perform transfer learning from models trained on the ImageNet dataset. By learning how to reconstruct input images from our dataset, the autoencoder learned the images' important features. We hypothesized that if an autoencoder would learn to reconstruct images from our domain, it would also learn a high-level representation of gravity wave patterns which could be extracted from the encoded representation of the images. This knowledge can then be transferred to a classification model.

We converted the raw hdf5 files into images and trained a convolutional autoencoder on these images. The autoencoder input data for this study is a three-dimensional array with the dimensions height x width x channel (256, 256, 1). The encoder block consists of three sets of alternating convolution and max pooling layers. The convolutional layers all have kernels of size 3 x 3, and the max pooling layers all have kernels of size 2 x 2. For all convolution layers, padding is set to "same," the activation function is ReLU, and the layers have 16, 8, and 8 filters respectively.

The decoder block consists of three sets of alternating convolution and upsampling layers, which ends with a convolution layer that outputs an image of the input image's dimensions. The convolutional layers have the same kernel size, padding, and activation functions as in the encoder block, and the order of the filters would be reversed with the layers having 8, 8, and 16 filters respectively. The autoencoder uses the Adam optimizer and the loss binary cross-entropy [20]. It was then trained over 100 epochs with a batch size of 32.

The classification model used the encoder block of the autoencoder after it had been trained on the unlabeled data. The layers in the encoder block were frozen, but each convolutional layer was given an l2 regularizer (l = 0.01) and the UnitNorm kernel constraint. The encoder layer of 64 x 64 x 8 was flattened into a dense layer of 32768 elements. Two dense layers of 256 and 64 units were added, each with ReLU activation and followed by dropout layers with rate 0.5. Dropout layers would reduce overfitting by randomly dropping a certain percentage of the input layer during training [21]. We found that for the autoencoder, dropping 50% of each dense layer yielded the best results for the autoencoder. The output layer has one neuron, for binary classification, and a sigmoid activation function. This model also used the Adam optimizer and binary cross-entropy loss, and was trained over 100 epochs with a batch size of 32. Figure 6 shows the architecture of the classification model.

### B. *Customization of Pre-trained Models*

Training a new model from scratch would require a large quantity of labeled training data. One of the challenges for this research was about working with a dataset that was relatively small, because no more data or datasets were available at the time of research. For this reason, it was necessary to work around the problem of using small dataset. We decided to create a custom

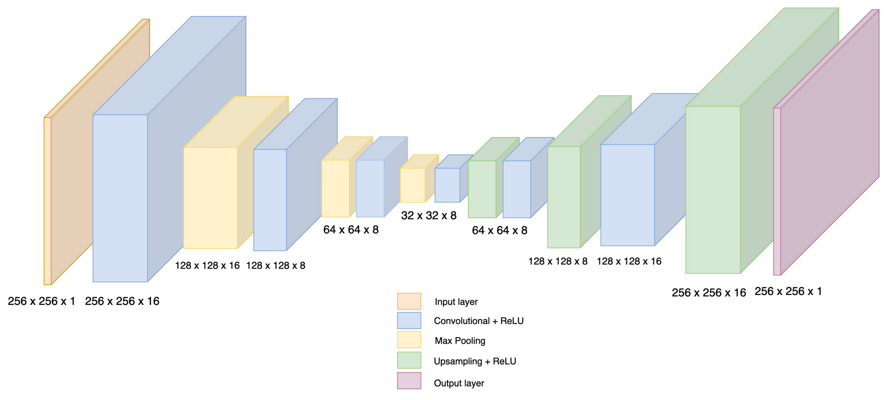


Fig. 5. Architecture of the convolutional autoencoder [17].

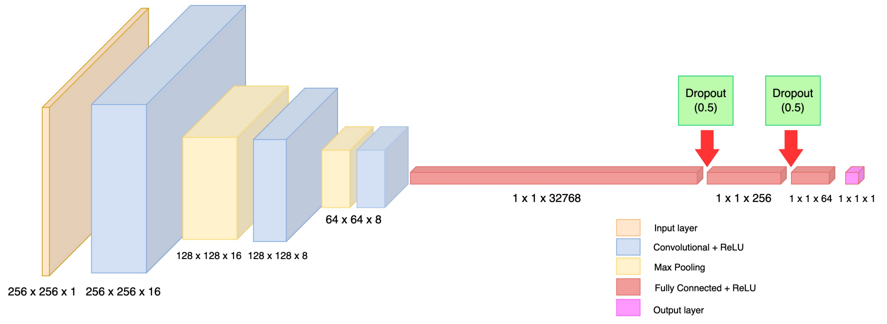


Fig. 6. Structure of the classification model. The left half is from convolutional autoencoder.

model that would require some parts of a pre-trained model together with a trainable custom classifier.

The underlying principle of this concept is to employ the transfer learning technique to apply the features that a pre-trained model learns while solving a problem, and use those learned features to apply it to the task at hand for this research, to acquire better results.

Since the pre-trained model InceptionV3 has already been trained on a large and general dataset, we may use that model's learned features instead of creating a new model from scratch.

This is because a deep learning model's initial layers are able to identify simple shapes, while later layers can identify more intricate visual patterns, and the last layer can be employed to create predictions.

We can reuse the majority of the pre-trained InceptionV3 model layers because employing the same low-level visual patterns is a need for any task. By doing so, we may overcome the issue with the limitations of only having a small dataset at our disposal, by building and training a customized model that is tailored for identifying gravity waves from our dataset.

The InceptionV3 [22] was chosen as the base model because it outperformed other state-of-the-art architectures.

The ImageNet dataset, which is a dataset contains fourteen million annotated images in over 20,000 categories, was used to train the InceptionV3 image recognition model. As shown in Figure 7, InceptionV3's deep learning network consists of 11 concatenated layers, or modules, named from "mixed0" to "mixed10". Convolutions, average pooling, max pooling, and other layers are included in each module. Following each convolution layer is a batch normalization and an activation input, which is typically "ReLU", which stands for Rectified Linear Activation Function. Lower layers detect simple patterns, while higher layers detect increasingly complex patterns.

We configured the InceptionV3 Model to exclude the top classification layers and made the pre-trained model's layers non-trainable, allowing us to use the model's network as an arbitrary feature extractor. By stopping at a specific layer, we can take the final layer's output as the features learned from the pre-trained model's neural network. The concatenated layer 'mixed7' was chosen as the final layer since it was the last module to preserve a big feature map (14x14), comprising of low and mid-level features.

Any additional layers would have preserved a 6x6 feature map instead, resulting in a smaller map with high level features that would only have brought specific details from the ImageNet dataset but would not have

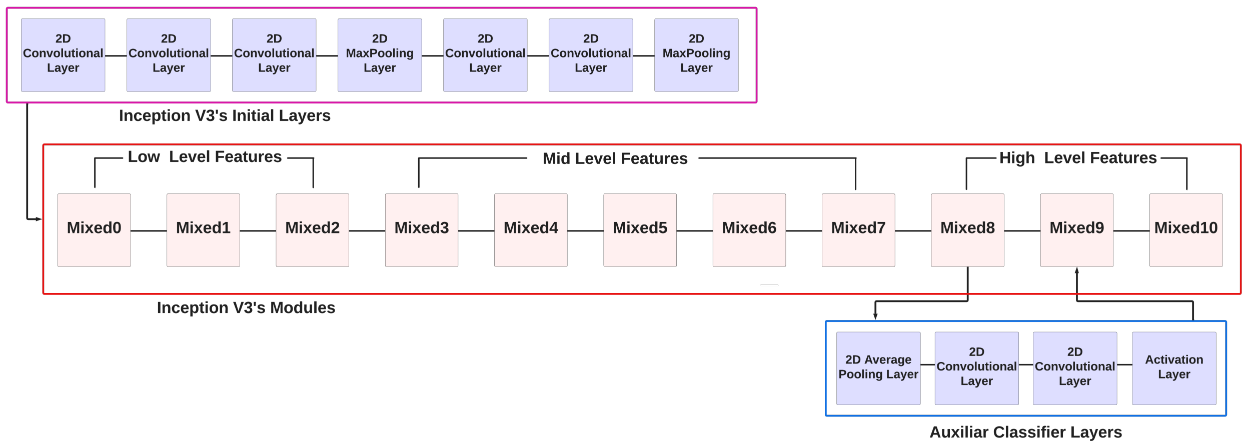


Fig. 7. Structure of the pre-trained model InceptionV3 [22].

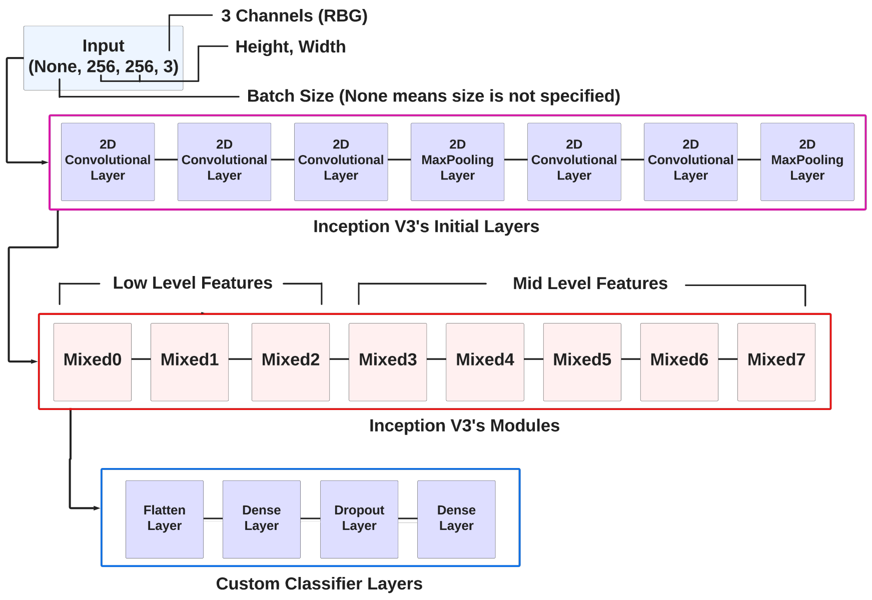


Fig. 8. Structure of the Customized Model.

helped us with gravity wave classification.

We added L1 and L2 Regularizers with 0.0001 values to each 2D convolutional layer in the InceptionV3 model to reduce overfitting. We added one Flatten layer on top of the final layer of the model to convert the multidimensional input to one-dimensional. A Dense layer of 1024 units was added to define the dense layer's output, followed by a Dropout layer of 0.3, which meant that 30% of the inputs would be randomly excluded from each update cycle. Finally, a Dense layer with a "sigmoid" activation was added to check only the labels that could be 0 or 1. Figure 8 depicts the custom layers.

With an initial learning rate of 1e-4, we used 'Adam' as the optimizer for the custom model. The metrics were from the accuracy class, and the loss function was binary cross-entropy. A learning rate scheduler called "ReduceL-ROnPlateau" was also added as a callback function. If the accuracy or loss value would not improve, the learning rate would be reduced by this function.

## VI. Experiments

We have 1086 image files in total for the experiment equally divided in 'gravity' and 'non gravity' classes. Figure 9 shows a real life example of an actual gravity and non gravity image.

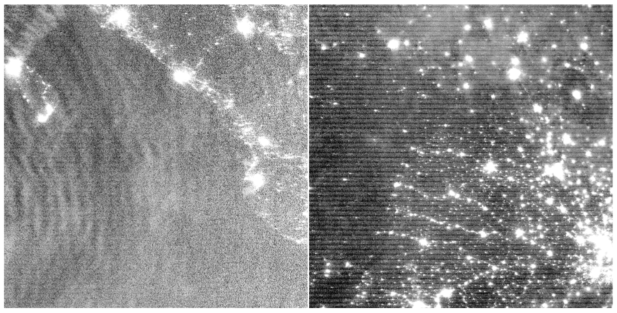


Fig. 9. An example of gravity wave image (left) and non gravity wave image (right).

For the experiment, we used UMBC's High Performance Computing Facility (HCPF) [23] and google colab as the operating platform. We used Python 3.7 along with tensorflow 2.4.0, keras 2.9.0, numpy 1.18.1, scikit-learn 0.23, Pandas 1.1.0, h5py 2.10.0, and Pillow 7.1.0 as the supporting libraries.

### A. Comparison Between Pre-Trained Computer Vision Models

To see how leading architectures performed on our data, we first ran several pretrained models on the same preprocessed dataset. ResNet50 is a deep CNN that achieved high accuracy on ImageNet by utilizing skip connections [24]. EfficientNetV2 is a small CNN that trains very quickly [25]. VGG16 is a 16-layer CNN that employs a series of small 3x3 filters [26]. As shown in Table I, the original InveptionV3 model outperformed all other tested base models.

We also compared Lai et al.'s [9] model, which consists of ten layers: an input, two CNN, two pooling, three dropout, one flatten, and finally a dense layer. The model performed poorly when tested with our dataset (shown in the Table I). Possible explanations for such low scores include a lack of data while training the model. We chose InceptionV3 as the base model for our custom model after considering all possibilities.

TABLE I
PERFORMANCE FROM BASELINES AND PRE-TRAINED MODELS

| Model | Train Acc. | Val Acc. | Test Acc. | F1 Score |
|---|---|---|---|---|
| ResNet50 | 1.0000 | 0.5000 | 0.5508 | 0.2418 |
| EfficientNet | 0.5507 | 0.6643 | 0.6525 | 0.5922 |
| VGG16 | 0.5104 | 0.5156 | 0.5593 | 0.2637 |
| InceptionV3 | 0.9394 | 0.7286 | 0.6949 | 0.4672 |
| CNN model [9] | 0.5900 | 0.5000 | 0.5800 | 0.0000 |

### B. Effects of AutoEncoder Approach

Table II shows the effects of transfer learning using an autoencoder pre-trained on unlabeled data, as explained in Section V-A. To make a comparison of the autoencoder model's performance, we constructed two models using the same architecture: the convolutional layers were the encoder layers of the autoencoder, and the dense layers were also the same. For the first model, we chose randomly initialized weights for the convolution layers. For the second model, we loaded the pre-trained autoencoder weights for the convolution layers. Both models were trained on the same dataset.

The results show that without any pre-training, the model predicts only the "non gravity waves" class. On an evenly split dataset, the training, validation, and test accuracies remained at 0.5 for both classes. Clearly, the baseline model was not learning. The train accuracy increased to over 97% when the autoencoder's pre-trained weights were used, and the validation and test accuracies increased to 70%. The pretrained model outperformed the randomly initialized baseline despite being overfitted.

TABLE II
PERFORMANCE FROM THE MODELS WITH AND WITHOUT THE AUTOENCODER TRANSFER

| Model | Train Acc. | Val Acc. | Test Acc. | F1 Score |
|---|---|---|---|---|
| Without Autoencoder | 0.5000 | 0.5000 | 0.5000 | 0.0000 |
| Autoencoder pretraining | 0.9753 | 0.7000 | 0.6992 | 0.7296 |

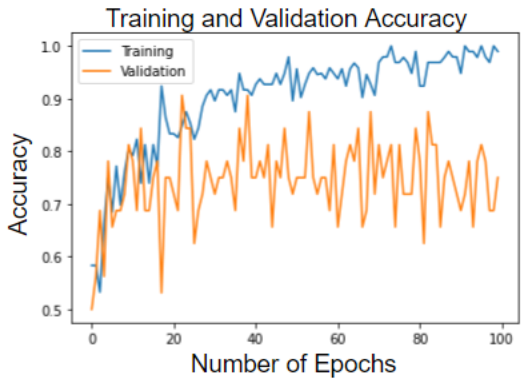
### C. Effects of Model Customization Approaches

Table III shows the performance of the custom models that used a pre-trained model for transfer learning, explained in Section V-B. The two models in the table share the same structure. The only difference is that the learning rate from the second model is decreased by the ReduceLROnPlateau function if the validation loss does not improve during training. The model using the learning rate scheduler outperformed the best performing baseline model, which was the pre-trained InceptionV3 architecture in Table I, by 6.36% in terms of test accuracy.

The results showed that both custom models could predict the "gravity waves" and "non gravity waves" classes. For the dataset that was split evenly for both classes, we can see that for the first customized model, which a constant learning rate of 0.0001, the training accuracy could achieve the 100 percent mark, but the validation accuracy, test accuracy and F1 Scores were still lower compared to the second customized model. As shown in Figure 10 and Figure 11, the overfitting has been reduced for the second approach of the custom model. Figure 12 shows how the learning rate for the model changes when ReduceLROnPlateau was activated.

TABLE III
PERFORMANCE FROM THE CUSTOM MODELS

| Model | Train Acc. | Val Acc. | Test Acc. | F1 Score |
|---|---|---|---|---|
| Constant LR | 1.0000 | 0.9063 | 0.6144 | 0.8800 |
| Changing LR | 0.9506 | 0.9499 | 0.7585 | 0.8181 |



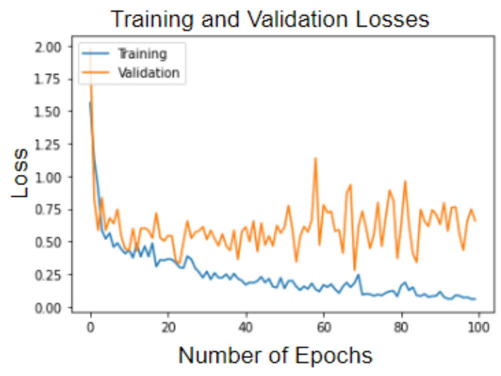
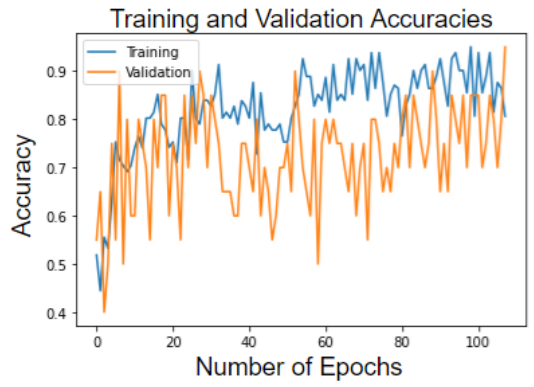


Fig. 10. Two plots showing the performance of the Custom Model with a Constant Learning Rate during the training.



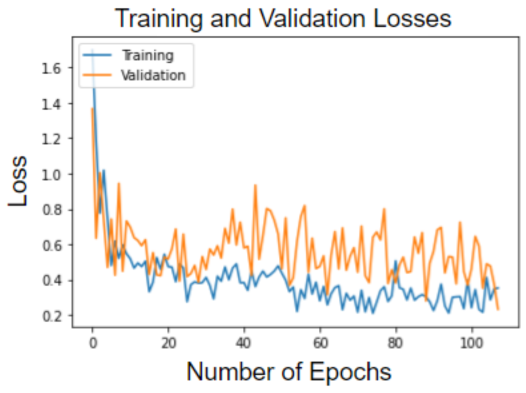


Fig. 11. Two plots showing the performance of the Custom Model with a Changing Learning Rate during the training.

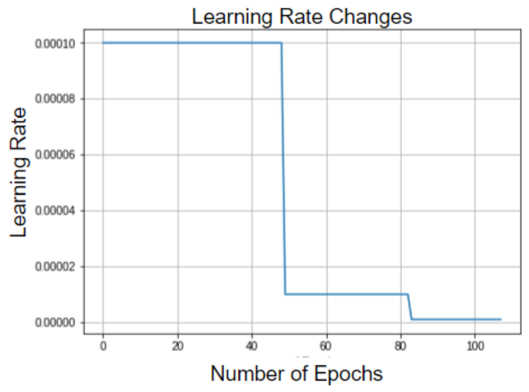


Fig. 12. Plot showing the learning rate changes during the training for the custom model that uses ReduceLROnPlateau.

### *D. Effects of FFT Denoising*

To deal with the noise in the dataset images, we implemented a denoising method that reconstructs cleaner images using the Fast Fourier Transform technique. Table IV compares the performance of our models when using noisy and denoised datasets.

With the exception of the randomly initialized model, which consistently predicts only one class, the results show that the majority of the models had a higher F1 score when classifying the denoised dataset.

TABLE IV
PERFORMANCES IN MODELS WITH NOISY AND FFT DENOISED DATA

| Model | F1 Score Noisy | F1 Score Denoised |
|---|---|---|
| Random Initialization | 0.0000 | 0.0000 |
| Autoencoder pretraining | 0.6577 | 0.7296 |
| Constant LR | 0.5714 | 0.8800 |
| Changing LR | 0.7500 | 0.8181 |

### *E. Discussion*

By referring to the results above, it is clear that using a state-of-the-art pre-trained model yields better results than using an autoencoder for transfer learning.

Even though the autoencoder was trained using data more relevant to our dataset, it was trained in less time and with a smaller dataset, unlike the InceptionV3. We discovered that when the autoencoder bottleneck gets too tight, the decoder stops reproducing gravity waves. This indicated that gravity wave patterns were difficult for the autoencoder to compress, whereas the other satellite image features were easier to compress. Because gravity wave shapes vary and are unstructured, more research may be required to address these concerns.

The custom models had the highest validation and test accuracies, with the model that used the learning rate scheduler being the only one to achieve the goal of reaching 95% validation accuracy. This demonstrates that combining the transfer learning method with a pre-trained model aided the custom model in predicting gravity waves. A larger dataset of satellite images would have allowed the model to learn more accurate general features and further reduce overfitting.

## VII. Conclusions

We presented preprocessing methods for converting raw HDF5 data from our chosen domain into usable images, as well as evaluated the performance of two transfer learning methods on our classification dataset. Our autoencoder method enabled us to use our large supply of unlabeled image data to identify meaningful structures within our images, which we then used to perform the classification task. Our customized InceptionV3 model used the most important parts of a cutting-edge architecture while adding our own layers and modifications that best fit our needs.

We anticipate that these findings will be useful for classifying gravity waves in small datasets of noisy satellite images, as well as for future gravity wave classification research. This work may also be useful for other transfer learning efforts, particularly those involving small datasets.

In the future, we will investigate how our models perform on datasets from other satellites, address overfitting issues, investigate other promising transfer learning models, and refine fake data generation techniques to compensate for our lack of labeled data.

## Acknowledgments

This work is supported by the NSF grant "REU Site: Online Interdisciplinary Big Data Analytics in Science and Engineering" (OAC–2050943) and the NASA grant "Machine Learning based Automatic Detection of Upper Atmosphere Gravity Waves from NASA Satellite Images" (80NSSC22K0641).

## References

[1] A. Mann, "To improve weather and climate models, researchers are chasing atmospheric gravity waves," *EARTH, ATMOSPHERIC, AND PLANETARY SCIENCES*, 2019.

[2] D. C. Fritts and M. J. Alexander, "Gravity wave dynamics and effects in the middle atmosphere," *Reviews of Geophysics*, vol. 41, no. 1, 2003. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2001RG000106

[3] D. Moran, "What are gravity waves?" https://www.weather.gov/source/zhu/ZHU_Training_Page/Miscellaneous/gravity_wave/gravity_wave.html, accessed: 2022-07-17.

[4] "Github repository for atmospheric gravity wave detection using transfer learning techniques." https://github.com/big-data-lab-umbc/big-data-reu/tree/main/2022-projects/team-1, [Online; Accessed: 2022-07-30 ].

[5] F. Zink and R. A. Vincent, "Wavelet analysis of stratospheric gravity wave packets over macquarie island: 2. intermittency and mean-flow accelerations," *Journal of Geophysical Research*, 2001.

[6] T. Colligan, J. Fowler, J. Godfrey, and C. Spangrude, "Detection of stratospheric gravity waves induced by the total solar eclipse of july 2, 2019," *Sci Rep 10*, 2020.

[7] P. Coïsson, P. Lognonné, D. Walwer, and L. M. Rolland, "First tsunami gravity wave detection in ionospheric radio occultation data," *Earth and Space Science*, vol. 2, no. 1, pp. 125–133, 2015.

[8] S. E. Koch and C. O'Handley, "Operational forecasting and detection of mesoscale gravity waves," *Weather and Forecasting*, vol. 12, no. 2, pp. 253–281, 1997.

[9] C. Lai, J. Xu, J. Yue, W. Yuan, X. Liu, W. Li, and Q. Li, "Automatic extraction of gravity waves from all-sky airglow image based on machine learning," *Remote Sensing*, vol. 11, no. 13, p. 1516, 2019.

[10] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[11] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *CoRR*, vol. abs/1911.02685, 2019. [Online]. Available: http://arxiv.org/abs/1911.02685

[12] J. Blitzer, M. Dredze, and F. C. Pereira, "Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification," in *ACL*, 2007.

[13] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *ICML*, 2011.

[14] NASA LAADS DAAC, "VNP02DNB Data Product: VIIRS/NPP Day/Night Band 6-Min L1B Swath 750m," https://ladsweb.modaps.eosdis.nasa.gov/missions-and-measurements/products/VNP02DNB, DOI:10.5067/VIIRS/VNP02DNB.002, accessed August 2, 2022.

[15] "The hierarchical data format version 5 (hdf5)," https://portal.hdfgroup.org/display/HDF5, [Online; Accessed: 2022-06-30 ].

[16] S. W. Smith *et al.*, "The scientist and engineer's guide to digital signal processing," 1997.

[17] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in *International conference on neural information processing*. Springer, 2017, pp. 373–382.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] J. Lu, N. Verma, and N. K. Jha, "Convolutional autoencoder-based transfer learning for multi-task image inferences," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 1045–1057, 2022.

[20] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, pp. 19–67, 2005.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research 15*, pp. 1929–1958, 2014.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[23] "UMBC High Performance Computing Facility," https://hpcf.umbc.edu/, accessed August 2, 2022.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[25] M. Tan and Q. V. Le, "Efficientnetv2: Smaller models and faster training," *ArXiv*, vol. abs/2104.00298, 2021.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.