Exploring Machine Learning based Atmospheric Gravity Wave Detection

REU Site: Online Interdisciplinary Big Data Analytics in Science and Engineering
Jorge López González[1], Theodore Chapman[2], Kathryn Chen[3], Hannah Nguyen[4], Logan Chambers[5], Seraj A.M. Mostafa[6], Dr. Jianwu Wang[6], Dr. Sanjay Purushotham[6], Dr. Chenxi Wang[7,8], Dr. Jia Yue[7,9]
[1]Department of Computer Science, University of Puerto Rico, Rio Piedras
Email: jorge.lopez19@upr.edu
[2]Goergen Institute of Data Science, University of Rochester
Email: tchapma6@u.rochester.edu
[3]Department of Statistics and Applied Probability, University of California, Santa Barbara
Email: kathryn_chen@ucsb.edu
[4]Department of Biological Sciences, University of Maryland, Baltimore County
Email: hannahn2@umbc.edu
[5] Department of Mathematics, Computer Science, and Physics, Albany State University
Email: lchambe3@students.asurams.edu
[6]Department of Information Systems, University of Maryland, Baltimore County
Email: {serajmost, jianwu, psanjay}@umbc.edu
[7]NASA Goddard Space Flight Center
Email:{chenxi.wang, jia.yue}@nasa.gov
[8]Goddard Earth Sciences Technology and Research (GESTAR) II, University of Maryland, Baltimore County
[9]Department of Physics, Catholic University of America

Technical Report HPCF–2022–11, hpcf.umbc.edu > Publications

Abstract

Atmospheric gravity waves are produced when gravity attempts to restore disturbances through stable layers in the atmosphere. This phenomena should be considered when predicting weather due to their association with weather fronts, wind currents, and extreme weather events. Despite their importance, little research has been conducted on how to computationally detect gravity waves. In this study, we explored various methods of preprocessing and transfer learning in order to work around the small size of our labeled dataset. We pre-trained an autoencoder on unlabeled data before training it to classify labeled data. We also created a CNN by combining certain pre-trained layers from the InceptionV3 Model trained on ImageNet with custom layers and a custom learning rate scheduler.

Key words. atmospheric gravity waves, machine learning, image denoising, transfer learning, custom model

1 Introduction

Gravity waves are generated when parcels of air are displaced from an equilibrated position and the force of gravity tries to restore the equilibrium. Atmospheric gravity waves are disturbances generated by various types of disturbances in the troposphere, including airflow from over mountains, jet streams, and thunderstorms. These disturbances can happen when the air is forced to rise upwards in stable air, creating a wave pattern while the air sinks back down over time, in a similar way that ripples are formed by a stone tossed into a still water surface [1]. Gravity waves can

sometimes be detected using radar or satellites, and inside images they can be seen as producing patterns shaped like ripples or clouds.

When present, gravity waves have a major effect on many weather phenomena. Simulations of middle atmosphere circulation are more accurate when correctly factoring in gravity waves, indicating that they influence circulation. It is theorized that they may even affect tidal waves [2]. Gravity waves also produce a type of turbulence called Clear Air Turbulence; up to 40% of all aviation accidents can be attributed to this kind of turbulence [3].

Due to increased understanding of the importance of gravity waves in recent years, interest in detecting and understanding them have drastically increased. Some researchers are turning to machine learning techniques to grasp a better understanding of this phenomena.

We worked on machine learning techniques to detect gravity waves in the atmosphere. For this paper, we address the two main challenges regarding the detection of gravity waves from satellite images. The first main challenge was working with a noisy dataset. Because the images were scanned from a satellite, the images could have been corrupted when acquired by a defective sensor, transmitted by a faulty channel, or other factors that could degraded the quality of the scanned images. The problem that we could expect from trying to use noisy images for training is that the data could not get understood or interpreted correctly by the learning model. The second main challenge was to deal with a small dataset that was provided to work with the gravity waves classification problem, a dataset that consisted of 710 images for training, 140 images for validation, and 236 images for testing. The problem when dealing with a small dataset is that the model gets fewer examples to learn general features from. If a model is trained on data that is too specific, then it will not be able to learn generalized features well and will increase the risk of over fitting.

To tackle the above two challenges, we explored various techniques and our contributions are summarized below. We also open sourced our implementation and the source code can be accessed at the Big Data REU GitHub repository [4].

- We explored different techniques to signify the signals and remove noises in our datasets. Because the radiation signals in our raw data are 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 steps, gravity waves are much more visible, and both signal to noise ratios (SNR) and pixel distributions are greatly improved.

- We proposed solutions using different types of transfer learning methods, such as autoencoders and custom models using a pre-trained model for feature extraction, before transferring that knowledge to a model that classifies whether or not the images contain gravity waves.

- We worked on several experiments to evaluate the performance of all the models we implemented for training and testing. The results show the comparison of the accuracy scores between all models, and present the ability of using deep learning methods to predict the presence of gravity waves on scanned satellite images.

For the rest of the paper, the organization is as follows. We present related work regarding studies that approach similar problems relatable to this research in Section 2. This is followed by showing the dataset and the techniques used for the preprocessing of the images in Section 4. Details are provided about the methods used for the approaches of using an autoencoder and a custom model, for training and testing, in Section 5. The discussion and results from the experiments are detailed in Section 6. Section 7 explained additional efforts that are in progress. The conclusion of this paper is found in Section 8.

2 Background and Related Works

2.1 Gravity Wave Detection

Most existing methods for gravity wave detection require researchers to take measurements of features from the atmosphere in the target region. For example, studies by Zink et al. [5] and Colligan et al. [6] identified gravity waves based on measurements of horizontal wind speed collected through radiosonde sounding. They then applied a linear transformation to the measurements to make them more directly intrepretable. In order to isolate the superimposed gravity waves, they scanned the resulting function for local maxima and recorded the wave packet data at those points. That data was recorded and further analyzed with Stokes parameter analysis to determine wave direction, speed, and height- all of which help identify gravity waves. This general technique produces fairly accurate results, but it requires access to various weather predicting instruments and can only detect gravity waves in a small area.

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 which suggested that the waves were not from the ionosphere and were instead gravity waves excited by tsunamis. Koch et al. proposed an automatic mesoscale gravity wave detection system in 1997 [8].

Gravity wave detection can be made much more efficient and accessible using the bountiful and publicly available satellite image data. Thus far, very little research has been conducted yet about how to detect them using artificial intelligence. There is a notable study by Lai et al. that developed a convolutional neural network based program that extracts gravity wave patterns in all-sky airglow images [9]. To achieve this, they used a convolutional neural network to classify images of clear skies and unwarp them onto geographic maps. Then, the gravity waves were 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 an accurate model that could accurately classify a small dataset of satellite images with 95% validation accuracy.

2.2 Transfer Learning

Transfer learning involves training a model on two tasks, typically referred to as the source 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 that the model is trained in [10]. The strategy of transfer learning covers a wide variety of approaches which are often further subdivided. One popular taxonomy categorizes approaches according to what sort of labels are 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 Imagenet dataset to categorizing grayscale satellite imagery. This is referred to as "inductive transfer learning" because the training on the source domain is 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, so that the features, which are important to understanding samples from the source domain, tend to also be informative in 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 is to perform unsupervised feature extraction on unlabeled images from a domain very similar to the target domain. Typically such approaches will 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. use an unsupervised autoencoder model to transform textual reviews into compressed summaries which they then used as input to sentiment classifiers, resulting in significantly improved classification performance [13].

3 Overall Pipeline

Figure 3.1 shows the overall pipeline we used. The pipeline starts with raw data in the form of hdf5 files supplied by the NASA Soumi NPP satellite. We then 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 5.1), we leveraged the unlabeled images to train an autoencoder. We then 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 5.2), we cut off the Inception V3 model pretrained on ImageNet at the last layer with the larger (14x14) feature map. We then added custom layers to this model and used transfer learning to train the model on the labeled dataset.

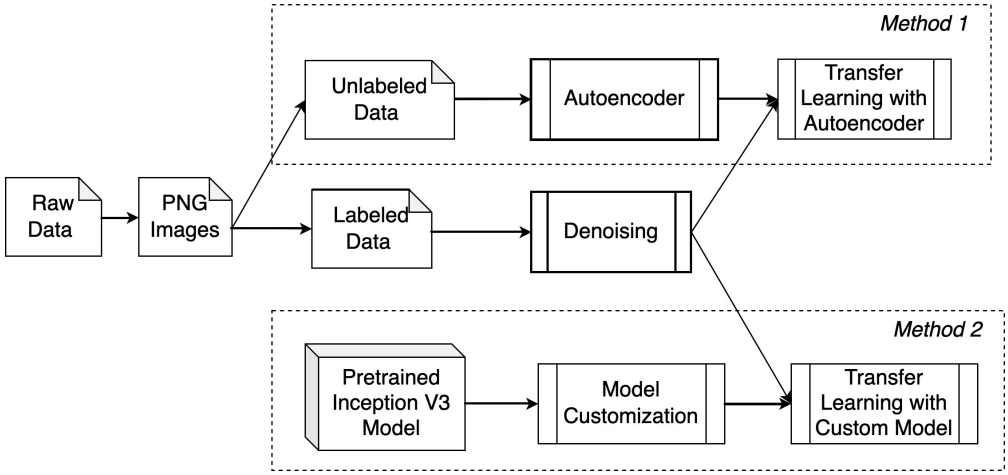

Figure 3.1: Diagram of the Overall Pipeline.

4 Data Preprocessing

In this study we used NASA VIIRS DNB (Day Night Band) images [14] collected from satellite Suomi NPP.

4.1 Image Generation From Raw Data

The raw data consisted of measurements of the radiance of light with wavelength 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 4.1 for an illustration. The resulting images were classified by hand and then further processed.

4.2 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 implemented this by taking the 2D Fourier Transform of the image and zeroing out all but the highest and lowest frequencies and finally taking the inverse Fourier transform of the remaining frequencies to produce a denoised image.

The main idea of Fourier transforms is to transform an image into the frequency domain by decomposing the image into sines and cosines of varying amplitudes and phases, which reveals repeating patterns within the image [15]. The Fourier Analysis is a special case of a concept called orthogonal functions. The main idea of the function is breaking down a complicated signal into a linear superposition of simpler "basis" functions to get the result for the original signal.

In this work all models are trained on denoised data as it offered an improvement in model performance (described in later section) compared to other denoising techniques such as image thresholding. See Algorithm 2 for a pseudocode implementation of the Fourier filtering used in this work.

At first, we applied the FFT (*fft2*) algorithm that returns the two-dimensional fourier transform matrix using a fast fourier transform algorithm. Later, we crop off all but the top and bottom 10%

Algorithm 1 HDF5 to PNG

Require: $arr \in \mathbb{R}^{1000x1000}$

$F(x) \leftarrow P(Z <= x)$ for $Z \sim$ Normal distribution fitted to the values of arr

$arr \leftarrow arr - min(arr)$

$arr \leftarrow \frac{arr}{median(arr)} * 0.5$

$arr \leftarrow clip(arr, 0, 1)$

$arr \leftarrow F(arr)$ ▷ Transform the approximately normally distributed values to uniform ones

$arr \leftarrow clip(arr, 0, 1)$

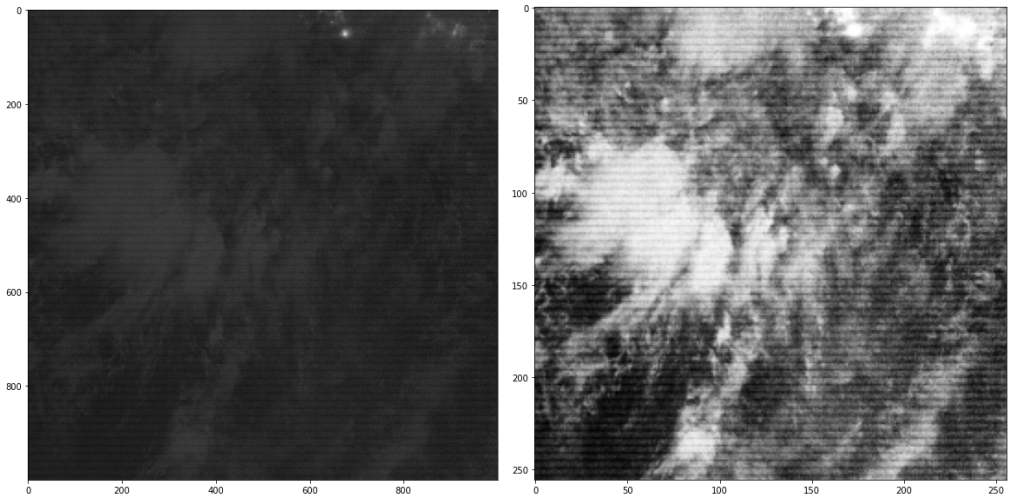

Figure 4.1: Normalized raw data versus preprocessed PNG

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: $I \leftarrow ifft2(i)$

of the signals row and column-wise to remove the unneeded frequency elements. The next steps are the reverse process of the first two steps where we reconstruct the image from signals. We utilized python's *SciPy* package in this process. Figures 4.2 and 4.2 illustrate the impact of denoising.

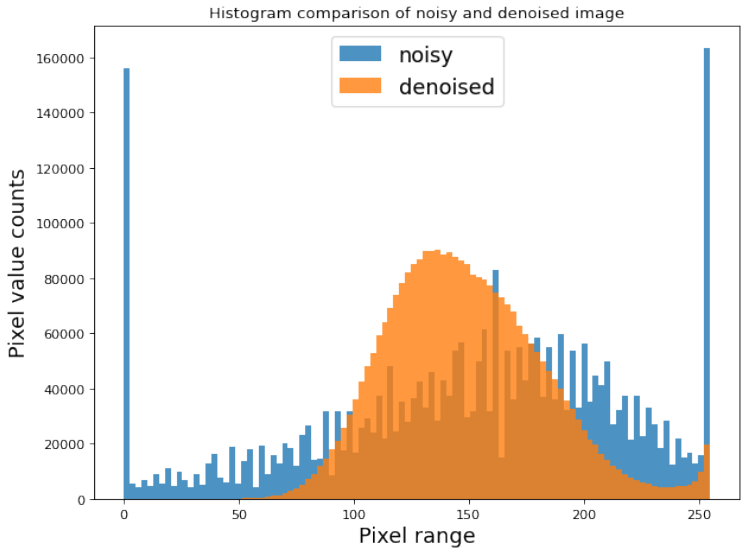

Figure 4.2: Histograms of pixel values of an image before and after FFT denoising

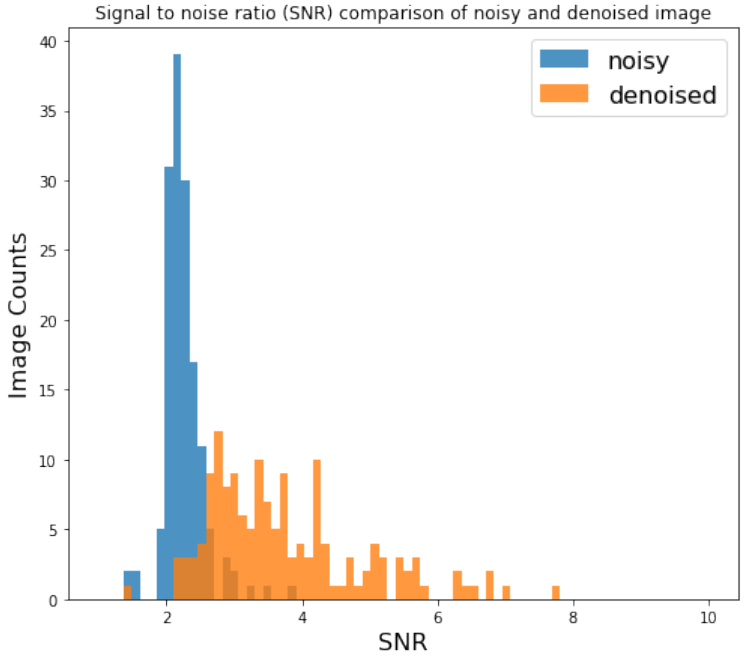

Figure 4.3: Signal to noise ratio (SNR) graph for images before and after denoising (N = 150)

5 Transfer Learning based Detection

5.1 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 [16]. 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 5.1 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 [18].

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 pretrained 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 learned 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 as detailed in section 3.2.1 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, then ends with a convolution layer that outputs an image of the input image's dimensions. The

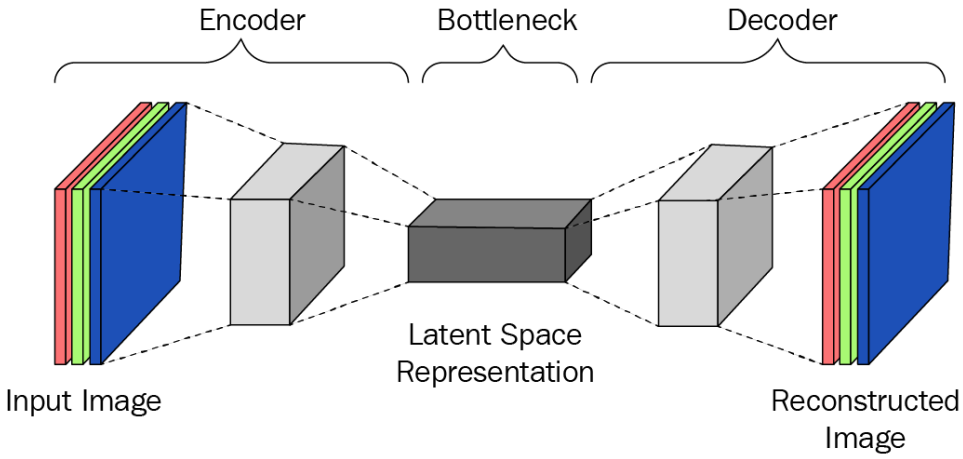

Figure 5.1: General structure of a convolutional autoencoder

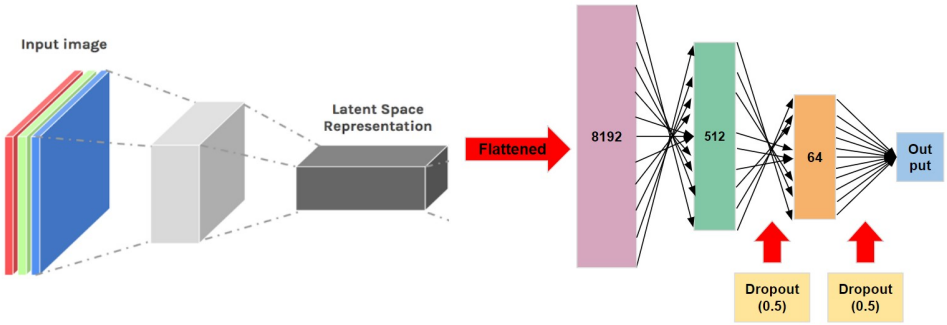

Figure 5.2: Structure of the classification model

convolutional layers have the same kernel size, padding, and activation functions as in the encoder block, and the order of the filters is reversed with the layers having 8, 8, and 16 filters respectively. The autoencoder uses the Adam optimizer and the loss binary cross-entropy [19]. 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 last autoencoder layer of 32 x 32 x 8 was flattened into a dense layer of 8192 elements. Two dense layers of 512 and 64 units were added, each followed by dropout layers with rate 0.5. Dropout layers reduce overfitting by randomly dropping a certain percentage of the input layer during training [20]. 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 5.2 shows the architecture of the classification model.

5.2 Customization of Pre-trained Models

Training a new model from scratch would requires a lot of labeled training data, but our challenge was to work with a dataset that was relatively small. To deal with this problem, we decided instead to create a custom model using some parts of a pre-trained model together with a trainable custom classifier.

The intuition behind this is because the pre-trained model has been already trained on a large and general dataset, thus it can serve as a generic model, and we can take advantage of the features that has been learned from the pre-trained model without training our own model from scratch. This way, we can create a custom model and train it to be specialized to identify gravity waves from our dataset, and thus resolve the problem we were dealing with by the limits of using a small dataset.

We used the Inception Model V3 as the base model because it performed well on our data compared to other state-of-the-art architectures. Inception Model V3 is an image recognition model that has been trained using the ImageNet dataset, which consists of millions of annotated images. As shown in Figure 5.3, Inception Model V3's deep learning network consists of 11 concatenated layers, or modules, named from "mixed0" to"mixed10." Each module includes layers like convolutions, average pooling, max pooling, and many more. Every convolution layer is followed by a batch normalization and applied to activation inputs, typically "ReLU", which stands for Rectified Linear Activation Function [21]. The lower layers detect simple patterns, and later layers detect

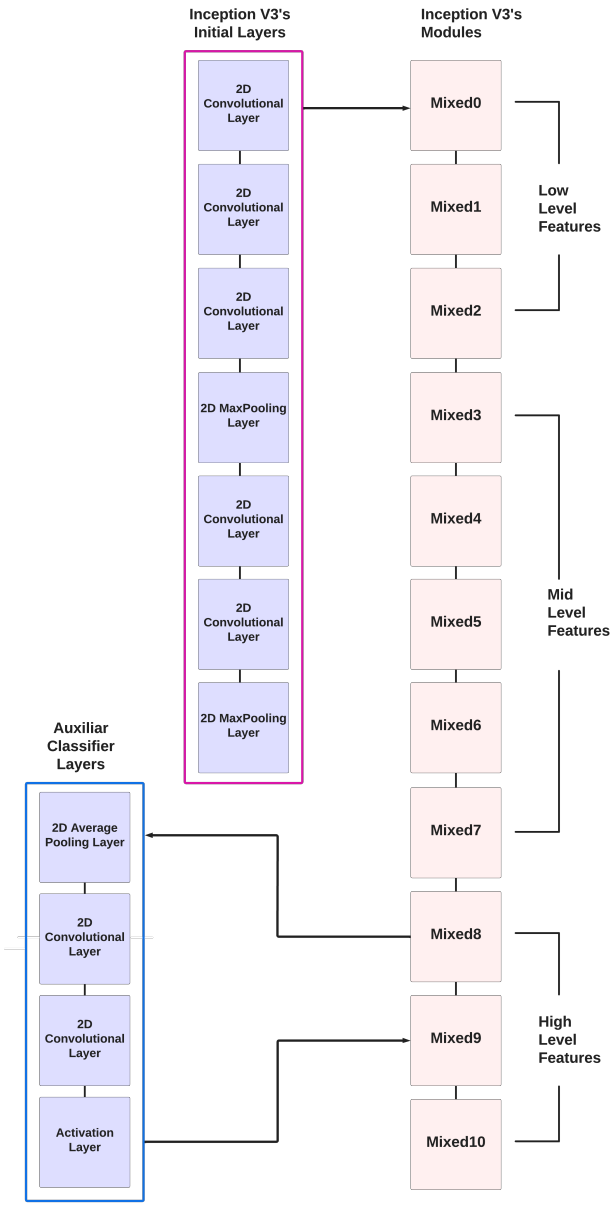

Figure 5.3: Architecture of Inception Model V3

progressively more complex patterns.

When we used the Inception Model, we specified to not include the classification layers at the top, and made the layers from the inception model non-trainable because we are only interested in using the pre-trained model for feature extraction. We specified the last layer of the model to be the concatenated layer 'mixed7', because it is the last module that keeps a large feature map (14x14). This keeps the model's ability to extract low and mid-level features. If we were to use any further layers, we would have a resulting 6x6 feature map which would be smaller and would contain high level features that are too specified for images from the ImageNet dataset, and would not help us for the classification of gravity waves.

For every 2D convolutional layer, we applied L1 and L2 Regularizers set to 0.0001. On top of

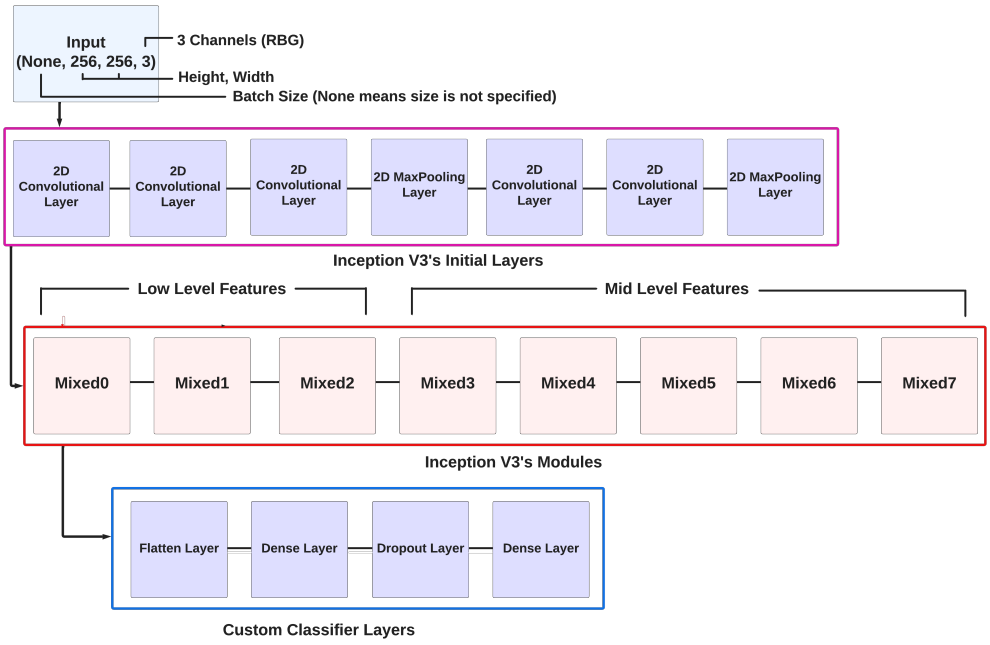

Figure 5.4: Custom Layers

the layers used from the Inception model, we applied layers to build a custom classifier on top that would be trainable with our dataset. For this we added a Flatten layer to make the multidimensional input one-dimensional. The we applied a Dense layer of 1024 units, which defines the output from the dense layer. After the fully connected layers we applied a Dropout layer of 0.3, meaning that 30% of inputs will be randomly excluded from each update cycle. At the end, we applied a Dense layer of 1 with a "sigmoid" activation, since we are looking only at labels that can be 0 or 1. The custom layers are shown in Figure 5.4.

We compiled the model using Adam as the optimizer, with an initial learning rate of 1e-4. The loss function was binary cross-entropy, and for the metrics we used the accuracy class. Additionally, we added a learning rate scheduler as a callback function which is known as "ReduceLROnPlateau." This function reduces the learning rate during training when there is no longer improvement in the accuracy or loss scores, and it is not required to define the number of epochs for the adjustment.

6 Experiments

Dataset. The dataset used in the experiments consisted of 710 images for training, 140 images for validation, and 236 images for testing.

Hardware/Software settings. For the hardware, we used Google Colab and UMBC's High Performance Computing Facility (HCPF) [22]. For the software environment, we used 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.

6.1 Comparison Between Pre-Trained Computer Vision Models

In order to see how leading architectures perform on our data, we first ran several pretrained models on the same preprocessed dataset. ResNet50 is a deep CNN that still achieved a high accuracy

on ImageNet through the use of skip connections [23]. EfficientNetV2 is a small CNN that trains very quickly [24]. VGG16 is a 16 layer CNN that uses a series of small 3x3 filters [25]. As shown in Table 6.1, the original InveptionV3 model significantly outperformed all other base models we tried. We also compared the model from Lai et. al. [9] which combines ten layers with an input, two CNN, two pooling, three dropout, one flatten and finally a dense layer. Testing the model with our dataset performs poorly (shown in the table). The poosible reasons of achieving the low scores are due to the noisy images and not having enough data to train the model. Considering all situations we chose InceptionV3 as the base model for our custom model.

Table 6.1: 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
EfficientNetV2L	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

6.2 Effects of AutoEncoder Approach

Table 6.2 shows the effects of transfer learning with an autoencoder pre-trained on unlabeled data which was explained in Section 5.1. We construct a model using the encoder part of the autoencoder and compare the model's performance if we choose the encoder weights by pretraining autoencoder against the same model with randomly initialized weights. For comparison, we also implemented a baseline model that uses the exact same architecture and trained on the same dataset, but randomly initialized weights.

The results show that without any pre-training, the model predicts only the "no gravity waves" class. On a evenly split dataset for the two classes, the training, validation, and test accuracies remained at 0.5. Clearly, the baseline model was not learning anything. When using the pre-trained weights from the autoencoder, the train accuracy rose to over 97%, and validation and test accuracies rose to 70%. Although the pretrained model was overfitted, it significantly outperformed the randomly initatialized baseline.

Table 6.2: Performance from the Models with and without the Autoencoder Transfer

Initialization Method	**Train Acc.**	**Val Acc.**	**Test Acc.**	**F1 Score**
Random Initialization	0.5000	0.5000	0.5000	0.0000
Autoencoder pretraining	0.9753	0.7000	0.6992	0.7296

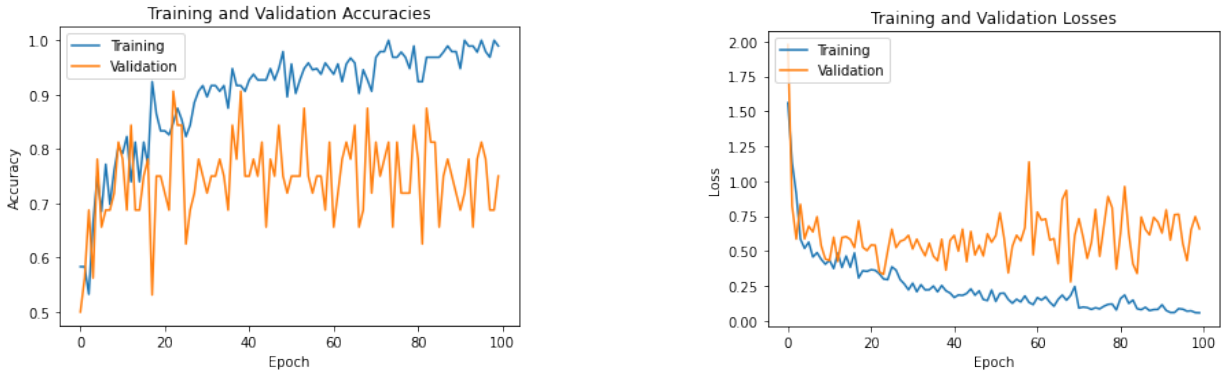

Figure 6.1: Two plots showing the performance of the Custom Model with Constant Learning Rate.

6.3 Effects of Model Customization Approaches

Table 6.3 shows the performance of the custom models that used a pre-trained model for transfer learning which was explained in Section 5.2. The two models in the table have the same structure. The only difference is that the learning rate of the second model is decreased by the ReduceLROnPlateau function when the validation loss is not improving during training. As the table demonstrates, the model that decreased learning rate outperformed the best performing baseline model, which is the pre-trained InceptionV3 architecture in Table 6.1, by 6.36% in terms of test accuracy.

The results showed that both custom models could predict the "gravity waves" and "no 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 6.3 and Figure 6.3, the overfitting has been reduced for the second approach of the custom Inception model. Figure 6.3 shows how the learning rate for the model changes when ReduceLROnPlateau is used.

Table 6.3: 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

6.4 Discussion

As shown by the above results, it appears that using a state-of-the-art pre-trained model yields better results than transfer learning from an autoencoder. One reason for this could be that the autoencoder, although trained on data that is far more relevant to our dataset, was trained for much less time and on a much smaller dataset compared to the InceptionV3 model that was trained on the much larger, much more varied ImageNet dataset. When examining the autoencoder, we also

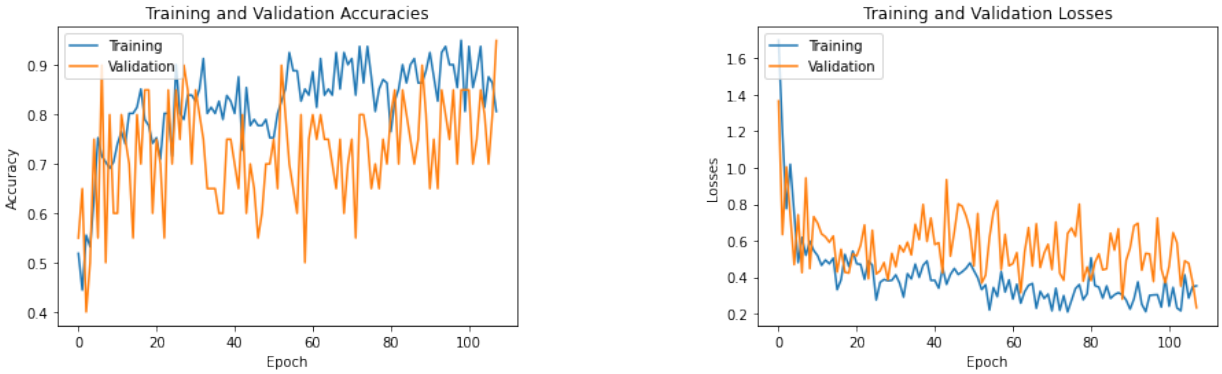

Figure 6.2: Two plots showing the performance of the Custom Model with Changing Learning Rate.

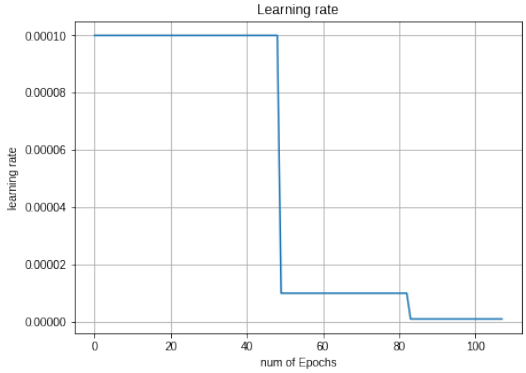

Figure 6.3: Plot showing the learning rate changes during the training for the custom model that uses ReduceLROnPlateau.

noticed that when the autoencoder bottleneck becomes too tight, the decoder stops reproducing gravity waves. This indicates that it is more difficult for the autoencoder to compress gravity wave patterns than other satellite image features, likely because of how varied and unstructured gravity waves are. Further exploration on using autoencoder transfer learning would have to address these concerns.

7 Other Efforts

7.1 Denoising Autoencoder Attempt

We tried to use a denoising autoencoder to remove noise from the images, but quickly abandoned the attempt when we realized that in order to successfully denoise images, the autoencoder would already need denoised versions of the images as labels.

7.2 Diffranet Preprocessing

Diffranet is a dataset of diffraction pattern images that comes with models that can classify said images with a high degree of accuracy [26]. The diffraction patterns in Diffranet show some similarities to the patterns of gravity waves, so we suspected that techniques that worked on the Diffranet dataset could also work on ours. After studying the source code of one of the models, Deepfreak, we suspected that the high accuracy of the Diffranet model was partially caused by preprocessing that better removed noise and more effectively showed their image's diffraction patterns. We studied their open source code and created a dataloader that preprocessed images the way Diffranet did. However, we saw no improvement in results. In retrospect, it is clear that although our data and the Diffranet data both contained diffraction patterns, there are many differences between the datasets that could have prevented Diffranet's preprocessing from working on ours. Not only that, but Diffranet's dataset had far more images and therefore could more easily attain a high accuracy regardless of the preprocessing used.

7.3 Horizontal Line Removal

Due to the nature of how the images are produced by the satellites, there is some "dead space" in each image that appears as a dark horizontal line. Each image has about 62 of these evenly spaced lines in them, and we believed these would act as noise, interfering with the model's ability to identify gravity waves. In attempt to remove the horizontal lines from each image, we used the fast fourier transform method to detect horizontal areas with darker brightness throughout the whole image. This was made easier by the fact that all the lines were spaced evenly, so we could predict about where each line would be. Then, the color of each pixel within the horizontal line line was changed so that it would be an average of the pixel above and below it. Although this method was effective in reducing appearance of the horizontal lines, they were not completely removed. Also, when training the model with the original dataset vs. the dataset with horizontal lines removed, there was no improvement in accuracy. We concluded that our method of removing the appearance of horizontal lines does not make a significant change in the model performance, but we believe it has potential if more work goes into it.

7.4 Pretraining on Synthetic Data

Our attempts to do transfer learning using the Imagenet dataset for our source task met with limited success. It seems likely that that was at least partially due to the fact that our images differ significantly from the images which appear in Imagenet.

We attempted to resolve this by creating a synthetic dataset using modified images from our target domain to make the model's understanding transfer more directly from the source task to the target one. We created a method to generate random wave patterns in the style of gravity waves and then overlayed them on existing unlabeled images to create a dataset of images in our domain with synthetic gravity waves with known locations. We then trained a model to localize synthetic gravity waves in hopes that that training would transfer effectively to our target task.

7.4.1 Synthetic Data

We created static validation and test sets from 0.1 and 0.2 of our unlabeled data, respectively. The training set was not explicitly stored on disk - instead every sample shown to the model was randomly generated, resulting in an approximately unlimited training dataset. We generated the synthetic wave patterns by randomly sampling a parameters controlling the size, period, brightness,

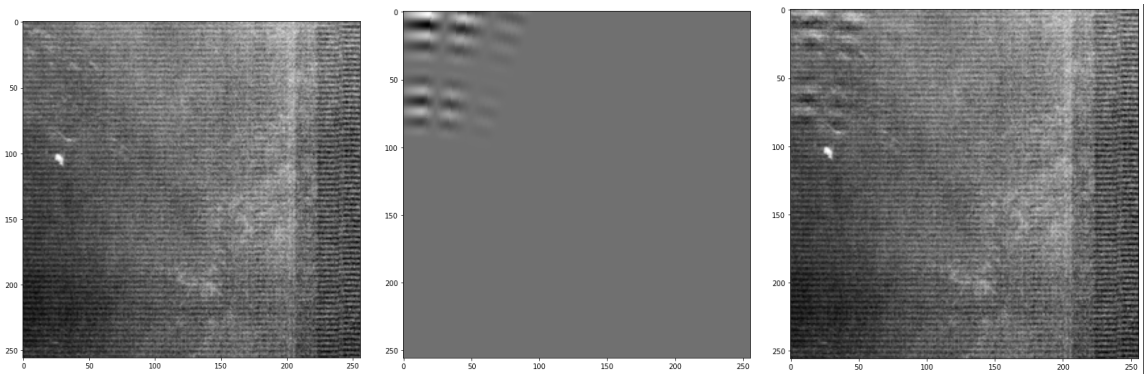

Figure 7.1: Left: unlabeled image. Center: generated wave pattern. Right: image after augmentation

orientation, and radius of curvature. Each generated pattern is a bounded portion of a circular wave pattern emanating outward from a fixed point source. See Figure 7.1

7.4.2 Training on the source task

Our model was an InceptionV3 model pretrained on Imagenet, with the output layer replaced by a Dense layer with 4 output parameters and a linear activation function. We trained it on the source task using the unlabeled data which was not part of the validation or test sets for 200 epochs, showing the model a total of 602,400 unique images. The loss was a mean squared error loss interpretting the 4 outputs as the locations of the leftmost, topmost, rightmost, and bottommost points in the added wave function (with all values normalized to lie between 0 and 1).

To evaluate the model, we measured its intersection over union (IOU) score. The IOU score of a localization model is the quotient of the number of pixels shared between the true bounding box and the predicted one (the intersection) and the number of pixels contained in either the true bounding box or the predicted one (the union).

The model reached a peak validation IOU of 0.53 and test IOU of 0.56. This sounds really unimpressive, but, due to limitations in the data generation process, the authors were unable to localize the modifications in many of the images. See 7.2 for an example of such an image. For images which the authors were able to solve, the model performed significantly better (test IOU $\sim$0.73). So we determined that the model had successfully learned the source task for purposes of transfer learning.

7.4.3 Transfer to the target task

Once the model started to overfit, we moved on to retraining it on the target task. We replaced the output layer with a Dense layer with a single output and sigmoid activation. Due to our limited data for the target task, we dropped all layers after the mixed7 layer in our trained InceptionV3, froze all remaining layers and added a small head which we retrained on the target task. The result was that the pretraining did not improve the outcome relative to doing only the transfer step without training on the synthetic dataset. The ultimate performance was not significantly better than using the same model without pretraining on the synthetic dataset.

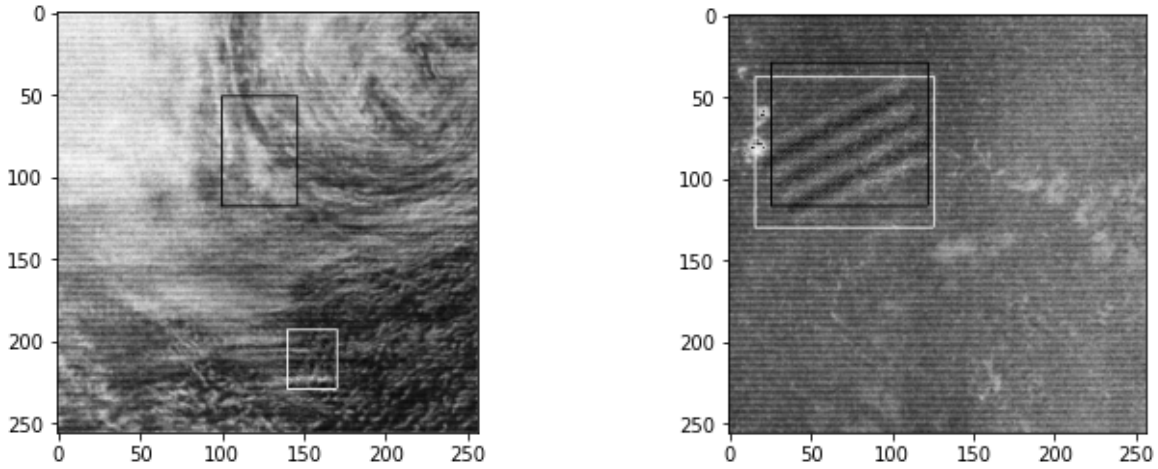

Figure 7.2: Left: unsolvable training image. Right: solvable image. The white border shows the 'correct' answer while the black is the predicted answer

Table 7.1: Synthetic Data Transfer

Model	**Train Acc.**	**Val Acc.**	**Test Acc.**	**F1 Score**
With Synthetic Transfer	0.9900	0.8833	0.7600	0.7843
Without Synthetic Transfer	1.0000	0.8167	0.7100	0.7027

7.4.4 Analysis

This approach hinged on the assumption that learning to localize synthetic gravity waves would require the model to learn to both ignore irrelevant patterns (clouds, city lights, noise, ...) and to attend to wave patterns. The former seems likely to be a valid assumption; those features are just as extraneous in the localization task as in the synthetic one. However, the latter assumption has a number of significant flaws. The most significant limitation to this approach is that the generated gravity waves are not nearly as diverse as the real ones. Thus, the model might learn to attend to the relevant patterns present in all of the synthetic gravity waves while only learning to recognize a small fraction of ways that real gravity waves can appear. For this to work, the space of possible synthetic gravity waves needs to be a superset of possible real gravity waves (no harm is done if the space of synthetic gravity waves also includes wildly unrealistic images since the model will never be asked to classify such images in the real task). A second major issue is that there are visual artifacts present in many of the synthetic wave patterns where the process which generated them skipped pixels or doubled the intended brightness of certain pixels. It is likely the model learned to identify these visual artifacts, which is useful for the synthetic task but will not generalize at all.

In order to go further with this approach, it would be necessary to significantly increase the sophistication of the synthetic data generator (ideally without significantly increasing its run-time).

8 Conclusions

We presented preprocessing methods to transform raw HDF5 data from our chosen domain into usable images and evaluated the performance of two transfer learning methods on our classification dataset. Our autoencoder method allowed us to leverage our large supply of unlabeled image data to identify meaningful structures within our images and then to use identified features to perform the classification task. Our customized IncpetionV3 model made use of the most relevant parts of a state-of-the-art architecture while adding our own layers and modifications that best suit our goals.

We anticipate these findings will be useful for classifying gravity waves in small datasets of noisy satellite images and for overall future research on gravity wave classification. This work can also be helpful for other transfer learning endeavors, especially those using small datasets.

For future research, we will observe how our models perform on datasets from other satellites, fix overfitting issues, explore other promising transfer learning models, and refine fake data generation techniques to combat 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). The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (OAC–1726023, CNS–1920079).

More at https://bigdatareu.umbc.edu

References

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

[2] David C. Fritts and M. Joan Alexander. Gravity wave dynamics and effects in the middle atmosphere. *Reviews of Geophysics*, 41(1), 2003.

[3] David 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] Florian Zink and Robert A. Vincent. Wavelet analysis of stratospheric gravity wave packets over macquarie island: 2. intermittency and mean-flow accelerations. *Journal of Geophysical Research*, 2001.

[6] Thomas Colligan, Jennifer Fowler, Jaxen Godfrey, and Carl Spangrude. Detection of stratospheric gravity waves induced by the total solar eclipse of july 2, 2019. *Sci Rep 10*, 2020.

[7] Pierdavide Coïsson, Philippe Lognonné, Damian Walwer, and Lucie M. Rolland. First tsunami gravity wave detection in ionospheric radio occultation data. *Earth and Space Science*, 2(1):125–133, 2015.

[8] Steven E. Koch and Christopher O'Handley. Operational forecasting and detection of mesoscale gravity waves. *Weather and Forecasting*, 12(2):253–281, 1997.

[9] Chang Lai, Jiyao Xu, Jia Yue, Wei Yuan, Xiao Liu, Wei Li, and Qinzeng Li. Automatic extraction of gravity waves from all-sky airglow image based on machine learning. *Remote Sensing*, 11(13):1516, 2019.

[10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[11] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019.

[12] John Blitzer, Mark Dredze, and Fernando C Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, 2007.

[13] Xavier Glorot, Antoine Bordes, and Yoshua 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] Steven W Smith et al. The scientist and engineer's guide to digital signal processing, 1997.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] Yifei Zhang. A better autoencoder for image: Convolutional autoencoder. 2018.

[18] Jie Lu, Naveen Verma, and Niraj K. Jha. Convolutional autoencoder-based transfer learning for multi-task image inferences. *IEEE Transactions on Emerging Topics in Computing*, 10(2):1045–1057, 2022.

[19] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134:19–67, 2005.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, pages 1929–1958, 2014.

[21] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

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

[23] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[24] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *ArXiv*, abs/2104.00298, 2021.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2015.

[26] Artur Souza, Leonardo B. Oliveira, Sabine Hollatz, Matt Feldman, Kunle Olukotun, James M. Holton, Aina E. Cohen, and Luigi Nardi. DeepFreak: Learning Crystallography Diffraction Patterns with Automated Machine Learning. *arXiv preprint arXiv:1904.11834*, 2019.