School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales, Sydney

COMP9417 – Machine Learning & Data Mining
Term 2 – 2023
Group Project Report

Kaggle Competition
Google Research - Identify Contrails to Reduce Global Warming

Name	Student ID	Email
Taimoor Shabih	z5206259	z5206259@ad.unsw.edu.au
Lachlan Ting	z5264855	z5264855@ad.unsw.edu.au
Sarth Bishnoi	z5287670	z5287670@ad.unsw.edu.au
Hoang Long Pham	z5417369	z5417369@ad.unsw.edu.au
Quoc (Peter) Tuan Pham	z5364829	z5364829@ad.unsw.edu.au

Introduction

Climate change is an increasingly complex issue that requires a comprehensive and collective global response from individuals, governments, and businesses to address its multifaceted challenges. In the context of mitigating the effects of climate change, the issue of contrail formation by aircraft presents an unique challenge. Contrails or condensation trials, are visible cloud-like trails that form when hot water vapour from a jet engine is expelled, then condenses and freezes. From 2000 - 2018, contrails contributed to 57% of the aviation's global warming impact significantly more compared to CO2 emissions [1]. They contribute to global warming by trapping heat in the atmosphere that would otherwise be released into space. Our task, set within the framework of the "Google Research - Identify Contrails to Reduce Global Warming" competition, was to develop a machine learning model capable of identifying contrails in satellite imagery provided by GOES-16 Advanced Baseline Imager (ABI) to help validate predictive models developed by researchers on contrail formation. In doing so, this helps airlines to minimise the impact by avoiding creating these environmentally damaging contrails.

Given the complex nature of the problem, an ensemble model was used to achieve robust results and to maximize overall performance. The ensemble model was composed of 4 models in total: DeeplabV3+, PSPNet, FCN and U-Net. By utilizing ensemble modeling, we were able to make use of the strengths of each of the individual models to come up with reliable, accurate predictions.

This specific competition was chosen due to the real-world implications of the competition. By participating in this competition, we had a chance to mitigate the issues related to contrails and climate change. By participating, we are contributing to a more sustainable future. The opportunity to work with a time-series dataset presented a good opportunity to learn valuable skills that are used in various fields of machine learning. The preprint went into detail about how the datasets were prepared and labeled for training, which means we wouldn't have to spend time creating our own training and testing data and could focus on the machine learning aspect of the problem.

Related Literature

As this challenge uses datasets from the "OpenContrails: Benchmarking Contrail Detection on GOES-16 ABI" research paper, extensive research into this preprint was done before tackling this competition. It was important to understand the aim of the research paper, how images were gathered, the rules behind what exactly was classified as a contrail, how the contrails were found, and how the contrails were labeled. The MDPI research paper was used to gain an understanding of how CNNs can be used to detect contrails in satellite imagery. The GOES-R ABI Bands Quick Information guide was used to gain a deeper understanding about the different spectral bands present in the satellite imagery provided by GOES-R. [2][3]

Experimental Setup

Data Exploration

Since we have a huge dataset, which contains roughly around 20000 training images and 1800 validation data, there is a need for us to explore the composition of both the training and validation data, and then select a portion of it from the entire dataset to train and validate. By exploring the composition of the data, we want to ensure that training dataset do not heavily favour any particular categories, which may negatively affect our models' performances.

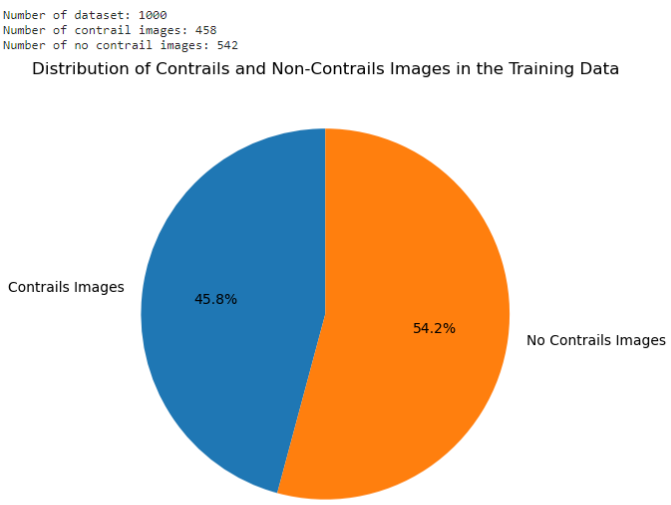


Figure 1: Distribution of Contrails and Non-Contrails in Training Data

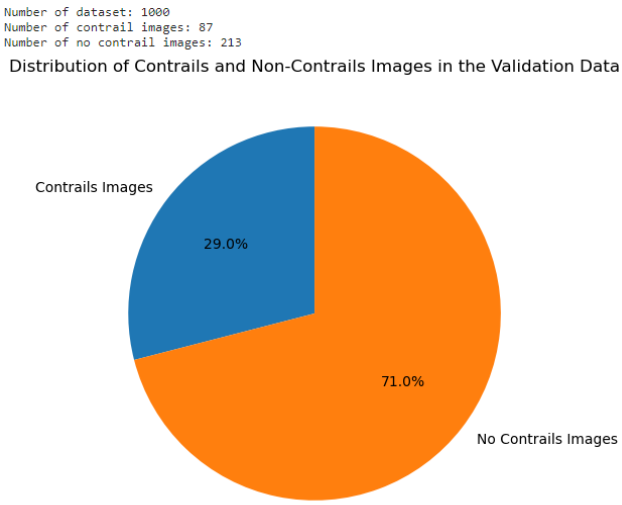


Figure 2: Distribution of Contrails and Non-Contrails in Validation Data

For the training set, the split between contrails images and non-contrails image were roughly even (Figure 1); however, the validation data was imbalanced (Figure 2), which can adversely affect our evaluation.

The number of data for training and validation that we choose is arbitrary, but it is based on the assumption that the more data we have for our models, the better outcomes we likely get. Another factor influencing our selection of the number of training data is our constrained resources such as computational power, and time needed to train. Therefore, we settle down on 1000 training samples and 300 validation samples.

Data Augmentation

Data augmentation techniques were applied in this machine learning project to enhance the generalisation and robustness of the selected models. Various transformations were employed

on the original dataset, including random horizontal and vertical flips, as well as rotations of up to 15 degrees. Additionally, noise was introduced into the images.

It's worth noting that flipping images across different axes and applying rotations only alters the image's orientation, without affecting the masks. This approach was chosen considering the standardized nature of satellite imagery, ensuring the model's familiarity with satellite views.

Noise was intentionally added to the images to mimic the inherent graininess often present in satellite imagery. This ensures that the model remains effective in identifying contrails even amidst the challenges posed by noise.

The results before and after data augmentation can be found in the Appendix. It is important to note that the masks were not augmented due to the fact that contrails have specific labelling requirements as set out by the experiment and any alteration to that would not be an accurate representation of the contrails.

Preprocessing Data Strategies

Since each image file consists of 16 bands, and each band contains 8 temporal frames with the contrails/ non-contrails frame at the 4th element. Since we chose 3 bands (11, 14, 15), we had an input with 24 channels. These channels provided us with the richness of temporal context of the data, and thus, they can help our deep learning models better internalise the underlying structure of the appearance of contrails in images, and thereby better predict the results. However, this abundance of contextual information comes at the cost of computational and time resources to us. As a result, we solved this dilemma by testing and comparing several preprocessing data strategies.

1. Applying a channel reduction method (Conv1x1): For this, we applied a Convolutional layer (1x1) to extract features from 24 channels to 3 channels. We wanted to capture valuable information and then fed them to our models. In this way, it could both reduce computational costs whilst mitigating the loss of input's information.

2. 3 channels method: In our data preprocessing step, we chose the 4th frame out of 8 frames of each brand, which would indicate the presence or absence of contrails. Since we had 3 bands, this means that overall we had 3 channels for each input. This strategy sacrificed the temporal qualities of the data, but it would give our models a clearer shape of a contrail.

3. 24 channels method: We fed the the entire 24 channels (8 channels for each band) to our models by adjusting the models' *in_channels* parameter to take 24 channels. While using more input channels and providing a comprehensive temporal context can potentially enhance the model's ability to capture complex patterns in the data, it also comes with trade-offs. Training models with more channels can be computationally demanding and might require a larger

dataset to prevent overfitting. In addition, it's worth considering to examine whether using 24 channels truly adds valuable information or if it could lead to increased noise in the data.

4. 9 channels method: We tried out with 1 frame before and 1 frame after the 4th frame, which would better give the temporal context but not consume too much resources. As a result, each band would have 3 channels, and overall we had 9 channels for each input. We hope that this might be able to find the number of channels that allowed us to strike a balance between the loss of information and the availability of resources. To do this, we had to experimentally test different models' *in_channels* parameters and adjust the input selection in our dataset preprocessing function accordingly.

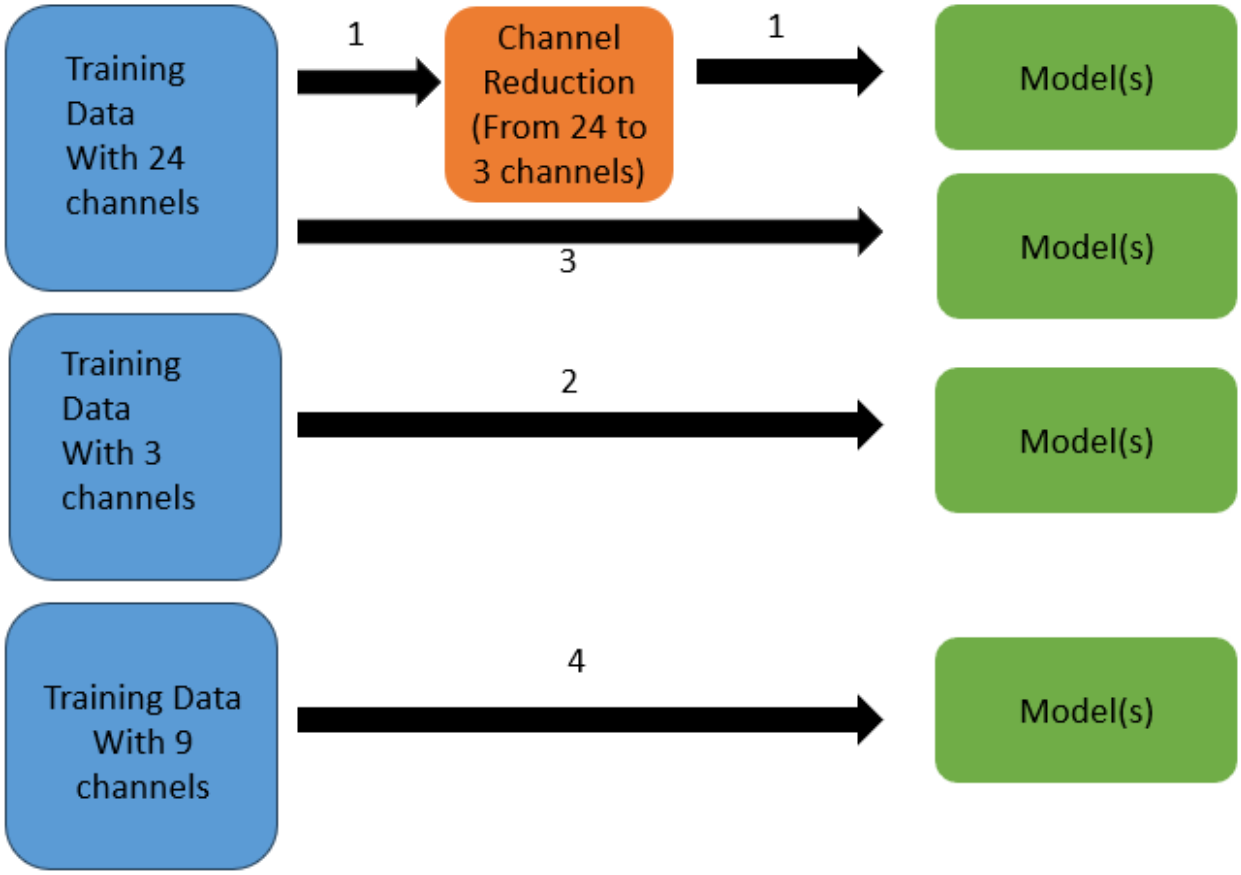


Figure 3: This is a high level illustration of our different attempts to find the best and most efficient inputs' sizes

For all of these strategies, we used the same model (PSPNet), with the same loss function (DiceLoss), same learning rate (0.001), and same epochs (20). This is because to test these strategies we need.

We employed *ceteris paribus* method to test our strategies by isolating the impact of each strategy while holding other variables constant. To maintain a controlled testing environment, we used consistent settings across all strategies. Specifically, we had the same PSPNet model,

employed the DiceLoss as the loss function, maintained a uniform learning rate of 0.001, and conducted training for the same number of epochs (20).

Models Selection Strategies

Since this is a semantic segmentation problem, which requires our models to classify every pixel. There is a variety of deep learning models that we could take advantage of using. In selecting models for this task, we aimed for models that possessed different architectures and then evaluated their performances. This led us to choose DeeplabV3+, PSPNet, UNet and FCN models. Each has their unique capabilities, and also we used FCN as our baseline model. Moreover, we also wanted to exploit different architectures of many models and thus, we utlised the ensemble deep learning method by stacking them together with the intention of having these different architectures complementing each others, resulting in better predictions.

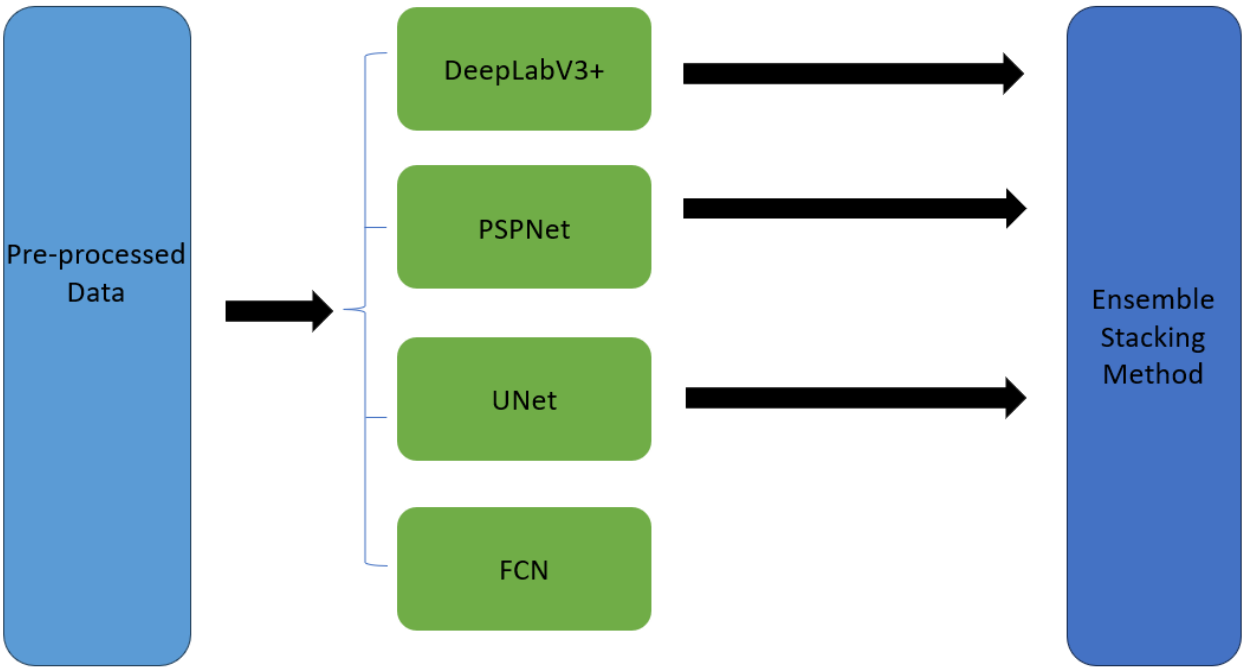


Figure 4

For the overview of these models' architectures, here are the links to each paper . Here, we want to focus on some of the prominent characteristics of each model that we wanted to utilise in the context of contrails detection.

DeepLabV3+

This was one of the models that was used as it is a very robust and capable deep learning model for semantic image segmentation. For this model, we wanted to leverage its dilated convolutions and Atrous Spatial Pyramid Pooling (ASPP) feature extraction techniques.

- Dilated convolutions strategically introduce gaps in filters, modulated by the dilation rate, to amplify receptive fields. This enables the model to encompass richer context without undue complexity. This approach becomes particularly valuable when extending the receptive field is needed, yet enlarging filters would lead to impractical computational demands.

- Atrous Spatial Pyramid Pooling uses multiple parallel atrous (dilated) convolutional layers with different dilation rates to capture contextual information at different scales. These atrous convolutional layers are combined to create a "pyramid" of features that contain both local and global contextual information. This pyramid of features is then merged to produce a more comprehensive representation of the input image.

- In the context of contrails detection, the application of ASPP architecture allows the model to better capture both global and local contexts of the image without compromising too much computational resources. Particularly, dilated convolutions with controlled gaps offers a strategic advantage. When identifying contrails in images, capturing contextual information across varying scales is crucial. Standard convolutions with fixed filters might struggle to encompass the entire range of contrail shapes and sizes, potentially leading to missed detections.

PSPNet

This was used as it is yet another capable deep learning model that is used for semantic image segmentation. It also captures multi-scale information albeit with a different method which is pyramid pooling.

- Pyramid Pooling Module (PPM): The PPM is the most essential component of PSPNet. It applies pooling operations at different pyramid levels to obtain multi-scale contextual data.

    - The encoder sends the input feature map into the PPM.
    - The feature map is divided into different regions of interest (ROIs) by the PPM.
    - The regions are then divided into different pyramid levels (each of which has a different level of pooling). For example, 2x2, 4x4, 7x7 etc.
    - Average pooling is applied to each ROI at different scales. This pooling operation generates fixed-size representations for each region, effectively capturing different scales of context.

    - An overall contextual representation is formed by combining the representations from each level.

- When identifying contrails in images, it's essential to consider that contrails can vary significantly in terms of length, thickness, and orientation. By applying the PPM, the model can effectively analyse the image at multiple scales, ensuring that it captures the full range of contrail characteristics. Furthermore, the combination of representations from different pyramid levels within the PPM contributes to a more comprehensive understanding of the image's context. This helps the model distinguish between genuine contrails and potential false positives that might arise from similar-looking features in the image.

UNet

U-net works by classifying pixels of an input image into different classes. It uses an encoder to compress the image and a decoder to restore it. The encoder in the U-net architecture performs a similar function to the feature extraction layers in a CNN (Convolutional Neural Network), which captures important features from the input data. However, in U-net, the encoder's primary purpose is to compress the spatial dimensions while increasing the number of feature channels, which helps extract hierarchical features from the image. This compressed representation is then used by the decoder to reconstruct the segmentation map. Also, the model uses Skip connections to allow the network to combine features from different scales. Overall, these combinations allow the model to detect intricate, fine details, and subtle features of contrails.

FCN

FCN works by segmenting the image into multiple semantic categories. It produces accurate pixel specific predictions.

- Encoder for Feature Learning: The initial CNN encoder learns intricate features from input aerial images, such as contrail shapes, patterns, and background sky details.
- Semantic Segmentation: FCN segments the image into binary categories, possibly "contrail" and "no contrail," providing precise pixel-level predictions.
- Upsampling and Skip Connections: These mechanisms help preserve thin and lengthy contrail features during segmentation and upscaling, improving detection accuracy.

This is the baseline model, so its features are included in other models as well.

Ensemble Method

The ensemble method allows us to combine our selected models' results to arrive a better prediction. There are many options for the ensemble deep learning methods such as majority voting, stacking, or weighted average, etc. In this project, we implemented the stacking method by concatenating each models' result through *torch.cat* and apply the newly formed result through a Convolutional layer (1 x 1) to extract channel features from 3 to 1 and then apply the activation function to it to better capture the non-linear nature of contrails which has a diverse shape, length and patterns.

This method offers a systematic approach comprising different specialties from UNet, PSPNet, and DeepLabV3 to address the problem. This method can enhance overall performances by minimising each individual model biases and errors, it can also aggregate information from different scales and feature representations, allowing for capturing multi-scale context in ensuring the robustness of the predictions.

Implementation of Models

For Unet, PSPNet, and DeepLabv3+, they were already implemented in Segmentation Models (*segmentation-models-pytorch*), we implemented these models by calling them from the imported library.

Unlike aforementioned models, we built FCN and Ensemble Stacking models from scratch. For FCN, by building this from scratch it allowed us to better understand the structures of the other semantic segmentation models. The feature extraction of FCN was based on CNN's structure, which had in 3 convolutional layers and 2 Maxpooling functions. During this operation, the inputs' channels were enlarged to 32, 64 channels and then passed it to the Upsampling layers . The maxpooling methods were used after every convolutional layer with the kernel size of 2 x 2, and this helped reduce the number of features. After feature extraction implementation, we built the upsampling layers by using Transposed convolutional layers.

Finally, after training and evaluating models' performances, we ensembled Unet, PSPNet, and DeepLabv3+ by stacking them. The high level implementation of this method has been described earlier with visualisation in Figure 4.

Hyper Parameterisation

In the hyperparameter optimization process, we systematically fine-tuned the performance of all the models by exploring various configurations of batch sizes, loss functions, optimizers, and

learning rates. Specifically, for each individual model, we conducted hyperparameter tuning on the following key aspects:

- Batch Size: We experimented with different batch sizes to find the one that strikes the right balance between computation efficiency and convergence speed.

- Loss Function: Different loss functions were tested to determine the one that best suited the characteristics of our problem and provided effective optimization.

- Optimizer: We explored a range of optimizers to identify the one that yielded optimal convergence and training efficiency for each model.

- Learning Rate: Through careful experimentation, we determined the most suitable learning rate for training, which significantly impacts the model's convergence and generalization.

By systematically adjusting and optimizing these hyperparameters, we aimed to enhance the overall performance of each model and ensure its effectiveness in addressing the specific challenges of our task. The results obtained from changing each of these hyperparameters were documented in tables that can be found in the appendix of this report.

Results and Discussion

Preprocessing Data Strategies

For the applying a channel reduction method, after applying the convolutional layer to reduce the channels, we compared the transformed input with the input from the 3 channels method. As illustrated in Appendix Section 2 Figure 1, the input extracted from the channel reduction method could not capture the essence of contrail and its relations when compared with the input generated by the 3 channels method. This result is reflected by Appendix Section 2 Figure 2, when the training and validation loss values fluctuated and did not converge.

When we compared this approach with other approaches, it was clear that using channel reduction method yielded the lowest performance.

Next, we compared the performance of processing 24 channels method with 9 channels and 3 channels methods. As seen the appendix Section 2 figure [3] [4] [5], it is clear that these approaches were roughly the same, albeit the 3 channels approach appeared to converge faster with training lower losses than the other two methods. Also, the training loss values of 9 channels and 3 channels methods seemed to be smoother than that of 24 channels method. With the validation loss values, the three models seemed to approximately give the same values, even though the 3 channels method had the validation loss converged faster than other

two models. This differences in the rate of convergence could be attributed to the complexities of the models and the data that required more resources to train.

Therefore, since these three approaches yielded roughly the same results and the 3 channels method appeared to be the most stable in both training and validation losses values, in the context of our limited resources we decided to use the 3 channels method.

Hyper Parameterisation

Models Comparisons

These results are from the training and validation loss located in the appendix section 3 Hyperparameterisation.

Unet:

- From the training and validation loss table of the UNet model we can see that 32 is the ideal batch size for training. This is so that the model can collect more global context during the encoding stage and facilitate more precise segmentation during the decoding stage with bigger batch sizes.
- Performance-wise, the Adam optimizer outperforms SGD according to the training and validation loss table. This is the case due to Unet's complicated loss surfaces and numerous local minima. The adaptive learning of Adam can help him navigate these surfaces.

Deeplabv3+:

- The batch size of 32 outperforms size of 16 in the training and validation . Since deeplabv3's semantic segmentation requires pixel-wise classification of objects in an image, making it a challenging task. Larger batch size enables faster computation and training.
- Adam is the best optimizer for this model since it makes it easier to navigate the optimisation environment.
- The loss decreases gradually with more epochs. This suggests that with more training the model will continue to improve.

PspNet:

- 16 is the ideal batch size for training. This is due to the fact that it achieves a balance between memory usage and computational effectiveness. Faster convergence and less overfitting are achieved as a result.
- The best optimizer is Adam. Similar to other models this is because it can navigate the complex landscape of PSPNet better than SGD can.

FCN:

- The best batch size for this model was 16 for similar reasons to why 16 was the best size for PspNet

- The Adam optimizer outperforms the SGD. This is for the same reasons as the other models.
- It was observed that there was a rapid decrease in loss during the initial epochs, however after a while it was observed that the model had reached a state of convergence. Thus, it does not require high amounts of epochs to train.

Ensemble Model

0.001 Learning Rate on Batch size of 16		Training Loss	Validation Loss
Epochs	1	0.693159	0.693471
	2	0.693133	0.693471
	3	0.693141	0.693550
	4	0.693125	0.693471
	5	0.693115	0.693450
	6	0.693122	0.693811
	7	0.693114	0.693471
	8	0.693113	0.693500
	9	0.693110	0.693450
	10	0.693131	0.693600

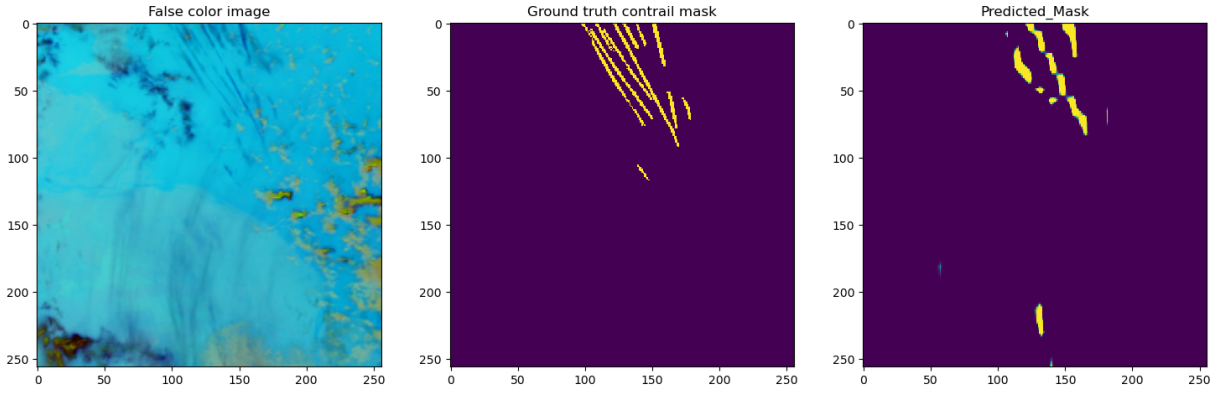
0.0009 Learning Rate on Batch size of 16		Training Loss	Validation Loss
Epochs	1	0.783980	0.770772
	2	0.704099	0.693559
	3	0.695391	0.804369
	4	0.693788	0.693285
	5	0.693479	0.693566

	6	0.693350	0.693152
	7	0.693278	0.693190
	8	0.693347	0.693182
	9	0.693273	0.693339
	10	0.693384	0.693147

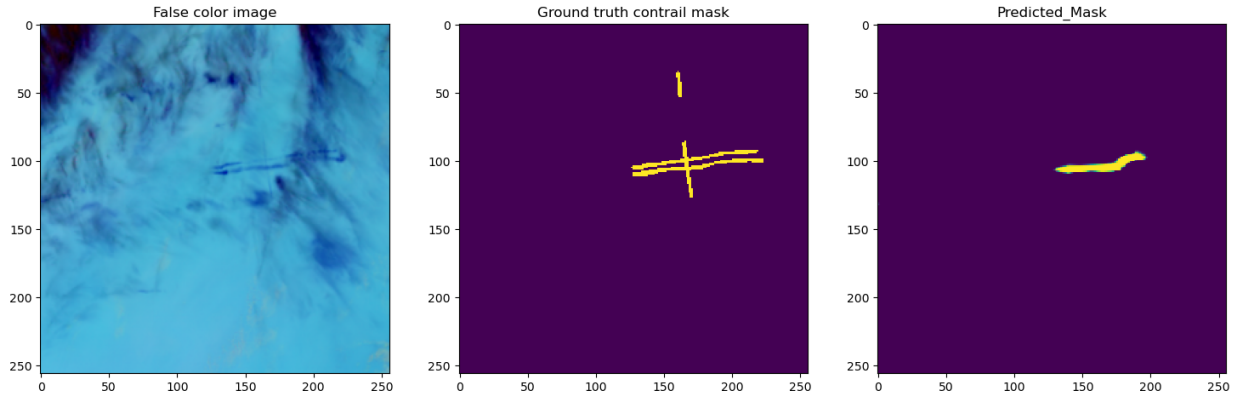
The following table describes the results of hyperparameter tuning on the ensemble model, which combined the results of Unet, PSPNet and Deeplabv3+. Based on the previous models a batch size of 16 was deemed to be the best batch size to train the ensemble model on. As for the learning rate, we chose 0.001 in the beginning with results seen in the table however, adjusting the learning rate to 0.005 or 0.0005 resulted in the model with a constant loss. Due to the model's sensitivity the learning rate of 0.0009 was chosen as the next learning rate value. This resulted in the model plateauing around the 0.6933 range. It's important to note that while the loss is high, it's hard to converge because of the nature of the task involves contrails of different shapes, forms, lengths and ranges.

Detection of Contrails

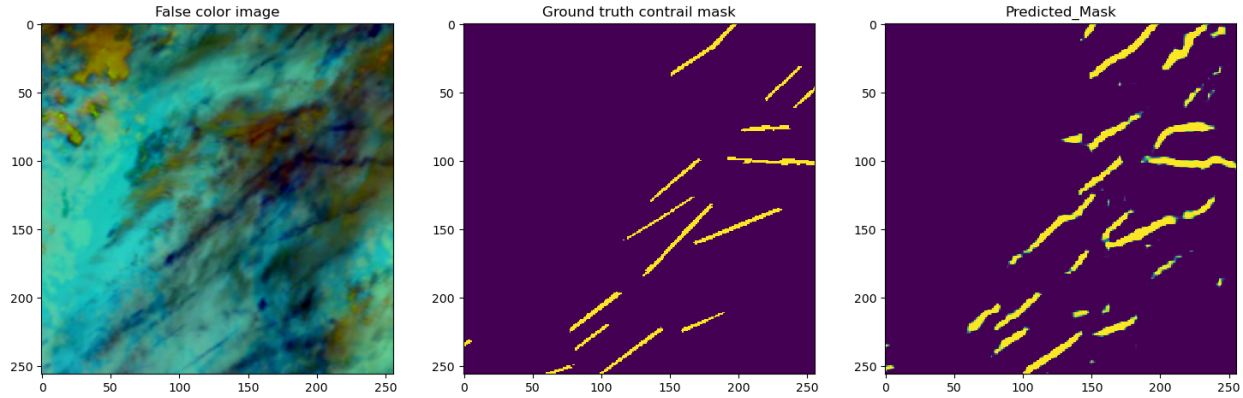
PSPNet



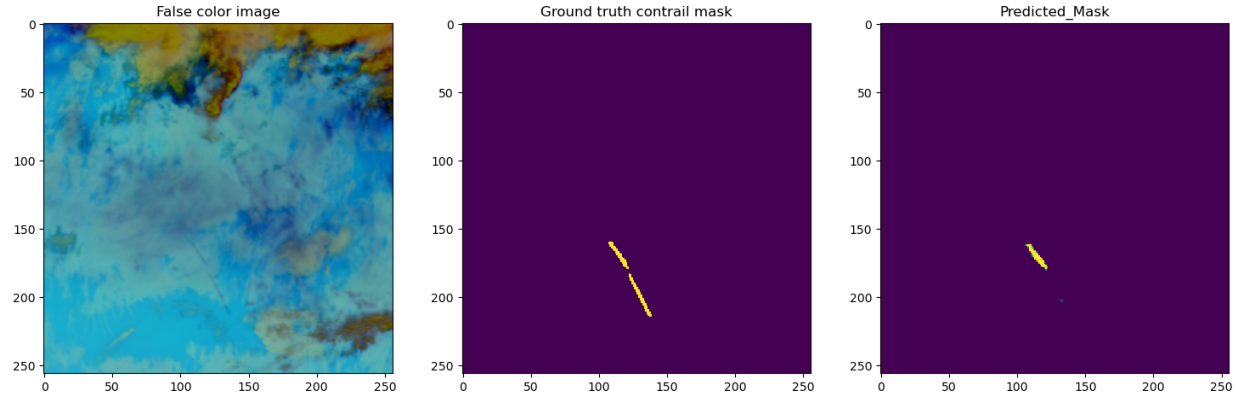
UNet



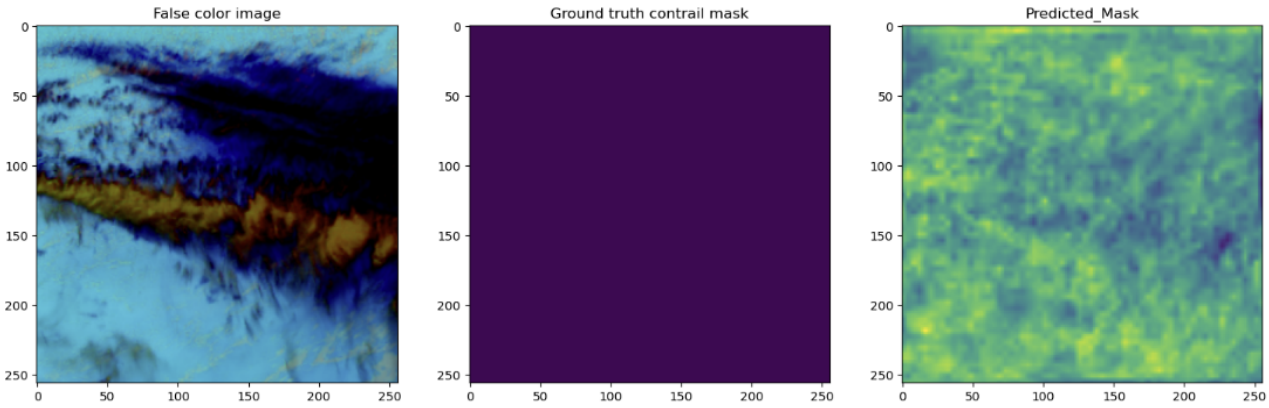
Ensemble Model



DeepLabv3+



FCN



Conclusion

The aim of this competition was to detect contrails in satellite imagery. Through the use of ensemble modeling and the fact that we had limited computational power with a quite complicated segmentation task, we were able to combine PSPNet, DeepLabV3+, and Unet to yield a reliable ensemble model that identifies contrails in images somewhat successfully.

As for future improvements to this model, one method that could be utilized for tuning hyperparameters is grid search. Grid search involves dividing the domain of hyperparameters into a grid and utilizing every permutation of the parameter values in that grid. Memory limitations prevented us from training the model on more training data.In the future, we would try to get access to more VRAM and a faster GPU for training purposes. Also, we can try to further mitigate the validation data imbalance by not only augmenting data and ensemble method, but also using other ways such as assigning weights to minority data, or implementing threshold. Lastly, other deep learning methods that we can do to better predict the contrails are of transformer based models like segformer or UNETR where these models integrate state-of-the-art architecture to solve semantic segmentation problems, and potentially we can use our understanding of implementing ensemble deep learning method to combine these visual transformer based models.

References

[1] Author links open overlay panelD.S. Lee a et al., “The contribution of Global Aviation to anthropogenic climate forcing for 2000 to 2018,” Atmospheric Environment, https://www.sciencedirect.com/science/article/pii/S1352231020305689 (accessed Jul. 26, 2023).

[2] J. P. Hoffman, T. F. Rahmes, A. J. Wimmers, and W. F. Feltz, “The application of a convolutional neural network for the detection of contrails in satellite imagery,” *Remote Sensing*, vol. 15, no. 11, p. 2854. doi:10.3390/rs15112854

[3] “ABI Bands Quick Information Guides,” ABI Bands Quick Information Guides, https://www.goes-r.gov/mission/ABI-bands-quick-info.html (accessed Jul. 24, 2023).

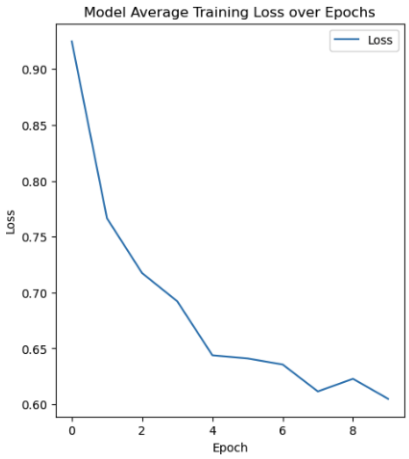
[4] Jais, I. K. M., Ismail, A. R., & Nisa, S. Q. (2019). “Adam optimization algorithm for wide and deep neural network. *Knowledge Engineering and Data Science”*, *2*(1), 41-46.

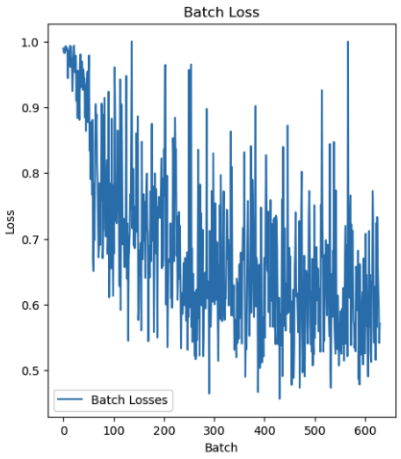
[5] Gupta, A., Ramanath, R., Shi, J., & Keerthi, S. S. (2021, December). “Adam vs. sgd: Closing the generalization gap on image classification”. In *OPT2021: 13th Annual Workshop on Optimization for Machine Learning*.

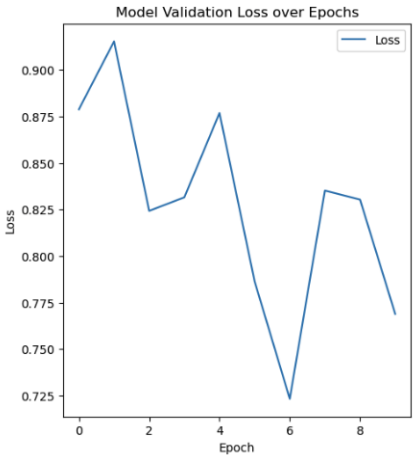
Appendix

Section 1:

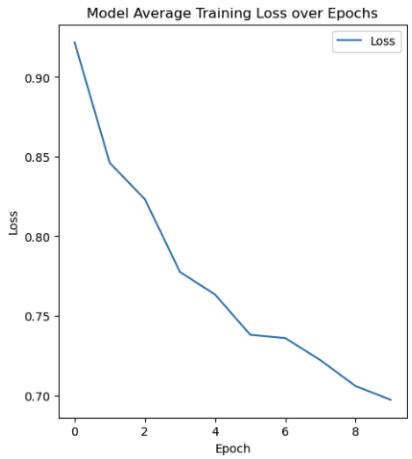
Deeplab3+

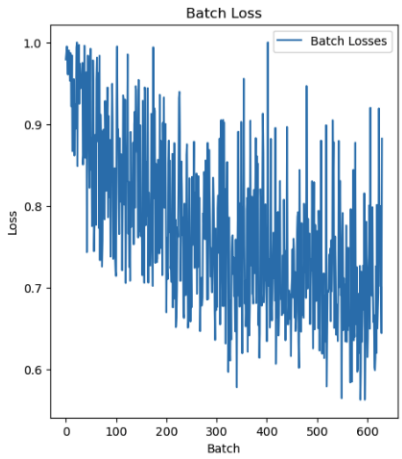


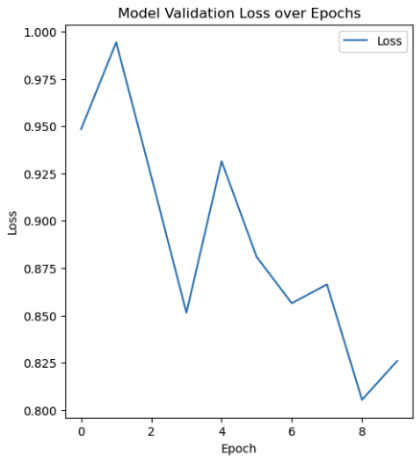




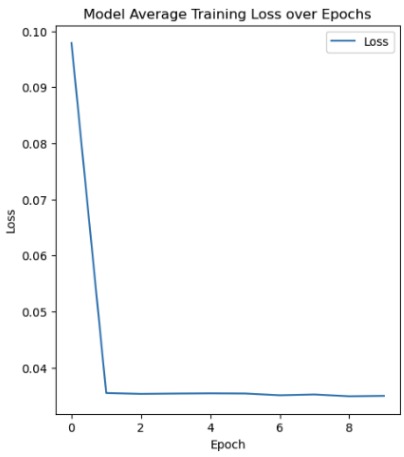
PSPNet

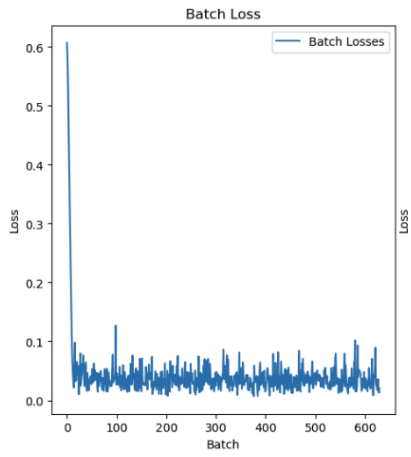


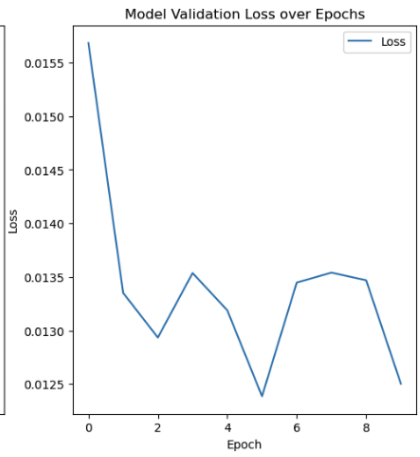




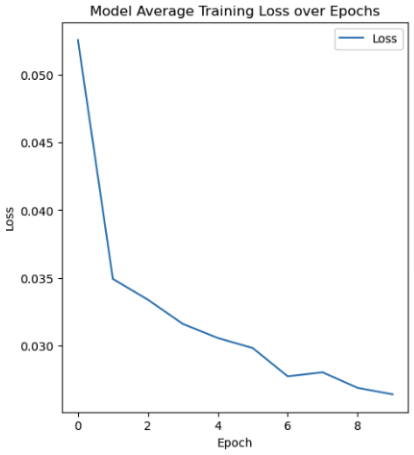
FCN

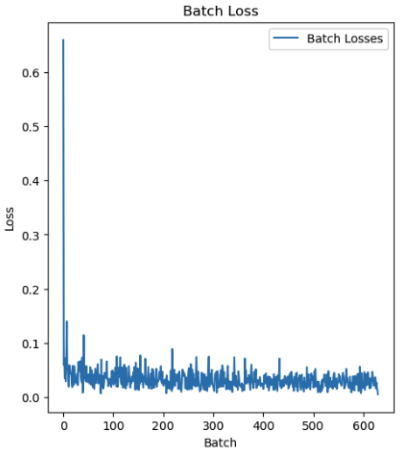


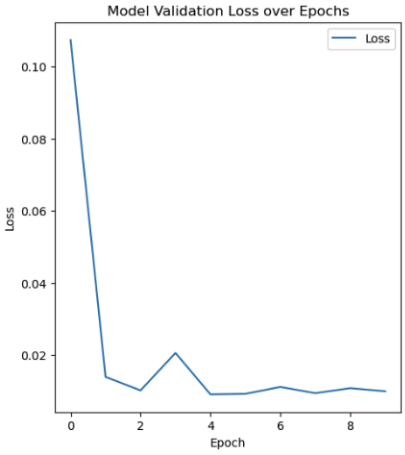




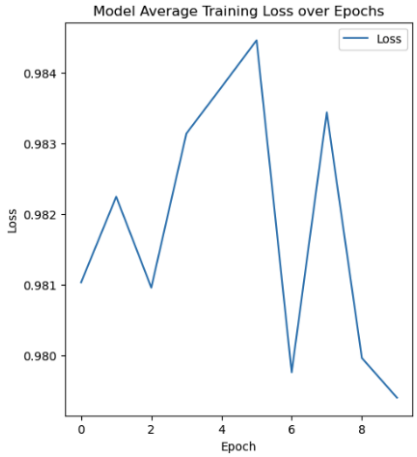
UNet

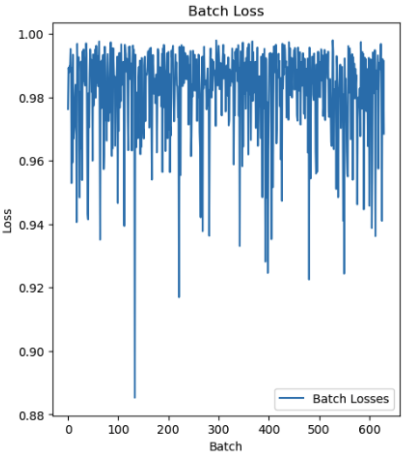


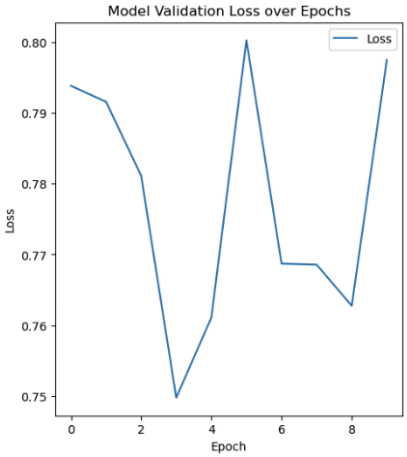




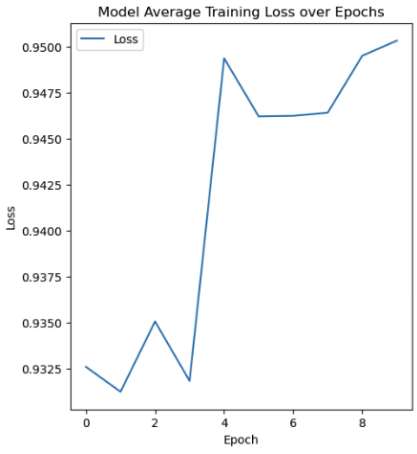
Deeplab with data augmentation

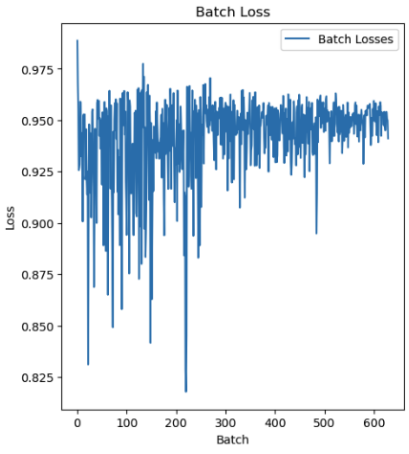


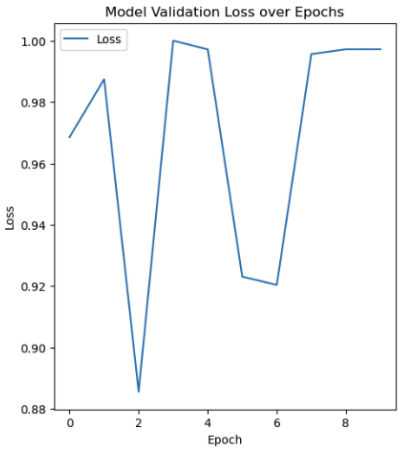




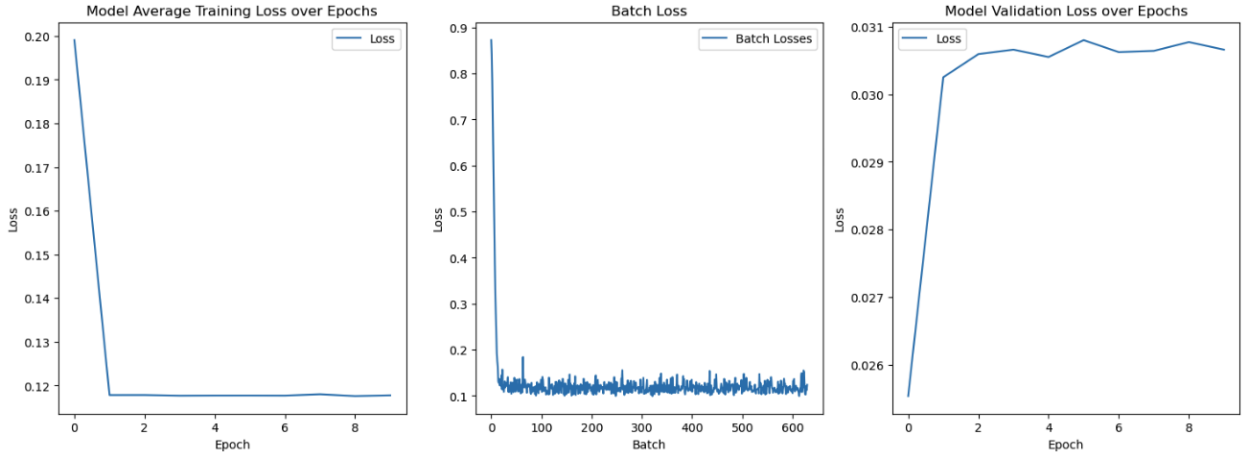
PSPNet with data augmentation



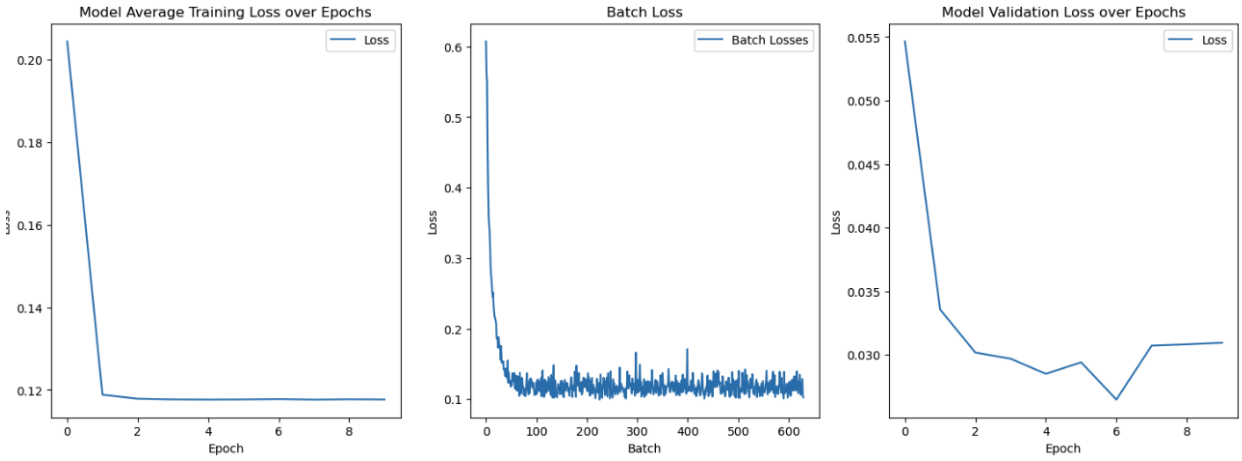




FCN with data augmentation



UNet with data augmentation



Section 2:

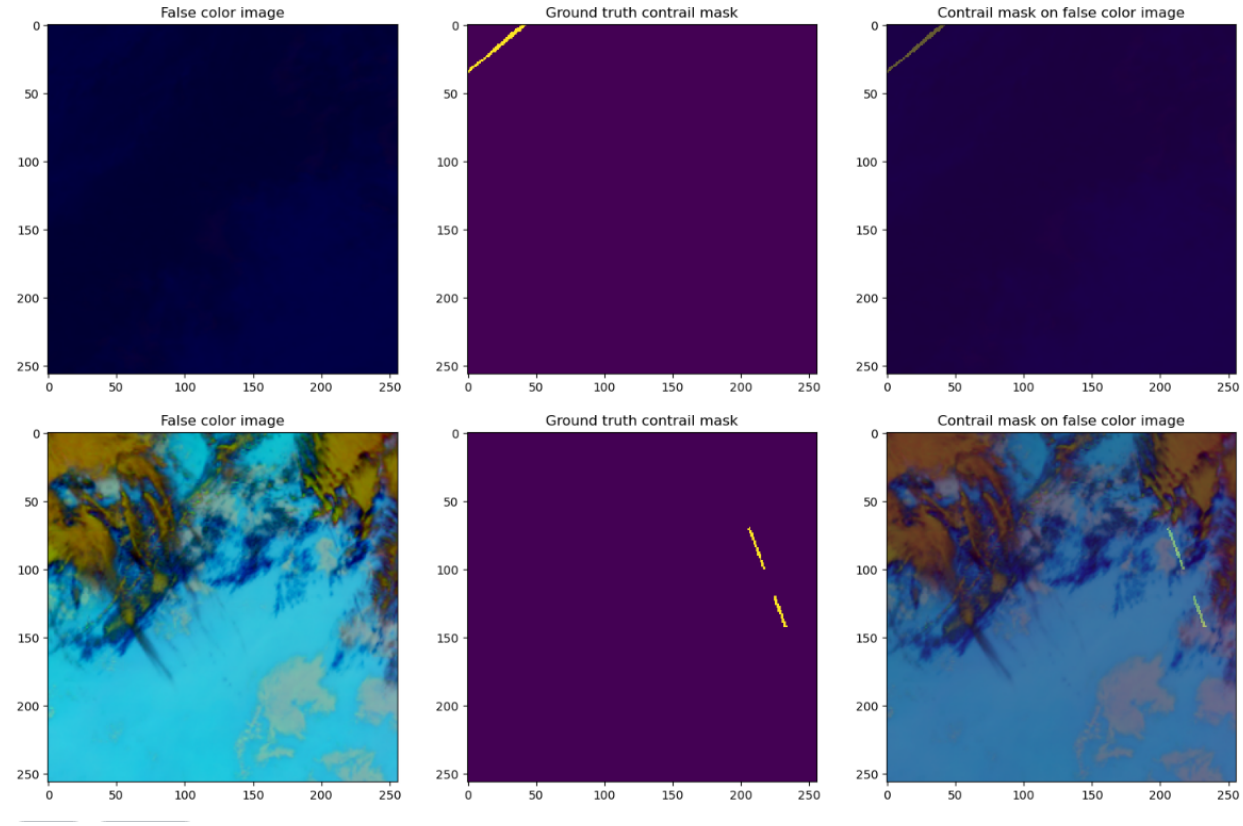


Figure 1: The first row is the outputs of applying a channel reduction approach, whilst the second row is the output of selecting the contrails/ non-contrails frame approach

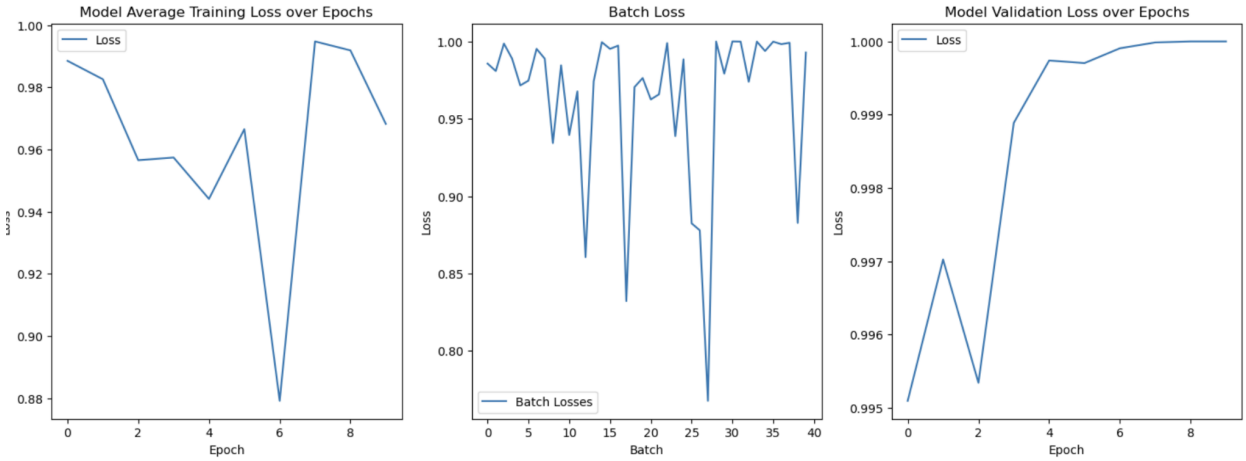


Figure 2: This is the result of transforming 24 channels to 3 channels

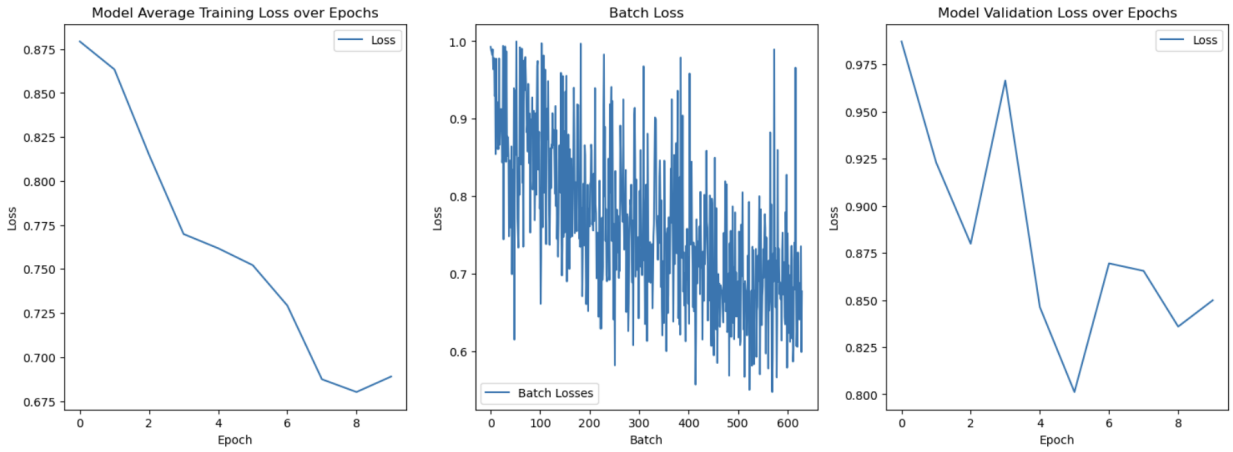


Figure 3: This is the result of 3 channels

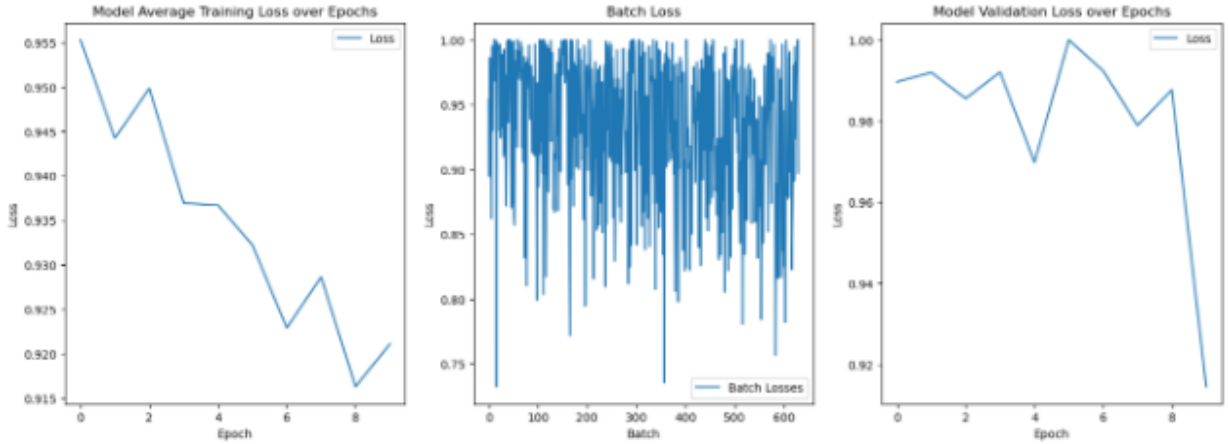


Figure 4: This is the result of 24 channels

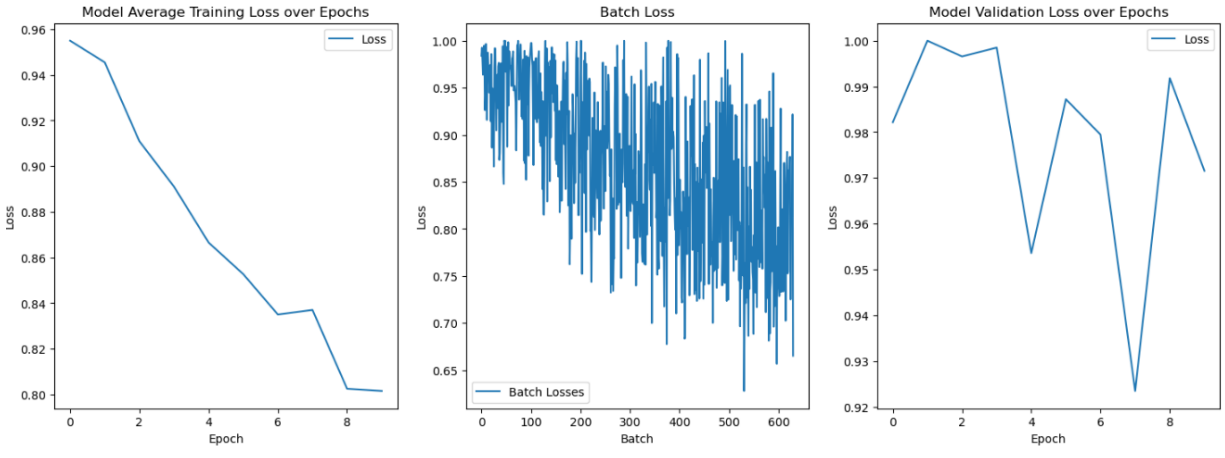


Figure 5: This is the result of 9 channels data

Section 3:

Hyper-Parameterisation Tables

UNet

The following are the loss results for U-net with different batch sizes.

Training Loss

		Batch Size	
		16	32
Epochs	1	0.698171	0.71026
	2	0.693147	0.693147
	3	0.693147	0.693147
	4	0.693147	0.693147
	5	0.693147	0.693147
	6	0.693147	0.693147
	7	0.693147	0.693147
	8	0.693147	0.693147
	9	0.693147	0.693147
	10	0.693147	0.693147

Batch Size Validation Loss

		Batch Size	
		16	32
	1	0.694572	0.695983
	2	0.694572	0.695983

Epochs	3	0.694572	0.695983
	4	0.694572	0.695983
	5	0.694572	0.695983
	6	0.694572	0.695983
	7	0.694572	0.695983
	8	0.694572	0.695983
	9	0.694572	0.695983
	10	0.694572	0.695983

Optimiser Loss

		Optimiser	
		Adam	SGD
Epochs	1	0.698171	0.684354
	2	0.693147	0.674853
	3	0.693147	0.689321
	4	0.693147	0.683324
	5	0.693147	0.675542
	6	0.693147	0.679356
	7	0.693147	0.663520
	8	0.693147	0.664356
	9	0.693147	0.661753
	10	0.693147	0.660642

Optimiser Validation Loss

		Optimiser	
		Adam	SGD
	1	0.703556	0.683556

Epochs	2	0.704312	0.681175
	3	0.703489	0.687956
	4	0.708246	0.681221
	5	0.710031	0.670956
	6	0.698612	0.663412
	7	0.699345	0.668790
	8	0.691211	0.666644
	9	0.697148	0.664123
	10	0.696323	0.660645

DeepLabV3+

The following are the loss results for DeepLabV3+ with different batch sizes.

Training Loss

		Batch Size	
		16	32
Epochs	1	0.983863	0.982774
	2	0.975889	0.971919
	3	0.962430	0.964705
	4	0.978200	0.950250
	5	0.967567	0.932948
	6	0.915200	0.937907
	7	0.976402	0.943174
	8	0.968216	0.952312
	9	0.952019	0.913382
	10	0.919304	0.898120

Validation Loss

		Batch Size	
		16	32
Epochs	1	0.997821	0.996178
	2	0.995642	0.997612
	3	0.997821	0.851202
	4	0.997120	0.997812
	5	0.996110	0.996980
	6	0.996532	0.997792
	7	0.999871	0.997981
	8	0.987431	0.997612
	9	0.987632	0.998432
	10	0.996784	0.997462

Optimiser Training Loss

		Optimiser	
		Adam	SGD
Epochs	1	0.703589	0.643503
	2	0.704225	0.643246
	3	0.703174	0.647893
	4	0.708209	0.647732
	5	0.710905	0.645812
	6	0.697567	0.650812
	7	0.699127	0.658413
	8	0.695213	0.658712
	9	0.697323	0.649523
	10	0.697456	0.647485

Optimiser Validation Loss

		Optimiser	
		Adam	SGD
Epochs	1	0.723526	0.664322
	2	0.724301	0.664162
	3	0.723495	0.661217
	4	0.728233	0.663124
	5	0.710121	0.667964
	6	0.718705	0.660165
	7	0.719167	0.657643
	8	0.701544	0.652345
	9	0.707851	0.653224
	10	0.706443	0.658976

PSPNet

Training loss

		Batch Size	
		16	32
Epochs	1	0.975191	0.979244
	2	0.962120	0.976780
	3	0.948955	0.975815
	4	0.938576	0.973611
	5	0.940370	0.972269

	6	0.930067	0.967672
	7	0.953462	0.964335
	8	0.934521	0.959754
	9	0.927616	0.953400
	10	0.928795	0.948396

Validation loss

		Batch Size	
		16	32
Epochs	1	0.973191	0.979744
	2	0.958120	0.977280
	3	0.951955	0.976315
	4	0.937576	0.972611
	5	0.941370	0.971769
	6	0.928067	0.968672
	7	0.952962	0.965835
	8	0.935521	0.958754
	9	0.928116	0.953900
	10	0.930795	0.947896

Training loss

		Optimizer	
		Adam	sgd
Epochs	1	0.975191	0.930158
	2	0.962120	0.938058
	3	0.948955	0.932921
	4	0.938576	0.892560
	5	0.940370	0.925738
	6	0.930067	0.920944
	7	0.953462	0.913006
	8	0.934521	0.914043
	9	0.927616	0.915502
	10	0.928795	0.889786

Validation loss

		Optimizer	
		Adam	sgd
Epochs	1	0.975191	0.926158
	2	0.962120	0.940058
	3	0.948955	0.931421
	4	0.938576	0.891260
	5	0.940370	0.922944
	6	0.930067	0.912935
	7	0.953462	0.912506

		Optimizer	
		Adam	sgd
	8	0.934521	0.914543
	9	0.927616	0.915102
	10	0.928795	0.890286

Training loss

		Optimizer	
		Dice loss	Soft bce
Epochs	1	0.975191	0.700337
	2	0.962120	0.698502
	3	0.948955	0.697559
	4	0.938576	0.695656
	5	0.940370	0.695031
	6	0.930067	0.694888
	7	0.953462	0.694514
	8	0.934521	0.693969
	9	0.927616	0.694041
	10	0.928795	0.693749

Validation loss

		Optimizer	
		Dice loss	Soft bce
Epochs	1	0.975191	0.703337
	2	0.962120	0.697502
	3	0.948955	0.697759
	4	0.938576	0.694156
	5	0.940370	0.694231
	6	0.930067	0.693488
	7	0.953462	0.693214
	8	0.934521	0.694469
	9	0.927616	0.694141
	10	0.928795	0.694749

FCN

Training loss

		Batch Size	
		16	32
Epochs	1	0.147349	0.261030
	2	0.049602	0.074109
	3	0.043836	0.045724
	4	0.041854	0.042426
	5	0.039497	0.041434
	6	0.040294	0.039152
	7	0.039243	0.038754

	8	0.037611	0.037381
	9	0.036768	0.036997
	10	0.035765	0.036443

Validation loss

		Batch Size	
		16	32
Epochs	1	0.146349	0.261530
	2	0.050102	0.074359
	3	0.044036	0.046224
	4	0.040854	0.042926
	5	0.039997	0.041934
	6	0.041794	0.039752
	7	0.038743	0.038854
	8	0.037111	0.037881
	9	0.036568	0.037597
	10	0.036265	0.036843

Training loss

		Optimizer	
		Adam	sgd
	1	0.147349	0.575394
	2	0.049602	0.507534

		Optimizer	
		Adam	sgd
Epochs	3	0.043836	0.447701
	4	0.041854	0.399367
	5	0.039497	0.359461
	6	0.040294	0.326177
	7	0.039243	0.298178
	8	0.037611	0.273691
	9	0.036768	0.253069
	10	0.035765	0.234626

Validation loss

		Batch Size	
		Adam	Sgd
Epochs	1	0.146349	0.577394
	2	0.050102	0.509534
	3	0.044036	0.397367
	4	0.040854	0.358961
	5	0.039997	0.358961
	6	0.041794	0.327177
	7	0.038743	0.297678
	8	0.037111	0.274191
	9	0.036568	0.252569

	10	0.036265	0.233678