Acoustics-Based UAV Preflight Diagnostic System

Abstract

Preflight inspection is required by the Federal Aviation Administration before every Unmanned Aerial Vehicle (UAV) flight. An automated system for UAV inspection would serve to make the entire flight process unmanned, which would save companies time, resources, and money. For my project, I built off of the work I did for my capstone project, and developed an ML classifier that can identify whether that acoustic signature of a propeller corresponds to one that has no, moderate, or severe damage. I tested four different models for this project, and saw very promising results as a result of training, with multiple models having nearly 100% accuracy on their respective evaluation data.

Introduction

Motivation

I did my first co-op at GreenSight, a Boston-based UAV startup. GreenSight provides autonomous aerial intelligence to golf courses and farms by flying over the course and reporting on groundwater, temperature, and soil metrics. As a result, I've seen firsthand how important preflight inspection is before any commercial UAV flight.

For my ECE Capstone project, my group and I developed a Raspberry Pi-based data collection system to systematically collect audio samples of propeller noise. The goal was to examine the FFTs of the audio samples, and do some preliminary work in identifying whether a propeller had a defect based off of the FFT. As there was a perceptible acoustic difference between a broken propeller and an unbroken one, we expected there would be some difference in the FFT as well. I reached out to GreenSight when we started working on the project, and they were able to lend us two drones for data collection.

The purpose of the capstone was to develop this interdisciplinary system to collect data, and then examine the signal to produce a binary good/bad label. Developing a multi-label classification model to identify the extent of the propeller defect is something that was outside the scope of the capstone project; however, this is what I focused on for this project. In fact, if it wasn't for this class, I am not sure whether I would have taken a ML based approach to this classification problem as examining the FFTs themselves seemed to provide a means to distinguish the two signals.

Background

Before diving into the specifics of the machine-learning approach to this problem, it is useful to gain an understanding of the scientific basis behind this classification task.

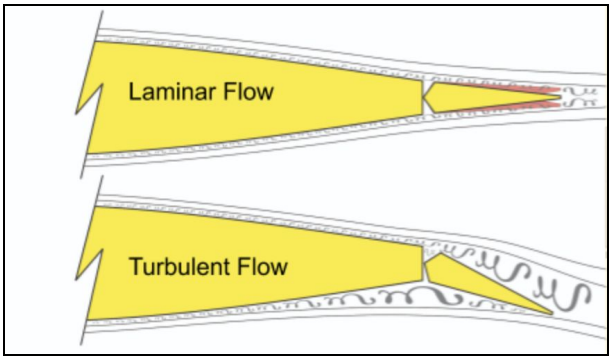

Figure 1: Propeller AIrflow [1]

As mentioned earlier, a damaged propeller and an undamaged propeller sound a bit different when they are spinning. This is due to the different airflow produced as a result of the blade hitting the air. An unbroken propeller is designed to cut cleanly through the air, in order to generate lift. This produces laminar airflow. A broken propeller, on the other hand, no longer produces the smooth laminar flow. Because of the nature of the defect, the propeller produces uneven, chaotic airflow when it spins which results in a different noise.

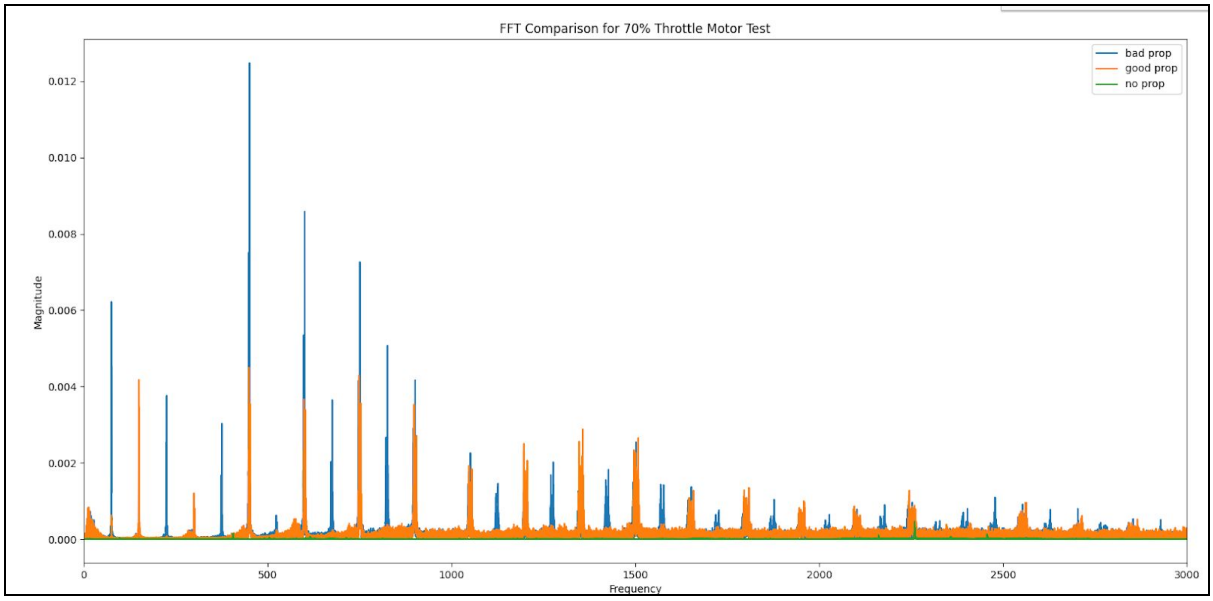

Figure 2: Comparison of Good (Orange) vs. Broken (Blue) Propeller FFTs on 70% Throttle

Much of the work done for the capstone project was focused on a signal processing-based analysis of the different acoustic signals. One particularly interesting example can be seen in Figure 2. The FFTs of the acoustic signals of a good and bad propeller were examined and there

are clear differences between the two. The magnitude of the broken propeller spikes are significantly higher, and it also has more harmonics present. This difference in the FFTs is part of the reason for the clear differences in the different spectrograms in the dataset.

Experimental Setup

Dataset Overview

Rather than classifying the audio signals as WAVs, I noticed a couple of state-of-the-art techniques online that used the spectrogram to visualize the sound as an image, and then leveraged image classification techniques. [2] The spectrogram captures both the time series and frequency information of an audio signal, as it takes a series of short term FFTs. In particular MEL spectrograms were used for this task, as they emphasize the human-hearable range of audio data. [3]

The datasets for this classification task consists of almost 2000 spectrograms corresponding to audio from three types of propellers: no damage, minor damage, and major damage. There are three datasets that the models were trained and evaluated on, with the idea being that the different datasets would present an easy, medium, and hard task for classification. The first dataset is a series of spectrograms corresponding to data collected at 70% throttle using a professional microphone. The second dataset corresponds to data collected at 30% throttle using a professional microphone. As it is easier to hear the difference between a good and bad propeller at higher throttles, the hypothesis was that images from the 30% dataset would be more difficult to classify than the 70% dataset. The last dataset corresponds to data collected at a variety of different throttles using both a professional microphone, a Raspberry Pi, and an iPhone. It was thought that this would be the most difficult dataset due to the variety in the data.

Data Collection

Two members of my capstone team have drones lent to them by GreenSight and we developed a data collection system to systematically collect multiple audio files of three propellers with varying levels of damage. The data collection system consists of two parts, the drone-to-computer interface and the recording system. The drone-to-computer interface is done by sending motor commands to the drone via PyMavlink, a Python wrapper around the MAVLink communication protocol, used for sending lightweight messages to the drone firmware. [4] The recording system consists of an acoustic microphone (the Adafruit MAX9814), and an ADC. The microphone has onboard circuitry to support reading in analog audio signals and this was then connected to one of the input terminals of the ADC. The ADC was connected to the Pi, which read data via the SPI interface.

We examine three types of propellers: no defects, a 1 inch crack (major damage), and a hairline fracture (minor damage). For each test, each propeller is held at a constant RPM for 1 minute. This 1 minute audio file is split into 4 second clips, with 2 second overlaps.

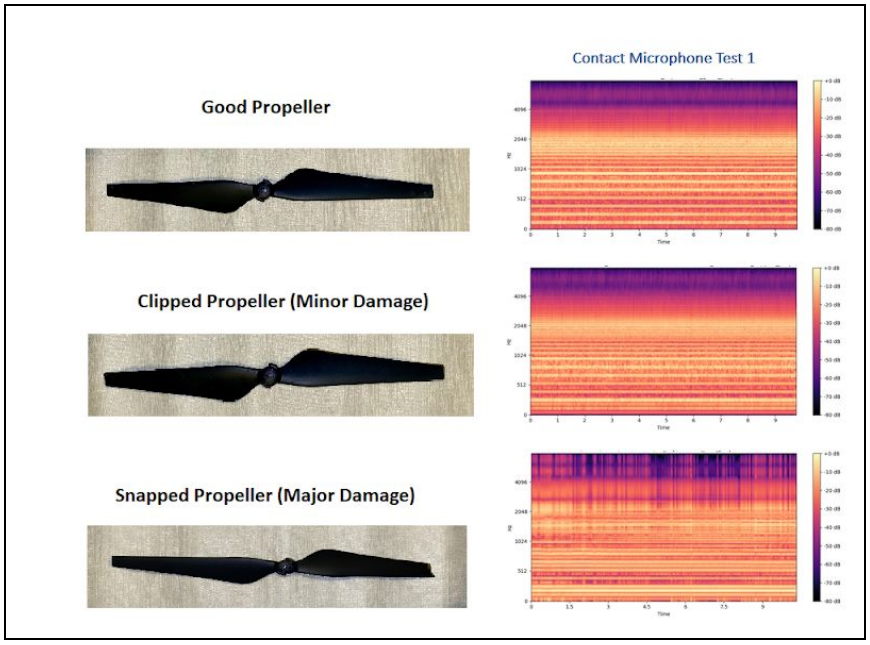

Figure 3: Sample Spectrograms for Each of the Three Propeller Defect Types

These audio files are converted into spectrograms and are labeled based on the defect type. As can be seen from the spectrograms in Figure 3, there are distinct patterns that distinguish a propeller with major damage, which indicate that this task might be easy to classify.

Models

For the project I chose to examine the performance of four separate models. I have two very simple models as a baseline, a simple linear model in which I transformed the image into a 1D vector and had a single linear layer from the input dimension to the number of classes, and a simple CNN model which has a single convolutional layer and then a linear layer. I also have a more complex CNN which has a series of convolutional layers, which are each followed by a batch norm, ReLU, and max pooling layers. Finally, I have a model that uses VGG, a state-of-the-art image model as a feature extractor, and then has a series of linear layers. [5]

Training

I trained each of these four models on each of the three datasets for 8 epochs, resulting in a total of 12 unique model-dataset combinations. Each time I trained, I did a random train-test split of 70/30% and saved the model accuracy on the test dataset after each epoch. I also saved the loss after each iteration.

Results & Discussion

Metrics

This problem can be construed as a multi-class classification problem, in which a spectrogram image needs to be classified as “good propeller”, “small defect”, or “large defect”. As a result, when evaluating the model performance, the primary metric I used was examining the percent of images correctly labeled. This is a standard measure of accuracy and provides a good idea for how well the model performs. I labeled this metric ‘Classification Accuracy’.

Additionally I wanted to develop a metric that would reflect how the model would be used in the real world. In a real world scenario, only propellers with ‘no damage’ would be allowed to fly; the minor/major damage classification is purely for additional diagnostic information for the user. Therefore, my second metric was the percent of correct flight go/no-go classifications. For this metric, predicting an image as ‘major damage’ when it is ‘minor damage’ would not be an issue. I labeled this metric ‘Flight Prediction Accuracy’.

Training Performance

As mentioned above, the accuracy of the models were logged for each epoch and the results can be seen in the following figures.

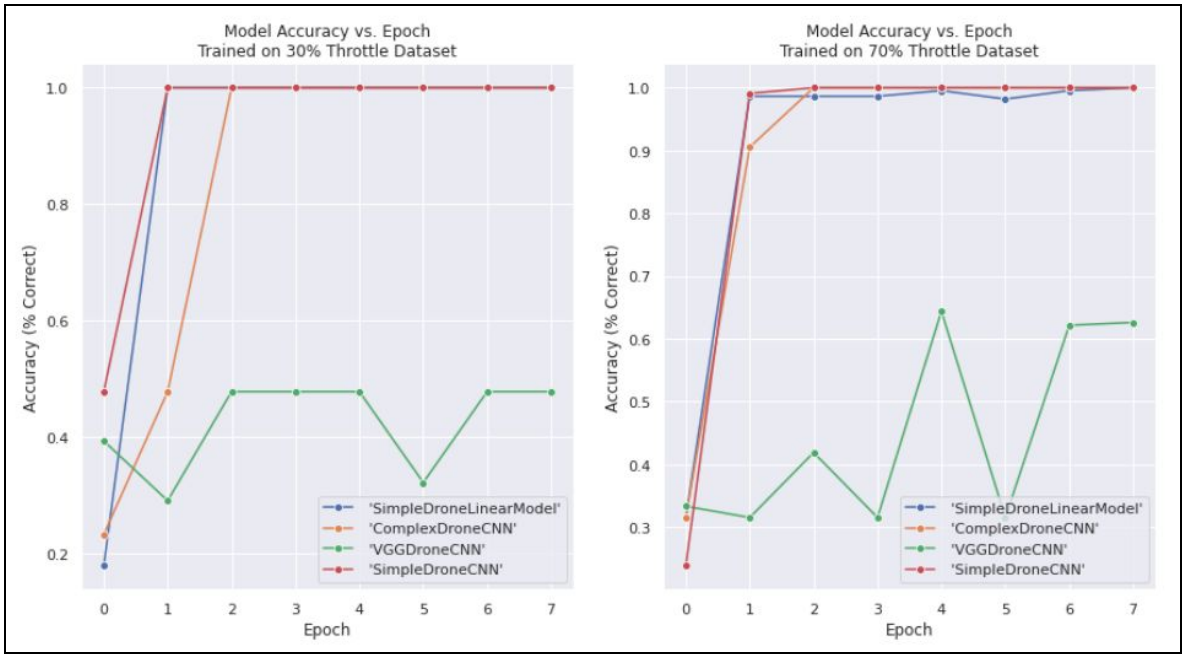

Figure 4: Model Accuracy vs. Epoch for 30% & 70% Dataset

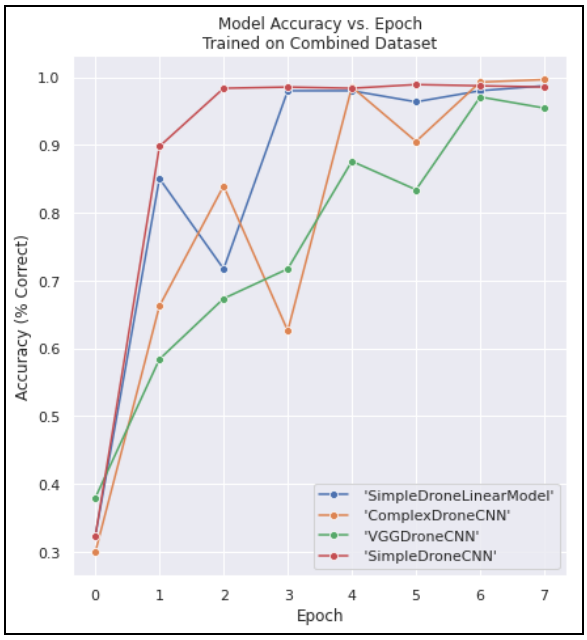

Figure 5: Model Accuracy vs. Epoch for Combined Dataset

By looking at these figures there are a couple of things that we can observe. It can be seen that all models except the VGG-based model did quite well on all three datasets. VGG on the other hand did poorly on the first two datasets, and then did well on the Combined one.

While good performance was expected, due to the fact that the differences can be observed both by listening to the audio signals, and by examining the FFTs and spectrograms, I did not expect the performance to be nearly 100% across the board. This definitely speaks to this being an easier classification task than I expected. In order to confirm that these results were genuine, I plotted the loss of one of the models just to verify that it went down over each iteration. As can be seen in Figure 6, the loss does in fact go down as the accuracy goes up.

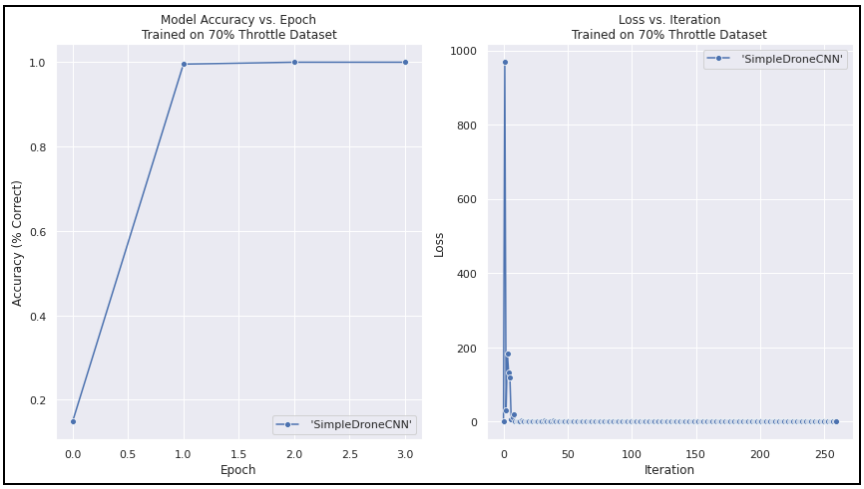

Figure 6: 'Model Accuracy vs. Epoch' and 'Loss vs. Iteration' for SimpleDroneCNN on 70% Dataset

Additionally, I expected the 'Combined' dataset to be the most difficult of the three. Thus, when the VGG-based model did very well on this dataset, despite not doing well on the other two, I was pretty surprised. There are a couple of reasons that could play a role in this. The 70% and 30% images are more homogenous than the combined dataset, as they are taken using the same throttle and recording equipment, which could result in the VGG-based model overfitting. As the Combined dataset has more variability between any two images with the same label, perhaps this reduces the overfitting concerns.

Another potential explanation could be due to the size of the dataset. The combined dataset has roughly 1800 images, which is more than twice as many images as the 30 and 70 datasets which have about 740 images each. The larger corpus of training data could lead to better performance as well. Finally, this could be due to the nature of the models themselves. For all the other models, I am backpropagating across all of the layers in the model. For the VGG-based model, however, I am using the pretrained VGG network as a feature extractor and then only backpropagating through the additional linear layers. As the VGG layers never 'see' the drone spectrograms this could result in worse performance.

Cross Evaluation Performance

As a result of training, there are a total of 12 unique combinations of models and train dataset. Each model was evaluated on all of the datasets, using the two metrics described above: classification accuracy and flight prediction accuracy.

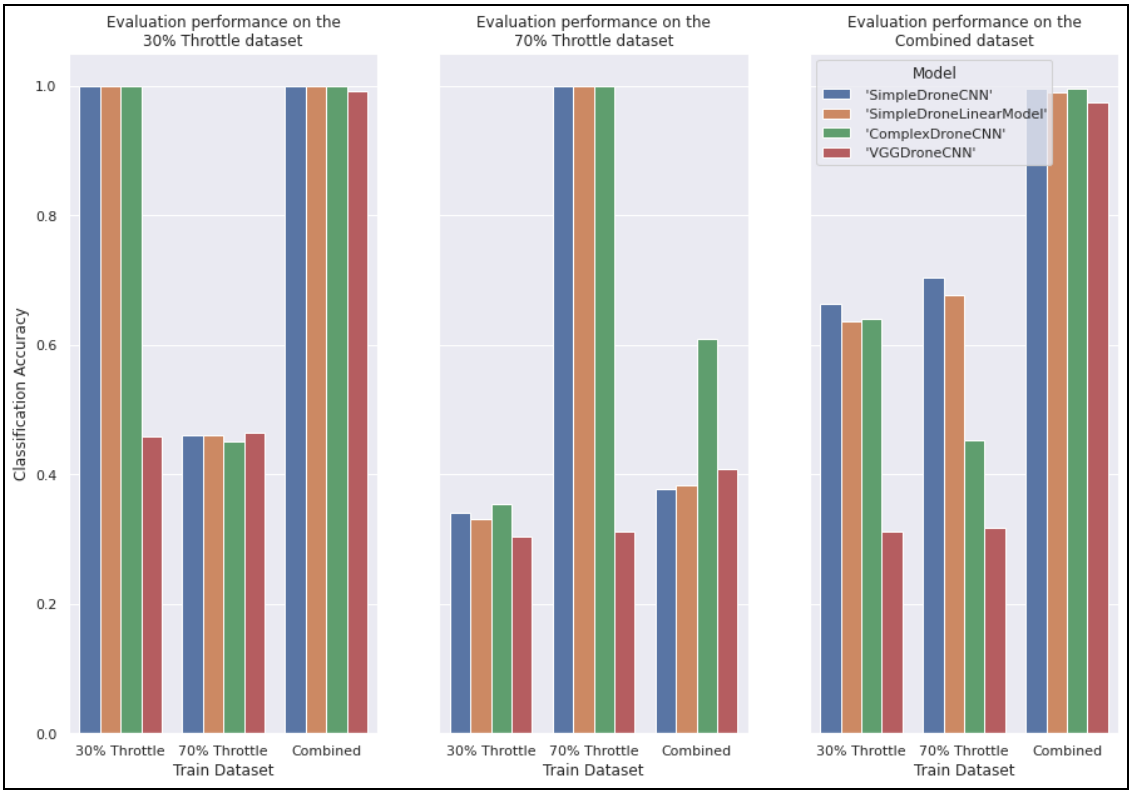

Figure 7: Classification Accuracy of Models on Each Dataset

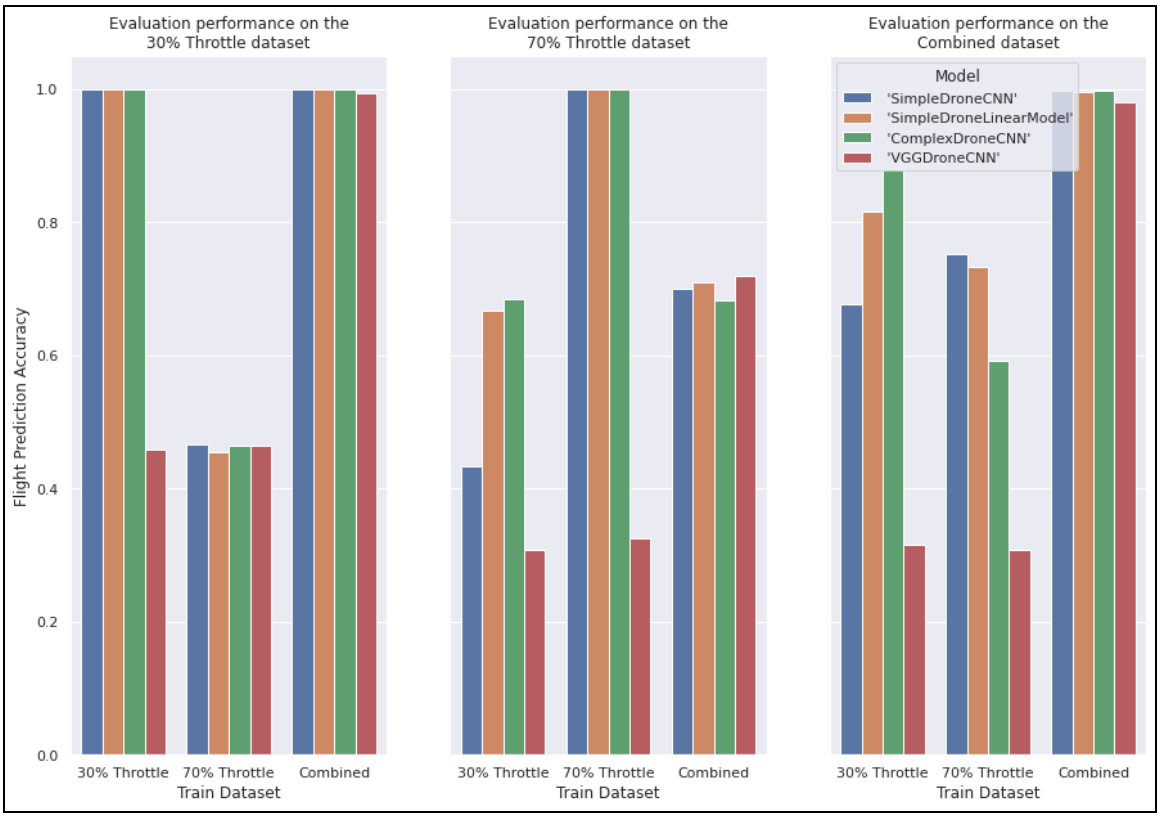

Figure 8: Flight Prediction Accuracy of Models on Each Dataset

There are a couple of interesting observations that can be made as a result of the above graphs. One would expect that for all of the models, they would have the best performance on the dataset that they were trained on. This is true in nearly all cases, except for the VGG-based model. For this model type, the one trained on the combined dataset does the best. This is probably due to similar reasons as explained above for why the VGG-based model had the best performance on the combined dataset.

Another thing to note here, is that for the other models, the models trained on the Combined dataset did second best when evaluated on the 70% and 30% data. This is to be expected, as the combined dataset also includes some spectrograms from that throttle range.

Additionally, we can see nearly perfect evaluation results for the 30% dataset on the models trained on the combined data. A potential explanation for this could be that the 30% audio signals could be similar to a lot of the ones in the combined dataset. The audio signals recorded at a lower throttle have a lower amplitude, as the propellers make less noise when spun at diminished power. The combined data includes iPhone and Raspberry Pi audio samples. which record more noise then the professional microphones. These low amplitude signals from the 30% dataset could be pretty similar to the noisier data in the combined dataset, especially when comparing the spectrograms. This would also explain why neither of those two datasets did as

well on the 70% data which has less noise due to the high quality microphones, and audio signals with large magnitudes due to the high throttle.

As expected, the 'Flight Prediction Accuracy' is higher than the 'Classification Accuracy' for every model-dataset pair. As it is a more lenient metric, this was the predicted outcome. The places where there is the largest difference between the 'Flight Prediction Accuracy' and 'Classification Accuracy' seems to be in the evaluation performance of the 70% throttle dataset by the models that were trained on either the 30% and Combined data. This corroborates with what was discussed earlier, with regards to the weaker signals in those two datasets. Perhaps those models classified some of the 70% data as 'major damage' because it was a louder sound, when they were in fact 'minor damage'.

Conclusions & Future Work

It can be seen that the system described in this paper had good success in identifying propeller defect types based on the acoustic signals they produce. However, an ideal UAV Preflight diagnostic system would be capable of identifying a myriad of defects on a drone prior to it taking off. Other potential defects include a misaligned chassis, or a cracked drone arm. Future work in the area of preflight diagnostics could include expanding this system, to identify these other defect types. This would involve collecting data from many different drones with different defects and labeling all of these defect types.

Another potential next step would be integrating onboard IMU data in addition to the microphone data currently used for this project. The drone currently has an accelerometer and a gyroscope which measure the drone vibration in the X, Y, and Z directions. Examining the spectrograms of these time signals would be an interesting next step. The primary benefit of using the IMU data would be that the entire system would be self-contained and wouldn't need to utilize any external hardware like a microphone.

One way to take the existing system to its final stage would be to develop a deploy-ready version of the code, in which a UAV operator could input their own audio file, and the program would output the propeller defect classification. To work towards this goal, I developed a function in the code to save all the models to a file, by leveraging Torch's save_dict function. The next steps would be saving the best model, and then putting together a standalone utility that would convert the user audio file to a spectrogram, and then run the trained model on the resultant spectrogram.

References

1. D. Scott, "Thicker Control Surfaces and Vortex Generators Explained | Model Aviation", Modelaviation.com, 2017. [Online]. Available: http://modelaviation.com/thicker-control-surfaces. [Accessed: 11- Dec- 2020].
2. Hannun, Awni & Case, Carl & Casper, Jared & Catanzaro, Bryan & Diamos, Greg & Elsen, Erich & Prenger, Ryan & Satheesh, Sanjeev & Sengupta, Shubho & Coates, Adam & Ng, Andrew. (2014). DeepSpeech: Scaling up end-to-end speech recognition.
3. D. Gartzman, "Getting to Know the Mel Spectrogram", Medium, 2019. [Online]. Available: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0. [Accessed: 11- Dec- 2020].
4. "Introduction · MAVLink Developer Guide", Mavlink.io, 2020. [Online]. Available: https://mavlink.io/en/. [Accessed: 11- Dec- 2020].
5. Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.