Predicting Tropical Cyclone Severity: A CV-Based Approach

Final Report

CSE 881
Data Mining
Professor Jiliang Tang
Michigan State University

Andrew McDonald
mcdon499@msu.edu

April 30, 2021

1 Abstract

In this project, we develop a model able to predict the wind speed of a tropical cyclone given only a greyscale satellite image of the storm. We leverage transfer learning and fine-tune three popular pretrained convolutional neural networks: ResNet-18, SqueezeNet v1.1, and ShuffleNet v2 1.0x. Over a 1,000 image test set, our best model achieves an RMSE of 11.69 knots, comparable with known baselines. Following an introduction to the problem and dataset, we review deep learning and the model architectures used in our experiments. We then present and discuss experimental results, recall key lessons learned, suggest directions for future work, and encourage machine learning researchers to join the fight against climate change. We provide all of our experimental code online via GitHub.[1]

2 Introduction

Hurricanes are among the most destructive, costly, and deadly natural disasters occurring on Earth, and are increasing in frequency and intensity with our warming climate [1, 2]. In order to plan, prepare, and evacuate coastal areas at risk of storm surge, flooding and wind damage, meteorologists must accurately measure or estimate winds for use in forecasting models—yet, this turns out to be a non-trivial task. Direct measurement of wind intensity via aircraft is expensive and dangerous, and existing techniques for estimating wind intensity from satellite imagery are subject to human judgement. Motivated by the broader movement to apply machine learning in the fight against climate change [3], we seek a computer vision-based approach to the task of tropical cyclone wind prediction with the potential to save lives.

2.1 Problem Setup

Let $\mathbf{X}_{s,t}$ be an square, greyscale satellite image of a tropical cyclone s at time t with dimension $d \times d$. We aim to develop a model $f : \mathbb{R}^{d\times d} \to \mathbb{R}$ which predicts the maximum sustained wind speed of the storm in this image, such that $f(\mathbf{X}_{s,t}) \approx y_{s,t}$, where $y_{s,t}$ is the true maximum sustained wind speed of storm s at time t. Specifically, we aim to solve the optimization problem

$$\min_f \mathbb{E}_{y\sim p(y|s,t)} \left[(y_{s,t} - f(\mathbf{X}_{s,t}))^2 \right],$$

where expected squared error is taken over the data-generating distribution $p(y|s,t)$, a distribution of maximum sustained wind speed y conditional on s and t. Though we have no way of knowing the true distribution of wind speeds $p(y|s,t)$, we assume access to a labeled dataset $\mathcal{D} = \{\mathbf{X}_i, y_i\}_{i=1}^N$ sampled from this distribution, and assume it is split into training and testing sets with $N_{\text{train}} + N_{\text{test}} = N$. Hence, allowing training on the N_{train} image-label pairs and holding out N_{test} image-label pairs for evaluation, we aim to construct a model which minimizes the empirical mean squared loss

$$\min_f \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left[(y_i - f(\mathbf{X}_i))^2 \right],$$

as this will approximate the expected mean squared error over all possible image-label pairs. In our experiments, we optimize against root mean squared error (RMSE): this leads to an equivalent solution, but provides better interpretability with units in knots (kt) instead of squared knots (kt^2). While f could be any type of machine learning model, we study convolutional neural networks (CNN) given their stellar performance on image-related tasks.

2.2 Data

We train and evaluate our model on the Tropical Cyclone Wind Estimation Competition dataset, originally constructed by the NASA Interagency Implementation and Advanced Concepts Team (IMPACT) in the

[1] https://github.com/andrewmcdonald27/CSE881Public

development of Deepti [4], a deep-learning-based tropical cyclone intensity estimation system available for interactive use online.[2] Following completion of the project within NASA, the dataset was released to DrivenData and used as the subject of an online predictive modeling competition in late 2020.[3] Now, the dataset remains available through the Radiant MLHub Training Data Registry [5].

The dataset consists of greyscale satellite images centered on the eye of tropical depressions, tropical storms, and hurricanes in the Atlantic and East Pacific Oceans from 2000 to 2019, paired with the corresponding maximum sustained wind speed of the storm in each image. Sample image-label pairs are presented in Figure 1. A total of 70,257 image-label pairs exist in the training set, with 56,205 image-label pairs reserved for testing. Due to computational constraints, we limit our consideration to an 11,000 image subset of the training set. From this, we reserve 1,000 images for model validation, leaving 10,000 for model training. We then compare models on a 1,000 image subset of the test set.

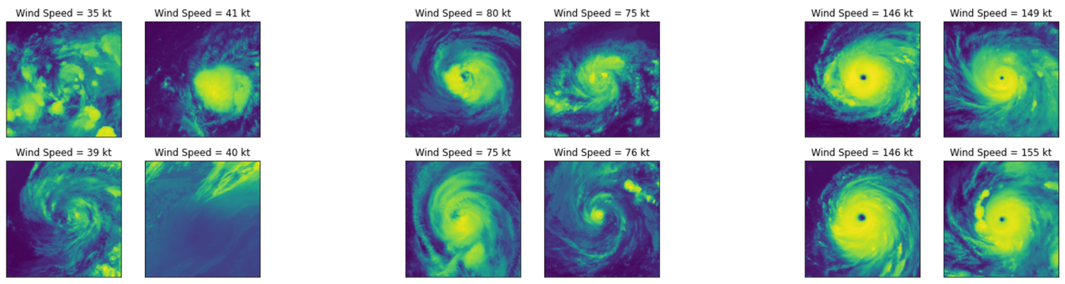


Figure 1: Randomly sampled image-label pairs from the dataset with wind speeds of approximately 40 (left), 80 (center), and 150 (right) knots. Note that the actual images are greyscale—we visualize them in Viridis for better perceptual quality.

Image-label pairs are associated with a unique storm id and time, as illustrated in Figure 2. Thus, one may exploit the temporal structure and autocorrelation among previous observations to predict future wind speeds for a given storm, if that is the goal. Forecasting is not our goal, however—we consider the simpler task of predicting maximum sustained wind speed y given an image $\mathbf{X}$ in isolation from contextual information related to time t and storm identifier s.

Wind speeds exhibit a long-tailed distribution, as do the number of images per storm, as illustrated in Figure 3. Thus, we expect difficulty in accurately predicting the wind speed of intense storms: training examples are few and far between. On the other hand, the long-tailed distribution of images per storm somewhat validates our ability to ignore temporal information: most storms have few images, hence a random sample of images is likely to come from distinct storms.

3 Experimental Methods

We begin by introducing the foundations of deep learning and convolutional neural networks essential to computer vision, then review the three CNN architectures used in our experiments: ResNet-18, SqueezeNet v1.1, and ShuffleNet v2 1.0x. All experiments are carried out by fine-tuning the pretrained models provided in the `torchvision.models` module of PyTorch [6]. We make use of the PyTorch-Lightning framework [7] to organize model training and evaluation logic, and draw inspiration from a number of tutorials.[4]

[2]http://hurricane.dsig.net/
[3]https://www.drivendata.org/competitions/72/predict-wind-speeds/
[4]https://www.drivendata.co/blog/predict-wind-speeds-benchmark/,
https://pytorch-lightning.readthedocs.io/en/latest/starter/new-project.html,
https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html,
https://pytorch.org/tutorials/beginner/data_loading_tutorial.html.

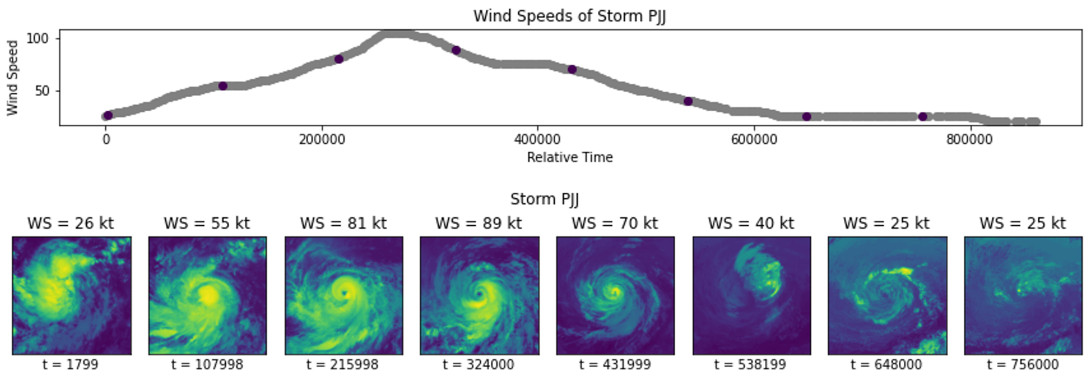


Figure 2: Wind speeds of a randomly selected storm over time, along with the corresponding satellite images. While this rich temporal structure may be of use in forecasting, we consider the simpler problem of de-contextualized wind speed prediction.

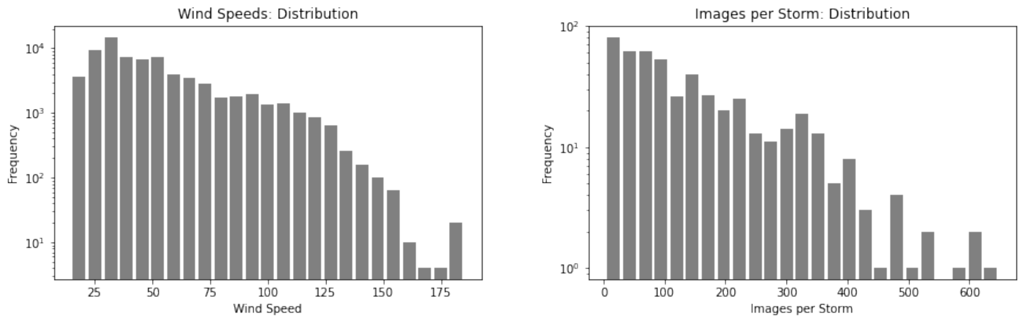


Figure 3: Distribution of wind speeds and number of images per storm in the Tropical Cyclone Wind Estimation Dataset [5]. Note that frequencies are plotted on a log-scale for ease of visualization.

Constrained by computational resources on a personal laptop without GPU support, we limit the scope of our experiments to relatively small (<50 MB) model architectures and training epochs. Nevertheless, we are still able to achieve success, which we discuss in Section 4. We encourage those with the computational means to do so to explore larger models, additional hyperparameter settings, longer training times and different architectures altogether.

3.1 Deep Learning

Neural approaches to machine learning draw inspiration from the human brain, constructing a mapping $\mathbf{x} \to y$ by composing the results of several intermediate functions—produced by so-called "neurons"—to generate an output. In their simplest form, modern networks compose the output of neurons in layer ℓ through a linear combination of outputs from layer $\ell - 1$, followed by the application of a nonlinear activation function σ allowing the network to learn nonlinear mappings:

$$h_i^\ell = \sigma\left(\sum_{i=1}^{k} w_i h_i^{\ell-1} + b\right).$$

First proposed in 1943 by McCullough and Pitts [8], the most basic neural model combined inputs through a weighted linear combination and applied a thresholding function to return a binary result:

$$y = \mathrm{th}_\delta\left(b + \sum_{i=1}^{k} w_i x_i\right) \quad \text{where} \quad \mathrm{th}_\delta(z) = \begin{cases} 1 & z \geq \delta \\ 0 & z < \delta. \end{cases}$$

While innovative, the McCullough-Pitts model suffered two major drawbacks: (i) it was only able to distinguish linearly-separable classes, and (ii) required a human to manually set the weights w_i and bias b. Rosenblatt overcame the second issue by proposing an optimization technique for the perceptron model [9], while Widrow and Hoff proposed stacking perceptron models in successive fashion to construct ADALINE [10], arguably the first neural network successfully applied in a practical setting. Minsky and Papert [11] diminished initial excitement in the field when pointing out the limitations of these early linear models and their inability to learn the XOR function, and the difficulty of optimizing complex nonlinear models. Thereafter, neural approaches to machine learning underwent several decades of insignificance.

An intermediate resurgence of neural learning came in the 1980s following Rumelhart's backpropagation algorithm [12], allowing the multilayer perceptron (MLP) neural network architecture to be used in practice. Applications to computer vision were highlighted by LeCun's original convolutional neural network (CNN) architecture [13] in the 1990s, which exploited the local structure and translational invariance present in images through the layering of convolutions and pooling operations. The architecture of LeNet-5 [13] is reproduced in Figure 4, and an amazing demonstration of the earlier LeNet-1 in action is available online.[5]

The modern age of deep learning began in 2012 following Krizhevsky's use of a deep CNN [14] to achieve state-of-the-art results on the ImageNet classification challenge [15], and has continued to grow in popularity thanks to an increased availability to computational resources, labeled datasets and improved optimization algorithms [16]. Notably, it has been proven that deep networks satisfying particular assumptions are able to approximate any function [17], giving theoretical backing to the practical success of the field.

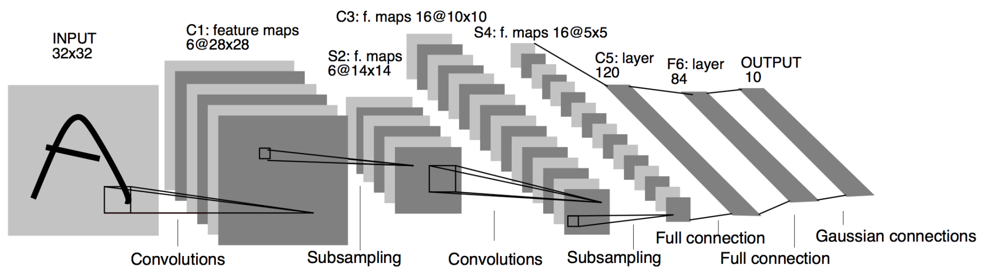


Figure 4: LeNet-5, a convolutional neural network designed for handwritten digit classification. Images, represented as matrices of size 32×32, are passed through sequential layers of convolutions, which extract features such as edges, corners and curves, then through pooling layers which downsample dimensionality. Weights in the network are optimized using backpropagation. Recent CNN architectures are more complex, but build upon the same general idea, exploiting the local structure and translational invariance found in images. Figure reproduced from [13].

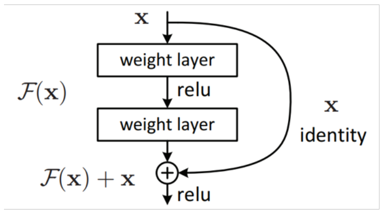
3.2 ResNet

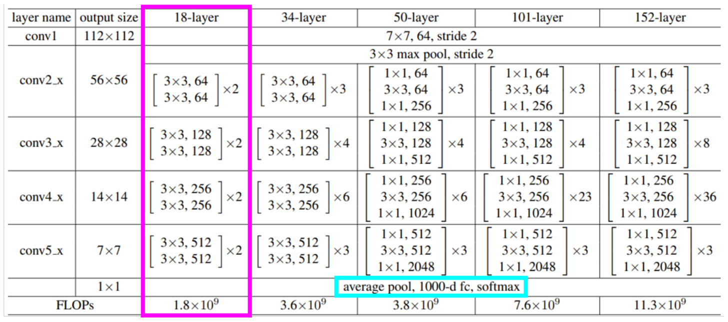
Proposed by He in [18], the ResNet architecture achieved 1$^{\text{st}}$ place at the 2015 ImageNet Large Scale Visual Recognition Challenge (ILSVRC2015), transforming the landscape of CV research. By making use of so-called skip-connections in which the output $h_i^{\ell-j}$ is fed as an input to h_i^ℓ for $j \geq 2$, the ResNet architecture tends to be easier to train than a strict feedforward architecture of the same size. Additionally, the authors

[5]One of my favorite videos: https://youtu.be/FwFduRA_L6Q.

explain that the residual architecture requires fewer FLOPS than comparable baseline architectures, thereby lowering model complexity without sacrificing the advantage of depth which allows for more sophisticated functional composition. Figure 5 presents the basic residual building block used in the ResNets, along with a table summarizing the architecture of the ResNet variants proposed in [18].

In our experiments, we leverage the ResNet-18 architecture, fine-tuning the pretrained `torchvision.models.resnet18` model. To adapt the model from its original use in classification to regression, we replace the final FC(512 $\to$ 1000) $\to$ Softmax(1000 $\to$ 1000) module with a FC(512 $\to$ 64) $\to$ FC(64 $\to$ 1) module.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 5: Left: The basic building block of the ResNet, consisting of a so-called skip-connection. Right: Various ResNet model architectures. We use the ResNet-18 architecture highlighted in pink, and replace the classification module highlighted in blue with a regression module. Figures reproduced from [18].

3.3 SqueezeNet

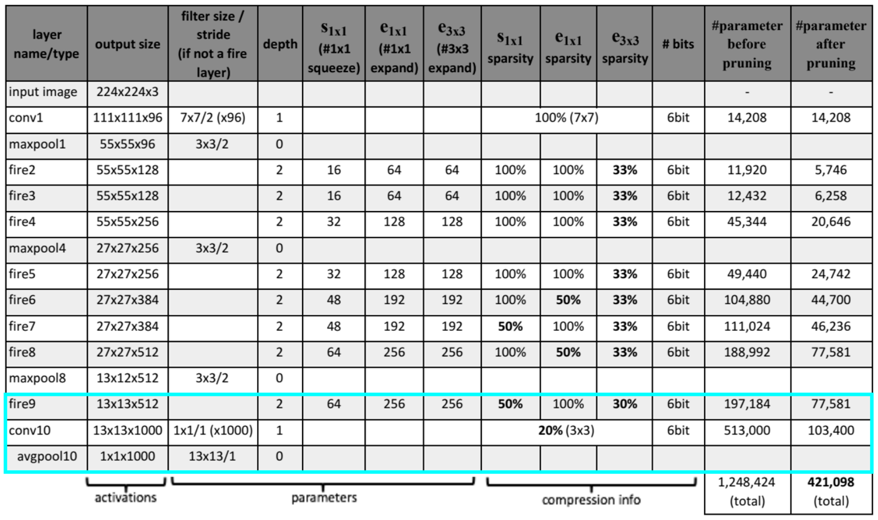
Proposed by Iandola in [19], the SqueezeNet architecture was designed with efficiency in mind, aiming to maintain the 57.2% top-1 and 80.3% top-5 accuracy of the 240 MB AlexNet [14] on the ImageNet benchmark at a significantly smaller model size. The authors are indeed successful in achieving this goal, maintaining 57.5% top-1 and 80.3% top-5 accuracy with a full-precision 32 bit model 4.8 MB, compressible to a 6 bit precision model of just 0.47 MB. Figure 6 outlines the architecture of SqueezeNet.

We make use of the pretrained `torchvision.models.squeezenet1_1` model, a revision of the original SqueezeNet 1.0 in [19] which further reduces computation and parameters while maintaining accuracy. We replace the final Fire9($13 \times 13 \to 512$) $\to$ Conv10($13 \times 13 \to 1000$) $\to$ AvgPool10($1000 \to 1000$) $\to$ Softmax($1000 \to 1000$) module with Fire9($13 \times 13 \to 1$) $\to$ Conv10($13 \times 13 \to 1$) $\to$ AvgPool10($1 \to 1$) to adapt the model to regression.

3.4 ShuffleNet

Proposed by Ma in [20], the ShuffleNet v2 architecture was designed to optimize for real-world speed as opposed to minimizing FLOPs, taking system architecture, memory access cost (MAC), degrees of parallelism and other overheads into account. The authors propose four design guidelines for runtime minimization, validated through numerical study on an NVIDIA GeForce GTX 1080Ti GPU and a Qualcomm Snapdragon 810 CPU, then use these guidelines to inform the construction of ShuffleNet v2:

- Equal channel width convolutions reduce MAC, and should be favored.
- Excessive group convolutions increase MAC, and should be avoided.
- Excessive fragmentation in network design reduces parallelism, and should be avoided.



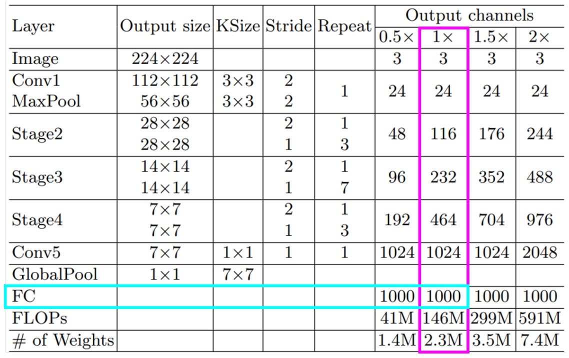
layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations	parameters					compression info				1,248,424 (total)	**421,098** (total)

Figure 6: SqueezeNet 1.0 model architecture. We replace the classification module highlighted in blue with a regression module. Figure reproduced from [19].

- Excessive element-wise operations increase MAC, and should be avoided.

The ShuffleNet v2 architecture offers speedup without sacrificing performance, achieving 69.4% top-1 accuracy on the ImageNet [15] benchmark. The architecture of ShuffleNet v2 is presented in Figure 7.

We utilize the pretrained `torchvision.models.shufflenet_v2_x1_0` model, the model with baseline (1.0x) complexity presented in [20]. We replace the FC($1024 \to 1000$) $\to$ Softmax($1000 \to 1000$) module with a FC($1024 \to 64$) $\to$ FC($64 \to 1$) module to adapt the network for regression.



Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

Figure 7: ShuffleNet v2 model architecture. We utilize the model with 1.0x complexity highlighted in pink, and replace the classification module highlighted in blue with a regression module. Figure reproduced from [20].

4 Experimental Results and Discussion

Despite of the aforementioned computational constraints of our experiments, our best model is able to achieve an RMSE of 11.69 knots on the 1,000 image test set, following two epochs of optimization over all 10,000 training images and two validation checkpoints over all 1,000 validation images. While this loss is not computed over the entire 56,205 image test set provided and is therefore a limited estimate, we note that this outperforms the benchmark RMSE of 12.69 knots presented in the DrivenData competition tutorial,[4] and 13.24 knots presented in the original Deepti publication [4].

Full experimental results are presented in Table 1, and the distribution of predictions compared to ground truth wind speeds is compared among models in Figure 8. Notably, our models produce predictions generally in line with the underlying wind speed distribution, never returning a wind speed below 15 kt or above 200 kt. However, we do note that the models tend to underpredict wind speed, particularly at the upper tail of the distribution. Surprisingly, all models perform comparably well in terms of RMSE, MAE and Pearson correlation.

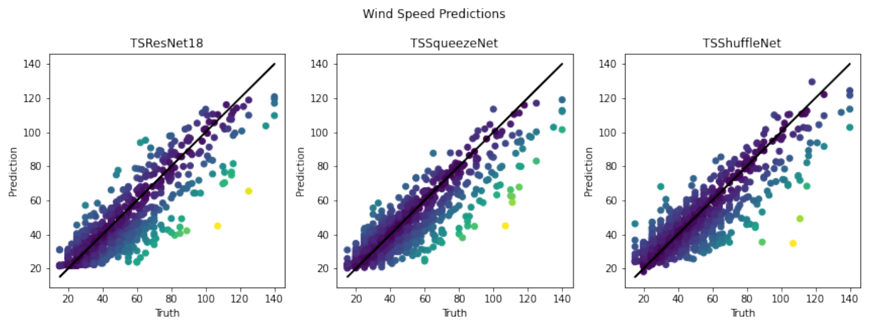


Figure 8: Predictions compared with ground-truth wind speeds for the ResNet-18, SqueezeNet and ShuffleNet architectures, from left to right. Black line denotes $\hat{y} = y$. Points are colored according to RMSE contribution.

	ResNet-18	SqueezeNet	ShuffleNet
RMSE	11.85512	11.78058	**11.68562**
MAE	8.466183	**8.220482**	8.350372
PearsonR	0.87516	**0.879434**	0.870252
Mean	42.05335	42.14731	43.98451
Std.	21.07747	18.71632	19.07181
Min.	21.46449	20.18949	18.15966
25%	27.26558	28.68371	30.94964
50%	35.27049	36.1914	37.58699
75%	47.68149	49.78672	51.14625
Max.	120.7348	118.9381	129.4513

Table 1: Model performance and summary statistics evaluated over a 1,000 image test set, following two training epochs over a 10,000 image training set. RMSE denotes root mean squared error, MAE denotes mean absolute error, PearsonR denotes Pearson correlation coefficient.

Training and validation loss curves of each model are presented in Figure 9, with ResNet-18 performance plotted in red, SqueezeNet performance plotted in orange, and ShuffleNet performance plotted in blue. It appears our models have converged on the train set, and perhaps even begun to overfit on the train set in the case of ResNet-18. Nevertheless, we expect that further training will reduce error, assuming we incorporate all 70,257 images from the original training set to avoid overfitting. Indeed, we know a lower RMSE is possible: the winner of the DrivenData competition, Igor Ivanov, achieved an RMSE of only 6.67 knots.[6]

[6] https://github.com/drivendataorg/wind-dependent-variables

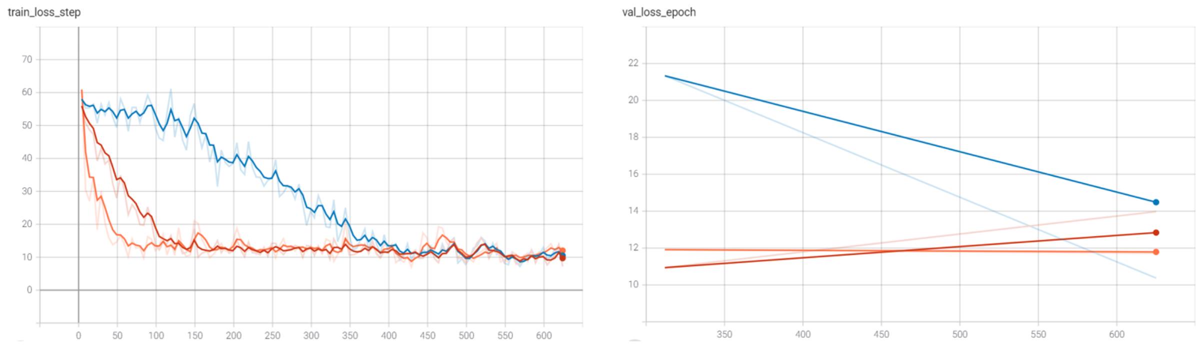


Figure 9: Training and validation loss curves for each model, with ResNet-18 plotted in red, SqueezeNet plotted in orange, and ShuffleNet plotted in blue. Despite computational constraints, we are still able to fit the training set and plateau in validation loss.

5 Lessons Learned and Future Work

This being the author's first significant project with deep learning, an uncountable number of lessons were learned: from the basics of PyTorch and PyTorch-Lightning to the intricacies of CNN architecture, everything was new. A few key takeaways stand out:

- **Start Small.** We began by attempting to fine-tune the ResNet-152 model, only to find a number of errors in our assumptions and training pipeline. Likewise, we attempted to train over the entire training set, only to find more bugs in our code. Deepening the network and expanding the amount of data used should always be the last step of the experimental process, following the construction of a minimal viable product.
- **Fail Fast.** Related to the previous note, we found fast failure to be quite valuable: we'd rather run a single minibatch through our model and realize our logger isn't configured properly, than to fail in the same manner after an overnight training run.
- **Avoid Boilerplate.** We found the deep learning loop of design and experimentation to be filled with a significant amount of repetitive, boilerplate tasks, obscuring the underlying art and science which brings joy to research. We eventually accelerated the experimental loop by writing helper scripts and leveraging the PyTorch Lightning library.
- **Don't Underestimate the Internet.** While deep learning is built upon publication, just as any other branch of science, wisdom is not limited to conference and journal papers: blog posts, forum threads, Twitter and YouTube are filled with gold. In particular, we found this blog post[7] by Andrej Karpathy quite helpful.
- **Try Again.** It's easy to get discouraged when results don't pan out—but that does not mean we have wasted our time. The greatest lessons from this project were borne from mistakes, and the models that do not converge make those which do far more rewarding. Grit is key.

Potential directions for future work have been discussed in previous sections, but we summarize them here:

- **Go Big.** Having built a working model, how will performance change as we increase the size of the training and testing sets? What about the network width and depth? How will data augmentation play a role? What effect will ensembling have? With access to a GPU or TPU, the game changes significantly.

[7] `http://karpathy.github.io/2019/04/25/recipe/`

- **Add Temporal Information.** If we view the wind speed prediction problem through a forecasting lens, how accurately can we predict future wind speeds given previous image-label pairs?
- **Talk.** The best way to learn is with a community—how did the winners of the competition develop their models? What observations did they make? What insights might they have?

6 Conclusion

In this project, we aimed to predict the wind speed y of a tropical cyclone, given a greyscale satellite image $\mathbf{X}$ of the storm. Constrained by computational resources, we fine-tuned three popular pretrained networks: ResNet-18, SqueezeNet v1.1, and ShuffleNet v2 1.0x. We were relatively successful in our approach, with our fine-tuned ShuffleNet achieving an RMSE of 11.69 knots over a 1,000 image test set after only two epochs of fine-tuning over a 10,000 image training set. We highlight the key takeaways from our experiment, and point out future directions for work on this problem, open-sourcing our code via GitHub.[1]

We believe our work illuminates the powerful role machine learning, deep learning and computer vision may play in 21$^{\text{st}}$-century earth science, and call on fellow researchers and practitioners to do their part in addressing the climate crisis.

References

[1] K. A. Emanuel, "The dependence of hurricane intensity on climate," *Nature*, vol. 326, no. 6112, pp. 483–485, 1987.

[2] K. Emanuel, "The hurricane-climate connection," *Bulletin of the American Meteorological Society*, vol. 89, no. 5, pp. ES10–ES20, 2008.

[3] D. Rolnick, P. L. Donti, L. H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A. S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown *et al.*, "Tackling climate change with machine learning," *arXiv*, 2019, arXiv preprint: https://arxiv.org/abs/1906.05433.

[4] M. Maskey, R. Ramachandran, M. Ramasubramanian, I. Gurung, B. Freitag, A. Kaulfus, D. Bollinger, D. J. Cecil, and J. Miller, "Deepti: Deep-learning-based tropical cyclone intensity estimation system," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 4271–4281, 2020.

[5] ——. (2020) Tropical cyclone wind estimation competition dataset. Version 1.0, RadiantMLHub. Accessed April 2021. [Online]. Available: http://registry.mlhub.earth/10.34911/rdnt.xs53up/

[6] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035, available at https://pytorch.org/.

[7] W. Falcon, "PyTorch Lightning," *GitHub*, vol. 3, 2019, available at https://github.com/PyTorchLightning/pytorch-lightning.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[9] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[10] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford University Electronics Labs, Tech. Rep., 1960.

[11] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry.* MIT Press, 1969.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[16] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning.* MIT Press Cambridge, 2016.

[17] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[19] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016, SqueezeNet v1.1 available at https://github.com/forresti/SqueezeNet/tree/master/SqueezeNet_v1.1.

[20] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131.