CS230

Detecting Tornado Formation from Radar Imagery (Computer Vision)

Handong Park
Department of Computer Science
Stanford University
hdparkhd@stanford.edu
https://github.com/hanadulcetjams/cs_230_final_project_with_tornet

Abstract

MIT's TorNet project for tornado detection [1] recently provided a benchmark dataset for training CNNs to identify whether a tornado is present in radar images. However, the dataset suffers from severe class imbalance with few examples of tornadoes. In this paper, we boost model performance by augmenting the dataset; by mirroring known examples to create new tornadic examples, we significantly improve the CNN's performance with a validation AUC of .8881 and CSI of .3546.

1 Introduction

Each year, around 1,200 tornadoes occur in the U.S. [2], threatening many lives and causing millions of dollars in economic losses [3]. Early tornado detection can save lives by allowing meteorologists to issue early warnings [4], but meteorologists struggle to detect tornadoes manually from radar; around 70% of tornado warnings are false warnings, leading the public to ignore even accurate warnings [2]. To address this, MIT's TorNet project collected tornado radar images to form a benchmark dataset and successfully trained a convolutional neural network (CNN) to detect tornadoes from radar [1]. Equipped with the same data, this project trains improved CNNs that achieve the same goal: given an input set of radar images, classify the radar imagery as either having a tornado or not. Our project aims to improve upon Tornet's initial results, first by using data augmentation to further address class imbalance, and then by using transfer learning with a pre-trained YOLOv5 classification model.

2 Related work

Most tornado detection work is done manually (either in-person or with radar). Storm spotters report on in-person observations [5], and meteorologists look for specific patterns in radar images indicating tornado formation, such as hook echoes [6] or tornado vortex signatures [7] [8]. But storm spotting is risky, requiring spotters to tread dangerously close to tornadoes' paths, while meteorologists from afar are imprecise, with a false positive rate on tornado warnings greater than 70% [2].

However, recent efforts integrate deep learning into meteorology via CNNs. Of particular note is TorNet [1], whose architecture and dataset is utilized here. TorNet achieved an AUC of .8742 on their U.S. radar imagery dataset with a CNN of VGG-like blocks of CoordConv [9] and pooling layers. Similar work (which also used CNN's for multi-task learning) succeeded with Chinese tornado weather data [10] and with hurricane intensity estimation (using CNNs with satellite imagery) [11].

Still, in all previous work, a fundamental problem still remains — severe weather events are extremely rare. Even with 10 years of data, TorNet only found 13,857 tornadic examples [1]. To address this

CS230: Deep Learning, Fall 2024, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

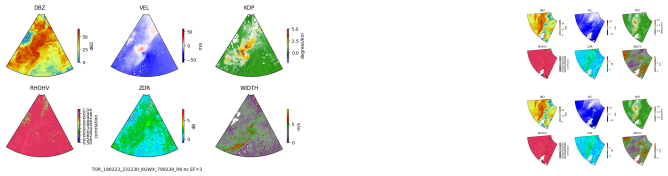
issue, our project will investigate whether mirroring is a promising data augmentation strategy for severe weather detection (by first testing whether it improves tornado detection).

We have also seen that YOLO (You Only Look Once) models are successful in detecting the presence of mold on surfaces with even just a small dataset (of roughly 2,050 images) used in transfer learning [12]. In general, mold has been successfully detected using transfer learning from pre-trained models [13]. We theorize that similar efforts with tornadoes could prove fruitful; tornado cells on a radar image could stand out as similarly-discolored "blob" shapes against a mostly uniform background.

3 Dataset

In April 2024, MIT's TorNet dataset [1] was published. It contains 203,133 examples of weather radar image-sets, of which 13,857 are from tornadoes; remaining examples contain random weather or false warnings. Each set, provided in NetCDF file format, contains images from two different tilt angles with respect to the horizon (0.5° and 0.9°). The data is split into train and test sets by year; for each image in each input, image data is given by an array with 120×240 pixels. Veillette et al. prevented data leakage by making sure that training examples were at least 30 minutes away in time or at least 0.25 degrees of latitude/longitude away from any validation example [1]. The resulting dataset was split into 171,666 examples of training data, and 31,467 examples of testing data [1].

For each tilt angle, six radar images are provided for a 60-degree sector (with radius 80 km.), which show reflectivity factor (DBZ), radial velocity (VEL), specific differential phase (KDP), correlation coefficient (RHOHV), differential reflectivity (ZDR), and spectrum width (WIDTH). When re-training YOLOv5's classification model using transfer learning, the model API required input images; hence, we created concatenated renderings of these sets of radar sweeps, which resulted in 1200×1200 images that each contained both the upper and lower tilt sweeps with all six images for both sweeps.



(a) Rendered example of a radar image-set for an EF-3 tornado from the dataset [1].

(b) An example of a concatenated input image for YOLOv5 containing both tilt sweeps.

Figure 1: Example data from the TorNet dataset [1].

4 Methods

Our plan is to experiment on different CNNs to detect the formation of tornadoes given an input set of six radar images for each of the two radar tilt angles. The goal is to improve the binary classification conducted in the original TorNet paper. To do so, we proceed with two main ideas:

- Create new example tornadoes via image mirroring to address the dataset's class imbalance.
- Utilize a pre-trained YOLOv5 classification model to transfer learn for tornado detection.

4.1 Mirroring tornadic training images to address class imbalance

Given that TorNet's CNN was already quite successful, this method addresses one of its main challenges: the severe class imbalance due to the rarity of tornadoes. With only 13,857 tornadoes out of its 203,133 samples (less than 7% of the dataset) [1], TorNet's class imbalance could have been one its largest barriers to binary classification success when detecting tornado presence.

Initially, we theorized that data augmentation via cropping or translation of tornadic example images was tempting, but soon realized this might not work. The radar images are polar and also capture imagery at a slant rather than on a flat, top-down angle [14]; translations would warp the storms in the images and could create weird "squishing" artifacts that our model could learn from mistakenly.

However, a promising data augmentation solution here was to reflect the data across the center (which avoids unintentionally squishing or stretching the image). Doing so could generate synthetic examples of tornadoes quite similar to real tornadoes. Hence, we propose the following strategy:

For each input radar imageset in the training data:

- Check whether the data depicts a tornado (according to its label).
- If not, simply append the data as is to our new training dataset.
- If so, add both the original data and a mirrored version (that flips all input radar image array data vertically across the center) to the new training dataset.

If this solution worked, we would correct the class imbalance in the dataset by providing doubly as many tornadoes when training the model - which could result in better performance.

While we augmented the dataset, we used the original TorNet CNN (to hold the architecture as a constant so that we could assess the benefit of dataset augmentation) with the following architecture:

- Six input layers (one per radar type), and the hard-coded coordinates and range-folding mask added to be used in CoordConv [9], all normalized to a $(-1, 1)$ scale and concatenated.
- Four VGG blocks of CoordConv [9] and MAXPOOL layers, each block consisting of one CoordConv layer (with filter size 3, same padding, and L2 regularization) and two MAXPOOL layers (each with filter size 2, stride of 2, and same padding).
- One final VGG block with regular CONV layers and a global MAXPOOL.

We normalize inputs to help minimize binary cross-entropy loss efficiently (rather than having asymmetrically-stretched loss along different input axes), while range-folding hides radar echoes that are outside of the radar's maximum range [15]. Each convolutional layer extracts features from each small region using a 3×3 filter, and then a pooling layer picks out the strongest activation from each 2×2 block. The final global pooling layer outputs a logit with likelihood of tornadic presence. Our project uses this same architecture but trains it against the mirroring-augmented training dataset using mini-batch gradient descent, and assesses AUC-ROC on the full test dataset to assess performance.

4.2 Applying transfer learning from pre-trained YOLOv5 classification model

While the TorNet dataset of tornado radar imagery is not extremely small (with 203,133 examples), it's still not as large as image sets seen in many image classification tasks. With so few tornadoes (13,857), it's worth wondering whether a transfer learning strategy could work. Even when our dataset is not large, starting with a pre-trained CNN that has been trained on millions of images can allow us to take useful feature extraction layers from that pre-trained CNN and apply them to a new dataset of much smaller size (even for task datasets as small as 2,050 images [12]).

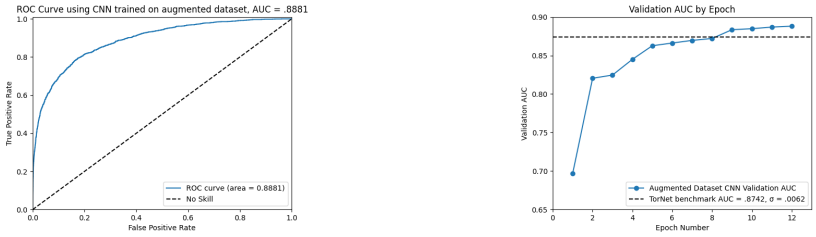
Unfortunately, we can't use the object detection algorithm that YOLOv5 typically uses without bounding box labels on our dataset, but we can still use a pre-trained YOLOv5 classification model to benefit from transfer learning. We use the pre-trained YOLOv5 classification provided by Ultralytics; we freeze early layers in the pre-trained CNN before re-training later layers with our modified, concatenated images from the TorNet dataset. Given the small scale of our dataset, we hope that transfer learning from a larger image dataset can deliver useful predictions and improve AUC.

5 Results

5.1 Significant improvements in model performance when augmenting the dataset with mirrored tornadoes

First, hyperparameters needed to re-train TorNet's CNN architecture from scratch were tuned to fit hardware limitations when performing mini-batch gradient descent to minimize the binary cross-entropy loss function (as is typical for binary classification). A batch size of 64 examples worked best (the largest size that our GPU could fit in memory). After searching on a log scale to determine learning rate, we settled on an initial learning rate of $\alpha = 10^{-4}$. With trial and error on the original dataset, it was discovered that manually decaying learning rate for later epochs helped the model converge more easily (by dampening noisy updates from mini-batching closer to the minimum).

After 12 epochs of training with the augmented dataset (9 epochs with the initial learning rate of $\alpha = 10^{-4}$, and 3 epochs with $\alpha = 10^{-5}$), training ended according to early stopping principles; while training loss continued to decrease with each epoch, test set validation loss had stopped decreasing, so training was halted to avoid overfitting (more details provided in Appendix 1a).



(a) ROC curve for the final model, with a validation AUC of .8881.

(b) Validation AUC increases by epoch, with multiple later models outperforming TorNet's benchmark.

Figure 2: Performance improves when a CNN is trained using an augmented dataset with additional mirrored tornadic examples. Note that validation AUC by epoch shows that even models trained on fewer epochs than the final model showed improvements over the TorNet baseline AUC.

Our final model has an improved AUC of **.8881**. This is a .0139 improvement over the previous baseline set by the TorNet paper [1], which had an AUC of .8742 with standard deviation of $\sigma = .0062$. With a Z-score of 2.242, the chance of observing an AUC this high given equal performance would be only 1.24% according to a one-sided Z-test, easily meeting the 5% statistical significance cutoff.

Our final model also has a better critical success index (CSI) than the original model, with a CSI of **.3546**; CSI is another metric used to measure performance, given by

$$CSI = \frac{TP}{TP + FP + FN}$$

where TP = true positives, FP = false positives, and FN = false negatives (from TorNet [1]). Increasing CSI aims to detect many tornadoes as possible while reducing false positives and negatives.

Our model's CSI of .3546 is a strong improvement over TorNet's original model CSI of .3380 with a standard deviation of $\sigma = .0063$ [1]. With a Z-score of 2.635, the chance of observing a CSI this high given only equal performance would be merely 0.42% - again a statistically significant improvement.

	AUC	CSI	F1-Score	Precision	Recall	Top Accuracy
Original TorNet CNN	.8742	.3380	NP	NP	NP	**.9499**
Mirroring-Augmented CNN	**.8881**	**.3546**	.5235	.5325	.5148	.9498

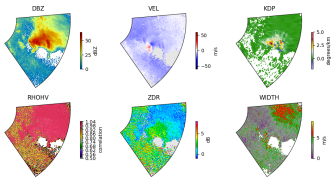
Table 1: Comparing results shows that our mirroring-augmented CNN improves on the original TorNet CNN [1] in classification AUC and CSI (NP designates that a value was not provided by [1].)

Hence, we find that our model trained on an augmented dataset is not only more accurate in predictions (with increased AUC), but also increases the ratio of true tornado detections to false positives and negatives. With these improvements over TorNet's CNN, we thus conclude that our new, mirroring-augmented model does outperform the previous benchmark set by the TorNet project.

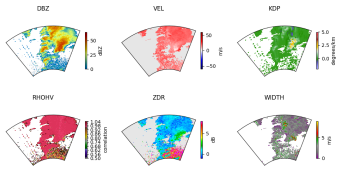
5.2 Strengths and weaknesses of mirroring-augmented model

When manually inspecting our mirroring-augmented model's predictions, we see a trend — in general, even with noise in surrounding areas, if there is a clear area of contrasting radial velocity (VEL) (indicating "high-speed spin") and a strong central cell of high KDP (indicating heavy rainfall, hail, or debris), the model does quite well at detecting it. Even when said signals aren't as easily pinpointed as tornadic from a human viewpoint, the model detects them with greater sensitivity than human eyes.

As a result, many false positives (which the mirroring-augmented CNN falsely classifies as "tornadoes") are "close misses". The distribution of predicted probability of tornadic presence for false positives is right-skewed, with most probabilities between 30 - 50%. From visual inspection, many



(a) Model correctly detects tornado with $p_{tornado} = .729$, despite noisiness in left side of several images.



(b) Model correctly detects tornado with $p_{tornado} = .925$ (not easily identifiable by visual inspection).

Figure 3: True positive examples from mirroring-augmented model. See Appendix 1b for more.

such cases have a reasonably visible contrast in VEL with a weak "spin" around a central point and have heavy KDP at that point. If precision was prized above all, we could avoid such false positives by raising our probability threshold for classifying a set of images as tornadic, but risk missing some tornadoes. (See Appendix 1b for more discussion of close misses, bad misses, and false negatives.)

In general, it seems that given similar accuracy (but higher AUC and higher CSI) compared to the original TorNet model, our augmentation with synthetic flipped tornadoes has the intended effect — by adding synthetic "tornadoes", we correct the class imbalance and make the CNN more sensitive to tornadoes. The cost is additional false positives (and hence .0001 less accuracy), but the benefit is a boost in performance. With higher AUC (better performance) and CSI (more true positives compared to false positives and negatives) and the visual trends discussed, it's likely that our model decreases false negatives significantly at the cost of a few more false positives. Given that our model is still 53.25% precise (over 20% better than human performance [2]), the trade-off seems worthwhile.

5.3 Subpar performance from transfer-learned YOLOv5 classification model

Comparatively, results from our YOLOv5 classification transfer learning were nowhere near as strong (hypothetical reasons are discussed in the "Conclusion & Future Work" section). In TorNet's paper, 1 to 2 years of data sufficed to train reasonably strong models. But when the pre-trained YOLOv5 classification model was trained with 1 to 2 years of data, it performed poorly, with only a best observed validation accuracy of around .908. Additional discussion is provided in Appendix 2.

6 Conclusion & Future Work

Mirroring-based dataset augmentation significantly improves the performance of the CNN by reducing false negatives in tornado detection. As theorized, the mirrored examples of tornadoes address a critical problem with the original dataset; by adding synthetic tornadoes similar to real ones, mirrored examples correct the class imbalance caused by relative rarity of tornado occurrence. Hence, we concluded that clearly, flipping the radar imagery is viable — researchers working on tornado detection should continue augmenting datasets with flipped examples as demonstrated here. The technique might even prove fruitful for other similarly rare events (such as hurricanes).

However, another important takeaway is that the TorNet CNN might still have even more room to improve — it may just need more data. Our model shows that a main flaw of the original model was the class imbalance present in the data, and succeeds by doubling the number of tornadic training examples; that augmentation corrects class imbalance enough to significantly increase AUC and CSI. If synthetic examples help, then adding more real-life tornadoes to the dataset could help further. Future work should expand the TorNet dataset to provide more fodder for CNN-based classification.

As for the YOLOv5 classification models, we have two hypothetical reasons for their poor performance. First, without bounding boxes for the tornadoes, we can't utilize YOLOv5 object detection. While we predicted that a pre-trained object classification CNN might still provide useful improvements from transfer learning, that result didn't materialize. With additional time, we could add bounding boxes to use pre-trained YOLOv5 object detection (rather than the classification CNN). Finally, because the 12 radar images were combined into single input images, each "concatenation" concentrated individual radar plots into tiny patches - so the YOLOv5 CNN had fewer pixels of interest to learn from. Future YOLOv5 efforts could fix this by focusing on one radar image at a time.

7 Contributions

This project was a solo project.

References

[1] M. S. Veillette, J. M. Kurdzo, P. M. Stepanian, John, S. Samsi, and J. McDonald, "A Benchmark Dataset for Tornado Detection and Prediction using Full-Resolution Polarimetric Weather Radar Data," arXiv.org, 2024. https://arxiv.org/abs/2401.16437 (accessed Oct. 02, 2024).

[2] "An AI dataset carves new paths to tornado detection | MIT Sustainability," Mit.edu, Apr. 29, 2024. https://sustainability.mit.edu/article/ai-dataset-carves-new-paths-tornado-detection (accessed Oct. 02, 2024).

[3] "How much damage did tornadoes cause in 2022?," USAFacts. https://usafacts.org/articles/how-much-damage-did-tornadoes-cause-in-2022/

[4] N. US Department of Commerce, "Media Roundup: Early Warnings Saved Lives on November 17," www.weather.gov. https://www.weather.gov/news/131121-media

[5] "Tornado Detection," NOAA National Severe Storms Laboratory, 2015. https://www.nssl.noaa.gov/education/svrwx101/tornadoes/detection/

[6] G. S. Forbes, 'On the Reliability of Hook Echoes as Tornado Indicators', Monthly Weather Review, vol. 109, no. 7, pp. 1457–1466, 1981.

[7] R. A. Brown and V. T. Wood, 'The Tornadic Vortex Signature: An Update', Weather and Forecasting, vol. 27, no. 2, pp. 525–530, 2012.

[8] "Locating Tornadoes: hook echoes and velocity couplets," ww2010.atmos.uiuc.edu. http://ww2010.atmos.uiuc.edu/(Gh)/guides/rs/rad/appl/trndo.rxml

[9] R. Liu et al., "An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution," arXiv:1807.03247 [cs, stat], Dec. 2018, Available: https://arxiv.org/abs/1807.03247

[10] J. Xie et al., 'Multi-Task Learning for Tornado Identification Using Doppler Radar Data', Geophysical Research Letters, vol. 51, no. 11, p. e2024GL108809, 2024.

[11] R. Pradhan, R. S. Aygun, M. Maskey, R. Ramachandran, and D. J. Cecil, 'Tropical Cyclone Intensity Estimation Using a Deep Convolutional Neural Network', IEEE Transactions on Image Processing, vol. 27, no. 2, pp. 692–702, 2018.

[12] F. Jubayer et al., "Detection of mold on the food surface using YOLOv5," Current Research in Food Science, vol. 4, pp. 724–728, 2021, doi: https://doi.org/10.1016/j.crfs.2021.10.003.

[13] P. Treepong and N. Theera-Ampornpunt, "Early bread mold detection through microscopic images using convolutional neural network," Current Research in Food Science, vol. 7, p. 100574, Jan. 2023, doi: https://doi.org/10.1016/j.crfs.2023.100574.

[14] Government of Canada, "Radar Image Distortions," Canada.ca, Nov. 20, 2015. https://natural-resources.canada.ca/maps-tools-and-publications/satellite-imagery-elevation-data-and-air-photos/tutorial-fundamentals-remote-sensing/microwave-remote-sensing/radar-image-distortions/9325 (accessed Nov. 14, 2024).

[15] B. Muller, "range_folding," Erau.edu, 2024. https://wx.erau.edu/faculty/mullerb/Wx365/Range_folding/range_folding.html (accessed Dec. 02, 2024).

[16] Ultralytics, "ultralytics/yolov5," GitHub, Aug. 21, 2020. https://github.com/ultralytics/yolov5

[17] Martín Abadi et al., 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems'. 2015.

[18] F. Pedregosa et al., 'Scikit-learn: Machine Learning in Python', Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

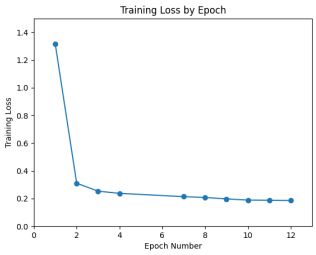
[19] T. Kluyver et al., 'Jupyter Notebooks – a publishing format for reproducible computational workflows', in Positioning and Power in Academic Publishing: Players, Agents and Agendas, 2016, pp. 87–90.

[20] A. Paszke et al., 'PyTorch: An Imperative Style, High-Performance Deep Learning Library', arXiv [cs.LG]. 2019.

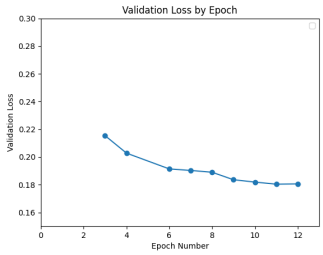
8 Appendix 1: Additional results and examples from mirroring-augmented CNN

8.1 Appendix 1a: Early stopping rationale + data

It was noted above that training was halted after epoch 12. Below are the graphs of training loss and validation loss by epoch (we trained with $\alpha = 10^{-4}$ for the first 9 epochs, then $\alpha = 10^{-5}$ for the last 3 epochs). Of note, validation loss slightly increased in epoch 12 (and had been flattening out over the previous epochs), so we chose to stop there.



(a) Training loss decreases by epoch, and continues to decrease slightly through epoch 12.



(b) Testing/validation loss decreased by epoch, but increased slightly at epoch 12 (so training was halted).

Figure 4: Plotting training and validation loss by epoch, to show why early stopping principles dictated that we stop training after epoch 12, at which point validation loss had flattened out. (Note that a few values were accidentally overwritten while running training and testing in Jupyter notebooks.)

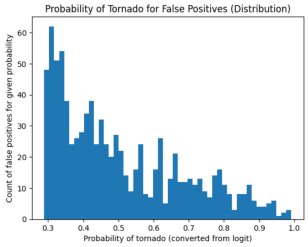
8.2 Appendix 1b: False positives were mostly "close misses"

As mentioned above, false positives were mostly not "bad" misses, but instead "close misses" - in the sense that the model still gave a relatively lower probability of a tornado being present there. To justify this, we show the actual distribution of both the false positives and the true positives to show the difference in distribution.

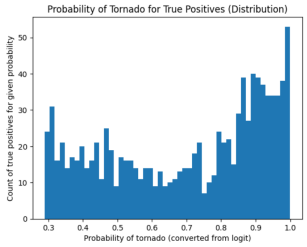
To convert the raw logit x of our CNN output to a prediction, we must take:

$$p = \frac{e^x}{1 + e^x}$$

to get a probability between 0 and 1. If that probability is greater than a given threshold, we predict a tornado as present, and else, we predict no tornado. At the best-performing prediction threshold for F1-score and CSI, we observed the following false + true positive distributions of predicted probabilities:



(a) False positives are mostly right-skewed, with most false positives with a probability between .3 to .5 in tornado presence probability.

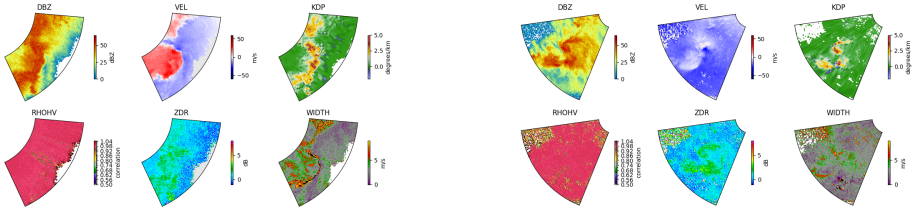


(b) In comparison, many true positives have $p_{tornado}$ between .7 and 1, with a left-skewed distribution (but also some low-likelihood predictions as well).

Figure 5: Distribution of predicted probabilities for false positives and true positive classifications.

Examples of true positive predictions that were "clear-cut" cases can help us understand exactly what kinds of false positives and false negatives we were seeing. A couple tornadoes that are very easy

to spot are shown below: the strong, concentrated difference in radial velocities spinning around a central point, along with high KDP (signaling very heavy rain or even hail/debris) seem to be strong indicators that the model leaned on, given the examples that were manually inspected.

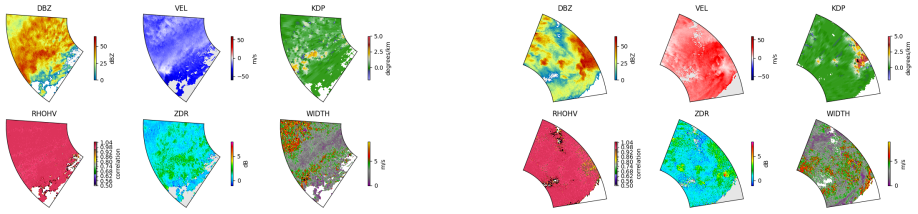


(a) Predicted probability of .918; observe the heavy contrast and rotation present in VEL, as well as the heavy KDP signaling heavy rain spun around the same areas.

(b) Predicted probability of .815; the radial velocity spin in VEL is a little less contrasting at first glance (potentially decreasing probability), but still visible.

Figure 6: More "clear" tornado examples where the probability of tornado detection predicted was quite high.

In comparison, false positives often featured similar-looking (albeit weaker) contrasts or heavy storms with similar visual indicators in VEL, KDP, and DBZ. The examples shown below in Figure 7 are "close misses" that also had similar-looking radial velocity spin patterns (weak, but centered around a small central cell) with weak but elevated KDP and DBZ.



(a) Predicted probability of .334; the small area in the lower-left corner of the VEL, KDP, and DBZ shows a weak but similar patterning of spin and heavy precipitation (but isn't actually a tornado).

(b) Predicted probability of .363; the small area in the lower-right corner of the VEL, KDP, and DBZ again shows a weak but similar patterning of spin and heavy precipitation (but isn't actually a tornado).

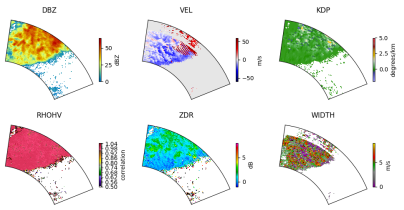
Figure 7: False positive examples that were "close misses", with weak, "tornado-like" visual indicators.

It's worth noting that a good number true positive detections also were weak tornado predictions. Given that the success of our mirroring-augmented model hinged upon reducing false negatives by increasing true positives via better class balance with more positive examples, it's probably worth still having a few additional false positives as a trade-off to catch more "hard-to-see" tornadoes.

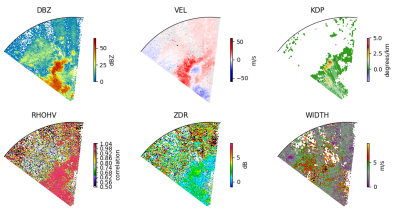
Meanwhile, a whole other class of false positives and false negatives actually involved noisy radar images or unusual "lines", "patches", "gaps", or occlusions in the data that seem to obstruct the "view" of the radar when looking at a storm cell (see Figure 8). In particular, if those lines appeared to make it seem as though there was an extremely severe contrast in radial velocity along those lines, it could confuse the CNN into thinking that there's a tornado present when there wasn't one.

Our hope is that in such noisy cases where inputs aren't very clear or clean from the images provided, we wouldn't be using the CNN to try to detect what's going on (we could instead try to grab an image just before or just after this given data point on the next sweep of the radar).

As for false negatives - sometimes, there wasn't enough visual indication that a tornado might be present from the images provided, and the CNN could not find much to detect. (Of note, as pointed out earlier, the model actually did spot some of these "hard-to-find" cases and made true positives where we would otherwise have false negatives - otherwise, we couldn't have boosted both AUC and



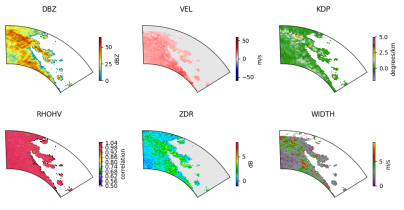
(a) Predicted probability of .911; the preponderance of noise, along with lines that go over the central cell might be confusing the CNN to think there's a severe contrast in VEL, causing the high likelihood.



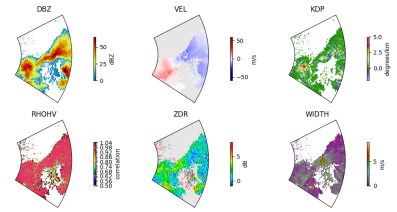
(b) Predicted probability of .904; we do see the VEL contrast and spin, but also a lot of noise in three of the channels is potentially causing additional difficulty in classification here.

Figure 8: False positive examples that were potentially caused by noisiness or odd lines in data.

CSI while maintaining an equal accuracy.) Alternatively, in other cases, the model might also not have interpreted two separate, slightly spread-out areas as comprising one larger tornado. Examples of both cases are shown below.



(a) Predicted probability of .017; this looks like a very normal non-tornadic storm cell, not much contrast or spin in VEL, maybe a tiny bit of high KDP in the upper left quadrant. Not much about the radar data seems tornadic, despite a tornado having been confirmed in this case.



(b) Predicted probability of .017; a bit more surprising of a false negative here given the contrast in VEL and DBZ. It's possible that maybe having multiple separate blobs in DBZ and KDP caused confusion (as compared to a single strong cell).

Figure 9: False negative examples for our mirroring-augmented CNN.

Despite some of these less-successful predictions from the model, the boosts to AUC and CSI show that our mirroring-augmented model is still more effective than the original TorNet CNN in spite of some classification errors.

9 Appendix 2: Results from transfer learning of pre-trained YOLOv5 classification CNN

After freezing different numbers of layers (between 5 to 8 of the blocks of the original CNN), training remaining layers over 100 epochs with a batch size of 4 (due to memory constraints from the GPU), and early stopping to prevent overfitting, we only saw a maximum accuracy of .908, nowhere near the .949 seen in earlier methods.

The Ultralytics Python packages provide us with automated result plotting when running several epochs of training with transfer learning against a given initial pre-trained model, shown below.

Of note is that while training loss continued to decrease with later and later epochs, to the point of overfitting the data, the "top1 accuracy" (which is just simply validation accuracy for our binary classification over the test data) peaked at epoch 22 or so and then continued to decrease over time. Validation loss got worse and worse for epochs 25 through 40; meaning that we clearly reached the best performance we could get (we don't need to run more epochs here) and we still didn't get anywhere near our accuracy with our other CNNs.

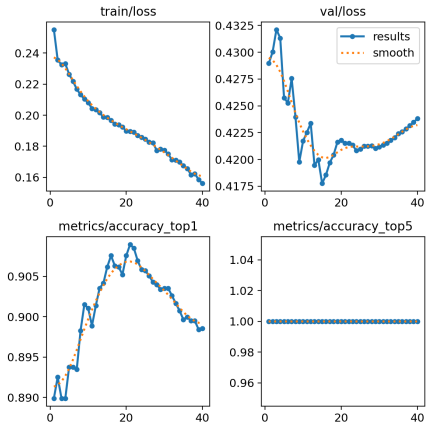


Figure 10: When transfer learning using the concatenated images that included the entire radar imagery set, the pre-trained YOLOv5 model didn't perform very well. While training loss (top left) continued to decrease by epoch, validation loss (top right) started worsening in the back half of training. Validation accuracy (bottom left) peaked at around .908.

Without the bounding boxes to tell an object detection algorithm exactly where to look (as in YOLOv5 object detection), the pre-trained classification CNN was mostly looking at these radar imageset concatenations and just trying to get useful information from small pixels within each subplot of the image - and probably wasn't doing so well. The information loss caused by concatenation was probably worsened when the images had to be shrunk even further to fit in memory in mini-batches.

Yet single images (which we theorized might do better) also seemed to not do nearly as well as one might hope either, although they showed slightly more promise. Based on the results of the earlier work with the TorNet CNN (which seemed to be keying in on patterns in radial velocity), we also created a dataset where we only exported single images of radial velocity, to see if that could work.

Those efforts did result in slightly higher accuracies (which gives us some hope for this avenue of work in the future) but still not did not perform as well as we'd hoped. In our initial efforts, we saw a peak validation accuracy of .911 given a single image (which is still better than the concatenated version but still quite poor performance overall.

Other attempts that did even more poorly are not included here, to keep this appendix concise.

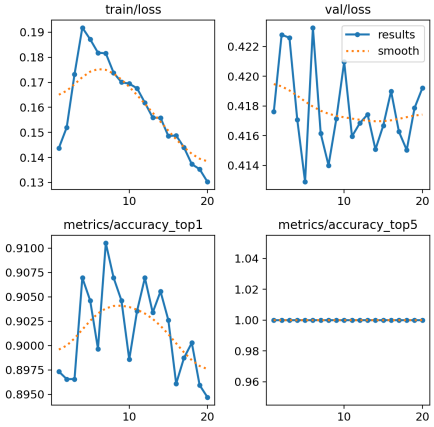


Figure 11: When transfer learning using solely images of radial velocity, the pre-trained YOLOv5 model still didn't perform as well as hoped (although better than the concatenated images). While training loss (top left) continued to decrease by epoch, validation loss (top right) started worsening in the back half of training. Validation accuracy (bottom left) peaked at around .911.