

CMPT 318 Project: Webcams, Predictions, and Weather

Johnson Pan (johnsonp), Jun Da Li (jundal), Jordan Siaw (jsiaw)

The problem we are addressing

The collection of weather data involves manual processes for the most part: measurements have to be read from instruments and entered manually into the data store. With the recent success of convolutional neural networks (CNNs) in image classification, perhaps the collection process could be automated by feeding a trained model images of a location and have it produce weather labels. This project is an attempt to prove that the process can be automated.

Data collection

The image data used for this project is from [Kat Kam](#), an organization that has been taking photos of English Bay since 1996. According to the professor, we have permission to use their archived images and the data set itself was provided by the professor.

The original images were 1280 x 960, which were too large to be practical. The professor [provided a version of the data set](#) where each image was scaled to 256 x 192.

The weather data used comes [from the Government of Canada](#). This data set contains hourly data from the Vancouver Airport (YVR) weather station for the period of June 2016 - June 2017, since it has the most complete data set.

Data cleaning

Matching images with weather data

We had 9237 rows of weather data and 5046 images to start with. Both data sets had gaps in their timelines in that there wasn't data available for every hour of every day. The difference in count between the data sets results from the fact that the weather data was recorded more frequently than the image data.

We first mapped each image to a row from the weather data. In turned out that there was a one-to-one mapping from images to weather data, so we ended up with 5046 of 'joined' data points.

Looking at that data set, we found that some rows were missing Weather labels, so we decided to exclude them since we had no reasonable way of filling in those gaps. We thought about using rows adjacent to a 'missing Weather label' row to approximate the missing label. For example, if it was 'Cloudy' from 3pm - 6pm and the row at 4pm was missing, it would be reasonable to fill it in with 'Cloudy'. However, some of those 'missing Weather label' rows were sometimes recorded at awkward hours that didn't have reasonable adjacent hours we could estimate from. We decided that it was not worth the time and effort to verify any of those estimations.

Another idea that we came up with was to train a model with our cleaned data and have it predict weather labels for those 'missing weather label' rows, but in the interest of time, we decided not to pursue it.

After excluding missing labels, we ended up with 2248 data points. Although this was a smaller data set than we would have liked, we were interested in seeing how the model would perform with a training set of this size (80% of 2248 \approx 1798).

Cleaning and mapping Weather labels

With this data set, we noticed that some Weather labels were very close in semantic meaning to each other. For example, 'Mainly Clear' and 'Clear' mean the same thing, so do 'Rain Showers' and 'Rain' and 'Moderate Rain Showers'.

To simplify the classification, we decided to replace all weather labels with the shortest and most general version of that weather's label. For example, all rows containing 'Rain Showers' and 'Moderate Rain Showers' labels were replaced with a single label 'Rain'.

On top of this, we noticed that certain rows would contain the 'Cloudy' label alongside a 'Rain' or 'Snow' label, but other rows would contain just a 'Rain' or 'Snow' label without a 'Cloudy' label. We decided to standardize this by adding a 'Cloudy' label to any row that has just a 'Rain' or 'Snow' label without a 'Cloudy' label. For example, a row containing only 'Rain' was replaced with 'Rain, Cloudy'.

Original Weather Label	Cleaned and Mapped Weather Label
Heavy Rain	Rain,Cloudy
Moderate Rain,Fog	Rain,Cloudy,Fog
Rain,Drizzle,Fog	Rain,Cloudy,Fog
Heavy Rain,Fog	Rain,Cloudy,Fog
Snow Showers	Rain,Cloudy,Snow
Snow	Snow,Cloudy
Snow,Fog	Snow,Cloudy,Fog

Rain,Snow	Rain,Cloudy,Snow
Freezing Fog	Fog

Figure 2 - 1, Weather Label Mapping Samples

	Date/Time	Year	Month	Day	Time_x	Filename	Mapped	Weather
1	2016-06-05 07:00:00	2016	6	5	07:00	katkam-20160605070000.jpg	Clear	Mainly Clear
3	2016-06-05 10:00:00	2016	6	5	10:00	katkam-20160605100000.jpg	Cloudy	Mostly Cloudy
6	2016-06-05 13:00:00	2016	6	5	13:00	katkam-20160605130000.jpg	Cloudy	Mostly Cloudy
9	2016-06-05 16:00:00	2016	6	5	16:00	katkam-20160605160000.jpg	Cloudy	Mostly Cloudy
12	2016-06-05 19:00:00	2016	6	5	19:00	katkam-20160605190000.jpg	Cloudy	Mostly Cloudy
16	2016-06-06 07:00:00	2016	6	6	07:00	katkam-20160606070000.jpg	Cloudy	Mostly Cloudy
19	2016-06-06 10:00:00	2016	6	6	10:00	katkam-20160606100000.jpg	Cloudy	Mostly Cloudy
22	2016-06-06 13:00:00	2016	6	6	13:00	katkam-20160606130000.jpg	Cloudy	Mostly Cloudy
25	2016-06-06 16:00:00	2016	6	6	16:00	katkam-20160606160000.jpg	Clear	Mainly Clear
28	2016-06-06 19:00:00	2016	6	6	19:00	katkam-20160606190000.jpg	Cloudy	Mostly Cloudy

Figure 2 - 2, Cleaned Data Samples

Techniques we used to analyse the data

For this project, we considered using Support Vector Machines (SVM) and Convolutional Neural Networks (CNN).

We chose CNNs after seeing that they have been the most dominant technique used in recent ImageNet competitions. Since those CNNs tackled a 1000-class image classification problem, we decided it would be a good place to start.

Adapting VGG16

Our initial thought was to use the VGG16 model used by the VGG team in the ILSVRC-2014 competition. We could take the advantage of the pre-trained weights available, which would us save a lot of training time.

We started by replacing the final fully connected layers in VGG16 with our own prediction layers to suit our classification problem. We then froze the weights on the convolutional blocks and tried training only the prediction layers. We were curious to see how the pre-trained weights in the convolutional layers would perform on our training data. The initial results were not good.

Tweaking both the convolutional and prediction layers didn't seem to help either. We think this could be because the pre-trained weights were trained using images that were very

categorically different from our data set. Therefore, the features extracted by those trained convolutional layers were not suitable for predicting what we wanted: weather features.

We then decided to train the VGG16 network without the pre-trained weights. In other words, we would train the convolutional layer to tweak its weights according to our training data. However, we immediately faced resource limitations. Loading all the initial weights in the convolutional layers required about a large amount of VRAM. We only had 6GB on our beefiest piece of hardware (an Nvidia GTX 1060), which apparently was not enough because we weren't even able to start a training run.

Our simplified version of VGG16

To work around this, we reduced the number of convolutional blocks in the network from 5 to 3. This allowed us to successfully begin training.

We later discovered that having convolutional blocks that were too deep makes it difficult for the network to tune its parameters. According to a Quora question, this is called the [vanishing gradient problem](#). In other words, having a shallow network makes the model easier to train. Furthermore, given that we only have ~2200 training images for 5 labels, using an overly complex CNN might result in overfitting.

On top of those 3 convolutional blocks, we have 2 branches of prediction layers. We call the first branch our 'primary branch' and the second branch our 'secondary branch'.

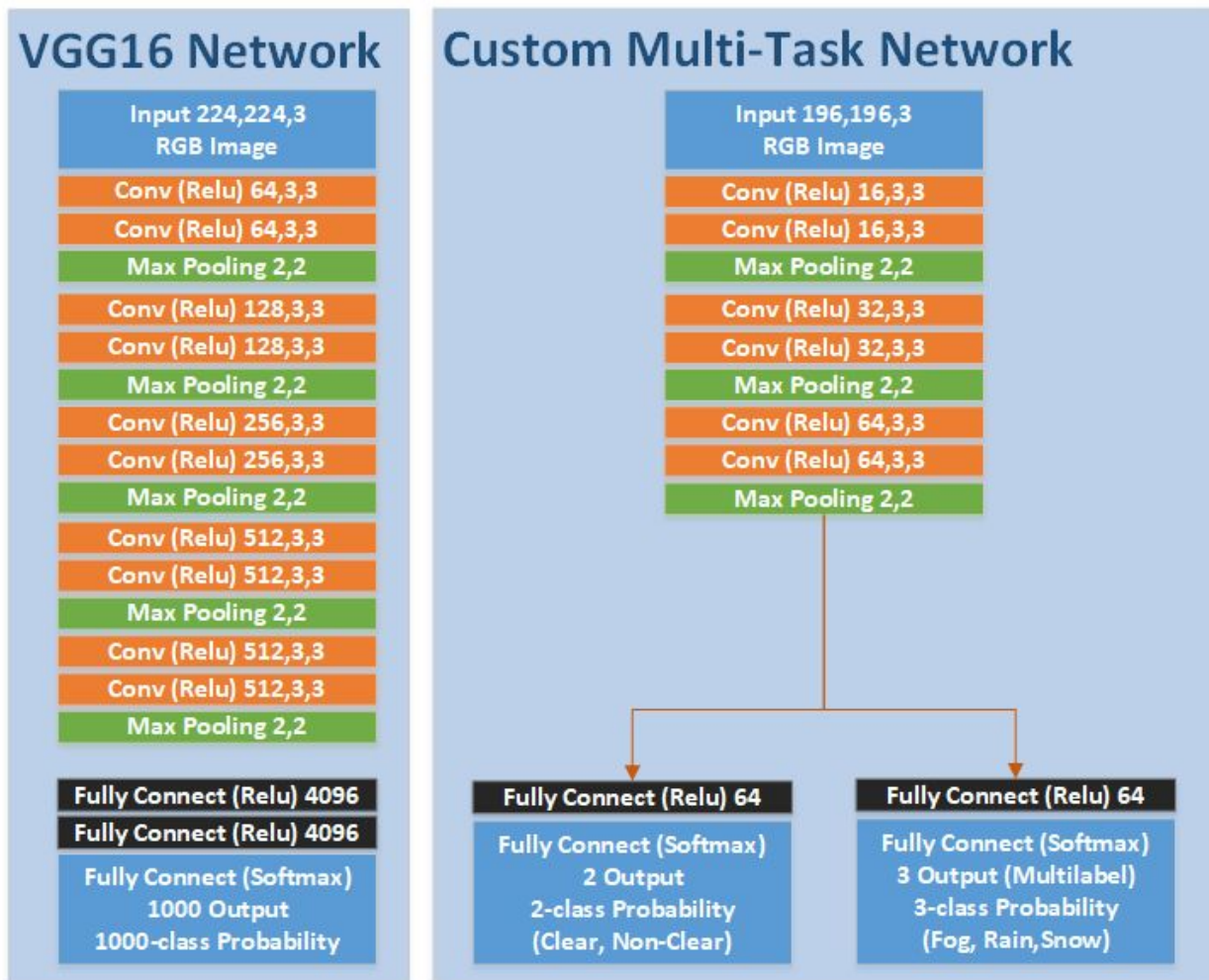


Figure 3 - 1, Simplified VGG16 Network with Branches.

Multi-label classification

Given an image, the model will produce 2 outputs: a list of probabilities from the primary branch and a list of probabilities from the secondary branch.

The probabilities from the primary branch represent the probability of predicting the given labels respectively: ['Clear', 'Non_Clear']. The probabilities from the secondary branch represent the probability of predicting the given labels respectively: ['Fog', 'Rain', 'Snow'].

We wrote additional logic to map these probabilities to their string representations. Any label that has a probability of >50% will have its string included in the final prediction.

There is a special case regarding probabilities on the secondary branch. If any of the probabilities from the secondary branch are below 50%, the resulting prediction will be 'Cloudy'. The rationale for this is that since we're confident that the primary branch has already predicted

a 'Non_Clear' label and the secondary branch doesn't think it's 'Fog', 'Rain' or 'Snow', the only weather label left has to be 'Cloudy'.

Image data pre-processing

The resolution of the images was 256 x 192. We performed the following steps before those images were fed into the network.

1. The image was resized to 196 x 196 to fit the network input requirement.
2. Images were randomly shifted/rotated so that the network would rarely see the same input twice.
3. RGB values in the image were normalized before they were fed into the network.

Two-phase training

The image data was split into 80% training data and 20% testing data. We trained the model using a two-phase approach.



Figure 3-2. Two-Phase Training

Phase 1 is focused on training the primary branch. We detached the secondary branch and fed the primary branch with only images that were labelled either 'Clear' or 'Non_Clear'.

After Phase 1 began performing reasonably (validation accuracy $\approx 90\%$), we started Phase 2. In this phase, we locked all the weights in the main branch (only the convolutional blocks) as well as the primary branch. Then, we fed the secondary branch training images that were labelled a combination of 'Fog', 'Rain', 'Snow' or 'Cloudy'.

Because we fed the same kind of images to the network, re-using the locked weights from Phase 1 really sped up the time it took for the secondary branch to converge to a reasonable accuracy (validation accuracy $\approx 70\%$).

Findings and conclusions

Let's take a look how our model performed for 10 random images. The red rows indicate wrongly-predicted weather labels.

Filename	Weather	Predicted Weather	Raw Network Output
katkam-20160809130000.jpg	Cloudy	Cloudy,Fog,Rain	[[0.0, 1.0], [0.708, 0.942, 0.0]]
katkam-20160807100000.jpg	Cloudy	Cloudy	[[0.011, 0.989], [0.0, 0.003, 0.0]]
katkam-20161209120000.jpg	Cloudy,Snow	Cloudy,Snow	[[0.0, 1.0], [0.372, 0.084, 0.938]]
katkam-20161105130000.jpg	Cloudy,Rain	Cloudy,Rain	[[0.0, 1.0], [0.047, 0.576, 0.001]]
katkam-20161024140000.jpg	Cloudy,Rain	Cloudy,Rain	[[0.0, 1.0], [0.0, 0.509, 0.0]]
katkam-20170612160000.jpg	Cloudy	Cloudy	[[0.203, 0.796], [0.0, 0.004, 0.0]]
katkam-20170515120000.jpg	Cloudy,Rain	Cloudy,Rain	[[0.001, 0.999], [0.0, 0.544, 0.0]]
katkam-20161101160000.jpg	Cloudy,Rain	Cloudy,Rain	[[0.0, 1.0], [0.162, 0.903, 0.008]]
katkam-20160726070000.jpg	Cloudy	Clear	[[0.931, 0.068], [0.0, 0.0, 0.0]]
katkam-20161219100000.jpg	Clear	Clear	[[0.972, 0.028], [0.0, 0.0, 0.0]]

Figure 4-1, 10 Test Samples.

Take a closer look at the images of the 2 wrongly-predicted weather labels.



Filename	Weather	Predict Weather	Raw Network Output
			
katkam-20160809130000.jpg	Cloudy	Cloudy,Fog,Rain	[[0.0, 1.0], [0.708, 0.942, 0.0]]
			
katkam-20160726070000.jpg	Cloudy	Clear	[[0.931, 0.068], [0.0, 0.0, 0.0]]

Figure 4 - 2, Wrongly Predicted Images

The first image here has its prediction marked as false because it doesn't match the ground truth label exactly. However, just by observing the actual image, we can see that the model's prediction of 'Cloudy', 'Fog' and 'Rain' is quite reasonable. The blurry background would suggest 'Cloudy' or 'Fog' to a human observer. So, the model makes quite a 'human' mistake.

The second image is interesting, the model predicted 'Clear' while the ground truth is 'Cloudy'. However, the image would look 'Clear' to a human observer.

This exposes the shortcomings of our original data set. The model obviously makes the correct prediction here, but the ground truth label from YVR was probably based on the actual weather at YVR at the time, not English Bay.

Overall, we think that our model performs with a reasonable accuracy, given the fact that the training weather data and image data come from different locations.

Limitations and challenges

Problematic data set

Since Vancouver Airport (YVR) and English Bay are far enough apart that they could be experiencing different weather, we could have the case where the model correctly predicts the weather in the test image, but the prediction doesn't match the associated truth label because YVR was actually experiencing different weather at that point in time (as illustrated in the test sample above).

This is a problem with the data that causes false negatives. So, the true accuracy of this model's predictions might be higher than our validation runs show.

A biased model

Our model will be biased towards pictures of English Bay from the specific vantage point in our training data set. It might not perform well with validation data from a different location in English Bay.

It would be interesting to see if the network's prediction accuracy will be affected by a slight shift in the point of view of the training images. A follow up to that would be to see how much of a difference in the point of view it takes completely destroy the network's prediction accuracy.

Things that can be improved

Get more data

Originally, we had more than 5000 images. After filtering, we were left with around 2200. If we could fill in the missing labels by reasonably interpolating the neighbouring data, we would have a much larger data set to work with.

Use other features to aid prediction

There were other features like temperature, visibility and humidity within the weather data that could probably have made predictions more accurate.

Fine-tune the network

Given the timeframe, we could not investigate other network structures.

Project Experience Summary

Johnson Pan

It's interesting to find out for such a small dataset, how to use CNN to solve a multi-label classification project with reasonable performance. During training, I am surprised by how fast the secondary branch converges when we reuse the primary weights. Also surprised to see how "adam" optimized perform better than the traditional SGD (stochastic gradient descent) optimizer.

Jun Da Li

Surprised by the powerful convolutional neural networks.

Jordan Siaw

Learned about the inner workings of convolutional neural networks and the importance of having good data sources.

Code Instructions

We used Python 3 in our project with the following packages: keras, tensorflow, numpy, pandas, matplotlib.

To run our code, use the following commands.

```
$ pip install keras tensorflow numpy pandas matplotlib
$ git clone https://github.com/onethirdzero/ml-weather.git
$ cd ml-weather
$ python3 project.py /PATH/TO/IMAGES
```

Note that /PATH/TO/IMAGES is the directory that holds the Kat Kam images.
For example, /PATH/TO/IMAGES/katkam-20160726070000.jpg.