Disaster Detection System using Deep Neural Networks

By: Venkatraghavan Kannan

Purpose/Motivation

- Almost all disasters are currently manually reported
    - People caught in disaster maybe unable to report
    - Much slower response time
    - Communication systems might be disrupted

- An Automated Disaster Detection System eliminates these issues
    - Aerial images fed into a neural network model can detect disasters
    - Possible camera locations
        - Stop Lights
        - Electric Towers
        - Surveillance Drones
        - Building tops
        - Etc.

Overview

This project uses Convolutional Neural Networks. Convolutional Neural Network (CNN) models allow us to effectively capture relevant patterns and information present in images at a relatively low computational and memory cost. I used pre-trained CNN models (trained on ImageNet dataset) and transfer learning to overcome high computational costs and need for longer training with large datasets.

This project is trained to detect 5 different scenarios:

- Fire/Smoke
- Floods
- Collapsed Building/Rubble
- Traffic Accidents
- Normal scenario without a disaster

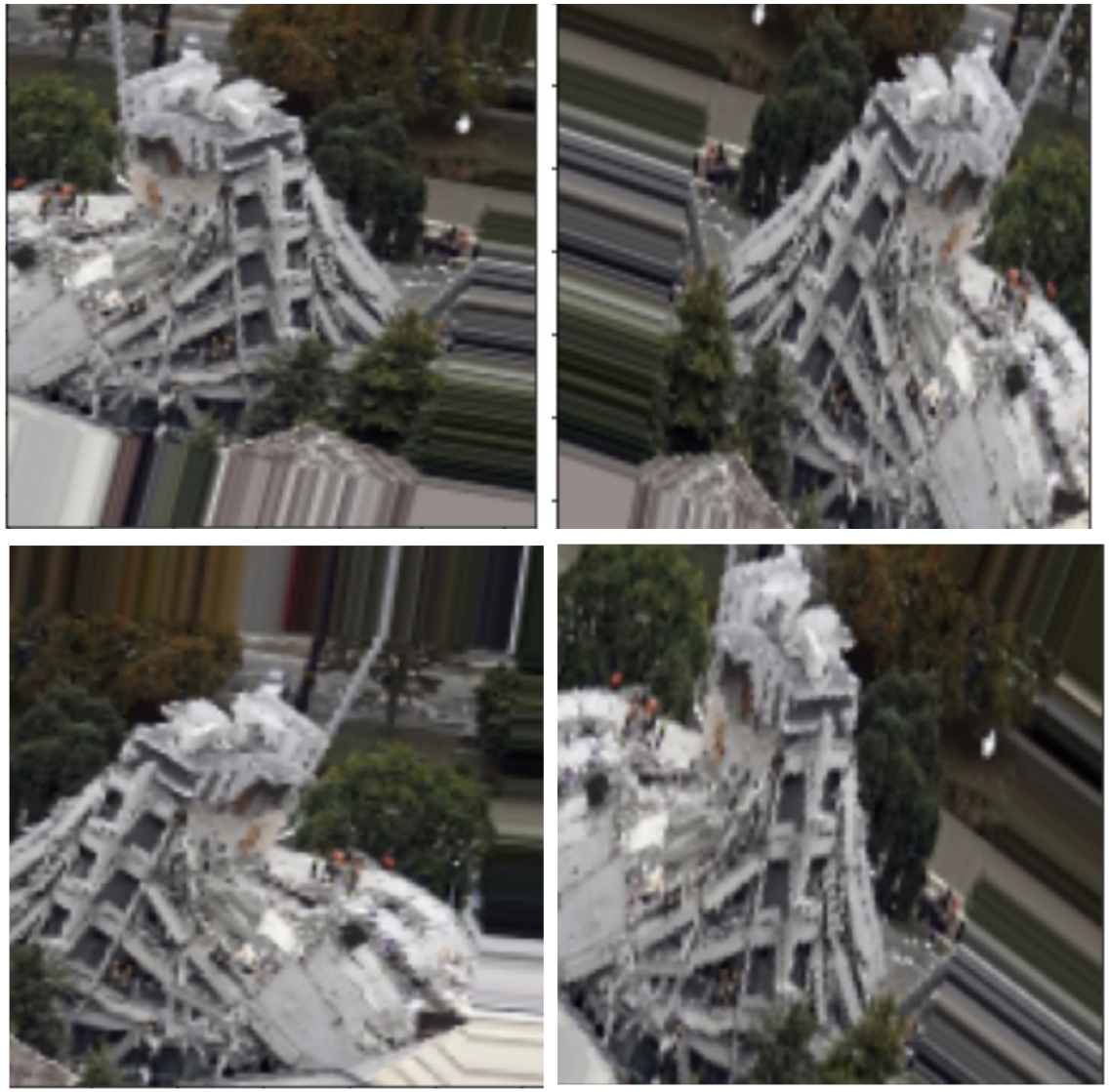
Dataset

- I used a trimmed version of the “AIDER” dataset downloaded from https://zenodo.org/record/3888300
- The original dataset was manually collected and verified from various online sources such as Google/Bing images, YouTube, news sites, and more.
- The images are originally of different sizes, resolution, illumination, and viewpoint to make it as general and replicable of real-world situations.
- The dataset contains approximately 500 images for each disaster and 4000 images for the normal scenario.
- The dataset is split into 400 training, 80 validation and 20 test images per class

Actual Dataset Images from each of the 4 disaster classes

Preprocessing

- Resized all images to 150x150 pixels.
- Used Data Augmentation to overcome the limitations of small training set and overfitting issues.
- The same image is not used twice during training.
- Augmentation Techniques used during training :
  - Random rotation
  - Random vertical and horizontal translation
  - Random shear transformations
  - Random zooming



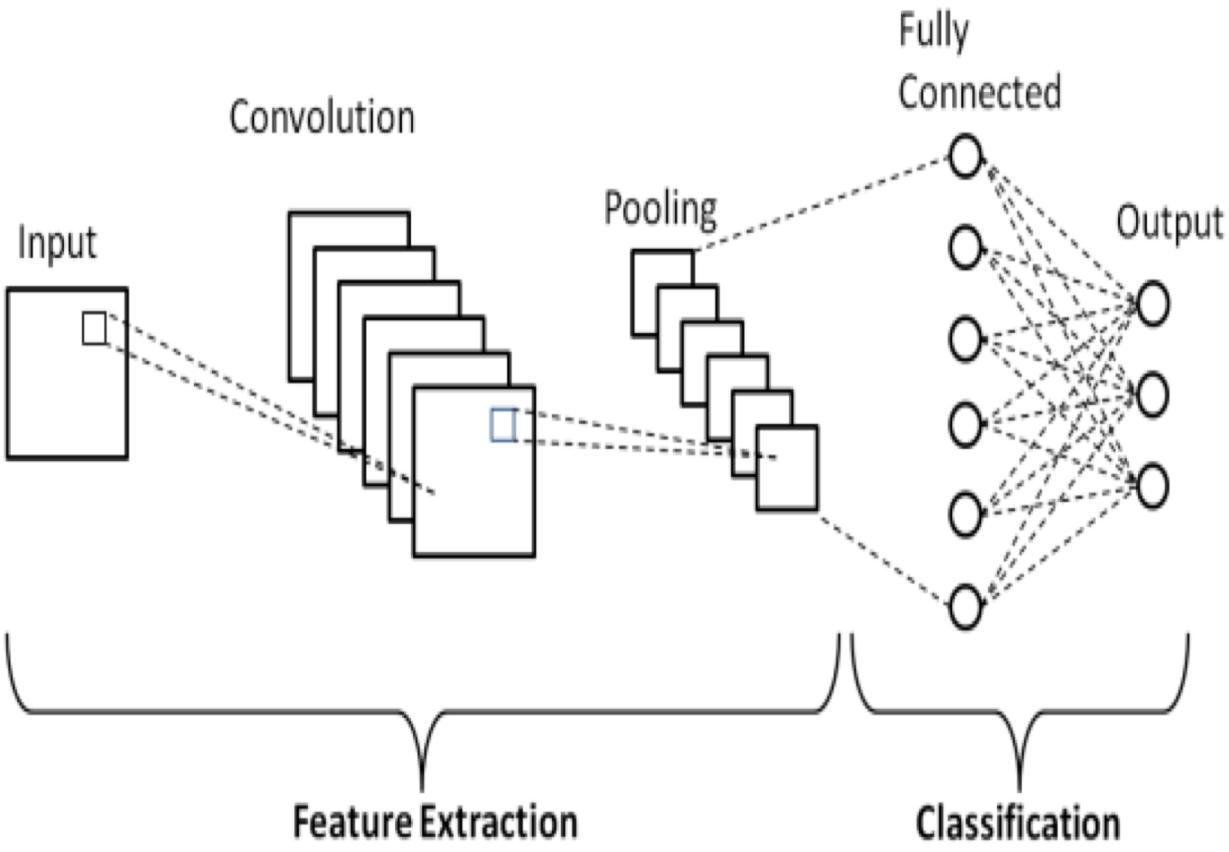
Data augmented images of a single source image.

CNN Background

CNN are a subclass of Neural Network and they are great for capturing local information like neighbor pixels in an image or surrounding words in a text. They are good in reducing the complexity of the model (faster training, needs fewer samples, reduces the chance of overfitting).

CNNs perform a series of **convolutions** and **pooling** operations on the input images. The convolutions are good in finding special patterns in images. Patterns could be as simple as vertical edges and curves all the way to facial features.

CNN work by considering n-pixels are a time. Given a sequence of pixels, a 1D convolution of width-k is the result of moving a sliding-window of size k over the image and applying the same **convolution filter** or **kernel** to each window in the image.

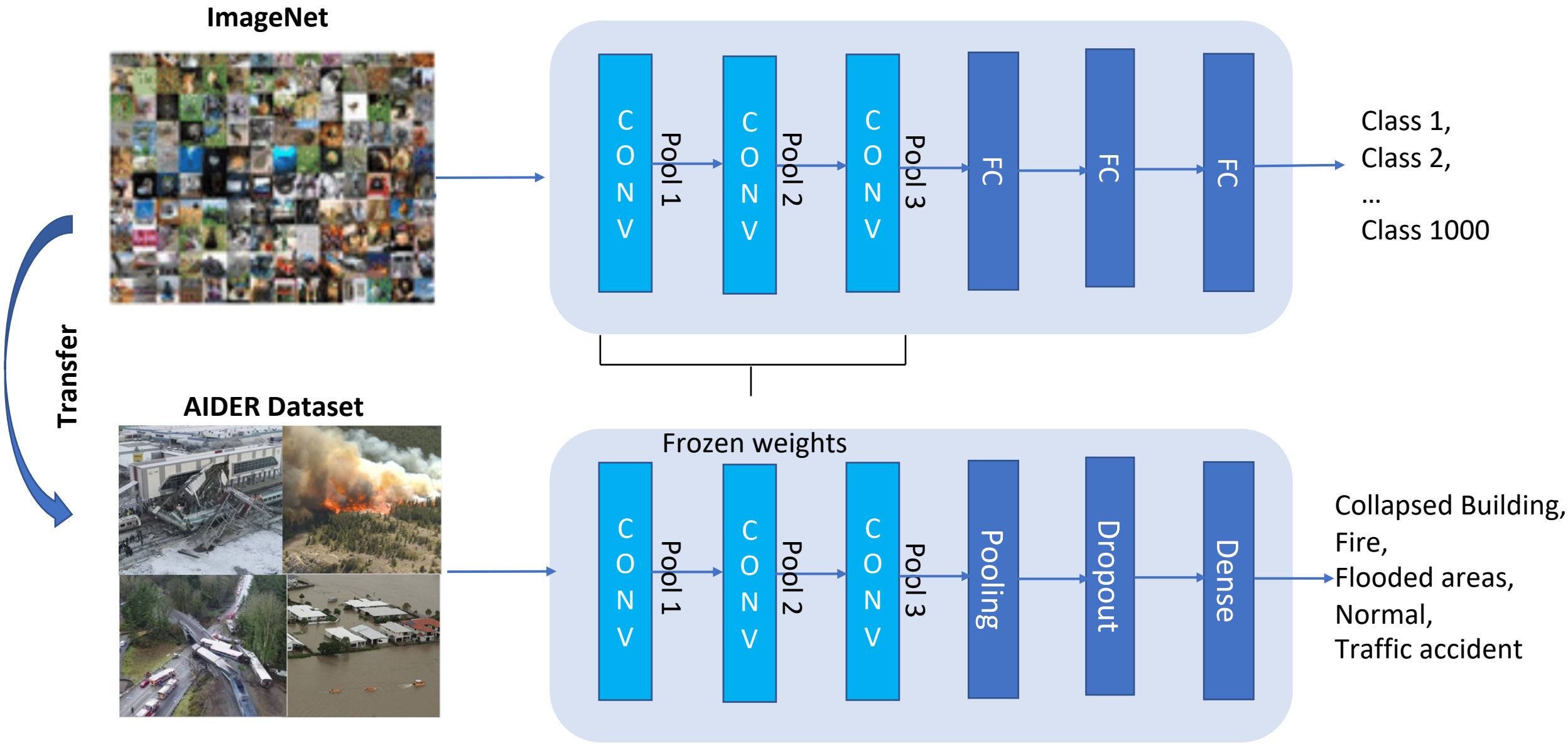


Method

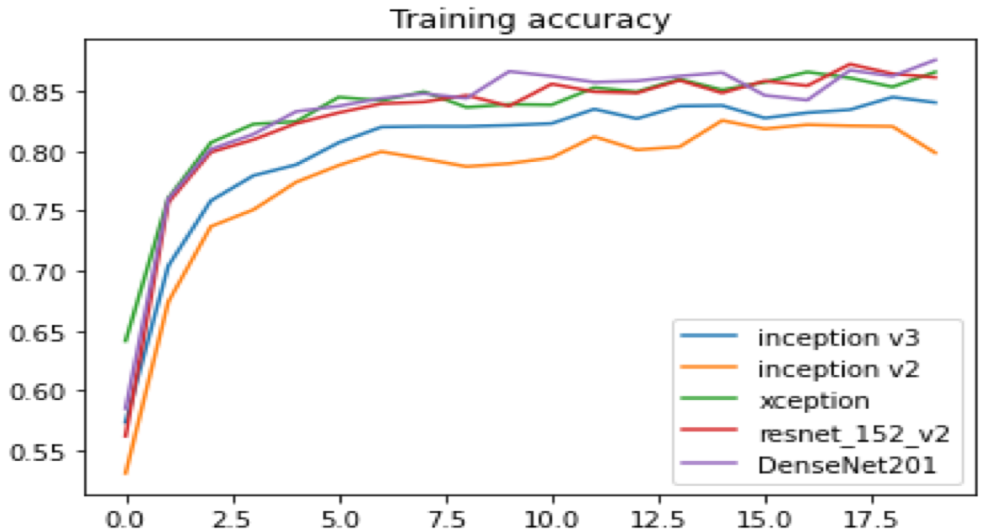
- Trained different models using Transfer Learning from existing pre-trained Keras CNN models (using ImageNet dataset):
    - Inception V2
    - Inception V3
    - Xception
    - Resnet 152 V2
    - DenseNet201

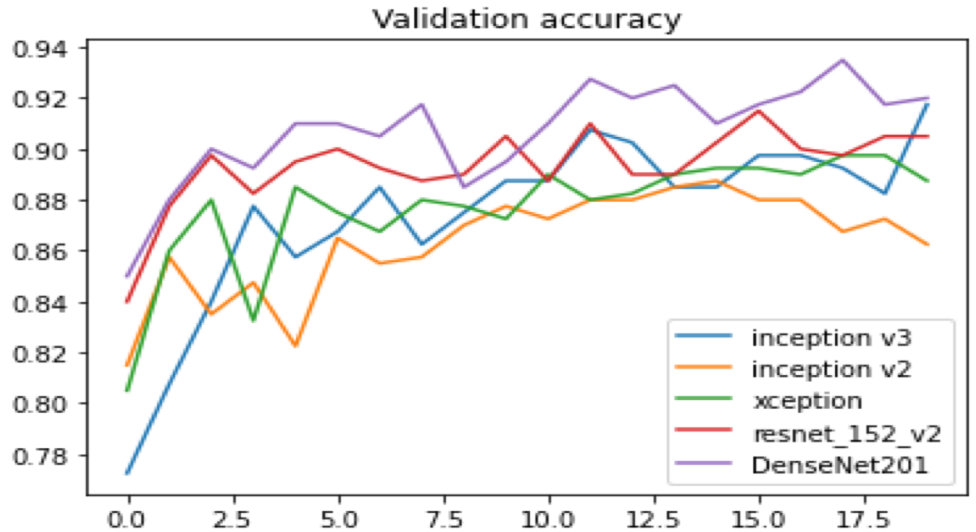
- Removed the default classifiers on the various pre-trained models and attached our own Softmax classifier
- Froze all the layers in pre-trained model
- Trained each model for 20 epochs
- Used Mini Batch Gradient Descent with RMSprop optimizer with learning rate (lr = 0.001)

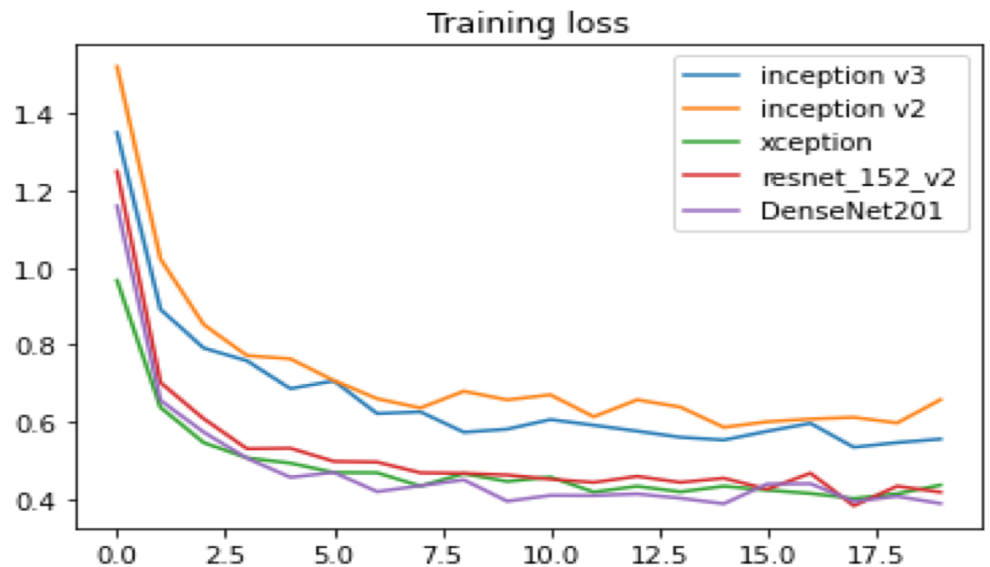
Method(cont.): Model Architecture

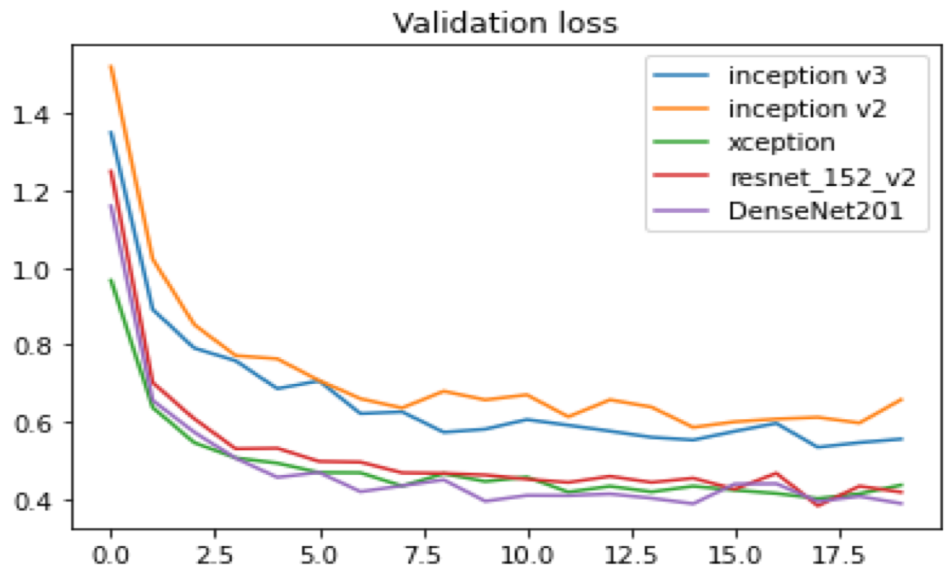


Results


Training accuracy
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
inception v3
inception v2
xception
resnet_152_v2
DenseNet201

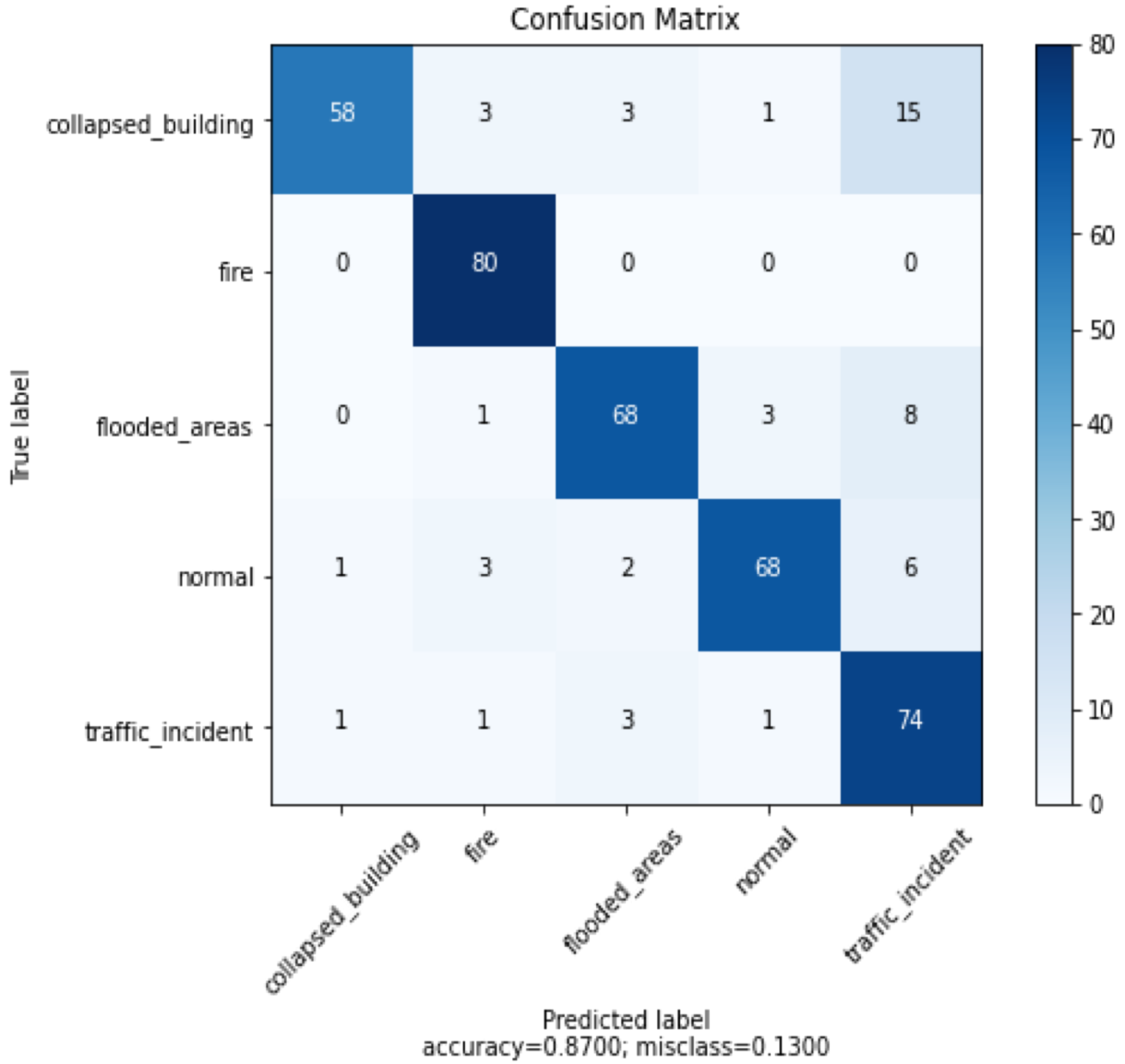

Validation accuracy
0.94
0.92
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
inception v3
inception v2
xception
resnet_152_v2
DenseNet201


Training loss
1.4
1.2
1.0
0.8
0.6
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
inception v3
inception v2
xception
resnet_152_v2
DenseNet201

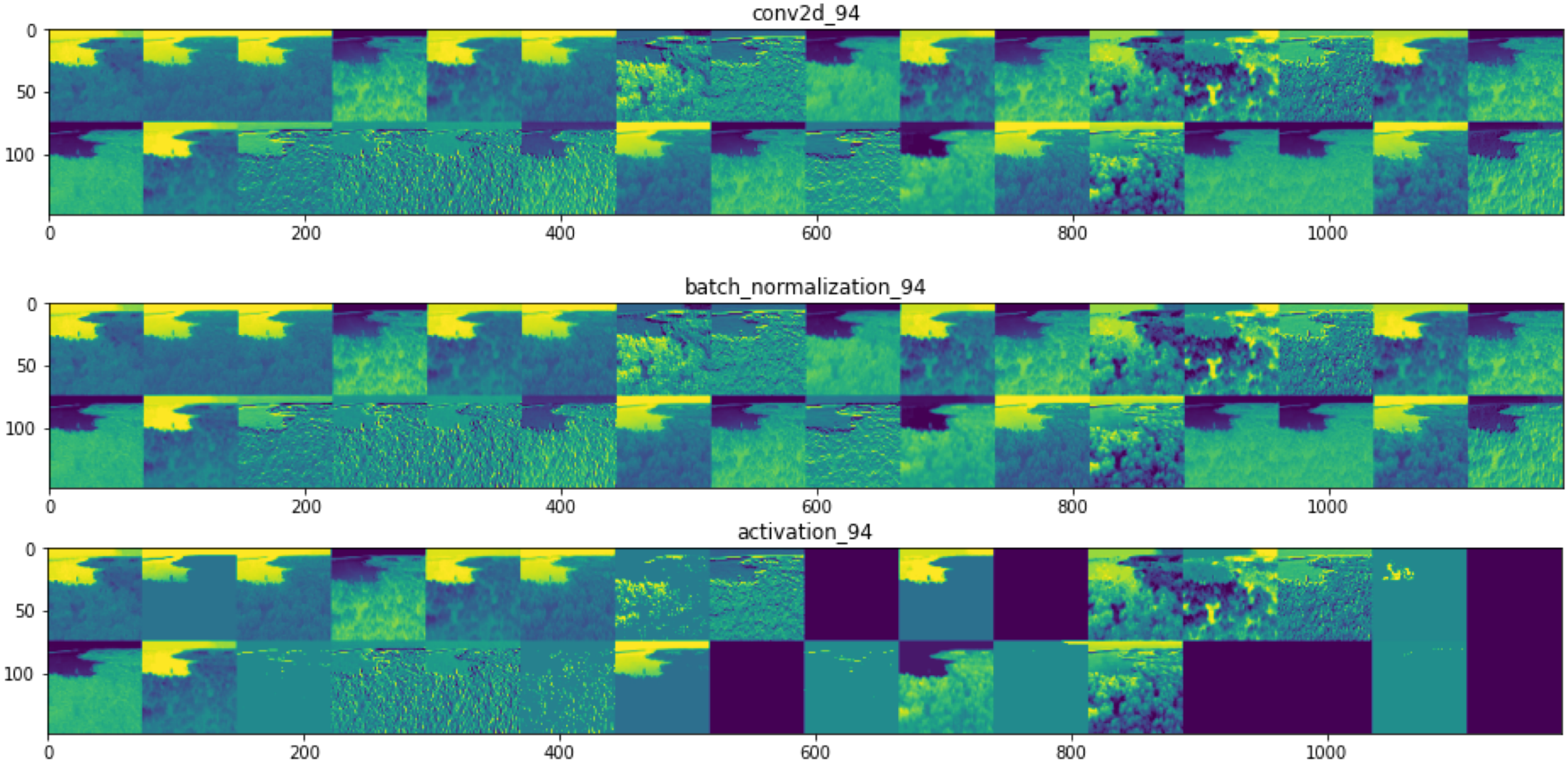

Validation loss
1.4
1.2
1.0
0.8
0.6
0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
inception v3
inception v2
xception
resnet_152_v2
DenseNet201

Results(Cont.)

	precision	recall	f1-score	support
collapsed_building	0.97	0.72	0.83	80
fire	0.91	1.00	0.95	80
flooded_areas	0.89	0.85	0.87	80
normal	0.93	0.85	0.89	80
traffic_incident	0.72	0.93	0.81	80
accuracy			0.87	400
macro avg	0.88	0.87	0.87	400
weighted avg	0.88	0.87	0.87	400



Visualizing activations of a single Convolution, Batch Normalization and Activation Layer.



Conclusions

- All Models had similar training and validation accuracy, 86 - 92%.
- Overall, DenseNet201 seemed to work best with 92% validation accuracy.
- Similar training time for all model on GPUs.
- ResNet 152 V2 needed the most memory: 232 MB.
- DenseNet201 needed the least memory: 80 MB.

Future

- Use image manipulations like illumination changes, color shifting, blurring, sharpening, and shadowing.
- Use more images for the Normal scenario to reflect real-world scenarios and solve the unbalanced case.
- Train and test with more diverse images.
- Build and deploy in real world.

References

- https://zenodo.org/record/3888300#.X3fOcZNKi3I
- https://keras.io/api/applications/
- http://www.image-net.org/
- https://colab.research.google.com/
- https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909
- https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-keras-260b36d60d0