CS767 Final Project

Fire detection

Student: Frank (Chen) Wu
Professor: Dr. Farshid Alizadeh-Shabdiz
Boston University

Data description

1.Distribution of Data:

Original: 1212 no fire images, 300 fire images (Highly imbalanced)
Resolution: Image size ranges from 1980 * 1080 * 3 to 240 * 160 * 3

2. Source:

Training Data: From a fire equipment company, camera captured images from workplace, inventory, street, etc.

The company collected images from where they provide fire equipment service and conducted fire tests to get fire images.

Testing Data: A brand new dataset, separated from training set, which are misclassified images by the current systems of the same company.

3. Solution to imbalanced dataset:

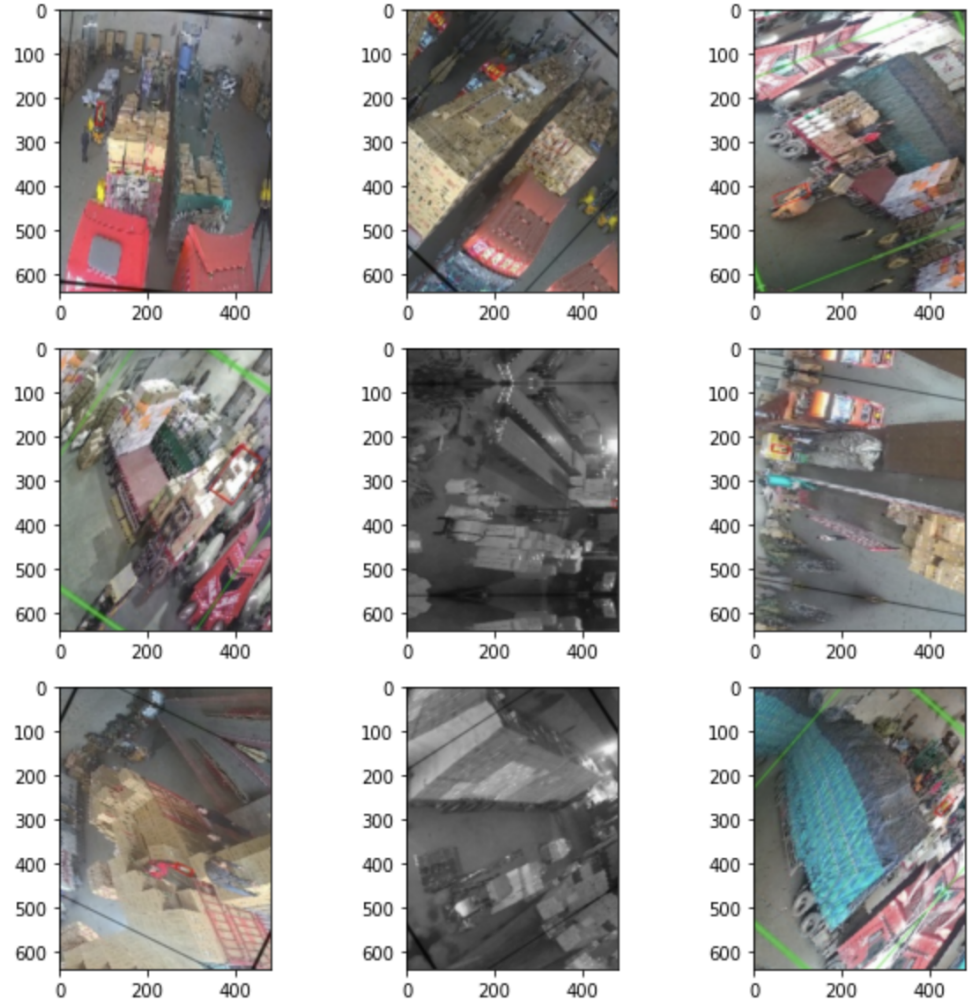
Image Augmentation — random flipping and contrast and rotation to create more fire images.
After augmentation: 1212 non-fire images, 1232 fire images

4. Data samples

Training samples

Label 0 is a non-fire image, label 1 is a fire image

New data samples:



No labels, but all are non-fire images, which were misclassified as fire images by the company's model.

Goal

The overall goal is to get a low error rate.

The issue that was addressed by the company was that their current image classification model had a fairly low accuracy around 70%. They used traditional machine models, the model performed terribly on objects that looked like fire, such as red objects or natural light. It classified all those as fires. Thus, this project focused on decreasing the error rate due to the detection of fire-like objects and to increase the accuracy rate to above 90%.

The classifier will be a Convolutional neural network (CNN) model to classify images. Traditional fire alarm systems use sensors to detect smoke or heat, that is more efficient than video cognition, but for some open spaces with streaming data from monitors, an image-based fire monitoring system could be helpful. For example, particle smoke sensors may not be sensitive in large settings, because of air circulation and other distractions.

Approach

1. CNN: Trained a standard CNN as a baseline model, which has a misclassification rate of 13 percent on the test set.
2. Transferred learning by auto-encoders: Used layers from Convolutional auto encoders on a CNN model to improve performance.

3. Used a Convolutional auto-encoder to generate fire images, and a CNN classifier as a discriminator against the generator. So the performance of the discriminator would be improved by competing with the generator.
4. Used a Variational Auto-encoder (VAN) to make a generative adversarial network (GANs). Used a VAN to generate fire images and fed them to a CNN classifier.

Testing

In addition to splitting the data of the training set, validation set, and testing set, a brand-new dataset was used for the purpose of testing misclassifications, which were mentioned in the data description.

The testing data included all non-fire images that were misclassified as fire by the company's system. The correct predictions for the new data should be all 0 (non-fire). I computed the misclassification ratio by how many images were misclassified divided by total number of images

I used the validation dataset with the training to check overfitting, and the testing dataset for evaluation purposes. Another brand new dataset to test error rate.

Ensemble Method

The final prediction will be the majority labeled from all models below:

Baseline Model
CNN with auto-encoder layers
Discriminator with convolutional generator
Discriminator with VAE generator

The ensemble methods provide a more reliable prediction compared to a single model. Prediction is not computationally expensive compared to training, so we can use multiple trained models to do majority voting to decide the final output label.

Models

Model 1-- Standard CNN:

3 convolution layers, 2 max pooling, batch normalization after each trainable layer.

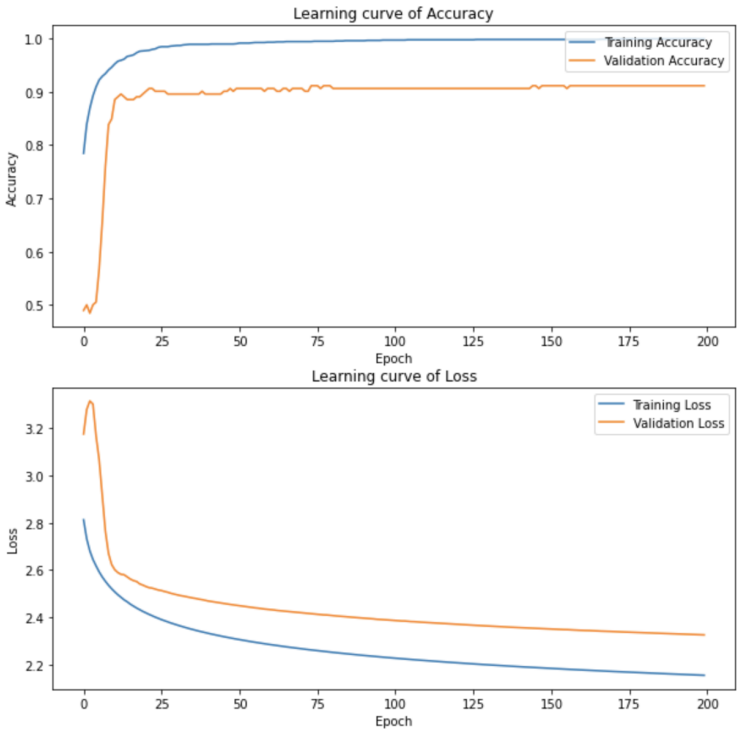
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 128, 96, 64)       4864
_________________________________________________________________
batch_normalization_10 (Batc (None, 128, 96, 64)       256
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 42, 32, 64)        36928
_________________________________________________________________
batch_normalization_11 (Batc (None, 42, 32, 64)        256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 20, 15, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 9, 7, 32)          18464
_________________________________________________________________
batch_normalization_12 (Batc (None, 9, 7, 32)          128
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 3, 32)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 384)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               49280
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dense_7 (Dense)              (None, 32)                4128
_________________________________________________________________
batch_normalization_14 (Batc (None, 32)                128
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 33
=================================================================
Total params: 114,977
Trainable params: 114,337
Non-trainable params: 640
```

Accuracy test set: 89%

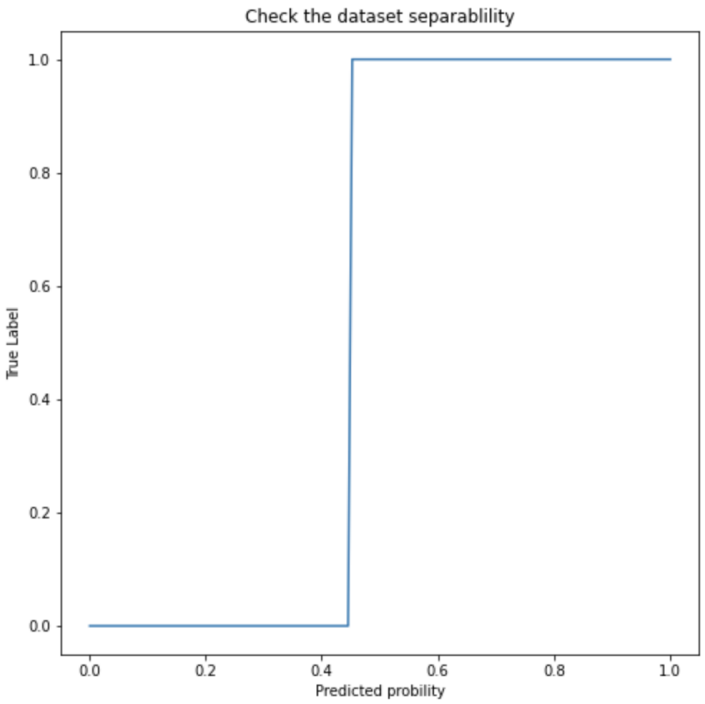
Accuracy New Data: 87%

Error rate: 13%

The learning curve:



Accuracy
Test set: 89%
New Data: 87%
Decision boundary:



We can see a much higher accuracy on the CNN model than on the company's current model (traditional machine learning). But the model does not generalize well on the new dataset, the error rate is still higher than 13%, which will cause problems on the production stage. Therefore, I needed to tune the model or use a different architecture to decrease the error rate.

Model 2: CNN with trained layers from an autoencoder

One of the applications of an autoencoder is to use the lower level layers of trained autoencoders of another model. The lower level layers of autoencoders learn features from the input data, which can be used for transfer learning purposes.

Step 1: Train a CNN autoencoder

Encoder

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 320, 240, 12)      912
_________________________________________________________________
batch_normalization_89 (Batc (None, 320, 240, 12)      48
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 160, 120, 24)      7224
_________________________________________________________________
batch_normalization_90 (Batc (None, 160, 120, 24)      96
_________________________________________________________________
48filter_layer (Conv2D)      (None, 80, 60, 48)        10416
_________________________________________________________________
batch_normalization_91 (Batc (None, 80, 60, 48)        192
_________________________________________________________________
72filter_layer (Conv2D)      (None, 40, 30, 72)        31176
_________________________________________________________________
batch_normalization_92 (Batc (None, 40, 30, 72)        288
_________________________________________________________________
96filter_layer (Conv2D)      (None, 20, 15, 96)        62304
_________________________________________________________________
batch_normalization_93 (Batc (None, 20, 15, 96)        384
=================================================================
Total params: 113,040
Trainable params: 112,536
Non-trainable params: 504
```

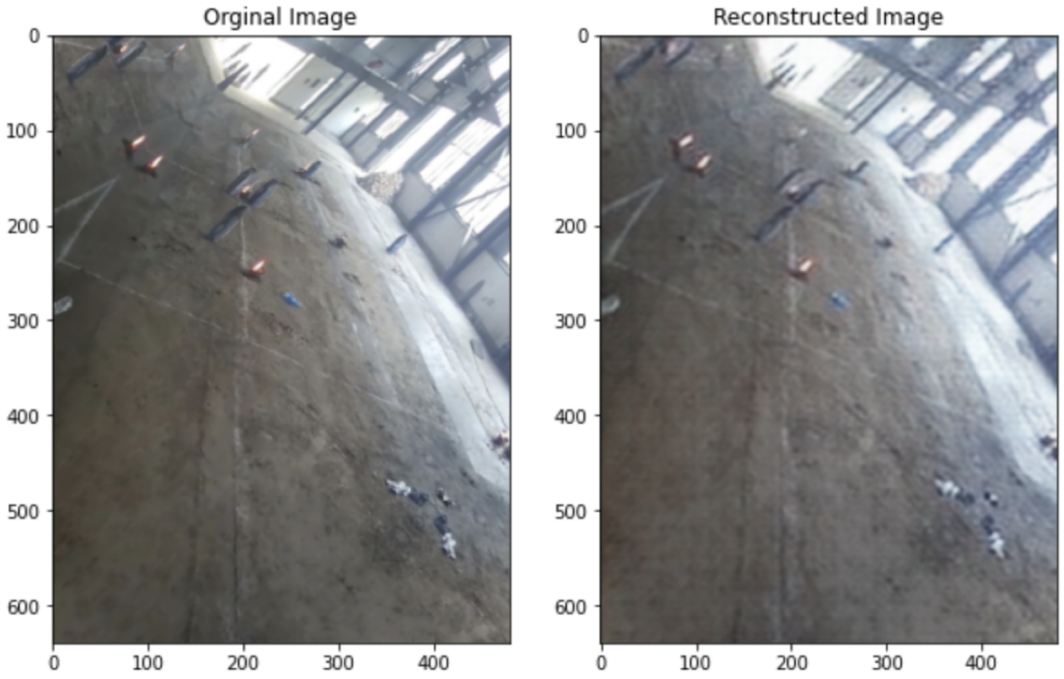
Decoder

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_transpose_47 (Conv2DT (None, 40, 30, 72)        62280
_________________________________________________________________
batch_normalization_85 (Batc (None, 40, 30, 72)        288
_________________________________________________________________
conv2d_transpose_48 (Conv2DT (None, 80, 60, 48)        31152
_________________________________________________________________
batch_normalization_86 (Batc (None, 80, 60, 48)        192
_________________________________________________________________
conv2d_transpose_49 (Conv2DT (None, 160, 120, 24)      10392
_________________________________________________________________
batch_normalization_87 (Batc (None, 160, 120, 24)      96
_________________________________________________________________
conv2d_transpose_50 (Conv2DT (None, 320, 240, 12)      7212
_________________________________________________________________
batch_normalization_88 (Batc (None, 320, 240, 12)      48
_________________________________________________________________
conv2d_transpose_51 (Conv2DT (None, 320, 240, 6)       1806
_________________________________________________________________
conv2d_transpose_52 (Conv2DT (None, 640, 480, 3)       165
=================================================================
Total params: 113,631
Trainable params: 113,319
Non-trainable params: 312
```

The CNN autoencoder could reconstruct images close to the original images. Example below:

Auto-encoder trained by all data

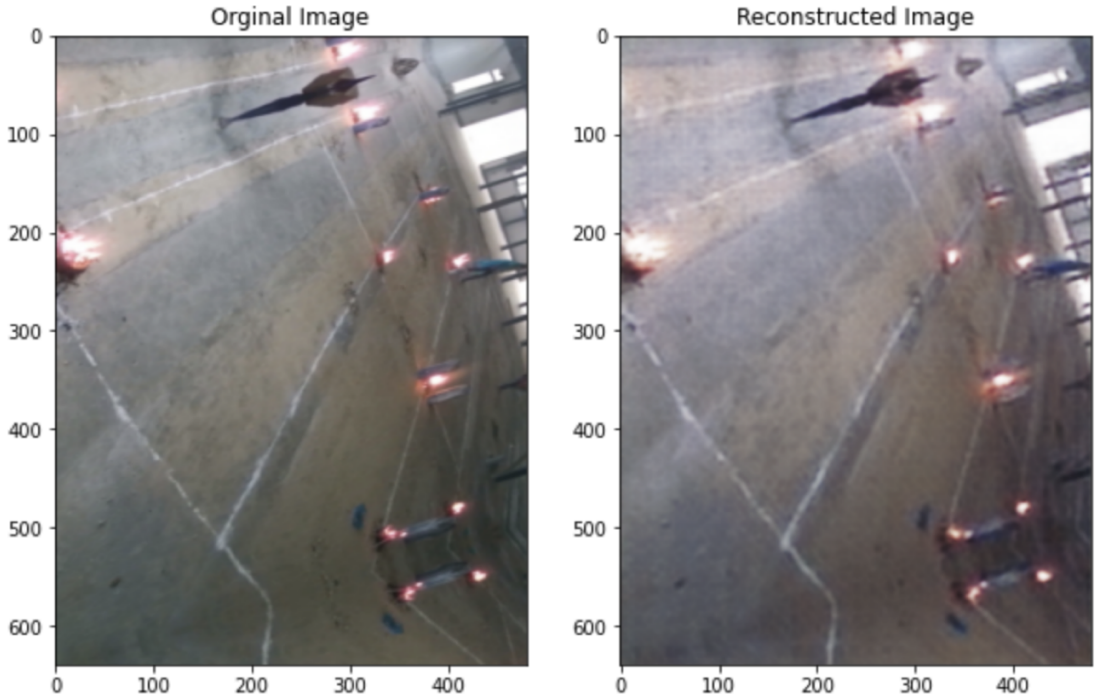
```
[ ]  sample_compare_plot(11)
```



The autoencoder trained by all the training data could reconstruct images with no obvious differences, i.e., almost 100% the same as the original images.

Auto-encoder trained by fire images
We can the emphasis on red color by the model

```
[ ]  sample_compare_plot(20)
```



The autoencoder trained by only fire images, reconstructs images with emphasis on light, as there is more red color on the reconstructed image.

Step 2: Train a CNN classifier using lower level layers from autoencoder

```
n_filters = 12
Conv2D = functools.partial(layers.Conv2D, padding='same', activation='relu')
BatchNormalization = layers.BatchNormalization
cnn_model = tf.keras.Sequential([
    # Define the first convolutional layer
    layers.Input(shape=(640, 480, 3), name='input'),
    Conv2D(filters=1*n_filters, kernel_size=5,  strides=2),
    BatchNormalization(),

    Conv2D(filters=2*n_filters, kernel_size=5,  strides=2),
    BatchNormalization(),

    Conv2D(filters=4*n_filters, kernel_size=3,  strides=2),
    BatchNormalization(),
    layers.MaxPool2D(pool_size=(3, 3), strides=1, padding='same'),

    filter72, batch_normal1, filter96, batch_normal2,

    # Define the max pooling layer
    layers.MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),

    layers.Flatten(),
    layers.Dense(128, activation='relu',
                 kernel_regularizer=tf.keras.regularizers.L2(0.01)),
    layers.Dropout(0.2),
    layers.BatchNormalization(),
    layers.Dense(32, activation='relu',
                 kernel_regularizer=tf.keras.regularizers.L2(0.01)),
    layers.BatchNormalization(),
    layers.Dense(1)
```

A trick to train this model is to freeze the lower level layers first, unfreeze them after a few epochs since the loss would be much higher at the beginning, although it may change trained weights too much.

Accuracy Test Set: 93.24%
Accuracy New Data: 100%
Error rate on new data: 0%

We see a surprising result, i.e., an error rate of 0%, which is already an ideal result. Since the misclassified images are not as much as the training data, this result could be the product of luck. Therefore, to test this we could train a GAN to get a higher performance discriminator.

GANs

General CNN generator GAN:

A basic Convolutional decoder as the generator.

```
def make_generator():
  n_filters=12
  input = keras.Input((100))
  x = layers.Dense(units=20 * 15* 8* n_filters)(input)
  x = layers.LeakyReLU(0.3)(x)
  x = layers.Reshape(target_shape=(20, 15, 8 * n_filters))(x)
  x = layers.BatchNormalization()(x)
  x = layers.LeakyReLU(0.1)(x)
  output = pretrained_model.decoder(x)
  generator = keras.Model(input, output)
  return generator
```

A discriminator uses lower level layers from the trained auto-coder from model 2.

```
def build_discriminator():
    n_filters = 12
    Conv2D = functools.partial(layers.Conv2D, padding='same', activation='relu')
    BatchNormalization = layers.BatchNormalization
    cnn_model = tf.keras.Sequential([
        # Define the first convolutional layer
        layers.Input(shape=(640, 480, 3), name='input'),
        Conv2D(filters=1*n_filters, kernel_size=5,  strides=2),
        BatchNormalization(),

        Conv2D(filters=2*n_filters, kernel_size=5,  strides=2),
        BatchNormalization(),

        Conv2D(filters=4*n_filters, kernel_size=3,  strides=2),
        BatchNormalization(),
        layers.MaxPool2D(pool_size=(3, 3), strides=1, padding='same'),

        filter72, batch_normal1, filter96, batch_normal2,

        # Define the max pooling layer
        layers.MaxPool2D(pool_size=(2, 2), strides=2, padding='same'),

        layers.Flatten(),
        layers.Dense(128, activation='relu',
```

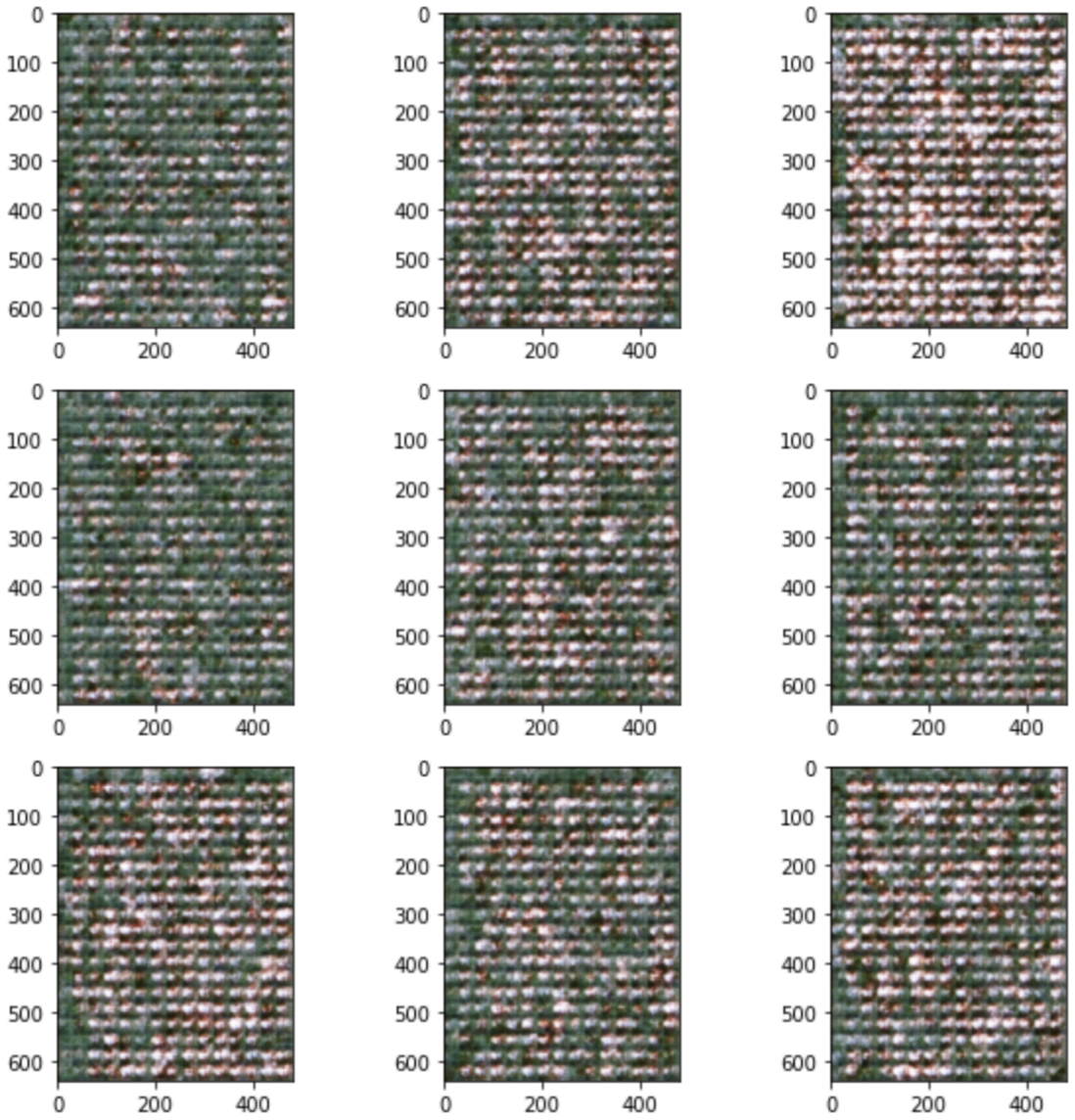
```
            kernel_regularizer=tf.keras.regularizers.L2(0.01)),
        layers.Dropout(0.2),
        layers.BatchNormalization(),
        layers.Dense(32, activation='relu',
            kernel_regularizer=tf.keras.regularizers.L2(0.01)),
        layers.BatchNormalization(),
        layers.Dense(1)
    ])

    return cnn_model
```

Accuracy Test Set: 93.24%

Accuracy New Data: 95%

Error rate on new data: 5%



Images generated from random noise by the generator at the last iteration

Conclusion:

The accuracy has not been improved compared to the CNN classifier with trained layers. The error rate increased from 0% to 5%. This result shows that the generator is not able to generate any images close to fire images. It failed against the discriminator as a result of the ever decreasing loss of the generator. The failure of the generator caused the complexity of neural networks to . A shallow generator cannot generate high-resolution images, so the generator only learned to generate light colors to fool the discriminator during training.

GAN with variational autoencoder(VAE)

VAE

Train a VAE is harder than a CNN autoencoder since VAE needs to sample from latent space by generating images from learned mean and variance vectors.

1. Encoder: Similar to CNN auto-encoder plus additional dense layers for latent dimension of variance and mean
2. Sampling Layer: Generate random samples from latent mean and variance
3. Decoder: Dense + ConvTranspose layers to recovery images
4. Different designs:

   VAE1-- with Customized Sampling layer and training step

   VAE2-- Add loss to Keras model object and use built-in training loop
5. Transfer Learning: Use pertained low level layers from Convolutional auto-encoder to improve performance
6. Optimization and loss: Use mean absolute error(MAE) + KL- divergence regularization as VAE loss.

Notice: Use MAE to prevent overflow since the final images size is (640, 480, 3)

Build a GAN by VAE generator

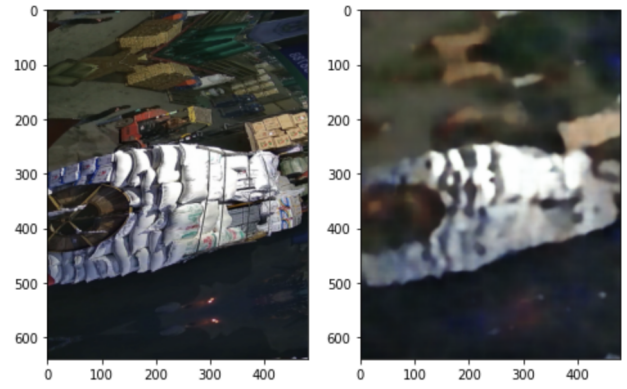
1. Generator: Use the trained VAE, which was trained to produce low-resolution fire images. The generator will take a batch of training data, reconstruct them, and then feed the reconstructed images to the discriminator. When generated images get close to the real images, its loss will decrease, the loss of the discriminator will increase. Feed a batch of training data of both fire and non-fire images is to limit the loss and make non-fire images close to fire.

2. Discriminator is the same as the general GAN.

A sample of how the VAE reconstruct images

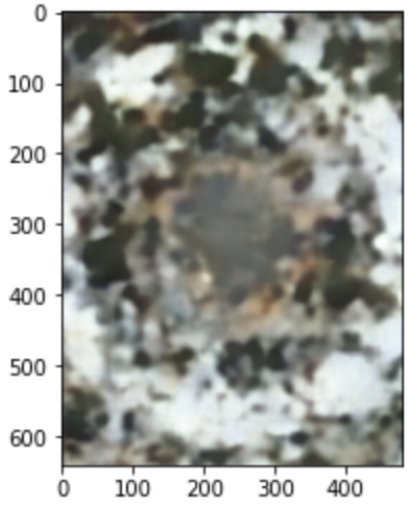
```
def sample_plot(sample_index, original, restructed):
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 8))
  ax1.imshow(original[sample_index].squeeze())
  ax2.imshow(restructed[sample_index].numpy().squeeze())
  plt.show()
```

```
sample_plot(19, test_sample, test)
```



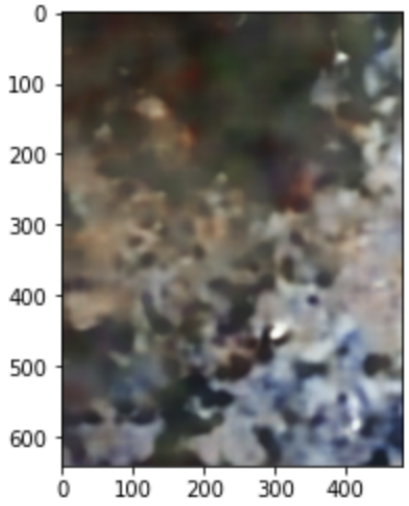
Images generated from the generator by random noise.

[ ] <matplotlib.image.AxesImage at 0x7f3039c57150>

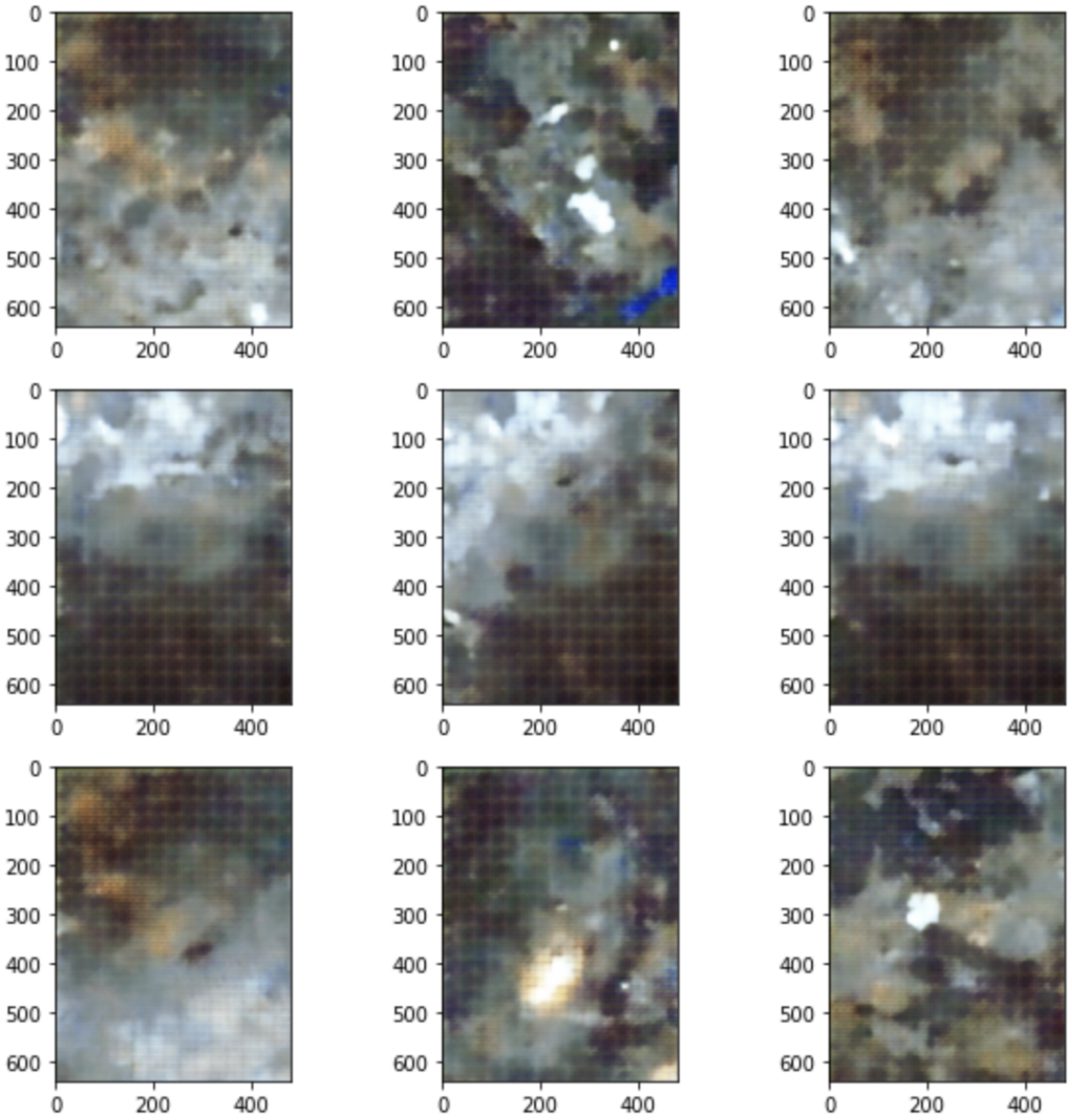


[ ]
```
# Generate images by random mean and variants
test_mean = np.random.normal(size=(1, 200)).astype(np.float32)
test_var = np.random.normal(size=(1, 200)).astype(np.float32)
test_z = vae.sampling(test_mean, test_var)
t3 = vae.decode(test_z)
plt.imshow(t3.numpy().squeeze())
```

<matplotlib.image.AxesImage at 0x7f3039d00e10>



Images generated after training:



Accuracy Test Set: 93.03%
Accuracy New Data: 97%
Error rate on new data: 3%

There is an improvement of accuracy on the new data.
Since the VAE is to construct variations of original images, the goal is to use the generator to produce images with fire features to fool the discriminator.

Lessons Learned

1. As deep learning models are data-hungry, limited data sources would be a big problem, especially for imbalanced label distribution. Image augmentation is a solution, but more data are still needed.

2. Data quality is an important factor for machine learning. My data were collected from a fire equipment company, the data variety was too low since those images did not include many scenes. For example, this model could correctly classify images from the office, factory, a warehouse's inventory, home, but it may not be good in open spaces like streets.

3. For real-world applications, I would spend time on image segmentation to try multi-object detection.

4. Compared to the general CNN model, using trained lower level Convolutional layers can improve the accuracy of the model, especially how the model detects high level features, which results in a low misclassification error rate.

5. The discriminator from GANs does a better job than a general CNN classifier. Using a generator against a classifier is a good way to improve performance.

6. How to design different deep learning model structures: I spent time on the original code of Tensorflow as an engineer to understand how the library was organized, especially customized training.

7. How to implement and use auto-encoders to do transfer learning

8. How to implement Adversarial Generative Neural network (GANs)

9. The limitations of training a deep neural network