Data C200 Graduate Project - Final Submission, Project Writeup

Sander August Heggland Schrader, Aksel Nordli Katralen

May 11, 2024

1 Abstract

This paper discusses how machine learning models can be used to help emergency services with quick information for decision-making. The paper focuses on two main tasks, Task A and B. Task A aims to create a model for classifying which type of natural disaster has occurred (either SoCal Fire or Midwest Flooding). Task B on the other hand aims to create a model which can classify the damage level after Hurricane Matthew. Both tasks use different models. Logistic regression models are used as a baseline, while CNN(Convolutional Neural Network) models are used as the most optimal models. The regression models are trained on features extracted from the images in the dataset, like RGB(Red, Green and Blue) and HSV(Hue, Saturation, Value) features. The results of the model were generally good. The regression models scored around 90-91% accuracy for Task A and around 54% F1 score for Task B. The CNN model were slightly better, coming in at 99.3% accuracy for task A.

2 Introduction

Emergency situations are often chaotic and difficult to get an overview over. Given that time is a factor in these kind of situations, can machine learning models help in getting an overview and information used for decision-making? Research papers like [2], explain how models can be built for extracting information for decision-making in natural disasters. In the paper they represent how a CNN model can be used for this purpose. The CNN model they use, is trained on a pre-trained ResNet 50 architecture. Other research papers have explored similar approaches like [1], which assesses damage on buildings after the tsunami in Japan. This paper also uses a CNN model to assess the damage level.

This project will explore two main areas, Task A and Task B. Task A is to help an agency to deploy the correct subdivision to the corresponding disaster. For example the firefighter to the SoCal Fire disaster. The task explores how satellite images can be used for disaster-type classification between SoCal Fire and Midwest Flooding.

Task B will explore how to assess the level of damage after a disaster, more specifically after Hurricane Matthew. The damage levels used in the dataset, vary in the form of different damage labels. The labels use a scale from 0 to 3, where 0 is no damage and 3 is completely destroyed.

We are given a dataset that is a subset of the larger dataset xBD dataset [2]. The dataset contains satellite images of three different disasters; Hurricane Matthew, SoCal Fire, and Midwest Flooding. In addition to the images and the disaster type, there is also information about the damage level for the specific area as well as information about the height, width, and size of the images. All the images are taken after a disaster has occurred.

The project will use different approaches for the models for both tasks. First, we will provide benchmark models using logistic regression. Then we will make CNN models and compare these against our regression models if the regression models cannot reach the treshlines.

3 Description of data

The granularity of the dataset is a image of an area affected by a disaster. Each image contains a certain amount of pixels. Each pixel contain vital information about color and texture we can use to train our model.

Since the dataset contains three different disasters, where the disasters correspond to different tasks, our first job was separating the disaster into Task A and Task B. Task A contains all the images for disaster types SoCal Fire and Midwest Flooding, while Task B contains only the images for Hurricane Matthew. The distribution of the amount of data per disaster is shown in figure 1. SoCal Fire and Midwest Flooding are about the same amount, while Hurricane Matthew has a considerably higher amount of data. There is no missing data in either disaster.

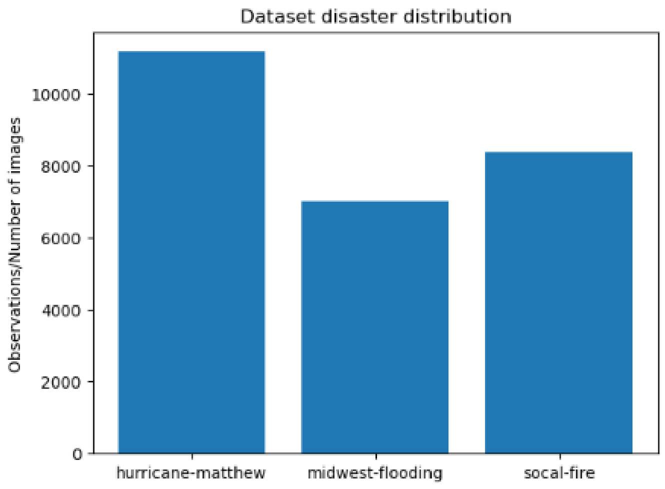


Figure 1: Distribution of disasters in the dataset.

The distribution of labels varies from disaster to disaster. The labels are divided into four categories; 0, 1, 2, and 3. They each represent a level of damage. 0 represents no damage, 1 minor damage, 2 major damage, and 3 destroyed. SoCal Fire and Midwest Flooding have almost no data in labels 1,2,3, while Hurricane Matthew is relatively more evenly distributed. The distribution of Hurricane Matthew can be seen in figure 2. As shown in the figure, Hurricane Matthew has most of its data concentrated in the "Minor Damage" label.

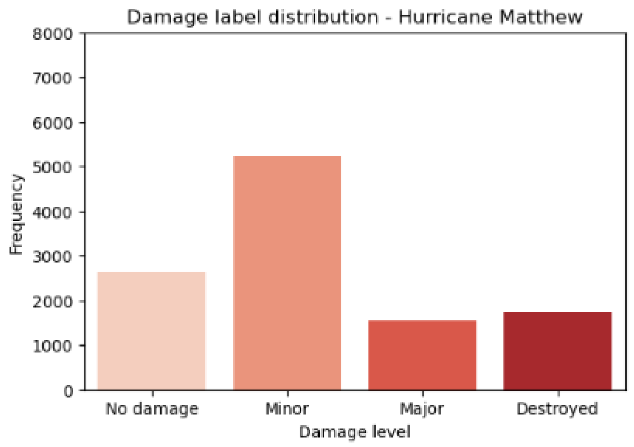


Figure 2: Distribution of labels for Hurricane Matthew.

3.1 Exploratory Data Analysis

Task A

For the EDA for task A, our main goal was to extract features from the images that clearly vary between the disasters SoCal Fire and Midwest Flooding. Our initial thought was to use color as a feature, as there usually are very strong color differences between fires and floods. In addition, there are different landscapes for normal conditions in California and the Midwest.

The way we explored color, was through exploring each RGB channel separately for each disaster. We took the average of the color distribution of all images for both disasters in Task A. We then presented the data in a histogram with 8 separate bins which represent the distribution of the color channel. By doing this, we could check if there were generally speaking more of a color for a disaster. We could also check the amount in each bin for every histogram and compare them with each other. The reason for why we chose 8 bins is through testing different amounts of bins and see what bin size in general scored the best. The histogram showing this color distribution for the disasters, is shown in figure 3.

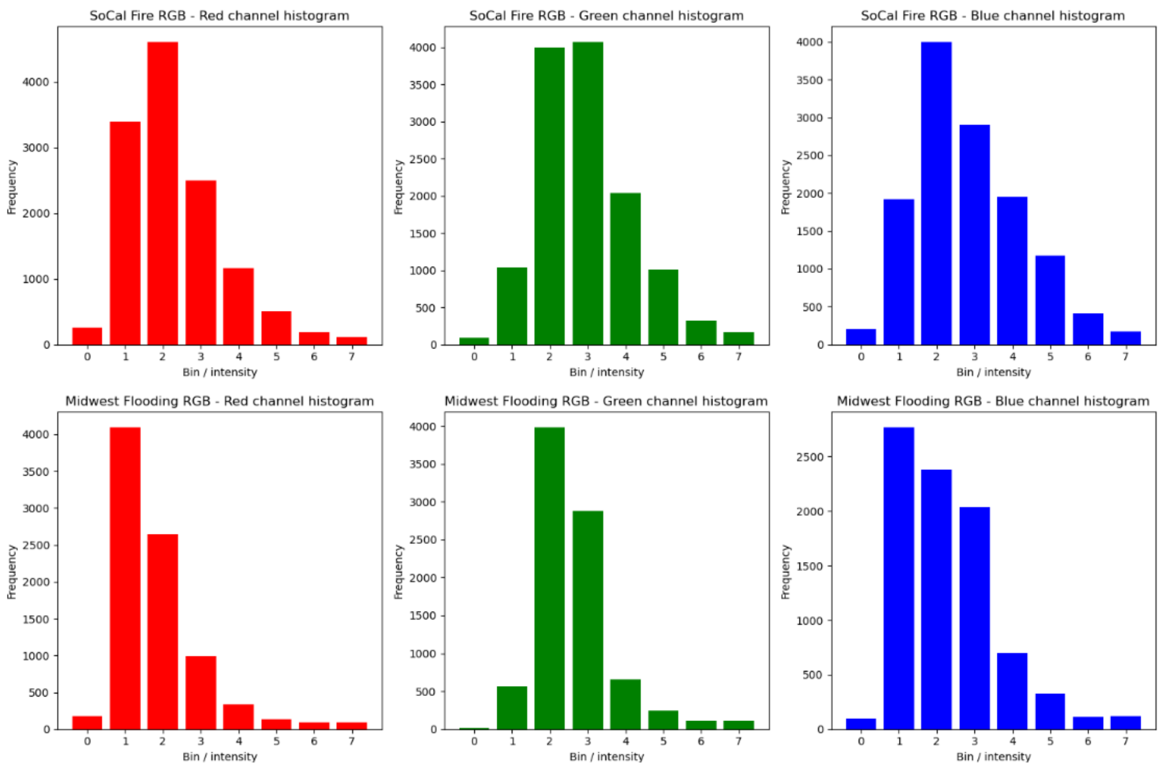


Figure 3: Distribution of color for RGB (red, green, and blue) channels per disaster.

As we can see from figure 3, there are clearly differences between the color distribution for the bins. As a result of this, we can then choose this as a feature for our model. This will give us 8 features per channel, which gives us a total of 24 features, just from these color distributions.

As we saw color distributions as a viable feature, we looked at how we could extract more information that resembled this. Therefore we tested out HSV (hue, saturation, value). We used the method as RGB, by plotting the distribution into a histogram with 8 bins. This is shown in figure 4. Like RGB, the bin distribution varies by disasters, making it a suitable feature we can use. The addition of HSV as a feature brings in a new 24 feature in which the model can train on. This brings our total amount of features up to 48 features.

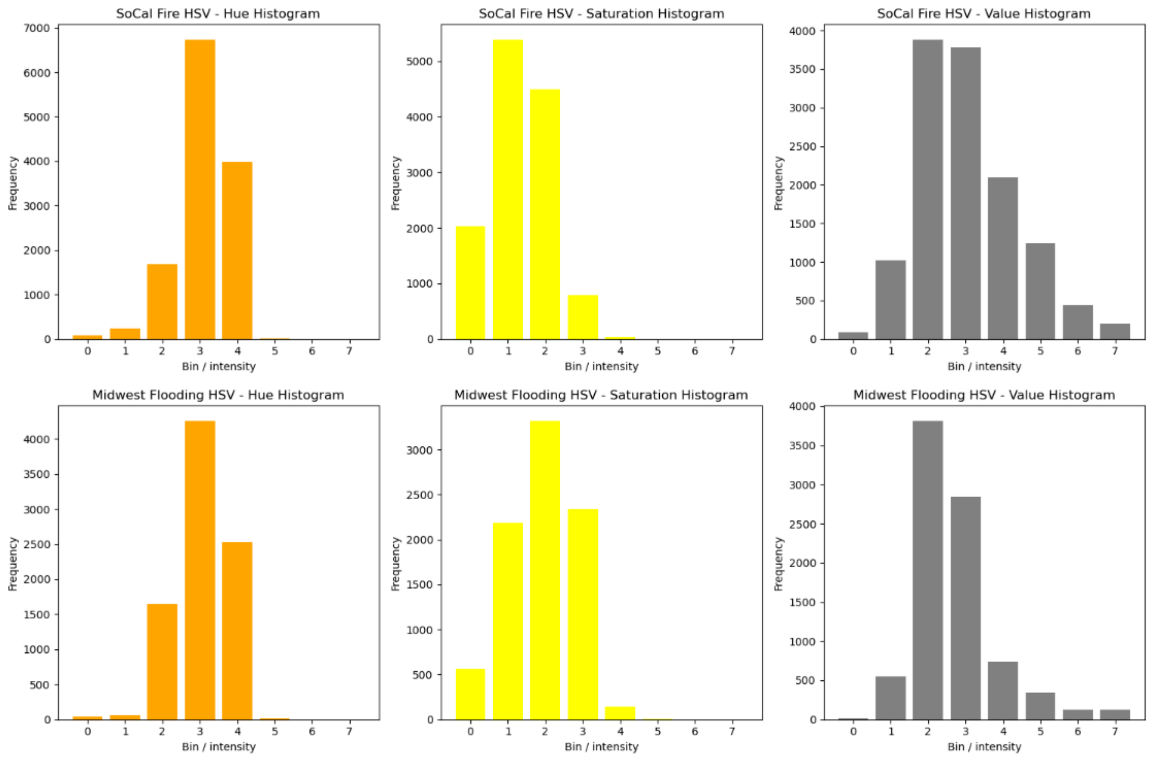


Figure 4: Distribution of color for HSV (hue, saturation, and value) channels per disaster.

Task B

In the EDA for Task B, we continued looking at using RGB and HSV as features. First we divided the data from Hurricane Matthew based on the labels. Then we used our previous histograms to plot the distribution for each channel in RGB and HSV for every label. The plots, as shown in figures 5 and 6.

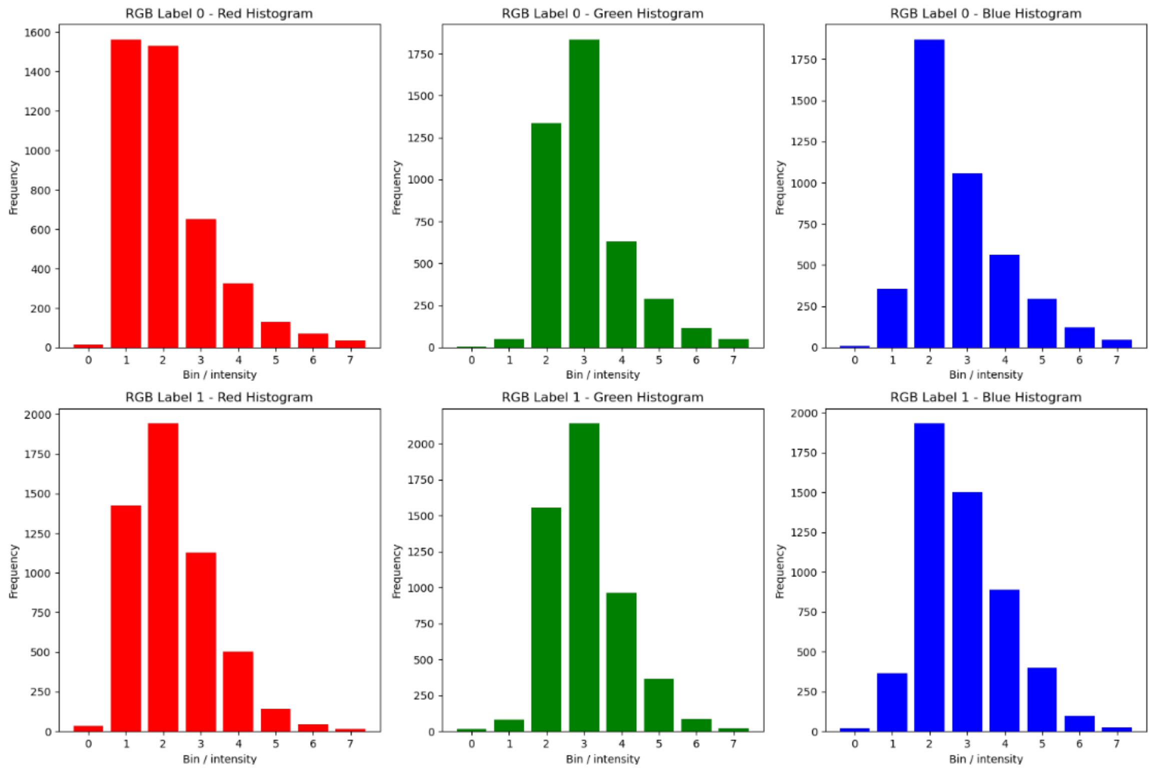


Figure 5: Distribution of RGB channels per label in Hurricane Matthew.

Another area we looked at was contrast. Our theory was that we would see some difference in contrast comparing labels because scattered debris would most likely increase the variation in intensity (contrast). Therefore, we have a plot showing the difference between labels. This can be seen in figure 7. The figure shows how the contrast also is divided into 8 bins.

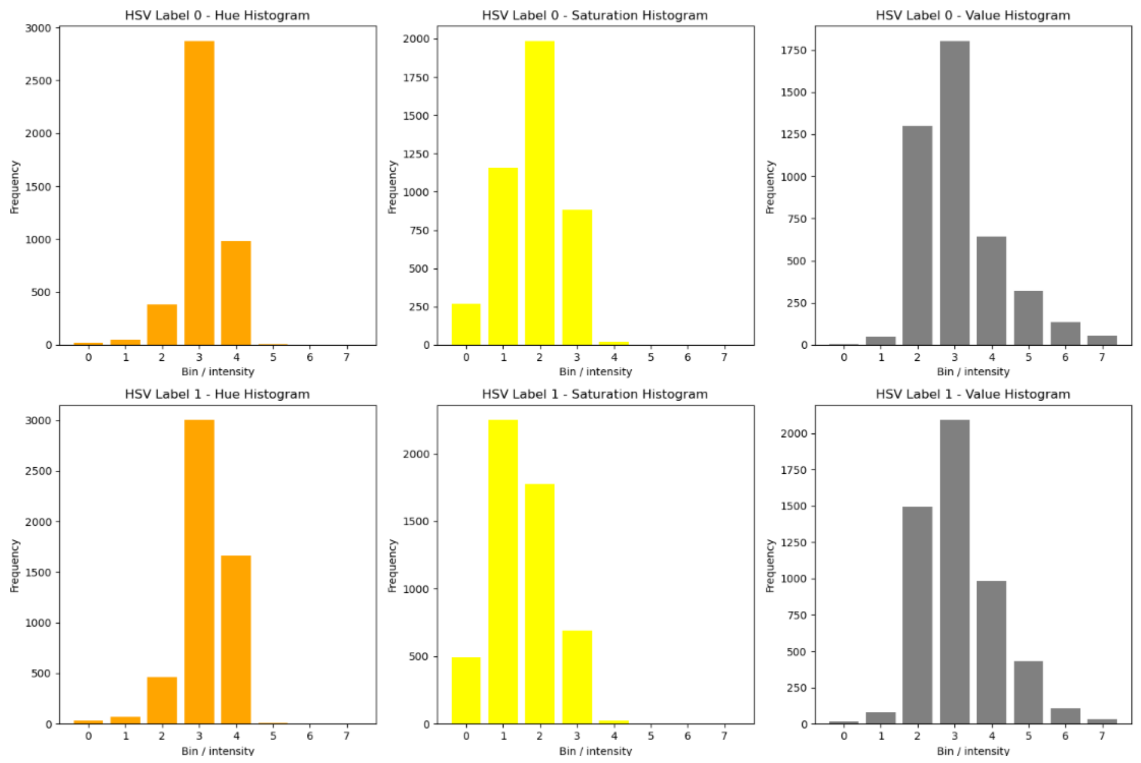


Figure 6: Distribution of HSV channels per label in Hurricane Matthew.

The last area we explored in our EDA, was the use of LBP (Local Binary Pattern). The theory behind this was that LBP is usually very good at detecting edge patterns. These patterns would vary from label to label, especially between no damage and destroyed buildings. We used uniform LBP to reduce complexity and increase robustness against noise. Figure 8 shows the relation between the different labels. As we can see from the figure, labels 0, 1, and 2 have similar distributions, although not the same values, while label 3 has a different distribution. All in all, this gives us a total of 66 features.

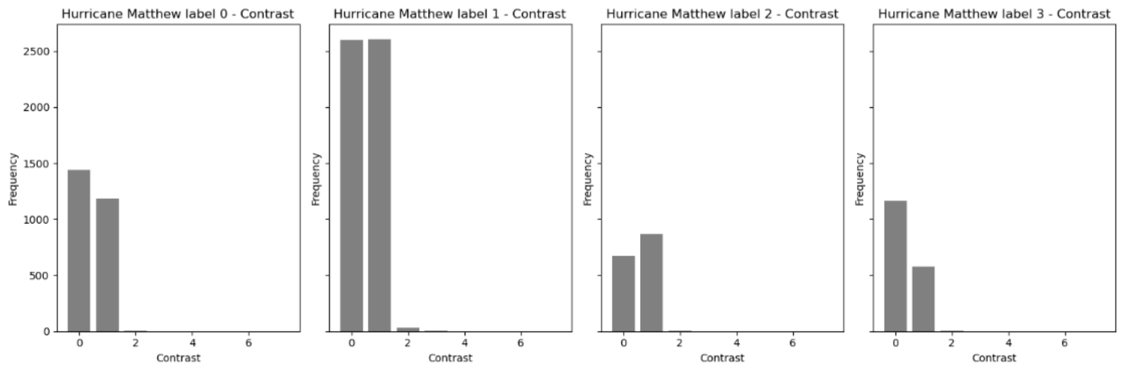


Figure 7: Contrast per label for Hurricane Matthew.

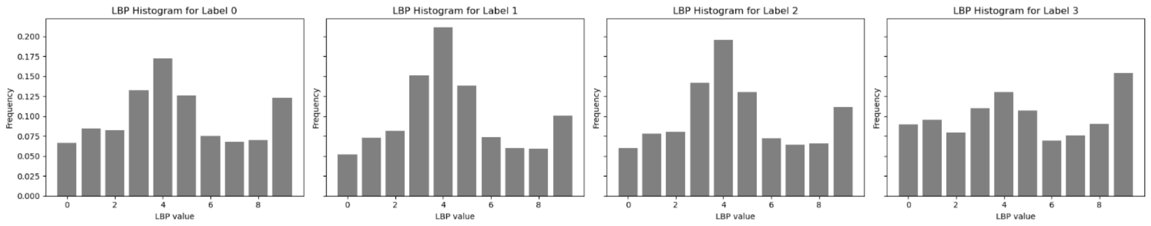


Figure 8: LBP per label for Hurricane Matthew.

4 Methodology

During the project, we decided to explore two different modeling approaches. One of them was the required logistic regression model. The second modeling type explored was Convolution Neural Network (CNN) as these have over time proven to be extremely efficient and accurate on image classification tasks.

4.1 Modeling

4.1.1 Feature Engineering

Feature engineering is an important step in regression models. Especially with logistic regression. Our first attempts included getting the RGB and HSV (hue saturation value) distributions. The distributions were then split into 8 bins each creating a total of 48 features that we could use in our regression model.

4.1.2 Logistic Regression

Our first model was a logistic regression model train of features generated during the feature engineering step. Before training the dataset was sampled and shuffled. The sampling was done by reducing the number of labels in each label to the same number of instances in the minimum label. This was done in order to reduce bias in the dataset, as the model might get confused from a destroyed building when it is trained on mostly non-damaged buildings. Since SoCal Fire and Midwest Flooding were pretty much even in size, we saw no need for sampling further.

Then the features were extracted from the new dataframe we made. The target values, the disaster types, were encoded from SoCal Fire/Midwest Flooding to 0/1. The feature vectors were scaled using a StandardScaler, and the data was split into a training and validation set. Then we train the logistic regression model and evaluate the model by calculating the mean accuracy with cross-validation.

4.1.3 Convolution Neural Network

Convolution Neural Networks are deep learning algorithms that use convolution layers. The network uses kernels (small matrices) to extract features from the input channels. The method has proven to be especially accurate and efficient for image classification. This was one of the reasons we decided to explore and use Convolutional Neural Networks for this classification task.

Convolution Neural Networks require a fixed data format. Due to this every image has to be resized to the same size. This was completed by padding the satellite photos to the same aspect ratios before resizing them. This ensures that the images keep as much detail and avoid smearing the images by changing the aspect ratios.

Custom Model

Our first trials utilizing Convolution neural networks were used with a custom model consisting of three convolution layers with max pooling. The convolution models used a kernel size of 3x3 and the number of filters was increased from 32-128 in increments of n*32. The model was then flattened before the data was put into two dense layers of size 256 and then 124 before the binary classification output was calculated. The model's architecture can be seen in Figure 9.

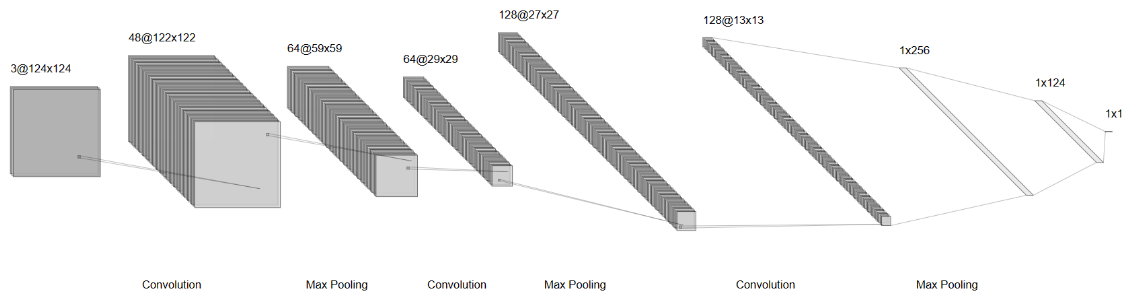


Figure 9: Image showing the architecture of the custom CNN model created

Transfer Learning

Transfer learning is a powerful technique used in machine learning. Transfer learning is done by using a pre-trained model and slightly adjusting the weights to fit a new problem. To adjust for new outputs a new dense layer is added and the model is refitted. This method is efficient and cost-effective. We decided on using the ResNet 50 which contains 48 convolution layers and outputs 1000 classes. The model is trained on 14 million images[3].

5 Summary of results

The group used multiple models to solve both Task A and Task B. The Logistic Regression models served as a baseline for our results and CNN models were used to get better results. This is because CNN's are generally larger models fit for image classification tasks.

5.1 Task A

5.1.1 Logistic Regression

Our logistic regression models were generally able to get good results for our classification task. For Task A our regression model was able to get an accuracy of 91% on our validation set and 90% on the provided test set. This was done with a total of 48 features created from our EDA analysis. To validate the model and protect against overfitting we decided to use a train test split by taking 20% of our data to validate our model during training. The results for Task A can be seen in table 1.

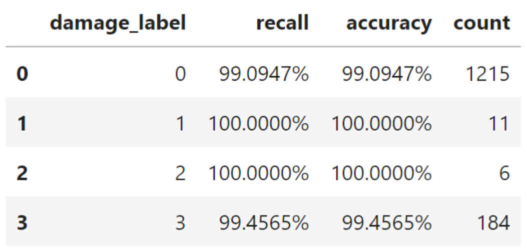
Fold Number	Accuracy
1	0.7857
2	0.9286
3	0.9196
4	0.9643
5	0.9554
Mean	**0.9107**

Table 1: Cross-validation accuracies for each fold and mean accuracy for the regression model.

5.1.2 Convolution Neural Networks

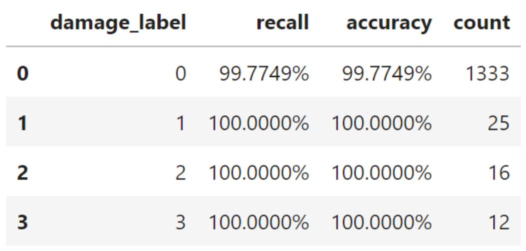
Custom Model

The first convolution neural network used was our custom model. Through hyperparameter testing, we found that training the model with around 50 epochs seemed to be a good fit. The model was quickly able to get a good accuracy but since the task required an extremely high accuracy we had to train it for more epochs as the model had to find the best parameter weights for our problem. Plotting the training accuracy and our validation accuracy we could see that the model was not overfitting but found that the model had trouble getting higher accuracy. There could be multiple reasons for this, such as having the learning rate to high or the model being to small such that it could not extract enough information from our dataset. The resulting model had a test accuracy of 98.6% which was not enough to reach the threshold of 99.27%.



	damage_label	recall	accuracy	count
0	0	99.0947%	99.0947%	1215
1	1	100.0000%	100.0000%	11
2	2	100.0000%	100.0000%	6
3	3	99.4565%	99.4565%	184

Figure 10: SoCal recall and accuracy by damage level



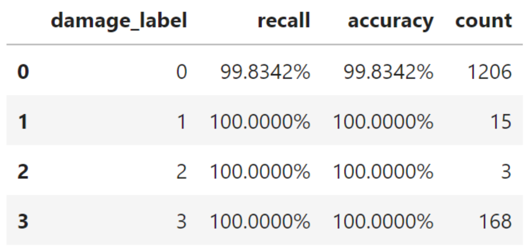
	damage_label	recall	accuracy	count
0	0	99.7749%	99.7749%	1333
1	1	100.0000%	100.0000%	25
2	2	100.0000%	100.0000%	16
3	3	100.0000%	100.0000%	12

Figure 11: Midwest recall and accuracy by damage level

Figure 12: Image showing the custom Model's accuracy, and recall grouped by damage label

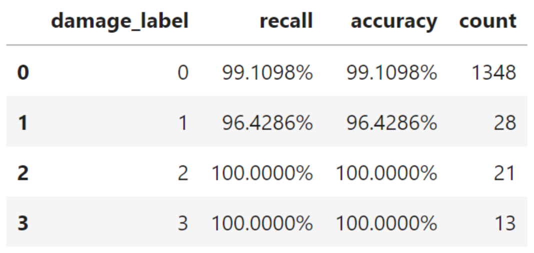
ResNet 50

As the team had experience with transfer learning from other classes and saw that this method was able to get a high accuracy in other problems we decided to try it out on this problem. Early results showed that the model was able to quickly adapt to our problem showing an initial high accuracy on our training set. The validation accuracy quickly followed although taking a little bit of time. The ResNet 50 model is a lot larger than our custom model and were able to get an accuracy of 99.5% on our training set. A little over our custom model. This proved to be valuable as the model was able to get an accuracy of approximately 99.3% on the kept-out testing data.



	damage_label	recall	accuracy	count
0	0	99.8342%	99.8342%	1206
1	1	100.0000%	100.0000%	15
2	2	100.0000%	100.0000%	3
3	3	100.0000%	100.0000%	168

Figure 13: SoCal recall and accuracy by damage level



	damage_label	recall	accuracy	count
0	0	99.1098%	99.1098%	1348
1	1	96.4286%	96.4286%	28
2	2	100.0000%	100.0000%	21
3	3	100.0000%	100.0000%	13

Figure 14: Midwest recall and accuracy by damage level

Figure 15: Image showing Fitted ResNet50's accuracy, and recall grouped by damage label

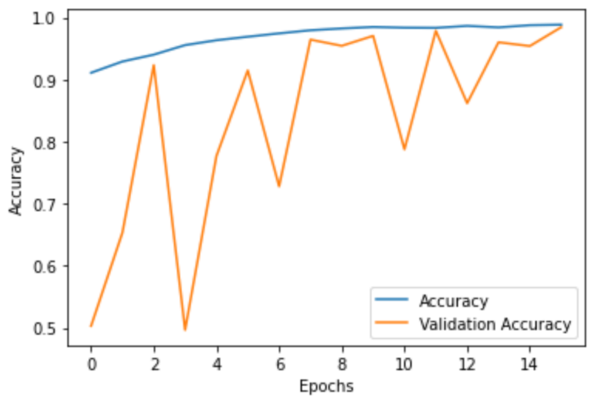


Figure 16: Image showing the Training vs Validation accuracy per epoch

The results in Figure 15 show the model does not seem biased toward any damage labels. However, the model seems better at predicting the social fire locations than the midwest flooding. Figure 16 shows part of the training of the ResNet 50 architecture where we see that the is more prone to overfitting.

5.2 Task B

5.3 Logistic Regression

Logistic regression was able to show good results on this classification task classifying the different damage labels found in Hurricane Matthew. The model had 66 features and had a mean F1 score of 54% on both the validation set and the test set. The F1 score of each fold is presented in table 2. The results for each fold vary from 53% to 56%, which may indicate that the model is sensitive to specific splits.

Fold Number	F1 Score
1	0.5499
2	0.5452
3	0.5344
4	0.5636
5	0.5514
Mean	**0.5489**

Table 2: F1 scores for each fold and the mean F1 score of the model.

6 Discussion

The results from Task A show that image classification to predict which disaster an image is taken from is a feasible task with high accuracy. The model shows low biases towards different damage labels and this is important since we might be more interested in predicting damaged buildings as these are the buildings that need attendance during a disaster. Although the model showed good results, there is a small amount of data from the different damage labels in the SoCal Fire and Midwest Flooding and this might therefore be misleading and is something that we should be aware of.

The three different approaches also show that there are multiple ways to accurately classify images. The different models were all able to predict with high accuracy. It was surprising that the ResNet 50 model performed the best, but this could be due to the architectural differences between the two CNN models as the custom CNN model is trained on smaller images vs the larger model. Therefore we might be able to see better results with larger models. However, this shows that tuning a pre-trained model is an approach that can be useful for image classification tasks.

Task B also showed how damage level classification is achievable. However, the results for the regression model showed that the results can be improved. To achieve better results a CNN model would be the best way to go. A CNN model would be able to catch minor differences and make better features than we currently have for damage-level classification.

Social impacts of the models
The social impacts of the models we have created vary. The models for task A aim to classify the type of disaster based on the provided images from each disaster. In itself, this is not particularly useful, since emergency responders often know the type of disaster in advance. Normally there is no need for responders to clarify this in advance. Furthermore, since most images in the dataset are labeled as not damaged, the models are more or less trained to classify which area the images are from, not the actual disasters. Therefore, the social impact and the social need for these models in task A would be low.

The models for task B on the other hand, have a bigger social impact. Emergency responders often do not know the extent of the damage when responding to natural disasters. A damage label classifier model could help decision-makers respond more effectively and faster to the areas affected the most by the disaster.

Ethical concerns
Using machine learning models to understand disasters and their impacts, also raises ethical concerns. These ethical concerns mainly dwell on the accuracy of the models. More specifically around the accuracy of false positives and false negatives. For Task B, a false positive would mean that emergency resources would be diverted to areas with lower damage. This could take up resources which is needed elsewhere. False negatives, on the other hand, are even worse. A high number of false negatives would mean that damaged areas would not be discovered and receive the help they need. Therefore, the accuracy of these models needs to be very high, and considerably higher than humans.

7 Conclusion

In conclusion, this paper has shown that machine learning models can be used to classify both disasters and damage levels. Through use of models like logistic regression and CNN, our models have performed very well. The logistic regression model for Task A scored around 90-91% in accuracy and for Task B 54% in F1 score. The CNN model for task A scored an amazingly 99%, outperforming the regression model by 9%. These results underline the importance and effectiveness of machine learning in emergency response situations and understanding satellite images.

References

[1] Aito Fujita et al. "Damage Detection from Aerial Images via Convolutional Neural Networks". In: *15th IAPR International Conference on Machine Vision Applications (MVA)*. Nagoya University. Nagoya, Japan, May 2017. URL: https://www.mva-org.jp/Proceedings/2017USB/papers/01-02.pdf.

[2] Ritwik Gupta et al. "xBD: A Dataset for Assessing Building Damage from Satellite Imagery". In: (2019). URL: https://arxiv.org/pdf/1911.09296.

[3] Nitish Kundu. "Exploring ResNet50: An In-Depth Look at the Model Architecture and Code Implementation". In: (Jan. 2023). URL: https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f.

8 Attachments

8.1 Video Presentation

Link to the video presentation: `https://www.youtube.com/watch?v=RBevr_4p6cs`

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LinearRegression, LogisticRegression
        from sklearn.utils import shuffle
        from sklearn.metrics import accuracy_score
        from keras.preprocessing.image import ImageDataGenerator
        import keras
        from skimage.transform import resize
```

```
2024-04-29 05:50:44.089969: I tensorflow/core/util/port.cc:110] oneDNN custom ope
rations are on. You may see slightly different numerical results due to floating-
point round-off errors from different computation orders. To turn them off, set t
he environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-04-29 05:50:44.132115: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in performa
nce-critical operations.
To enable the following instructions: AVX512F AVX512_VNNI, in other operations, r
ebuild TensorFlow with the appropriate compiler flags.
```

```
In [ ]: # Reading data from NPZ
        # ! REMEMBER TO GENERATE THE DATA WITH DATASET_PIPLELINE FIRST !
        train_data = np.load('../../data/CNN Disaster/train_data.npz')
        test_data = np.load('../../data/CNN Disaster/test_data.npz')
```

```
In [ ]: # Hyper parameters
        epochs = 10
        batch_size = 64
        optimizer = "adam"

        datagen = ImageDataGenerator(
                rotation_range=0,  # (degrees, 0 to 180)
                zoom_range = 0, # Randomly Zoom
                width_shift_range=0.1,  # Randomly Shift image % of width
                height_shift_range=0.1,  # Randomly Shift image % of height
                horizontal_flip=False,  # Randomly flip image
                vertical_flip=True)  # Randomly flip image
```

```
In [ ]: train_gen = datagen.flow(train_data["images"], train_data["labels"], batch_size=
        test_gen = datagen.flow(test_data["images"], test_data["labels"], batch_size=bat
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipykernel_36130/4148954939.py in <module>
----> 1 train_gen = datagen.flow(train_data["images"], train_data["labels"], batc
h_size=batch_size)
      2 test_gen = datagen.flow(test_data["images"], test_data["labels"], batch_s
ize=batch_size)

~/.local/lib/python3.10/site-packages/numpy/lib/npyio.py in __getitem__(self, ke
y)
    254                 if magic == format.MAGIC_PREFIX:
    255                     bytes = self.zip.open(key)
--> 256                     return format.read_array(bytes,
    257                                              allow_pickle=self.allow_pickle,
    258                                              pickle_kwargs=self.pickle_kwarg
s,

~/.local/lib/python3.10/site-packages/numpy/lib/format.py in read_array(fp, allow
_pickle, pickle_kwargs, max_header_size)
    830                     read_size = int(read_count * dtype.itemsize)
    831                     data = _read_bytes(fp, read_size, "array data")
--> 832                     array[i:i+read_count] = numpy.frombuffer(data, dtype=
dtype,
    833                                                              count=read_c
ount)
    834 

KeyboardInterrupt: 
```

In [ ]:
```
np.sum(train_data["labels"])/len( train_data["labels"]), len(train_data["labels"
```

In [ ]:
```
from keras.applications import ResNet50

base_model_1 = ResNet50(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)

model = keras.Sequential()
model.add(base_model_1)
model.add(keras.layers.Dense(1, activation="sigmoid"))

"""
model = keras.Sequential()
model.add(keras.Input(shape=(124,124,3)))
model.add(keras.layers.Conv2D(filters=48, kernel_size=(3, 3), activation="relu")
model.add(keras.layers.Conv2D(filters=48, kernel_size=(3, 3), activation="relu")
model.add(keras.layers.MaxPool2D(pool_size=(2, 2))) # 64 -> 32
#model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation="relu")
model.add(keras.layers.MaxPool2D(pool_size=(2, 2))) # 32 -> 16
#model.add(keras.layers.BatchNormalization())
#model.add(keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation="relu
model.add(keras.layers.Conv2D(filters=128, kernel_size=(3, 3), activation="relu"
model.add(keras.layers.MaxPool2D(pool_size=(2, 2))) # 16 -> 8
#model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256))
model.add(keras.layers.Dense(124))
model.add(keras.layers.Dense(1, activation="sigmoid"))
"""
```

In [ ]:
```
model.summary()
```

In [ ]:
```
model.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=["accurac
```

In [ ]:
```
model_hist = model.fit(train_gen, epochs=epochs, batch_size=batch_size, validati
```

In [ ]:
```
x_values = np.arange(0, epochs)
plt.plot(x_values, model_hist.history["accuracy"], label="Accuracy")
plt.plot(x_values, model_hist.history["val_accuracy"], label="Validation Accurac
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.plot()
```

In [ ]:
```
model.save('model_9.keras')
```

In [ ]:

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
```

In [ ]:
```
# Reading from HDF5
df = pd.read_hdf('../../data/CNN Disaster/data.h5', 'df')
df
```

Out[ ]:

	disaster	image	label	height	width	size
18769	1.0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	0	124	124	15376
23335	0.0	[[[13, 22, 15], [13, 22, 15], [13, 22, 15], [1...	0	124	124	15376
18627	1.0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	0	124	124	15376
18367	1.0	[[[2, 3, 2], [2, 3, 2], [3, 3, 3], [2, 3, 2], ...	0	124	124	15376
14623	1.0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	0	124	124	15376
...	...	...	...	...	...	...
20951	0.0	[[[19, 22, 24], [18, 21, 24], [14, 17, 19], [1...	0	124	124	15376
15603	1.0	[[[2, 3, 3], [3, 4, 3], [8, 12, 11], [27, 37, ...	0	124	124	15376
23468	0.0	[[[67, 89, 96], [67, 89, 96], [67, 89, 96], [6...	0	124	124	15376
14357	1.0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	3	124	124	15376
17420	1.0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	3	124	124	15376

15384 rows × 6 columns

Split, Balance and normalizing the dataset

This code splits the dataset into a 20/80 test train split. It also normalizes the RGB values from 0-1 and applies sampling to balance the two labels found in the dataset.

In [ ]:
```
def balance_data(df, n_samples=None, random_state=42):
    new_df = pd.DataFrame()
    for label in df['disaster'].unique():
        label_df = df[df['disaster'] == label]
        if n_samples is None:
            n_samples = len(label_df)
        resampled_df = resample(label_df, replace=True, n_samples=n_samples, ran
        new_df = pd.concat([new_df, resampled_df], axis=0)
    return new_df

min_samples = min(df['disaster'].value_counts())

balanced__df = balance_data(df, n_samples=min_samples)
```

```
print("-------------------------------------")
print("After sampling data")

disaster_counts = balanced__df['disaster'].value_counts()
print("\nCounts for each disaster type:\n", disaster_counts)

total_records = disaster_counts.sum()
proportions = disaster_counts / total_records
print("\nProportions for each disaster type:\n", proportions)

print("\nDamage level distribution for Midwest flooding:")
print(balanced__df[balanced__df['disaster'] == 0]['label'].value_counts(normaliz

print("\nDamage level distribution for SoCal fire:")
print(balanced__df[balanced__df['disaster'] == 1]['label'].value_counts(normaliz
```

```
-------------------------------------
After sampling data

Counts for each disaster type:
 1.0    7004
0.0    7004
Name: disaster, dtype: int64

Proportions for each disaster type:
 1.0    0.5
0.0    0.5
Name: disaster, dtype: float64

Damage level distribution for Midwest flooding:
0    0.960166
1    0.017561
2    0.013135
3    0.009138
Name: label, dtype: float64

Damage level distribution for SoCal fire:
0    0.866077
3    0.121216
1    0.008709
2    0.003998
Name: label, dtype: float64
```

In [ ]:

```
float_images = []
for image in balanced__df["image"].values:
    float_image = np.array(image).astype(np.float32)
    float_images.append(float_image)

float_images = np.array(float_images)/255
```

In [ ]:

```
X_train_cnn, X_test_cnn, y_train_cnn, y_test_cnn = train_test_split(float_images
```

Rotate The Images

Rotating the images 180 degrees and then 90* degrees gives us the images rotated at 0, 90, 180, and 270 degrees giving us a lot more data to work with.

* On bigger images the 90 degree rotation were disabled due to the large amounts of data created and limits on GPU memory size.

```
In [ ]: y_train_cnn = y_train_cnn["disaster"]
```

```
In [ ]: from skimage.transform import rotate

        def rotate_images(X_train, angle):
            num_images = X_train.shape[0]
            rotated_images = []

            for i in range(num_images):
                image = X_train[i]
                rotated_image = rotate(image, angle, preserve_range=True).astype(np.floa
                rotated_images.append(rotated_image)

            return np.array(rotated_images)
```

```
In [ ]: X_train_rotated = rotate_images(X_train_cnn, angle=180)

        X_train_combined = np.concatenate([X_train_cnn, X_train_rotated], axis=0)

        num_rotated_images = X_train_rotated.shape[0]
        rotated_labels = y_train_cnn[:num_rotated_images]

        y_train_combined = np.concatenate([y_train_cnn, rotated_labels], axis=0)
        X_train_combined, y_train_combined = shuffle(X_train_combined, y_train_combined,
```

```
In [ ]: """
        X_train_rotated = rotate_images(X_train_combined, angle=90)

        X_train_combined = np.concatenate([X_train_combined, X_train_rotated], axis=0)

        num_rotated_images = X_train_rotated.shape[0]
        rotated_labels = y_train_combined[:num_rotated_images]

        y_train_combined = np.concatenate([y_train_combined, rotated_labels], axis=0)
        X_train_combined, y_train_combined = shuffle(X_train_combined, y_train_combined,
        """;
```

```
In [ ]: X_train_combined.shape

Out[ ]: (22412, 124, 124, 3)
```

```
In [ ]: len(X_train_combined), len(X_test_cnn), len(X_test_cnn)/(len(X_train_combined)+

Out[ ]: (22412, 2802, 11.112873800269691)
```

Rezising the images

Optinally resize the images, this operating is ran on multiple threads to utelize full system resources

```
In [ ]: import concurrent.futures
        import numpy as np
        from skimage.transform import resize
```

```
def resize_image(image):
    # Resize Image to (224, 224)
    return resize(image, (224, 224), anti_aliasing=True, preserve_range=True).as

def process_images(image_list):
    # Using threads, resize every image and return a list with all images
    with concurrent.futures.ThreadPoolExecutor() as executor:
        return list(executor.map(resize_image, image_list))
```

In [ ]:
```
X_train_combined = process_images(X_train_combined)
X_test_cnn = process_images(X_test_cnn)
```

Save Dataset

Test/Train dataset is saved to NPZ. The damage labels are also saved used to analyze the results of our predictions.

In [ ]:
```
# Saving to NPZ
np.savez('../../data/CNN Disaster/train_data.npz', images=X_train_combined, labe
np.savez('../../data/CNN Disaster/test_data.npz', images=X_test_cnn, labels=y_te
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.utils import shuffle
from sklearn.metrics import accuracy_score
from keras.preprocessing.image import ImageDataGenerator
import keras
```

```
2024-04-29 05:53:18.749415: I tensorflow/core/util/port.cc:110] oneDNN custom ope
rations are on. You may see slightly different numerical results due to floating-
point round-off errors from different computation orders. To turn them off, set t
he environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-04-29 05:53:18.791069: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in performa
nce-critical operations.
To enable the following instructions: AVX512F AVX512_VNNI, in other operations, r
ebuild TensorFlow with the appropriate compiler flags.
```

In [ ]:
```
model = keras.models.load_model('Models/model_9929.keras')

model.summary()
```

```
2024-04-29 05:53:20.216399: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.252087: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.255870: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.261976: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.265671: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.269236: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.387315: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.388810: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.390210: I tensorflow/compiler/xla/stream_executor/cuda/cuda_g
pu_executor.cc:995] successful NUMA node read from SysFS had negative value (-1),
but there must be at least one NUMA node, so returning NUMA node zero. See more a
t https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus
-pci#L344-L355
2024-04-29 05:53:20.391538: I tensorflow/core/common_runtime/gpu/gpu_device.cc:16
39] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 20741 MB mem
ory:  -> device: 0, name: NVIDIA A10, pci bus id: 0000:08:00.0, compute capabilit
y: 8.6
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet50 (Functional)       (None, 1000)              25636712  
                                                                 
 dense_4 (Dense)             (None, 1)                 1001      
                                                                 
=================================================================
Total params: 25637713 (97.80 MB)
Trainable params: 25584593 (97.60 MB)
Non-trainable params: 53120 (207.50 KB)
_________________________________________________________________
```

In [ ]:
```
df = pd.read_hdf('../../data/CNN Disaster/test_verify_data.h5', 'df')
df
```

Out[ ]:

	image	width	height	size
0	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
1	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
2	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
3	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
4	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
...	...	...	...	...
3842	[[[9, 15, 11], [9, 14, 11], [9, 14, 11], [9, 1...	124	124	15376
3843	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
3844	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
3845	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376
3846	[[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...	124	124	15376

3847 rows × 4 columns

Resize the test image dataset

Resizing the test dataset and normalizing the values before it gets inserted into the model

In [ ]:
```
import concurrent.futures
import numpy as np
from skimage.transform import resize

def resize_image(image):
    return resize(image, (224, 224), anti_aliasing=True, preserve_range=True).as

def process_images(image_list):
    with concurrent.futures.ThreadPoolExecutor() as executor:
        return list(executor.map(resize_image, image_list))
```

```
imgs = process_images(df["image"].values)
```

In [ ]:
```
test_float_images = []
for image in imgs:
    float_image = np.array(image).astype(np.float32)
    test_float_images.append(float_image)

test_float_images = np.array(test_float_images)/255
```

Predict values and output to CSV file

In [ ]:
```
res = model.predict(test_float_images).round()
```

```
2024-04-29 05:54:50.299027: I tensorflow/compiler/xla/stream_executor/cuda/cuda_d
nn.cc:432] Loaded cuDNN version 8902
2024-04-29 05:54:50.598308: I tensorflow/compiler/xla/stream_executor/cuda/cuda_b
las.cc:606] TensorFloat-32 will be used for the matrix multiplication. This will
only be logged once.
121/121 [==============================] - 5s 32ms/step
```

In [ ]:
```
df["pred"] = res
df["pred"] = df["pred"].astype(int)
```

In [ ]:
```
df[["pred"]].to_csv("test_images_flooding-fire_predictions.csv")
```

Evaluate validation set accuracy and recall on damage level

In [ ]:
```
validation_dataset = np.load('../../data/CNN Disaster/test_data.npz', allow_pick
```

In [ ]:
```
validation_df = pd.DataFrame()
validation_df["label"] = validation_dataset["labels"]
validation_df["damage_label"] = validation_dataset["damage_labels"]
validation_df["predicted"] = model.predict(validation_dataset["images"]).round()

validation_df
```

```
88/88 [==============================] - 3s 34ms/step
```

Out[ ]:

	label	damage_label	predicted
0	1.0	0	1.0
1	1.0	0	1.0
2	0.0	0	0.0
3	0.0	0	0.0
4	1.0	0	1.0
...	...	...	...
2797	0.0	0	0.0
2798	0.0	0	0.0
2799	1.0	0	1.0
2800	1.0	3	1.0
2801	0.0	0	0.0

2802 rows × 3 columns

In [ ]:
```
socal = validation_df[validation_df["label"] == 1]
midwest = validation_df[validation_df["label"] == 0]
```

In [ ]:
```
socal_res = []
for name, group in socal.groupby("damage_label"):
    TP = sum((group['predicted'] == 1) & (group['label'] == 1))
    FN = sum((group['predicted'] != 1) & (group['label'] == 1))
    TN = sum((group['predicted'] != 1) & (group['label'] != 1))
    FP = sum((group['predicted'] == 1) & (group['label'] != 1))

    recall = TP / (TP + FN)
    accuracy = (TP + TN) / (TP + TN + FP + FN)

    socal_res.append({
        'damage_label': name,
        'recall': f"{recall:.4%}",
        'accuracy': f"{accuracy:.4%}",
        'count': len(group)
    })

print("Socal predictions")
pd.DataFrame(socal_res)
```

```
Socal predictions
```

Out[ ]:

	damage_label	recall	accuracy	count
0	0	99.7628%	99.7628%	1265
1	1	80.0000%	80.0000%	10
2	2	100.0000%	100.0000%	9
3	3	99.4413%	99.4413%	179

```
In [ ]: midwest_res = []
        for name, group in midwest.groupby("damage_label"):
            TP = sum((group['predicted'] == 0) & (group['label'] == 0))
            FN = sum((group['predicted'] != 0) & (group['label'] == 0))
            TN = sum((group['predicted'] != 0) & (group['label'] != 0))
            FP = sum((group['predicted'] == 0) & (group['label'] != 0))

            recall = TP / (TP + FN)
            accuracy = (TP + TN) / (TP + TN + FP + FN)

            midwest_res.append({
                'damage_label': name,
                'recall': f"{recall:.4%}",
                'accuracy': f"{accuracy:.4%}",
                'count': len(group)
            })

        print("Midwest predictions")
        pd.DataFrame(midwest_res)
```

```
Midwest predictions
```

Out[ ]:

	damage_label	recall	accuracy	count
0	0	99.4582%	99.4582%	1292
1	1	100.0000%	100.0000%	18
2	2	100.0000%	100.0000%	14
3	3	100.0000%	100.0000%	15

CHECKPOINT 1

Loading file

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import json
import seaborn as sns

from skimage import img_as_float, util
from skimage import color
from skimage.color import rgb2gray, rgb2hsv
from skimage.transform import rescale, downscale_local_mean
from skimage.feature import local_binary_pattern
from sklearn.utils import resample

from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.pipeline import make_pipeline
```

In [ ]:
```
# Reading from HDF5
df = pd.read_hdf('../../data/Regression/all_training.h5', 'train_df')
train_df = df
train_df
```

Out[ ]:

	disaster	image	label	height	width	size
0	hurricane-matthew	[[[58, 94, 83], [60, 97, 86], [61, 98, 86], [6...	3	65	54	10530
1	hurricane-matthew	[[[150, 177, 183], [147, 174, 182], [153, 180,...	0	67	105	21105
2	hurricane-matthew	[[[59, 81, 75], [53, 75, 69], [47, 68, 62], [5...	1	54	56	9072
3	hurricane-matthew	[[[194, 209, 205], [137, 161, 157], [99, 127, ...	0	114	124	42408
4	hurricane-matthew	[[[127, 156, 147], [134, 165, 159], [129, 162,...	2	58	51	8874
...	...	...	...	...	...	...
26530	midwest-flooding	[[[60, 94, 76], [65, 99, 82], [66, 102, 84], [...	0	29	29	2523
26531	midwest-flooding	[[[84, 122, 114], [87, 120, 116], [87, 117, 11...	0	144	39	16848
26532	midwest-flooding	[[[53, 80, 57], [55, 81, 60], [60, 86, 66], [6...	0	86	64	16512
26533	midwest-flooding	[[[39, 69, 43], [39, 69, 44], [39, 69, 44], [3...	0	116	140	48720
26534	midwest-flooding	[[[84, 104, 108], [86, 107, 110], [85, 106, 10...	0	42	83	10458

26535 rows × 6 columns

In [ ]:

```
distribution = train_df[["disaster", "size"]].groupby("disaster").count().rename
plt.title("Dataset disaster distribution")
plt.bar(x=distribution.index, height=distribution["count"])
distribution
```

Out[ ]:

	count
disaster	
hurricane-matthew	11151
midwest-flooding	7004
socal-fire	8380

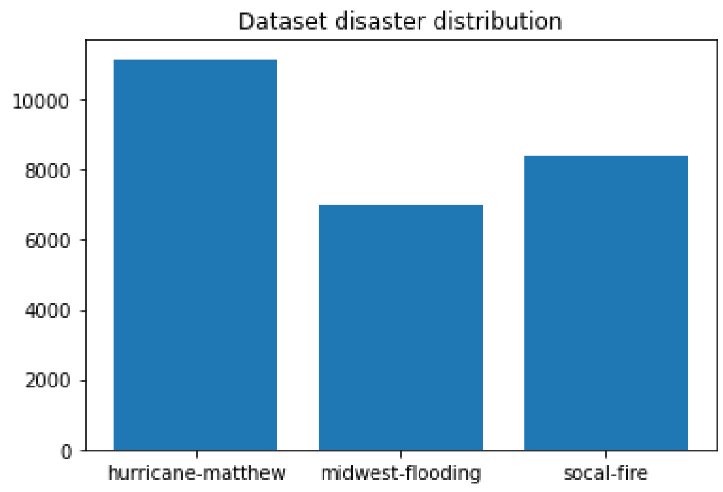
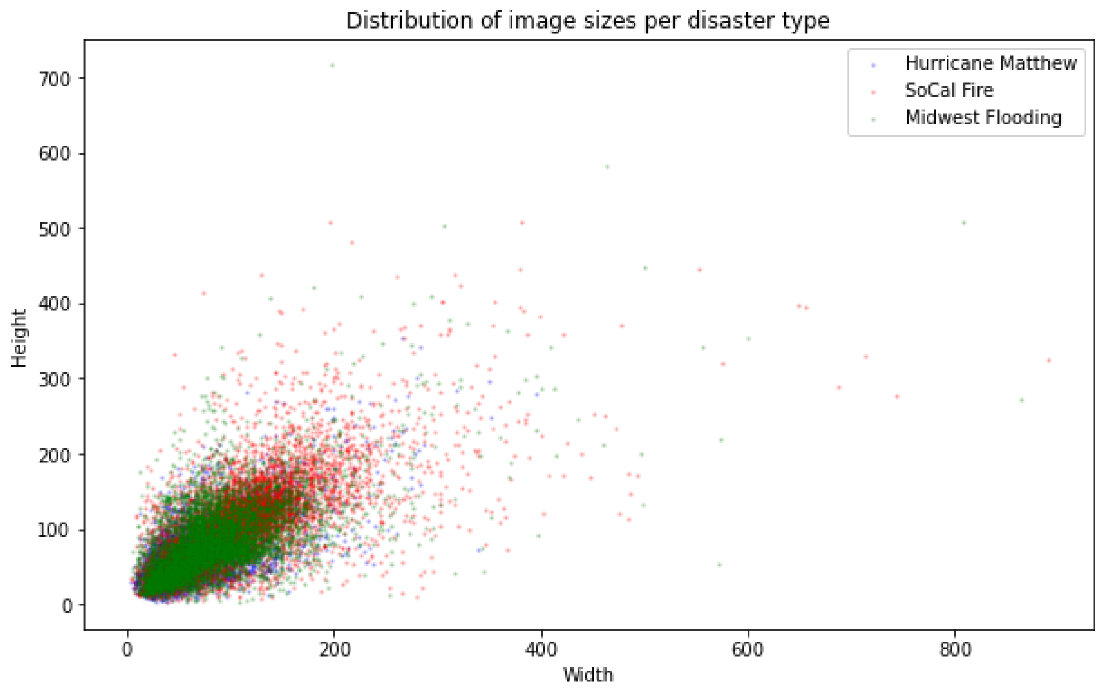


Image sizes across disasters

In [ ]:
```
hurricane_matthew = train_df[train_df['disaster'] == 'hurricane-matthew']['image
socal_fire = train_df[train_df['disaster'] == 'socal-fire']['image']
midwest_flooding = train_df[train_df['disaster'] == 'midwest-flooding']['image']

sizes_hurricane_matthew = np.array([img.shape for img in hurricane_matthew])
sizes_socal_fire = np.array([img.shape for img in socal_fire])
sizes_midwest_flooding = np.array([img.shape for img in midwest_flooding])

plt.figure(figsize=(10, 6))
plt.scatter(sizes_hurricane_matthew[:, 1], sizes_hurricane_matthew[:, 0], alpha=
plt.scatter(sizes_socal_fire[:, 1], sizes_socal_fire[:, 0], alpha=0.2, label="So
plt.scatter(sizes_midwest_flooding[:, 1], sizes_midwest_flooding[:, 0], alpha=0.

plt.title("Distribution of image sizes per disaster type")
plt.xlabel("Width")
plt.ylabel("Height")
plt.legend()
plt.show()


def plot_image_sizes(sizes, title, color):
    plt.figure(figsize=(10, 6))
    plt.scatter(sizes[:, 1], sizes[:, 0], alpha=0.4, color=color, s=1)
    plt.title("Image Sizes - " + title)
    plt.ylim(0, 700)
    plt.xlim(0, 800)
    plt.xlabel("Width")
    plt.ylabel("Height")
    plt.show()

plot_image_sizes(sizes_hurricane_matthew, "Hurricane Matthew", "blue")
plot_image_sizes(sizes_socal_fire, "SoCal Fire", "red")
plot_image_sizes(sizes_midwest_flooding, "Midwest Flooding", "green")
```


Distribution of image sizes per disaster type
Hurricane Matthew
SoCal Fire
Midwest Flooding
Height
Width
0
100
200
300
400
500
600
700
0
200
400
600
800

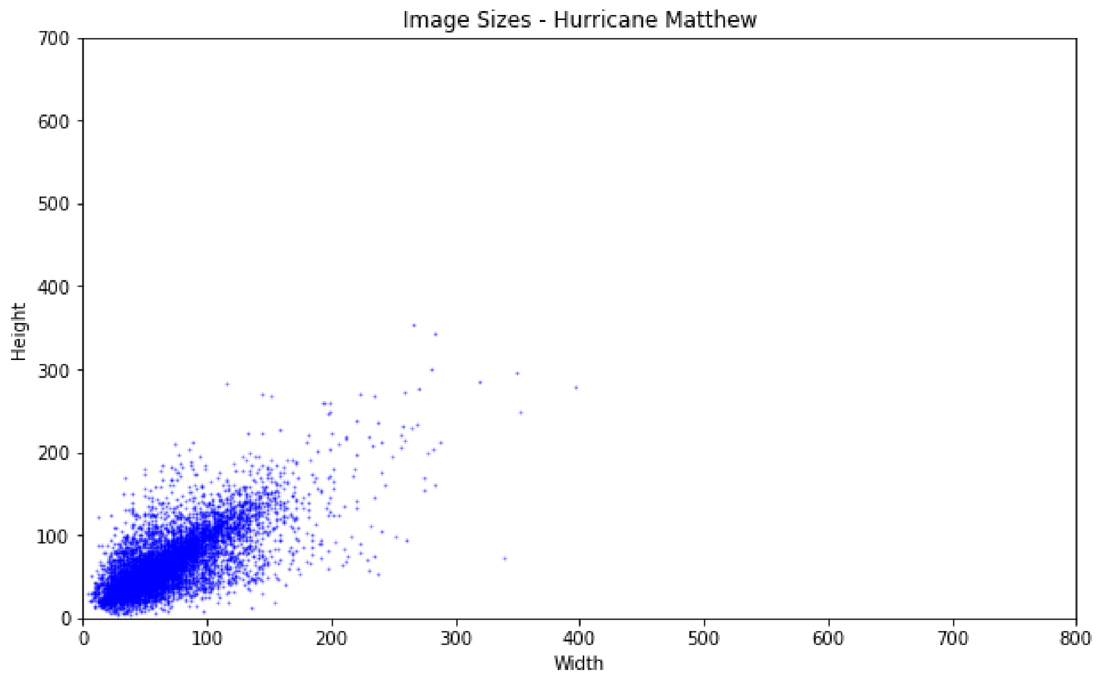
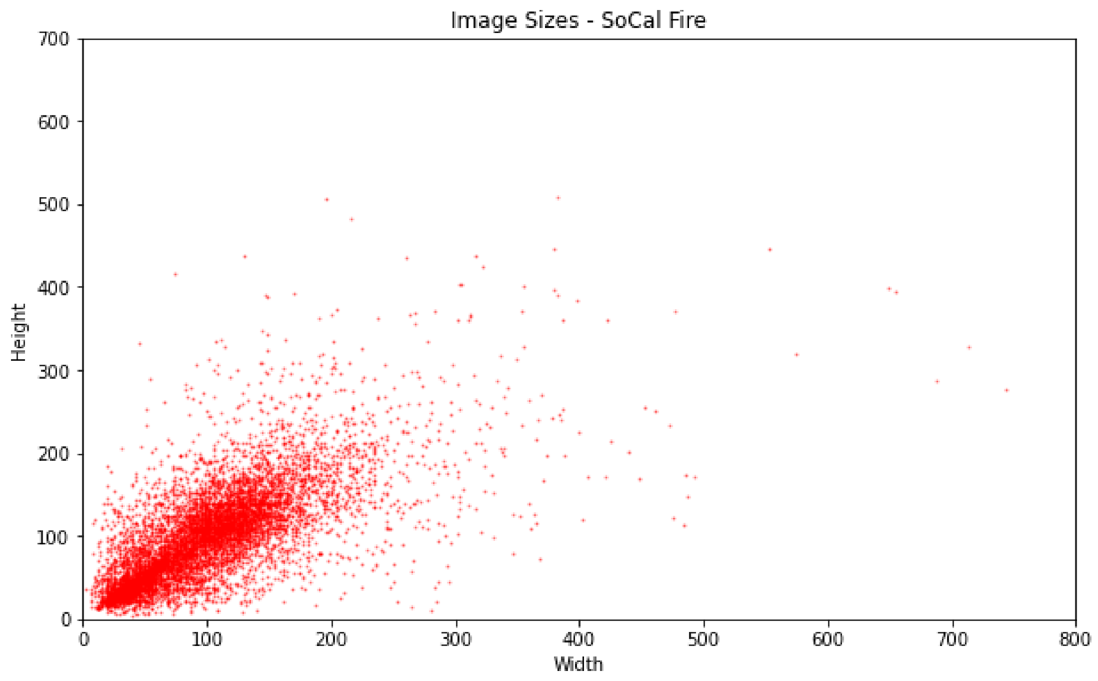
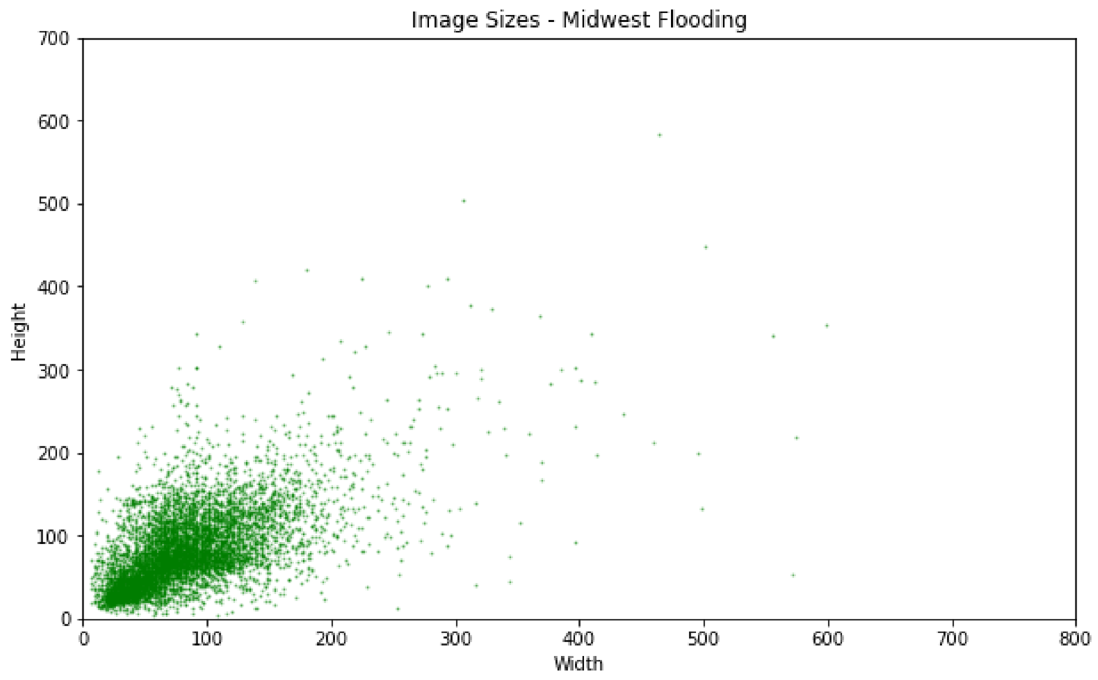


Image Sizes - Hurricane Matthew
Height
Width
0
100
200
300
400
500
600
700
0
100
200
300
400
500
600
700
800

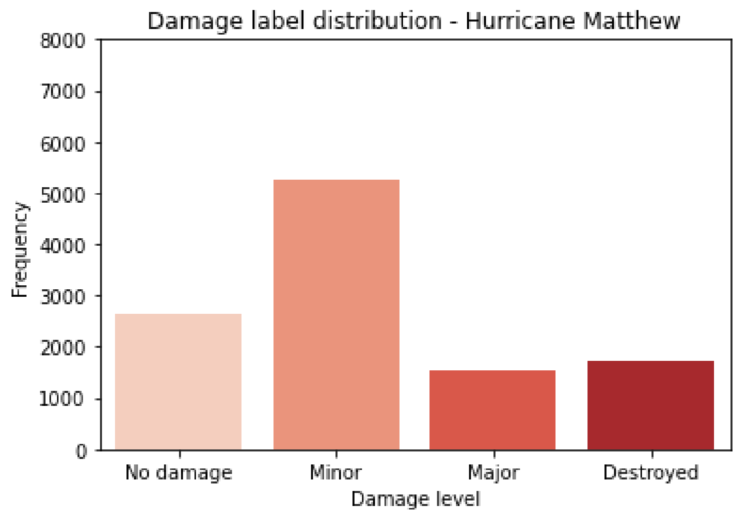


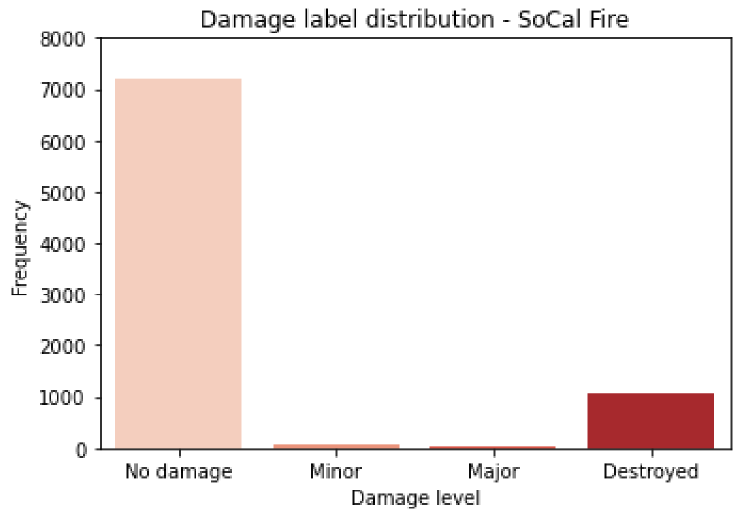


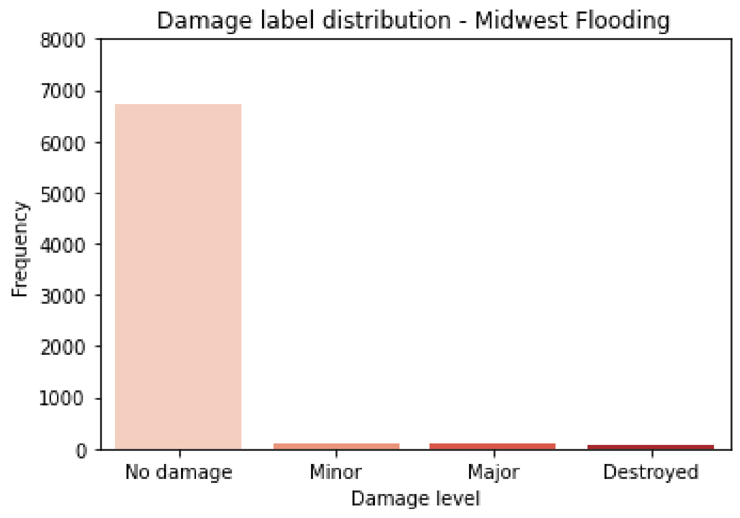
Distribution of labels

```
In [ ]: damage_labels_hurricane_matthew = np.array(train_df[train_df['disaster'] == 'hur
        damage_labels_socal_fire = np.array(train_df[train_df['disaster'] == 'socal-fire
        damage_labels_midwest_flooding = np.array(train_df[train_df['disaster'] == 'midw

        def plot_label_distribution_each_disaster(labels, title):
            plt.figure(figsize=(6, 4))
            sns.countplot(x=labels, palette="Reds")
            plt.title("Damage label distribution - " + title)
            plt.ylim(0, 8000)
            plt.xlabel('Damage level')
            plt.ylabel('Frequency')
```

```
    plt.xticks(range(4), ['No damage', 'Minor', 'Major', 'Destroyed'])
    plt.show()

plot_label_distribution_each_disaster(damage_labels_hurricane_matthew, 'Hurrican
plot_label_distribution_each_disaster(damage_labels_socal_fire, 'SoCal Fire')
plot_label_distribution_each_disaster(damage_labels_midwest_flooding, 'Midwest F
```





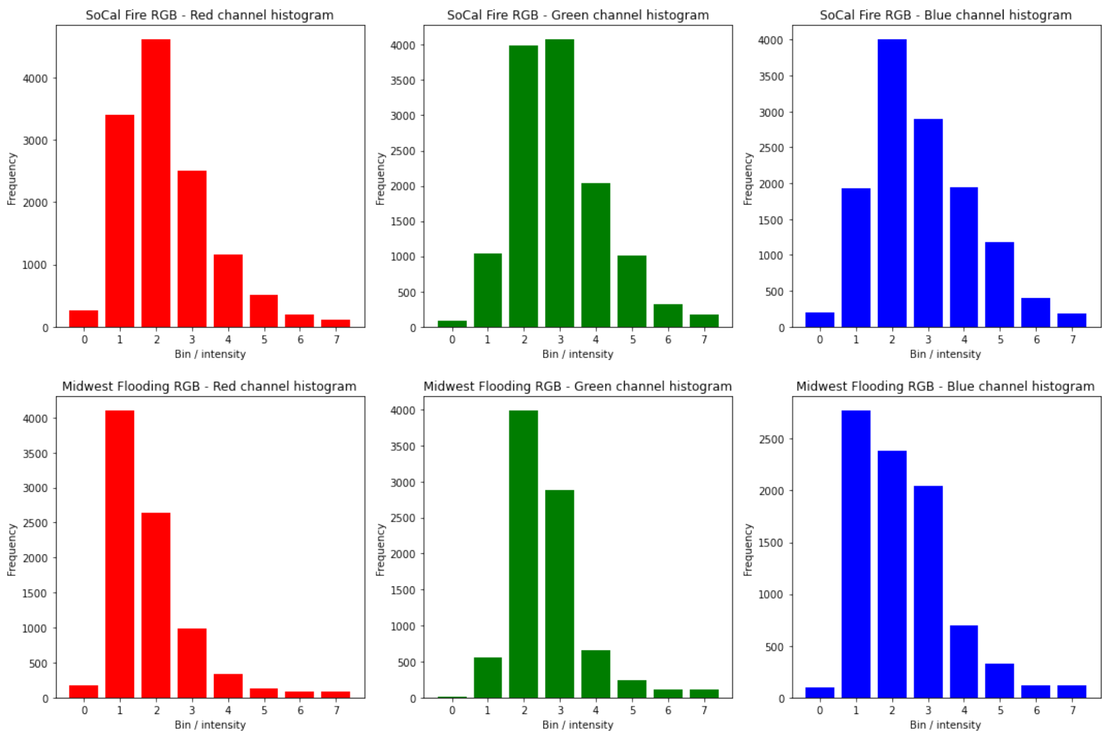


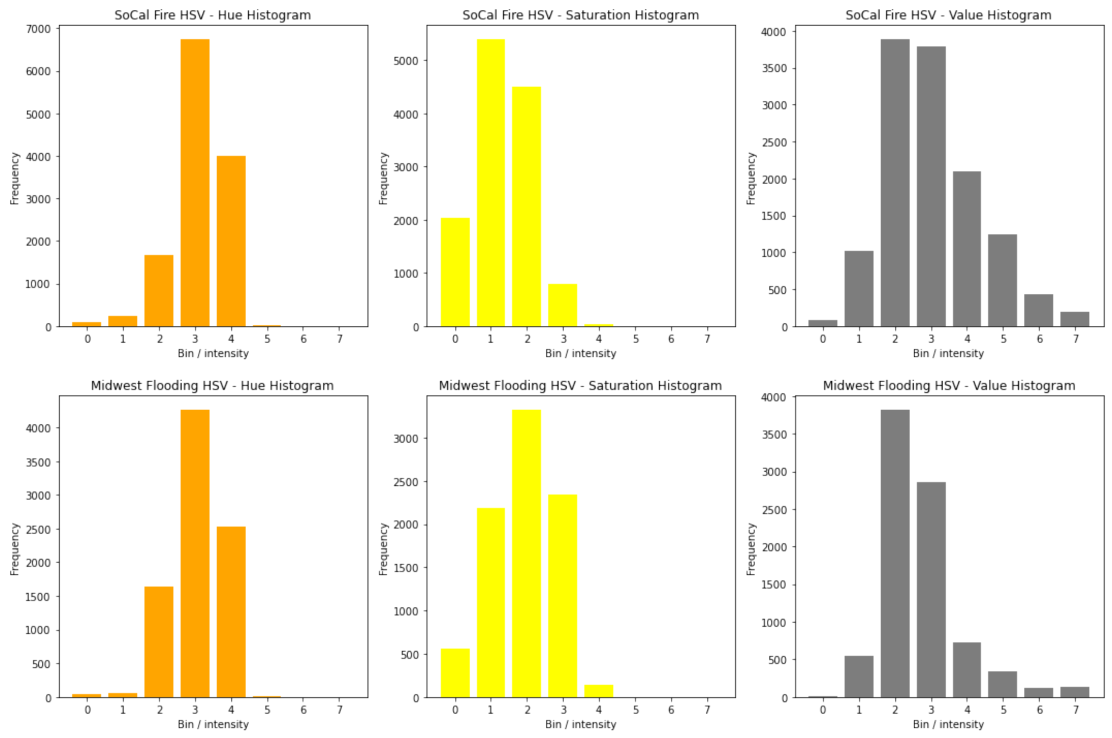
Task A - EDA and final features

RGB and HSV channels

In [ ]:
```
from skimage.color import rgb2hsv


# For this task we looked at the difference in color for SoCal Fire and Midwest

# First we defined our dataframe with only SoCal and Midwest

socal_fire_df = train_df[train_df['disaster'] == 'socal-fire']
midwest_flood_df = train_df[train_df['disaster'] == 'midwest-flooding']



# Then we plotted a histogram for every RGB channel per disaster, where each his
# Shows the average of all images for each channel

# Calculate histograms for RGB channels
def calculate_rgb_histograms(image_data, bins=8):
    histograms = {'red': [], 'green': [], 'blue': []}
    for image in image_data:
        histograms['red'].append(np.histogram(image[:, :, 0], bins=bins, range=(
        histograms['green'].append(np.histogram(image[:, :, 1], bins=bins, range
        histograms['blue'].append(np.histogram(image[:, :, 2], bins=bins, range=
    for key in histograms:
        histograms[key] = np.mean(histograms[key], axis=0)
    return histograms

# Plot the histograms
def plot_rgb_histograms(histograms, title):
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))  # Adjust the figsize if nec
    color_labels = ['Red', 'Green', 'Blue']
    colors = ['red', 'green', 'blue']

    for i, ax in enumerate(axes):
        ax.bar(range(len(histograms[color_labels[i].lower()])), histograms[color
        ax.set_title(f'{title} - {color_labels[i]} channel histogram')
        ax.set_xlabel('Bin / intensity')
        ax.set_ylabel('Frequency')

    plt.tight_layout()
    plt.show()




# Then we did the same for each channel in HSV



# Calculate histograms for HSV channels
def calculate_hsv_histograms(image_data, bins=8):
    histograms = {'hue': [], 'saturation': [], 'value': []}
    for image in image_data:
        hsv_image = rgb2hsv(image)
        histograms['hue'].append(np.histogram(hsv_image[:, :, 0], bins=bins, ran
        histograms['saturation'].append(np.histogram(hsv_image[:, :, 1], bins=bi
        histograms['value'].append(np.histogram(hsv_image[:, :, 2], bins=bins, r
    for key in histograms:
        histograms[key] = np.mean(histograms[key], axis=0)
    return histograms
```

```
# Plot the histograms
def plot_hsv_histograms(histograms, title):
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    color_labels = ['Hue', 'Saturation', 'Value']
    colors = ['orange', 'yellow', 'grey']
    for i, ax in enumerate(axes):
        ax.bar(range(len(histograms[color_labels[i].lower()])), histograms[color
        ax.set_title(f'{title} - {color_labels[i]} Histogram')
        ax.set_xlabel('Bin / intensity')
        ax.set_ylabel('Frequency')

    plt.tight_layout()
    plt.show()



# Calculate and plot histograms

# RGB
socal_fire_histograms = calculate_rgb_histograms(socal_fire_df['image'].values)
midwest_flood_histograms = calculate_rgb_histograms(midwest_flood_df['image'].va

plot_rgb_histograms(socal_fire_histograms, 'SoCal Fire RGB')
plot_rgb_histograms(midwest_flood_histograms, 'Midwest Flooding RGB')

# HSV

socal_fire_hsv_histograms = calculate_hsv_histograms(socal_fire_df['image'].valu
midwest_flood_hsv_histograms = calculate_hsv_histograms(midwest_flood_df['image'

plot_hsv_histograms(socal_fire_hsv_histograms, 'SoCal Fire HSV')
plot_hsv_histograms(midwest_flood_hsv_histograms, 'Midwest Flooding HSV')
```



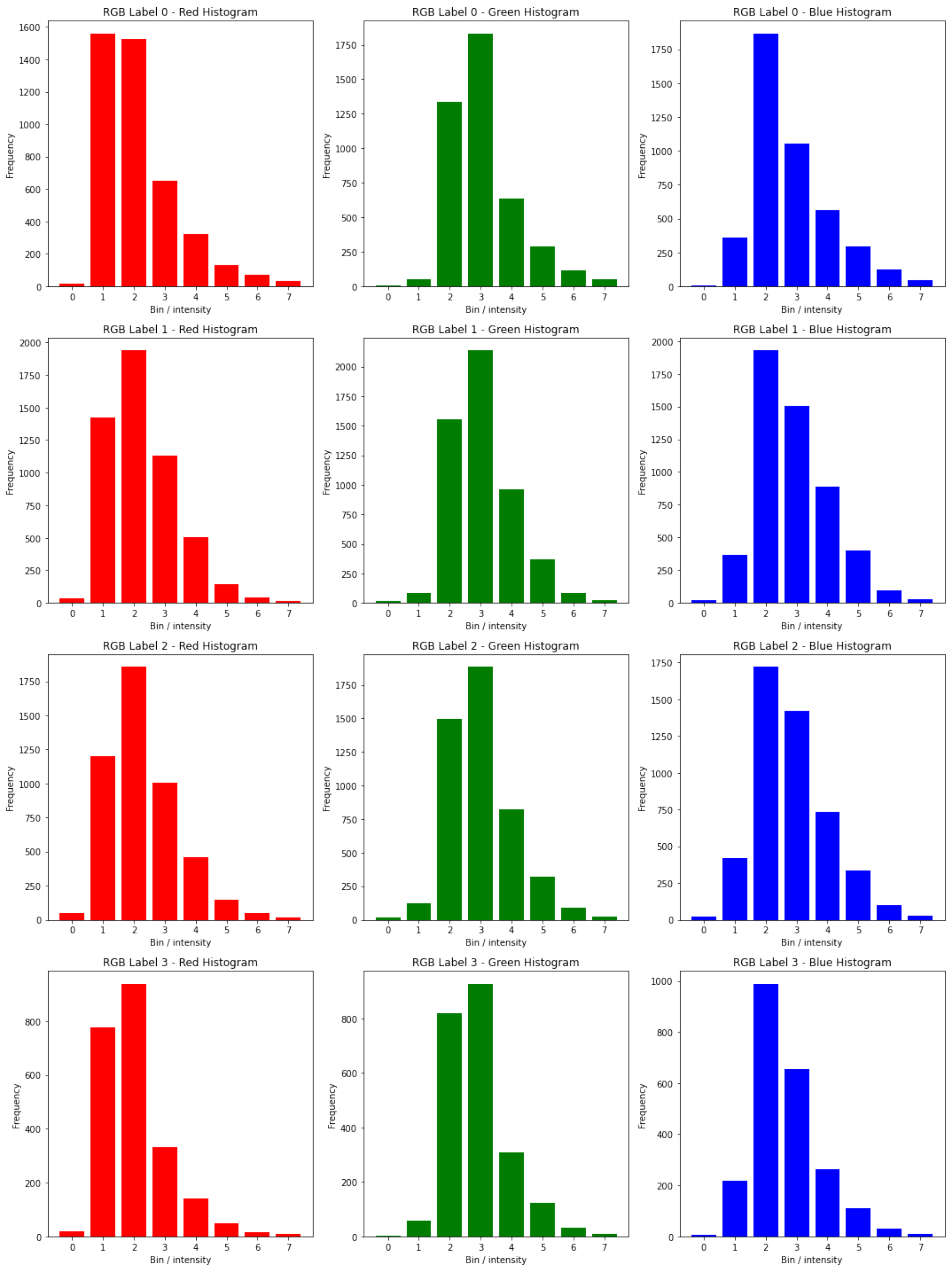


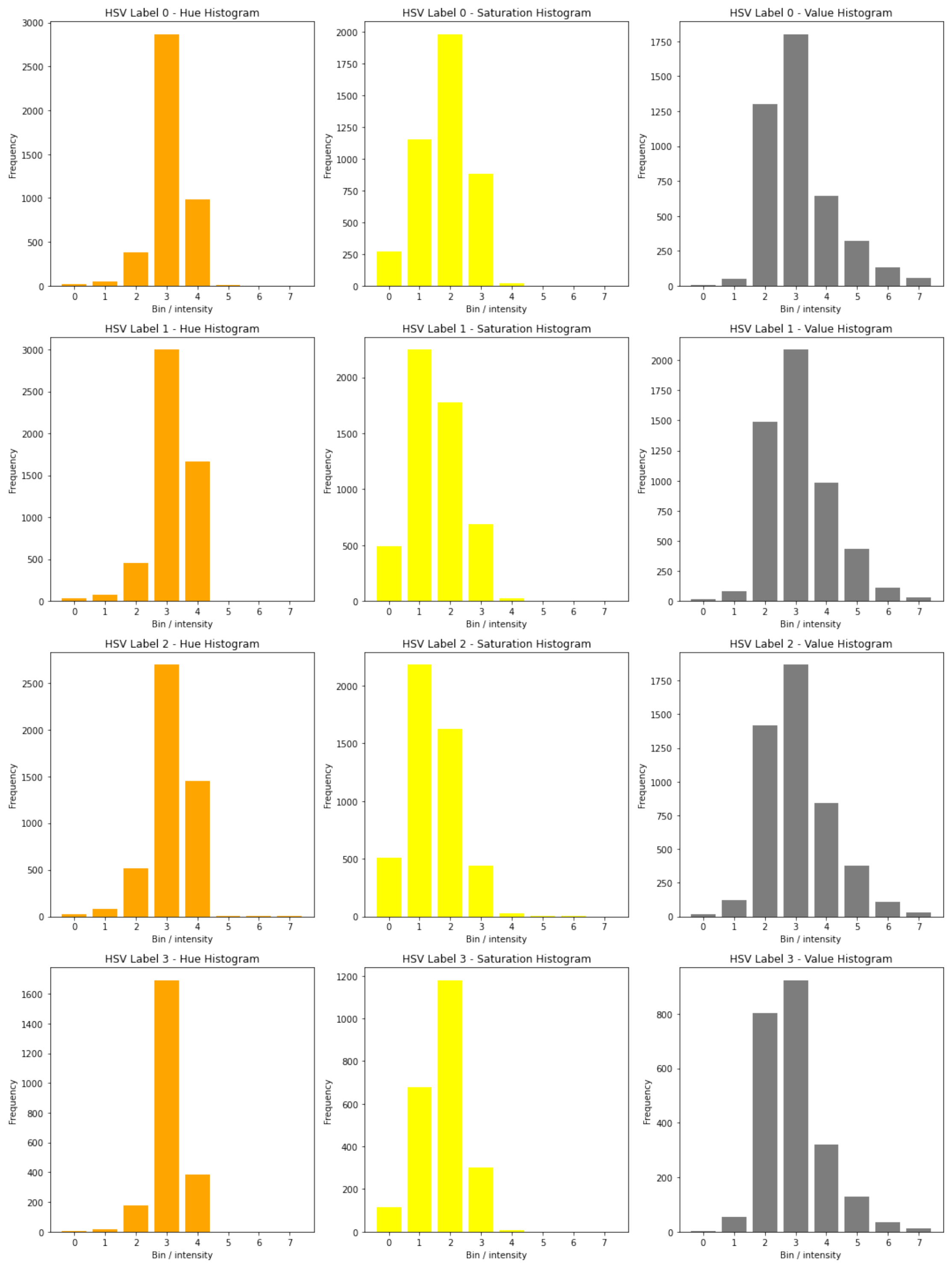
Task B - EDA and final features

RGB and HSV channels

```
In [ ]: # The first features we looked at where RGB and HSV for each label


def plot_all_labels_histograms(histograms, title_prefix):
    fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15, 20))
    if 'red' in histograms[0]:
        color_labels = ['Red', 'Green', 'Blue']
    else:
        color_labels = ['Hue', 'Saturation', 'Value']
    if 'red' in histograms[0]:
        colors = ['red', 'green', 'blue']
    else:
        colors = ['orange', 'yellow', 'grey']

    for i, label in enumerate(labels):
        for j, (key, color) in enumerate(zip(color_labels, colors)):
            axes[i, j].bar(range(len(histograms[label][key.lower()])), histogram
            axes[i, j].set_title(f'{title_prefix} {label} - {key} Histogram')
            axes[i, j].set_xlabel('Bin / intensity')
            axes[i, j].set_ylabel('Frequency')

    plt.tight_layout()
    plt.show()

# filter out labels
labels = [0, 1, 2, 3]
hurricane_data = {
    label: train_df[(train_df['disaster'] == 'hurricane-matthew') & (train_df['l
    for label in labels
```

```
}

# Calculate, store and plot histograms for each label
rgb_histograms = {label: calculate_rgb_histograms(hurricane_data[label]) for lab
hsv_histograms = {label: calculate_hsv_histograms(hurricane_data[label]) for lab

plot_all_labels_histograms(rgb_histograms, 'RGB Label')
plot_all_labels_histograms(hsv_histograms, 'HSV Label')
```

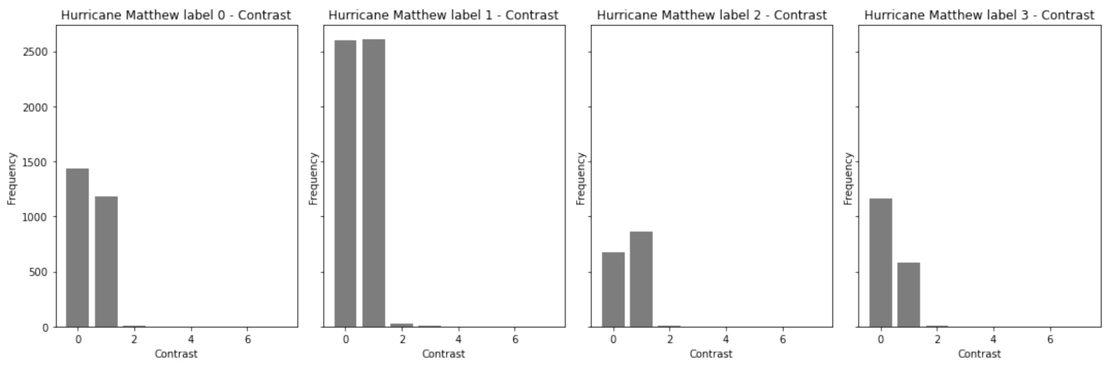




Contrast

```
In [ ]: # Then we looked at contrast to see if there were any correlation between labels

        def compute_contrast_and_bin(image, bins=5):
            image_gray = rgb2gray(img_as_float(image))
            contrast = np.std(image_gray.flatten())
            bins_edges = np.linspace(0, np.max([image_gray.max(), 0.5]), bins+1)
            bin_index = np.digitize(contrast, bins_edges) - 1
            return bin_index

        def calculate_contrast_histograms(data, bins=8):
```

```
    contrast_histograms = {label: np.zeros(bins) for label in labels}
    for label in labels:
        for image in data[label]:
            bin_index = compute_contrast_and_bin(image, bins)
            contrast_histograms[label][bin_index] += 1
    return contrast_histograms

def plot_contrast_histograms(histograms, title_prefix):
    fig, axes = plt.subplots(1, len(histograms), figsize=(15, 5), sharey=True)
    for i, (label, histogram) in enumerate(histograms.items()):
        axes[i].bar(range(len(histogram)), histogram, color='gray')
        axes[i].set_title(f'{title_prefix} {label} - Contrast')
        axes[i].set_xlabel('Contrast')
        axes[i].set_ylabel('Frequency')
    plt.tight_layout()
    plt.show()

contrast_histograms = calculate_contrast_histograms(hurricane_data)
plot_contrast_histograms(contrast_histograms, 'Hurricane Matthew label')
```



From the results of the plot, we can see that labels 2 and 3 have significant lower contrast. Label 0 is in the middle range, while label 1 is clearly higher than the other labels. The reason for this huge difference for label 1 may be caused by a larger amount of debris around the building. A large amount of debris may cause a high variation in intensity in the image (therefore higher contrast).

In [ ]:

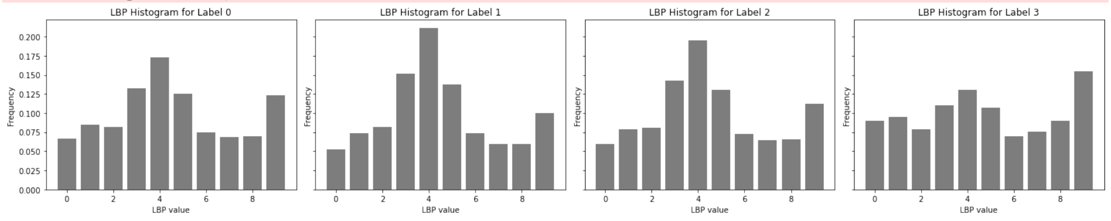
Local Binary Pattern

In [ ]:

```
# Then we looked at LBP since this methode is good for see patterns of edges whi

# LBP features
# Have to convert to grayscale since LBP only uses intensity and not color (P an
# Using uniform patterns to reduce complexity
def extract_lbp_features(img, P=8, R=1, method='uniform'):
    img_grayscaled = rgb2gray(img)
    lbp = local_binary_pattern(img_grayscaled, P, R, method)
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp, density=True, bins=n_bins, range=(0, n_bins))
    return hist

# Dividing labels and store lbp info for each label
labels = [0, 1, 2, 3]
lbp_features_by_label = {}
```

```
for label in labels:
    images = train_df[(train_df['disaster'] == 'hurricane-matthew') & (train_df[
    lbp_features = [extract_lbp_features(img) for img in images]
    lbp_features_by_label[label] = lbp_features


def plot_lbp_histograms_by_label(lbp_features_by_label):
    fig, axes = plt.subplots(nrows=1, ncols=len(lbp_features_by_label), figsize=
    for i, label in enumerate(lbp_features_by_label):
        lbp_histograms = np.vstack(lbp_features_by_label[label])
        mean_histogram = np.mean(lbp_histograms, axis=0)
        axes[i].bar(range(len(mean_histogram)), mean_histogram, color='gray')
        axes[i].set_title(f'LBP Histogram for Label {label}')
        axes[i].set_xlabel('LBP value')
        axes[i].set_ylabel('Frequency')
    plt.tight_layout()
    plt.show()

plot_lbp_histograms_by_label(lbp_features_by_label)
```

```
/home/ubuntu/.local/lib/python3.10/site-packages/skimage/feature/texture.py:360:
UserWarning: Applying `local_binary_pattern` to floating-point images may give un
expected results when small numerical differences between adjacent pixels are pre
sent. It is recommended to use this function with images of integer dtype.
  warnings.warn(
```



As we can see form the average of the LBP for each label, we can say that 1 and 2 look very similar. Also label 0 seems similar to 1 and 2. However label 3 look distinctively different from the others. This makes sense given that the pattern of edges LBP can detect is different when a building is destoyed. For example walls can have fallen and/or moved to unnatural positions compared to a undamaged house. 0, 1 and 2 are similar because the building still standing meaning no serious changes to the overall pattern of edges.

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

In [ ]:

In [ ]:

In [ ]:

Task A - Disaster Classification - Logistic Regression

Creating new dataframe, filter and shuffle

In [ ]:
```
train_df = df

filtered_df = train_df[(train_df['disaster'] == 'socal-fire') | (train_df['disas

filtered_df['disaster'] = filtered_df['disaster'].map({'midwest-flooding': 0, 's

# shuffle data
filtered_df = filtered_df.sample(frac=1, random_state=42).reset_index(drop=True)
```

Sampling

In [ ]:
```
print("------------------------------------")
print("Before sampling data")

disaster_counts = filtered_df['disaster'].value_counts()
print("\nCounts for each disaster type:\n", disaster_counts)

total_records = disaster_counts.sum()
proportions = disaster_counts / total_records
print("\nProportions for each disaster type:\n", proportions)

print("\nDamage level distribution for Midwest flooding:")
print(filtered_df[filtered_df['disaster'] == 0]['label'].value_counts(normalize=

print("\nDamage level distribution for SoCal fire:")
print(filtered_df[filtered_df['disaster'] == 1]['label'].value_counts(normalize=


def balance_data(df, n_samples=None, random_state=42):
    balanced_df = pd.DataFrame()
    for label in df['label'].unique():
        label_df = df[df['label'] == label]
        if n_samples is None:
            n_samples = len(label_df)
        resampled_df = resample(label_df, replace=True, n_samples=n_samples, ran
        balanced_df = pd.concat([balanced_df, resampled_df], axis=0)
    return balanced_df

min_samples = min(filtered_df['label'].value_counts())
```

```
balanced_train_df = balance_data(filtered_df, n_samples=min_samples)

print("-------------------------------------")
print("After sampling data")

disaster_counts = balanced_train_df['disaster'].value_counts()
print("\nCounts for each disaster type:\n", disaster_counts)

total_records = disaster_counts.sum()
proportions = disaster_counts / total_records
print("\nProportions for each disaster type:\n", proportions)

print("\nDamage level distribution for Midwest flooding:")
print(balanced_train_df[balanced_train_df['disaster'] == 0]['label'].value_count

print("\nDamage level distribution for SoCal fire:")
print(balanced_train_df[balanced_train_df['disaster'] == 1]['label'].value_count
-------------------------------------
Before sampling data

Counts for each disaster type:
 1    8380
0    7004
Name: disaster, dtype: int64

Proportions for each disaster type:
 1    0.544722
0    0.455278
Name: disaster, dtype: float64

Damage level distribution for Midwest flooding:
0    0.961451
1    0.016276
2    0.013849
3    0.008424
Name: label, dtype: float64

Damage level distribution for SoCal fire:
0    0.859666
3    0.126969
1    0.008234
2    0.005131
Name: label, dtype: float64
-------------------------------------
After sampling data

Counts for each disaster type:
 1    313
0    247
Name: disaster, dtype: int64

Proportions for each disaster type:
 1    0.558929
0    0.441071
Name: disaster, dtype: float64

Damage level distribution for Midwest flooding:
2    0.384615
1    0.307692
0    0.267206
3    0.040486
Name: label, dtype: float64

Damage level distribution for SoCal fire:
3    0.415335
0    0.236422
1    0.204473
2    0.143770
Name: label, dtype: float64
```

Making features (HSV and RGB)

In [ ]:
```
def combine_histogram_features(image, bins=8):
    image_float = img_as_float(image)
    rgb_hist_features = np.concatenate([
        np.histogram(image_float[:, :, i], bins=bins, range=(0, 1), density=True
        for i in range(3)
    ])
    hsv_image = rgb2hsv(image_float)
    hsv_hist_features = np.concatenate([
        np.histogram(hsv_image[:, :, i], bins=bins, range=(0, 1), density=True)[
        for i in range(3)
    ])

    combined_features = np.concatenate([rgb_hist_features, hsv_hist_features])

    return combined_features
```

In [ ]:
```
combined_feature_list = balanced_train_df['image'].apply(lambda img: combine_his

combined_features_df = pd.DataFrame(list(combined_feature_list))

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(balanced_train_df['disaster'])

scaler = StandardScaler()
X_scaled = scaler.fit_transform(combined_features_df)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_si

model = LogisticRegression(max_iter=1000, multi_class='multinomial', solver='lbf
model.fit(X_train, y_train)

predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
print("Disaster classification accuracy (using combined RGB and HSV features): "

# Cross valdiation
cv_scores = cross_val_score(model, X_scaled, y_encoded, cv=5, scoring='accuracy'

print("Accuracies for each fold:", cv_scores)
print("Mean cross-validation accuracy:", np.mean(cv_scores))
```

```
Disaster classification accuracy (using combined RGB and HSV features):  0.973214
2857142857
Accuracies for each fold: [0.78571429 0.92857143 0.91964286 0.96428571 0.9553571
4]
Mean cross-validation accuracy: 0.9107142857142858
```

In [ ]:
```
combined_features_df
```

Out[ ]:

	0	1	2	3	4	5	6	7	
0	0.643367	5.023469	1.357653	0.864796	0.080612	0.019388	0.010204	0.000510	0.0
1	0.022816	0.718717	3.248485	3.761854	0.248128	0.000000	0.000000	0.000000	0.0
2	0.039216	2.011765	3.709804	0.925490	0.533333	0.419608	0.313725	0.047059	0.0
3	0.313649	3.899044	2.673854	0.766479	0.337172	0.009802	0.000000	0.000000	0.0
4	0.020418	1.920174	2.724301	2.574423	0.566060	0.122943	0.065164	0.006516	0.0
...	...	...	...	...	...	...	...	...	
555	0.002658	1.134884	6.059801	0.063787	0.108970	0.531561	0.098339	0.000000	0.0
556	0.008593	0.128894	6.977444	0.317938	0.045829	0.028643	0.057286	0.435374	0.0
557	0.002658	1.134884	6.059801	0.063787	0.108970	0.531561	0.098339	0.000000	0.0
558	0.055385	1.341538	3.042051	0.957949	0.640000	0.082051	0.061538	1.819487	0.0
559	0.152542	4.451977	1.251412	0.590395	0.338983	0.446328	0.437853	0.330508	0.0

560 rows × 48 columns

Predicting and writing to csv file

In [ ]:
```python
# Here we download the test dataset, generate the same features as we did in the
test_df = pd.read_hdf('../../data/Regression/test_fire_flooding.h5', 'test_image

test_features_list = [combine_histogram_features(img) for img in test_df['image'
test_features_df = pd.DataFrame(test_features_list)

model.fit(X_train, y_train)

X_test_scaled = scaler.transform(test_features_df)
test_predictions = model.predict(X_test_scaled)

predictions_df = pd.DataFrame(test_predictions, columns=['pred'])
predictions_df.to_csv("test_images_flooding-fire_predictions.csv", index=False)
```

Task B - Damage level classification - Creating new dataframe, filter, sample and shuffle

In [ ]:
```python
train_df = df

filtered_df = train_df[(train_df['disaster'] == 'hurricane-matthew')].copy()

# shuffle data
filtered_df = filtered_df.sample(frac=1, random_state=42).reset_index(drop=True)
```

Sampling for Hurricane Matthew

```
In [ ]: print("-------------------------------------")
        print("Before sampling data")

        disaster_counts = filtered_df['label'].value_counts()
        print("\nCounts for each label type:\n", disaster_counts)

        total_records = disaster_counts.sum()
        proportions = disaster_counts / total_records
        print("\nProportions for each disaster type:\n", proportions)

        min_samples = min(filtered_df['label'].value_counts())

        balanced_train_df = balance_data(filtered_df, n_samples=min_samples)

        print("-------------------------------------")
        print("After sampling data")

        disaster_counts = balanced_train_df['label'].value_counts()
        print("\nCounts for each label type:\n", disaster_counts)

        total_records = disaster_counts.sum()
        proportions = disaster_counts / total_records
        print("\nProportions for each disaster type:\n", proportions)


        balanced_train_df = balanced_train_df.sample(frac=1, random_state=42).reset_inde
        balanced_train_df.head(20)
```

```
-------------------------------------
Before sampling data

Counts for each label type:
 1    5236
0    2631
3    1740
2    1544
Name: label, dtype: int64

Proportions for each disaster type:
 1    0.469554
0    0.235943
3    0.156040
2    0.138463
Name: label, dtype: float64
-------------------------------------
After sampling data

Counts for each label type:
 0    1544
3    1544
1    1544
2    1544
Name: label, dtype: int64

Proportions for each disaster type:
 0    0.25
3    0.25
1    0.25
2    0.25
Name: label, dtype: float64
```

Out[ ]:

	disaster	image	label	height	width	size
0	hurricane-matthew	[[[71, 111, 85], [60, 98, 72], [52, 87, 63], [...	3	54	53	8586
1	hurricane-matthew	[[[61, 90, 81], [65, 95, 85], [62, 91, 80], [6...	3	50	52	7800
2	hurricane-matthew	[[[71, 98, 94], [70, 96, 92], [77, 102, 98], [...	2	65	65	12675
3	hurricane-matthew	[[[63, 100, 80], [64, 101, 81], [62, 100, 79],...	0	55	48	7920
4	hurricane-matthew	[[[79, 90, 86], [90, 102, 94], [117, 129, 114]...	2	35	72	7560
5	hurricane-matthew	[[[122, 145, 149], [120, 143, 147], [115, 138,...	0	93	84	23436
6	hurricane-matthew	[[[62, 91, 76], [62, 91, 76], [50, 76, 62], [5...	3	30	36	3240
7	hurricane-matthew	[[[40, 73, 54], [45, 81, 60], [48, 86, 63], [5...	0	47	47	6627
8	hurricane-matthew	[[[78, 104, 94], [73, 99, 89], [69, 94, 84], [...	0	68	42	8568
9	hurricane-matthew	[[[55, 93, 69], [55, 91, 69], [54, 89, 68], [5...	0	53	47	7473
10	hurricane-matthew	[[[89, 117, 110], [102, 130, 123], [109, 138, ...	3	39	37	4329
11	hurricane-matthew	[[[133, 145, 150], [70, 85, 86], [111, 125, 13...	1	65	65	12675
12	hurricane-matthew	[[[161, 166, 148], [154, 159, 142], [144, 151,...	1	27	31	2511
13	hurricane-matthew	[[[57, 86, 76], [54, 83, 73], [57, 86, 77], [6...	3	38	42	4788
14	hurricane-matthew	[[[61, 102, 81], [59, 99, 79], [59, 98, 78], [...	0	124	201	74772
15	hurricane-matthew	[[[69, 105, 83], [69, 104, 82], [63, 97, 75], ...	3	47	43	6063
16	hurricane-matthew	[[[67, 92, 80], [67, 90, 80], [59, 82, 72], [5...	2	80	125	30000
17	hurricane-matthew	[[[138, 157, 160], [126, 145, 146], [132, 152,...	2	104	69	21528
18	hurricane-matthew	[[[61, 94, 76], [57, 91, 73], [52, 84, 67], [5...	2	108	121	39204
19	hurricane-matthew	[[[61, 102, 87], [69, 111, 95], [70, 111, 96],...	3	57	72	12312

Features - damage level classification

RGB/HSV, LBG and contrast

In [ ]:
```
def extract_lbp_features(image, P=8, R=1, method='uniform'):
    image_gray = rgb2gray(image)
    lbp = local_binary_pattern(image_gray, P, R, method)
    n_bins = int(lbp.max() + 1)
    hist, _ = np.histogram(lbp, density=True, bins=n_bins, range=(0, n_bins))
    return hist

def compute_contrast_features(image, bins=8):
    image_gray = color.rgb2gray(img_as_float(image))
    contrast = np.std(image_gray)
    contrast_normalized = contrast / image_gray.max()
    binned_contrast = np.digitize(contrast_normalized, bins=np.linspace(0, 1, bi
    return np.eye(bins)[binned_contrast - 1]

lbp_features = balanced_train_df['image'].apply(lambda img: extract_lbp_features
lbp_features_df = pd.DataFrame(lbp_features.tolist())
lbp_features_df.columns = [f'lbp_{i}' for i in range(lbp_features_df.shape[1])]

balanced_train_df['combined_features'] = balanced_train_df['image'].apply(combin
combined_features_df = pd.DataFrame(balanced_train_df['combined_features'].tolis
combined_features_df.columns = [f'combined_{i}' for i in range(combined_features

balanced_train_df['contrast_features'] = balanced_train_df['image'].apply(comput
contrast_features_df = pd.DataFrame(np.vstack(balanced_train_df['contrast_featur
contrast_features_df.columns = [f'contrast_bin_{i}' for i in range(contrast_feat


new_features_df = pd.concat([lbp_features_df, combined_features_df, contrast_fea
```

```
/home/ubuntu/.local/lib/python3.10/site-packages/skimage/feature/texture.py:360:
UserWarning: Applying `local_binary_pattern` to floating-point images may give un
expected results when small numerical differences between adjacent pixels are pre
sent. It is recommended to use this function with images of integer dtype.
  warnings.warn(
```

In [ ]:
```
new_features_df
```

Out[ ]:

	lbp_0	lbp_1	lbp_2	lbp_3	lbp_4	lbp_5	lbp_6	lbp_7	
0	0.104472	0.100978	0.080713	0.092243	0.090496	0.079665	0.068134	0.083159	0
1	0.116538	0.107308	0.074615	0.069231	0.076923	0.067692	0.055000	0.100385	0
2	0.099645	0.092308	0.081893	0.089231	0.110296	0.099172	0.073373	0.090888	0
3	0.048106	0.073485	0.079167	0.151515	0.226515	0.143182	0.077273	0.054924	0
4	0.048810	0.077778	0.095238	0.152778	0.188889	0.134127	0.080556	0.066667	0
...	...	...	...	...	...	...	...	...	
6171	0.065309	0.078652	0.077809	0.127528	0.179775	0.131180	0.083146	0.065309	0
6172	0.038302	0.060041	0.079710	0.170290	0.259058	0.151139	0.069617	0.053054	0
6173	0.100470	0.103995	0.075793	0.081669	0.097532	0.093420	0.064042	0.078731	0
6174	0.047826	0.097826	0.095652	0.141304	0.191304	0.141304	0.076087	0.050000	0
6175	0.034444	0.055556	0.074444	0.185556	0.270370	0.172593	0.067778	0.043333	0

6176 rows × 66 columns

Training the model

In [ ]:
```python
X = new_features_df.values
y = balanced_train_df['label'].values.astype(int)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)

model = LogisticRegression(max_iter=10000)

# Cross validation
cv_scores = cross_val_score(model, X_scaled, y, cv=5, scoring='f1_weighted')

print("F1 Scores for each fold:", cv_scores)
print("Mean F1 Score:", np.mean(cv_scores))
```

```
F1 Scores for each fold: [0.54988532 0.54524266 0.53438438 0.56437343 0.55145606]
Mean F1 Score: 0.5490683698122785
```

Predicting and writing to file

In [ ]:
```python
# Here we download the test dataset, generate the same features as we did in the
test_df = pd.read_hdf('../../data/Regression/test_hurricane.h5', 'test_images')

test_features_list = [np.concatenate([extract_lbp_features(img), combine_histogr
test_features_df = pd.DataFrame(test_features_list)
```

```
model.fit(X_train, y_train)

X_test_scaled = scaler.transform(test_features_df)
test_predictions = model.predict(X_test_scaled)

predictions_df = pd.DataFrame(test_predictions, columns=['pred'])
predictions_df.to_csv("./test_images_hurricane-matthew_predictions.csv", index=F
```

```
/home/ubuntu/.local/lib/python3.10/site-packages/skimage/feature/texture.py:360:
UserWarning: Applying `local_binary_pattern` to floating-point images may give un
expected results when small numerical differences between adjacent pixels are pre
sent. It is recommended to use this function with images of integer dtype.
  warnings.warn(
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: