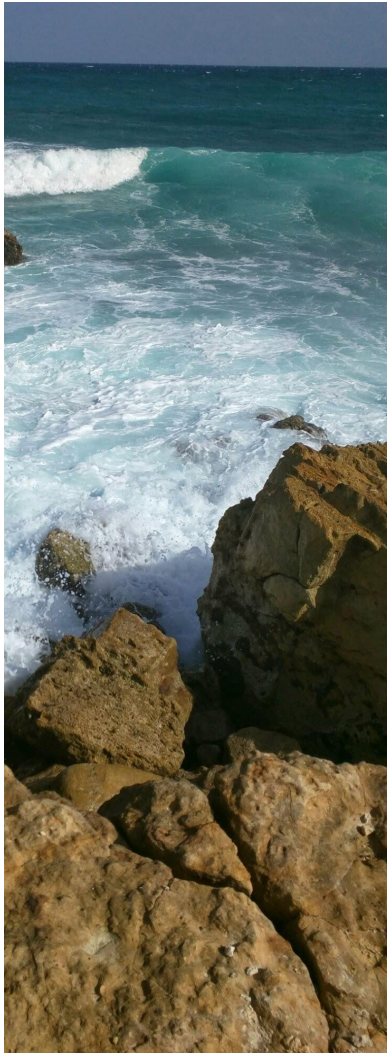


Deep Learning: Weather Image Classification

Mike, Curtis

Problem Statement/Abstract

Problem Statement

We have 11 classes of weather items in image format, we then pick 4 items to perform image classification with Convolutional Neural Networks as Modeling Project

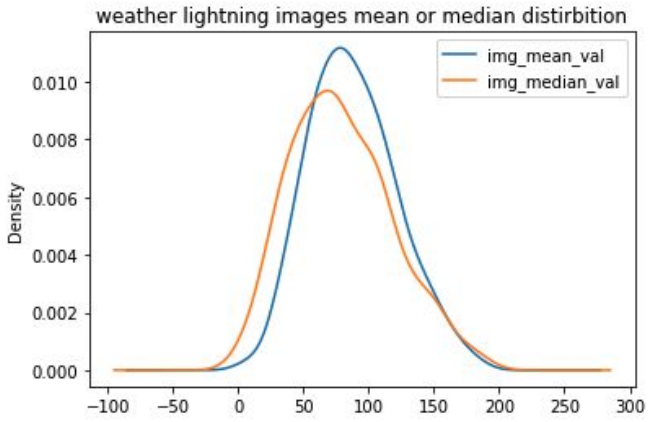
Fog vs Glaze vs Lightning vs Rainbow

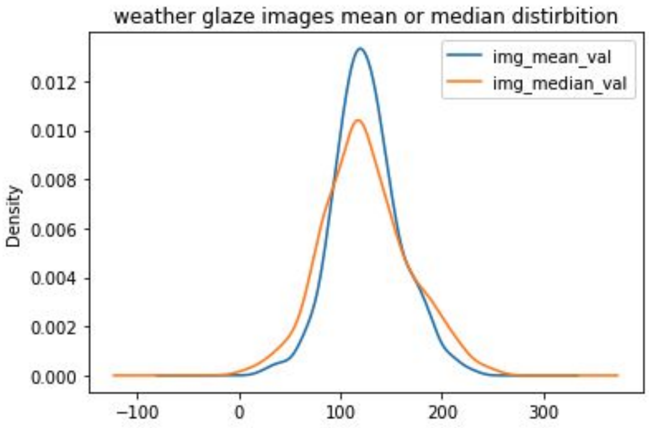
Approach Framework & EDA

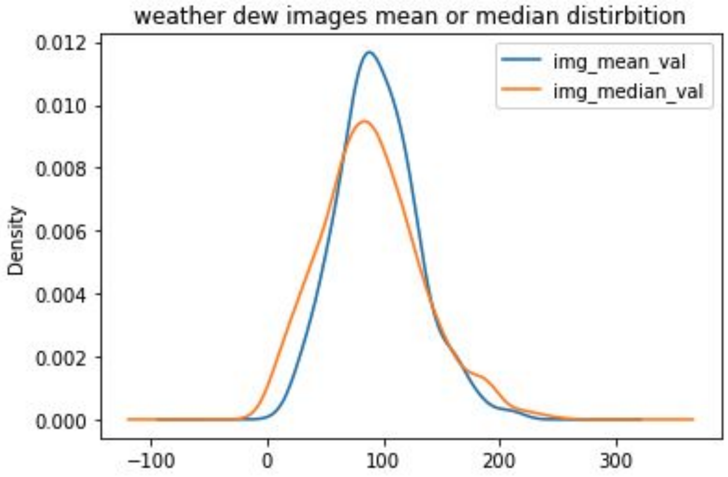
Project Framework

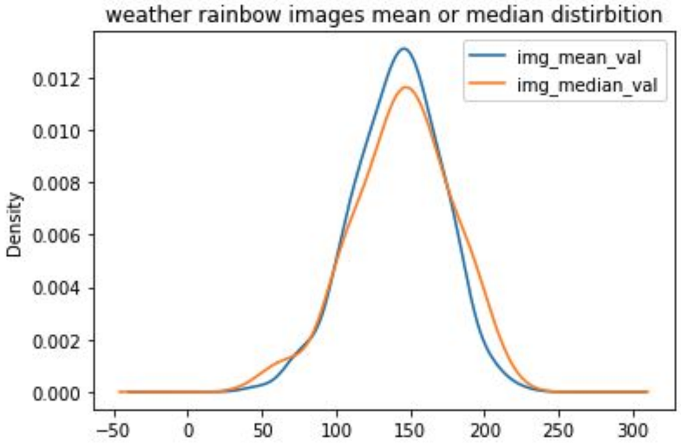
- Image Data EDA
- Image Train/Test Split : 80/20 Percentage
- Image Train/Test Images Input Generators Engineering (**from** tensorflow.keras.preprocessing.image **import** ImageDataGenerator)
- ResNet Model Direct Prediction
- CNN Baseline Model to Train/Predict
- CNN Baseline Model Validation and Over/underfitting testing
- Implement a Transfer Learning Model to Train/Predict
- Transfer Model Validation and Over/underfitting testing
- Compare CNN baseline model vs Transfer Learning Model

EDA - classified Images density Plotting

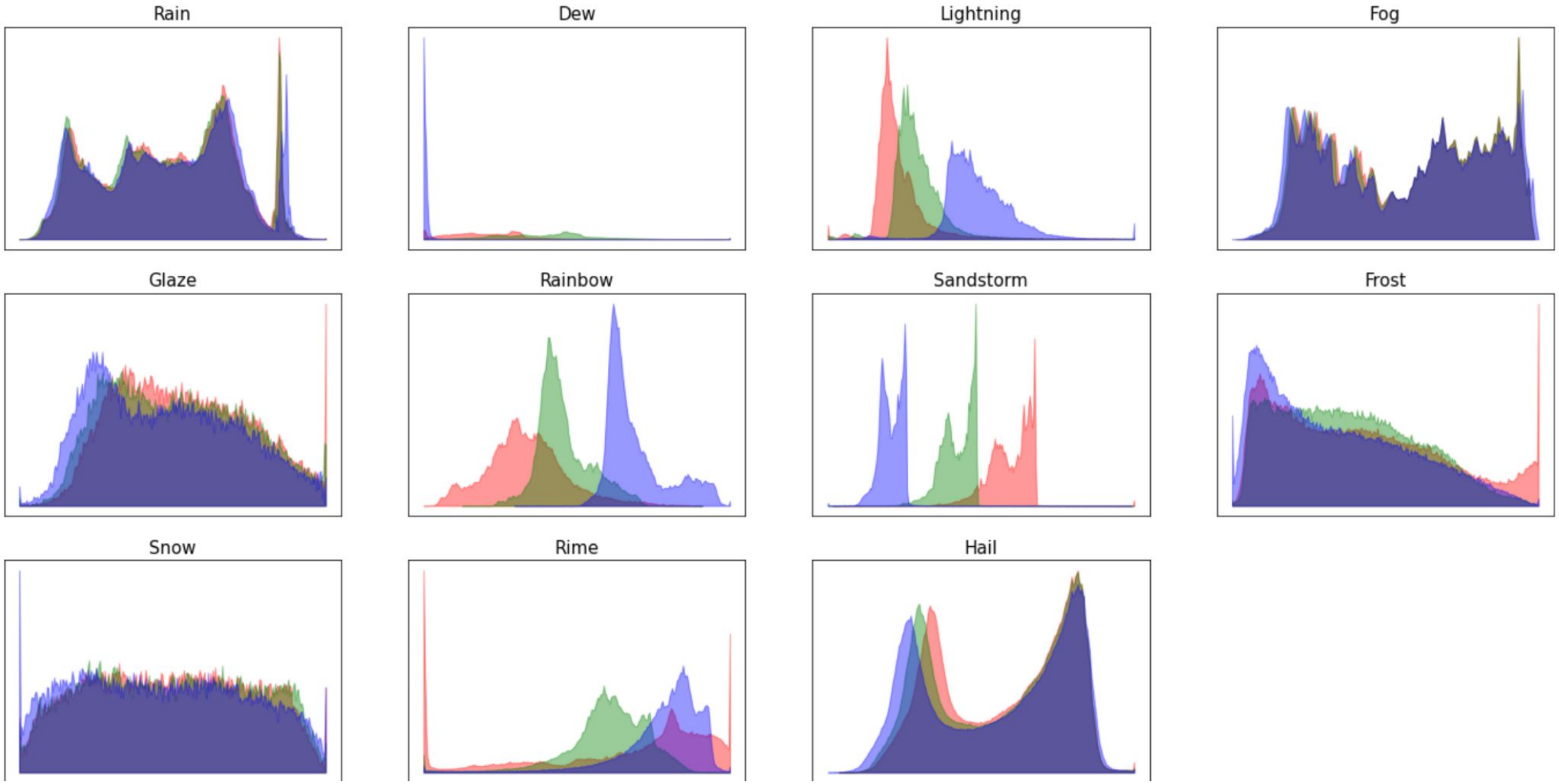








EDA - RGB distribution



Unsupervised Classification With ResNet

Apply ResNet Model Directly on Test Set of Data without training

Predicted Items from Fog testset

'toilet_tissue': 1, 'stupa': 6, 'digital_clock': 1, 'tub': 2, 'unicycle': 9, 'tricycle': 1, 'crash_helmet': 2, 'theater_curtain': 1, 'suspension_bridge': 27

Predicted Items from Lightning testset

'fountain': 53, 'geyser': 43, 'volcano': 43, 'stage': 35, 'spotlight': 33, 'spider_web': 31, 'monitor': 26, 'alp': 22, 'space_shuttle': 14, 'fireboat': 14, 'loudspeaker': 12, 'oscilloscope': 12, 'jellyfish': 11, 'garden_spider': 10

Predicted Items from Glaze testset

'coral_fungus': 37, 'hip': 37, 'cliff': 26, 'spider_web': 24, 'alp': 19, 'chainlink_fence': 18, 'fountain': 18, 'picket_fence': 17, 'pot': 12, 'hen-of-the-woods': 12, 'coral_reef': 12

Predicted Items from Rainbow testset

'fountain': 42, 'fireboat': 34, 'geyser': 27, 'lakeside': 18, 'bubble': 14, 'volcano': 13, 'spotlight': 10, 'alp': 9, 'seashore': 7, 'space_shuttle': 6, 'wing': 5, 'sandbar': 4, 'cannon': 4

Model Development

Baseline CNN Model

```
model = Sequential()
model.add(Conv2D(input_shape=(128, 128 ,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4, activation="softmax"))

from tensorflow.keras.optimizers import Adam
opt = Adam(lr=0.00001)
model.compile(optimizer=opt, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.summary()

model.save_weights("saved_models/new_initial_weights.h5")
```

VGG16 Model

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.models import Model

# load the model
vgg16 = VGG16(input_shape=(128, 128,3), include_top=False, weights='imagenet')
```

```
model_vgg16 = Sequential()
model_vgg16.add(Input(shape=(128, 128, 3)))
model_vgg16.add(vgg16)
model_vgg16.add(Flatten())
model_vgg16.add(Dropout(0.3))
model_vgg16.add(Dense(768, activation='relu'))
model_vgg16.add(Dropout(0.25))
model_vgg16.add(Dense(256, activation='relu'))
model_vgg16.add(Dropout(0.25))
model_vgg16.add(Dense(4, activation='softmax'))
model_vgg16.summary()

model_vgg16.compile( loss = 'sparse_categorical_crossentropy',
#     loss = 'categorical_crossentropy',
             optimizer =keras.optimizers.Adam(learning_rate=0.0001),
             metrics = ["accuracy"])
```

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_3 (Flatten)	(None, 8192)	0
dropout (Dropout)	(None, 8192)	0
dense_9 (Dense)	(None, 768)	6292224
dropout_1 (Dropout)	(None, 768)	0
dense_10 (Dense)	(None, 256)	196864
dropout_2 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 4)	1028

Total params: 21,204,804
Trainable params: 21,204,804
Non-trainable params: 0

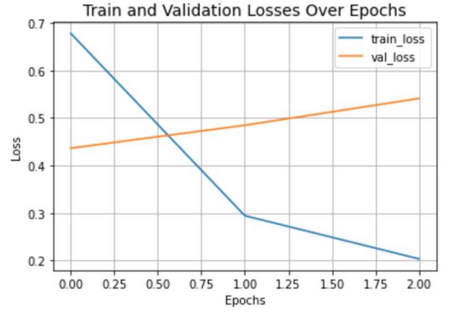
EfficientNetV2-S

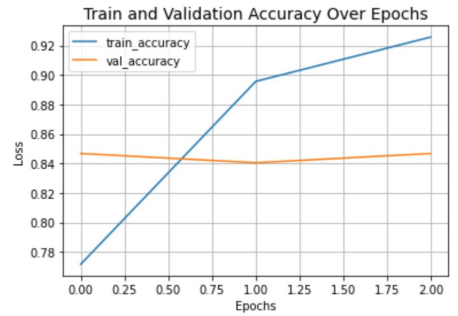
```
[61] # evaluation
     test_loss, test_acc = model_fi.evaluate(x, y, verbose=2)

     8/8 - 27s - loss: 0.5082 - accuracy: 0.8359 - 27s/epoch - 3s/step
```

```
# add classification head after Efficient net
inputs = tf.keras.Input(shape=(180, 180, 3))
x = preprocess_input(inputs)
x = model(x, training = False)
x = global_average_layer(x)
x = Flatten()(x)
x = Dense(1024, activation = 'relu')(x)
x = Dropout(0.2)(x)
x = Dense(1024, activation = 'relu')(x)
outputs = Dense(11, activation = 'softmax')(x)

model_fi = Model(inputs, outputs)
```






```
[ ] #model.compile(loss = "sparse_categorical_crossentropy", optimizer = 'Adam', metrics=["accuracy"])

    model_fi.compile(loss = "sparse_categorical_crossentropy", optimizer = 'Adam', metrics=["accuracy"])

    callbacks = [EarlyStopping(monitor = 'val_loss', min_delta = 0, patience = 2, mode = 'auto')]
```

Model Validation

Model Performance Validation

{'fog': 0, 'glaze': 1, 'lightning': 2, 'rainbow': 3}

Model	Fog	Glaze	Lightning	Rainbow	Overall	Over/Under-Fitting
Baseline CNN	Test F1: 0.81 Train F1:0.81	Test F1: 0.93 Train F1:0.93	Test F1: 0.55 Train F1:0.55	Test F1: 0.89 Train F1:0.91	Test Accuracy: 0.86 Train Accuracy: 0.87	N/A
Vgg16	Test F1: 0.97 Train F1:0.98	Test F1: 0.97 Train F1:0.98	Test F1: 0.92 Train F1:0.98	Test F1: 0.96 Train F1:0.98	Test Accuracy: 0.96 Train Accuracy: 0.98	Slight overfitted

Model Performance Validation with Testset

Model	Dew	Fog	Frost	Glaze	Hail	Lightning	Rain	Rainbow	Rime	Sandstorm	Snow	Overall
Accuracy		FP		FP FN			FP FN		FP		FN	0.84
F1	0.97	0.84	0.83	0.69	0.90	0.95	0.71	0.88	0.78	0.87	0.68	
Precision (TP/TP+FP)	0.95	0.78	0.86	0.71	0.88	1.00	0.67	0.92	0.71	0.91	1.00	
Recall (TP/TP+FN)	1.00	0.91	0.80	0.67	0.92	0.90	0.75	0.85	0.88	0.84	0.52	

'dew',

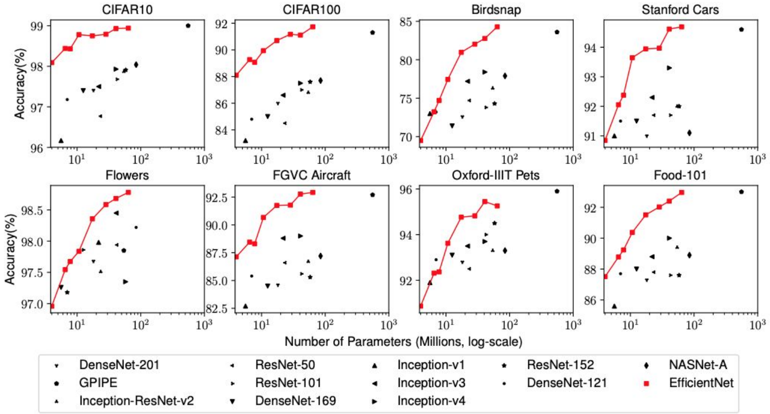
Model Investigation

Differences between models and effect

- Our ResNet predicting four weather types show great result and EfficientNet predicting 11 weather types is also descent

- If we want to further fine-tune our model, EfficientNet will be our choice as compared with other convolution models, EfficientNet is more accurate and smaller when trained on same dataset (Tan & Le, 2020)

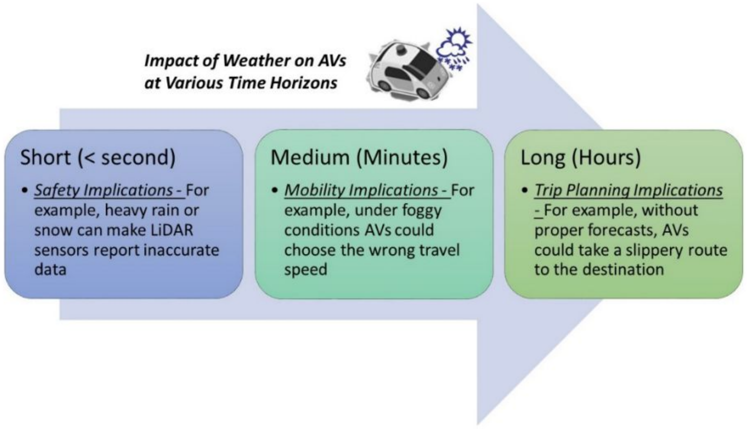
- Still a lot more samples will be needed for training



Model Further Deployment

API and Application

- Our project provides a base for many higher-level applications related to weather
- Video traps for footage shooting of wildlife
  - Capture weather condition for different timestamp
  - Helpful for behavioural study
- Weather identification for self-driving
  - Short-time decision making
  - Long-term trip planning



Source: https://rosap.ntl.bts.gov/view/dot/32494/dot_32494_DS1.pdf?

Testing our Swin-Transfomer

- Train the same data with Swin and compare with EfficientNet
- Swin has been seen efficient when train with enough data and it will be interesting to compare Swin with state-of-the-art CNN models
- Visual Transformers can be the future?

Swin Transformer

```
class SwinTransformer(layers.Layer):
    def __init__(
        self,
        dim,
        num_patch,
        num_heads,
        window_size=7,
        shift_size=0,
        num_mlp=1024,
        qkv_bias=True,
        dropout_rate=0.0,
        **kwargs,
    ):
        super(SwinTransformer, self).__init__(**kwargs)

        self.dim = dim  # number of input dimensions
        self.num_patch = num_patch  # number of embedded patches
        self.num_heads = num_heads  # number of attention heads
        self.window_size = window_size  # size of window
        self.shift_size = shift_size  # size of window shift
        self.num_mlp = num_mlp  # number of MLP nodes

        self.norm1 = layers.LayerNormalization(epsilon=1e-5)
        self.attn = WindowAttention(
            dim,
            window_size=(self.window_size, self.window_size),
            num_heads=num_heads,
            qkv_bias=qkv_bias,
            dropout_rate=dropout_rate,
        )
        self.drop_path = DropPath(dropout_rate)
        self.norm2 = layers.LayerNormalization(epsilon=1e-5)
```

References

1. https://medium.com/@mygreatlearning/what-is-vgg16-introduction-to-vgg16-f2d63849f615#:~:text=VGG16%20is%20a%20simple%20and,visual%20object%20recognition%20software%20research.
2. https://www.kaggle.com/jehanbhathena/weather-dataset
3. https://www.kaggle.com/jiongjiet/efficientnetb4-model
4. https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/
5. https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
6. https://arxiv.org/abs/1905.11946
7. https://www.tensorflow.org/tutorials/images/classification
8. https://rosap.ntl.bts.gov/view/dot/32494/dot_32494_DS1.pdf?