Image Scenario Detection Using Convolutional Neural Networks

Allen Wang
Data Science Capstone 3 Aug 2020 Cohort

The Problem

Camera apps are lackluster in adjusting picture settings for different types of scenarios. For many amateur camera users and photographers, a great way to adjust for lighting, ISO, focus, and other basic camera settings is to identify the scene of the image being taken. This is a problem that can be easily solved with computer vision. Cameras can easily adjust for their settings when the app already understands the type of scenario it’s going to be shooting and implementing computer vision into camera apps help identify scenes using feature detection in determining the type of shot that is going to be taken.

Who might care

Microsoft

SAMSUNG

Google

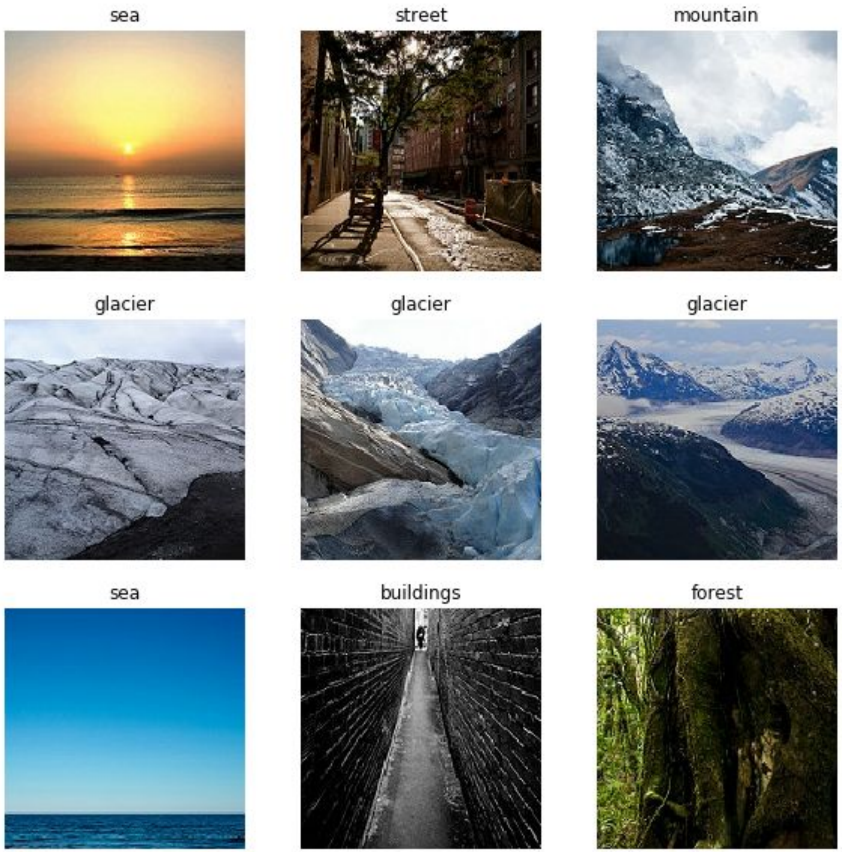
Data Acquisition

14,034 image files pre-categorized into 6 different labels in training dataset

3000 image files pre-categorized into 6 different labels in test dataset

The data consists of natural scenes around the world separated into 6 different classes: sea, street, mountains, glacier, forest, buildings. The data comes from a dataset on Kaggle.com that was published by Intel.

File format: jpeg


sea
street
mountain
glacier
glacier
glacier
sea
buildings
forest

Model 1: Standard Convolutional Neural Network

Data Preprocessing Steps:

-Keras function: tensorflow.keras.preprocessing.image_dataset_from_directory

-Used a validation split on the training data: 14034 files -> 11228 training 2286 validation

-separate into 32 batch size, and normalized RGB values by dividing by 255, and used an image size of 150x150

Model 1 Layers

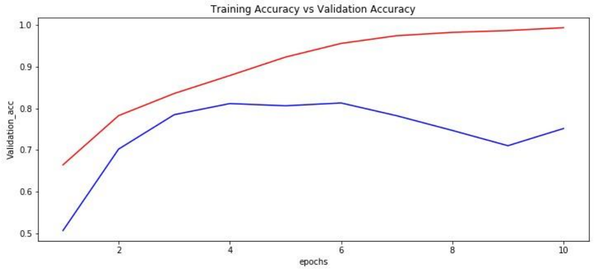
```
num_classes = 6

model = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(32, 3, activation='relu'),
  layers.Dropout(.3),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, activation='relu'),
  layers.Dropout(.2),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, activation='relu'),
  layers.Dropout(.15),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

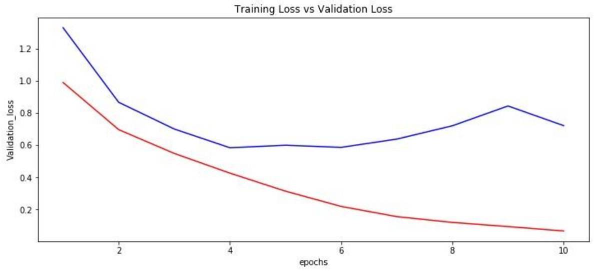
Model 1 Training and Testing Results

Evidence of overfitting after 3rd/4th epoch

Achieved a 75% accuracy on testing data

Large amount of loss






```
model.evaluate(test_ds)
```

```
94/94 [==============================] - 22s 225ms/step - loss: 0.7618 - accuracy: 0.7457
```

Model 2: ResNet50

Data Preprocessing Steps:

-Function: prepare_dataset()

```
def prepare_dataset(path,label):
    x_train=[]
    y_train=[]
    all_images_path=glob.glob(path+'/*.jpg')
    for img_path in all_images_path :
            img=load_img(img_path, target_size=(150,150))
            img=img_to_array(img)
            img=img/255.0
            x_train.append(img)
            y_train.append(label)
    return np.array(x_train),np.array(y_train)
```

-Create an trainX_label, trainY_label, testX_label, testY_label for each individual image in each label.

-Use pd.concatenate to concat all 6 labels together into features: x_train, x_test and labels: y_train, y_test

-Use a train_test_split to split the x_train and y_train into validation data

-20% validation split on the training data: 14034 files -> 11228 training 2286 validation

-Normalized RGB values by dividing by 255, and used an image size of 150x150

Model 2 Layers

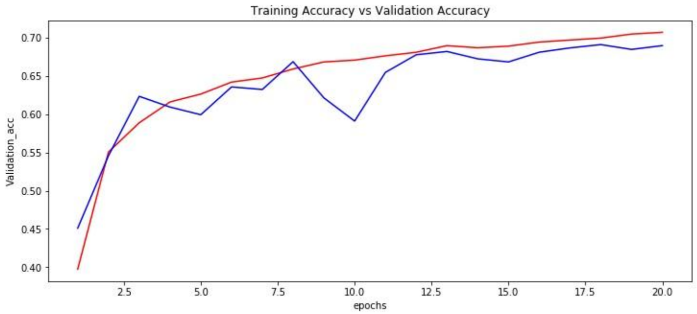
```
x = layers.Flatten()(last_output)
x = layers.Dense(1024, activation='relu')(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(6, activation='softmax')(x)


#step5
#model_resnet = Model(pretrained_model.input, x)
#step6

headModel = pretrained_model.output
headModel = MaxPooling2D(pool_size=(5, 5))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(256, activation="relu")(headModel)
headModel = Dropout(0.2)(headModel)
headModel = Dense(6, activation="softmax")(headModel)

modelresnet50 = Model(inputs=pretrained_model.input, outputs=headModel)
```

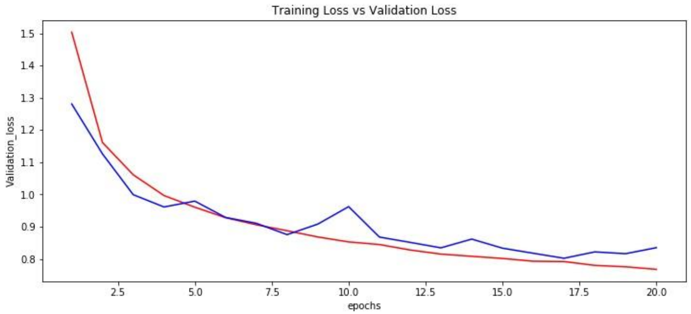
Model 2 Training and Testing Results

Consistent Fitting

Low Accuracy ~72%

Large amount of loss






```
modelresnet50.evaluate(x_test, y_test)
```

```
88/88 [==============================] - 68s 768ms/step - loss: 0.7578 - accuracy: 0.7164
```

Model 3: VGG16

Data Preprocessing Steps:

-Keras function: tensorflow.keras.preprocessing.image ImageDataGenerator & flow_from_directory

-Used a validation split on the training data: 14034 files -> 11228 training 2286 validation

-Separate into 32 batch size, and normalized RGB values by dividing by 255, and used an image size of 150x150

Model 3 Layers

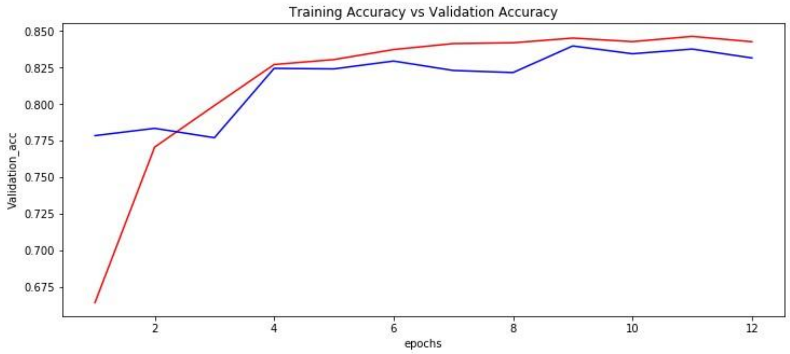
```
model = Sequential(base_modelx)
model.add(Flatten())
model.add(Dense(1024,activation ='relu'))
model.add(Dense(512,activation = 'relu'))
model.add(Dropout(0.3))
model.add(Dense(256,activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(128,activation = 'relu'))
model.add(Dropout(0.15))
model.add(Dense(6,activation='softmax'))
```

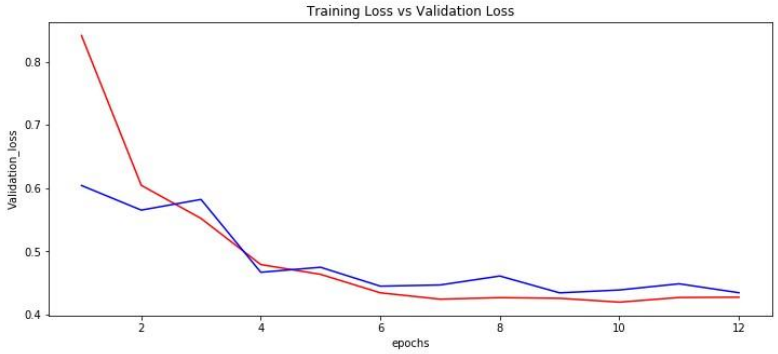
Model 3 Training and Testing

Consistent fitting

High Accuracy ~86%

Low amount of loss






```
modelvgg16.evaluate(test_datagen)
```

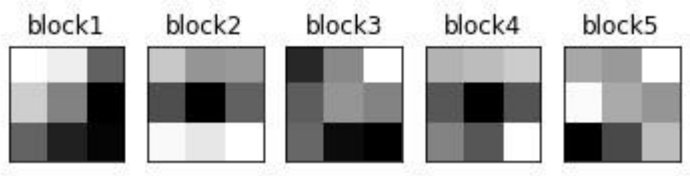
```
94/94 [==============================] - 164s 2s/step - loss: 0.3810 - accuracy: 0.8530
```

Final Model: VGG16

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              8389632
_________________________________________________________________
dense_5 (Dense)              (None, 512)               524800
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 6)                 774
=================================================================
Total params: 23,794,118
Trainable params: 9,079,430
Non-trainable params: 14,714,688
```

Model Visualization VGG16

Convolutional Layer Weights



Test Image

Convolutional Layer Feature Maps