

# Project 4 Writeup

## Instructions

- Provide an overview about how your project functions.
- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- List any extra credit implementation and result (optional).
- Use as many pages as you need, but err on the short side.
- **Please make this document anonymous.**

## Project Overview

In this project, we have designed our own CNN model for scene recognition task on 15-way scene dataset using the TensorFlow 2.0. A VGG-F pre-trained CNN model is also used for scene recognition on the same dataset, where the CNN was pretrained on ImageNet. The accuracy reported by two models are 80% and 91%, respectively, which indicates that a pretrained CNN model on another large dataset performs much better than a model from scratch on the same small scene dataset.

## Implementation Detail

### Task 1

*your\_model*: below is the *your\_model* code snippets. We adopted three convolutional layers, each layer followed by maxpooling layer to shrinking the data size and dropout layer to avoid overfitting. And the output data is then flattened and fed into two fully-connected layers, one followed by BatchNormalization and one followed by dropout. The final layer contains neurons with the number being the number of classes. The activation functions for each layer are relu except for the final layer, which is softmax to calculate the prediction probability.

```

self.optimizer = tf.keras.optimizers.Adam hp.learning_rate)

self.architecture = [
    Conv2D(filters=64, kernel_size=5, activation="relu", padding="same"),
    Dropout(rate=0.15),
    Conv2D(filters=64, kernel_size=5, activation="relu", padding="same"),
    MaxPool2D(pool_size=(3, 3), strides=2, padding="valid"),
    Dropout(rate=0.15),
    Conv2D(filters=128, kernel_size=5, activation="relu", padding="same"),
    Dropout(rate=0.15),
    Conv2D(filters=128, kernel_size=5, activation="relu", padding="same"),
    MaxPool2D(pool_size=(3, 3), strides=2, padding="valid"),
    Conv2D(filters=256, kernel_size=5, activation="relu", padding="same"),
    Dropout(rate=0.15),
    Conv2D(filters=256, kernel_size=5, activation="relu", padding="same"),
    MaxPool2D(pool_size=(3, 3), strides=2, padding="valid"),
    Dropout(rate=0.15),
    GlobalAveragePooling2D(),
    Dense(256, activation="relu"),
    BatchNormalization(),
    Dropout(rate=0.5),
    Dense(hp.num_classes, activation="softmax"),
]

### loss definition

loss = tf.keras.losses.sparse_categorical_crossentropy(labels,
                                                         predictions)

```

## Task 2

Local Interpretable Model-agnostic Explanation (LIME) can explain how the input features of a machine learning model affect its predictions. For image classification tasks, LIME finds the region of an image (set of superpixels) with the strongest association with a prediction label.

*Example 1:* The LIME plots are shown in Fig. 1. The image is falsely classified as *Tall building* while the ground truth label is *Suburb*. From the LIME plots, we could see that superpixels are mainly focused on the building and ignored the environment. That is why the model would classify this image into *Tall building* rather than *Suburb*.

*Example 2:* The LIME plots are shown in Fig. 2. The image is falsely classified as *Open Country* while the ground truth label is *Highway*. From the LIME plots, we could see

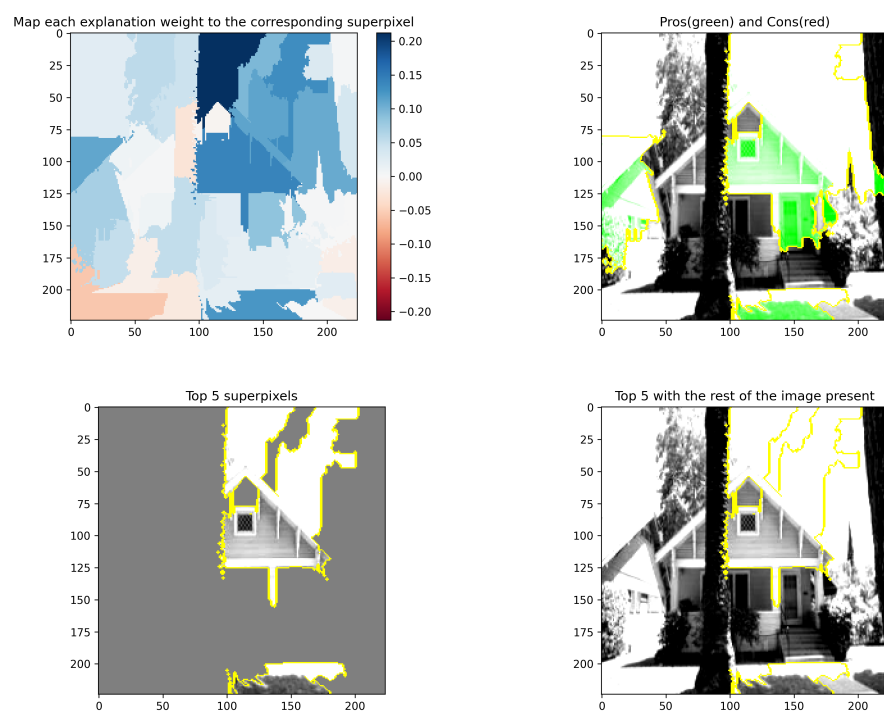


Figure 1: LIME plots for the first falsely categorized image, predicted as *Tall building* but ground truth is *Suburb*

that superpixels are mainly focused on the sky and wide ground. It ignores a large part of the highway. That is why the model would classify this image into *Open Country* rather than *Highway*.

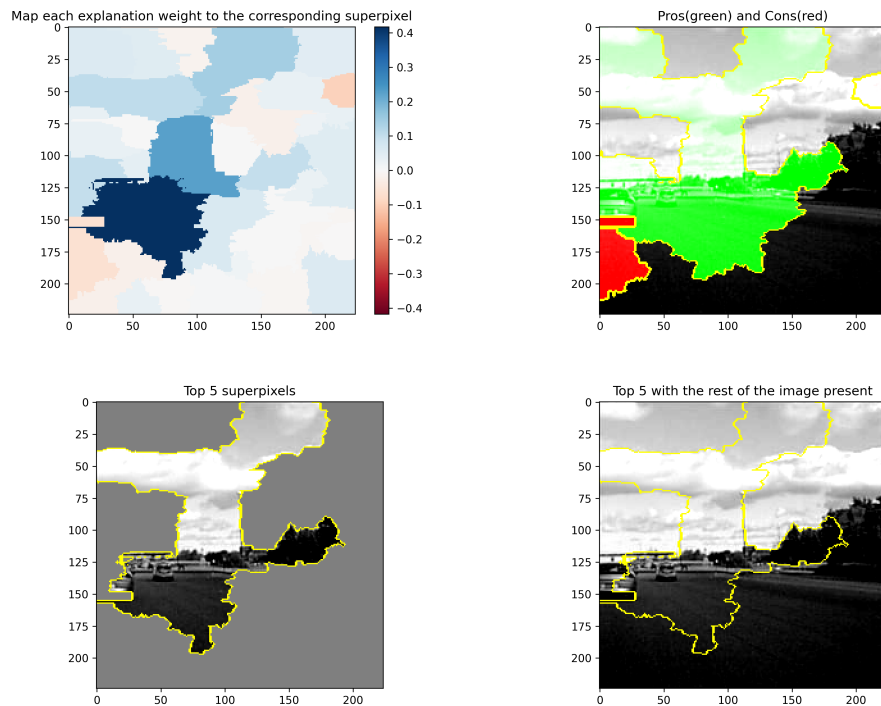


Figure 2: LIME plots for the second falsely categorized image, predicted as *Open Country* but ground truth is *Highway*

### Task 3

*Vgg\_head*: To make the model be able to classify scences, we first need to flatten the CNN layers, which is followed by two fully-connected layers with BatchNormalization and Dropout layer respectively to avoid overfitting. The final layer is a fully-connected layer with neuron number as the number of classes. The activation functions for each layer are relu except for the final layer, which is softmax to calculate the prediction probability.

```
self.optimizer = tf.keras.optimizers.Adam (hp.learning_rate)

self.head = [
    Conv2D(256, 1, 1, padding='same', activation='relu'),
    GlobalAveragePooling2D(),
    Dropout (rate=0.5),
    Dense (hp.num_classes, activation="softmax"),
]

### loss definition
```

```
loss = tf.keras.losses.sparse_categorical_crossentropy(labels,
                                                       predictions)
```

## Result

The model summary are presented in Fig. 3 and Fig. 4. The graphs of loss function over time during training are presented in Fig. 5 and Fig. 6.

After implementing the model structures described above and hyperparameters optimization, the best achieved performance of *your\_model* is 80% and VGG is 91%. The classification performance for each step of two models are plotted in Fig. 5 and Fig. 6.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	4864
dropout (Dropout)	(None, 224, 224, 64)	0
conv2d_1 (Conv2D)	(None, 224, 224, 64)	102464
max_pooling2d (MaxPooling2D)	(None, 111, 111, 64)	0
dropout_1 (Dropout)	(None, 111, 111, 64)	0
conv2d_2 (Conv2D)	(None, 111, 111, 128)	204928
dropout_2 (Dropout)	(None, 111, 111, 128)	0
conv2d_3 (Conv2D)	(None, 111, 111, 128)	409728
max_pooling2d_1 (MaxPooling2D)	(None, 55, 55, 128)	0
conv2d_4 (Conv2D)	(None, 55, 55, 256)	819456
dropout_3 (Dropout)	(None, 55, 55, 256)	0
conv2d_5 (Conv2D)	(None, 55, 55, 256)	1638656
max_pooling2d_2 (MaxPooling2D)	(None, 27, 27, 256)	0
dropout_4 (Dropout)	(None, 27, 27, 256)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
dense (Dense)	(None, 256)	65792
batch_normalization (Batch Normalization)	(None, 256)	1024
dropout_5 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 15)	3855
Total params: 3,250,767		
Trainable params: 3,250,255		
Non-trainable params: 512		

Figure 3: screenshot of *your\_model* summary

Model: "vgg_base"		
Layer (type)	Output Shape	Param #
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
Total params: 14,714,688		
Trainable params: 0		
Non-trainable params: 14,714,688		

Model: "vgg_head"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 7, 7, 256)	131328
global_average_pooling2d (GlobalAveragePooling2D)	(None, 256)	0
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 15)	3855
Total params: 135,183		
Trainable params: 135,183		
Non-trainable params: 0		

Figure 4: screenshot of VGG model summary

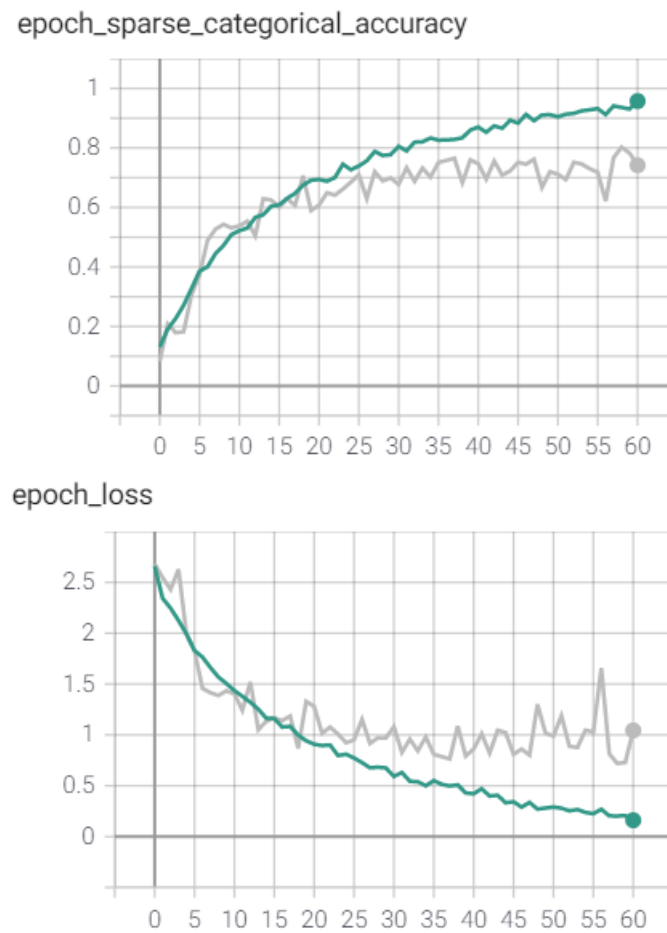


Figure 5: *Above*:classification performance for each step in your\_model. *Below*: your\_model loss function over time during training

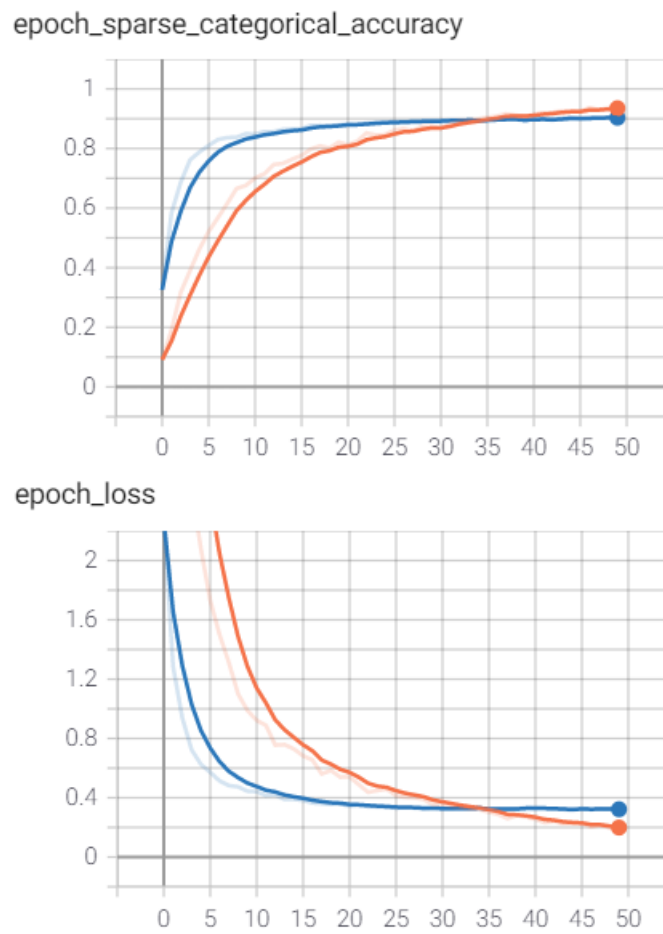


Figure 6: *Above*:classification performance for each step in VGG. *Below*: VGG loss function over time during training