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 scence recognition task on 15-way scence dataset using the TensorFlow 2.0. A VGG-F pre-trained CNN model is also used for scence recognition on the same dataset, where the CNN was pretrained on ImageNet. The accuracy reported by two models are 75% 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 scence 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
                                        "),
    MaxPool2D(pool_size=(3, 3), strides=2, padding="valid"),
    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"),
    MaxPool2D(pool_size=(3, 3), strides=2, padding="valid"),
    Dropout(rate=0.15),
    Conv2D(filters=128, 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 strongst 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 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*.

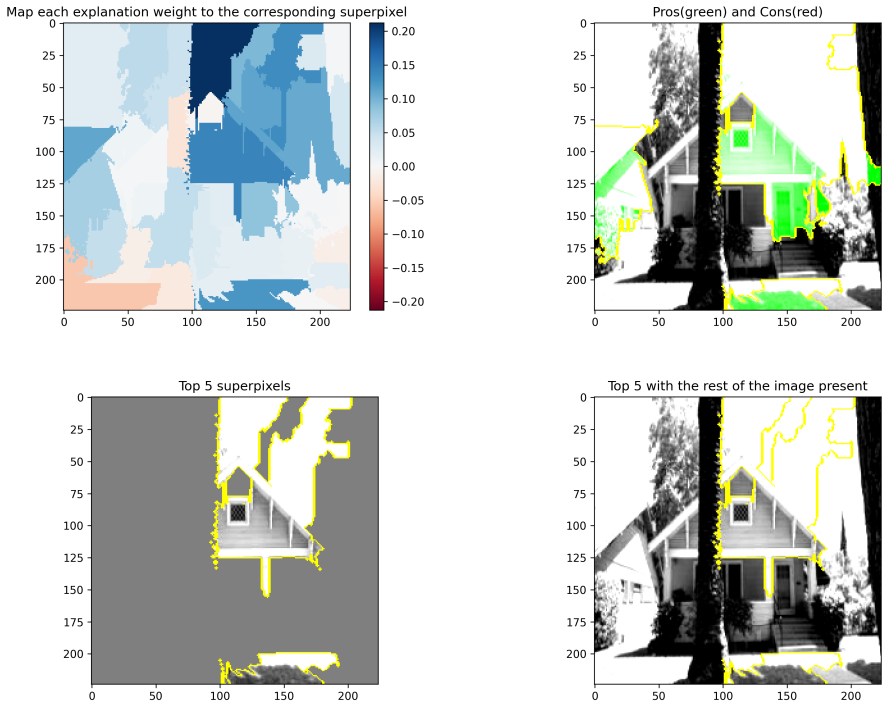


Figure 1: LIME plots for the first falsely categorized image, predicted as *Tall building* but ground truth is *Suburb*

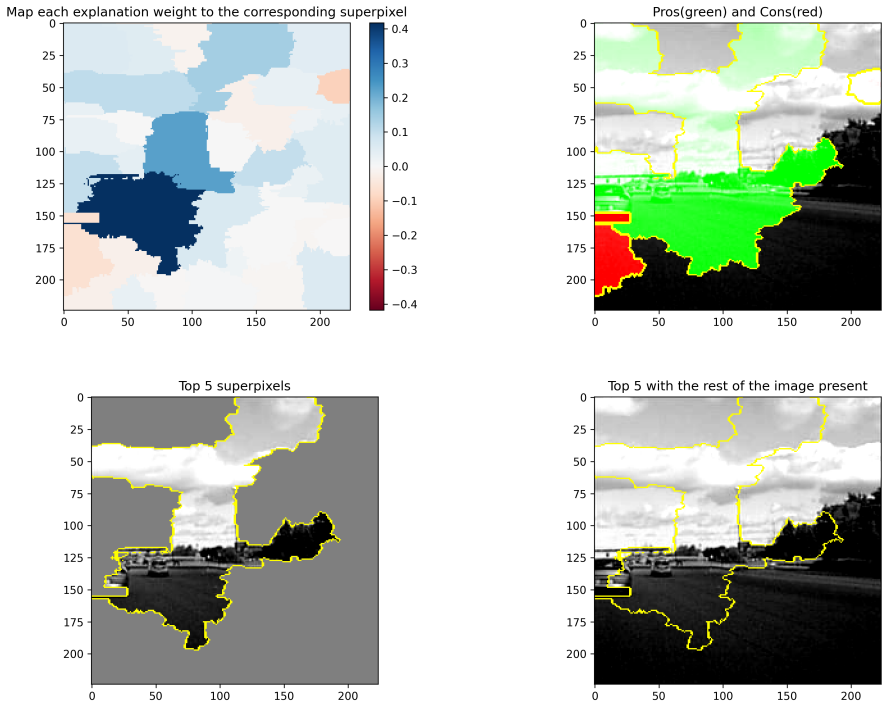


Figure 2: LIME plots for the second falsely categorized image, predicted as *Open Country* but ground truth is *Highway*

Task 3

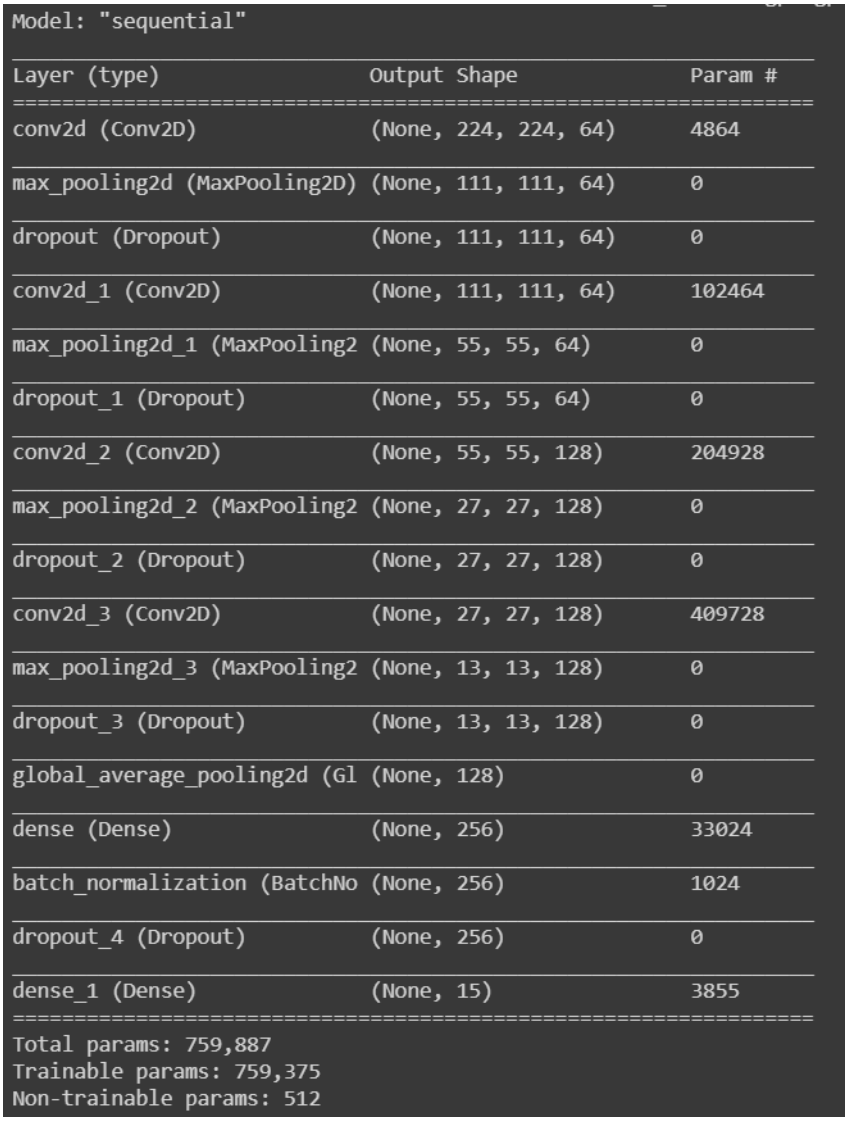
Vgg_head: To make the model be able to classify scences, we first need to flatten the CNN layers, which is followed by two fully-connected layers with BatchNormalization and Dropout layer respectively to avoid overfitting. The final layer is a fully-connected layer with neuron number as the number of 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

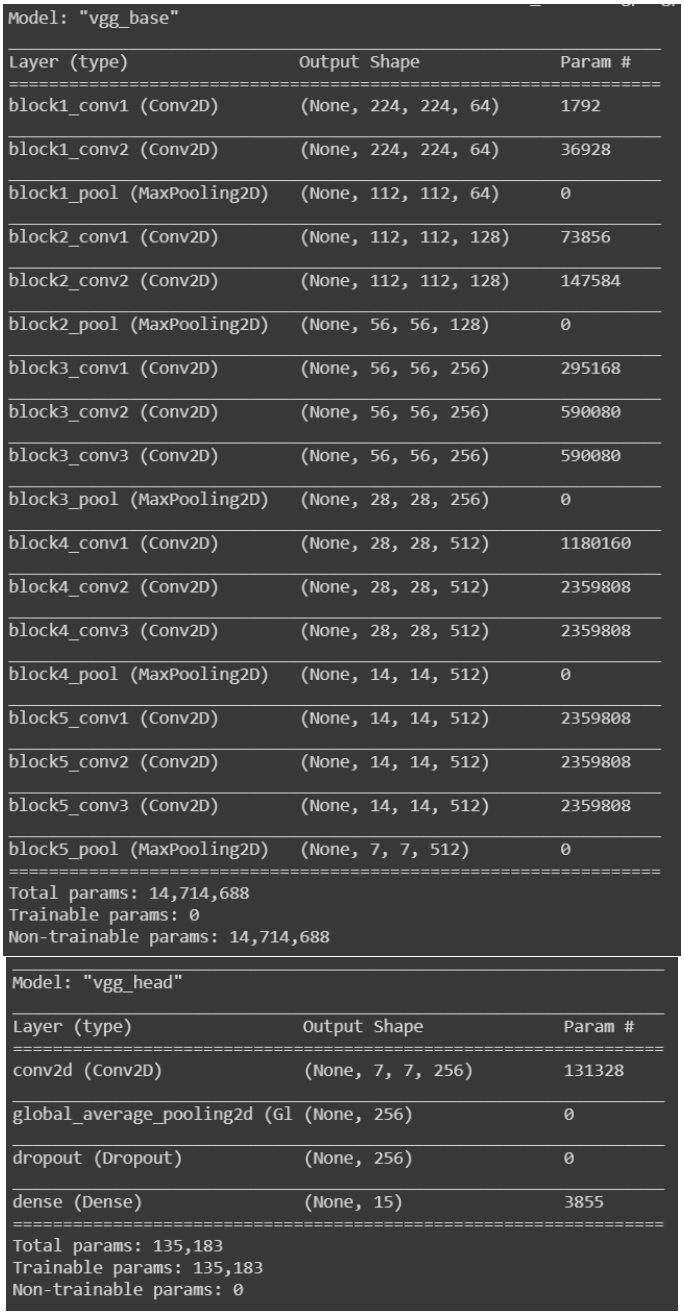
The model summary are presented in Fig. 3 and Fig. 4. The graphs of loss function over time during training are presented in Fig. 5 and Fig. 6.

After implementing the model structures described above and hyperparameters optimization, the best achieved performance of *your_model* is 75% and VGG is 91%. The classification performance for each step of two models are plotted in Fig. 5 and Fig. 6.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 64)      4864
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 64)      0
_________________________________________________________________
dropout (Dropout)            (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 111, 111, 64)      102464
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 55, 55, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 55, 55, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 55, 55, 128)       204928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 27, 27, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 27, 27, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 27, 27, 128)       409728
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 13, 128)       0
_________________________________________________________________
global_average_pooling2d (Gl (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 256)               33024
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 15)                3855
=================================================================
Total params: 759,887
Trainable params: 759,375
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 (Gl (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

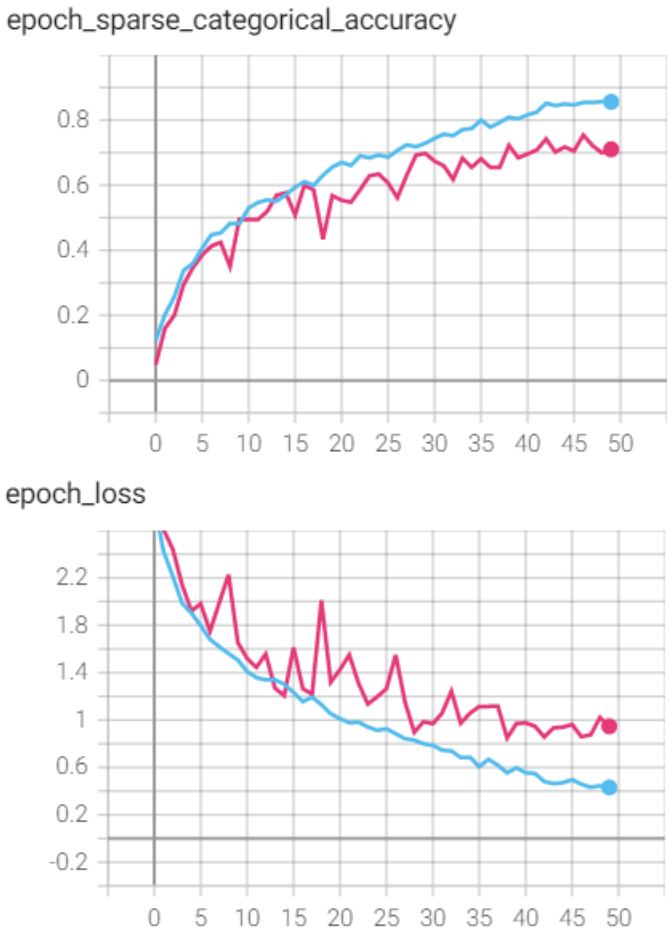


Figure 5: *Above*:classification performance for each step in your_model. *Below*: your_model loss function over time during training

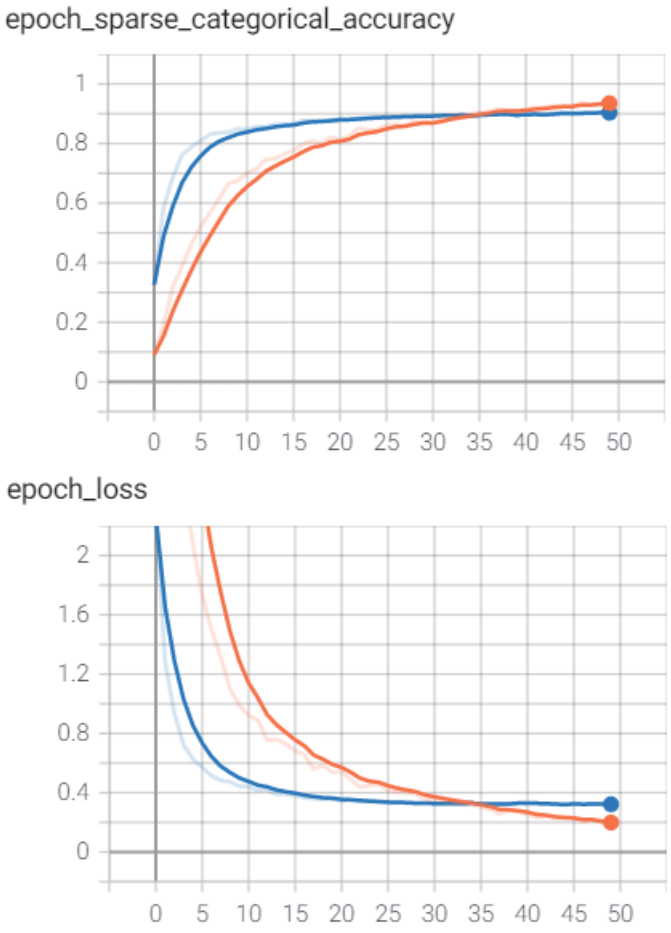


Figure 6: *Above*:classification performance for each step in VGG. *Below*: VGG loss function over time during training