

```
In [1]: from numpy.random import seed
seed(1)
import tensorflow
tensorflow.random.set_seed(2)
```

```
In [2]: #Import Stuff here

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import tensorflow.keras as keras
from tensorflow import Tensor
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, ReLU, Conv2D, MaxPooling2D, Flatten, GlobalAveragePooling2D
from tensorflow.keras.layers import DepthwiseConv2D, BatchNormalization, Add, Input, AveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
import numpy as np
from tensorflow.keras.utils import plot_model
from focal_loss import SparseCategoricalFocalLoss
from keras.models import load_model
```

```
In [3]: #creates a Image Generator from the directories and scales the images by dividing by 255. Also changes the images

def load_data(train_folder = 'data/train/'):

    datagen = ImageDataGenerator(rescale = 1/255)

    train_it = datagen.flow_from_directory (train_folder, target_size = (256, 256),
                                            class_mode = 'sparse', color_mode="rgb", batch_size=64, seed = 1)

    val_it = datagen.flow_from_directory ('data/validation/', target_size = (256, 256),
                                            class_mode = 'sparse', color_mode="rgb", batch_size=64, seed = 1)

    test_it = datagen.flow_from_directory ('data/test/', target_size = (256, 256),
                                            class_mode = 'sparse', color_mode="rgb", batch_size=64, seed = 1, shuffle=False)

    return train_it, val_it, test_it

train_it, val_it, test_it = load_data('resized/')
```

```
Found 13600 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
```

```
In [4]: batchX, batchy = train_it.next()
print('Batch shape=%s, min=%3f, max=%3f' % (batchX.shape, batchX.min(), batchX.max()))
```

```
Batch shape=(64, 256, 256, 3), min=0.000, max=1.000
```

```
In [5]: plt.imshow(batchX[0])
```

```
Out[5]: <matplotlib.image.AxesImage at 0x1a9dbdfb3d0>
```



```
In [6]: batchy[63]
```

```
Out[6]: 11.0
```

```
In [7]: def res_block(x: Tensor, filters: int, kernel_size: int, stride : int):  
    output1 = Conv2D (filters, kernel_size = kernel_size, strides = stride, padding = 'same')(x)  
    #   output1 = BatchNormalization()(output1)  
  
    output1 = Conv2D (filters, kernel_size = kernel_size, padding = 'same')(output1)  
  
    if stride == 2:  
        x = Conv2D (filters, kernel_size = kernel_size, strides = 2, padding = 'same')(x)  
  
    output_added = Add()([x, output1])  
    output_added = ReLU() (output_added)  
    output_added = BatchNormalization()(output_added)  
  
    return output_added
```

```
In [8]: def build_model():  
    # TODO: build the model,  
  
    inputs = Input(shape=(256, 256, 3, ))  
  
    x = BatchNormalization()(inputs)  
  
    x = Conv2D (64, kernel_size = 3, strides = 1, activation = 'relu', padding = 'same')(x)  
  
    x = res_block(x, filters = 64, kernel_size = 3, stride = 1)  
    x = res_block(x, filters = 64, kernel_size = 3, stride = 1)  
  
    x = res_block(x, filters = 128, kernel_size = 3, stride = 2)  
    x = res_block(x, filters = 128, kernel_size = 3, stride = 1)  
  
    x = res_block(x, filters = 256, kernel_size = 3, stride = 2)  
    x = res_block(x, filters = 256, kernel_size = 3, stride = 1)  
  
    x = res_block(x, filters = 512, kernel_size = 3, stride = 2)  
    x = res_block(x, filters = 512, kernel_size = 3, stride = 1)  
  
    x = Conv2D(128, kernel_size= 3, activation = 'relu', padding='same')(x) #added back in  
    x = BatchNormalization()(x)  
  
    x = AveragePooling2D(4)(x)  
    x = Flatten()(x)  
  
    x = Dense(64, activation = 'relu')(x)  
    my_output = Dropout (0.2)(x)  
  
    probs = Dense(17, activation = 'softmax')(my_output)
```

```

model = keras.Model(inputs= inputs, outputs=probs)

model.summary()

return model

model = build_model()

```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[None, 256, 256, 3 0 ]		[]
batch_normalization (BatchNorm alization)	(None, 256, 256, 3) 12		['input_1[0][0]']
conv2d (Conv2D)	(None, 256, 256, 64 1792 )		['batch_normalization[0][0]']
conv2d_1 (Conv2D)	(None, 256, 256, 64 36928 )		['conv2d[0][0]']
conv2d_2 (Conv2D)	(None, 256, 256, 64 36928 )		['conv2d_1[0][0]']
add (Add)	(None, 256, 256, 64 0 )		['conv2d[0][0]', 'conv2d_2[0][0]']
re_lu (ReLU)	(None, 256, 256, 64 0 )		['add[0][0]']
batch_normalization_1 (BatchNo rmalization)	(None, 256, 256, 64 256 )		['re_lu[0][0]']
conv2d_3 (Conv2D)	(None, 256, 256, 64 36928 )		['batch_normalization_1[0][0]']
conv2d_4 (Conv2D)	(None, 256, 256, 64 36928 )		['conv2d_3[0][0]']
add_1 (Add)	(None, 256, 256, 64 0 )		['batch_normalization_1[0][0]', 'conv2d_4[0][0]']
re_lu_1 (ReLU)	(None, 256, 256, 64 0 )		['add_1[0][0]']
batch_normalization_2 (BatchNo rmalization)	(None, 256, 256, 64 256 )		['re_lu_1[0][0]']
conv2d_5 (Conv2D)	(None, 128, 128, 12 73856 8)		['batch_normalization_2[0][0]']
conv2d_7 (Conv2D)	(None, 128, 128, 12 73856 8)		['batch_normalization_2[0][0]']
conv2d_6 (Conv2D)	(None, 128, 128, 12 147584 8)		['conv2d_5[0][0]']
add_2 (Add)	(None, 128, 128, 12 0 8)		['conv2d_7[0][0]', 'conv2d_6[0][0]']
re_lu_2 (ReLU)	(None, 128, 128, 12 0 8)		['add_2[0][0]']
batch_normalization_3 (BatchNo rmalization)	(None, 128, 128, 12 512 8)		['re_lu_2[0][0]']
conv2d_8 (Conv2D)	(None, 128, 128, 12 147584 8)		['batch_normalization_3[0][0]']
conv2d_9 (Conv2D)	(None, 128, 128, 12 147584 8)		['conv2d_8[0][0]']

add_3 (Add)	(None, 128, 128, 12 8)	0	['batch_normalization_3[0][0]', 'conv2d_9[0][0]']
re_lu_3 (ReLU)	(None, 128, 128, 12 8)	0	['add_3[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 128, 128, 12 8)	512	['re_lu_3[0][0]']
conv2d_10 (Conv2D)	(None, 64, 64, 256)	295168	['batch_normalization_4[0][0]']
conv2d_12 (Conv2D)	(None, 64, 64, 256)	295168	['batch_normalization_4[0][0]']
conv2d_11 (Conv2D)	(None, 64, 64, 256)	590080	['conv2d_10[0][0]']
add_4 (Add)	(None, 64, 64, 256)	0	['conv2d_12[0][0]', 'conv2d_11[0][0]']
re_lu_4 (ReLU)	(None, 64, 64, 256)	0	['add_4[0][0]']
batch_normalization_5 (BatchNormalization)	(None, 64, 64, 256)	1024	['re_lu_4[0][0]']
conv2d_13 (Conv2D)	(None, 64, 64, 256)	590080	['batch_normalization_5[0][0]']
conv2d_14 (Conv2D)	(None, 64, 64, 256)	590080	['conv2d_13[0][0]']
add_5 (Add)	(None, 64, 64, 256)	0	['batch_normalization_5[0][0]', 'conv2d_14[0][0]']
re_lu_5 (ReLU)	(None, 64, 64, 256)	0	['add_5[0][0]']
batch_normalization_6 (BatchNormalization)	(None, 64, 64, 256)	1024	['re_lu_5[0][0]']
conv2d_15 (Conv2D)	(None, 32, 32, 512)	1180160	['batch_normalization_6[0][0]']
conv2d_17 (Conv2D)	(None, 32, 32, 512)	1180160	['batch_normalization_6[0][0]']
conv2d_16 (Conv2D)	(None, 32, 32, 512)	2359808	['conv2d_15[0][0]']
add_6 (Add)	(None, 32, 32, 512)	0	['conv2d_17[0][0]', 'conv2d_16[0][0]']
re_lu_6 (ReLU)	(None, 32, 32, 512)	0	['add_6[0][0]']
batch_normalization_7 (BatchNormalization)	(None, 32, 32, 512)	2048	['re_lu_6[0][0]']
conv2d_18 (Conv2D)	(None, 32, 32, 512)	2359808	['batch_normalization_7[0][0]']
conv2d_19 (Conv2D)	(None, 32, 32, 512)	2359808	['conv2d_18[0][0]']
add_7 (Add)	(None, 32, 32, 512)	0	['batch_normalization_7[0][0]', 'conv2d_19[0][0]']
re_lu_7 (ReLU)	(None, 32, 32, 512)	0	['add_7[0][0]']
batch_normalization_8 (BatchNormalization)	(None, 32, 32, 512)	2048	['re_lu_7[0][0]']
conv2d_20 (Conv2D)	(None, 32, 32, 128)	589952	['batch_normalization_8[0][0]']
batch_normalization_9 (BatchNormalization)	(None, 32, 32, 128)	512	['conv2d_20[0][0]']
average_pooling2d (AveragePooling2D)	(None, 8, 8, 128)	0	['batch_normalization_9[0][0]']
flatten (Flatten)	(None, 8192)	0	['average_pooling2d[0][0]']
dense (Dense)	(None, 64)	524352	['flatten[0][0]']
dropout (Dropout)	(None, 64)	0	['dense[0][0]']

```
dense_1 (Dense)           (None, 17)          1105      ['dropout[0][0]']
```

```
=====
Total params: 13,663,901
Trainable params: 13,659,799
Non-trainable params: 4,102
```

In [9]:

```
def compile_model(model):
    # TODO: compile the model

    model.compile(optimizer = keras.optimizers.Adam(learning_rate = 0.0001),
                  loss=keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])

    return model

def train_model(model, train_it, val_it):
    # TODO: train the model
    callback = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights= True)

    history = model.fit(train_it, epochs = 50, steps_per_epoch = 107, verbose = 2,
                        validation_data= val_it, validation_steps = 27, callbacks=[callback])
    return model, history

model = compile_model(model)

model, history = train_model(model, train_it, val_it)

len(history.history['loss']) #
```

Epoch 1/50  
107/107 - 109s - loss: 1.6914 - accuracy: 0.4764 - val\_loss: 7.2650 - val\_accuracy: 0.0600 - 109s/epoch - 1s/step  
Epoch 2/50  
107/107 - 94s - loss: 0.9402 - accuracy: 0.7113 - val\_loss: 7.5909 - val\_accuracy: 0.0806 - 94s/epoch - 876ms/ste  
p  
Epoch 3/50  
107/107 - 94s - loss: 0.7666 - accuracy: 0.7493 - val\_loss: 3.7221 - val\_accuracy: 0.2653 - 94s/epoch - 876ms/ste  
p  
Epoch 4/50  
107/107 - 94s - loss: 0.6683 - accuracy: 0.7836 - val\_loss: 1.9318 - val\_accuracy: 0.5029 - 94s/epoch - 882ms/ste  
p  
Epoch 5/50  
107/107 - 94s - loss: 0.5556 - accuracy: 0.8179 - val\_loss: 0.9300 - val\_accuracy: 0.7153 - 94s/epoch - 878ms/ste  
p  
Epoch 6/50  
107/107 - 94s - loss: 0.5241 - accuracy: 0.8305 - val\_loss: 1.1060 - val\_accuracy: 0.7000 - 94s/epoch - 879ms/ste  
p  
Epoch 7/50  
107/107 - 94s - loss: 0.4547 - accuracy: 0.8534 - val\_loss: 2.0877 - val\_accuracy: 0.5300 - 94s/epoch - 882ms/ste  
p  
Epoch 8/50  
107/107 - 94s - loss: 0.3840 - accuracy: 0.8725 - val\_loss: 1.4776 - val\_accuracy: 0.6653 - 94s/epoch - 879ms/ste  
p  
Epoch 9/50  
107/107 - 94s - loss: 0.3584 - accuracy: 0.8842 - val\_loss: 0.7129 - val\_accuracy: 0.8159 - 94s/epoch - 882ms/ste  
p  
Epoch 10/50  
107/107 - 94s - loss: 0.3292 - accuracy: 0.8911 - val\_loss: 0.6004 - val\_accuracy: 0.8353 - 94s/epoch - 883ms/ste  
p  
Epoch 11/50  
107/107 - 94s - loss: 0.2993 - accuracy: 0.8998 - val\_loss: 0.5384 - val\_accuracy: 0.8524 - 94s/epoch - 882ms/ste  
p  
Epoch 12/50  
107/107 - 94s - loss: 0.2842 - accuracy: 0.9049 - val\_loss: 0.5989 - val\_accuracy: 0.8412 - 94s/epoch - 878ms/ste  
p  
Epoch 13/50  
107/107 - 94s - loss: 0.2391 - accuracy: 0.9184 - val\_loss: 0.9760 - val\_accuracy: 0.7794 - 94s/epoch - 882ms/ste  
p  
Epoch 14/50  
107/107 - 94s - loss: 0.2838 - accuracy: 0.9062 - val\_loss: 0.9229 - val\_accuracy: 0.7706 - 94s/epoch - 879ms/ste  
p  
Epoch 15/50

```
107/107 - 94s - loss: 0.2124 - accuracy: 0.9305 - val_loss: 0.5434 - val_accuracy: 0.8541 - 94s/epoch - 881ms/ste
p
Epoch 16/50
107/107 - 95s - loss: 0.2162 - accuracy: 0.9293 - val_loss: 0.5544 - val_accuracy: 0.8447 - 95s/epoch - 886ms/ste
p
Epoch 17/50
107/107 - 95s - loss: 0.2012 - accuracy: 0.9347 - val_loss: 0.4759 - val_accuracy: 0.8771 - 95s/epoch - 883ms/ste
p
Epoch 18/50
107/107 - 94s - loss: 0.1840 - accuracy: 0.9396 - val_loss: 0.5534 - val_accuracy: 0.8582 - 94s/epoch - 878ms/ste
p
Epoch 19/50
107/107 - 94s - loss: 0.2042 - accuracy: 0.9318 - val_loss: 0.5714 - val_accuracy: 0.8600 - 94s/epoch - 882ms/ste
p
Epoch 20/50
107/107 - 94s - loss: 0.1547 - accuracy: 0.9479 - val_loss: 0.5989 - val_accuracy: 0.8518 - 94s/epoch - 882ms/ste
p
Epoch 21/50
107/107 - 94s - loss: 0.1519 - accuracy: 0.9495 - val_loss: 0.4721 - val_accuracy: 0.8818 - 94s/epoch - 880ms/ste
p
Epoch 22/50
107/107 - 94s - loss: 0.1647 - accuracy: 0.9435 - val_loss: 0.5588 - val_accuracy: 0.8600 - 94s/epoch - 881ms/ste
p
Epoch 23/50
107/107 - 94s - loss: 0.1329 - accuracy: 0.9559 - val_loss: 0.7540 - val_accuracy: 0.7976 - 94s/epoch - 879ms/ste
p
Epoch 24/50
107/107 - 94s - loss: 0.1399 - accuracy: 0.9539 - val_loss: 0.4651 - val_accuracy: 0.8824 - 94s/epoch - 882ms/ste
p
Epoch 25/50
107/107 - 94s - loss: 0.1323 - accuracy: 0.9580 - val_loss: 0.4356 - val_accuracy: 0.8900 - 94s/epoch - 880ms/ste
p
Epoch 26/50
107/107 - 94s - loss: 0.1167 - accuracy: 0.9610 - val_loss: 0.3952 - val_accuracy: 0.9000 - 94s/epoch - 877ms/ste
p
Epoch 27/50
107/107 - 94s - loss: 0.1112 - accuracy: 0.9654 - val_loss: 0.5938 - val_accuracy: 0.8535 - 94s/epoch - 881ms/ste
p
Epoch 28/50
107/107 - 94s - loss: 0.0984 - accuracy: 0.9680 - val_loss: 0.3787 - val_accuracy: 0.9000 - 94s/epoch - 880ms/ste
p
Epoch 29/50
107/107 - 94s - loss: 0.1062 - accuracy: 0.9651 - val_loss: 0.5491 - val_accuracy: 0.8653 - 94s/epoch - 882ms/ste
p
Epoch 30/50
107/107 - 94s - loss: 0.0972 - accuracy: 0.9695 - val_loss: 0.5748 - val_accuracy: 0.8518 - 94s/epoch - 879ms/ste
p
Epoch 31/50
107/107 - 95s - loss: 0.1031 - accuracy: 0.9657 - val_loss: 0.5012 - val_accuracy: 0.8871 - 95s/epoch - 886ms/ste
p
31
```

Out[9]:

```
In [10]: def eval_model(model, test_it):
    # TODO: evaluate the model

    test_loss, test_accuracy = model.evaluate (test_it, steps = 27)

    return test_loss, test_accuracy

test_loss, test_accuracy = eval_model(model, test_it)
```

27/27 [=====] - 21s 793ms/step - loss: 0.3405 - accuracy: 0.9012

```
In [11]: test_it.reset()
preds = model.predict(test_it, steps = 27)
```

```
In [12]: y_predict = np.argmax(preds, axis=1)
```

```
In [13]: print(classification_report(test_it.classes, y_predict, target_names=test_it.class_indices))
```

	precision	recall	f1-score	support
1977	0.93	0.99	0.96	100
Amaro	0.83	0.91	0.87	100
Brannan	0.96	0.98	0.97	100
Clarendon	0.79	0.82	0.80	100
Gingham	0.84	0.97	0.90	100
He-Fe	0.89	0.92	0.91	100
Hudson	0.97	0.96	0.96	100
Lo-Fi	0.77	0.75	0.76	100
Mayfair	0.85	0.95	0.90	100
Nashville	0.98	0.99	0.99	100
Original	0.82	0.58	0.68	100
Perpetua	0.89	0.91	0.90	100
Sutro	0.96	0.96	0.96	100
Toaster	0.96	0.95	0.95	100
Valencia	0.92	0.82	0.87	100
Willow	1.00	0.92	0.96	100
X-ProII	0.97	0.94	0.95	100
accuracy			0.90	1700
macro avg	0.90	0.90	0.90	1700
weighted avg	0.90	0.90	0.90	1700

In [18]:

```
error_count = 0

for x in range (0,len(y_predict)):
    if y_predict[x] != test_it.classes[x]:
        error_count += 1

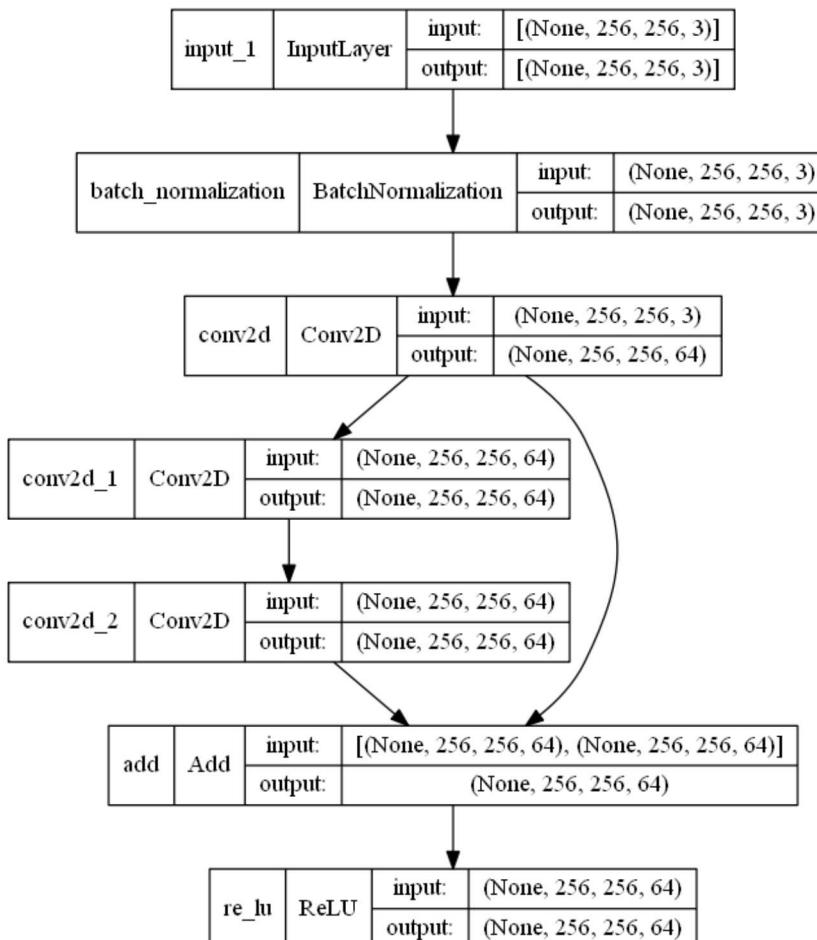
error_count
```

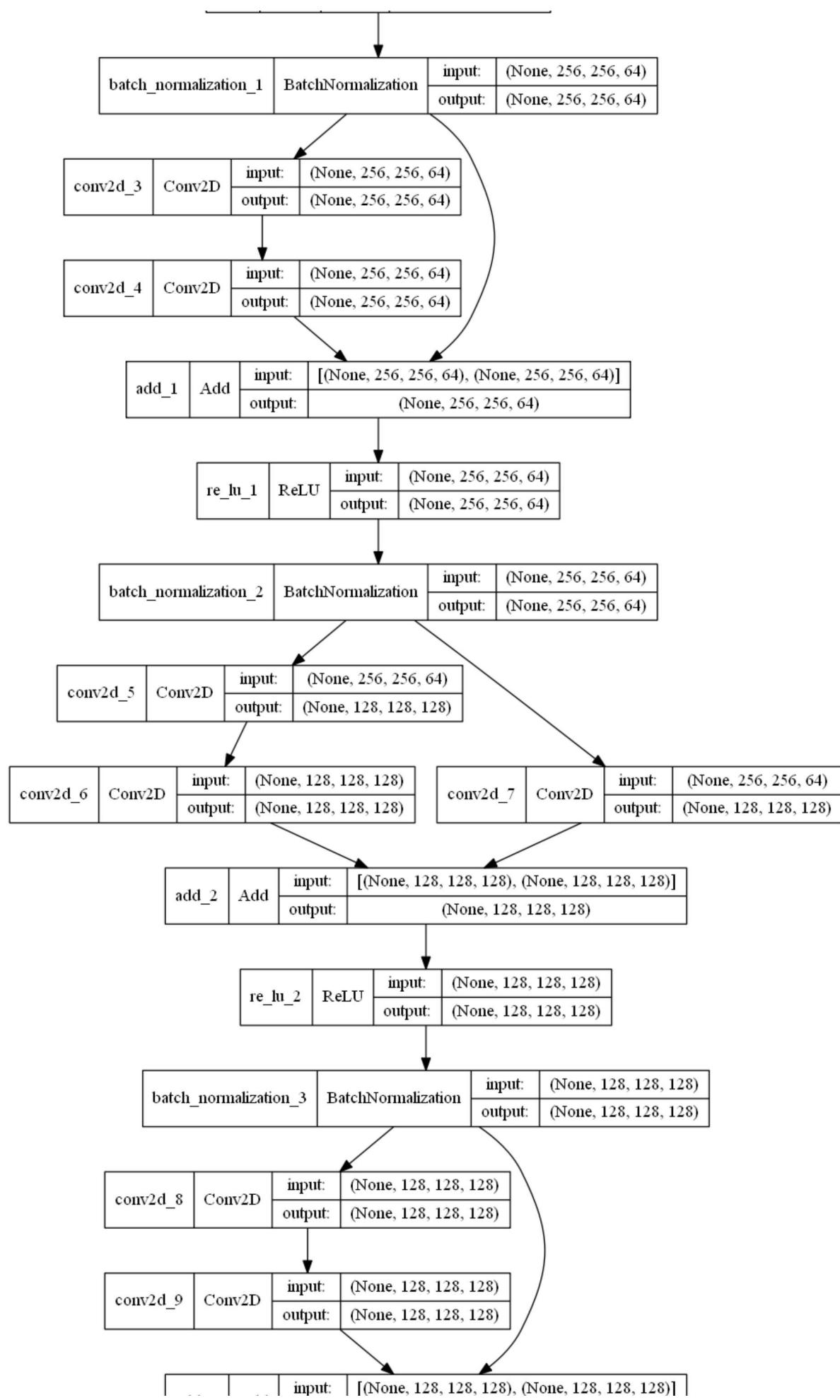
Out[18]: 168

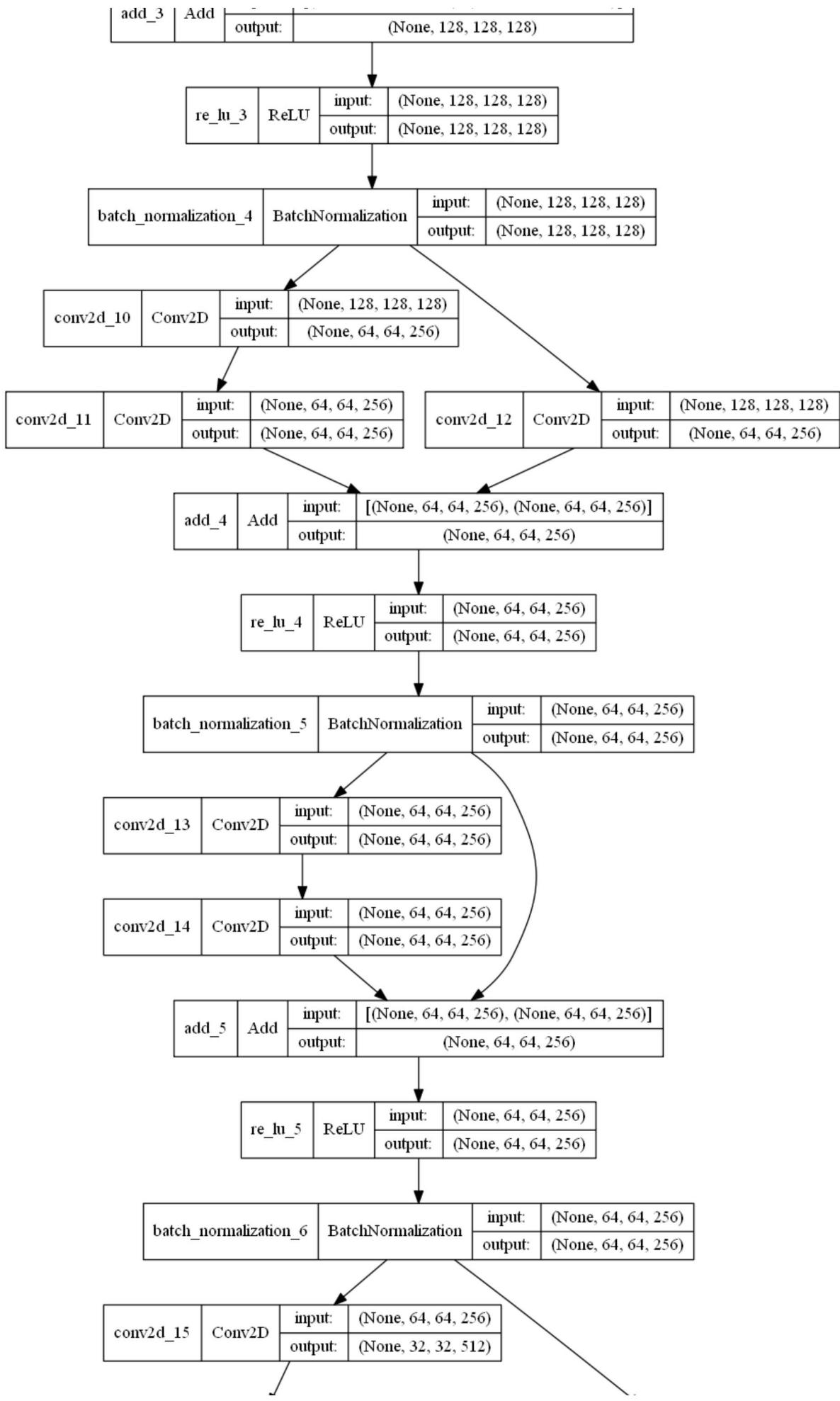
In [15]:

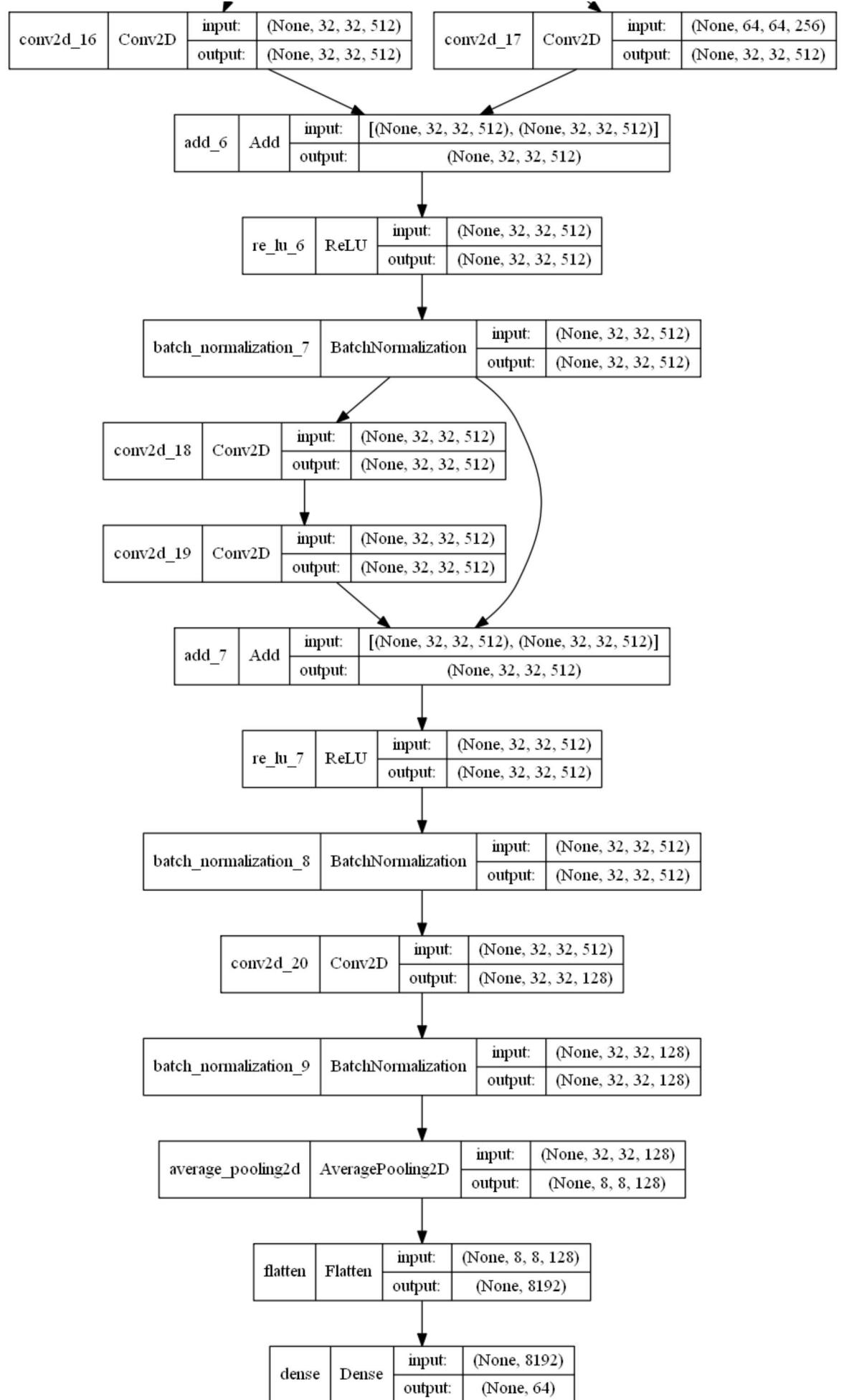
```
plot_model (model, 'model.png', show_shapes=True)
```

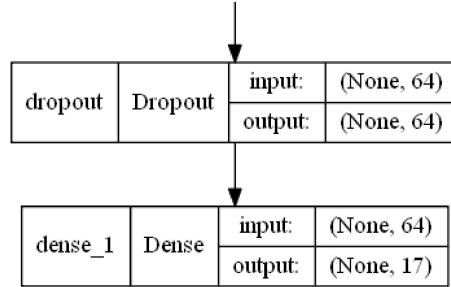
Out[15]:











```
In [17]: model.save('ResNet_augmented_90_final.h5', overwrite=True,  
    include_optimizer=True)
```

```
In [ ]:
```