Emotion recognition with MobileNet

In this portion of the assignment, I will leverage a pre-trained model (based on either the MobileNet or EfficientNet-B0) for the task of emotion recognition.

[Give an overview about the advantages and differences of the model]

Import the libraries and dataset

In [1]:
```
from datasets import load_dataset                # Function to load a dataset from the HuggingFace database
import matplotlib.pyplot as plt
import numpy as np                               #
from numpy import round, sqrt
from numpy import random                         # For showing a random array of images from the dataset
from PIL import Image

import tensorflow as tf
import keras
from keras import layers
from keras.applications import mobilenet, MobileNet

dataset_raw = load_dataset("FastJobs/Visual_Emotional_Analysis")
num_data = dataset_raw['train'].shape[0]

# Number of unique labels
num_classes = len(set(dataset_raw['train']['label']))
# Dimension of the input images (taken a single image)
image_dim = np.array(dataset_raw['train']['image'][0]).shape

# Dictionary to decode the meaning of the numerical labels
label_dict = {
    0: 'anger',
    1: 'contempt',
    2: 'disgust',
    3: 'fear',
    4: 'happy',
    5: 'neural',
    6: 'sad',
    7: 'suprise'
}
```

```
Resolving data files:   0%|          | 0/800 [00:00<?, ?it/s]
```

Make a train-test split to separate the training+validation data from the holdout set

In [2]:
```
dataset_split = dataset_raw['train'].train_test_split(test_size=0.2)

print(dataset_split)

num_train = len(dataset_split['train']['image'])
# num_train = dataset_split['train'].shape[0]
num_test = len(dataset_split['test']['image'])
# num_test = dataset_split['test'].shape[0]

# Print the number of samples in each set
print(f"There are {num_train} samples in the training set \n"
      f"There are {num_test} samples in the testing set")
```

```
DatasetDict({
    train: Dataset({
        features: ['image', 'label'],
        num_rows: 640
    })
    test: Dataset({
        features: ['image', 'label'],
        num_rows: 160
    })
})
There are 640 samples in the training set
There are 160 samples in the testing set
```

Data discovery

Display some examples from the training dataset

In [3]:
```
num_samples = 9                                                                            # number of samples
```

```
num_rows = np.int8(round(sqrt(num_samples))); num_cols = np.int8(num_samples/num_rows)      # number of rows an
rand = random.randint(num_train,size = (num_samples))                                        # random index for

# Sample random images and their indices
image_rand = dataset_split['train'][rand]['image']
label_rand = dataset_split['train'][rand]['label']

fig, axes = plt.subplots(num_rows,num_cols,figsize=(num_rows*2,num_cols*2))

for i in range(num_rows):
    for j in range(num_cols):
        index = i * num_cols + j
        image = image_rand[index]  # Extract the image
        label = label_rand[index]  # Extract the label

        ax = axes[i,j]
        # Display the image
        ax.imshow(image)
        ax.set_title(f"Label: {label_dict[label]}")
        ax.axis("off")
```

To investigate the dimension of an image in this dataset, the image first need to be converted to a numpy array. The code block below shows that dimension of each image in terms of width x height x depth

In [4]:
```
# Let's take a random image from the sample
image = image_rand[random.randint(num_samples)]

# Dimension of the PIL image
print(f"The 2D size of the image is {image.size}\n"
      f"The depth of the image can be inferred from the image mode: {image.mode}\n")

# Dimension of the image_array
image_array = np.array(image)
image_dim = image_array.shape
print(f"Once converted to a numpy array, we can explicitly see the 3D size as \n{image_dim}")
```

```
The 2D size of the image is (96, 96)
The depth of the image can be inferred from the image mode: RGB

Once converted to a numpy array, we can explicitly see the 3D size as
(96, 96, 3)
```

Data Unpacking

In [5]:
```
image_train_array = np.array(dataset_split['train']['image'])
label_train_array = np.array(dataset_split['train']['label'])

image_test_array = np.array(dataset_split['test']['image'])
label_test_array = np.array(dataset_split['test']['label'])
```

Image data augmentation layer

Given that the model only has about 800 instances, I will perform data augmentation by applying some random transformation to the data. The goal of this layer is to present the pre-trained model with different transformation of each image every time the image is fed through the network. Thus, it does not overfit on certain feature extraction This is accomplished by adding some keras sequential layers that apply the transformation to the image before it is fed into the neural network.

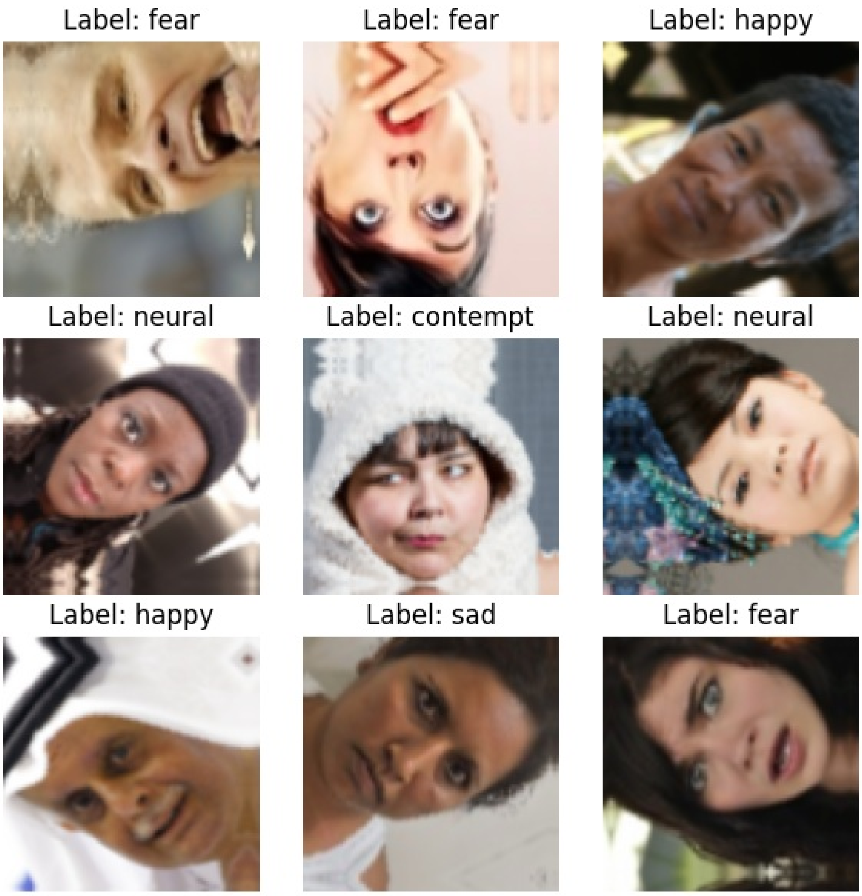
The code block belows show the construction of an image augmentation layer that will be added as the first layer of a pre-trained model.

In [6]:
```
keras.backend.clear_session()

image_augmentation = keras.Sequential([
    layers.Input(shape=(96,96,3)),
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.4),
    layers.RandomTranslation(0.1,0.1),
    layers.RandomZoom(0.1,0.1)
])
```

```
WARNING:tensorflow:From d:\Minh Nguyen\TME_6015\.venv\Lib\site-packages\keras\src\backend\common\global_state.py
:82: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.
```

The data augmentation is visualized in the code block below. I will maintain the number of samples and the sample of images/labels from the start of this notebook for a comparison.

In [7]:
```
# Sample random images and their indices
image_rand_array = image_train_array[rand]
label_rand_array = label_train_array[rand]

fig, axes1 = plt.subplots(num_rows,num_cols,figsize=(num_rows*2,num_cols*2))
fig, axes2 = plt.subplots(num_rows,num_cols,figsize=(num_rows*2,num_cols*2))

for i in range(num_rows):
    for j in range(num_cols):
        index = i * num_cols + j
        image = image_rand_array[index]  # Extract the image
        label = label_rand_array[index]  # Extract the label

        # Original pictures (no augmentation layer applied)
        ax1 = axes1[i,j]
        ax1.axis("off")
        # Display the image
        ax1.imshow(image)
        ax1.set_title(f"Label: {label_dict[label]}")


        # Apply the augmentation layer on the image
        augmented_image = image_augmentation(tf.expand_dims(image, axis=0), training=True)/255

        ax2 = axes2[i,j]
        ax2.axis("off")
        # Display the image
        ax2.imshow(tf.squeeze(augmented_image).numpy())
        ax2.set_title(f"Label: {label_dict[label]}")


        plt.tight_layout()
```



Building the MobileNet-based model

MobileNet preprocessing

In [8]:
```
# Define the image augmentation layer
image_set_augmentation = keras.Sequential([
    layers.Input(shape=(96, 96, 3)),
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.2),
    layers.RandomTranslation(0.1, 0.1),
    layers.RandomZoom(0.1, 0.1)
])

augmented_images = image_set_augmentation(image_train_array)
print(f"Dimension of the augmented training images: \n{augmented_images.shape}\n")
```

```python
preprocessed_images = mobilenet.preprocess_input(augmented_images)
print(f"Dimension of the images after preprocessing for MobileNet: \n{preprocessed_images.shape}\n")

# Uncomment the following line to observe that the intensity of the pixels are normalized to [-1,1]
# print(preprocessed_images)
```

```
Dimension of the augmented training images:
(640, 96, 96, 3)

Dimension of the images after preprocessing for MobileNet:
(640, 96, 96, 3)
```

MobileNet Base Model

In [9]:

```python
base_model = MobileNet(input_shape=image_dim,
                       include_top=False)
base_model.trainable = False
base_model.summary()
```

```
C:\Users\caomi\AppData\Local\Temp\ipykernel_44704\3163079816.py:1: UserWarning: `input_shape` is undefined or no
n-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the def
ault.
  base_model = MobileNet(input_shape=image_dim,
```

Model: "mobilenet_1.00_224"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 96, 96, 3)	0
conv1 (Conv2D)	(None, 48, 48, 32)	864
conv1_bn (BatchNormalization)	(None, 48, 48, 32)	128
conv1_relu (ReLU)	(None, 48, 48, 32)	0
conv_dw_1 (DepthwiseConv2D)	(None, 48, 48, 32)	288
conv_dw_1_bn (BatchNormalization)	(None, 48, 48, 32)	128
conv_dw_1_relu (ReLU)	(None, 48, 48, 32)	0
conv_pw_1 (Conv2D)	(None, 48, 48, 64)	2,048
conv_pw_1_bn (BatchNormalization)	(None, 48, 48, 64)	256
conv_pw_1_relu (ReLU)	(None, 48, 48, 64)	0
conv_pad_2 (ZeroPadding2D)	(None, 49, 49, 64)	0
conv_dw_2 (DepthwiseConv2D)	(None, 24, 24, 64)	576
conv_dw_2_bn (BatchNormalization)	(None, 24, 24, 64)	256
conv_dw_2_relu (ReLU)	(None, 24, 24, 64)	0
conv_pw_2 (Conv2D)	(None, 24, 24, 128)	8,192
conv_pw_2_bn (BatchNormalization)	(None, 24, 24, 128)	512
conv_pw_2_relu (ReLU)	(None, 24, 24, 128)	0
conv_dw_3 (DepthwiseConv2D)	(None, 24, 24, 128)	1,152
conv_dw_3_bn (BatchNormalization)	(None, 24, 24, 128)	512
conv_dw_3_relu (ReLU)	(None, 24, 24, 128)	0
conv_pw_3 (Conv2D)	(None, 24, 24, 128)	16,384
conv_pw_3_bn (BatchNormalization)	(None, 24, 24, 128)	512
conv_pw_3_relu (ReLU)	(None, 24, 24, 128)	0
conv_pad_4 (ZeroPadding2D)	(None, 25, 25, 128)	0

conv_dw_4 (DepthwiseConv2D)	(None, 12, 12, 128)	1,152
conv_dw_4_bn (BatchNormalization)	(None, 12, 12, 128)	512
conv_dw_4_relu (ReLU)	(None, 12, 12, 128)	0
conv_pw_4 (Conv2D)	(None, 12, 12, 256)	32,768
conv_pw_4_bn (BatchNormalization)	(None, 12, 12, 256)	1,024
conv_pw_4_relu (ReLU)	(None, 12, 12, 256)	0
conv_dw_5 (DepthwiseConv2D)	(None, 12, 12, 256)	2,304
conv_dw_5_bn (BatchNormalization)	(None, 12, 12, 256)	1,024
conv_dw_5_relu (ReLU)	(None, 12, 12, 256)	0
conv_pw_5 (Conv2D)	(None, 12, 12, 256)	65,536
conv_pw_5_bn (BatchNormalization)	(None, 12, 12, 256)	1,024
conv_pw_5_relu (ReLU)	(None, 12, 12, 256)	0
conv_pad_6 (ZeroPadding2D)	(None, 13, 13, 256)	0
conv_dw_6 (DepthwiseConv2D)	(None, 6, 6, 256)	2,304
conv_dw_6_bn (BatchNormalization)	(None, 6, 6, 256)	1,024
conv_dw_6_relu (ReLU)	(None, 6, 6, 256)	0
conv_pw_6 (Conv2D)	(None, 6, 6, 512)	131,072
conv_pw_6_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_pw_6_relu (ReLU)	(None, 6, 6, 512)	0
conv_dw_7 (DepthwiseConv2D)	(None, 6, 6, 512)	4,608
conv_dw_7_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_dw_7_relu (ReLU)	(None, 6, 6, 512)	0
conv_pw_7 (Conv2D)	(None, 6, 6, 512)	262,144
conv_pw_7_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_pw_7_relu (ReLU)	(None, 6, 6, 512)	0
conv_dw_8 (DepthwiseConv2D)	(None, 6, 6, 512)	4,608
conv_dw_8_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_dw_8_relu (ReLU)	(None, 6, 6, 512)	0
conv_pw_8 (Conv2D)	(None, 6, 6, 512)	262,144
conv_pw_8_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_pw_8_relu (ReLU)	(None, 6, 6, 512)	0
conv_dw_9 (DepthwiseConv2D)	(None, 6, 6, 512)	4,608
conv_dw_9_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_dw_9_relu (ReLU)	(None, 6, 6, 512)	0
conv_pw_9 (Conv2D)	(None, 6, 6, 512)	262,144
conv_pw_9_bn	(None, 6, 6, 512)	2,048

(BatchNormalization)		
conv_pw_9_relu (ReLU)	(None, 6, 6, 512)	0
conv_dw_10 (DepthwiseConv2D)	(None, 6, 6, 512)	4,608
conv_dw_10_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_dw_10_relu (ReLU)	(None, 6, 6, 512)	0
conv_pw_10 (Conv2D)	(None, 6, 6, 512)	262,144
conv_pw_10_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_pw_10_relu (ReLU)	(None, 6, 6, 512)	0
conv_dw_11 (DepthwiseConv2D)	(None, 6, 6, 512)	4,608
conv_dw_11_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_dw_11_relu (ReLU)	(None, 6, 6, 512)	0
conv_pw_11 (Conv2D)	(None, 6, 6, 512)	262,144
conv_pw_11_bn (BatchNormalization)	(None, 6, 6, 512)	2,048
conv_pw_11_relu (ReLU)	(None, 6, 6, 512)	0
conv_pad_12 (ZeroPadding2D)	(None, 7, 7, 512)	0
conv_dw_12 (DepthwiseConv2D)	(None, 3, 3, 512)	4,608
conv_dw_12_bn (BatchNormalization)	(None, 3, 3, 512)	2,048
conv_dw_12_relu (ReLU)	(None, 3, 3, 512)	0
conv_pw_12 (Conv2D)	(None, 3, 3, 1024)	524,288
conv_pw_12_bn (BatchNormalization)	(None, 3, 3, 1024)	4,096
conv_pw_12_relu (ReLU)	(None, 3, 3, 1024)	0
conv_dw_13 (DepthwiseConv2D)	(None, 3, 3, 1024)	9,216
conv_dw_13_bn (BatchNormalization)	(None, 3, 3, 1024)	4,096
conv_dw_13_relu (ReLU)	(None, 3, 3, 1024)	0
conv_pw_13 (Conv2D)	(None, 3, 3, 1024)	1,048,576
conv_pw_13_bn (BatchNormalization)	(None, 3, 3, 1024)	4,096
conv_pw_13_relu (ReLU)	(None, 3, 3, 1024)	0

Total params: 3,228,864 (12.32 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 3,228,864 (12.32 MB)

Full Model

In [10]:
```
# Create the model
model = keras.Sequential([
    layers.Input(shape=image_dim),
    image_augmentation,                                     # Augment the training images
    layers.Lambda(mobilenet.preprocess_input),              # mobilenet preprocessing function
    base_model,                                             # MobileNet network
    layers.BatchNormalization(),
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.4),                                    # To help with generalization
    layers.Dense(num_classes, activation="softmax")         # Custom classification layer
])

initial_weights = model.get_weights()
```

```
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 96, 96, 3)	0
lambda (Lambda)	(None, 96, 96, 3)	0
mobilenet_1.00_224 (Functional)	(None, 3, 3, 1024)	3,228,864
batch_normalization (BatchNormalization)	(None, 3, 3, 1024)	4,096
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 8)	8,200

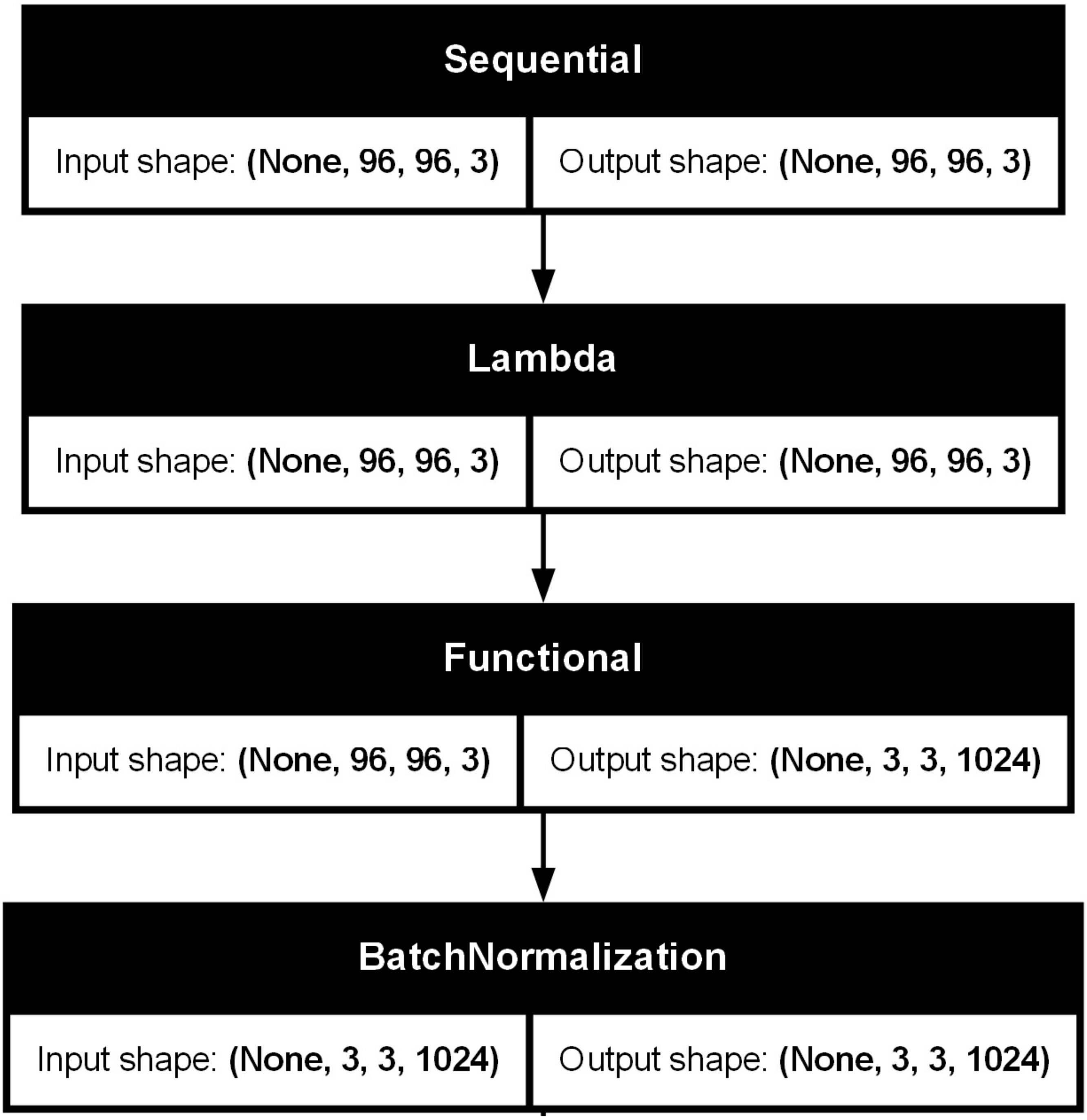
Total params: 3,241,160 (12.36 MB)

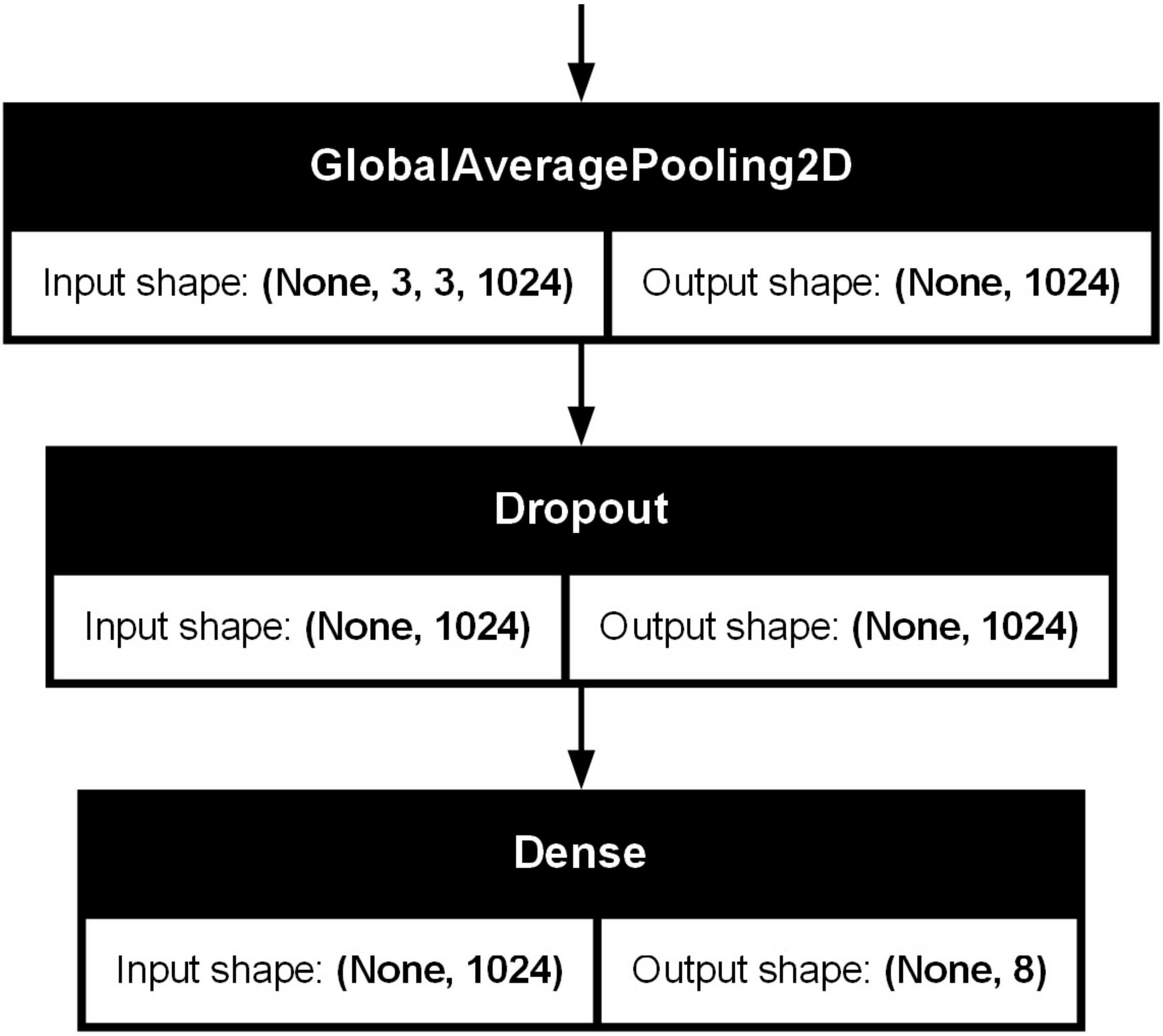
Trainable params: 10,248 (40.03 KB)

Non-trainable params: 3,230,912 (12.32 MB)

In [11]:
```
len(model.trainable_variables)
keras.utils.plot_model(model,show_shapes=True)
```

Out[11]:





Model Training

Before any training, the accuracy of the model is fairly poor and is no better than random chance (1/8=12.5%)

In [12]:
```
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.005),
              loss=keras.losses.SparseCategoricalCrossentropy,
              metrics=["accuracy"])
loss0, acc0 = model.evaluate(image_train_array,label_train_array)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(acc0))
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.1174 - loss: 4.5053
initial loss: 4.63
initial accuracy: 0.12
```

In [13]:
```
# Intitial training with frozen base model
initial_lr = 0.0005
initial_batch_size = 32
initial_epochs = 200



model.set_weights(initial_weights)

# # Learning rate scheduler
# initial_lr_scheduler = keras.optimizers.schedules.ExponentialDecay(
#     initial_learning_rate=initial_lr,
#     decay_steps=1000,
#     decay_rate=0.5
# )
base_model.trainable = False
model.compile(optimizer=keras.optimizers.Adam(learning_rate=initial_lr),
              loss=keras.losses.SparseCategoricalCrossentropy,
              metrics=["accuracy"])
model.summary()

history_initial = model.fit(image_train_array,
                    label_train_array,
```

```
                    epochs=initial_epochs,
                    batch_size=initial_batch_size,
                    validation_split=0.2,
                    verbose=2)

# Fine tuning by unfreezing some later layers of MobileNet
ftune_lr = 0.00005
ftune_batch_size = 16
ftune_epochs = 100
unfrozen_layer = -10

base_model.trainable = True

for layer in base_model.layers[:unfrozen_layer]:
    layer.trainable = False

# Learning rate scheduler
ftune_lr_scheduler = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=ftune_lr,
    decay_steps=1000,
    decay_rate=0.5
)

model.compile(optimizer=keras.optimizers.Adam(learning_rate=ftune_lr_scheduler),
              loss=keras.losses.SparseCategoricalCrossentropy,
              metrics=["accuracy"])
model.summary()

history_fine = model.fit(image_train_array,
                         label_train_array,
                         epochs=initial_epochs + ftune_epochs,
                         validation_split=0.2,
                         initial_epoch=history_initial.epoch[-1],       # Resume from previous training
                         verbose = 2)
```

Model: "sequential_2"

Layer (type)	**Output Shape**	**Param #**
sequential (Sequential)	(None, 96, 96, 3)	0
lambda (Lambda)	(None, 96, 96, 3)	0
mobilenet_1.00_224 (Functional)	(None, 3, 3, 1024)	3,228,864
batch_normalization (BatchNormalization)	(None, 3, 3, 1024)	4,096
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 8)	8,200

Total params: 3,241,160 (12.36 MB)

Trainable params: 10,248 (40.03 KB)

Non-trainable params: 3,230,912 (12.32 MB)

```
Epoch 1/200
16/16 - 3s - 187ms/step - accuracy: 0.1211 - loss: 2.7069 - val_accuracy: 0.0938 - val_loss: 3.7358
Epoch 2/200
16/16 - 1s - 49ms/step - accuracy: 0.1504 - loss: 2.6170 - val_accuracy: 0.0859 - val_loss: 3.1438
Epoch 3/200
16/16 - 1s - 50ms/step - accuracy: 0.1758 - loss: 2.4230 - val_accuracy: 0.1250 - val_loss: 2.8460
Epoch 4/200
16/16 - 1s - 50ms/step - accuracy: 0.2344 - loss: 2.2968 - val_accuracy: 0.1250 - val_loss: 2.6429
Epoch 5/200
16/16 - 1s - 49ms/step - accuracy: 0.2031 - loss: 2.2202 - val_accuracy: 0.1406 - val_loss: 2.5354
Epoch 6/200
16/16 - 1s - 56ms/step - accuracy: 0.2363 - loss: 2.1302 - val_accuracy: 0.1641 - val_loss: 2.4253
Epoch 7/200
16/16 - 1s - 56ms/step - accuracy: 0.2520 - loss: 2.0680 - val_accuracy: 0.1797 - val_loss: 2.3438
Epoch 8/200
16/16 - 1s - 50ms/step - accuracy: 0.2383 - loss: 2.1001 - val_accuracy: 0.1797 - val_loss: 2.2745
Epoch 9/200
16/16 - 1s - 50ms/step - accuracy: 0.2578 - loss: 2.0535 - val_accuracy: 0.1875 - val_loss: 2.2363
Epoch 10/200
16/16 - 1s - 48ms/step - accuracy: 0.3047 - loss: 1.9915 - val_accuracy: 0.1875 - val_loss: 2.2236
Epoch 11/200
16/16 - 1s - 49ms/step - accuracy: 0.2676 - loss: 2.0246 - val_accuracy: 0.1875 - val_loss: 2.2114
Epoch 12/200
16/16 - 1s - 48ms/step - accuracy: 0.2969 - loss: 2.0137 - val_accuracy: 0.1875 - val_loss: 2.2004
```

```
Epoch 13/200
16/16 - 1s - 50ms/step - accuracy: 0.2871 - loss: 2.0507 - val_accuracy: 0.1797 - val_loss: 2.1699
Epoch 14/200
16/16 - 1s - 50ms/step - accuracy: 0.3008 - loss: 1.9107 - val_accuracy: 0.1719 - val_loss: 2.1624
Epoch 15/200
16/16 - 1s - 50ms/step - accuracy: 0.2871 - loss: 1.8863 - val_accuracy: 0.2031 - val_loss: 2.1312
Epoch 16/200
16/16 - 1s - 52ms/step - accuracy: 0.3125 - loss: 1.8715 - val_accuracy: 0.2109 - val_loss: 2.1203
Epoch 17/200
16/16 - 1s - 48ms/step - accuracy: 0.3613 - loss: 1.8257 - val_accuracy: 0.2266 - val_loss: 2.1136
Epoch 18/200
16/16 - 1s - 52ms/step - accuracy: 0.3145 - loss: 1.8241 - val_accuracy: 0.2109 - val_loss: 2.1097
Epoch 19/200
16/16 - 1s - 50ms/step - accuracy: 0.3594 - loss: 1.7941 - val_accuracy: 0.2031 - val_loss: 2.1011
Epoch 20/200
16/16 - 1s - 50ms/step - accuracy: 0.3418 - loss: 1.7567 - val_accuracy: 0.1953 - val_loss: 2.1085
Epoch 21/200
16/16 - 1s - 49ms/step - accuracy: 0.3652 - loss: 1.7405 - val_accuracy: 0.1875 - val_loss: 2.1351
Epoch 22/200
16/16 - 1s - 48ms/step - accuracy: 0.3242 - loss: 1.8515 - val_accuracy: 0.1875 - val_loss: 2.1280
Epoch 23/200
16/16 - 1s - 49ms/step - accuracy: 0.3652 - loss: 1.6945 - val_accuracy: 0.2109 - val_loss: 2.1031
Epoch 24/200
16/16 - 1s - 49ms/step - accuracy: 0.3711 - loss: 1.6718 - val_accuracy: 0.2031 - val_loss: 2.1196
Epoch 25/200
16/16 - 1s - 49ms/step - accuracy: 0.3535 - loss: 1.7519 - val_accuracy: 0.1953 - val_loss: 2.1356
Epoch 26/200
16/16 - 1s - 49ms/step - accuracy: 0.3438 - loss: 1.7052 - val_accuracy: 0.2109 - val_loss: 2.1246
Epoch 27/200
16/16 - 1s - 50ms/step - accuracy: 0.3535 - loss: 1.7039 - val_accuracy: 0.2188 - val_loss: 2.1208
Epoch 28/200
16/16 - 1s - 50ms/step - accuracy: 0.3984 - loss: 1.6576 - val_accuracy: 0.1953 - val_loss: 2.1111
Epoch 29/200
16/16 - 1s - 49ms/step - accuracy: 0.3848 - loss: 1.6475 - val_accuracy: 0.1953 - val_loss: 2.1092
Epoch 30/200
16/16 - 1s - 50ms/step - accuracy: 0.3691 - loss: 1.6865 - val_accuracy: 0.1875 - val_loss: 2.1022
Epoch 31/200
16/16 - 1s - 49ms/step - accuracy: 0.4199 - loss: 1.6538 - val_accuracy: 0.1953 - val_loss: 2.0806
Epoch 32/200
16/16 - 1s - 51ms/step - accuracy: 0.3926 - loss: 1.6345 - val_accuracy: 0.2188 - val_loss: 2.0699
Epoch 33/200
16/16 - 1s - 51ms/step - accuracy: 0.4062 - loss: 1.5766 - val_accuracy: 0.2031 - val_loss: 2.0796
Epoch 34/200
16/16 - 1s - 49ms/step - accuracy: 0.3965 - loss: 1.6024 - val_accuracy: 0.1953 - val_loss: 2.0795
Epoch 35/200
16/16 - 1s - 51ms/step - accuracy: 0.4297 - loss: 1.5623 - val_accuracy: 0.1953 - val_loss: 2.0771
Epoch 36/200
16/16 - 1s - 51ms/step - accuracy: 0.4023 - loss: 1.5836 - val_accuracy: 0.1953 - val_loss: 2.0683
Epoch 37/200
16/16 - 1s - 51ms/step - accuracy: 0.4023 - loss: 1.5986 - val_accuracy: 0.2031 - val_loss: 2.0947
Epoch 38/200
16/16 - 1s - 51ms/step - accuracy: 0.4180 - loss: 1.5692 - val_accuracy: 0.2188 - val_loss: 2.0777
Epoch 39/200
16/16 - 1s - 51ms/step - accuracy: 0.4102 - loss: 1.5787 - val_accuracy: 0.2031 - val_loss: 2.0650
Epoch 40/200
16/16 - 1s - 50ms/step - accuracy: 0.4004 - loss: 1.5636 - val_accuracy: 0.2031 - val_loss: 2.0567
Epoch 41/200
16/16 - 1s - 51ms/step - accuracy: 0.3887 - loss: 1.6083 - val_accuracy: 0.2109 - val_loss: 2.0528
Epoch 42/200
16/16 - 1s - 50ms/step - accuracy: 0.4434 - loss: 1.5068 - val_accuracy: 0.2109 - val_loss: 2.0566
Epoch 43/200
16/16 - 1s - 51ms/step - accuracy: 0.4434 - loss: 1.5146 - val_accuracy: 0.2188 - val_loss: 2.0653
Epoch 44/200
16/16 - 1s - 50ms/step - accuracy: 0.4238 - loss: 1.5649 - val_accuracy: 0.2109 - val_loss: 2.0652
Epoch 45/200
16/16 - 1s - 51ms/step - accuracy: 0.4297 - loss: 1.4848 - val_accuracy: 0.2031 - val_loss: 2.0568
Epoch 46/200
16/16 - 1s - 51ms/step - accuracy: 0.4512 - loss: 1.4731 - val_accuracy: 0.2031 - val_loss: 2.0729
Epoch 47/200
16/16 - 1s - 51ms/step - accuracy: 0.4219 - loss: 1.5216 - val_accuracy: 0.1797 - val_loss: 2.0837
Epoch 48/200
16/16 - 1s - 49ms/step - accuracy: 0.4160 - loss: 1.5348 - val_accuracy: 0.1953 - val_loss: 2.0739
Epoch 49/200
16/16 - 1s - 51ms/step - accuracy: 0.4570 - loss: 1.4111 - val_accuracy: 0.2109 - val_loss: 2.0734
Epoch 50/200
16/16 - 1s - 51ms/step - accuracy: 0.4453 - loss: 1.4757 - val_accuracy: 0.2188 - val_loss: 2.0770
Epoch 51/200
16/16 - 1s - 53ms/step - accuracy: 0.4297 - loss: 1.4770 - val_accuracy: 0.2188 - val_loss: 2.0878
Epoch 52/200
16/16 - 1s - 50ms/step - accuracy: 0.4531 - loss: 1.4340 - val_accuracy: 0.2031 - val_loss: 2.0796
Epoch 53/200
16/16 - 1s - 50ms/step - accuracy: 0.4863 - loss: 1.5040 - val_accuracy: 0.2109 - val_loss: 2.0575
Epoch 54/200
```

```
16/16 - 1s - 50ms/step - accuracy: 0.4668 - loss: 1.4247 - val_accuracy: 0.2109 - val_loss: 2.0422
Epoch 55/200
16/16 - 1s - 51ms/step - accuracy: 0.4824 - loss: 1.4295 - val_accuracy: 0.1875 - val_loss: 2.0565
Epoch 56/200
16/16 - 1s - 51ms/step - accuracy: 0.4512 - loss: 1.4698 - val_accuracy: 0.1797 - val_loss: 2.0807
Epoch 57/200
16/16 - 1s - 50ms/step - accuracy: 0.4355 - loss: 1.4860 - val_accuracy: 0.1797 - val_loss: 2.0747
Epoch 58/200
16/16 - 1s - 51ms/step - accuracy: 0.4941 - loss: 1.3507 - val_accuracy: 0.2031 - val_loss: 2.0817
Epoch 59/200
16/16 - 1s - 49ms/step - accuracy: 0.4395 - loss: 1.4622 - val_accuracy: 0.2109 - val_loss: 2.0876
Epoch 60/200
16/16 - 1s - 52ms/step - accuracy: 0.4355 - loss: 1.4692 - val_accuracy: 0.1953 - val_loss: 2.0696
Epoch 61/200
16/16 - 1s - 50ms/step - accuracy: 0.4727 - loss: 1.4661 - val_accuracy: 0.2031 - val_loss: 2.0409
Epoch 62/200
16/16 - 1s - 50ms/step - accuracy: 0.4414 - loss: 1.4670 - val_accuracy: 0.2109 - val_loss: 2.0416
Epoch 63/200
16/16 - 1s - 52ms/step - accuracy: 0.5117 - loss: 1.4138 - val_accuracy: 0.2031 - val_loss: 2.0300
Epoch 64/200
16/16 - 1s - 51ms/step - accuracy: 0.4629 - loss: 1.4556 - val_accuracy: 0.2031 - val_loss: 2.0163
Epoch 65/200
16/16 - 1s - 52ms/step - accuracy: 0.4785 - loss: 1.4129 - val_accuracy: 0.2031 - val_loss: 2.0335
Epoch 66/200
16/16 - 1s - 50ms/step - accuracy: 0.4590 - loss: 1.4297 - val_accuracy: 0.2031 - val_loss: 2.0434
Epoch 67/200
16/16 - 1s - 50ms/step - accuracy: 0.5078 - loss: 1.4077 - val_accuracy: 0.2109 - val_loss: 2.0449
Epoch 68/200
16/16 - 1s - 51ms/step - accuracy: 0.4961 - loss: 1.3656 - val_accuracy: 0.2031 - val_loss: 2.0455
Epoch 69/200
16/16 - 1s - 53ms/step - accuracy: 0.4941 - loss: 1.4101 - val_accuracy: 0.1875 - val_loss: 2.0690
Epoch 70/200
16/16 - 1s - 52ms/step - accuracy: 0.4844 - loss: 1.3964 - val_accuracy: 0.2188 - val_loss: 2.0659
Epoch 71/200
16/16 - 1s - 50ms/step - accuracy: 0.4688 - loss: 1.4468 - val_accuracy: 0.2266 - val_loss: 2.0663
Epoch 72/200
16/16 - 1s - 51ms/step - accuracy: 0.4727 - loss: 1.3808 - val_accuracy: 0.2109 - val_loss: 2.0547
Epoch 73/200
16/16 - 1s - 51ms/step - accuracy: 0.5039 - loss: 1.3640 - val_accuracy: 0.2188 - val_loss: 2.0617
Epoch 74/200
16/16 - 1s - 52ms/step - accuracy: 0.4746 - loss: 1.4369 - val_accuracy: 0.1875 - val_loss: 2.0591
Epoch 75/200
16/16 - 1s - 52ms/step - accuracy: 0.4707 - loss: 1.4776 - val_accuracy: 0.2031 - val_loss: 2.0563
Epoch 76/200
16/16 - 1s - 50ms/step - accuracy: 0.4570 - loss: 1.4326 - val_accuracy: 0.2109 - val_loss: 2.0441
Epoch 77/200
16/16 - 1s - 50ms/step - accuracy: 0.5039 - loss: 1.4030 - val_accuracy: 0.2109 - val_loss: 2.0358
Epoch 78/200
16/16 - 1s - 51ms/step - accuracy: 0.4746 - loss: 1.3894 - val_accuracy: 0.2188 - val_loss: 2.0235
Epoch 79/200
16/16 - 1s - 51ms/step - accuracy: 0.4824 - loss: 1.3516 - val_accuracy: 0.2188 - val_loss: 2.0255
Epoch 80/200
16/16 - 1s - 50ms/step - accuracy: 0.5000 - loss: 1.3298 - val_accuracy: 0.2344 - val_loss: 2.0420
Epoch 81/200
16/16 - 1s - 50ms/step - accuracy: 0.4785 - loss: 1.3596 - val_accuracy: 0.2109 - val_loss: 2.0439
Epoch 82/200
16/16 - 1s - 50ms/step - accuracy: 0.4844 - loss: 1.3475 - val_accuracy: 0.2188 - val_loss: 2.0485
Epoch 83/200
16/16 - 1s - 53ms/step - accuracy: 0.4922 - loss: 1.3675 - val_accuracy: 0.2266 - val_loss: 2.0401
Epoch 84/200
16/16 - 1s - 53ms/step - accuracy: 0.4844 - loss: 1.3516 - val_accuracy: 0.2188 - val_loss: 2.0235
Epoch 85/200
16/16 - 1s - 54ms/step - accuracy: 0.5234 - loss: 1.3300 - val_accuracy: 0.2188 - val_loss: 2.0311
Epoch 86/200
16/16 - 1s - 54ms/step - accuracy: 0.5293 - loss: 1.3183 - val_accuracy: 0.2266 - val_loss: 2.0324
Epoch 87/200
16/16 - 1s - 53ms/step - accuracy: 0.4941 - loss: 1.3758 - val_accuracy: 0.2109 - val_loss: 2.0559
Epoch 88/200
16/16 - 1s - 51ms/step - accuracy: 0.4844 - loss: 1.3877 - val_accuracy: 0.2109 - val_loss: 2.0649
Epoch 89/200
16/16 - 1s - 51ms/step - accuracy: 0.5195 - loss: 1.3360 - val_accuracy: 0.2031 - val_loss: 2.0691
Epoch 90/200
16/16 - 1s - 50ms/step - accuracy: 0.4922 - loss: 1.3354 - val_accuracy: 0.2031 - val_loss: 2.0562
Epoch 91/200
16/16 - 1s - 50ms/step - accuracy: 0.4941 - loss: 1.3756 - val_accuracy: 0.1953 - val_loss: 2.0396
Epoch 92/200
16/16 - 1s - 52ms/step - accuracy: 0.5273 - loss: 1.3533 - val_accuracy: 0.1797 - val_loss: 2.0514
Epoch 93/200
16/16 - 1s - 51ms/step - accuracy: 0.4883 - loss: 1.3540 - val_accuracy: 0.1875 - val_loss: 2.0562
Epoch 94/200
16/16 - 1s - 51ms/step - accuracy: 0.5215 - loss: 1.3317 - val_accuracy: 0.1953 - val_loss: 2.0778
Epoch 95/200
16/16 - 1s - 53ms/step - accuracy: 0.4902 - loss: 1.3419 - val_accuracy: 0.2109 - val_loss: 2.0834
```

```
Epoch 96/200
16/16 - 1s - 50ms/step - accuracy: 0.5215 - loss: 1.3141 - val_accuracy: 0.2031 - val_loss: 2.0866
Epoch 97/200
16/16 - 1s - 52ms/step - accuracy: 0.5215 - loss: 1.2979 - val_accuracy: 0.2109 - val_loss: 2.0834
Epoch 98/200
16/16 - 1s - 52ms/step - accuracy: 0.5098 - loss: 1.3781 - val_accuracy: 0.2266 - val_loss: 2.0668
Epoch 99/200
16/16 - 1s - 50ms/step - accuracy: 0.5254 - loss: 1.3085 - val_accuracy: 0.2188 - val_loss: 2.0529
Epoch 100/200
16/16 - 1s - 50ms/step - accuracy: 0.5254 - loss: 1.3198 - val_accuracy: 0.2031 - val_loss: 2.0630
Epoch 101/200
16/16 - 1s - 52ms/step - accuracy: 0.5195 - loss: 1.3654 - val_accuracy: 0.2109 - val_loss: 2.0659
Epoch 102/200
16/16 - 1s - 51ms/step - accuracy: 0.5137 - loss: 1.3283 - val_accuracy: 0.1797 - val_loss: 2.0871
Epoch 103/200
16/16 - 1s - 50ms/step - accuracy: 0.4727 - loss: 1.3974 - val_accuracy: 0.2188 - val_loss: 2.0973
Epoch 104/200
16/16 - 1s - 50ms/step - accuracy: 0.5195 - loss: 1.3031 - val_accuracy: 0.1953 - val_loss: 2.1097
Epoch 105/200
16/16 - 1s - 50ms/step - accuracy: 0.5156 - loss: 1.2637 - val_accuracy: 0.1797 - val_loss: 2.1369
Epoch 106/200
16/16 - 1s - 51ms/step - accuracy: 0.5273 - loss: 1.2729 - val_accuracy: 0.2031 - val_loss: 2.1345
Epoch 107/200
16/16 - 1s - 51ms/step - accuracy: 0.5020 - loss: 1.3652 - val_accuracy: 0.2109 - val_loss: 2.1040
Epoch 108/200
16/16 - 1s - 51ms/step - accuracy: 0.5020 - loss: 1.3399 - val_accuracy: 0.2031 - val_loss: 2.0844
Epoch 109/200
16/16 - 1s - 52ms/step - accuracy: 0.4980 - loss: 1.3212 - val_accuracy: 0.2109 - val_loss: 2.0992
Epoch 110/200
16/16 - 1s - 51ms/step - accuracy: 0.5254 - loss: 1.3083 - val_accuracy: 0.2188 - val_loss: 2.1161
Epoch 111/200
16/16 - 1s - 52ms/step - accuracy: 0.5098 - loss: 1.3286 - val_accuracy: 0.1953 - val_loss: 2.1350
Epoch 112/200
16/16 - 1s - 50ms/step - accuracy: 0.5000 - loss: 1.3242 - val_accuracy: 0.2031 - val_loss: 2.1493
Epoch 113/200
16/16 - 1s - 51ms/step - accuracy: 0.4980 - loss: 1.3495 - val_accuracy: 0.1797 - val_loss: 2.1669
Epoch 114/200
16/16 - 1s - 50ms/step - accuracy: 0.5059 - loss: 1.3308 - val_accuracy: 0.1875 - val_loss: 2.1596
Epoch 115/200
16/16 - 1s - 51ms/step - accuracy: 0.5488 - loss: 1.2507 - val_accuracy: 0.2109 - val_loss: 2.1457
Epoch 116/200
16/16 - 1s - 51ms/step - accuracy: 0.4863 - loss: 1.3377 - val_accuracy: 0.2188 - val_loss: 2.1447
Epoch 117/200
16/16 - 1s - 50ms/step - accuracy: 0.5312 - loss: 1.2911 - val_accuracy: 0.2266 - val_loss: 2.1231
Epoch 118/200
16/16 - 1s - 51ms/step - accuracy: 0.5430 - loss: 1.2587 - val_accuracy: 0.2188 - val_loss: 2.1290
Epoch 119/200
16/16 - 1s - 50ms/step - accuracy: 0.5039 - loss: 1.2891 - val_accuracy: 0.2344 - val_loss: 2.1301
Epoch 120/200
16/16 - 1s - 54ms/step - accuracy: 0.5059 - loss: 1.3156 - val_accuracy: 0.2344 - val_loss: 2.1331
Epoch 121/200
16/16 - 1s - 50ms/step - accuracy: 0.5117 - loss: 1.3031 - val_accuracy: 0.2188 - val_loss: 2.1311
Epoch 122/200
16/16 - 1s - 50ms/step - accuracy: 0.5234 - loss: 1.3271 - val_accuracy: 0.2266 - val_loss: 2.1209
Epoch 123/200
16/16 - 1s - 51ms/step - accuracy: 0.5234 - loss: 1.2710 - val_accuracy: 0.2500 - val_loss: 2.1206
Epoch 124/200
16/16 - 1s - 50ms/step - accuracy: 0.5059 - loss: 1.3322 - val_accuracy: 0.2344 - val_loss: 2.1154
Epoch 125/200
16/16 - 1s - 52ms/step - accuracy: 0.5332 - loss: 1.2622 - val_accuracy: 0.2344 - val_loss: 2.1096
Epoch 126/200
16/16 - 1s - 51ms/step - accuracy: 0.5273 - loss: 1.2675 - val_accuracy: 0.2188 - val_loss: 2.0946
Epoch 127/200
16/16 - 1s - 49ms/step - accuracy: 0.5215 - loss: 1.2769 - val_accuracy: 0.2500 - val_loss: 2.0936
Epoch 128/200
16/16 - 1s - 50ms/step - accuracy: 0.4531 - loss: 1.4170 - val_accuracy: 0.2266 - val_loss: 2.0992
Epoch 129/200
16/16 - 1s - 50ms/step - accuracy: 0.5586 - loss: 1.2488 - val_accuracy: 0.2266 - val_loss: 2.1242
Epoch 130/200
16/16 - 1s - 51ms/step - accuracy: 0.5059 - loss: 1.2752 - val_accuracy: 0.2344 - val_loss: 2.1509
Epoch 131/200
16/16 - 1s - 50ms/step - accuracy: 0.5391 - loss: 1.2347 - val_accuracy: 0.2109 - val_loss: 2.1603
Epoch 132/200
16/16 - 1s - 52ms/step - accuracy: 0.5156 - loss: 1.3233 - val_accuracy: 0.2188 - val_loss: 2.1286
Epoch 133/200
16/16 - 1s - 51ms/step - accuracy: 0.5098 - loss: 1.2593 - val_accuracy: 0.2031 - val_loss: 2.1116
Epoch 134/200
16/16 - 1s - 51ms/step - accuracy: 0.5195 - loss: 1.2859 - val_accuracy: 0.2031 - val_loss: 2.1178
Epoch 135/200
16/16 - 1s - 51ms/step - accuracy: 0.5332 - loss: 1.2703 - val_accuracy: 0.2031 - val_loss: 2.0945
Epoch 136/200
16/16 - 1s - 50ms/step - accuracy: 0.5410 - loss: 1.2611 - val_accuracy: 0.2109 - val_loss: 2.0909
Epoch 137/200
```

```
16/16 - 1s - 50ms/step - accuracy: 0.5664 - loss: 1.2648 - val_accuracy: 0.2031 - val_loss: 2.1169
Epoch 138/200
16/16 - 1s - 50ms/step - accuracy: 0.5469 - loss: 1.2815 - val_accuracy: 0.2109 - val_loss: 2.1178
Epoch 139/200
16/16 - 1s - 51ms/step - accuracy: 0.5605 - loss: 1.2567 - val_accuracy: 0.2266 - val_loss: 2.1150
Epoch 140/200
16/16 - 1s - 50ms/step - accuracy: 0.5195 - loss: 1.2455 - val_accuracy: 0.2344 - val_loss: 2.1147
Epoch 141/200
16/16 - 1s - 51ms/step - accuracy: 0.5176 - loss: 1.2900 - val_accuracy: 0.2266 - val_loss: 2.1111
Epoch 142/200
16/16 - 1s - 51ms/step - accuracy: 0.5312 - loss: 1.2410 - val_accuracy: 0.2344 - val_loss: 2.1116
Epoch 143/200
16/16 - 1s - 51ms/step - accuracy: 0.5254 - loss: 1.2497 - val_accuracy: 0.2344 - val_loss: 2.0943
Epoch 144/200
16/16 - 1s - 52ms/step - accuracy: 0.5312 - loss: 1.2897 - val_accuracy: 0.2422 - val_loss: 2.1009
Epoch 145/200
16/16 - 1s - 52ms/step - accuracy: 0.5410 - loss: 1.2798 - val_accuracy: 0.2422 - val_loss: 2.1184
Epoch 146/200
16/16 - 1s - 54ms/step - accuracy: 0.5449 - loss: 1.2888 - val_accuracy: 0.2422 - val_loss: 2.1254
Epoch 147/200
16/16 - 1s - 51ms/step - accuracy: 0.5586 - loss: 1.2546 - val_accuracy: 0.2266 - val_loss: 2.1235
Epoch 148/200
16/16 - 1s - 52ms/step - accuracy: 0.5078 - loss: 1.2946 - val_accuracy: 0.2266 - val_loss: 2.1207
Epoch 149/200
16/16 - 1s - 51ms/step - accuracy: 0.5488 - loss: 1.2578 - val_accuracy: 0.2422 - val_loss: 2.1293
Epoch 150/200
16/16 - 1s - 50ms/step - accuracy: 0.5254 - loss: 1.3044 - val_accuracy: 0.2266 - val_loss: 2.1575
Epoch 151/200
16/16 - 1s - 51ms/step - accuracy: 0.5391 - loss: 1.2421 - val_accuracy: 0.2266 - val_loss: 2.1735
Epoch 152/200
16/16 - 1s - 51ms/step - accuracy: 0.5293 - loss: 1.3186 - val_accuracy: 0.2422 - val_loss: 2.1649
Epoch 153/200
16/16 - 1s - 52ms/step - accuracy: 0.5449 - loss: 1.2702 - val_accuracy: 0.2188 - val_loss: 2.1780
Epoch 154/200
16/16 - 1s - 52ms/step - accuracy: 0.5234 - loss: 1.2347 - val_accuracy: 0.2266 - val_loss: 2.2077
Epoch 155/200
16/16 - 1s - 51ms/step - accuracy: 0.5430 - loss: 1.3152 - val_accuracy: 0.2266 - val_loss: 2.2069
Epoch 156/200
16/16 - 1s - 50ms/step - accuracy: 0.5156 - loss: 1.2804 - val_accuracy: 0.2422 - val_loss: 2.2020
Epoch 157/200
16/16 - 1s - 52ms/step - accuracy: 0.5684 - loss: 1.1703 - val_accuracy: 0.2344 - val_loss: 2.2094
Epoch 158/200
16/16 - 1s - 51ms/step - accuracy: 0.5293 - loss: 1.2370 - val_accuracy: 0.2188 - val_loss: 2.2186
Epoch 159/200
16/16 - 1s - 50ms/step - accuracy: 0.5293 - loss: 1.2678 - val_accuracy: 0.2188 - val_loss: 2.2055
Epoch 160/200
16/16 - 1s - 51ms/step - accuracy: 0.5078 - loss: 1.2998 - val_accuracy: 0.2266 - val_loss: 2.1838
Epoch 161/200
16/16 - 1s - 51ms/step - accuracy: 0.5430 - loss: 1.2472 - val_accuracy: 0.2344 - val_loss: 2.1530
Epoch 162/200
16/16 - 1s - 51ms/step - accuracy: 0.5605 - loss: 1.2569 - val_accuracy: 0.2266 - val_loss: 2.1238
Epoch 163/200
16/16 - 1s - 51ms/step - accuracy: 0.5508 - loss: 1.2550 - val_accuracy: 0.2266 - val_loss: 2.1239
Epoch 164/200
16/16 - 1s - 50ms/step - accuracy: 0.5176 - loss: 1.3149 - val_accuracy: 0.2266 - val_loss: 2.1151
Epoch 165/200
16/16 - 1s - 51ms/step - accuracy: 0.5332 - loss: 1.2094 - val_accuracy: 0.2344 - val_loss: 2.1326
Epoch 166/200
16/16 - 1s - 52ms/step - accuracy: 0.5098 - loss: 1.3073 - val_accuracy: 0.2422 - val_loss: 2.1468
Epoch 167/200
16/16 - 1s - 52ms/step - accuracy: 0.5449 - loss: 1.2556 - val_accuracy: 0.2422 - val_loss: 2.1356
Epoch 168/200
16/16 - 1s - 50ms/step - accuracy: 0.5215 - loss: 1.2781 - val_accuracy: 0.2500 - val_loss: 2.1192
Epoch 169/200
16/16 - 1s - 51ms/step - accuracy: 0.5352 - loss: 1.2793 - val_accuracy: 0.2734 - val_loss: 2.1068
Epoch 170/200
16/16 - 1s - 51ms/step - accuracy: 0.5293 - loss: 1.3161 - val_accuracy: 0.2734 - val_loss: 2.1068
Epoch 171/200
16/16 - 1s - 51ms/step - accuracy: 0.5371 - loss: 1.2862 - val_accuracy: 0.2578 - val_loss: 2.1149
Epoch 172/200
16/16 - 1s - 51ms/step - accuracy: 0.5176 - loss: 1.2982 - val_accuracy: 0.2656 - val_loss: 2.1196
Epoch 173/200
16/16 - 1s - 50ms/step - accuracy: 0.5684 - loss: 1.2120 - val_accuracy: 0.2500 - val_loss: 2.1247
Epoch 174/200
16/16 - 1s - 51ms/step - accuracy: 0.5430 - loss: 1.2719 - val_accuracy: 0.2344 - val_loss: 2.1503
Epoch 175/200
16/16 - 1s - 50ms/step - accuracy: 0.5254 - loss: 1.2393 - val_accuracy: 0.2188 - val_loss: 2.1432
Epoch 176/200
16/16 - 1s - 52ms/step - accuracy: 0.5625 - loss: 1.2174 - val_accuracy: 0.2344 - val_loss: 2.1114
Epoch 177/200
16/16 - 1s - 53ms/step - accuracy: 0.5176 - loss: 1.2770 - val_accuracy: 0.2344 - val_loss: 2.1000
Epoch 178/200
16/16 - 1s - 50ms/step - accuracy: 0.5547 - loss: 1.2224 - val_accuracy: 0.2422 - val_loss: 2.0934
```

```
Epoch 179/200
16/16 - 1s - 50ms/step - accuracy: 0.5742 - loss: 1.1978 - val_accuracy: 0.2422 - val_loss: 2.1144
Epoch 180/200
16/16 - 1s - 51ms/step - accuracy: 0.5430 - loss: 1.2540 - val_accuracy: 0.2422 - val_loss: 2.1376
Epoch 181/200
16/16 - 1s - 51ms/step - accuracy: 0.5352 - loss: 1.2340 - val_accuracy: 0.2344 - val_loss: 2.1264
Epoch 182/200
16/16 - 1s - 51ms/step - accuracy: 0.5430 - loss: 1.2801 - val_accuracy: 0.2344 - val_loss: 2.1180
Epoch 183/200
16/16 - 1s - 51ms/step - accuracy: 0.5664 - loss: 1.2427 - val_accuracy: 0.2422 - val_loss: 2.1185
Epoch 184/200
16/16 - 1s - 50ms/step - accuracy: 0.5215 - loss: 1.3169 - val_accuracy: 0.2422 - val_loss: 2.1323
Epoch 185/200
16/16 - 1s - 52ms/step - accuracy: 0.5488 - loss: 1.2951 - val_accuracy: 0.2344 - val_loss: 2.1323
Epoch 186/200
16/16 - 1s - 50ms/step - accuracy: 0.5781 - loss: 1.2001 - val_accuracy: 0.2422 - val_loss: 2.1341
Epoch 187/200
16/16 - 1s - 50ms/step - accuracy: 0.4922 - loss: 1.3068 - val_accuracy: 0.2344 - val_loss: 2.1328
Epoch 188/200
16/16 - 1s - 51ms/step - accuracy: 0.5137 - loss: 1.3152 - val_accuracy: 0.2422 - val_loss: 2.1303
Epoch 189/200
16/16 - 1s - 51ms/step - accuracy: 0.5234 - loss: 1.2624 - val_accuracy: 0.2422 - val_loss: 2.1439
Epoch 190/200
16/16 - 1s - 51ms/step - accuracy: 0.5352 - loss: 1.2736 - val_accuracy: 0.2344 - val_loss: 2.1754
Epoch 191/200
16/16 - 1s - 50ms/step - accuracy: 0.5293 - loss: 1.2794 - val_accuracy: 0.2188 - val_loss: 2.1797
Epoch 192/200
16/16 - 1s - 50ms/step - accuracy: 0.5137 - loss: 1.2737 - val_accuracy: 0.2266 - val_loss: 2.1824
Epoch 193/200
16/16 - 1s - 50ms/step - accuracy: 0.5781 - loss: 1.1481 - val_accuracy: 0.2188 - val_loss: 2.1917
Epoch 194/200
16/16 - 1s - 51ms/step - accuracy: 0.5586 - loss: 1.2127 - val_accuracy: 0.2266 - val_loss: 2.1883
Epoch 195/200
16/16 - 1s - 51ms/step - accuracy: 0.5234 - loss: 1.3147 - val_accuracy: 0.2422 - val_loss: 2.1817
Epoch 196/200
16/16 - 1s - 50ms/step - accuracy: 0.5430 - loss: 1.2505 - val_accuracy: 0.2266 - val_loss: 2.1929
Epoch 197/200
16/16 - 1s - 50ms/step - accuracy: 0.5508 - loss: 1.2722 - val_accuracy: 0.2344 - val_loss: 2.1821
Epoch 198/200
16/16 - 1s - 50ms/step - accuracy: 0.5664 - loss: 1.1416 - val_accuracy: 0.2344 - val_loss: 2.1807
Epoch 199/200
16/16 - 1s - 51ms/step - accuracy: 0.5625 - loss: 1.2043 - val_accuracy: 0.2344 - val_loss: 2.1617
Epoch 200/200
16/16 - 1s - 50ms/step - accuracy: 0.5352 - loss: 1.2705 - val_accuracy: 0.2109 - val_loss: 2.1584
```

Model: "sequential_2"

Layer (type)	**Output Shape**	**Param #**
sequential (Sequential)	(None, 96, 96, 3)	0
lambda (Lambda)	(None, 96, 96, 3)	0
mobilenet_1.00_224 (Functional)	(None, 3, 3, 1024)	3,228,864
batch_normalization (BatchNormalization)	(None, 3, 3, 1024)	4,096
global_average_pooling2d (GlobalAveragePooling2D)	(None, 1024)	0
dropout (Dropout)	(None, 1024)	0
dense (Dense)	(None, 8)	8,200

Total params: 3,241,160 (12.36 MB)

Trainable params: 1,598,472 (6.10 MB)

Non-trainable params: 1,642,688 (6.27 MB)

```
Epoch 200/300
16/16 - 4s - 225ms/step - accuracy: 0.4668 - loss: 1.4426 - val_accuracy: 0.2500 - val_loss: 2.2293
Epoch 201/300
16/16 - 1s - 59ms/step - accuracy: 0.5078 - loss: 1.3902 - val_accuracy: 0.2578 - val_loss: 2.1483
Epoch 202/300
16/16 - 1s - 61ms/step - accuracy: 0.4668 - loss: 1.4285 - val_accuracy: 0.2344 - val_loss: 2.1536
Epoch 203/300
16/16 - 1s - 62ms/step - accuracy: 0.4609 - loss: 1.4290 - val_accuracy: 0.2500 - val_loss: 2.1310
Epoch 204/300
16/16 - 1s - 61ms/step - accuracy: 0.5098 - loss: 1.3578 - val_accuracy: 0.2578 - val_loss: 2.0734
Epoch 205/300
16/16 - 1s - 60ms/step - accuracy: 0.4824 - loss: 1.3857 - val_accuracy: 0.2656 - val_loss: 2.1124
Epoch 206/300
16/16 - 1s - 63ms/step - accuracy: 0.5352 - loss: 1.2870 - val_accuracy: 0.2656 - val_loss: 2.1593
```

```
Epoch 207/300
16/16 - 1s - 65ms/step - accuracy: 0.5391 - loss: 1.2762 - val_accuracy: 0.2656 - val_loss: 2.1205
Epoch 208/300
16/16 - 1s - 64ms/step - accuracy: 0.4941 - loss: 1.3373 - val_accuracy: 0.2734 - val_loss: 2.1027
Epoch 209/300
16/16 - 1s - 64ms/step - accuracy: 0.5215 - loss: 1.3081 - val_accuracy: 0.2656 - val_loss: 2.0820
Epoch 210/300
16/16 - 1s - 64ms/step - accuracy: 0.5273 - loss: 1.2812 - val_accuracy: 0.2344 - val_loss: 2.1273
Epoch 211/300
16/16 - 1s - 63ms/step - accuracy: 0.5430 - loss: 1.2469 - val_accuracy: 0.2500 - val_loss: 2.1035
Epoch 212/300
16/16 - 1s - 64ms/step - accuracy: 0.5898 - loss: 1.1971 - val_accuracy: 0.2344 - val_loss: 2.0765
Epoch 213/300
16/16 - 1s - 77ms/step - accuracy: 0.5391 - loss: 1.2608 - val_accuracy: 0.2812 - val_loss: 2.0072
Epoch 214/300
16/16 - 1s - 69ms/step - accuracy: 0.5469 - loss: 1.2231 - val_accuracy: 0.2734 - val_loss: 2.0455
Epoch 215/300
16/16 - 1s - 75ms/step - accuracy: 0.5742 - loss: 1.2132 - val_accuracy: 0.3047 - val_loss: 2.0078
Epoch 216/300
16/16 - 1s - 63ms/step - accuracy: 0.5664 - loss: 1.1477 - val_accuracy: 0.2578 - val_loss: 1.9902
Epoch 217/300
16/16 - 1s - 62ms/step - accuracy: 0.5742 - loss: 1.1873 - val_accuracy: 0.2500 - val_loss: 2.0242
Epoch 218/300
16/16 - 1s - 65ms/step - accuracy: 0.5156 - loss: 1.2386 - val_accuracy: 0.2891 - val_loss: 2.0587
Epoch 219/300
16/16 - 1s - 64ms/step - accuracy: 0.5664 - loss: 1.1942 - val_accuracy: 0.2656 - val_loss: 2.0897
Epoch 220/300
16/16 - 1s - 63ms/step - accuracy: 0.5723 - loss: 1.1065 - val_accuracy: 0.2578 - val_loss: 2.1136
Epoch 221/300
16/16 - 1s - 64ms/step - accuracy: 0.5605 - loss: 1.1982 - val_accuracy: 0.2578 - val_loss: 2.0352
Epoch 222/300
16/16 - 1s - 66ms/step - accuracy: 0.5586 - loss: 1.0977 - val_accuracy: 0.2812 - val_loss: 2.0387
Epoch 223/300
16/16 - 1s - 67ms/step - accuracy: 0.5801 - loss: 1.1431 - val_accuracy: 0.2734 - val_loss: 2.0395
Epoch 224/300
16/16 - 1s - 69ms/step - accuracy: 0.6035 - loss: 1.0518 - val_accuracy: 0.2812 - val_loss: 2.0124
Epoch 225/300
16/16 - 1s - 71ms/step - accuracy: 0.5762 - loss: 1.1648 - val_accuracy: 0.2734 - val_loss: 2.0045
Epoch 226/300
16/16 - 1s - 63ms/step - accuracy: 0.5957 - loss: 1.0972 - val_accuracy: 0.2734 - val_loss: 1.9938
Epoch 227/300
16/16 - 1s - 64ms/step - accuracy: 0.5742 - loss: 1.1711 - val_accuracy: 0.2969 - val_loss: 1.9607
Epoch 228/300
16/16 - 1s - 65ms/step - accuracy: 0.6211 - loss: 1.0667 - val_accuracy: 0.2812 - val_loss: 1.9671
Epoch 229/300
16/16 - 1s - 66ms/step - accuracy: 0.6230 - loss: 1.0581 - val_accuracy: 0.2812 - val_loss: 1.9513
Epoch 230/300
16/16 - 1s - 64ms/step - accuracy: 0.6289 - loss: 0.9988 - val_accuracy: 0.2734 - val_loss: 1.9753
Epoch 231/300
16/16 - 1s - 63ms/step - accuracy: 0.6016 - loss: 1.0771 - val_accuracy: 0.2656 - val_loss: 1.9946
Epoch 232/300
16/16 - 1s - 66ms/step - accuracy: 0.6133 - loss: 1.0541 - val_accuracy: 0.2656 - val_loss: 2.0111
Epoch 233/300
16/16 - 1s - 64ms/step - accuracy: 0.6523 - loss: 1.0368 - val_accuracy: 0.2734 - val_loss: 2.0007
Epoch 234/300
16/16 - 1s - 64ms/step - accuracy: 0.6387 - loss: 1.0263 - val_accuracy: 0.2656 - val_loss: 1.9861
Epoch 235/300
16/16 - 1s - 63ms/step - accuracy: 0.6211 - loss: 1.0570 - val_accuracy: 0.2969 - val_loss: 2.0025
Epoch 236/300
16/16 - 1s - 65ms/step - accuracy: 0.6211 - loss: 1.0529 - val_accuracy: 0.2969 - val_loss: 2.0040
Epoch 237/300
16/16 - 1s - 68ms/step - accuracy: 0.6562 - loss: 1.0062 - val_accuracy: 0.3203 - val_loss: 1.9936
Epoch 238/300
16/16 - 1s - 74ms/step - accuracy: 0.5977 - loss: 1.0652 - val_accuracy: 0.2656 - val_loss: 2.0063
Epoch 239/300
16/16 - 1s - 62ms/step - accuracy: 0.6406 - loss: 1.0258 - val_accuracy: 0.2891 - val_loss: 2.0339
Epoch 240/300
16/16 - 1s - 64ms/step - accuracy: 0.6250 - loss: 1.0276 - val_accuracy: 0.2969 - val_loss: 2.0292
Epoch 241/300
16/16 - 1s - 63ms/step - accuracy: 0.6406 - loss: 0.9689 - val_accuracy: 0.2812 - val_loss: 2.0142
Epoch 242/300
16/16 - 1s - 63ms/step - accuracy: 0.6582 - loss: 0.9585 - val_accuracy: 0.2812 - val_loss: 2.0400
Epoch 243/300
16/16 - 1s - 64ms/step - accuracy: 0.6113 - loss: 1.0320 - val_accuracy: 0.2891 - val_loss: 2.0309
Epoch 244/300
16/16 - 1s - 64ms/step - accuracy: 0.6484 - loss: 0.9783 - val_accuracy: 0.2891 - val_loss: 2.0253
Epoch 245/300
16/16 - 1s - 62ms/step - accuracy: 0.6504 - loss: 0.9836 - val_accuracy: 0.2812 - val_loss: 1.9916
Epoch 246/300
16/16 - 1s - 62ms/step - accuracy: 0.6328 - loss: 1.0036 - val_accuracy: 0.2734 - val_loss: 1.9636
Epoch 247/300
16/16 - 1s - 63ms/step - accuracy: 0.6738 - loss: 0.8883 - val_accuracy: 0.2969 - val_loss: 1.9909
Epoch 248/300
```

```
16/16 - 1s - 62ms/step - accuracy: 0.6445 - loss: 0.9523 - val_accuracy: 0.3047 - val_loss: 1.9905
Epoch 249/300
16/16 - 1s - 64ms/step - accuracy: 0.6465 - loss: 0.9948 - val_accuracy: 0.2969 - val_loss: 2.0153
Epoch 250/300
16/16 - 1s - 62ms/step - accuracy: 0.6602 - loss: 0.9037 - val_accuracy: 0.3203 - val_loss: 2.0169
Epoch 251/300
16/16 - 1s - 63ms/step - accuracy: 0.6699 - loss: 0.9861 - val_accuracy: 0.2969 - val_loss: 1.9914
Epoch 252/300
16/16 - 1s - 63ms/step - accuracy: 0.6699 - loss: 0.9464 - val_accuracy: 0.2969 - val_loss: 1.9986
Epoch 253/300
16/16 - 1s - 62ms/step - accuracy: 0.6543 - loss: 0.9430 - val_accuracy: 0.2812 - val_loss: 2.0207
Epoch 254/300
16/16 - 1s - 64ms/step - accuracy: 0.6758 - loss: 0.8984 - val_accuracy: 0.2812 - val_loss: 2.0373
Epoch 255/300
16/16 - 1s - 63ms/step - accuracy: 0.6582 - loss: 0.9373 - val_accuracy: 0.2734 - val_loss: 2.0590
Epoch 256/300
16/16 - 1s - 63ms/step - accuracy: 0.6387 - loss: 0.9801 - val_accuracy: 0.2656 - val_loss: 2.0675
Epoch 257/300
16/16 - 1s - 62ms/step - accuracy: 0.6641 - loss: 0.9034 - val_accuracy: 0.2812 - val_loss: 2.0511
Epoch 258/300
16/16 - 1s - 63ms/step - accuracy: 0.6855 - loss: 0.8894 - val_accuracy: 0.2734 - val_loss: 2.0660
Epoch 259/300
16/16 - 1s - 66ms/step - accuracy: 0.6641 - loss: 0.9025 - val_accuracy: 0.2812 - val_loss: 2.0572
Epoch 260/300
16/16 - 1s - 62ms/step - accuracy: 0.6738 - loss: 0.9381 - val_accuracy: 0.2656 - val_loss: 2.0353
Epoch 261/300
16/16 - 1s - 61ms/step - accuracy: 0.6680 - loss: 0.9471 - val_accuracy: 0.2812 - val_loss: 2.0067
Epoch 262/300
16/16 - 1s - 65ms/step - accuracy: 0.6934 - loss: 0.8521 - val_accuracy: 0.3047 - val_loss: 1.9815
Epoch 263/300
16/16 - 1s - 62ms/step - accuracy: 0.6797 - loss: 0.9157 - val_accuracy: 0.2891 - val_loss: 1.9850
Epoch 264/300
16/16 - 1s - 65ms/step - accuracy: 0.6934 - loss: 0.8885 - val_accuracy: 0.3047 - val_loss: 1.9933
Epoch 265/300
16/16 - 1s - 62ms/step - accuracy: 0.6973 - loss: 0.8647 - val_accuracy: 0.2969 - val_loss: 1.9995
Epoch 266/300
16/16 - 1s - 62ms/step - accuracy: 0.6738 - loss: 0.9419 - val_accuracy: 0.2812 - val_loss: 1.9937
Epoch 267/300
16/16 - 1s - 64ms/step - accuracy: 0.6641 - loss: 0.8965 - val_accuracy: 0.2578 - val_loss: 1.9883
Epoch 268/300
16/16 - 1s - 62ms/step - accuracy: 0.6680 - loss: 0.9095 - val_accuracy: 0.2891 - val_loss: 1.9897
Epoch 269/300
16/16 - 1s - 65ms/step - accuracy: 0.7070 - loss: 0.8395 - val_accuracy: 0.2969 - val_loss: 1.9943
Epoch 270/300
16/16 - 1s - 64ms/step - accuracy: 0.7109 - loss: 0.8654 - val_accuracy: 0.2891 - val_loss: 2.0171
Epoch 271/300
16/16 - 1s - 63ms/step - accuracy: 0.6914 - loss: 0.8433 - val_accuracy: 0.2969 - val_loss: 2.0123
Epoch 272/300
16/16 - 1s - 67ms/step - accuracy: 0.6973 - loss: 0.8548 - val_accuracy: 0.3125 - val_loss: 2.0195
Epoch 273/300
16/16 - 1s - 64ms/step - accuracy: 0.6914 - loss: 0.8645 - val_accuracy: 0.3047 - val_loss: 2.0123
Epoch 274/300
16/16 - 1s - 64ms/step - accuracy: 0.7012 - loss: 0.8634 - val_accuracy: 0.2969 - val_loss: 2.0169
Epoch 275/300
16/16 - 1s - 62ms/step - accuracy: 0.7324 - loss: 0.7823 - val_accuracy: 0.3047 - val_loss: 2.0244
Epoch 276/300
16/16 - 1s - 61ms/step - accuracy: 0.7207 - loss: 0.8607 - val_accuracy: 0.3203 - val_loss: 2.0340
Epoch 277/300
16/16 - 1s - 63ms/step - accuracy: 0.7051 - loss: 0.7907 - val_accuracy: 0.3281 - val_loss: 2.0044
Epoch 278/300
16/16 - 1s - 62ms/step - accuracy: 0.7168 - loss: 0.8606 - val_accuracy: 0.3203 - val_loss: 1.9939
Epoch 279/300
16/16 - 1s - 64ms/step - accuracy: 0.7129 - loss: 0.8283 - val_accuracy: 0.3281 - val_loss: 1.9888
Epoch 280/300
16/16 - 1s - 63ms/step - accuracy: 0.7500 - loss: 0.7795 - val_accuracy: 0.3125 - val_loss: 1.9719
Epoch 281/300
16/16 - 1s - 64ms/step - accuracy: 0.7090 - loss: 0.8240 - val_accuracy: 0.2969 - val_loss: 1.9808
Epoch 282/300
16/16 - 1s - 63ms/step - accuracy: 0.6641 - loss: 0.8868 - val_accuracy: 0.3203 - val_loss: 1.9769
Epoch 283/300
16/16 - 1s - 63ms/step - accuracy: 0.6953 - loss: 0.8241 - val_accuracy: 0.3125 - val_loss: 1.9683
Epoch 284/300
16/16 - 1s - 65ms/step - accuracy: 0.6484 - loss: 0.9539 - val_accuracy: 0.3125 - val_loss: 1.9778
Epoch 285/300
16/16 - 1s - 63ms/step - accuracy: 0.7578 - loss: 0.7510 - val_accuracy: 0.3125 - val_loss: 1.9705
Epoch 286/300
16/16 - 1s - 65ms/step - accuracy: 0.6934 - loss: 0.8265 - val_accuracy: 0.3203 - val_loss: 1.9615
Epoch 287/300
16/16 - 1s - 64ms/step - accuracy: 0.7051 - loss: 0.8157 - val_accuracy: 0.3047 - val_loss: 1.9560
Epoch 288/300
16/16 - 1s - 65ms/step - accuracy: 0.7148 - loss: 0.8111 - val_accuracy: 0.2812 - val_loss: 1.9656
Epoch 289/300
16/16 - 1s - 66ms/step - accuracy: 0.7148 - loss: 0.7945 - val_accuracy: 0.3047 - val_loss: 1.9728
```

```
Epoch 290/300
16/16 - 1s - 63ms/step - accuracy: 0.7109 - loss: 0.7863 - val_accuracy: 0.2969 - val_loss: 1.9689
Epoch 291/300
16/16 - 1s - 64ms/step - accuracy: 0.7188 - loss: 0.8203 - val_accuracy: 0.2969 - val_loss: 1.9682
Epoch 292/300
16/16 - 1s - 65ms/step - accuracy: 0.7402 - loss: 0.8052 - val_accuracy: 0.2891 - val_loss: 1.9609
Epoch 293/300
16/16 - 1s - 73ms/step - accuracy: 0.7070 - loss: 0.8323 - val_accuracy: 0.2969 - val_loss: 1.9619
Epoch 294/300
16/16 - 1s - 69ms/step - accuracy: 0.7344 - loss: 0.7888 - val_accuracy: 0.2891 - val_loss: 1.9667
Epoch 295/300
16/16 - 1s - 64ms/step - accuracy: 0.7168 - loss: 0.7906 - val_accuracy: 0.2891 - val_loss: 1.9921
Epoch 296/300
16/16 - 1s - 65ms/step - accuracy: 0.7148 - loss: 0.8282 - val_accuracy: 0.3047 - val_loss: 2.0029
Epoch 297/300
16/16 - 1s - 64ms/step - accuracy: 0.7344 - loss: 0.7472 - val_accuracy: 0.3047 - val_loss: 2.0070
Epoch 298/300
16/16 - 1s - 62ms/step - accuracy: 0.7363 - loss: 0.7708 - val_accuracy: 0.3047 - val_loss: 1.9929
Epoch 299/300
16/16 - 1s - 65ms/step - accuracy: 0.7441 - loss: 0.7773 - val_accuracy: 0.3047 - val_loss: 1.9982
Epoch 300/300
16/16 - 1s - 63ms/step - accuracy: 0.7383 - loss: 0.7639 - val_accuracy: 0.3047 - val_loss: 2.0106
```

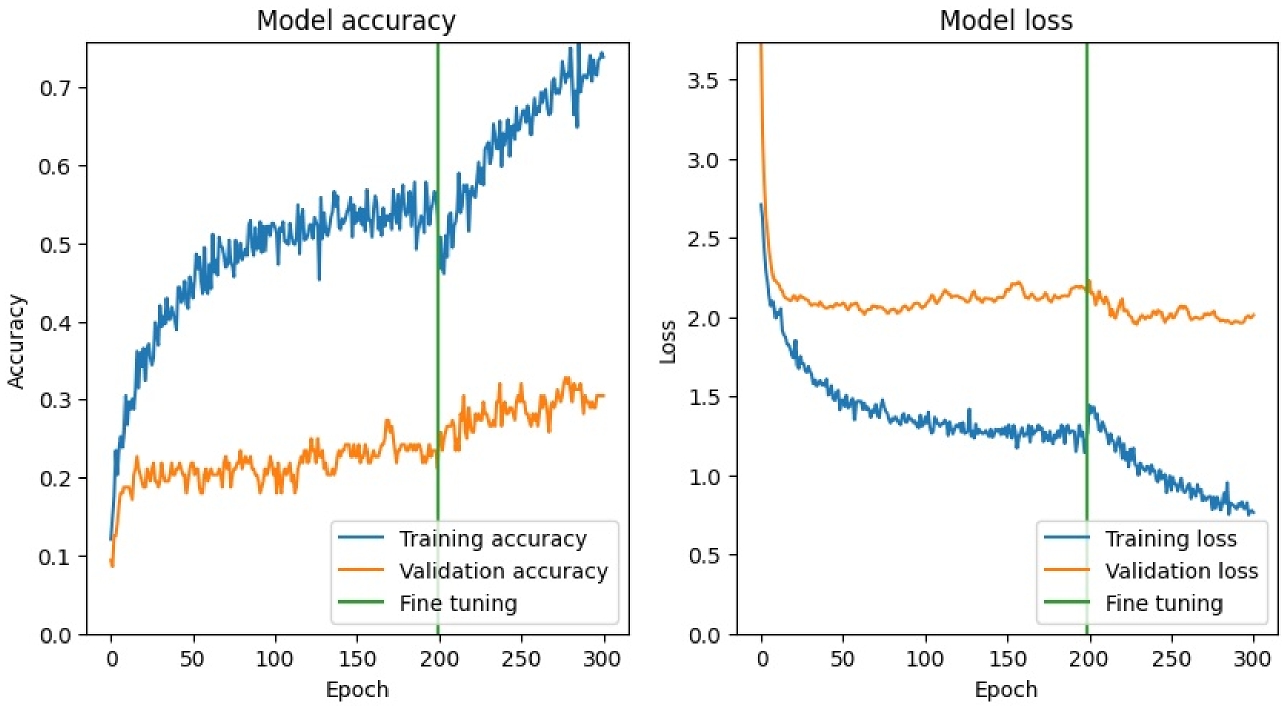
In [14]:

```python
def plot_performance(history, learning_rate=None, batch_size=None, finetune_epochs=None):
  plt.figure(figsize=(10,5))



  # Determine whether history is keras history or a dictionary to appropriately extract the history data
  if isinstance(history, keras.src.callbacks.history.History):
    history_data = history.history        # Extract the history dictionary
  else:
    history_data = history                # Assume it's already a dictionary

  # Accuracy of model training and validation vs training epoch
  plt.subplot(1,2,1)
  ylim_acc = [0, max(max(history_data['accuracy']),max(history_data['val_accuracy']))]
  plt.plot(history_data['accuracy'], label = 'Training accuracy')
  plt.plot(history_data['val_accuracy'], label = 'Validation accuracy')
  plt.ylim(ylim_acc)
  if finetune_epochs:
    plt.plot([finetune_epochs-1, finetune_epochs-1],plt.ylim(), label = 'Fine tuning')
  else:
    pass

  if learning_rate and batch_size:
    plt.title(f'Model accuracy \n lr = {learning_rate}, batch size = {batch_size}')
  else: plt.title('Model accuracy')

  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(loc='lower right')

  # Loss during model training and validation
  plt.subplot(1,2,2)
  ylim_loss = [0, max(max(history_data['loss']),max(history_data['val_loss']))]
  # print(len(history_data['loss']))
  plt.plot(history_data['loss'], label = 'Training loss')
  plt.plot(history_data['val_loss'], label = 'Validation loss')
  plt.ylim(ylim_loss)
  if finetune_epochs:
    plt.plot([finetune_epochs-1, finetune_epochs-1],plt.ylim(), label = 'Fine tuning')
  else:
    pass

  if learning_rate and batch_size:
    plt.title(f'Model loss \n lr = {learning_rate}, batch size = {batch_size}')
  else: plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(loc='lower right')
  plt.show()

  print(f"The model has a training accuracy of {history_data['accuracy'][-1]*100:.2f}%\n"
      f"The model has a validation accuracy of {history_data['val_accuracy'][-1]*100:.2f}%")
  return
```

In [15]:

```python
combined_history = {
    'accuracy': history_initial.history['accuracy'] + history_fine.history['accuracy'],
    'val_accuracy': history_initial.history['val_accuracy'] + history_fine.history['val_accuracy'],
    'loss': history_initial.history['loss'] + history_fine.history['loss'],
    'val_loss': history_initial.history['val_loss'] + history_fine.history['val_loss']
}
```

```
plot_performance(combined_history,finetune_epochs=initial_epochs)
```




```
The model has a training accuracy of 73.83%
The model has a validation accuracy of 30.47%
```

Model evaluation and prediction

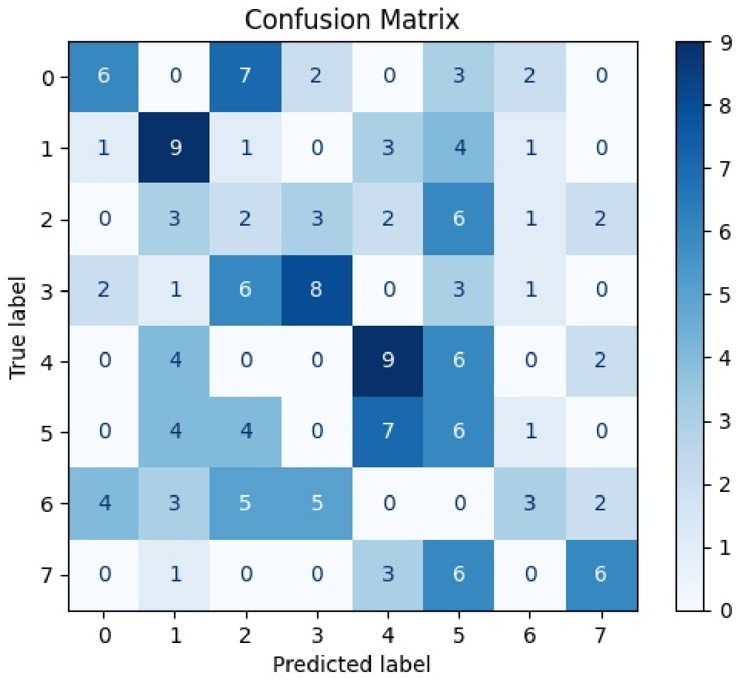
Model evaluation

In [16]:
```
test_loss, test_acc =  model.evaluate(image_test_array,label_test_array)
print(f"Test accuracy: {test_acc}\n"
      f"Test loss: {test_loss}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.2983 - loss: 2.1395
Test accuracy: 0.30625000596046495
Test loss: 2.011277437210083
```

In [17]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

prediction_array = np.argmax(model.predict(image_test_array), axis=1)

cm = confusion_matrix(label_test_array, prediction_array)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)  # You can change the color map as desired
plt.title("Confusion Matrix")
plt.show()
print(label_dict)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step
```




```
{0: 'anger', 1: 'contempt', 2: 'disgust', 3: 'fear', 4: 'happy', 5: 'neural', 6: 'sad', 7: 'suprise'}
```

Model prediction & visualization

In [18]:
```
# Sample random images and their indices
num_samples = 25                                                                         # number of sample:
num_rows = int(round(sqrt(num_samples))); num_cols = int(num_samples/num_rows)       # number of rows and column:
rand = random.randint(num_test,size = (num_samples))                                   # random index for ci

image_test_rand_array = image_test_array[rand]
label_test_rand_array = label_test_array[rand]
prediction_rand_array = np.argmax(model.predict(image_test_rand_array),axis=1)

plt.figure(figsize=(num_rows*2,num_cols*2))
# fig, axes1 = plt.subplots(num_rows,num_cols,figsize=(num_rows*2,num_cols*2))

for i in range(num_rows):
    for j in range(num_cols):
        index = i * num_cols + j
        plt.subplot(num_rows,num_cols,index+1)
        image = image_test_rand_array[index]  # Extract the image
        label = label_test_rand_array[index]  # Extract the label
        prediction = prediction_rand_array[index]

        # Original pictures (no augmentation layer applied)
        plt.axis("off")
        # Display the image
        plt.imshow(image)
        plt.title(f"Label: {label_dict[label]}\n"
                  f"Predict: {label_dict[prediction]}",
                  fontsize = 8)
```

```python
plt.tight_layout()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step
```