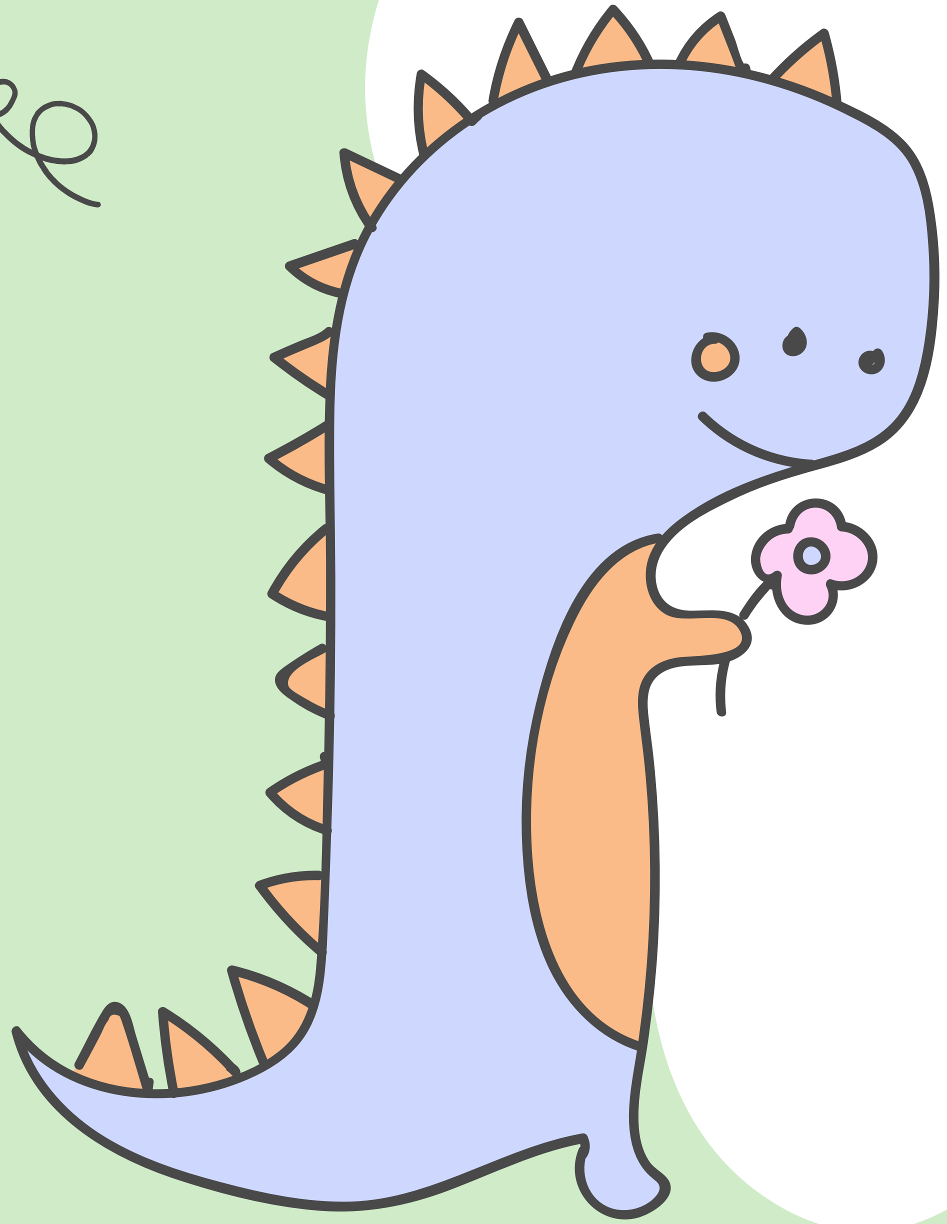
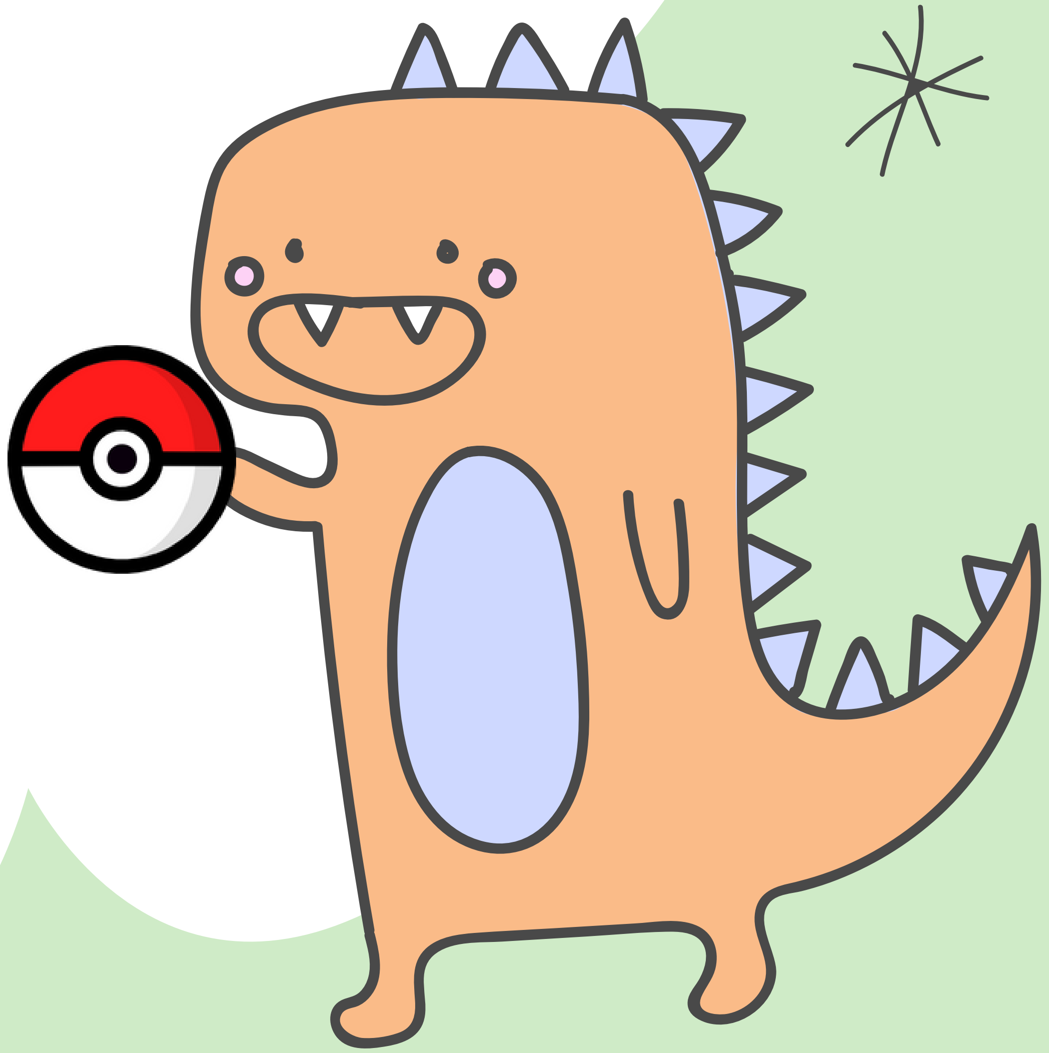


Pokémon™ classification

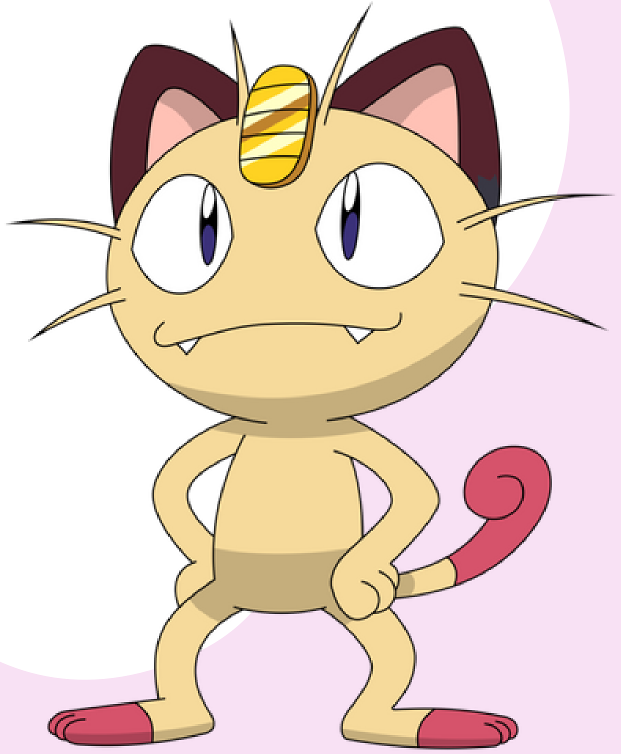


Team member

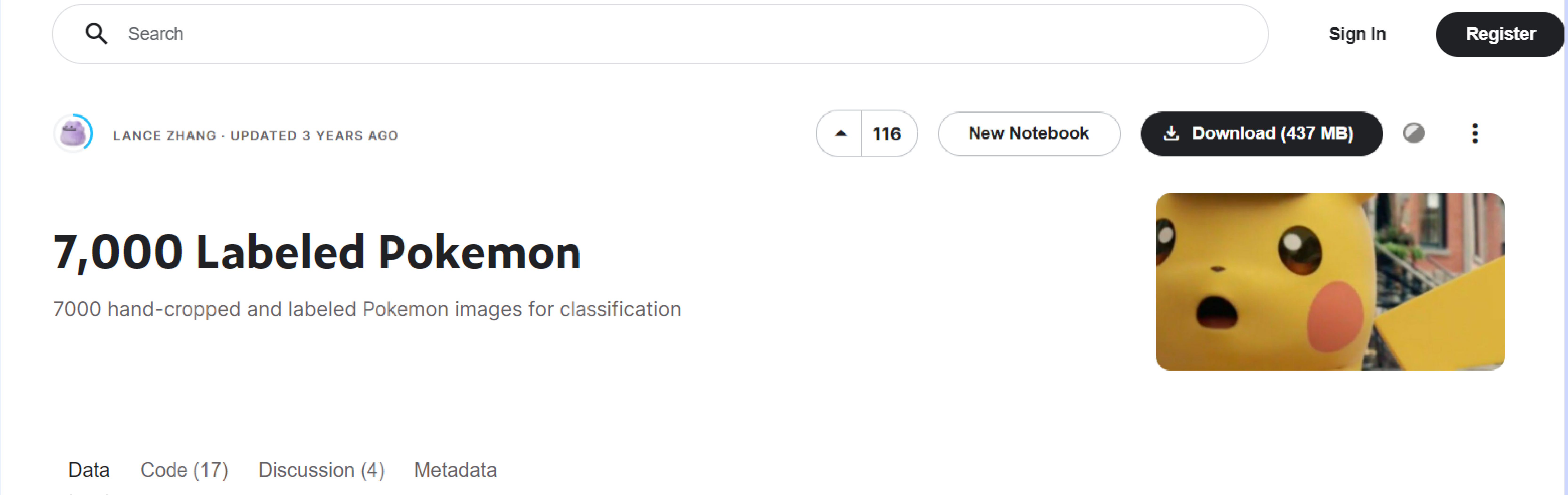
นายทีฆทัศน์ วงศ์สืบสันตติ

อาจารย์ที่ปรึกษา

ดร.ไพสิฐ ขันอาสา



DATASET



Search

Sign In

Register

LANCE ZHANG · UPDATED 3 YEARS AGO

116

New Notebook

Download (437 MB)

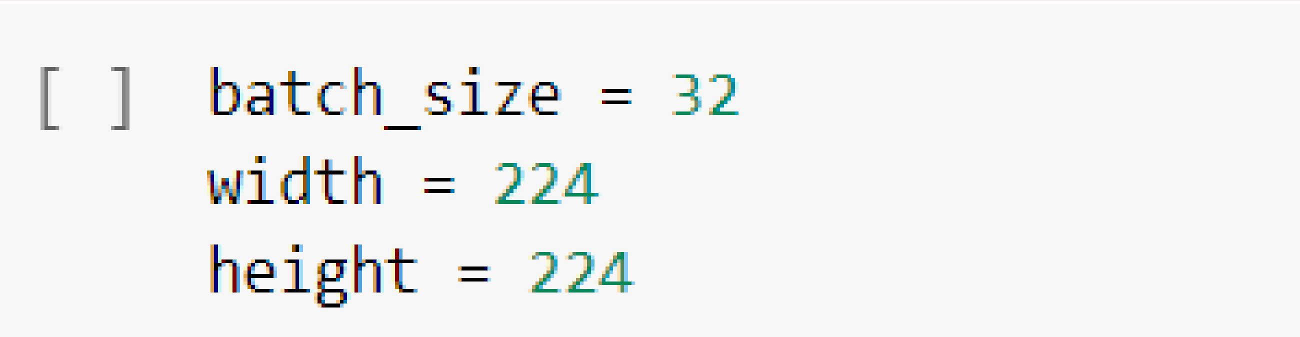
7,000 Labeled Pokemon

7000 hand-cropped and labeled Pokemon images for classification

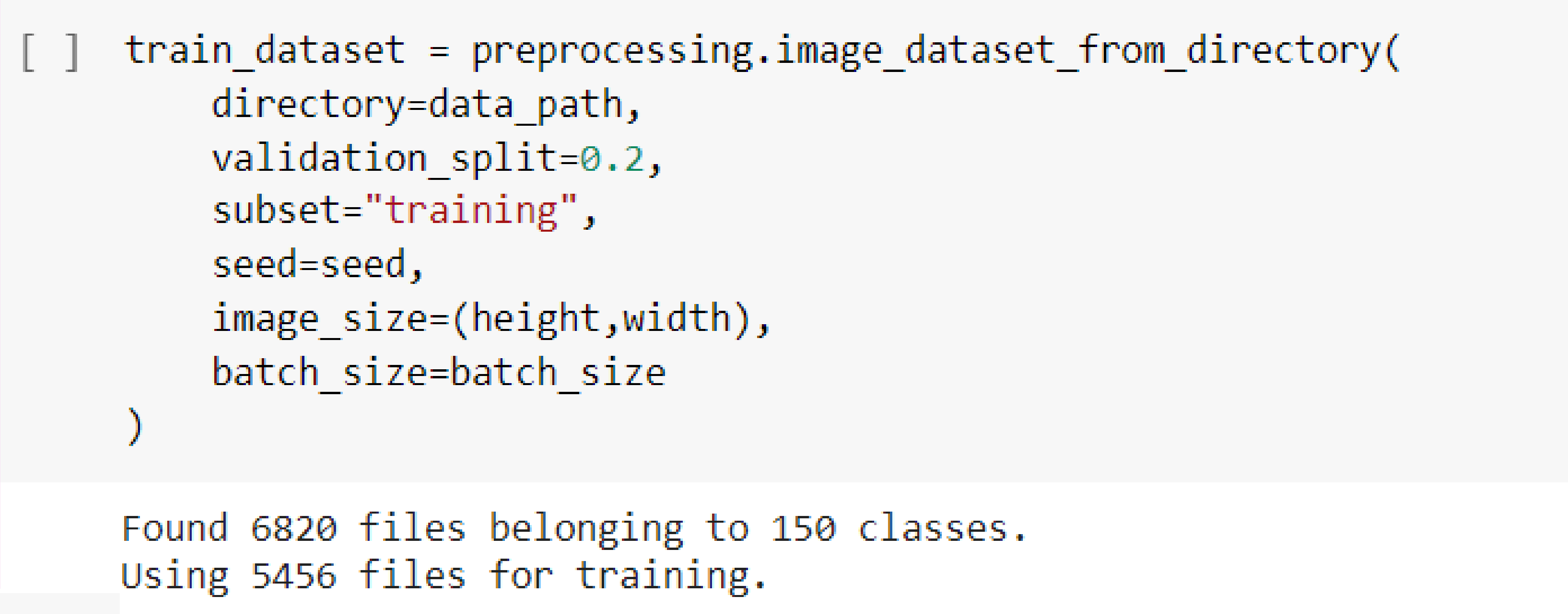
Data Code (17) Discussion (4) Metadata

The Dataset has 150 classes of pokemon

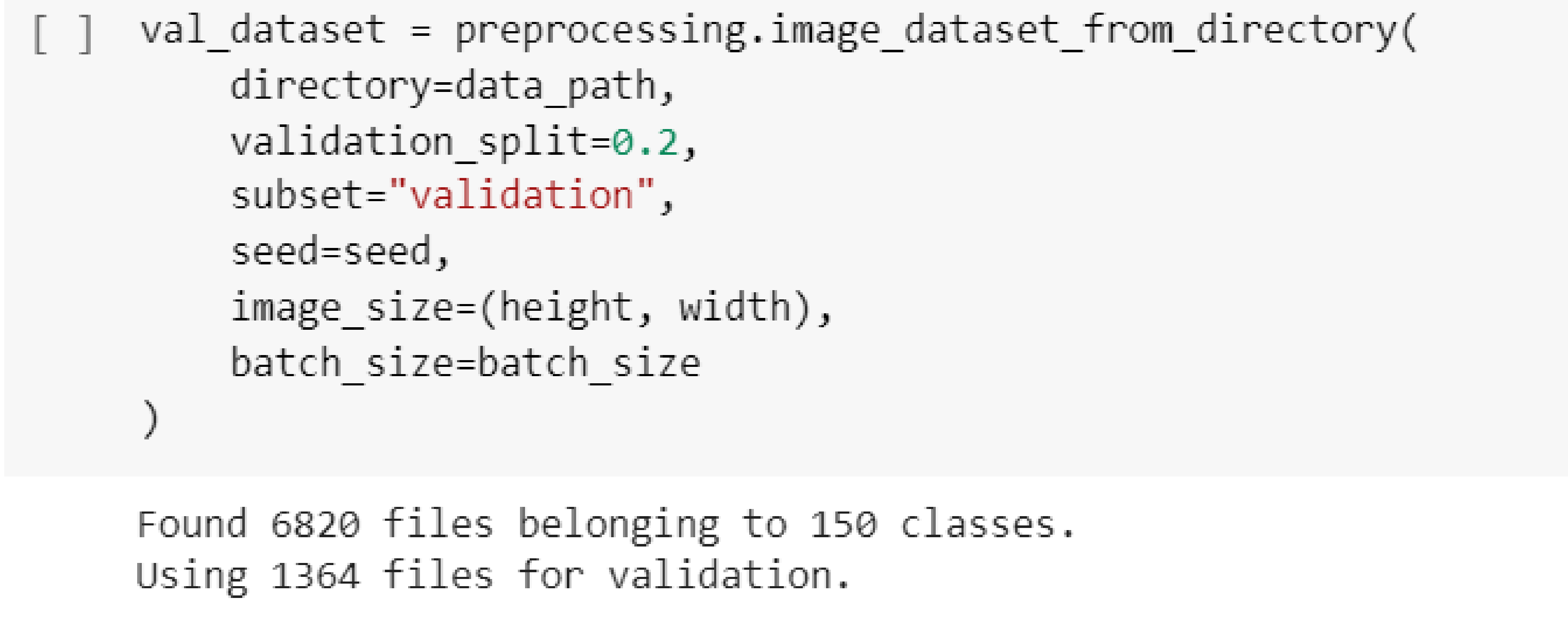
Manage Dataset



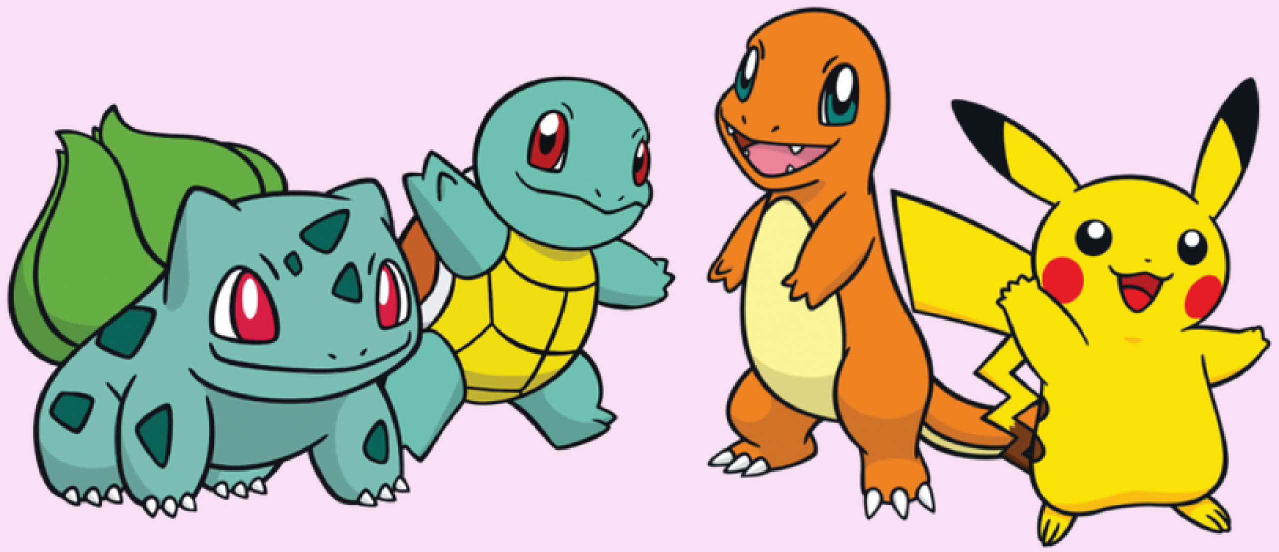
```
[ ]  batch_size = 32
     width = 224
     height = 224
```



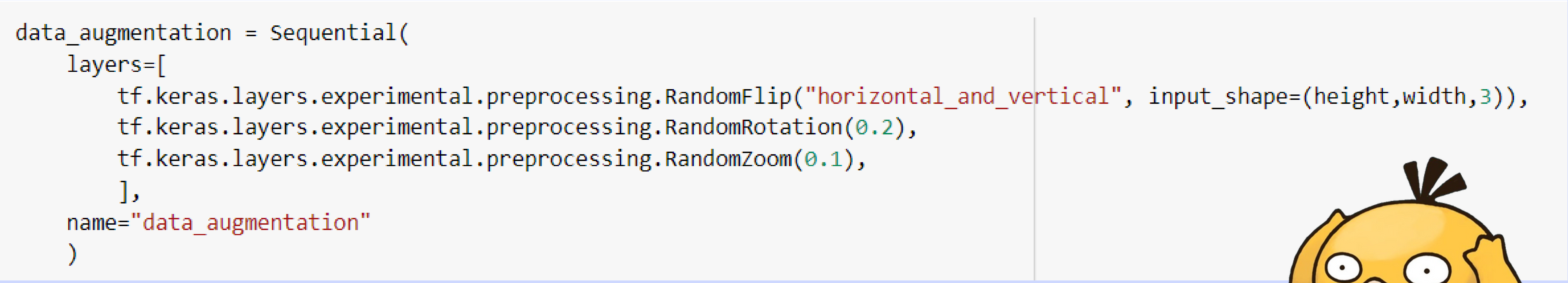
```
[ ]  train_dataset = preprocessing.image_dataset_from_directory(
         directory=data_path,
         validation_split=0.2,
         subset="training",
         seed=seed,
         image_size=(height,width),
         batch_size=batch_size
     )

     Found 6820 files belonging to 150 classes.
     Using 5456 files for training.
```



```
[ ]  val_dataset = preprocessing.image_dataset_from_directory(
         directory=data_path,
         validation_split=0.2,
         subset="validation",
         seed=seed,
         image_size=(height, width),
         batch_size=batch_size
     )

     Found 6820 files belonging to 150 classes.
     Using 1364 files for validation.
```

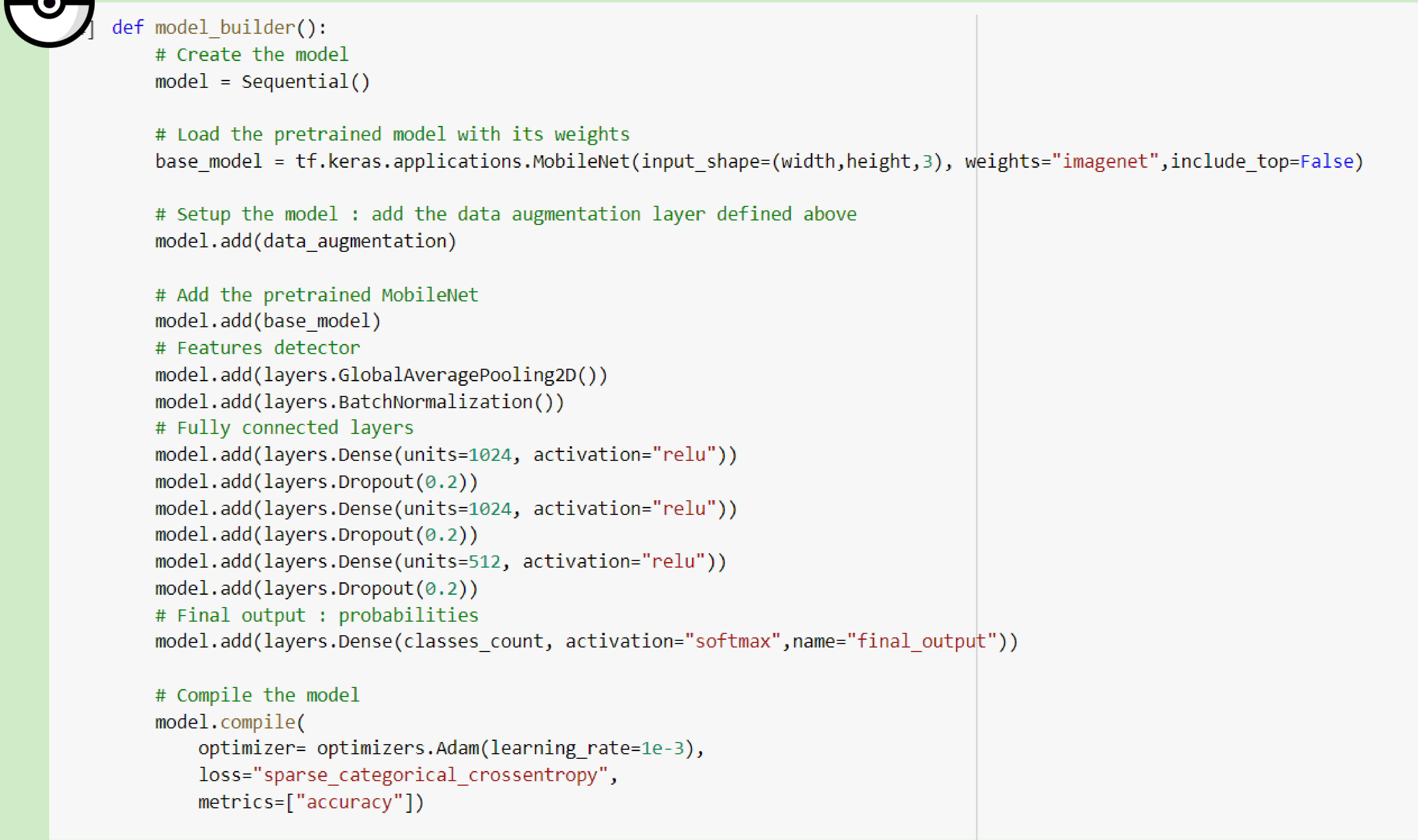


Data augmentation



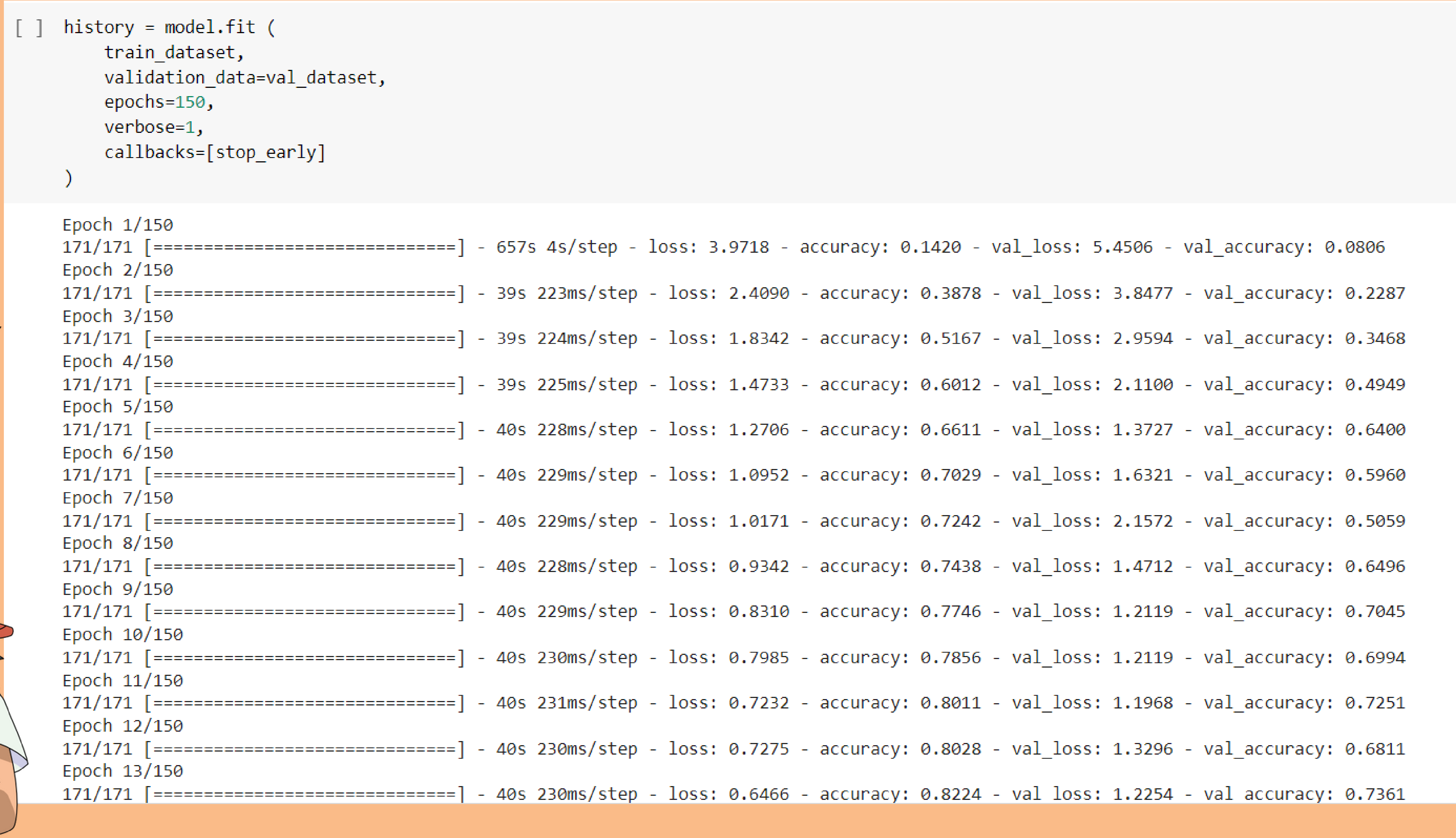
```
data_augmentation = Sequential(
    layers=[
        tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical", input_shape=(height,width,3)),
        tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
        tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
        ],
    name="data_augmentation"
    )
```

Model



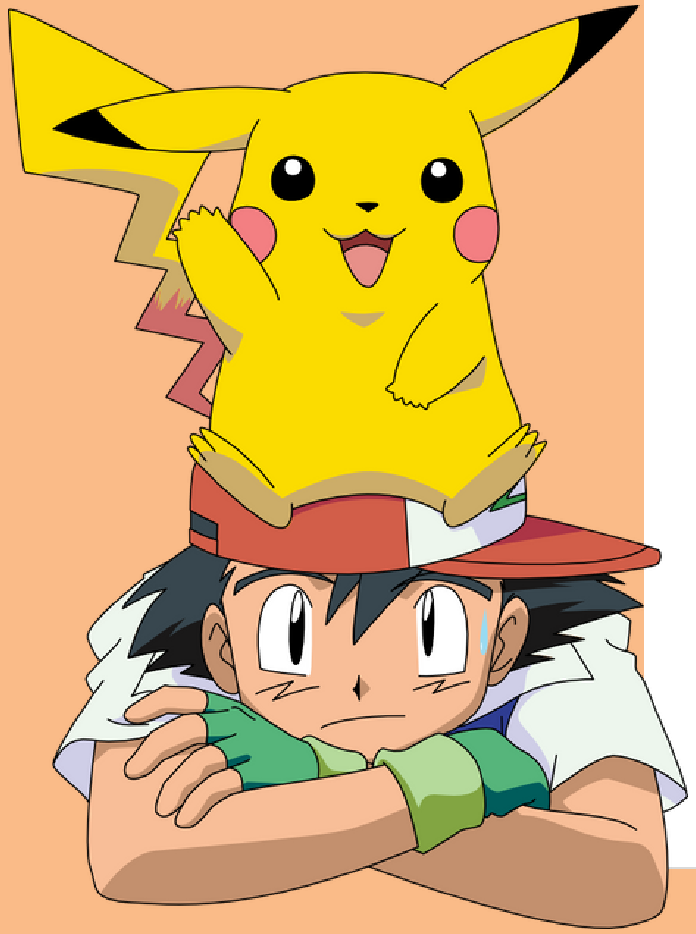
```
def model_builder():
    # Create the model
    model = Sequential()

    # Load the pretrained model with its weights
    base_model = tf.keras.applications.MobileNet(input_shape=(width,height,3), weights="imagenet",include_top=False)

    # Setup the model : add the data augmentation layer defined above
    model.add(data_augmentation)

    # Add the pretrained MobileNet
    model.add(base_model)
    # Features detector
    model.add(layers.GlobalAveragePooling2D())
    model.add(layers.BatchNormalization())
    # Fully connected layers
    model.add(layers.Dense(units=1024, activation="relu"))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(units=1024, activation="relu"))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(units=512, activation="relu"))
    model.add(layers.Dropout(0.2))
    # Final output : probabilities
    model.add(layers.Dense(classes_count, activation="softmax",name="final_output"))

    # Compile the model
    model.compile(
        optimizer= optimizers.Adam(learning_rate=1e-3),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"])
```

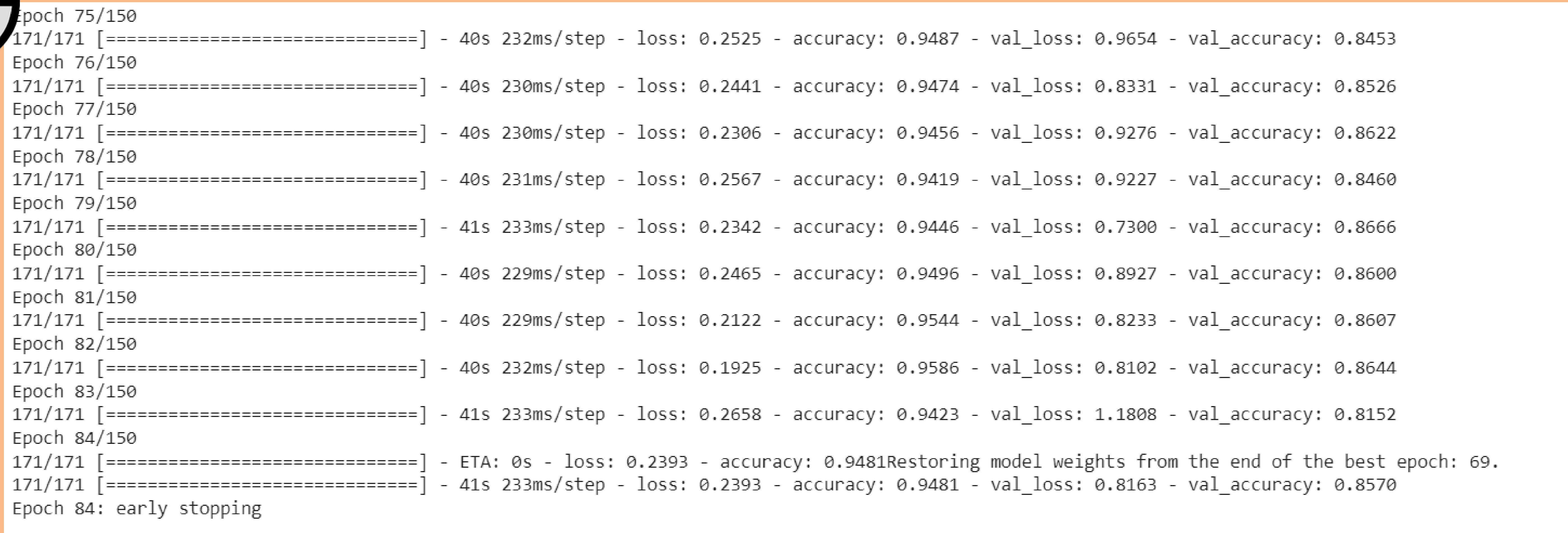
Train Model



```
[ ]  history = model.fit (
         train_dataset,
         validation_data=val_dataset,
         epochs=150,
         verbose=1,
         callbacks=[stop_early]
     )
```

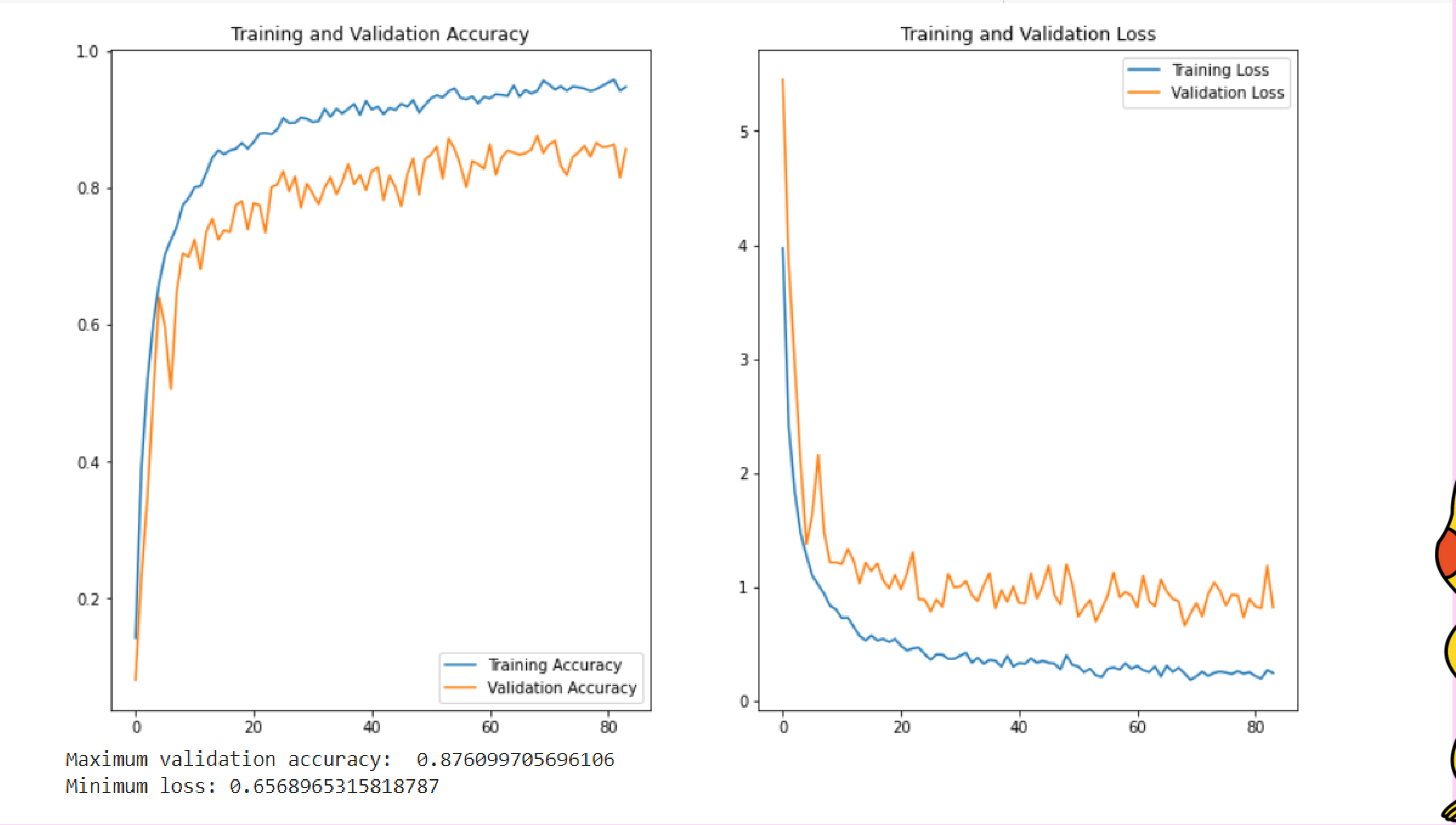
```
Epoch 1/150
171/171 [==============================] - 657s 4s/step - loss: 3.9718 - accuracy: 0.1420 - val_loss: 5.4506 - val_accuracy: 0.0806
Epoch 2/150
171/171 [==============================] - 39s 223ms/step - loss: 2.4090 - accuracy: 0.3878 - val_loss: 3.8477 - val_accuracy: 0.2287
Epoch 3/150
171/171 [==============================] - 39s 224ms/step - loss: 1.8342 - accuracy: 0.5167 - val_loss: 2.9594 - val_accuracy: 0.3468
Epoch 4/150
171/171 [==============================] - 39s 225ms/step - loss: 1.4733 - accuracy: 0.6012 - val_loss: 2.1100 - val_accuracy: 0.4949
Epoch 5/150
171/171 [==============================] - 40s 228ms/step - loss: 1.2706 - accuracy: 0.6611 - val_loss: 1.3727 - val_accuracy: 0.6400
Epoch 6/150
171/171 [==============================] - 40s 229ms/step - loss: 1.0952 - accuracy: 0.7029 - val_loss: 1.6321 - val_accuracy: 0.5960
Epoch 7/150
171/171 [==============================] - 40s 229ms/step - loss: 1.0171 - accuracy: 0.7242 - val_loss: 2.1572 - val_accuracy: 0.5059
Epoch 8/150
171/171 [==============================] - 40s 228ms/step - loss: 0.9342 - accuracy: 0.7438 - val_loss: 1.4712 - val_accuracy: 0.6496
Epoch 9/150
171/171 [==============================] - 40s 229ms/step - loss: 0.8310 - accuracy: 0.7746 - val_loss: 1.2119 - val_accuracy: 0.7045
Epoch 10/150
171/171 [==============================] - 40s 230ms/step - loss: 0.7985 - accuracy: 0.7856 - val_loss: 1.2119 - val_accuracy: 0.6994
Epoch 11/150
171/171 [==============================] - 40s 231ms/step - loss: 0.7232 - accuracy: 0.8011 - val_loss: 1.1968 - val_accuracy: 0.7251
Epoch 12/150
171/171 [==============================] - 40s 230ms/step - loss: 0.7275 - accuracy: 0.8028 - val_loss: 1.3296 - val_accuracy: 0.6811
Epoch 13/150
171/171 [==============================] - 40s 230ms/step - loss: 0.6466 - accuracy: 0.8224 - val_loss: 1.2254 - val_accuracy: 0.7361
```





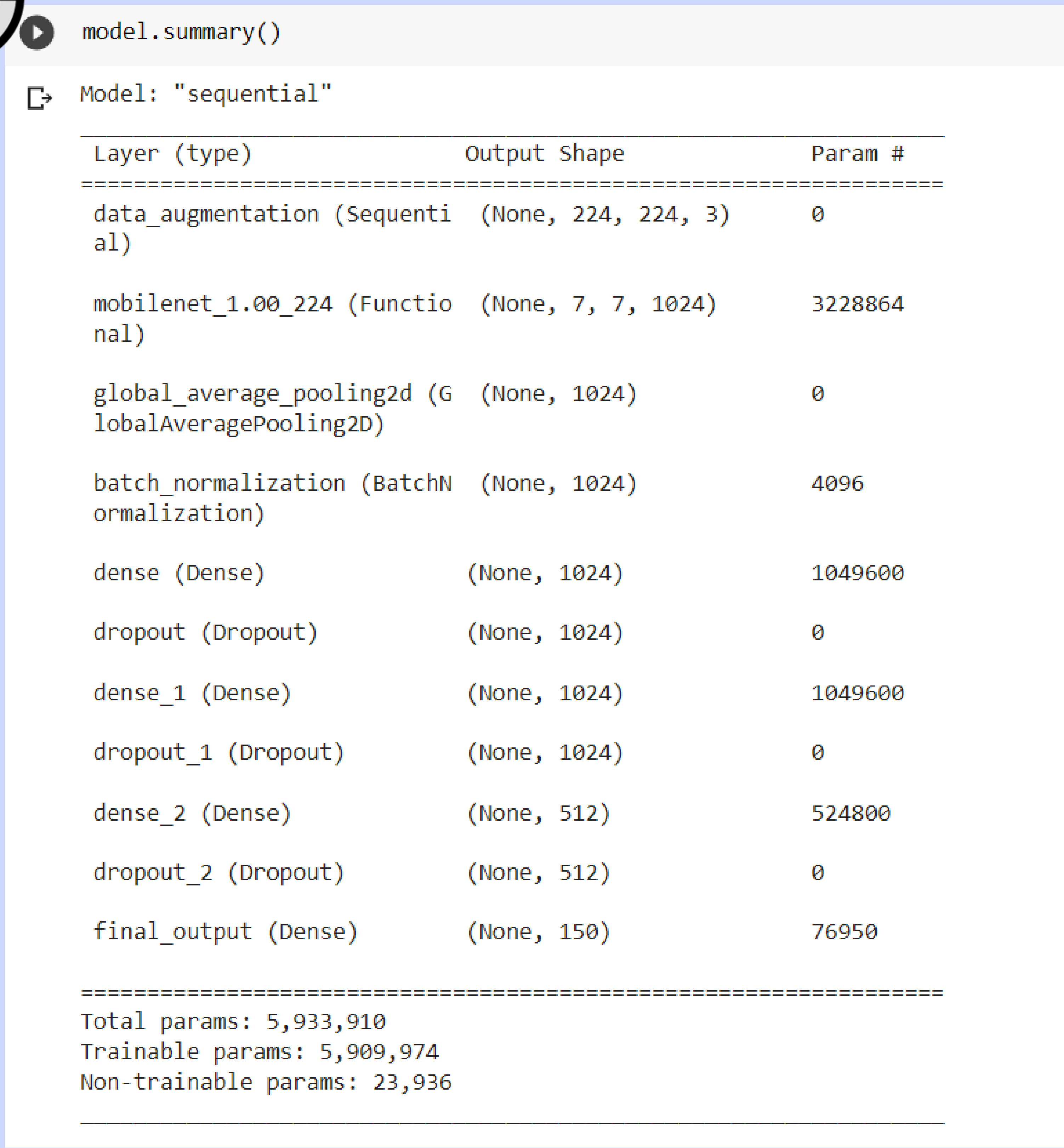
```
Epoch 75/150
171/171 [==============================] - 40s 232ms/step - loss: 0.2525 - accuracy: 0.9487 - val_loss: 0.9654 - val_accuracy: 0.8453
Epoch 76/150
171/171 [==============================] - 40s 230ms/step - loss: 0.2441 - accuracy: 0.9474 - val_loss: 0.8331 - val_accuracy: 0.8526
Epoch 77/150
171/171 [==============================] - 40s 230ms/step - loss: 0.2306 - accuracy: 0.9456 - val_loss: 0.9276 - val_accuracy: 0.8622
Epoch 78/150
171/171 [==============================] - 40s 231ms/step - loss: 0.2567 - accuracy: 0.9419 - val_loss: 0.9227 - val_accuracy: 0.8460
Epoch 79/150
171/171 [==============================] - 41s 233ms/step - loss: 0.2342 - accuracy: 0.9446 - val_loss: 0.7300 - val_accuracy: 0.8666
Epoch 80/150
171/171 [==============================] - 40s 229ms/step - loss: 0.2465 - accuracy: 0.9496 - val_loss: 0.8927 - val_accuracy: 0.8600
Epoch 81/150
171/171 [==============================] - 40s 229ms/step - loss: 0.2122 - accuracy: 0.9544 - val_loss: 0.8233 - val_accuracy: 0.8607
Epoch 82/150
171/171 [==============================] - 40s 232ms/step - loss: 0.1925 - accuracy: 0.9586 - val_loss: 0.8102 - val_accuracy: 0.8644
Epoch 83/150
171/171 [==============================] - 41s 233ms/step - loss: 0.2658 - accuracy: 0.9423 - val_loss: 1.1808 - val_accuracy: 0.8152
Epoch 84/150
171/171 [==============================] - ETA: 0s - loss: 0.2393 - accuracy: 0.9481Restoring model weights from the end of the best epoch: 69.
171/171 [==============================] - 41s 233ms/step - loss: 0.2393 - accuracy: 0.9481 - val_loss: 0.8163 - val_accuracy: 0.8570
Epoch 84: early stopping
```

Model performance



```
Maximum validation accuracy:  0.876099705696106
Minimum loss: 0.6568965315818787
```

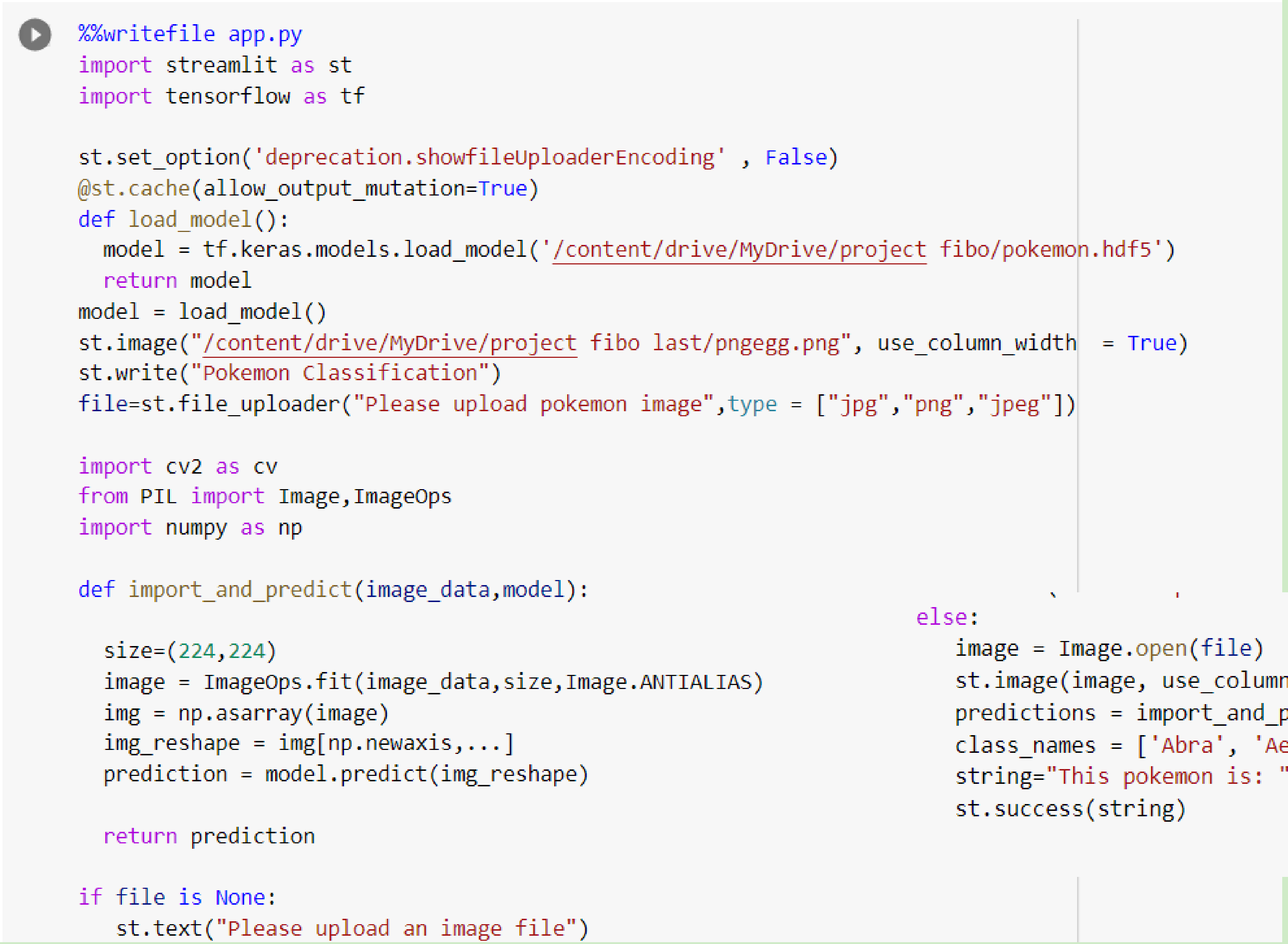
Model summary



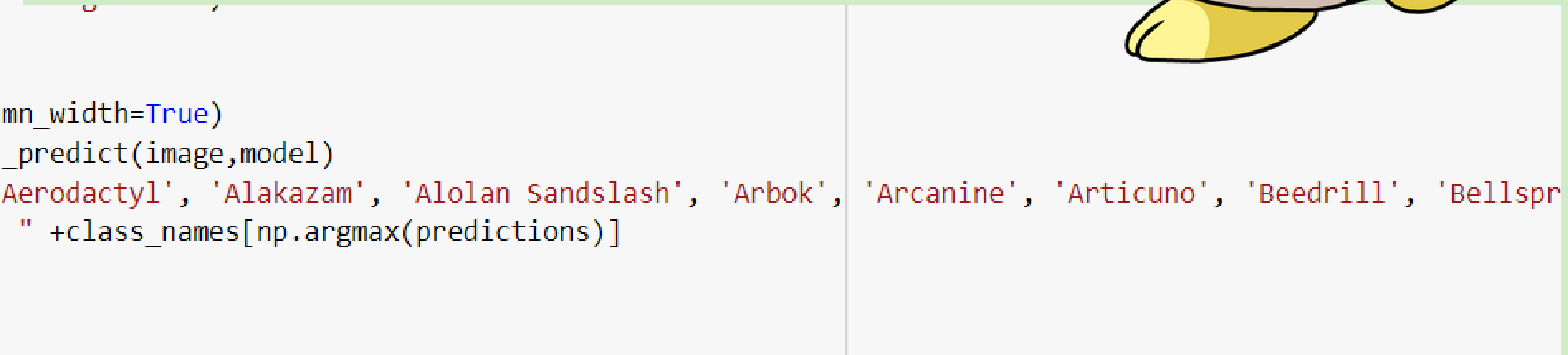
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 data_augmentation (Sequenti  (None, 224, 224, 3)      0
 al)

 mobilenet_1.00_224 (Functio  (None, 7, 7, 1024)       3228864
 nal)

 global_average_pooling2d (G  (None, 1024)             0
 lobalAveragePooling2D)

 batch_normalization (BatchN  (None, 1024)             4096
 ormalization)

 dense (Dense)              (None, 1024)              1049600

 dropout (Dropout)          (None, 1024)              0

 dense_1 (Dense)            (None, 1024)              1049600

 dropout_1 (Dropout)        (None, 1024)              0

 dense_2 (Dense)            (None, 512)               524800

 dropout_2 (Dropout)        (None, 512)               0

 final_output (Dense)       (None, 150)               76950

=================================================================
Total params: 5,933,910
Trainable params: 5,909,974
Non-trainable params: 23,936
_________________________________________________________________
```

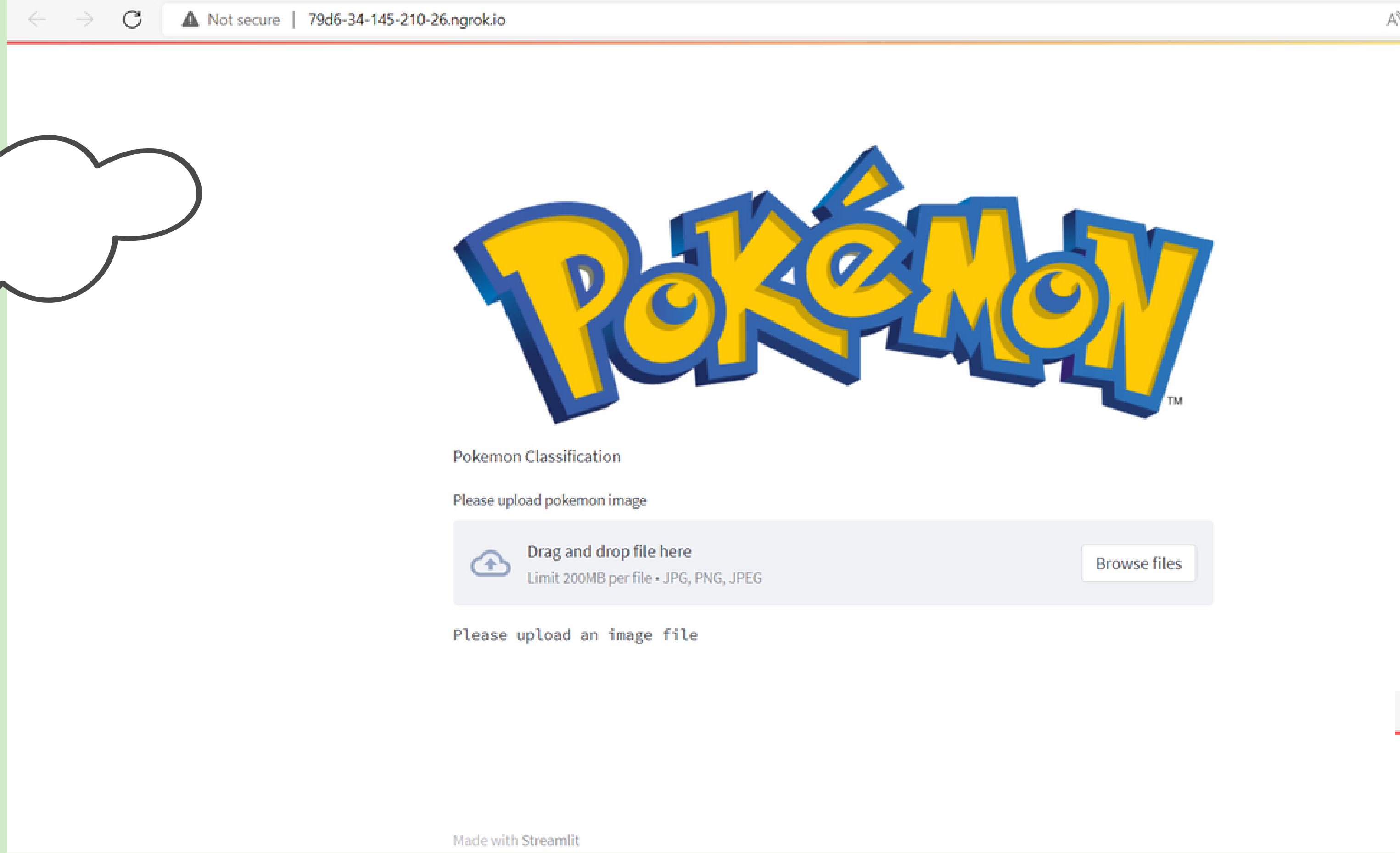
Web App for predicted



```
%%writefile app.py
import streamlit as st
import tensorflow as tf

st.set_option('deprecation.showfileUploaderEncoding' , False)
@st.cache(allow_output_mutation=True)
def load_model():
  model = tf.keras.models.load_model('/content/drive/MyDrive/project fibo/pokemon.hdf5')
  return model
model = load_model()
st.image("/content/drive/MyDrive/project fibo last/pngegg.png", use_column_width  = True)
st.write("Pokemon Classification")
file=st.file_uploader("Please upload pokemon image",type = ["jpg","png","jpeg"])

import cv2 as cv
from PIL import Image,ImageOps
import numpy as np

def import_and_predict(image_data,model):

  size=(224,224)
  image = ImageOps.fit(image_data,size,Image.ANTIALIAS)
  img = np.asarray(image)
  img_reshape = img[np.newaxis,...]
  prediction = model.predict(img_reshape)

  return prediction

if file is None:
   st.text("Please upload an image file")
```



```
else:
   image = Image.open(file)
   st.image(image, use_column_width=True)
   predictions = import_and_predict(image,model)
   class_names = ['Abra', 'Aerodactyl', 'Alakazam', 'Alolan Sandslash', 'Arbok', 'Arcanine', 'Articuno', 'Beedrill', 'Bellspr
   string="This pokemon is: " +class_names[np.argmax(predictions)]
   st.success(string)
```

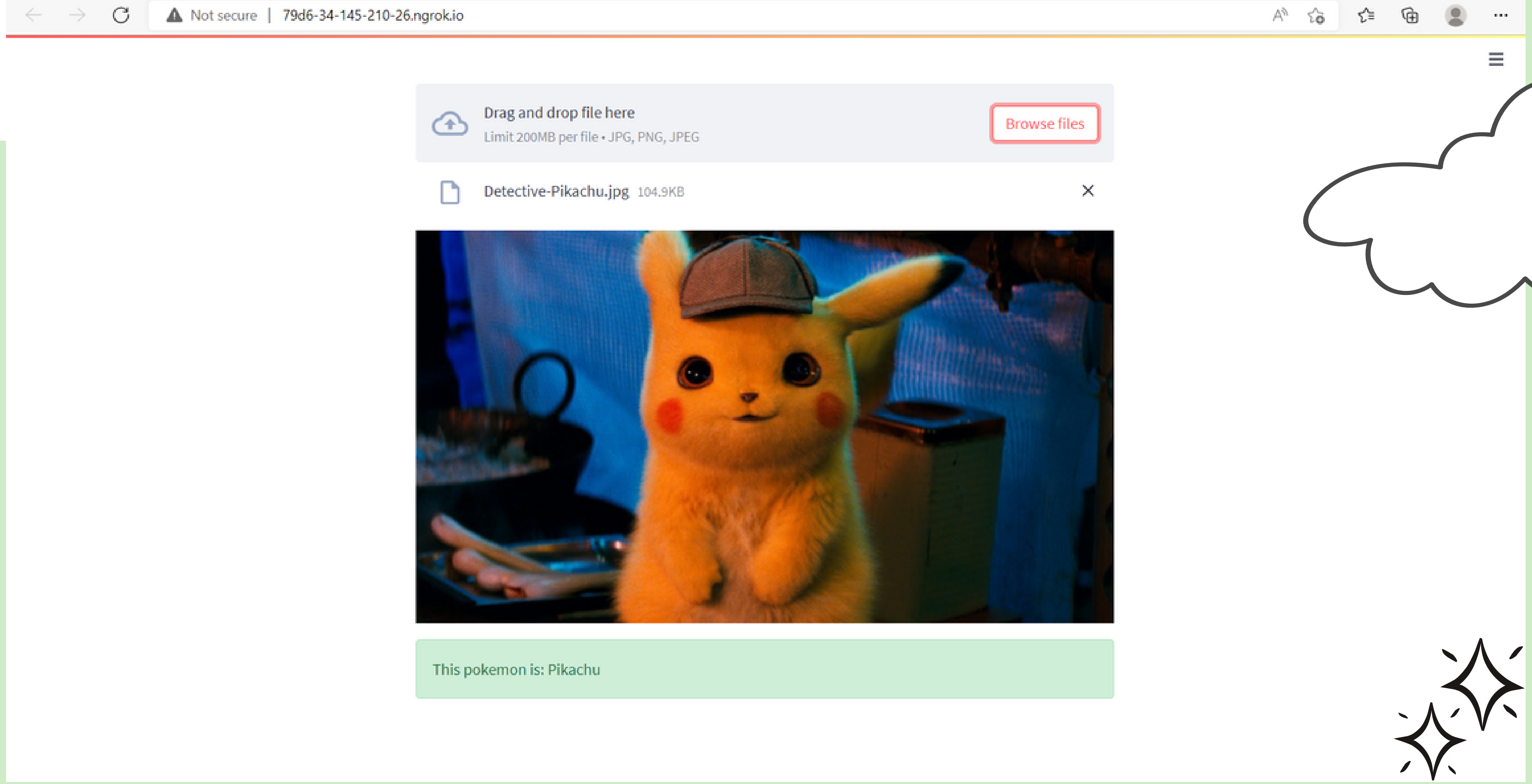

Not secure | 79d6-34-145-210-26.ngrok.io
POKéMON™

Pokemon Classification

Please upload pokemon image

Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files

```
Please upload an image file
```


Not secure | 79d6-34-145-210-26.ngrok.io
Drag and drop file here
Limit 200MB per file • JPG, PNG, JPEG
Browse files
Detective-Pikachu.jpg 104.9KB

This pokemon is: Pikachu

Thank you