

✓ Step 1: Download the Intel Image Classification Dataset

To download the dataset from Kaggle, you'll need a Kaggle API token. Follow these steps to obtain and set up your token:

1. **Go to Kaggle:** Log in to your Kaggle account.
2. **Navigate to your profile:** Click on your profile picture in the top right corner and select "My Account".
3. **Create New API Token:** Scroll down to the "API" section and click on "Create New API Token". This will download a `kaggle.json` file to your computer.
4. **Upload `kaggle.json` to Colab:** In the left sidebar of Google Colab, click on the folder icon to open the file browser. Click the "Upload" icon and upload the `kaggle.json` file to your Colab environment. **Ensure it's uploaded to the root directory `/content/` for the following code to work correctly.**

```
# Install the Kaggle API client
!pip install kaggle

# Make sure the kaggle.json file is in the correct directory
# If you uploaded it to '/content/', the following command will move it to the
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: jupyter in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: kagglesdk<1.0,>=0.1.20 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: markdown-it-py>=1.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: mdit-py-plugins in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: nbformat in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.12/dist-packages
```

```
Requirement already satisfied: fastjsonschema>=2.15 in /usr/local/lib/python3.12
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.12/dist
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /usr/local/lib/pyth
Requirement already satisfied: traitlets>=5.1 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/loca
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.12/
Requirement already satisfied: rpds-py>=0.25.0 in /usr/local/lib/python3.12/dist
Requirement already satisfied: platformdirs>=2.5 in /usr/local/lib/python3.12/di
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python
mv: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

```
# Download the Intel Image Classification dataset
!kaggle datasets download -d puneet6060/intel-image-classification
```

```
# Unzip the dataset
import zipfile
import os
```

```
zip_ref = zipfile.ZipFile('intel-image-classification.zip', 'r')
zip_ref.extractall('intel_images')
zip_ref.close()
```

```
print("Dataset downloaded and extracted to 'intel_images/'")
print(os.listdir('intel_images'))
```

```
Warning: Looks like you're using an outdated `kaggle` version (installed: 2.0.2)
Dataset URL: https://www.kaggle.com/datasets/puneet6060/intel-image-classification
License(s): copyright-authors
Downloading intel-image-classification.zip to /content
100% 346M/346M [00:03<00:00, 103MB/s]
```

```
Dataset downloaded and extracted to 'intel_images/'
['seg_test', 'seg_train', 'seg_pred']
```

Now that the dataset is downloaded and extracted, we can proceed with splitting the dataset.

✓ Step 2: Splitting the Dataset

We will now split the `seg_train` data into training and validation sets, and use `seg_test` as our test set. We'll use a 80/20 split for training and validation.

```
from sklearn import train_test_split
```

or the dataset

```
th.join(base_dir, 'seg_train')
h.join(base_dir, 'seg_test')
```

ted 'seg_train' and 'seg_test' directories

e the zip file might contain an extra top-level folder inside 'intel_images'

```
.join(initial_train_dir, 'seg_train')):
oin(initial_train_dir, 'seg_train')
```

rain_dir

```
.join(initial_test_dir, 'seg_test')):
in(initial_test_dir, 'seg_test')
```

st_dir

```
g data directory: {train_dir}")
data directory: {test_dir}")
```

for the splits

```
rain'
alidation'
st'
```

```
if necessary
:
```

```
IR, exist_ok=True)
IR, exist_ok=True)
R, exist_ok=True)
```

ectories) from the training data

irectories only, as classes are subfolders

```
listdir(train_dir) if os.path.isdir(os.path.join(train_dir, d))]
lasses}")
```

to the new test split directory

```
:
oin(TEST_SPLIT_DIR, class_name), exist_ok=True)
s.path.join(test_dir, class_name)
rce_class_path):
, not directories
s.listdir(source_class_path):
.path.join(source_class_path, img_name)
file(img_path):
py(img_path, os.path.join(TEST_SPLIT_DIR, class_name, img_name))
```

to {TEST_SPLIT_DIR}")

```

o create train and validation splits
:
join(train_dir, class_name)
(class_path):
Class directory not found: {class_path}")

ly actual image files are included
n(class_path, img) for img in os.listdir(class_path) if os.path.isfile(os.path.j

images into training and validation sets
le empty class directories gracefully
No images found in {class_path}. Skipping split for this class.")

ges = train_test_split(images, test_size=0.2, random_state=42)

ories in the new train and validation splits
oin(TRAIN_SPLIT_DIR, class_name), exist_ok=True)
oin(VALID_SPLIT_DIR, class_name), exist_ok=True)

r respective new directories
_images:
ath, os.path.join(TRAIN_SPLIT_DIR, class_name, os.path.basename(img_path)))
images:
ath, os.path.join(VALID_SPLIT_DIR, class_name, os.path.basename(img_path)))

ta into {TRAIN_SPLIT_DIR} and {VALID_SPLIT_DIR}")

t Summary ---")
ny class directories that might not have been created if no images were found
r c in os.listdir(TRAIN_SPLIT_DIR) if os.path.isdir(os.path.join(TRAIN_SPLIT_DIR,
r c in os.listdir(VALID_SPLIT_DIR) if os.path.isdir(os.path.join(VALID_SPLIT_DIR,
c in os.listdir(TEST_SPLIT_DIR) if os.path.isdir(os.path.join(TEST_SPLIT_DIR, c

{sum([len(os.listdir(os.path.join(TRAIN_SPLIT_DIR, c))) for c in all_train_class
: {sum([len(os.path.join(VALID_SPLIT_DIR, c)) for c in all_valid_classes])})")
sum([len(os.listdir(os.path.join(TEST_SPLIT_DIR, c))) for c in all_test_classes]

```

```

Effective training data directory: intel_images/seg_train/seg_train
Effective testing data directory: intel_images/seg_test/seg_test
Found classes: ['forest', 'street', 'glacier', 'mountain', 'sea', 'buildings']
Copied test data to data/test
Split training data into data/train and data/validation

```

```

--- Dataset Split Summary ---
Training images: 11224
Validation images: 135
Testing images: 3000

```

The dataset is now split into `train`, `validation`, and `test` directories, each containing subdirectories for the different image classes. Next, we will apply data augmentation.

✓ Step 3: Applying Data Augmentation

Data augmentation is a technique used to increase the diversity of your training data by applying random transformations to the images. This helps prevent overfitting and improves the model's generalization capabilities. We'll use Keras's

`ImageDataGenerator` for this purpose.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define image dimensions
IMG_WIDTH = 150
IMG_HEIGHT = 150
BATCH_SIZE = 32

# Data augmentation and preprocessing for training data
train_datagen = ImageDataGenerator(
    rescale=1./255,          # Normalize pixel values to [0, 1]
    rotation_range=40,       # Randomly rotate images by up to 40 degrees
    width_shift_range=0.2,    # Randomly shift images horizontally
    height_shift_range=0.2,   # Randomly shift images vertically
    shear_range=0.2,         # Apply shear transformation
    zoom_range=0.2,          # Apply random zoom
    horizontal_flip=True,     # Randomly flip images horizontally
    fill_mode='nearest'      # Fill newly created pixels with the nearest value
)

# Only rescale for validation and test data (no augmentation)
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load images from directories using flow_from_directory
train_generator = train_datagen.flow_from_directory(
    TRAIN_SPLIT_DIR,
    target_size=(IMG_WIDTH, IMG_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode='categorical' # For multi-class classification
)

validation_generator = validation_datagen.flow_from_directory(
    VALID_SPLIT_DIR,
    target_size=(IMG_WIDTH, IMG_HEIGHT),
```

```

        batch_size=BATCH_SIZE,
        class_mode='categorical'
    )

    test_generator = test_datagen.flow_from_directory(
        TEST_SPLIT_DIR,
        target_size=(IMG_WIDTH, IMG_HEIGHT),
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=False # Keep data in order for evaluation
    )

    print("\nData augmentation and generators setup complete.")
    print(f"Found {train_generator.samples} training images belonging to {train_generator.class_indices}")
    print(f"Found {validation_generator.samples} validation images belonging to {validation_generator.class_indices}")
    print(f"Found {test_generator.samples} testing images belonging to {test_generator.class_indices}")

    Found 11224 images belonging to 6 classes.
    Found 2810 images belonging to 6 classes.
    Found 3000 images belonging to 6 classes.

    Data augmentation and generators setup complete.
    Found 11224 training images belonging to 6 classes.
    Found 2810 validation images belonging to 6 classes.
    Found 3000 testing images belonging to 6 classes.

```

With the data generators set up, we are now ready to define and train our models.

✓ Step 4: Training and Testing with 2 Models

We will now define, compile, train, and evaluate two different Convolutional Neural Network (CNN) models to classify the Intel Image Classification dataset.

✓ Model 1: Simple CNN

This model will consist of a few convolutional layers followed by max-pooling layers, and then dense layers for classification. This architecture is common for image classification tasks.

```

from tensorflow.keras import layers, models

# Define Model 1: Simple CNN
model_1 = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_WIDTH, IMG_HEIGHT, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),

```

```

layers.MaxPooling2D((2, 2)),
layers.Conv2D(128, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Flatten(),
layers.Dense(512, activation='relu'),
layers.Dense(train_generator.num_classes, activation='softmax') # Output layer
])

```

```

# Compile Model 1
model_1.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

model_1.summary()
print("Model 1 (Simple CNN) compiled successfully.")

```

```

/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv
super().__init__(activity_regularizer=activity_regularizer, **kwargs)

```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
flatten (Flatten)	(None, 36992)	0
dense (Dense)	(None, 512)	18,940,416
dense_1 (Dense)	(None, 6)	3,078

Total params: 19,036,742 (72.62 MB)
Trainable params: 19,036,742 (72.62 MB)
Non-trainable params: 0 (0.00 B)

Model 1 (Simple CNN) compiled successfully

```

# Train Model 1
EPOCHS = 10 # You can adjust the number of epochs

history_1 = model_1.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)

```

)

```
print("\nModel 1 training complete.")
```

```
Epoch 1/10
350/350 ————— 85s 223ms/step - accuracy: 0.5432 - loss: 1.1855 -
Epoch 2/10
 1/350 ————— 12s 34ms/step - accuracy: 0.6562 - loss: 0.6841/usr
self._interrupted_warning()
350/350 ————— 2s 6ms/step - accuracy: 0.6562 - loss: 0.6841 - val
Epoch 3/10
350/350 ————— 73s 208ms/step - accuracy: 0.6401 - loss: 0.9349 -
Epoch 4/10
350/350 ————— 2s 6ms/step - accuracy: 0.7812 - loss: 0.6848 - val
Epoch 5/10
350/350 ————— 73s 209ms/step - accuracy: 0.6841 - loss: 0.8377 -
Epoch 6/10
350/350 ————— 2s 6ms/step - accuracy: 0.6562 - loss: 1.1077 - val
Epoch 7/10
350/350 ————— 74s 210ms/step - accuracy: 0.7059 - loss: 0.7813 -
Epoch 8/10
350/350 ————— 3s 7ms/step - accuracy: 0.6250 - loss: 0.7787 - val
Epoch 9/10
350/350 ————— 74s 211ms/step - accuracy: 0.7338 - loss: 0.7237 -
Epoch 10/10
350/350 ————— 2s 6ms/step - accuracy: 0.7812 - loss: 0.5499 - val

Model 1 training complete.
```

```
# Evaluate Model 1 on the test set
print("\nEvaluating Model 1...")
loss_1, accuracy_1 = model_1.evaluate(test_generator,
                                     steps=test_generator.samples // BATCH_SIZE,
                                     verbose=1)
print(f"Model 1 Test Loss: {loss_1:.4f}")
print(f"Model 1 Test Accuracy: {accuracy_1:.4f}")
```

```
Evaluating Model 1...
93/93 ————— 3s 28ms/step - accuracy: 0.7739 - loss: 0.6342
Model 1 Test Loss: 0.6342
Model 1 Test Accuracy: 0.7739
```

✓ Model 2: Transfer Learning with MobileNetV2

For our second model, we will use a pre-trained convolutional neural network (MobileNetV2) as a feature extractor and add our own classification layers on top. This technique, called transfer learning, is highly effective for image classification tasks, especially when data is limited.

```

from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D

# Load the pre-trained MobileNetV2 model without the top (classification) layer
base_model = MobileNetV2(input_shape=(IMG_WIDTH, IMG_HEIGHT, 3),
                          include_top=False,
                          weights='imagenet')

# Freeze the layers of the base model so they are not updated during training
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification layers on top of the base model
x = base_model.output
x = GlobalAveragePooling2D()(x) # Global average pooling layer
x = Dense(1024, activation='relu')(x) # A new dense layer
predictions = Dense(train_generator.num_classes, activation='softmax')(x) # Out

# Create the full model
model_2 = Model(inputs=base_model.input, outputs=predictions)

# Compile Model 2
model_2.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])

model_2.summary()
print("Model 2 (MobileNetV2 with Transfer Learning) compiled successfully.")

```


(ReLU)	32)		
expanded_conv_proj... (Conv2D)	(None, 75, 75, 16)	512	expanded_conv_de...
expanded_conv_proj... (BatchNormalizatio...	(None, 75, 75, 16)	64	expanded_conv_pr...
block_1_expand (Conv2D)	(None, 75, 75, 96)	1,536	expanded_conv_pr...
block_1_expand_BN (BatchNormalizatio...	(None, 75, 75, 96)	384	block_1_expand[0...
block_1_expand_relu (ReLU)	(None, 75, 75, 96)	0	block_1_expand_B...
block_1_pad (ZeroPadding2D)	(None, 77, 77, 96)	0	block_1_expand_r...
block_1_depthwise (DepthwiseConv2D)	(None, 38, 38, 96)	864	block_1_pad[0][0]
block_1_depthwise_... (BatchNormalizatio...	(None, 38, 38, 96)	384	block_1_depthwis...
block_1_depthwise_... (ReLU)	(None, 38, 38, 96)	0	block_1_depthwis...
block_1_project	(None, 38, 38,	2,304	block_1_depthwis...

(Conv2D)	24)		
block_1_project_BN (BatchNormalizatio...	(None, 38, 38, 24)	96	block_1_project[...
block_2_expand (Conv2D)	(None, 38, 38, 144)	3,456	block_1_project_...
block_2_expand_BN (BatchNormalizatio...	(None, 38, 38, 144)	576	block_2_expand[0...
block_2_expand_relu (ReLU)	(None, 38, 38, 144)	0	block_2_expand_B...
block_2_depthwise (DepthwiseConv2D)	(None, 38, 38, 144)	1,296	block_2_expand_r...
block_2_depthwise_... (BatchNormalizatio...	(None, 38, 38, 144)	576	block_2_depthwis...
block_2_depthwise_... (ReLU)	(None, 38, 38, 144)	0	block_2_depthwis...
block_2_project (Conv2D)	(None, 38, 38, 24)	3,456	block_2_depthwis...
block_2_project_BN (BatchNormalizatio...	(None, 38, 38, 24)	96	block_2_project[...
block_2_add (Add)	(None, 38, 38, 24)	0	block_1_project_... block_2_project_...
block_3_expand (Conv2D)	(None, 38, 38, 144)	3,456	block_2_add[0][0]
block_3_expand_BN (BatchNormalizatio...	(None, 38, 38, 144)	576	block_3_expand[0...
block_3_expand_relu (ReLU)	(None, 38, 38, 144)	0	block_3_expand_B...
block_3_pad (ZeroPadding2D)	(None, 39, 39, 144)	0	block_3_expand_r...
block_3_depthwise (DepthwiseConv2D)	(None, 19, 19, 144)	1,296	block_3_pad[0][0]
block_3_depthwise_... (BatchNormalizatio...	(None, 19, 19, 144)	576	block_3_depthwis...
block_3_depthwise_... (ReLU)	(None, 19, 19, 144)	0	block_3_depthwis...
block_3_project (Conv2D)	(None, 19, 19, 32)	4,608	block_3_depthwis...
block_3_project_BN (BatchNormalizatio...	(None, 19, 19, 32)	128	block_3_project[...

block_4_expand (Conv2D)	(None, 19, 19, 192)	6,144	block_3_project_...
block_4_expand_BN (BatchNormalizatio...	(None, 19, 19, 192)	768	block_4_expand[0...
block_4_expand_relu (ReLU)	(None, 19, 19, 192)	0	block_4_expand_B...
block_4_depthwise (DepthwiseConv2D)	(None, 19, 19, 192)	1,728	block_4_expand_r...
block_4_depthwise_... (BatchNormalizatio...	(None, 19, 19, 192)	768	block_4_depthwis...
block_4_depthwise_... (ReLU)	(None, 19, 19, 192)	0	block_4_depthwis...
block_4_project (Conv2D)	(None, 19, 19, 32)	6,144	block_4_depthwis...
block_4_project_BN (BatchNormalizatio...	(None, 19, 19, 32)	128	block_4_project[...
block_4_add (Add)	(None, 19, 19, 32)	0	block_3_project_... block_4_project_...
block_5_expand (Conv2D)	(None, 19, 19, 192)	6,144	block_4_add[0][0]
block_5_expand_BN (BatchNormalizatio...	(None, 19, 19, 192)	768	block_5_expand[0...
block_5_expand_relu (ReLU)	(None, 19, 19, 192)	0	block_5_expand_B...
block_5_depthwise (DepthwiseConv2D)	(None, 19, 19, 192)	1,728	block_5_expand_r...
block_5_depthwise_... (BatchNormalizatio...	(None, 19, 19, 192)	768	block_5_depthwis...
block_5_depthwise_... (ReLU)	(None, 19, 19, 192)	0	block_5_depthwis...
block_5_project (Conv2D)	(None, 19, 19, 32)	6,144	block_5_depthwis...
block_5_project_BN (BatchNormalizatio...	(None, 19, 19, 32)	128	block_5_project[...
block_5_add (Add)	(None, 19, 19, 32)	0	block_4_add[0][0... block_5_project_...
block_6_expand (Conv2D)	(None, 19, 19, 192)	6,144	block_5_add[0][0]

block_6_expand_BN (BatchNormalizatio...	(None, 19, 19, 192)	768	block_6_expand[0...
block_6_expand_relu (ReLU)	(None, 19, 19, 192)	0	block_6_expand_B...
block_6_pad (ZeroPadding2D)	(None, 21, 21, 192)	0	block_6_expand_r...
block_6_depthwise (DepthwiseConv2D)	(None, 10, 10, 192)	1,728	block_6_pad[0][0]
block_6_depthwise_... (BatchNormalizatio...	(None, 10, 10, 192)	768	block_6_depthwis...
block_6_depthwise_... (ReLU)	(None, 10, 10, 192)	0	block_6_depthwis...
block_6_project (Conv2D)	(None, 10, 10, 64)	12,288	block_6_depthwis...
block_6_project_BN (BatchNormalizatio...	(None, 10, 10, 64)	256	block_6_project[...
block_7_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_6_project_...
block_7_expand_BN (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_7_expand[0...
block_7_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_7_expand_B...
block_7_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_7_expand_r...
block_7_depthwise_... (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_7_depthwis...
block_7_depthwise_... (ReLU)	(None, 10, 10, 384)	0	block_7_depthwis...
block_7_project (Conv2D)	(None, 10, 10, 64)	24,576	block_7_depthwis...
block_7_project_BN (BatchNormalizatio...	(None, 10, 10, 64)	256	block_7_project[...
block_7_add (Add)	(None, 10, 10, 64)	0	block_6_project_... block_7_project_...
block_8_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_7_add[0][0]
block_8_expand_BN (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_8_expand[0...

block_8_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_8_expand_B...
block_8_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_8_expand_r...
block_8_depthwise_... (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_8_depthwis...
block_8_depthwise_... (ReLU)	(None, 10, 10, 384)	0	block_8_depthwis...
block_8_project (Conv2D)	(None, 10, 10, 64)	24,576	block_8_depthwis...
block_8_project_BN (BatchNormalizatio...	(None, 10, 10, 64)	256	block_8_project[...
block_8_add (Add)	(None, 10, 10, 64)	0	block_7_add[0][0... block_8_project_...
block_9_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_8_add[0][0]
block_9_expand_BN (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_9_expand[0...
block_9_expand_relu (ReLU)	(None, 10, 10, 384)	0	block_9_expand_B...
block_9_depthwise (DepthwiseConv2D)	(None, 10, 10, 384)	3,456	block_9_expand_r...
block_9_depthwise_... (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_9_depthwis...
block_9_depthwise_... (ReLU)	(None, 10, 10, 384)	0	block_9_depthwis...
block_9_project (Conv2D)	(None, 10, 10, 64)	24,576	block_9_depthwis...
block_9_project_BN (BatchNormalizatio...	(None, 10, 10, 64)	256	block_9_project[...
block_9_add (Add)	(None, 10, 10, 64)	0	block_8_add[0][0... block_9_project_...
block_10_expand (Conv2D)	(None, 10, 10, 384)	24,576	block_9_add[0][0]
block_10_expand_BN (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_10_expand[...
block_10_expand_re... (ReLU)	(None, 10, 10, 384)	0	block_10_expand_...
block_10_depthwise	(None, 10, 10,	3,456	block_10_expand_...

(DepthwiseConv2D)	384)		
block_10_depthwise... (BatchNormalizatio...	(None, 10, 10, 384)	1,536	block_10_depthwi...
block_10_depthwise... (ReLU)	(None, 10, 10, 384)	0	block_10_depthwi...
block_10_project (Conv2D)	(None, 10, 10, 96)	36,864	block_10_depthwi...
block_10_project_BN (BatchNormalizatio...	(None, 10, 10, 96)	384	block_10_project...
block_11_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_10_project...
block_11_expand_BN (BatchNormalizatio...	(None, 10, 10, 576)	2,304	block_11_expand[...
block_11_expand_re... (ReLU)	(None, 10, 10, 576)	0	block_11_expand_...
block_11_depthwise (DepthwiseConv2D)	(None, 10, 10, 576)	5,184	block_11_expand_...
block_11_depthwise... (BatchNormalizatio...	(None, 10, 10, 576)	2,304	block_11_depthwi...
block_11_depthwise... (ReLU)	(None, 10, 10, 576)	0	block_11_depthwi...
block_11_project (Conv2D)	(None, 10, 10, 96)	55,296	block_11_depthwi...
block_11_project_BN (BatchNormalizatio...	(None, 10, 10, 96)	384	block_11_project...
block_11_add (Add)	(None, 10, 10, 96)	0	block_10_project... block_11_project...
block_12_expand (Conv2D)	(None, 10, 10, 576)	55,296	block_11_add[0][...
block_12_expand_BN (BatchNormalizatio...	(None, 10, 10, 576)	2,304	block_12_expand[...
block_12_expand_re... (ReLU)	(None, 10, 10, 576)	0	block_12_expand_...
block_12_depthwise (DepthwiseConv2D)	(None, 10, 10, 576)	5,184	block_12_expand_...
block_12_depthwise... (BatchNormalizatio...	(None, 10, 10, 576)	2,304	block_12_depthwi...
block_12_depthwise... (ReLU)	(None, 10, 10, 576)	0	block_12_depthwi...