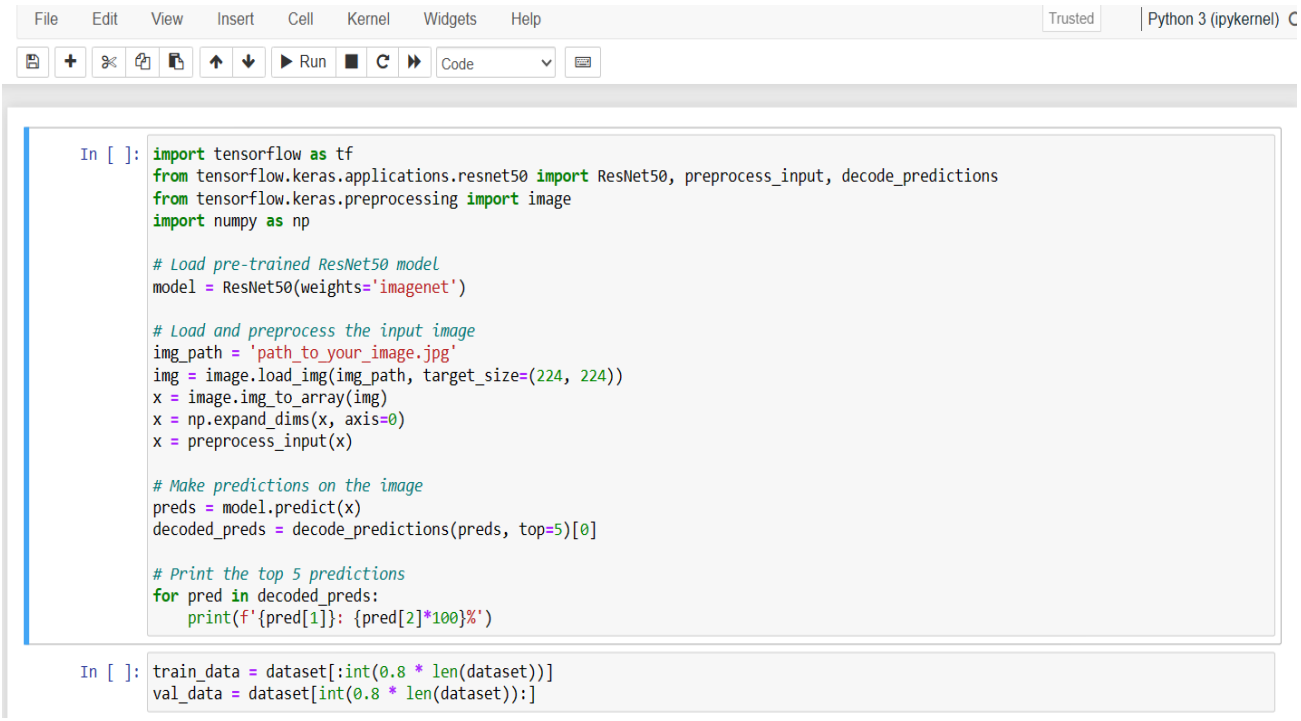
HUMAN DETECTION

- Completed model training
- Completed CNN feature extraction
- In process of testing of the model

CODE:



In [ ]:
```python
import tensorflow as tf
from tensorflow.keras.applications import ResNet50

# Select the pre-trained CNN model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model's layers
base_model.trainable = False

# Add additional layers for crowd counting
model = tf.keras.models.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(1)  # Output layer for counting
])
```

In [ ]:
```python
# Preprocess the dataset
# Define any necessary data augmentation techniques
data_augmentation = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    preprocessing_function=tf.keras.applications.resnet50.preprocess_input
)
```

```python
# Train the CNN model
batch_size = 32
epochs = 10

train_generator = data_augmentation.flow_from_directory(
    train_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='sparse',
    shuffle=True
)

val_generator = data_augmentation.flow_from_directory(
    val_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='sparse',
    shuffle=False
)

model.compile(optimizer='adam', loss='mse')  # Use mean square error loss
model.fit(train_generator, validation_data=val_generator, epochs=epochs)
```

In [ ]:
```
#svm
import numpy as np

# Extract features from the trained CNN model
cnn_features_train = base_model.predict(train_data)  # Assuming you have extracted features for the training set
cnn_features_val = base_model.predict(val_data)      # Assuming you have extracted features for the validation set

# Combine features with counts
train_counts = train_labels  # Assuming you have the ground truth counts for the training set
val_counts = val_labels      # Assuming you have the ground truth counts for the validation set

# Train the SVM for individual person count prediction
class MultiOutputSVM:
    def _init_(self, C=1.0, learning_rate=0.01, max_iterations=1000):
        self.C = C
        self.learning_rate = learning_rate
        self.max_iterations = max_iterations
        self.models = []  # List of SVM models, one for each person

    def fit(self, X, y):
        num_samples, num_features = X.shape
        num_persons = y.shape[1]  # Number of persons to predict

        self.models = []
        for i in range(num_persons):
            model = SVM(C=self.C, learning_rate=self.learning_rate, max_iterations=self.max_iterations)
            model.fit(X, y[:, i])
            self.models.append(model)

    def predict(self, X):
        num_samples = X.shape[0]
        num_persons = len(self.models)
        predictions = np.zeros((num_samples, num_persons))
```

```
    def predict(self, X):
        num_samples = X.shape[0]
        num_persons = len(self.models)
        predictions = np.zeros((num_samples, num_persons))

        for i in range(num_persons):
            predictions[:, i] = self.models[i].predict(X)

        return predictions

svm_model = MultiOutputSVM(C=1.0, learning_rate=0.01, max_iterations=1000)
svm_model.fit(cnn_features_train, train_counts)
```