Funkcija load_data je uporabljena za zajem podatkov in vračanje teh podatkov z returnom. Ker je vsebina map drugačna v test in training, sem ločil na test in training vsebino.

```python
def load_data(data_dir, img_size=(64, 64), is_test=False):
    images = []
    labels = []

    valid_extensions = ['.jpg', '.jpeg', '.png', '.bmp', '.tiff', '.ppm']

    if is_test:
        # Load images directly from test directory
        print(f"Loading test images from {data_dir}")
        for img_name in tqdm(os.listdir(data_dir), desc="Loading test images"):
            img_path = os.path.join(data_dir, img_name)
            ext = os.path.splitext(img_name)[1].lower()
            if ext in valid_extensions:
                try:
                    img = load_img(img_path, target_size=img_size)
                    img = img_to_array(img)
                    images.append(img)
                    labels.append(0)
                except Exception as e:
                    print(f"Error loading image {img_path}: {e}")
```

```python
else:
    # Load images from training subdirectories
    class_names = sorted(os.listdir(data_dir))
    print(f"Found classes: {class_names}")
    for label, class_name in enumerate(class_names):
        class_dir = os.path.join(data_dir, class_name)
        if os.path.isdir(class_dir):
            print(f"Processing directory: {class_dir}")
            for img_name in tqdm(os.listdir(class_dir), desc=f"Loading {class_name}"):
                img_path = os.path.join(class_dir, img_name)
                ext = os.path.splitext(img_name)[1].lower()
                if ext in valid_extensions:
                    try:
                        img = load_img(img_path, target_size=img_size)
                        img = img_to_array(img)
                        images.append(img)
                        labels.append(label)
                    except Exception as e:
                        print(f"Error loading image {img_path}: {e}")
                else:
```

Funkcija load and prepare data vzame in pripravi podatke. Vzame podatke iz load_data, prav tako pa izpiše shape test in train. To izpišem, saj mi je delalo probleme s type missmatch

```
def load_and_prepare_data():
    train_dir = 'Images'
    test_dir = 'Final_Test/Images'

    X_train, y_train = load_data(train_dir)
    X_test, y_test = load_data(test_dir, is_test=True)

    # Print the shapes of X_train, y_train, X_test, and y_test
    print("Shapes after loading data:")
    print("X_train shape:", X_train.shape)
    print("y_train shape:", y_train.shape)
    print("X_test shape:", X_test.shape)
    print("y_test shape:", y_test.shape)

    # Normalizacija podatkov
    X_train, X_test = X_train / 255.0, X_test / 255.0


# Razdelitev učnih podatkov na učne in validacijske množice (80-20%)
if len(X_train) == 0 or len(y_train) == 0:
    raise ValueError(f"No training data found in {train_dir}")

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, stratify=y_train)

return X_train, X_val, X_test, y_train, y_val, y_test
```

Uporaba Albumenations za definicijo augmentacije slik

```
augmentation_pipeline = A.Compose([
    A.RandomRotate90(),
    A.Flip(),
    A.Transpose(),
    A.OneOf([
        A.Sharpen(),
        A.GaussNoise(),
    ], p=0.2),
    A.OneOf([
        A.MotionBlur(p=0.2),
        A.MedianBlur(blur_limit=3, p=0.1),
        A.Blur(blur_limit=3, p=0.1),
    ], p=0.2),
    A.ShiftScaleRotate(shift_limit=0.0625, scale_limit=0.2, rotate_limit=45, p=0.2),
    A.OneOf([
        A.OpticalDistortion(p=0.3),
        A.GridDistortion(p=0.1),
        A.PiecewiseAffine(p=0.3),
    ], p=0.2),
    A.OneOf([
    A.OneOf([
        A.CLAHE(clip_limit=2),
        A.Emboss(),
        A.RandomBrightnessContrast(),
    ], p=0.3),
    A.HueSaturationValue(p=0.3),
])
```

Definicija slik augmentiranih in originalnih za prikaz. Uporaba prejšnje funkcije za augmentacijo augmentirane slike. Prav tako sem uporabil og_image za pretvorbo originalne slike, saj sem imel probleme z izrisom črne slike

```
# Custom augmentation function
def custom_augment(image):
    # Convert the image to uint8
    image_uint8 = (image * 255).astype(np.uint8)
    # Apply augmentation pipeline
    augmented = augmentation_pipeline(image=image_uint8)
    return augmented['image']

# Custom augmentation function
def og_image(image):
    # Convert the image to uint8
    image_uint8 = (image * 255).astype(np.uint8)
    return image_uint8


# Example of augmenting an image
sample_image = og_image(X_train[0])
augmented_image = custom_augment(image=sample_image)
```

Nalaganje in augmentacija skupin slik:

```python
class TrafficSignDataLoader(Sequence):
    def __init__(self, images, labels, batch_size, augment=False):
        self.images = images
        self.labels = labels
        self.batch_size = batch_size
        self.augment = augment
        self.indices = np.arange(len(self.images))

    def __len__(self):
        return int(np.ceil(len(self.images) / self.batch_size))

    def __getitem__(self, index):
        batch_indices = self.indices[index * self.batch_size:(index + 1) * self.batch_size]
        batch_images = self.images[batch_indices]
        batch_labels = self.labels[batch_indices]

        if self.augment:
            batch_images = np.array([custom_augment(img) for img in batch_images])

        return batch_images, batch_labels

    def on_epoch_end(self):
        np.random.shuffle(self.indices)

batch_size = 32
train_loader = TrafficSignDataLoader(X_train, y_train, batch_size, augment=True)
val_loader = TrafficSignDataLoader(X_val, y_val, batch_size)
```

Gradnja modela

```python
def build_model(kernel_size):
    model = Sequential()

    # Convolutional layers
    for filters in [32, 64, 128]:
        model.add(Conv2D(filters, kernel_size, padding='same', input_shape=(64, 64, 3)))
        model.add(ELU())
        model.add(Dropout(0.5))
        model.add(MaxPooling2D(pool_size=(2, 2)))

    # Flatten and dense layers
    model.add(Flatten())
    model.add(Dense(128, activation='tanh'))
    model.add(Dense(len(np.unique(y_train)), activation='softmax'))

    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model
```

Potek skozi vse kernel velikosti. Uporabil sem samo 3x3, saj mi drugače vzame 2 uri

```python
# Train the model with different kernel sizes
# kernel_sizes = [(3, 3), (5, 5), (7, 7)]
kernel_sizes = [(3, 3)]
histories = []

for kernel_size in kernel_sizes:
    model = build_model(kernel_size)
    history = model.fit(train_loader, validation_data=val_loader, epochs=10)
    histories.append(history)

    # Evaluate on test data
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    print(f"Kernel size {kernel_size}: Test accuracy = {test_accuracy}")
```

Izpis grafično za izgubo in natančnost:

```
def plot_history(histories, kernel_sizes):
    plt.figure(figsize=(12, 8))

    for i, history in enumerate(histories):
        plt.subplot(2, len(histories), i + 1)
        plt.plot(history.history['loss'], label='train_loss')
        plt.plot(history.history['val_loss'], label='val_loss')
        plt.title(f'Loss (Kernel size {kernel_sizes[i]})')
        plt.legend()

        plt.subplot(2, len(histories), len(histories) + i + 1)
        plt.plot(history.history['accuracy'], label='train_acc')
        plt.plot(history.history['val_accuracy'], label='val_acc')
        plt.title(f'Accuracy (Kernel size {kernel_sizes[i]})')
        plt.legend()

    plt.tight_layout()
    plt.show()

plot_history(histories, kernel_sizes)
```

Nalaganje in shranjevanje:

```
def load_and_predict(image_path, model_path='traffic_sign_model.h5'):
    model = load_model(model_path)
    image = load_img(image_path, target_size=(64, 64))
    image = img_to_array(image) / 255.0
    image = np.expand_dims(image, axis=0)
    prediction = model.predict(image)
    predicted_class = np.argmax(prediction)
    return predicted_class

sample_image_path = 'Final_Test/Images/00000.ppm'
predicted_class = load_and_predict(sample_image_path)
print(f"Predicted class: {predicted_class}")
```