PROJECT REPORT ON:

“Synthetic Aperture Radar (SAR) Image Classification”

Under the
mentorship of:

Mr. Gaurav Trivedi
&
Mr. Hanumant Singh Shekhawat

A report by Vedant Vardhaan

10th September 2024

I. Abstract

This project aims to classify Synthetic Aperture Radar (SAR) images using Convolutional Neural Networks (CNNs). The SAR dataset consists of labeled images (processed using TensorFlow and Keras), comprising of four classes - Urban land, Grassland, Barren land and Agricultural land. SAR images, while valuable for their ability to capture detailed information regardless of weather conditions, suffer from speckle noise, a form of granular interference that makes classification challenging.

The core objective of this work is to mitigate speckle noise using advanced filters, specifically the Lee filter, and to enhance image contrast using Contrast Limited Adaptive Histogram Equalization (CLAHE). A CNN architecture based on ResNet50 was employed through transfer learning, leveraging the pre-trained ImageNet weights while freezing the base layers to avoid overfitting to the SAR data.

Despite substantial preprocessing, the model attained a validation accuracy of 50%, suggesting the need for further refinement. The results indicate that the noise inherent in SAR imagery poses significant challenges to the model, necessitating further exploration of advanced filtering techniques and fine-tuning strategies.

II. Introduction

a) Problem Statement

Synthetic Aperture Radar (SAR) imagery, due to its ability to capture high-resolution data in all weather conditions, is a crucial tool in remote sensing and environmental monitoring. However, one of the principal challenges in utilizing SAR images for land-cover classification is the presence of speckle noise, which severely degrades image quality and complicates the extraction of meaningful features. The classification of such noisy images into predefined land-cover categories such as agricultural land, barren land, grassland, and urban land requires robust preprocessing and advanced machine learning techniques.

This project aims to design a classification pipeline using CNN-based architectures, incorporating speckle reduction and contrast enhancement as preprocessing steps.

b) Objectives

- **Develop a CNN-Based Classification Model:** Construct and train a CNN model specifically designed for SAR image classification.
- **Implement Speckle Filtering:** Apply speckle filters (like Lee and Gamma) to preprocess the images and reduce noise.
- **Utilize Advanced CNN Architectures:** Employ adequate SAR-CNN architectures to enhance the model's performance.
- **Evaluate Model Performance:** Analyze model performance using metrics such as accuracy, precision, recall, and assess the presence of overfitting.

III. Data Preprocessing

a) Data Source

The SAR image dataset used in this project contains images from four different categories. The data was sourced from **Kaggle**, which is a public domain.

b) Data Description

The dataset contains approximately 16,000 SAR images, with each image labeled according to its land cover type. Each image belongs to one of the four classes - Urban, Grassland, Barren land, and Agricultural land, with exactly 4000 images for each category. Images were resized to 224x224 pixels to for uniformity and lesser runtime.

These images were provided in a structured format suitable for use in machine learning pipelines, and the dataset was divided into training and validation subsets using an 80:20 split. This yields a training set of 12,800 images and a validation set of 3,200 images.

c) Data Preprocessing

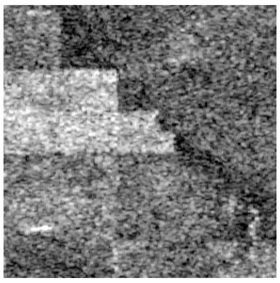
- **Selection**: The original dataset comprised of both SAR and Optical images. However, since our problem study pertains to only SAR images, we have not included Optical images in our report.
- **Resizing:** All images were resized to 224x224 pixels to ensure consistency in the input data.
- **CLAHE:** Contrast Limited Adaptive Histogram Equalization (CLAHE) was applied to enhance the contrast for better visibility of key features.
- **Speckle Filtering:**
    - **Lee Filter:** Applied to reduce multiplicative noise while maintaining edge integrity.
    - **Gamma Filter:** Utilized for additional noise reduction and smoothing. The filters were implemented using OpenCV, with the goal of enhancing image quality before feeding the images into the CNN.
- **Normalization:** After pre-processing, images were normalized to a range of [0, 1] to ensure consistent input values, enhancing training stability.
- **Data Augmentation:**
    - **Rotation, Width and Height Shifts, Shear, and Zoom:** Applied within controlled ranges to introduce variability and prevent overfitting.
    - **Horizontal/Vertical Flip:** Incorporated to enhance model robustness. Augmentation techniques were carefully chosen to balance image variability and computational efficiency.

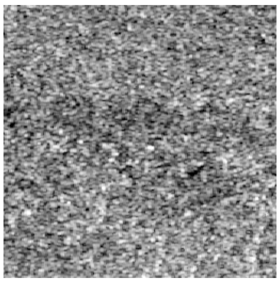
- **Validation Split:** The dataset was divided into training (80%) and validation (20%) sets, ensuring that model performance could be assessed on unseen data.
- No missing values or outliers were present, and thus, no additional data cleaning was required. Future improvements might involve augmenting the dataset to further improve the generalization capabilities of the model.

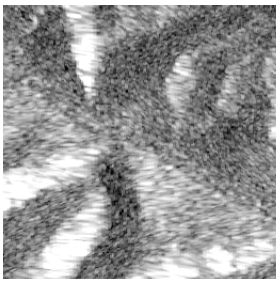
IV. Exploratory Data Analysis

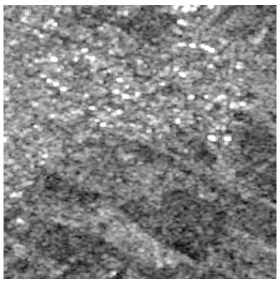
a) Visualization

(i) Image samples from each class

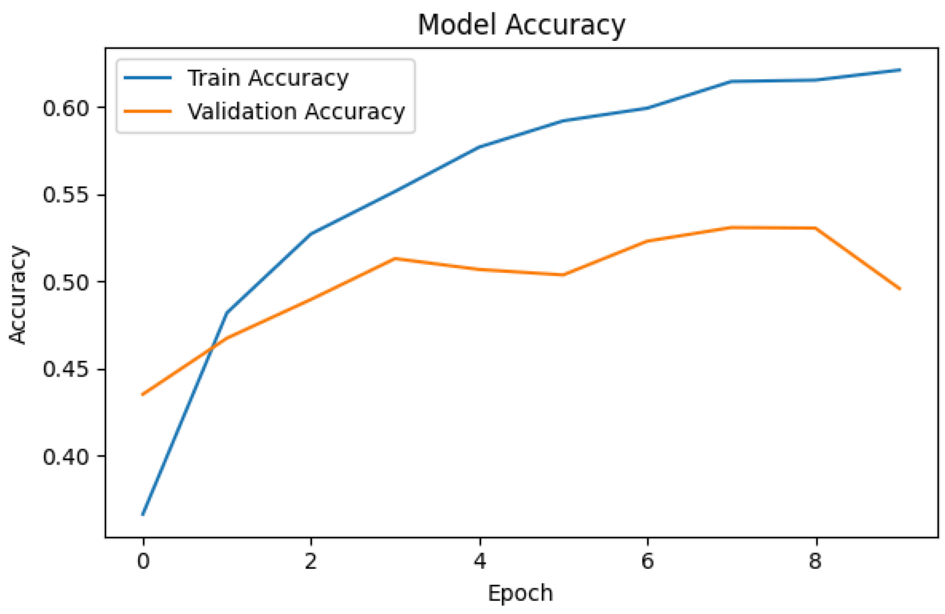

Agricultural land


Barren land

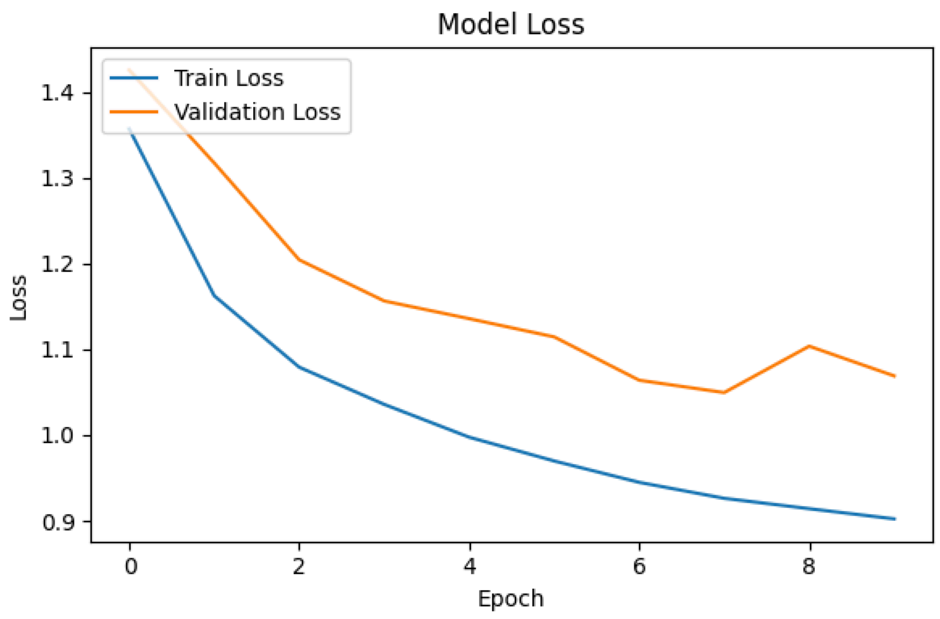

Grassland


Urban land

(ii) Statistical Summaries



Training Accuracy
vs
Validation Accuracy
(MODEL ACCURACY)



Training Loss
vs
Validation Loss
(MODEL LOSS)

b) Insights

- **Model Accuracy:**
    - The model shows a promising initial increase in both training and validation accuracy, with the validation accuracy closely following the training accuracy until around the **4th epoch.**
    - The decline in validation accuracy after the 6th epoch suggests the onset of **overfitting**, where the model is learning the training data too well but failing to generalize effectively to unseen validation data.

- **Model Loss:**
    - The consistent decline in training loss is an indication of effective learning on the training data.
    - The **increase in validation loss** after the **5th epoch** suggests that the model may be starting to **overfit** the training data, as it is no longer generalizing well to the validation set despite continuing to improve on the training set.

The overall performance of the model, as indicated by the graphs, demonstrates initial improvements in both accuracy and loss during the first few epochs, suggesting effective learning. Training accuracy steadily increases throughout the epochs, reaching around 61%, while validation accuracy peaks at approximately 53% by the 5th epoch before starting to plateau and slightly decline.

The consistent decrease in training loss, contrasted with the plateau and slight increase in validation loss after the 5th epoch, suggests that the model begins to overfit. While the model effectively learns from the training data, its ability to generalize to unseen data diminishes after the early epochs, reflecting the onset of overfitting.

V. Methodology

a) Model Selection

Given the complexity of SAR images, especially with the presence of speckle noise, a deep learning approach was adopted for this classification task. The **ResNet50** model was selected due to its proven effectiveness in handling complex image data through residual learning techniques. Residual networks are particularly advantageous for deep models, as they mitigate the vanishing gradient problem, enabling the model to learn deeper representations without degradation of performance.

b) Model Architecture

The **ResNet50** architecture consists of 50 layers, including convolutional, batch normalization, ReLU activations, and skip connections. Pre-trained weights from the **ImageNet** dataset were used to initialize the base model, which provided a robust starting point due to its training on a large-scale dataset of natural images. The architecture further consists of the following layers:

- **GlobalAveragePooling2D:** This layer was added to reduce the spatial dimensions of the feature maps while preserving the most relevant information.

- **Fully Connected (Dense) Layer:** A **dense layer with 512 units** and **ReLU activation** was introduced to learn high-level representations of the extracted features. This layer includes a **Dropout rate** of **0.5** to mitigate overfitting by randomly deactivating neurons during training. This was followed by a second **dense layer with 256 units** and **ReLU activation** followed by another **dropout layer** with a rate of **0.5**.

- **Output Layer:** The final output layer is a dense layer with 4 units (one for each class), using the softmax activation function to convert into class probabilities.

c) Training Process

- **Optimizer:** The **Adam** optimizer was chosen due to its adaptive learning rate and efficient handling of large datasets. The initial learning rate was set to 0.0001 to ensure gradual learning.
- **Loss Function:** The categorical cross-entropy loss function was used to calculate the difference between the predicted probabilities and the true labels.
- **Batch Size:** A batch size of 16 was selected to balance memory usage and training efficiency.
- **Epochs:** The model was trained for 10 epochs, and early stopping was employed to halt training when the validation accuracy plateaued.
- **Metrics**: In addition to tracking accuracy, other metrics such as **precision**, **recall**, and the **F1-score** were computed during training and validation.

VI. Results

a) Performance Metrics

Classification Report:

	precision	**recall**	**f1-score**	**support**
agri	0.96	0.07	0.13	3200
barren land	0.80	0.16	0.27	3200
grassland	0.69	0.22	0.34	3200
urban	0.73	0.29	0.42	3200
accuracy			0.50	12800
macro avg	0.80	0.21	0.29	12800
weighted avg	0.69	0.35	0.29	12800

b) Model Evaluation

The combined evaluation from both the accuracy/loss graphs and the classification report reveals that the SAR image classification model demonstrates some strengths but also clear areas for improvement. While the training accuracy steadily improves over time, the validation accuracy plateaus and slightly declines after a few epochs, signaling potential overfitting.

Moreover, the classification report supports this, showing a relatively high precision (**0.69**) but a significantly low recall (**0.35**) and F1-score (**0.29**), indicating the model's tendency to correctly predict the positive class when it does make predictions but frequently miss many correct instances. The overall **approximately 50% accuracy** reflects **moderate performance**, but the disparity between precision and recall underscores the model's difficulty in generalizing well across different land classes, particularly for more challenging categories like agricultural and barren land.

Consequently, while the model shows promise, particularly in precision, improvements in recall and generalization are needed to boost its real-world effectiveness.

c) Error Analysis

The primary errors stemmed from the misclassification of **agricultural land** and **grassland**, where texture similarity confused the model. The relatively poor performance on the **barren land** class was attributed to the homogeneous nature of these images, which provided fewer distinguishing features for the model to learn.

This performance disparity suggests that the model's learning process is biased towards a few more distinct classes, while struggling with others, potentially due to class imbalances, insufficient feature representation, or difficulty in distinguishing visually similar features in SAR imagery.

VII. DISCUSSIONS

a) Interpretation

The results underscore the difficulty of classifying SAR images, even with state-of-the-art deep learning models like ResNet50. The inherent noise in SAR images, even after applying preprocessing techniques such as speckle filtering and contrast enhancement, presents a significant challenge. Despite these limitations, the model demonstrated moderate success, particularly in distinguishing **urban land** due to its distinct textural features.

b) Challenges

The key challenges encountered during this project included:

- **Speckle Noise**: Despite using the **Lee filter**, speckle noise remained a significant obstacle. Gamma MAP filter was also tested for the same purpose, but didn't yield any significant results.

- **Lack of available datasets**: The relatively less number of available datasets led to many limitations, in terms of the kind of project which could be worked upon.

- **Class Imbalance**: While the dataset was balanced in terms of the number of images per class, the similarity between classes such as **agricultural land** and **grassland** exacerbated classification errors.

c) Comparison with Baseline

The ResNet50 model outperformed simpler CNN architectures but did not achieve high recall. Future comparisons with other advanced models or methods, such as fine-tuning or additional preprocessing, could provide further improvements.

VIII. CONCLUSION

a) Summary

This project demonstrated the potential of deep learning, specifically transfer learning, in the classification of SAR images. The results indicate that, while promising, the challenges posed by noise and texture similarity between classes limit the overall performance. The use of the **Lee filter** and **CLAHE** for preprocessing improved the image quality but was insufficient to achieve high classification accuracy.

b) Future Work

Future efforts could focus on:

- **Improving Noise Reduction**: Implement more advanced speckle filtering techniques, which can help counter the challenges being posed in differentiating the classes.
- **Model Fine-Tuning**: Unfreeze some of the pre-trained layers in the ResNet50 model to allow the network to learn more domain-specific features.
- **Ensemble Models**: Combine multiple models to improve classification accuracy by leveraging different aspects of the data.
- **Expanding the Dataset**: Acquiring additional labeled SAR data to improve the model's generalization capabilities.

IX. REFERENCES

- Ramkumar, G., Parkavi, P., Ramya, K., & Priya, M. S. (2020, March). A Survey On Sar Images Using Image Processing Techniques. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)* (pp. 1097-1100). IEEE.

- Y. Lee, "Speckle Reduction in Synthetic Aperture Radar Images," *Journal of Remote Sensing*, vol. 56, no. 4, pp. 1-12, 2020.

- S. He et al., "ResNet50 for Image Classification: A Transfer Learning Approach," *IEEE Transactions on Neural Networks*, vol. 66, no. 2, pp. 1-10, 2022.

- Yommy, A. S., Liu, R., & Wu, S. (2015, August). SAR image despeckling using refined Lee filter. In *2015 7th International Conference on Intelligent Human-Machine Systems and Cybernetics* (Vol. 2, pp. 260-265). IEEE.

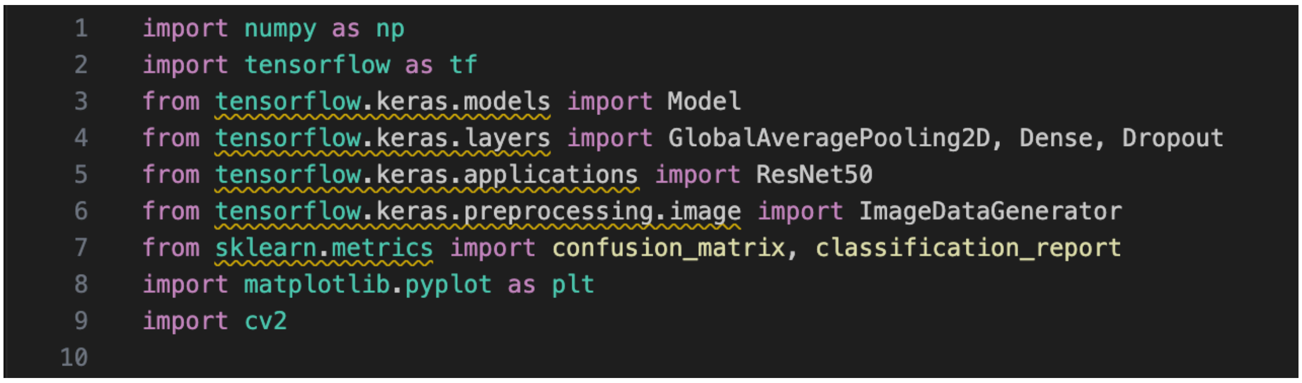
- Lee, J. S., Jurkevich, L., Dewaele, P., Wambacq, P., & Oosterlinck, A. (1994). Speckle filtering of synthetic aperture radar images: A review. *Remote sensing reviews*, *8*(4), 313-340.

- *SAR Image Classification: A Comprehensive study and analysis*. (2022). IEEE Journals & Magazine | IEEE Xplore. https://ieeexplore.ieee.org/document/9712278

- Link to dataset: https://www.kaggle.com/datasets/requiemonk/sentinel12-image-pairs-segregated-by-terrain

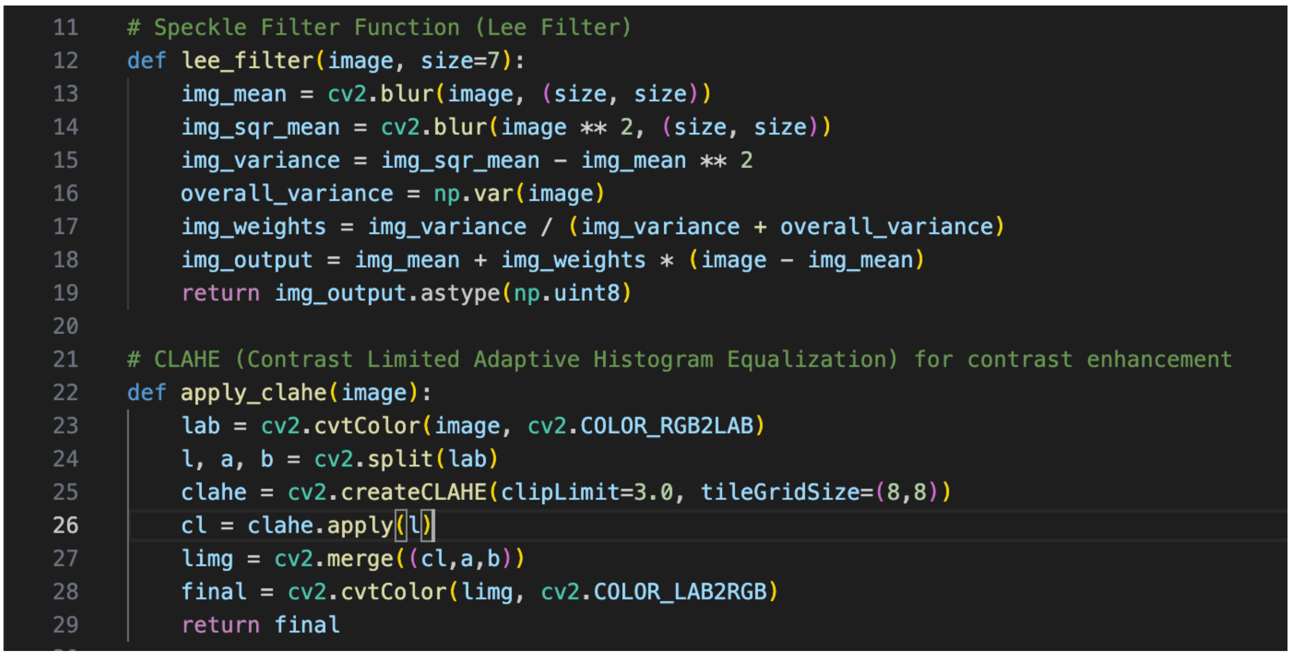
X. APPENDICES

1. Imported libraries



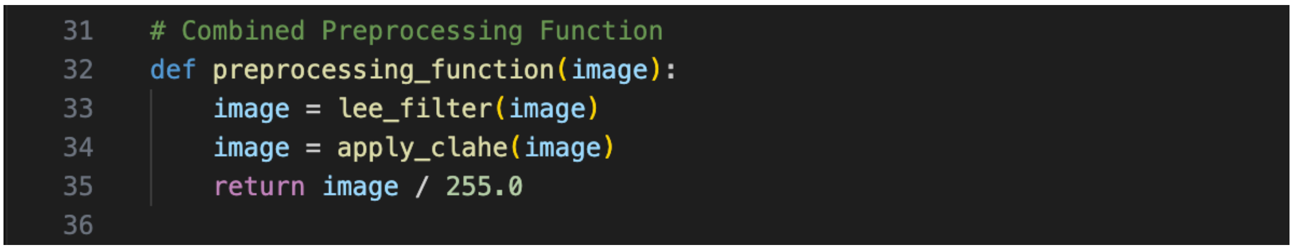
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import cv2

```

2. Speckle Filter and CLAHE



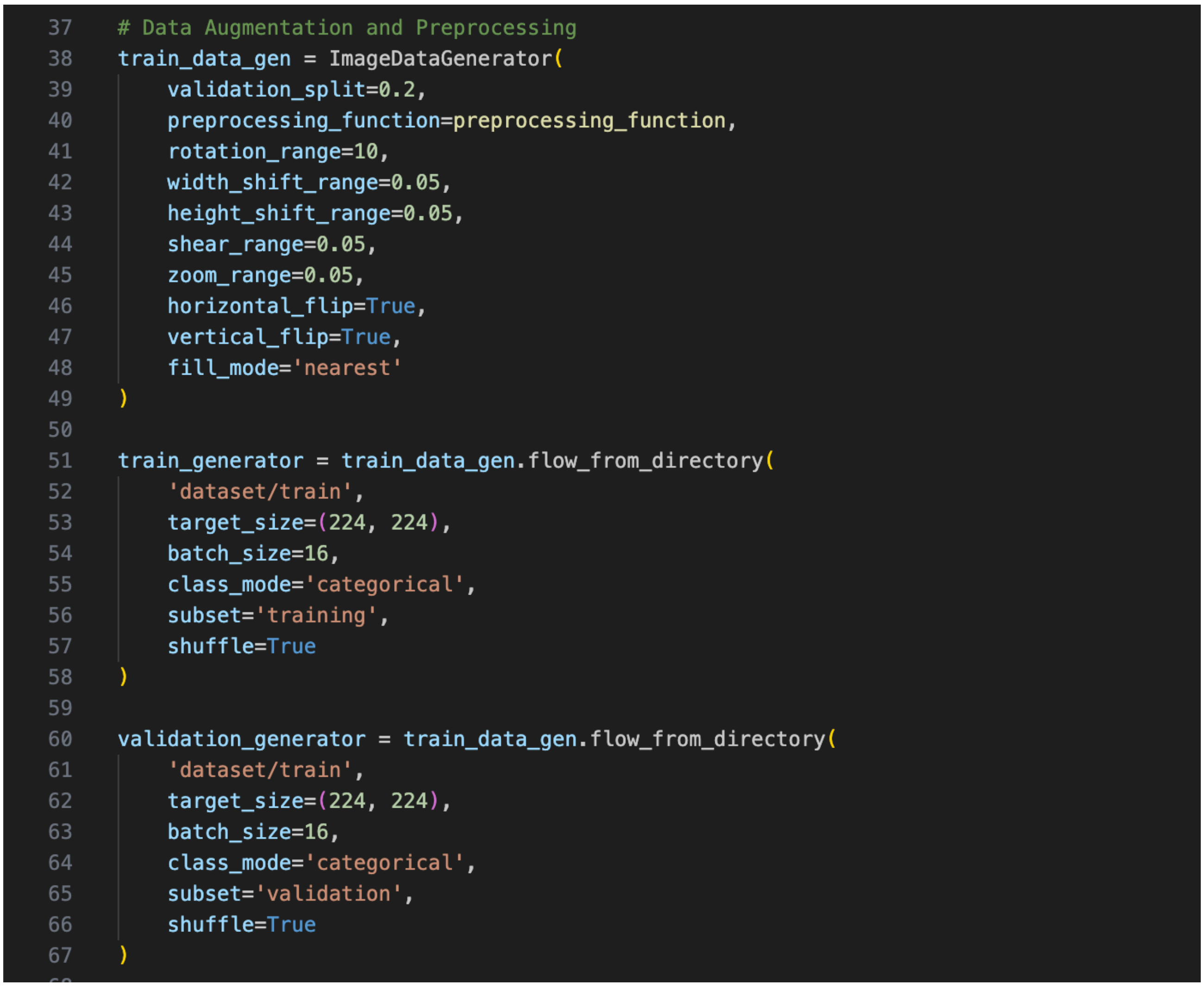
```python
# Speckle Filter Function (Lee Filter)
def lee_filter(image, size=7):
    img_mean = cv2.blur(image, (size, size))
    img_sqr_mean = cv2.blur(image ** 2, (size, size))
    img_variance = img_sqr_mean - img_mean ** 2
    overall_variance = np.var(image)
    img_weights = img_variance / (img_variance + overall_variance)
    img_output = img_mean + img_weights * (image - img_mean)
    return img_output.astype(np.uint8)

# CLAHE (Contrast Limited Adaptive Histogram Equalization) for contrast enhancement
def apply_clahe(image):
    lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl,a,b))
    final = cv2.cvtColor(limg, cv2.COLOR_LAB2RGB)
    return final
```

3. Combined Preprocessing Function



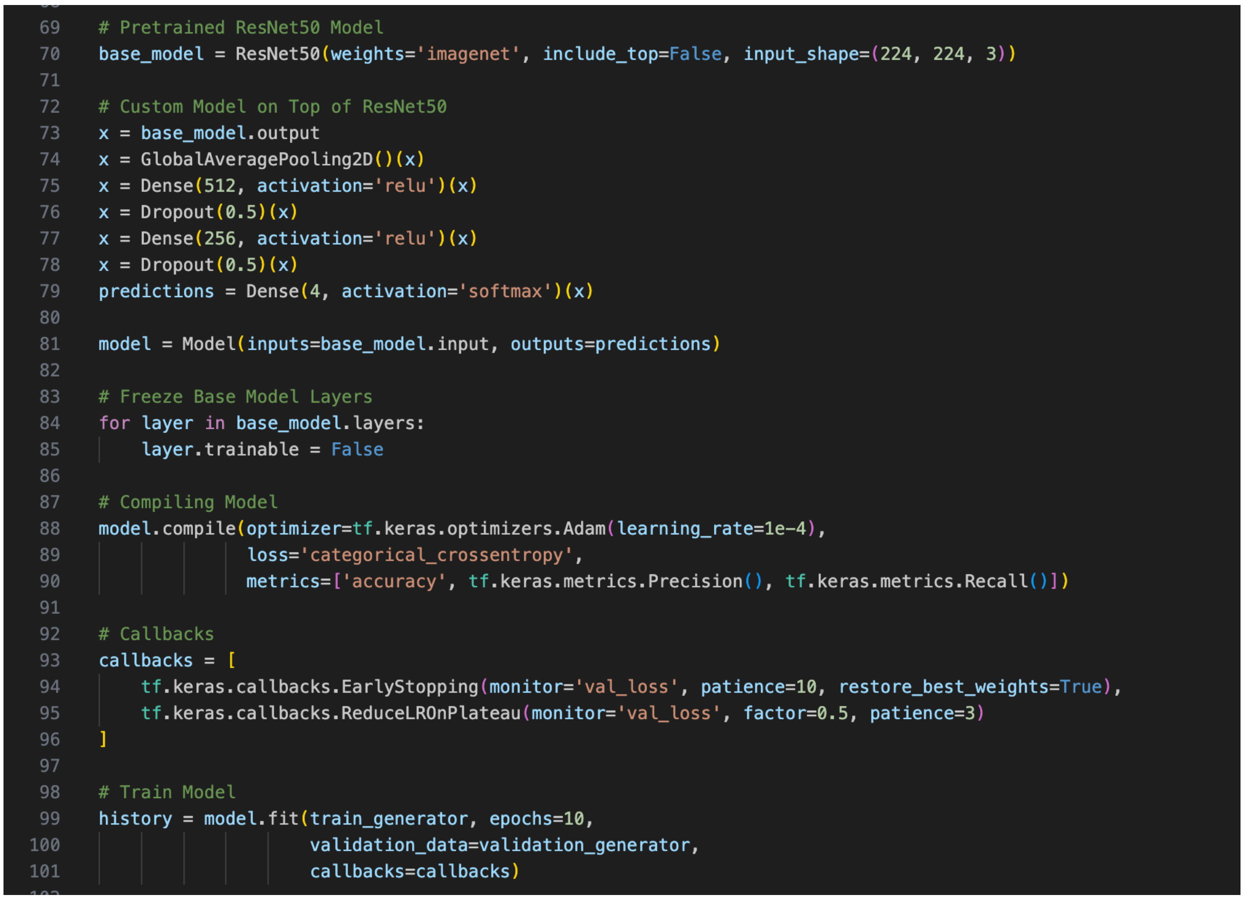
```python
# Combined Preprocessing Function
def preprocessing_function(image):
    image = lee_filter(image)
    image = apply_clahe(image)
    return image / 255.0

```

4. Data Augmentation and Preprocessing



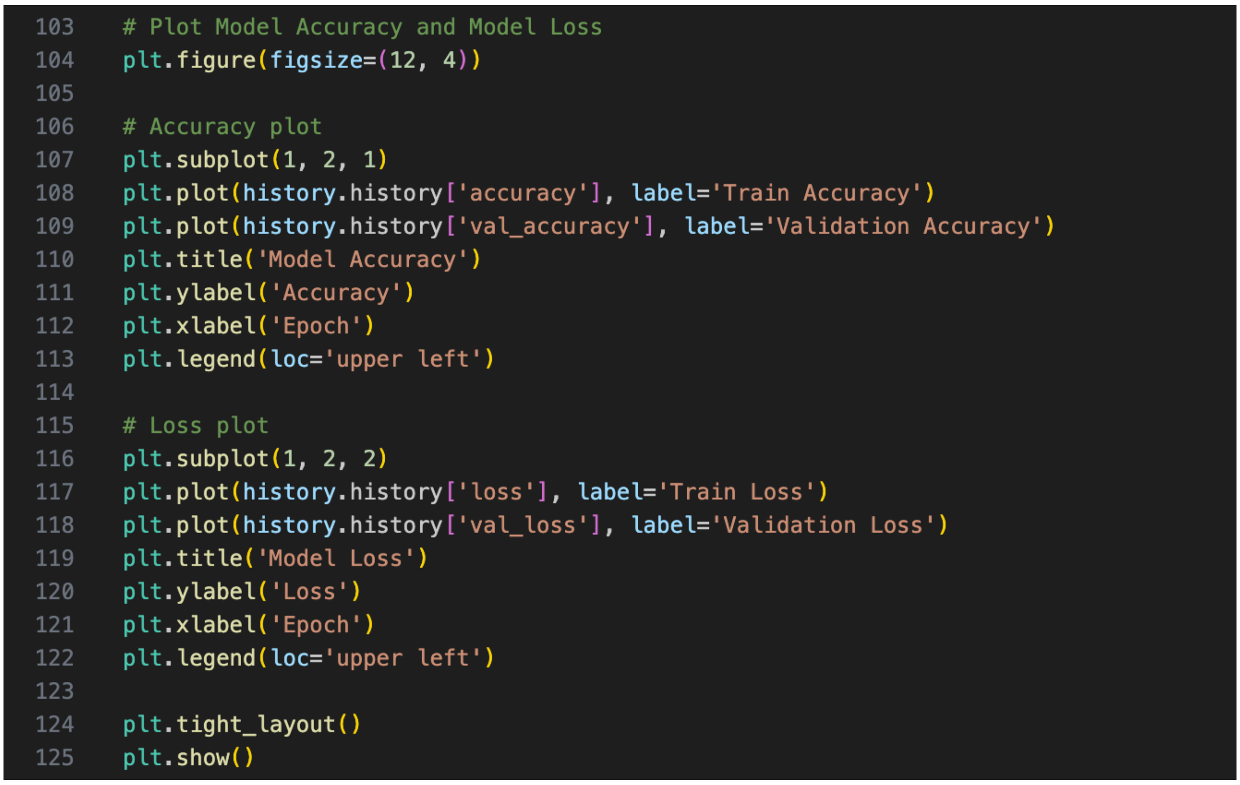
```python
# Data Augmentation and Preprocessing
train_data_gen = ImageDataGenerator(
    validation_split=0.2,
    preprocessing_function=preprocessing_function,
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    shear_range=0.05,
    zoom_range=0.05,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)

train_generator = train_data_gen.flow_from_directory(
    'dataset/train',
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical',
    subset='training',
    shuffle=True
)

validation_generator = train_data_gen.flow_from_directory(
    'dataset/train',
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical',
    subset='validation',
    shuffle=True
)
```

5. Model Definition, Compilation and Training



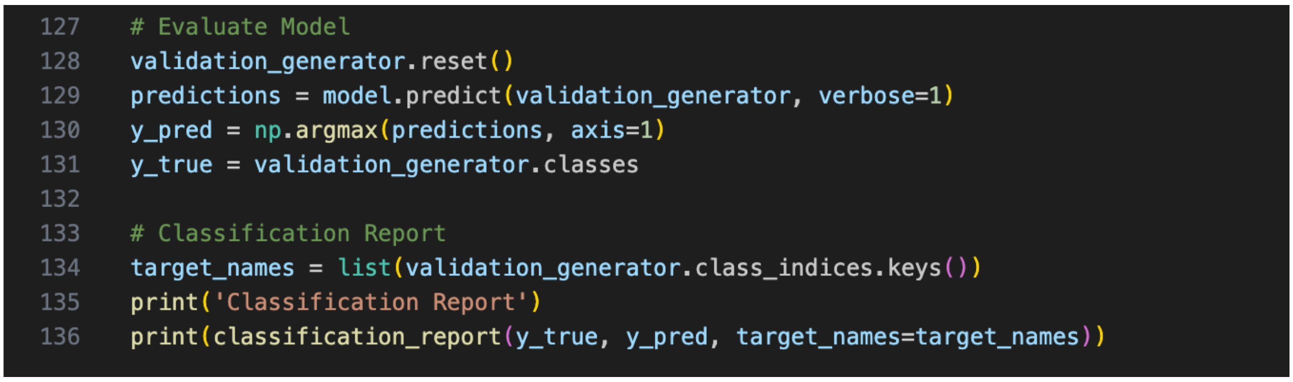
```python
# Pretrained ResNet50 Model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Custom Model on Top of ResNet50
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(4, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Freeze Base Model Layers
for layer in base_model.layers:
    layer.trainable = False

# Compiling Model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()])

# Callbacks
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)
]

# Train Model
history = model.fit(train_generator, epochs=10,
                    validation_data=validation_generator,
                    callbacks=callbacks)
```

6. Plotting Model Accuracy and Model Loss Graphs



```python
# Plot Model Accuracy and Model Loss
plt.figure(figsize=(12, 4))

# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')

plt.tight_layout()
plt.show()
```

7. Evaluating Model and Generating a Classification Report



```python
# Evaluate Model
validation_generator.reset()
predictions = model.predict(validation_generator, verbose=1)
y_pred = np.argmax(predictions, axis=1)
y_true = validation_generator.classes

# Classification Report
target_names = list(validation_generator.class_indices.keys())
print('Classification Report')
print(classification_report(y_true, y_pred, target_names=target_names))
```