Sheep_Full_Images_Classification

April 28, 2024

[6]:
```
import numpy as np
import pandas as pd
import os
from PIL import Image
from keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.applications import VGG19
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model, load_model
from keras.layers import Dense, GlobalAveragePooling2D, Dropout,␣
 ↪BatchNormalization, Flatten, Conv2D, Activation, MaxPooling2D
from keras.callbacks import EarlyStopping, Callback, ModelCheckpoint
from sklearn.model_selection import train_test_split, KFold
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras.optimizers import SGD
from sklearn.utils import shuffle
from sklearn.model_selection import StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
```

[9]:
```
# Load images and labels
path = '/content/drive/MyDrive/Deep_learning_projects/SheepFullImages'
sheep_breed_list = os.listdir(path)
print(sheep_breed_list)
```

```
['Suffolk', 'Poll Dorset', 'White Suffolk', 'Marino']
```

[10]:
```
def load_and_preprocess_images(path, target_size=(224, 224)):
    img_data_list = []
    labels = []

    for idx, sheep_breed in enumerate(sheep_breed_list):
        sheep_breed_files = os.listdir(os.path.join(path, sheep_breed))
        print(sheep_breed_files)
        for img_file in sheep_breed_files:
            sheep_image_path = os.path.join(path, sheep_breed, img_file)
```

```
            try:
                img = image.load_img(sheep_image_path, target_size=target_size)
                img_array = image.img_to_array(img)
                img_array = preprocess_input(img_array)
                img_data_list.append(img_array)
                labels.append(idx)
            except Exception as e:
                print(f"Error loading image {img_file}: {e}")

    # Convert lists to numpy arrays and preprocess
    img_data = np.array(img_data_list)
    labels = np.array(labels)

    # Shuffle data
    img_data, labels = shuffle(img_data, labels, random_state=777)

    return img_data, labels, sheep_breed_list

img_data_path = '/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/
 ↪sheep_img_data.npy'
labels_path = '/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/
 ↪sheep_labels.npy'

if os.path.exists(img_data_path) and os.path.exists(labels_path):
    img_data = np.load(img_data_path)
    labels = np.load(labels_path)
else:
    img_data, labels, sheep_breed_list = load_and_preprocess_images(path)
    # Save the processed data
    np.save(img_data_path, img_data)
    np.save(labels_path, labels)

print('Data Shape:', img_data.shape)
print('Labels Shape:', labels.shape)
```

```
Data Shape: (1619, 224, 224, 3)
Labels Shape: (1619,)
```

[11]:

```
# One-hot encode labels
labels_categorical = to_categorical(labels, num_classes=len(sheep_breed_list))

print(labels_categorical.shape)
```

```
(1619, 4)
```

[12]:

```
def build_model():
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,
↪224, 3))

# Freeze first 10 convolutional layers
for layer in base_model.layers[:15]:
    layer.trainable = False

x = base_model.output

# Flatten to prepare for the fully connected layers
x = Flatten()(x)

# Fully connected + ReLU
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)

# Adding output layer
predictions = Dense(len(sheep_breed_list), activation='softmax')(x)

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

# Custom learning rates
lr_mult = {}

# Set learning rate to 0 for the first 10 layers
for layer in model.layers[:15]:
    lr_mult[layer.name + '/kernel:0'] = 0.0
    lr_mult[layer.name + '/bias:0'] = 0.0

# Set learning rate to 0.0001 for the next Conv2D layers and the following
↪layers
for layer in model.layers[15:-4]:
    if isinstance(layer, Conv2D):
        lr_mult[layer.name + '/kernel:0'] = 0.0001
        lr_mult[layer.name + '/bias:0'] = 0.0001

# Set learning rate to 10 for the last two dense layers
for layer in model.layers[-4:]:
    lr_mult[layer.name + '/kernel:0'] = 10
    lr_mult[layer.name + '/bias:0'] = 10

# Use SGD optimizer with initial learning rate of 1e-4
optimizer = SGD(learning_rate=1e-4,momentum=0.9)

# Compile the model
```

```
    model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'], loss_weights=[1.] + [10.]*2)

    return model

accuracy_values = []
precision_values = []
recall_values = []
f1_values = []

train_acc_list = []
val_acc_list = []
train_loss_list = []
val_loss_list = []

# KFold Cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=777)
fold_var = 1

for train_index, test_index in kf.split(img_data):
    X_train, X_test = img_data[train_index], img_data[test_index]
    y_train, y_test = labels_categorical[train_index],␣
↪labels_categorical[test_index]

    model = build_model()

    #early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1,␣
↪mode='min')
    save_path='/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/
↪sheepFace_one.h5'
    checkpoint =␣
↪ModelCheckpoint(save_path,monitor='val_loss',save_best_only=True,verbose=1)
    history = model.fit(X_train, y_train, batch_size=10, epochs=10, verbose=1,␣
↪validation_split=0.2, callbacks=[checkpoint])

    print("\n")
    print(f'Evaluating the Test metrics')
    model=load_model(save_path)
    # Evaluate the model
    scores = model.evaluate(X_test, y_test, verbose=1)
    print(f'Test loss for fold {fold_var}: {scores[0]}')
    print(f'Test accuracy for fold {fold_var}: {scores[1]}')

    accuracy_values.append(scores[1])  # Appending accuracy to the list
```

```
    # Get predictions from the model
    y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels back␣
 ↪to categorical labels
    y_pred = np.argmax(model.predict(X_test), axis=1)  # Get predictions from␣
 ↪the model

    # Calculate and append evaluation metrics for this fold
    precision = precision_score(y_true, y_pred, average='weighted',␣
 ↪zero_division=1)
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    precision_values.append(precision)
    recall_values.append(recall)
    f1_values.append(f1)

    # Print evaluation metrics for this fold
    print(f'Precision for fold {fold_var}: {precision}')
    print(f'Recall for fold {fold_var}: {recall}')
    print(f'F1 Score for fold {fold_var}: {f1}')
    print("\n")

    train_acc_list.append(history.history['accuracy'])
    val_acc_list.append(history.history['val_accuracy'])
    train_loss_list.append(history.history['loss'])
    val_loss_list.append(history.history['val_loss'])

    # Increment the fold number
    fold_var += 1

# Calculate standard deviation of accuracy across folds
accuracy_std_dev = np.std(accuracy_values)
print(f'Standard deviation of accuracy across folds: {accuracy_std_dev}')

# Calculate average of accuracy,precision, recall, and F1 across folds
avg_accuracy = np.mean(accuracy_values)
avg_precision = np.mean(precision_values)
avg_recall = np.mean(recall_values)
avg_f1 = np.mean(f1_values)

print(f'Average Accuracy across folds: {avg_accuracy}')
print(f'Average Precision across folds: {avg_precision}')
print(f'Average Recall across folds: {avg_recall}')
print(f'Average F1 Score across folds: {avg_f1}')
```

```
Epoch 1/10
104/104 [==============================] - ETA: 0s - loss: 1.5677 - accuracy:
```

```
0.5502
Epoch 1: val_loss improved from inf to 0.93505, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(

104/104 [==============================] - 29s 218ms/step - loss: 1.5677 -
accuracy: 0.5502 - val_loss: 0.9350 - val_accuracy: 0.5792
Epoch 2/10
104/104 [==============================] - ETA: 0s - loss: 0.5933 - accuracy:
0.7683
Epoch 2: val_loss improved from 0.93505 to 0.49222, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 84ms/step - loss: 0.5933 -
accuracy: 0.7683 - val_loss: 0.4922 - val_accuracy: 0.7876
Epoch 3/10
104/104 [==============================] - ETA: 0s - loss: 0.3133 - accuracy:
0.8851
Epoch 3: val_loss improved from 0.49222 to 0.41848, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 10s 95ms/step - loss: 0.3133 -
accuracy: 0.8851 - val_loss: 0.4185 - val_accuracy: 0.8610
Epoch 4/10
104/104 [==============================] - ETA: 0s - loss: 0.1248 - accuracy:
0.9595
Epoch 4: val_loss improved from 0.41848 to 0.29627, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 89ms/step - loss: 0.1248 -
accuracy: 0.9595 - val_loss: 0.2963 - val_accuracy: 0.9112
Epoch 5/10
104/104 [==============================] - ETA: 0s - loss: 0.0681 - accuracy:
0.9797
Epoch 5: val_loss did not improve from 0.29627
104/104 [==============================] - 9s 83ms/step - loss: 0.0681 -
accuracy: 0.9797 - val_loss: 0.3818 - val_accuracy: 0.8842
Epoch 6/10
103/104 [============================>.] - ETA: 0s - loss: 0.0374 - accuracy:
0.9893
Epoch 6: val_loss improved from 0.29627 to 0.25930, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 10s 93ms/step - loss: 0.0372 -
accuracy: 0.9894 - val_loss: 0.2593 - val_accuracy: 0.9189
Epoch 7/10
103/104 [============================>.] - ETA: 0s - loss: 0.0136 - accuracy:
0.9981
```

```
Epoch 7: val_loss improved from 0.25930 to 0.23679, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 82ms/step - loss: 0.0136 -
accuracy: 0.9981 - val_loss: 0.2368 - val_accuracy: 0.9151
Epoch 8/10
103/104 [============================>.] - ETA: 0s - loss: 0.0056 - accuracy:
1.0000
Epoch 8: val_loss improved from 0.23679 to 0.21585, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 82ms/step - loss: 0.0056 -
accuracy: 1.0000 - val_loss: 0.2158 - val_accuracy: 0.9305
Epoch 9/10
103/104 [============================>.] - ETA: 0s - loss: 0.0032 - accuracy:
1.0000
Epoch 9: val_loss did not improve from 0.21585
104/104 [==============================] - 8s 79ms/step - loss: 0.0032 -
accuracy: 1.0000 - val_loss: 0.2173 - val_accuracy: 0.9228
Epoch 10/10
103/104 [============================>.] - ETA: 0s - loss: 0.0024 - accuracy:
1.0000
Epoch 10: val_loss did not improve from 0.21585
104/104 [==============================] - 8s 79ms/step - loss: 0.0024 -
accuracy: 1.0000 - val_loss: 0.2197 - val_accuracy: 0.9228


Evaluating the Test metrics
11/11 [==============================] - 10s 295ms/step - loss: 0.3883 -
accuracy: 0.8827
Test loss for fold 1: 0.3883349597454071
Test accuracy for fold 1: 0.8827160596847534
11/11 [==============================] - 2s 137ms/step
Precision for fold 1: 0.885663930768558
Recall for fold 1: 0.8827160493827161
F1 Score for fold 1: 0.883510431785993


Epoch 1/10
104/104 [==============================] - ETA: 0s - loss: 1.6973 - accuracy:
0.5145
Epoch 1: val_loss improved from inf to 0.86636, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
104/104 [==============================] - 10s 90ms/step - loss: 1.6973 -
```

```
accuracy: 0.5145 - val_loss: 0.8664 - val_accuracy: 0.6448
Epoch 2/10
104/104 [==============================] - ETA: 0s - loss: 0.6605 - accuracy:
0.7249
Epoch 2: val_loss improved from 0.86636 to 0.51221, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 86ms/step - loss: 0.6605 -
accuracy: 0.7249 - val_loss: 0.5122 - val_accuracy: 0.8108
Epoch 3/10
104/104 [==============================] - ETA: 0s - loss: 0.3179 - accuracy:
0.8861
Epoch 3: val_loss improved from 0.51221 to 0.38019, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 10s 97ms/step - loss: 0.3179 -
accuracy: 0.8861 - val_loss: 0.3802 - val_accuracy: 0.8687
Epoch 4/10
104/104 [==============================] - ETA: 0s - loss: 0.1594 - accuracy:
0.9488
Epoch 4: val_loss improved from 0.38019 to 0.32078, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 85ms/step - loss: 0.1594 -
accuracy: 0.9488 - val_loss: 0.3208 - val_accuracy: 0.8996
Epoch 5/10
104/104 [==============================] - ETA: 0s - loss: 0.0806 - accuracy:
0.9788
Epoch 5: val_loss did not improve from 0.32078
104/104 [==============================] - 8s 81ms/step - loss: 0.0806 -
accuracy: 0.9788 - val_loss: 0.3937 - val_accuracy: 0.8764
Epoch 6/10
103/104 [============================>.] - ETA: 0s - loss: 0.0436 - accuracy:
0.9845
Epoch 6: val_loss improved from 0.32078 to 0.29129, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 91ms/step - loss: 0.0435 -
accuracy: 0.9846 - val_loss: 0.2913 - val_accuracy: 0.9035
Epoch 7/10
103/104 [============================>.] - ETA: 0s - loss: 0.0171 - accuracy:
0.9990
Epoch 7: val_loss improved from 0.29129 to 0.28810, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 83ms/step - loss: 0.0173 -
accuracy: 0.9990 - val_loss: 0.2881 - val_accuracy: 0.9073
Epoch 8/10
103/104 [============================>.] - ETA: 0s - loss: 0.0071 - accuracy:
0.9990
Epoch 8: val_loss improved from 0.28810 to 0.24419, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 83ms/step - loss: 0.0071 -
```

```
accuracy: 0.9990 - val_loss: 0.2442 - val_accuracy: 0.9035
Epoch 9/10
104/104 [==============================] - ETA: 0s - loss: 0.0035 - accuracy:
1.0000
Epoch 9: val_loss did not improve from 0.24419
104/104 [==============================] - 8s 80ms/step - loss: 0.0035 -
accuracy: 1.0000 - val_loss: 0.2603 - val_accuracy: 0.9073
Epoch 10/10
104/104 [==============================] - ETA: 0s - loss: 0.0027 - accuracy:
1.0000
Epoch 10: val_loss did not improve from 0.24419
104/104 [==============================] - 8s 81ms/step - loss: 0.0027 -
accuracy: 1.0000 - val_loss: 0.2639 - val_accuracy: 0.9073


Evaluating the Test metrics
11/11 [==============================] - 2s 125ms/step - loss: 0.2831 -
accuracy: 0.9228
Test loss for fold 2: 0.28305599093437195
Test accuracy for fold 2: 0.9228395223617554
11/11 [==============================] - 2s 135ms/step
Precision for fold 2: 0.9236707633133129
Recall for fold 2: 0.9228395061728395
F1 Score for fold 2: 0.9228414232212712


Epoch 1/10
103/104 [============================>.] - ETA: 0s - loss: 2.0800 - accuracy:
0.4194
Epoch 1: val_loss improved from inf to 1.07769, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
104/104 [==============================] - 10s 88ms/step - loss: 2.0730 -
accuracy: 0.4208 - val_loss: 1.0777 - val_accuracy: 0.5637
Epoch 2/10
104/104 [==============================] - ETA: 0s - loss: 0.8500 - accuracy:
0.6525
Epoch 2: val_loss improved from 1.07769 to 0.79991, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 84ms/step - loss: 0.8500 -
accuracy: 0.6525 - val_loss: 0.7999 - val_accuracy: 0.6873
Epoch 3/10
104/104 [==============================] - ETA: 0s - loss: 0.6054 - accuracy:
```

```
0.7654
Epoch 3: val_loss improved from 0.79991 to 0.52569, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 85ms/step - loss: 0.6054 -
accuracy: 0.7654 - val_loss: 0.5257 - val_accuracy: 0.8069
Epoch 4/10
104/104 [==============================] - ETA: 0s - loss: 0.4074 - accuracy:
0.8446
Epoch 4: val_loss improved from 0.52569 to 0.45125, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 84ms/step - loss: 0.4074 -
accuracy: 0.8446 - val_loss: 0.4512 - val_accuracy: 0.8378
Epoch 5/10
104/104 [==============================] - ETA: 0s - loss: 0.2413 - accuracy:
0.9122
Epoch 5: val_loss improved from 0.45125 to 0.33636, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 84ms/step - loss: 0.2413 -
accuracy: 0.9122 - val_loss: 0.3364 - val_accuracy: 0.8842
Epoch 6/10
104/104 [==============================] - ETA: 0s - loss: 0.1420 - accuracy:
0.9537
Epoch 6: val_loss improved from 0.33636 to 0.27814, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 83ms/step - loss: 0.1420 -
accuracy: 0.9537 - val_loss: 0.2781 - val_accuracy: 0.9151
Epoch 7/10
103/104 [============================>.] - ETA: 0s - loss: 0.0698 - accuracy:
0.9786
Epoch 7: val_loss improved from 0.27814 to 0.23493, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 83ms/step - loss: 0.0695 -
accuracy: 0.9788 - val_loss: 0.2349 - val_accuracy: 0.9112
Epoch 8/10
104/104 [==============================] - ETA: 0s - loss: 0.0360 - accuracy:
0.9942
Epoch 8: val_loss did not improve from 0.23493
104/104 [==============================] - 8s 80ms/step - loss: 0.0360 -
accuracy: 0.9942 - val_loss: 0.2950 - val_accuracy: 0.9151
Epoch 9/10
103/104 [============================>.] - ETA: 0s - loss: 0.0229 - accuracy:
0.9961
Epoch 9: val_loss did not improve from 0.23493
104/104 [==============================] - 8s 80ms/step - loss: 0.0231 -
accuracy: 0.9961 - val_loss: 0.2548 - val_accuracy: 0.9344
Epoch 10/10
103/104 [============================>.] - ETA: 0s - loss: 0.0102 - accuracy:
1.0000
```

```
Epoch 10: val_loss did not improve from 0.23493
104/104 [==============================] - 8s 80ms/step - loss: 0.0101 -
accuracy: 1.0000 - val_loss: 0.2448 - val_accuracy: 0.9421


Evaluating the Test metrics
11/11 [==============================] - 2s 127ms/step - loss: 0.2914 -
accuracy: 0.8951
Test loss for fold 3: 0.29137712717056274
Test accuracy for fold 3: 0.895061731338501
11/11 [==============================] - 1s 133ms/step
Precision for fold 3: 0.8962645225192823
Recall for fold 3: 0.8950617283950617
F1 Score for fold 3: 0.8950502549577888


Epoch 1/10
104/104 [==============================] - ETA: 0s - loss: 1.9542 - accuracy:
0.4614
Epoch 1: val_loss improved from inf to 0.92482, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(

104/104 [==============================] - 10s 88ms/step - loss: 1.9542 -
accuracy: 0.4614 - val_loss: 0.9248 - val_accuracy: 0.6139
Epoch 2/10
104/104 [==============================] - ETA: 0s - loss: 0.7298 - accuracy:
0.7037
Epoch 2: val_loss improved from 0.92482 to 0.74546, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 84ms/step - loss: 0.7298 -
accuracy: 0.7037 - val_loss: 0.7455 - val_accuracy: 0.7027
Epoch 3/10
104/104 [==============================] - ETA: 0s - loss: 0.3575 - accuracy:
0.8639
Epoch 3: val_loss improved from 0.74546 to 0.44737, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 85ms/step - loss: 0.3575 -
accuracy: 0.8639 - val_loss: 0.4474 - val_accuracy: 0.8224
Epoch 4/10
104/104 [==============================] - ETA: 0s - loss: 0.1788 - accuracy:
0.9431
Epoch 4: val_loss did not improve from 0.44737
104/104 [==============================] - 9s 82ms/step - loss: 0.1788 -
```

```
accuracy: 0.9431 - val_loss: 0.5606 - val_accuracy: 0.7838
Epoch 5/10
103/104 [============================>.] - ETA: 0s - loss: 0.1307 - accuracy:
0.9544
Epoch 5: val_loss did not improve from 0.44737
104/104 [==============================] - 8s 81ms/step - loss: 0.1301 -
accuracy: 0.9546 - val_loss: 0.4535 - val_accuracy: 0.8378
Epoch 6/10
104/104 [==============================] - ETA: 0s - loss: 0.0710 - accuracy:
0.9720
Epoch 6: val_loss improved from 0.44737 to 0.22405, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 91ms/step - loss: 0.0710 -
accuracy: 0.9720 - val_loss: 0.2241 - val_accuracy: 0.9305
Epoch 7/10
103/104 [============================>.] - ETA: 0s - loss: 0.0295 - accuracy:
0.9922
Epoch 7: val_loss did not improve from 0.22405
104/104 [==============================] - 8s 80ms/step - loss: 0.0294 -
accuracy: 0.9923 - val_loss: 0.2351 - val_accuracy: 0.9266
Epoch 8/10
103/104 [============================>.] - ETA: 0s - loss: 0.0101 - accuracy:
0.9990
Epoch 8: val_loss improved from 0.22405 to 0.21597, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 88ms/step - loss: 0.0102 -
accuracy: 0.9990 - val_loss: 0.2160 - val_accuracy: 0.9344
Epoch 9/10
103/104 [============================>.] - ETA: 0s - loss: 0.0116 - accuracy:
0.9981
Epoch 9: val_loss did not improve from 0.21597
104/104 [==============================] - 8s 80ms/step - loss: 0.0115 -
accuracy: 0.9981 - val_loss: 0.2353 - val_accuracy: 0.9112
Epoch 10/10
104/104 [==============================] - ETA: 0s - loss: 0.0046 - accuracy:
1.0000
Epoch 10: val_loss did not improve from 0.21597
104/104 [==============================] - 8s 80ms/step - loss: 0.0046 -
accuracy: 1.0000 - val_loss: 0.2213 - val_accuracy: 0.9266


Evaluating the Test metrics
11/11 [==============================] - 2s 127ms/step - loss: 0.2731 -
accuracy: 0.9167
Test loss for fold 4: 0.2730903923511505
Test accuracy for fold 4: 0.9166666865348816
11/11 [==============================] - 1s 134ms/step
Precision for fold 4: 0.9176818682115054
```

```
Recall for fold 4: 0.9166666666666666
F1 Score for fold 4: 0.9168440981220994
```

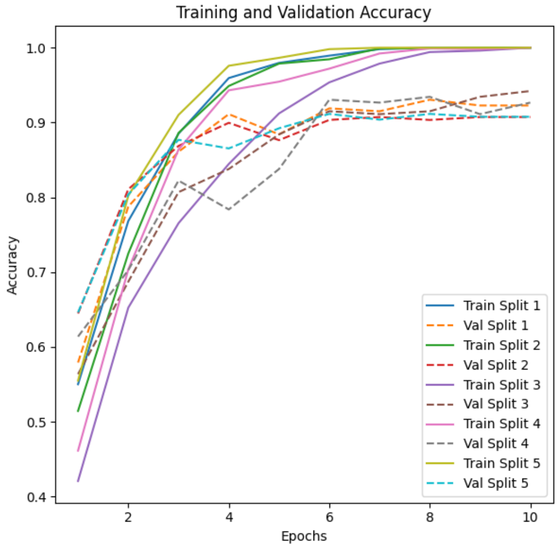
```
Epoch 1/10
103/104 [============================>.] - ETA: 0s - loss: 1.6284 - accuracy:
0.5544
Epoch 1: val_loss improved from inf to 0.84469, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
```

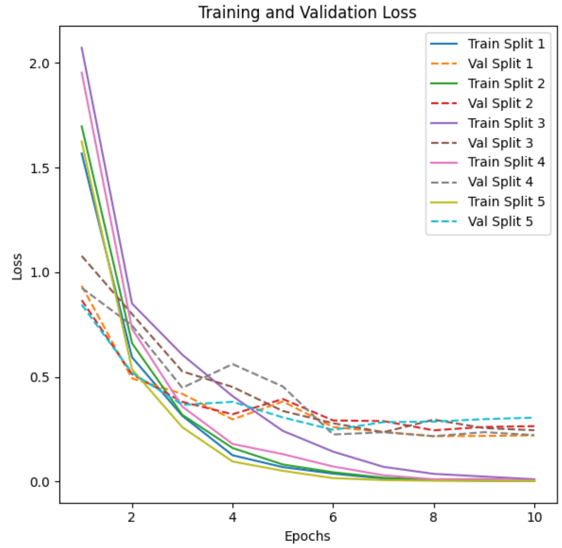
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
104/104 [==============================] - 10s 89ms/step - loss: 1.6251 -
accuracy: 0.5550 - val_loss: 0.8447 - val_accuracy: 0.6462
Epoch 2/10
104/104 [==============================] - ETA: 0s - loss: 0.5358 - accuracy:
0.8021
Epoch 2: val_loss improved from 0.84469 to 0.51912, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 84ms/step - loss: 0.5358 -
accuracy: 0.8021 - val_loss: 0.5191 - val_accuracy: 0.8038
Epoch 3/10
104/104 [==============================] - ETA: 0s - loss: 0.2590 - accuracy:
0.9102
Epoch 3: val_loss improved from 0.51912 to 0.36596, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 86ms/step - loss: 0.2590 -
accuracy: 0.9102 - val_loss: 0.3660 - val_accuracy: 0.8769
Epoch 4/10
103/104 [============================>.] - ETA: 0s - loss: 0.0956 - accuracy:
0.9757
Epoch 4: val_loss did not improve from 0.36596
104/104 [==============================] - 9s 83ms/step - loss: 0.0951 -
accuracy: 0.9759 - val_loss: 0.3802 - val_accuracy: 0.8654
Epoch 5/10
104/104 [==============================] - ETA: 0s - loss: 0.0504 - accuracy:
0.9865
Epoch 5: val_loss improved from 0.36596 to 0.30576, saving model to
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 90ms/step - loss: 0.0504 -
accuracy: 0.9865 - val_loss: 0.3058 - val_accuracy: 0.8923
Epoch 6/10
103/104 [============================>.] - ETA: 0s - loss: 0.0156 - accuracy:
0.9981
Epoch 6: val_loss improved from 0.30576 to 0.24608, saving model to
```

```
/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_one.h5
104/104 [==============================] - 9s 83ms/step - loss: 0.0155 -
accuracy: 0.9981 - val_loss: 0.2461 - val_accuracy: 0.9115
Epoch 7/10
103/104 [============================>.] - ETA: 0s - loss: 0.0061 - accuracy:
1.0000
Epoch 7: val_loss did not improve from 0.24608
104/104 [==============================] - 8s 80ms/step - loss: 0.0061 -
accuracy: 1.0000 - val_loss: 0.2833 - val_accuracy: 0.9038
Epoch 8/10
103/104 [============================>.] - ETA: 0s - loss: 0.0030 - accuracy:
1.0000
Epoch 8: val_loss did not improve from 0.24608
104/104 [==============================] - 8s 80ms/step - loss: 0.0030 -
accuracy: 1.0000 - val_loss: 0.2858 - val_accuracy: 0.9115
Epoch 9/10
103/104 [============================>.] - ETA: 0s - loss: 0.0023 - accuracy:
1.0000
Epoch 9: val_loss did not improve from 0.24608
104/104 [==============================] - 8s 80ms/step - loss: 0.0023 -
accuracy: 1.0000 - val_loss: 0.2973 - val_accuracy: 0.9077
Epoch 10/10
103/104 [============================>.] - ETA: 0s - loss: 0.0018 - accuracy:
1.0000
Epoch 10: val_loss did not improve from 0.24608
104/104 [==============================] - 8s 81ms/step - loss: 0.0018 -
accuracy: 1.0000 - val_loss: 0.3048 - val_accuracy: 0.9077


Evaluating the Test metrics
11/11 [==============================] - 3s 273ms/step - loss: 0.1921 -
accuracy: 0.9195
Test loss for fold 5: 0.192056804895401
Test accuracy for fold 5: 0.9195046424865723
11/11 [==============================] - 2s 135ms/step
Precision for fold 5: 0.9195274331859169
Recall for fold 5: 0.9195046439628483
F1 Score for fold 5: 0.9192853929696035


Standard deviation of accuracy across folds: 0.01569900346178792
Average Accuracy across folds: 0.9073577284812927
Average Precision across folds: 0.9085617035997151
Average Recall across folds: 0.9073577189160265
Average F1 Score across folds: 0.9075063202113511
```

[15]:
```python
import matplotlib.pyplot as plt

# Determine the maximum length among all lists
max_length = max(len(train_acc) for train_acc in train_acc_list)

# Pad the shorter lists with zeros to match the maximum length
for i in range(n_splits):
    train_acc_list[i] += [0] * (max_length - len(train_acc_list[i]))
    val_acc_list[i] += [0] * (max_length - len(val_acc_list[i]))
    train_loss_list[i] += [0] * (max_length - len(train_loss_list[i]))
    val_loss_list[i] += [0] * (max_length - len(val_loss_list[i]))

# Plotting
epochs = range(1, max_length + 1)

plt.figure(figsize=(12, 6))

# Plotting Training and Validation Accuracy
plt.subplot(1, 2, 1)
for i in range(n_splits):
    plt.plot(epochs, train_acc_list[i], label=f'Train Split {i+1}')
    plt.plot(epochs, val_acc_list[i], label=f'Val Split {i+1}', linestyle='--')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting Training and Validation Loss
plt.subplot(1, 2, 2)
for i in range(n_splits):
    plt.plot(epochs, train_loss_list[i], label=f'Train Split {i+1}')
    plt.plot(epochs, val_loss_list[i], label=f'Val Split {i+1}', linestyle='--')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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





[11]:
```
def build_model():
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,␣
↪224, 3))

    # Freeze first 10 convolutional layers
    for layer in base_model.layers[:15]:
        layer.trainable = False

    x = base_model.output

    # Adding Conv2D + ReLU Layers
    x = Conv2D(512, (3, 3), padding='same')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(512, (3, 3), padding='same', activation='relu')(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # Adding Max Pooling
    x = MaxPooling2D((2, 2), strides=(2, 2))(x)

    # Add GlobalAveragePooling2D layer
    x = GlobalAveragePooling2D()(x)

    # Replacing the last three layers with new fully connected layers
```

```
    x = Dense(256)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dropout(0.5)(x)

    x = Dense(256)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Dropout(0.5)(x)

    predictions = Dense(len(sheep_breed_list), activation='softmax')(x)  # Add␣
↪output layer

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

    # Custom learning rates
    lr_mult = {}

    # Set learning rate to 0 for the first 10 layers
    for layer in model_normalizedd.layers[:15]:
        lr_mult[layer.name + '/kernel:0'] = 0.0
        lr_mult[layer.name + '/bias:0'] = 0.0

    # Set learning rate to 0.0001 for the next Conv2D layers and the following␣
↪layers
    for layer in model_normalizedd.layers[15:30]:
        if isinstance(layer, Conv2D):
            lr_mult[layer.name + '/kernel:0'] = 0.0001
            lr_mult[layer.name + '/bias:0'] = 0.0001

    # Set learning rate to 10 for the last two dense layers
    for layer in model_normalizedd.layers[30:]:
        lr_mult[layer.name + '/kernel:0'] = 1
        lr_mult[layer.name + '/bias:0'] = 1

    # Use SGD optimizer with initial learning rate of 1e-4
    optimizer = SGD(learning_rate=1e-4, momentum=0.9)

    # Compile the model
    model_normalizedd.compile(optimizer=optimizer,␣
↪loss='categorical_crossentropy', metrics=['accuracy'], loss_weights=[1.] +␣
↪[10.]*2)

    return model_normalizedd

accuracy_values = []
precision_values = []
```

```python
recall_values = []
f1_values = []

train_acc_list = []
val_acc_list = []
train_loss_list = []
val_loss_list = []

# KFold Cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=777)
fold_var = 1

for train_index, test_index in kf.split(img_data):
    X_train, X_test = img_data[train_index], img_data[test_index]
    y_train, y_test = labels_categorical[train_index], labels_categorical[test_index]

    model_normalizedd = build_model()

        # Define the ImageDataGenerator for data augmentation
    datagen = ImageDataGenerator(
        width_shift_range=0.1,  # translate horizontally by 10% of total width
        height_shift_range=0.1,  # translate vertically by 10% of total height
        fill_mode='nearest'  # strategy for filling in newly created pixels
    )

    # Generate augmented images batches during training
    train_datagen = datagen.flow(X_train, y_train, batch_size=10)

    #early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1, mode='min')
    save_path='/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/sheepFace_two.keras'
    checkpoint = ModelCheckpoint(save_path,monitor='val_loss',save_best_only=True,verbose=0)
    history = model_normalizedd.fit(train_datagen, epochs=20, validation_data=(X_test,y_test), verbose=1, callbacks=[checkpoint])

    print("\n")
    print(f'Evaluating the Test metrics')
    model_normalizedd=load_model(save_path)
    # Evaluate the model
    scores = model_normalizedd.evaluate(X_test, y_test, verbose=1)
    print(f'Test loss for fold {fold_var}: {scores[0]}')
    print(f'Test accuracy for fold {fold_var}: {scores[1]}')
```

```python
    accuracy_values.append(scores[1])  # Appending accuracy to the list

    # Get predictions from the model
    y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels back␣
 ↪to categorical labels
    y_pred = np.argmax(model_normalizedd.predict(X_test), axis=1)  # Get␣
 ↪predictions from the model

    # Calculate and append evaluation metrics for this fold
    precision = precision_score(y_true, y_pred, average='weighted',␣
 ↪zero_division=1)
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    precision_values.append(precision)
    recall_values.append(recall)
    f1_values.append(f1)

    # Print evaluation metrics for this fold
    print(f'Precision for fold {fold_var}: {precision}')
    print(f'Recall for fold {fold_var}: {recall}')
    print(f'F1 Score for fold {fold_var}: {f1}')

    train_acc_list.append(history.history['accuracy'])
    val_acc_list.append(history.history['val_accuracy'])
    train_loss_list.append(history.history['loss'])
    val_loss_list.append(history.history['val_loss'])
    print("\n")

    # Increment the fold number
    fold_var += 1

# Calculate standard deviation of accuracy across folds
accuracy_std_dev = np.std(accuracy_values)
print(f'Standard deviation of accuracy across folds: {accuracy_std_dev}')

# Calculate average of accuracy,precision, recall, and F1 across folds
avg_accuracy = np.mean(accuracy_values)
avg_precision = np.mean(precision_values)
avg_recall = np.mean(recall_values)
avg_f1 = np.mean(f1_values)

print(f'Average Accuracy across folds: {avg_accuracy}')
print(f'Average Precision across folds: {avg_precision}')
print(f'Average Recall across folds: {avg_recall}')
print(f'Average F1 Score across folds: {avg_f1}')
```

```
Epoch 1/20
130/130 [==============================] - 24s 146ms/step - loss: 1.7989 -
accuracy: 0.3220 - val_loss: 1.2810 - val_accuracy: 0.3827
Epoch 2/20
130/130 [==============================] - 16s 120ms/step - loss: 1.2778 -
accuracy: 0.4587 - val_loss: 1.0687 - val_accuracy: 0.5093
Epoch 3/20
130/130 [==============================] - 15s 118ms/step - loss: 1.0691 -
accuracy: 0.5761 - val_loss: 0.7457 - val_accuracy: 0.7377
Epoch 4/20
130/130 [==============================] - 16s 119ms/step - loss: 0.8291 -
accuracy: 0.6587 - val_loss: 0.5276 - val_accuracy: 0.8210
Epoch 5/20
130/130 [==============================] - 15s 114ms/step - loss: 0.6811 -
accuracy: 0.7421 - val_loss: 0.5913 - val_accuracy: 0.7840
Epoch 6/20
130/130 [==============================] - 15s 118ms/step - loss: 0.5906 -
accuracy: 0.7838 - val_loss: 0.3870 - val_accuracy: 0.8642
Epoch 7/20
130/130 [==============================] - 16s 119ms/step - loss: 0.5173 -
accuracy: 0.8139 - val_loss: 0.3757 - val_accuracy: 0.8673
Epoch 8/20
130/130 [==============================] - 15s 115ms/step - loss: 0.4585 -
accuracy: 0.8363 - val_loss: 0.5545 - val_accuracy: 0.8056
Epoch 9/20
130/130 [==============================] - 15s 118ms/step - loss: 0.4209 -
accuracy: 0.8471 - val_loss: 0.2577 - val_accuracy: 0.9259
Epoch 10/20
130/130 [==============================] - 15s 114ms/step - loss: 0.4046 -
accuracy: 0.8602 - val_loss: 0.4228 - val_accuracy: 0.8673
Epoch 11/20
130/130 [==============================] - 14s 111ms/step - loss: 0.3067 -
accuracy: 0.8927 - val_loss: 0.3194 - val_accuracy: 0.9012
Epoch 12/20
130/130 [==============================] - 15s 118ms/step - loss: 0.2787 -
accuracy: 0.9097 - val_loss: 0.1761 - val_accuracy: 0.9537
Epoch 13/20
130/130 [==============================] - 15s 113ms/step - loss: 0.2792 -
accuracy: 0.9120 - val_loss: 0.1818 - val_accuracy: 0.9321
Epoch 14/20
130/130 [==============================] - 14s 110ms/step - loss: 0.2458 -
accuracy: 0.9158 - val_loss: 0.3486 - val_accuracy: 0.8827
Epoch 15/20
130/130 [==============================] - 15s 111ms/step - loss: 0.2472 -
accuracy: 0.9266 - val_loss: 0.2110 - val_accuracy: 0.9383
Epoch 16/20
130/130 [==============================] - 15s 114ms/step - loss: 0.2637 -
accuracy: 0.9189 - val_loss: 0.2293 - val_accuracy: 0.9228
```

```
Epoch 17/20
130/130 [==============================] - 15s 112ms/step - loss: 0.2059 -
accuracy: 0.9375 - val_loss: 0.1787 - val_accuracy: 0.9506
Epoch 18/20
130/130 [==============================] - 14s 110ms/step - loss: 0.1843 -
accuracy: 0.9483 - val_loss: 0.1783 - val_accuracy: 0.9568
Epoch 19/20
130/130 [==============================] - 15s 111ms/step - loss: 0.1959 -
accuracy: 0.9405 - val_loss: 0.2128 - val_accuracy: 0.9321
Epoch 20/20
130/130 [==============================] - 15s 117ms/step - loss: 0.1712 -
accuracy: 0.9467 - val_loss: 0.1443 - val_accuracy: 0.9475


Evaluating the Test metrics
11/11 [==============================] - 2s 132ms/step - loss: 0.1443 -
accuracy: 0.9475
Test loss for fold 1: 0.1442643404006958
Test accuracy for fold 1: 0.9475308656692505
11/11 [==============================] - 2s 141ms/step
Precision for fold 1: 0.9485916873803986
Recall for fold 1: 0.9475308641975309
F1 Score for fold 1: 0.9472657986768309


Epoch 1/20
130/130 [==============================] - 18s 121ms/step - loss: 1.6908 -
accuracy: 0.3452 - val_loss: 1.1815 - val_accuracy: 0.5185
Epoch 2/20
130/130 [==============================] - 15s 116ms/step - loss: 1.2419 -
accuracy: 0.4996 - val_loss: 0.8404 - val_accuracy: 0.6451
Epoch 3/20
130/130 [==============================] - 15s 118ms/step - loss: 0.9764 -
accuracy: 0.6224 - val_loss: 0.6378 - val_accuracy: 0.7870
Epoch 4/20
130/130 [==============================] - 15s 117ms/step - loss: 0.7801 -
accuracy: 0.6996 - val_loss: 0.4306 - val_accuracy: 0.8457
Epoch 5/20
130/130 [==============================] - 15s 117ms/step - loss: 0.6128 -
accuracy: 0.7568 - val_loss: 0.3875 - val_accuracy: 0.8580
Epoch 6/20
130/130 [==============================] - 16s 119ms/step - loss: 0.5599 -
accuracy: 0.7915 - val_loss: 0.2982 - val_accuracy: 0.8981
Epoch 7/20
130/130 [==============================] - 16s 120ms/step - loss: 0.4627 -
accuracy: 0.8309 - val_loss: 0.2729 - val_accuracy: 0.9167
Epoch 8/20
130/130 [==============================] - 16s 120ms/step - loss: 0.4165 -
```

```
accuracy: 0.8602 - val_loss: 0.2565 - val_accuracy: 0.9198
Epoch 9/20
130/130 [==============================] - 15s 113ms/step - loss: 0.4071 -
accuracy: 0.8649 - val_loss: 0.4176 - val_accuracy: 0.8457
Epoch 10/20
130/130 [==============================] - 15s 118ms/step - loss: 0.3641 -
accuracy: 0.8764 - val_loss: 0.2443 - val_accuracy: 0.9105
Epoch 11/20
130/130 [==============================] - 16s 120ms/step - loss: 0.3183 -
accuracy: 0.8919 - val_loss: 0.1918 - val_accuracy: 0.9444
Epoch 12/20
130/130 [==============================] - 15s 117ms/step - loss: 0.3177 -
accuracy: 0.8896 - val_loss: 0.1871 - val_accuracy: 0.9352
Epoch 13/20
130/130 [==============================] - 15s 116ms/step - loss: 0.2619 -
accuracy: 0.9243 - val_loss: 0.1730 - val_accuracy: 0.9568
Epoch 14/20
130/130 [==============================] - 15s 112ms/step - loss: 0.2515 -
accuracy: 0.9166 - val_loss: 0.2513 - val_accuracy: 0.9105
Epoch 15/20
130/130 [==============================] - 15s 113ms/step - loss: 0.2557 -
accuracy: 0.9166 - val_loss: 0.1984 - val_accuracy: 0.9475
Epoch 16/20
130/130 [==============================] - 14s 111ms/step - loss: 0.2162 -
accuracy: 0.9297 - val_loss: 0.2119 - val_accuracy: 0.9352
Epoch 17/20
130/130 [==============================] - 15s 115ms/step - loss: 0.1816 -
accuracy: 0.9459 - val_loss: 0.1426 - val_accuracy: 0.9506
Epoch 18/20
130/130 [==============================] - 15s 118ms/step - loss: 0.2266 -
accuracy: 0.9282 - val_loss: 0.1416 - val_accuracy: 0.9537
Epoch 19/20
130/130 [==============================] - 15s 118ms/step - loss: 0.1728 -
accuracy: 0.9483 - val_loss: 0.1248 - val_accuracy: 0.9537
Epoch 20/20
130/130 [==============================] - 15s 113ms/step - loss: 0.2038 -
accuracy: 0.9413 - val_loss: 0.2455 - val_accuracy: 0.9290


Evaluating the Test metrics
11/11 [==============================] - 2s 131ms/step - loss: 0.1248 -
accuracy: 0.9537
Test loss for fold 2: 0.12477420270442963
Test accuracy for fold 2: 0.9537037014961243
11/11 [==============================] - 2s 141ms/step
Precision for fold 2: 0.9537671286921809
Recall for fold 2: 0.9537037037037037
F1 Score for fold 2: 0.9536984533043342
```

```
Epoch 1/20
130/130 [==============================] - 18s 123ms/step - loss: 1.7451 -
accuracy: 0.3328 - val_loss: 1.4398 - val_accuracy: 0.3765
Epoch 2/20
130/130 [==============================] - 15s 118ms/step - loss: 1.3072 -
accuracy: 0.4819 - val_loss: 0.8803 - val_accuracy: 0.5957
Epoch 3/20
130/130 [==============================] - 15s 118ms/step - loss: 1.0268 -
accuracy: 0.5892 - val_loss: 0.6872 - val_accuracy: 0.7315
Epoch 4/20
130/130 [==============================] - 16s 120ms/step - loss: 0.8356 -
accuracy: 0.6811 - val_loss: 0.4653 - val_accuracy: 0.8673
Epoch 5/20
130/130 [==============================] - 16s 120ms/step - loss: 0.6616 -
accuracy: 0.7313 - val_loss: 0.3635 - val_accuracy: 0.8765
Epoch 6/20
130/130 [==============================] - 15s 118ms/step - loss: 0.5258 -
accuracy: 0.8023 - val_loss: 0.3396 - val_accuracy: 0.8642
Epoch 7/20
130/130 [==============================] - 15s 112ms/step - loss: 0.5094 -
accuracy: 0.8170 - val_loss: 0.6713 - val_accuracy: 0.7623
Epoch 8/20
130/130 [==============================] - 15s 112ms/step - loss: 0.4224 -
accuracy: 0.8525 - val_loss: 0.3464 - val_accuracy: 0.8611
Epoch 9/20
130/130 [==============================] - 15s 117ms/step - loss: 0.3778 -
accuracy: 0.8595 - val_loss: 0.2020 - val_accuracy: 0.9228
Epoch 10/20
130/130 [==============================] - 15s 113ms/step - loss: 0.3311 -
accuracy: 0.8849 - val_loss: 0.2830 - val_accuracy: 0.9012
Epoch 11/20
130/130 [==============================] - 15s 111ms/step - loss: 0.2922 -
accuracy: 0.9012 - val_loss: 0.2803 - val_accuracy: 0.8951
Epoch 12/20
130/130 [==============================] - 15s 111ms/step - loss: 0.3015 -
accuracy: 0.9027 - val_loss: 0.2312 - val_accuracy: 0.9074
Epoch 13/20
130/130 [==============================] - 15s 114ms/step - loss: 0.2444 -
accuracy: 0.9220 - val_loss: 0.2469 - val_accuracy: 0.9105
Epoch 14/20
130/130 [==============================] - 15s 118ms/step - loss: 0.2490 -
accuracy: 0.9166 - val_loss: 0.1782 - val_accuracy: 0.9383
Epoch 15/20
130/130 [==============================] - 15s 114ms/step - loss: 0.2286 -
accuracy: 0.9259 - val_loss: 0.2394 - val_accuracy: 0.9074
Epoch 16/20
```

```
130/130 [==============================] - 14s 110ms/step - loss: 0.2167 -
accuracy: 0.9313 - val_loss: 0.3007 - val_accuracy: 0.8951
Epoch 17/20
130/130 [==============================] - 15s 112ms/step - loss: 0.1966 -
accuracy: 0.9467 - val_loss: 0.2711 - val_accuracy: 0.9074
Epoch 18/20
130/130 [==============================] - 15s 117ms/step - loss: 0.2054 -
accuracy: 0.9367 - val_loss: 0.1777 - val_accuracy: 0.9475
Epoch 19/20
130/130 [==============================] - 15s 114ms/step - loss: 0.2099 -
accuracy: 0.9359 - val_loss: 0.3097 - val_accuracy: 0.8920
Epoch 20/20
130/130 [==============================] - 14s 110ms/step - loss: 0.1629 -
accuracy: 0.9552 - val_loss: 0.1895 - val_accuracy: 0.9444


Evaluating the Test metrics
11/11 [==============================] - 2s 132ms/step - loss: 0.1777 -
accuracy: 0.9475
Test loss for fold 3: 0.1777118742465973
Test accuracy for fold 3: 0.9475308656692505
11/11 [==============================] - 2s 141ms/step
Precision for fold 3: 0.951475860330027
Recall for fold 3: 0.9475308641975309
F1 Score for fold 3: 0.947713761208653


Epoch 1/20
130/130 [==============================] - 19s 121ms/step - loss: 1.5454 -
accuracy: 0.3753 - val_loss: 1.7065 - val_accuracy: 0.3519
Epoch 2/20
130/130 [==============================] - 16s 119ms/step - loss: 1.2504 -
accuracy: 0.4880 - val_loss: 1.0285 - val_accuracy: 0.5648
Epoch 3/20
130/130 [==============================] - 15s 117ms/step - loss: 0.9940 -
accuracy: 0.5931 - val_loss: 0.7108 - val_accuracy: 0.6944
Epoch 4/20
130/130 [==============================] - 15s 118ms/step - loss: 0.8390 -
accuracy: 0.6625 - val_loss: 0.5412 - val_accuracy: 0.8056
Epoch 5/20
130/130 [==============================] - 15s 118ms/step - loss: 0.6787 -
accuracy: 0.7282 - val_loss: 0.4743 - val_accuracy: 0.8241
Epoch 6/20
130/130 [==============================] - 15s 117ms/step - loss: 0.5811 -
accuracy: 0.7830 - val_loss: 0.2881 - val_accuracy: 0.9074
Epoch 7/20
130/130 [==============================] - 15s 111ms/step - loss: 0.5089 -
accuracy: 0.8255 - val_loss: 0.3594 - val_accuracy: 0.8796
```

```
Epoch 8/20
130/130 [==============================] - 15s 119ms/step - loss: 0.4549 -
accuracy: 0.8378 - val_loss: 0.2355 - val_accuracy: 0.9167
Epoch 9/20
130/130 [==============================] - 16s 119ms/step - loss: 0.4253 -
accuracy: 0.8463 - val_loss: 0.2027 - val_accuracy: 0.9475
Epoch 10/20
130/130 [==============================] - 15s 117ms/step - loss: 0.3932 -
accuracy: 0.8579 - val_loss: 0.1719 - val_accuracy: 0.9537
Epoch 11/20
130/130 [==============================] - 14s 110ms/step - loss: 0.3530 -
accuracy: 0.8764 - val_loss: 0.2438 - val_accuracy: 0.9167
Epoch 12/20
130/130 [==============================] - 15s 112ms/step - loss: 0.3104 -
accuracy: 0.9035 - val_loss: 0.1917 - val_accuracy: 0.9352
Epoch 13/20
130/130 [==============================] - 15s 115ms/step - loss: 0.2967 -
accuracy: 0.9027 - val_loss: 0.1484 - val_accuracy: 0.9568
Epoch 14/20
130/130 [==============================] - 15s 112ms/step - loss: 0.2738 -
accuracy: 0.9135 - val_loss: 0.1562 - val_accuracy: 0.9475
Epoch 15/20
130/130 [==============================] - 15s 117ms/step - loss: 0.2885 -
accuracy: 0.9035 - val_loss: 0.1459 - val_accuracy: 0.9537
Epoch 16/20
130/130 [==============================] - 15s 113ms/step - loss: 0.2519 -
accuracy: 0.9181 - val_loss: 0.1509 - val_accuracy: 0.9506
Epoch 17/20
130/130 [==============================] - 15s 117ms/step - loss: 0.2636 -
accuracy: 0.9143 - val_loss: 0.1104 - val_accuracy: 0.9691
Epoch 18/20
130/130 [==============================] - 15s 117ms/step - loss: 0.1862 -
accuracy: 0.9398 - val_loss: 0.1086 - val_accuracy: 0.9660
Epoch 19/20
130/130 [==============================] - 15s 118ms/step - loss: 0.1978 -
accuracy: 0.9429 - val_loss: 0.0909 - val_accuracy: 0.9691
Epoch 20/20
130/130 [==============================] - 15s 114ms/step - loss: 0.1954 -
accuracy: 0.9382 - val_loss: 0.1214 - val_accuracy: 0.9506


Evaluating the Test metrics
11/11 [==============================] - 2s 130ms/step - loss: 0.0909 -
accuracy: 0.9691
Test loss for fold 4: 0.09088943898677826
Test accuracy for fold 4: 0.9691358208656311
11/11 [==============================] - 2s 139ms/step
Precision for fold 4: 0.9700717250918054
```

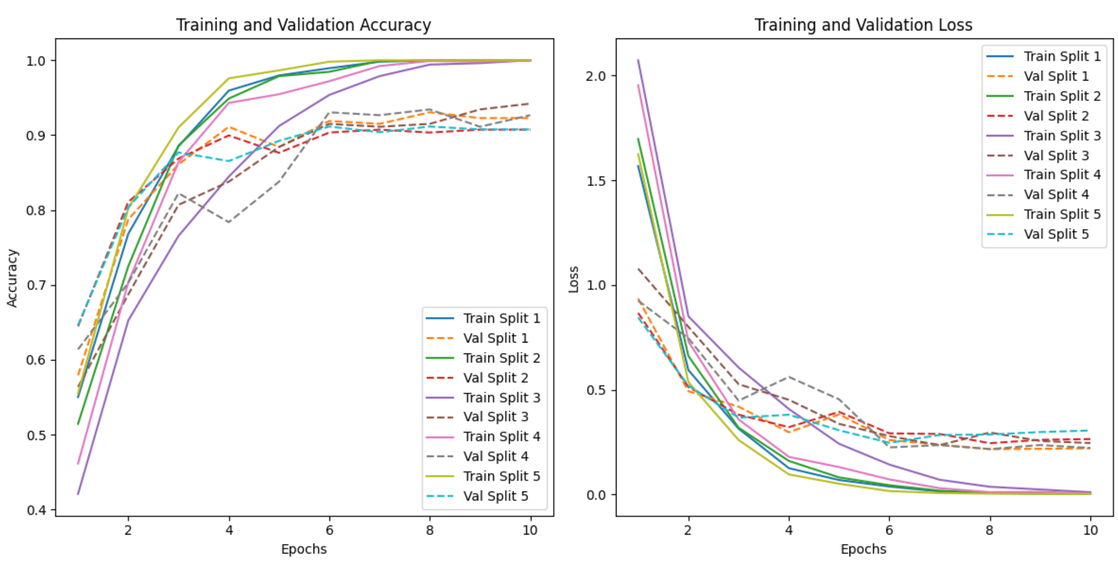
```
Recall for fold 4: 0.9691358024691358
F1 Score for fold 4: 0.9692712878650681


Epoch 1/20
130/130 [==============================] - 19s 126ms/step - loss: 1.6624 -
accuracy: 0.3488 - val_loss: 1.2553 - val_accuracy: 0.4056
Epoch 2/20
130/130 [==============================] - 16s 121ms/step - loss: 1.2464 -
accuracy: 0.4985 - val_loss: 0.8642 - val_accuracy: 0.6625
Epoch 3/20
130/130 [==============================] - 15s 118ms/step - loss: 0.9862 -
accuracy: 0.6011 - val_loss: 0.5330 - val_accuracy: 0.8142
Epoch 4/20
130/130 [==============================] - 15s 117ms/step - loss: 0.8055 -
accuracy: 0.6798 - val_loss: 0.3830 - val_accuracy: 0.8576
Epoch 5/20
130/130 [==============================] - 15s 118ms/step - loss: 0.6612 -
accuracy: 0.7415 - val_loss: 0.3250 - val_accuracy: 0.8762
Epoch 6/20
130/130 [==============================] - 16s 119ms/step - loss: 0.5472 -
accuracy: 0.8009 - val_loss: 0.2941 - val_accuracy: 0.8885
Epoch 7/20
130/130 [==============================] - 15s 118ms/step - loss: 0.4592 -
accuracy: 0.8387 - val_loss: 0.2613 - val_accuracy: 0.8885
Epoch 8/20
130/130 [==============================] - 15s 117ms/step - loss: 0.4070 -
accuracy: 0.8596 - val_loss: 0.1884 - val_accuracy: 0.9350
Epoch 9/20
130/130 [==============================] - 14s 110ms/step - loss: 0.3688 -
accuracy: 0.8719 - val_loss: 0.2418 - val_accuracy: 0.9164
Epoch 10/20
130/130 [==============================] - 15s 119ms/step - loss: 0.3401 -
accuracy: 0.8935 - val_loss: 0.1851 - val_accuracy: 0.9443
Epoch 11/20
130/130 [==============================] - 15s 119ms/step - loss: 0.2952 -
accuracy: 0.8927 - val_loss: 0.1689 - val_accuracy: 0.9412
Epoch 12/20
130/130 [==============================] - 15s 114ms/step - loss: 0.2971 -
accuracy: 0.9020 - val_loss: 0.1788 - val_accuracy: 0.9350
Epoch 13/20
130/130 [==============================] - 15s 111ms/step - loss: 0.2646 -
accuracy: 0.9174 - val_loss: 0.2108 - val_accuracy: 0.9195
Epoch 14/20
130/130 [==============================] - 15s 119ms/step - loss: 0.2432 -
accuracy: 0.9267 - val_loss: 0.1453 - val_accuracy: 0.9505
Epoch 15/20
130/130 [==============================] - 15s 113ms/step - loss: 0.2126 -
```

```
accuracy: 0.9313 - val_loss: 0.2594 - val_accuracy: 0.9102
Epoch 16/20
130/130 [==============================] - 15s 112ms/step - loss: 0.2189 -
accuracy: 0.9306 - val_loss: 0.2209 - val_accuracy: 0.9288
Epoch 17/20
130/130 [==============================] - 15s 112ms/step - loss: 0.1967 -
accuracy: 0.9398 - val_loss: 0.1937 - val_accuracy: 0.9288
Epoch 18/20
130/130 [==============================] - 15s 113ms/step - loss: 0.1958 -
accuracy: 0.9406 - val_loss: 0.1550 - val_accuracy: 0.9659
Epoch 19/20
130/130 [==============================] - 15s 116ms/step - loss: 0.1739 -
accuracy: 0.9560 - val_loss: 0.1395 - val_accuracy: 0.9505
Epoch 20/20
130/130 [==============================] - 15s 113ms/step - loss: 0.1839 -
accuracy: 0.9421 - val_loss: 0.2541 - val_accuracy: 0.9071


Evaluating the Test metrics
11/11 [==============================] - 2s 131ms/step - loss: 0.1395 -
accuracy: 0.9505
Test loss for fold 5: 0.1395476907491684
Test accuracy for fold 5: 0.9504643678665161
11/11 [==============================] - 2s 141ms/step
Precision for fold 5: 0.9515184076329587
Recall for fold 5: 0.9504643962848297
F1 Score for fold 5: 0.9502868449010309


Standard deviation of accuracy across folds: 0.0080603563205253
Average Accuracy across folds: 0.9536731243133545
Average Precision across folds: 0.9550849618254741
Average Recall across folds: 0.9536731261705462
Average F1 Score across folds: 0.9536472291911835
```

[16]:

```
import matplotlib.pyplot as plt

# Determine the maximum length among all lists
max_length = max(len(train_acc) for train_acc in train_acc_list)

# Pad the shorter lists with zeros to match the maximum length
for i in range(n_splits):
    train_acc_list[i] += [0] * (max_length - len(train_acc_list[i]))
    val_acc_list[i] += [0] * (max_length - len(val_acc_list[i]))
    train_loss_list[i] += [0] * (max_length - len(train_loss_list[i]))
    val_loss_list[i] += [0] * (max_length - len(val_loss_list[i]))
```

```python
# Plotting
epochs = range(1, max_length + 1)

plt.figure(figsize=(12, 6))

# Plotting Training and Validation Accuracy
plt.subplot(1, 2, 1)
for i in range(n_splits):
    plt.plot(epochs, train_acc_list[i], label=f'Train Split {i+1}')
    plt.plot(epochs, val_acc_list[i], label=f'Val Split {i+1}', linestyle='--')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting Training and Validation Loss
plt.subplot(1, 2, 2)
for i in range(n_splits):
    plt.plot(epochs, train_loss_list[i], label=f'Train Split {i+1}')
    plt.plot(epochs, val_loss_list[i], label=f'Val Split {i+1}', linestyle='--')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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



[8]:
```
def build_model():
    base_model = VGG19(weights='imagenet', include_top=False, input_shape=(224,␣
↪224, 3))

    # Freeze first 10 convolutional layers
    for layer in base_model.layers[:17]:
        layer.trainable = False

    x = base_model.output

    # Flatten to prepare for the fully connected layers
    x = Flatten()(x)

    # Fully connected + ReLU
    x = Dense(512, activation='relu')(x)
    x = Dense(1024, activation='relu')(x)

    # Adding output layer
    predictions = Dense(len(sheep_breed_list), activation='softmax')(x)

    # Create the final model
    model_VGG19 = Model(inputs=base_model.input, outputs=predictions)

    # Custom learning rates
    lr_mult = {}

    # Set learning rate to 0 for the first 10 layers
    for layer in model_VGG19 .layers[:17]:
        lr_mult[layer.name + '/kernel:0'] = 0.0
        lr_mult[layer.name + '/bias:0'] = 0.0

    # Set learning rate to 0.0001 for the next Conv2D layers and the following␣
↪layers
    for layer in model_VGG19 .layers[17:23]:
        if isinstance(layer, Conv2D):
            lr_mult[layer.name + '/kernel:0'] = 0.001
            lr_mult[layer.name + '/bias:0'] = 0.001

    # Set learning rate to 10 for the last two dense layers
    for layer in model_VGG19 .layers[23:]:
        lr_mult[layer.name + '/kernel:0'] = 0.001
        lr_mult[layer.name + '/bias:0'] = 0.001

    # Use SGD optimizer with initial learning rate of 1e-4
    optimizer = SGD(learning_rate=1e-4,momentum=0.9)

    # Compile the model
```

```
    model_VGG19.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'], loss_weights=[1.] + [10.]*2)

    return model_VGG19

accuracy_values = []
precision_values = []
recall_values = []
f1_values = []

train_acc_list = []
val_acc_list = []
train_loss_list = []
val_loss_list = []

# KFold Cross-validation
n_splits = 5
kf = KFold(n_splits=n_splits, shuffle=True, random_state=777)
fold_var = 1

for train_index, test_index in kf.split(img_data):
    X_train, X_test = img_data[train_index], img_data[test_index]
    y_train, y_test = labels_categorical[train_index],␣
↪labels_categorical[test_index]

    model_VGG19 = build_model()

    #early_stop = EarlyStopping(monitor='val_loss', patience=3, verbose=1,␣
↪mode='min')
    save_path='/content/drive/MyDrive/Deep_learning_projects/SheepFull_Data/
↪SheepFace_three.keras'
    checkpoint =␣
↪ModelCheckpoint(save_path,monitor='val_loss',save_best_only=True,verbose=0)
    history = model_VGG19.fit(X_train, y_train, batch_size=10, epochs=10,␣
↪verbose=1, validation_split=0.2, callbacks=[checkpoint])

    print("\n")
    print(f'Evaluating the Test metrics')
    model_VGG19=load_model(save_path)
    # Evaluate the model
    scores = model_VGG19.evaluate(X_test, y_test, verbose=1)
    print(f'Test loss for fold {fold_var}: {scores[0]}')
    print(f'Test accuracy for fold {fold_var}: {scores[1]}')

    accuracy_values.append(scores[1])  # Appending accuracy to the list
```

```
    # Get predictions from the model
    y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels back␣
 ↪to categorical labels
    y_pred = np.argmax(model_VGG19.predict(X_test), axis=1)  # Get predictions␣
 ↪from the model

    # Calculate and append evaluation metrics for this fold
    precision = precision_score(y_true, y_pred, average='weighted',␣
 ↪zero_division=1)
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    precision_values.append(precision)
    recall_values.append(recall)
    f1_values.append(f1)

    # Print evaluation metrics for this fold
    print(f'Precision for fold {fold_var}: {precision}')
    print(f'Recall for fold {fold_var}: {recall}')
    print(f'F1 Score for fold {fold_var}: {f1}')
    print("\n")

    train_acc_list.append(history.history['accuracy'])
    val_acc_list.append(history.history['val_accuracy'])
    train_loss_list.append(history.history['loss'])
    val_loss_list.append(history.history['val_loss'])

    # Increment the fold number
    fold_var += 1

# Calculate standard deviation of accuracy across folds
accuracy_std_dev = np.std(accuracy_values)
print(f'Standard deviation of accuracy across folds: {accuracy_std_dev}')

# Calculate average of accuracy,precision, recall, and F1 across folds
avg_accuracy = np.mean(accuracy_values)
avg_precision = np.mean(precision_values)
avg_recall = np.mean(recall_values)
avg_f1 = np.mean(f1_values)

print(f'Average Accuracy across folds: {avg_accuracy}')
print(f'Average Precision across folds: {avg_precision}')
print(f'Average Recall across folds: {avg_recall}')
print(f'Average F1 Score across folds: {avg_f1}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
```

```
80134624/80134624 [==============================] - 4s 0us/step
Epoch 1/10
104/104 [==============================] - 22s 154ms/step - loss: 1.6248 -
accuracy: 0.5830 - val_loss: 0.6109 - val_accuracy: 0.7683
Epoch 2/10
104/104 [==============================] - 11s 104ms/step - loss: 0.4059 -
accuracy: 0.8542 - val_loss: 0.3482 - val_accuracy: 0.8571
Epoch 3/10
104/104 [==============================] - 10s 95ms/step - loss: 0.1213 -
accuracy: 0.9614 - val_loss: 0.4795 - val_accuracy: 0.8263
Epoch 4/10
104/104 [==============================] - 11s 106ms/step - loss: 0.0400 -
accuracy: 0.9923 - val_loss: 0.2234 - val_accuracy: 0.9266
Epoch 5/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0071 -
accuracy: 1.0000 - val_loss: 0.2361 - val_accuracy: 0.9305
Epoch 6/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0034 -
accuracy: 1.0000 - val_loss: 0.2241 - val_accuracy: 0.9266
Epoch 7/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0021 -
accuracy: 1.0000 - val_loss: 0.2182 - val_accuracy: 0.9266
Epoch 8/10
104/104 [==============================] - 10s 98ms/step - loss: 0.0017 -
accuracy: 1.0000 - val_loss: 0.2192 - val_accuracy: 0.9305
Epoch 9/10
104/104 [==============================] - 10s 98ms/step - loss: 0.0014 -
accuracy: 1.0000 - val_loss: 0.2218 - val_accuracy: 0.9344
Epoch 10/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 0.2165 - val_accuracy: 0.9382


Evaluating the Test metrics
11/11 [==============================] - 10s 291ms/step - loss: 0.2751 -
accuracy: 0.9074
Test loss for fold 1: 0.27506136894226074
Test accuracy for fold 1: 0.9074074029922485
11/11 [==============================] - 2s 143ms/step
Precision for fold 1: 0.9083414893583627
Recall for fold 1: 0.9074074074074074
F1 Score for fold 1: 0.9076904694338656


Epoch 1/10
104/104 [==============================] - 13s 113ms/step - loss: 1.6889 -
accuracy: 0.5618 - val_loss: 0.7950 - val_accuracy: 0.6988
Epoch 2/10
```

```
104/104 [==============================] - 11s 108ms/step - loss: 0.5662 -
accuracy: 0.7867 - val_loss: 0.4806 - val_accuracy: 0.8340
Epoch 3/10
104/104 [==============================] - 11s 108ms/step - loss: 0.2349 -
accuracy: 0.9112 - val_loss: 0.2357 - val_accuracy: 0.9189
Epoch 4/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0917 -
accuracy: 0.9759 - val_loss: 0.2065 - val_accuracy: 0.9266
Epoch 5/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0405 -
accuracy: 0.9913 - val_loss: 0.1976 - val_accuracy: 0.9344
Epoch 6/10
104/104 [==============================] - 11s 106ms/step - loss: 0.0143 -
accuracy: 0.9981 - val_loss: 0.1843 - val_accuracy: 0.9459
Epoch 7/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0077 -
accuracy: 1.0000 - val_loss: 0.1702 - val_accuracy: 0.9537
Epoch 8/10
104/104 [==============================] - 11s 106ms/step - loss: 0.0036 -
accuracy: 1.0000 - val_loss: 0.1670 - val_accuracy: 0.9498
Epoch 9/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0024 -
accuracy: 1.0000 - val_loss: 0.1683 - val_accuracy: 0.9537
Epoch 10/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0018 -
accuracy: 1.0000 - val_loss: 0.1669 - val_accuracy: 0.9459


Evaluating the Test metrics
11/11 [==============================] - 2s 134ms/step - loss: 0.2931 -
accuracy: 0.9352
Test loss for fold 2: 0.29307472705841064
Test accuracy for fold 2: 0.9351851940155029
11/11 [==============================] - 2s 141ms/step
Precision for fold 2: 0.9375098284515365
Recall for fold 2: 0.9351851851851852
F1 Score for fold 2: 0.9348004574033981


Epoch 1/10
104/104 [==============================] - 13s 113ms/step - loss: 1.8972 -
accuracy: 0.5203 - val_loss: 0.8484 - val_accuracy: 0.6371
Epoch 2/10
104/104 [==============================] - 11s 108ms/step - loss: 0.5761 -
accuracy: 0.7867 - val_loss: 0.3972 - val_accuracy: 0.8571
Epoch 3/10
104/104 [==============================] - 10s 98ms/step - loss: 0.3207 -
accuracy: 0.8851 - val_loss: 0.4739 - val_accuracy: 0.8456
```

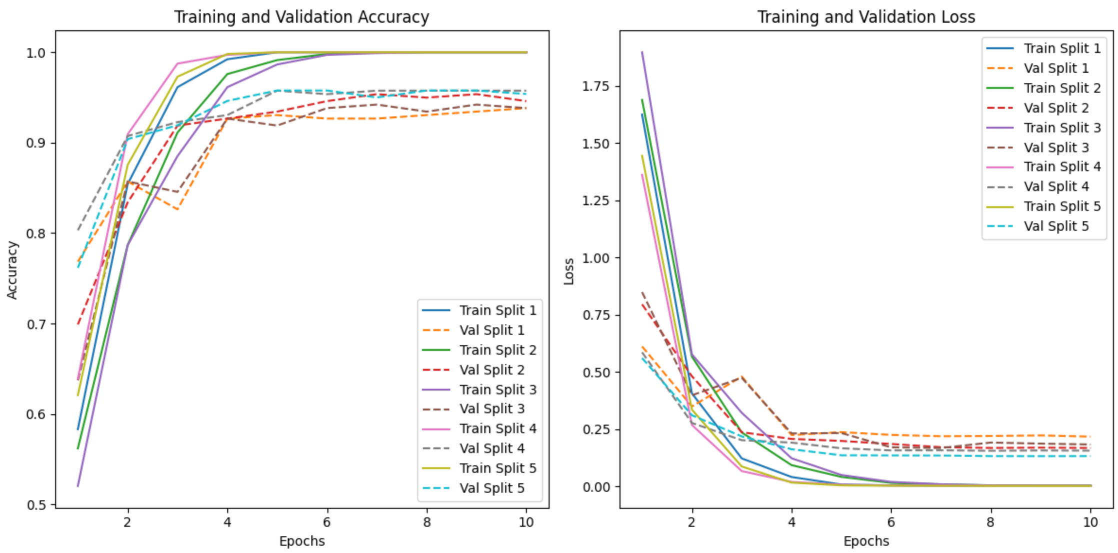
```
Epoch 4/10
104/104 [==============================] - 11s 107ms/step - loss: 0.1219 -
accuracy: 0.9614 - val_loss: 0.2305 - val_accuracy: 0.9266
Epoch 5/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0487 -
accuracy: 0.9865 - val_loss: 0.2321 - val_accuracy: 0.9189
Epoch 6/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0185 -
accuracy: 0.9971 - val_loss: 0.1695 - val_accuracy: 0.9382
Epoch 7/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0082 -
accuracy: 0.9990 - val_loss: 0.1667 - val_accuracy: 0.9421
Epoch 8/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0034 -
accuracy: 1.0000 - val_loss: 0.1907 - val_accuracy: 0.9344
Epoch 9/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0021 -
accuracy: 1.0000 - val_loss: 0.1859 - val_accuracy: 0.9421
Epoch 10/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0017 -
accuracy: 1.0000 - val_loss: 0.1821 - val_accuracy: 0.9382


Evaluating the Test metrics
11/11 [==============================] - 2s 135ms/step - loss: 0.2426 -
accuracy: 0.9198
Test loss for fold 3: 0.24258004128932953
Test accuracy for fold 3: 0.9197530746459961
11/11 [==============================] - 2s 141ms/step
Precision for fold 3: 0.9225461497023901
Recall for fold 3: 0.9197530864197531
F1 Score for fold 3: 0.9202892779027154


Epoch 1/10
104/104 [==============================] - 13s 113ms/step - loss: 1.3616 -
accuracy: 0.6390 - val_loss: 0.5852 - val_accuracy: 0.8031
Epoch 2/10
104/104 [==============================] - 11s 108ms/step - loss: 0.2675 -
accuracy: 0.9093 - val_loss: 0.2763 - val_accuracy: 0.9073
Epoch 3/10
104/104 [==============================] - 11s 108ms/step - loss: 0.0658 -
accuracy: 0.9875 - val_loss: 0.2009 - val_accuracy: 0.9228
Epoch 4/10
104/104 [==============================] - 11s 108ms/step - loss: 0.0186 -
accuracy: 0.9971 - val_loss: 0.1904 - val_accuracy: 0.9305
Epoch 5/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0063 -
```

```
accuracy: 1.0000 - val_loss: 0.1654 - val_accuracy: 0.9575
Epoch 6/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0030 -
accuracy: 1.0000 - val_loss: 0.1562 - val_accuracy: 0.9537
Epoch 7/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0020 -
accuracy: 1.0000 - val_loss: 0.1567 - val_accuracy: 0.9575
Epoch 8/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0016 -
accuracy: 1.0000 - val_loss: 0.1546 - val_accuracy: 0.9575
Epoch 9/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0013 -
accuracy: 1.0000 - val_loss: 0.1560 - val_accuracy: 0.9575
Epoch 10/10
104/104 [==============================] - 10s 97ms/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 0.1551 - val_accuracy: 0.9575


Evaluating the Test metrics
11/11 [==============================] - 2s 134ms/step - loss: 0.1838 -
accuracy: 0.9198
Test loss for fold 4: 0.18379390239715576
Test accuracy for fold 4: 0.9197530746459961
11/11 [==============================] - 2s 142ms/step
Precision for fold 4: 0.9203762688044644
Recall for fold 4: 0.9197530864197531
F1 Score for fold 4: 0.91991264595853


Epoch 1/10
104/104 [==============================] - 13s 113ms/step - loss: 1.4446 -
accuracy: 0.6207 - val_loss: 0.5601 - val_accuracy: 0.7615
Epoch 2/10
104/104 [==============================] - 11s 108ms/step - loss: 0.3344 -
accuracy: 0.8755 - val_loss: 0.3099 - val_accuracy: 0.9038
Epoch 3/10
104/104 [==============================] - 11s 108ms/step - loss: 0.0859 -
accuracy: 0.9730 - val_loss: 0.2175 - val_accuracy: 0.9192
Epoch 4/10
104/104 [==============================] - 11s 108ms/step - loss: 0.0155 -
accuracy: 0.9981 - val_loss: 0.1613 - val_accuracy: 0.9462
Epoch 5/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0033 -
accuracy: 1.0000 - val_loss: 0.1348 - val_accuracy: 0.9577
Epoch 6/10
104/104 [==============================] - 11s 108ms/step - loss: 0.0019 -
accuracy: 1.0000 - val_loss: 0.1344 - val_accuracy: 0.9577
Epoch 7/10
```

```
104/104 [==============================] - 11s 107ms/step - loss: 0.0015 -
accuracy: 1.0000 - val_loss: 0.1339 - val_accuracy: 0.9500
Epoch 8/10
104/104 [==============================] - 12s 112ms/step - loss: 0.0012 -
accuracy: 1.0000 - val_loss: 0.1313 - val_accuracy: 0.9577
Epoch 9/10
104/104 [==============================] - 11s 107ms/step - loss: 0.0010 -
accuracy: 1.0000 - val_loss: 0.1310 - val_accuracy: 0.9577
Epoch 10/10
104/104 [==============================] - 10s 97ms/step - loss: 8.8104e-04 -
accuracy: 1.0000 - val_loss: 0.1314 - val_accuracy: 0.9538


Evaluating the Test metrics
11/11 [==============================] - 3s 266ms/step - loss: 0.1983 -
accuracy: 0.9226
Test loss for fold 5: 0.19832882285118103
Test accuracy for fold 5: 0.9226006269454956
11/11 [==============================] - 2s 142ms/step
Precision for fold 5: 0.9223827986465892
Recall for fold 5: 0.9226006191950464
F1 Score for fold 5: 0.92243921584306


Standard deviation of accuracy across folds: 0.008850220332613429
Average Accuracy across folds: 0.9209398746490478
Average Precision across folds: 0.9222313069926686
Average Recall across folds: 0.9209398769254291
Average F1 Score across folds: 0.9210264133083138
```

[9]:
```
import matplotlib.pyplot as plt

# Determine the maximum length among all lists
max_length = max(len(train_acc) for train_acc in train_acc_list)

# Pad the shorter lists with zeros to match the maximum length
for i in range(n_splits):
    train_acc_list[i] += [0] * (max_length - len(train_acc_list[i]))
    val_acc_list[i] += [0] * (max_length - len(val_acc_list[i]))
    train_loss_list[i] += [0] * (max_length - len(train_loss_list[i]))
    val_loss_list[i] += [0] * (max_length - len(val_loss_list[i]))

# Plotting
epochs = range(1, max_length + 1)

plt.figure(figsize=(12, 6))
```

```python
# Plotting Training and Validation Accuracy
plt.subplot(1, 2, 1)
for i in range(n_splits):
    plt.plot(epochs, train_acc_list[i], label=f'Train Split {i+1}')
    plt.plot(epochs, val_acc_list[i], label=f'Val Split {i+1}', linestyle='--')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plotting Training and Validation Loss
plt.subplot(1, 2, 2)
for i in range(n_splits):
    plt.plot(epochs, train_loss_list[i], label=f'Train Split {i+1}')
    plt.plot(epochs, val_loss_list[i], label=f'Val Split {i+1}', linestyle='--')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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



[19]:
```python
from keras.applications.vgg16 import VGG16, preprocess_input
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
from sklearn.model_selection import StratifiedKFold
import numpy as np

encoder = LabelEncoder()
labels_encoded = encoder.fit_transform(labels)

# Load the pre-trained VGG16 model without the top (classification) layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))

# Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Define SVM pipeline
svm_pipeline = Pipeline([('scaler', StandardScaler()), ('svm',␣
 ↪SVC(kernel='linear', probability=True))])

# Initialize lists to store evaluation metrics
accuracy_values = []
precision_values = []
recall_values = []
f1_values = []

# Perform k-fold cross-validation
n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=777)
fold_var = 1

for train_index, test_index in skf.split(img_data, labels_encoded):  # Ensure␣
 ↪labels_encoded is defined correctly as the non-categorical labels
    X_train, X_test = img_data[train_index], img_data[test_index]
    y_train, y_test = labels_encoded[train_index], labels_encoded[test_index]

    # Extract features using the VGG16 model and flatten them
    X_train_features = base_model.predict(preprocess_input(X_train))
    X_train_features = X_train_features.reshape(X_train_features.shape[0], -1)

    X_test_features = base_model.predict(preprocess_input(X_test))
    X_test_features = X_test_features.reshape(X_test_features.shape[0], -1)

    # Fit and transform the training data with StandardScaler and train the SVM
    svm_pipeline.fit(X_train_features, y_train)

    # Predict using the trained SVM pipeline
```

```
    predictions = svm_pipeline.predict(X_test_features)
    print(f'Fold {fold_var}:')
    print(f'Accuracy: {accuracy_score(y_test, predictions)}')
    print(f'Precision: {precision_score(y_test, predictions,␣
 ↪average="weighted")}')
    print(f'Recall: {recall_score(y_test, predictions, average="weighted")}')
    print(f'F1 Score: {f1_score(y_test, predictions, average="weighted")}')

    # Calculate evaluation metrics
    accuracy_values.append(accuracy_score(y_test, predictions))
    precision_values.append(precision_score(y_test, predictions,␣
 ↪average='weighted'))
    recall_values.append(recall_score(y_test, predictions, average='weighted'))
    f1_values.append(f1_score(y_test, predictions, average='weighted'))

    # Increment the fold number
    fold_var += 1

# Calculate average metrics across all folds
avg_accuracy = np.mean(accuracy_values)
avg_precision = np.mean(precision_values)
avg_recall = np.mean(recall_values)
avg_f1 = np.mean(f1_values)

print(f'Average test Accuracy across folds: {avg_accuracy}')
print(f'Average Precision across folds: {avg_precision}')
print(f'Average Recall across folds: {avg_recall}')
print(f'Average F1 Score across folds: {avg_f1}')
```

```
41/41 [==============================] - 10s 250ms/step
11/11 [==============================] - 2s 148ms/step
Fold 1:
Accuracy: 0.9598765432098766
Precision: 0.9620569043250449
Recall: 0.9598765432098766
F1 Score: 0.9595809528875071
41/41 [==============================] - 6s 138ms/step
11/11 [==============================] - 1s 141ms/step
Fold 2:
Accuracy: 0.9290123456790124
Precision: 0.9307860902476693
Recall: 0.9290123456790124
F1 Score: 0.9292381459241569
41/41 [==============================] - 6s 144ms/step
11/11 [==============================] - 1s 143ms/step
Fold 3:
Accuracy: 0.9660493827160493
```

```
Precision: 0.9663775683757447
Recall: 0.9660493827160493
F1 Score: 0.9658663233744746
41/41 [==============================] - 6s 140ms/step
11/11 [==============================] - 1s 142ms/step
Fold 4:
Accuracy: 0.9598765432098766
Precision: 0.9613916475549054
Recall: 0.9598765432098766
F1 Score: 0.9598198652756754
41/41 [==============================] - 10s 250ms/step
11/11 [==============================] - 1s 144ms/step
Fold 5:
Accuracy: 0.9597523219814241
Precision: 0.9601101715746051
Recall: 0.9597523219814241
F1 Score: 0.9598046349590906
Average test Accuracy across folds: 0.9549134273592479
Average Precision across folds: 0.956144476415594
Average Recall across folds: 0.9549134273592479
Average F1 Score across folds: 0.9548619844841809
```

[5]:
```
!jupyter nbconvert --to pdf '/content/drive/MyDrive/Colab_Notebooks/
↪Sheep_Full_Images_Classification.ipynb'
```

```
[NbConvertApp] Converting notebook
/content/drive/MyDrive/Colab_Notebooks/Sheep_Full_Images_Classification.ipynb to
pdf
[NbConvertApp] Support files will be in Sheep_Full_Images_Classification_files/
[NbConvertApp] Making directory ./Sheep_Full_Images_Classification_files
[NbConvertApp] Making directory ./Sheep_Full_Images_Classification_files
[NbConvertApp] Making directory ./Sheep_Full_Images_Classification_files
[NbConvertApp] Writing 178570 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 544764 bytes to
/content/drive/MyDrive/Colab_Notebooks/Sheep_Full_Images_Classification.pdf
```