CHRIST (Deemed to be University)

Department of Computer Science

5MCA-A - Neural Networks and Deep Learning (MCA572)

Regular Lab Questions - Lab 5

Implementing CNN on the Fashion-MNIST Dataset

Anupam Kumar 2347104

25 October 2024

1. Dataset Overview:

- Visualize a few samples from the dataset, displaying their corresponding labels.

```
!pip install tensorflow --upgrade

# Install required libraries if not already installed
!pip install tensorflow numpy matplotlib seaborn scikit-learn

# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

Downloading Dataset from kaggle

```
from google.colab import files
files.upload()  # Upload `kaggle.json`

# Move kaggle.json to the correct directory
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

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

```
# Unzip the dataset
!unzip intel-image-classification.zip -d intel_dataset
```

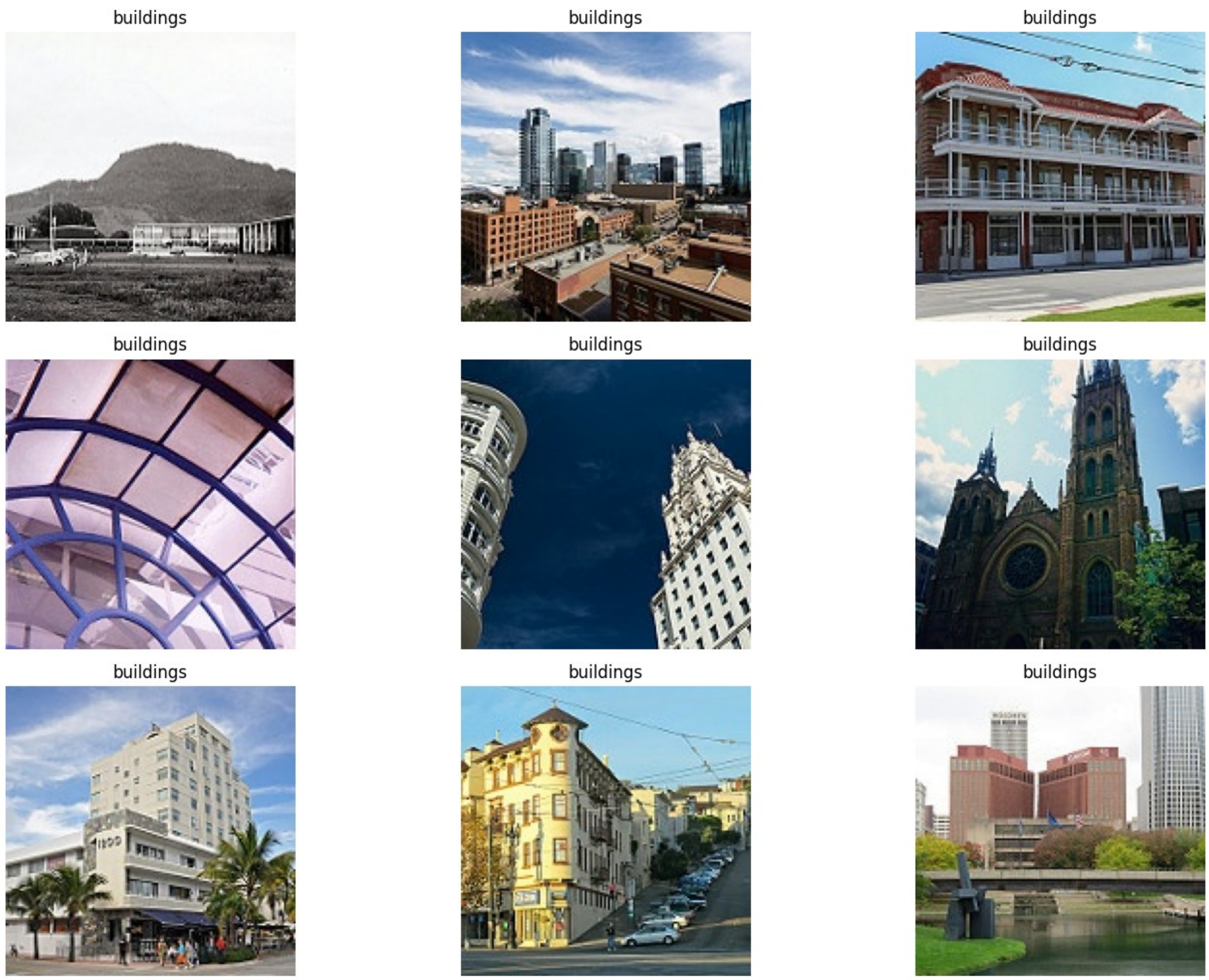
```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
Dataset URL: https://www.kaggle.com/datasets/puneet6060/intel-image-
classification
License(s): copyright-authors
intel-image-classification.zip: Skipping, found more recently modified
local copy (use --force to force download)
Archive:  intel-image-classification.zip
replace intel_dataset/seg_pred/seg_pred/10004.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: y
  inflating: intel_dataset/seg_pred/seg_pred/10004.jpg
replace intel_dataset/seg_pred/seg_pred/10005.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: a
error:  invalid response [a]
replace intel_dataset/seg_pred/seg_pred/10005.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename:
```

```
# Set random seed for reproducibility
tf.random.set_seed(42)
np.random.seed(42)

# Define the data paths (adjust these based on your Google Drive
structure)
base_dir = '/content/intel_dataset'
train_dir = os.path.join(base_dir, 'seg_train/seg_train')
test_dir = os.path.join(base_dir, 'seg_test/seg_test')

# Task 1: Dataset Overview
# Create a data generator for visualization
datagen = ImageDataGenerator(rescale=1./255)

# Load training data
train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

# Get class names
class_names = list(train_generator.class_indices.keys())
print("\nClass Names:", class_names)

# Function to display sample images
def display_samples():
```

```python
    # Get a batch of images and their labels
    images, labels = next(train_generator)

    plt.figure(figsize=(15, 10))
    for i in range(9):
        plt.subplot(3, 3, i+1)
        plt.imshow(images[i])
        plt.title(class_names[np.argmax(labels[i])])
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Display sample count per class
print("\nSample count per class:")
for folder in os.listdir(train_dir):
    path = os.path.join(train_dir, folder)
    print(f"{folder}: {len(os.listdir(path))} images")

# Display sample images
print("\nDisplaying sample images from each class:")
display_samples()

# Calculate and display basic statistics
print("\nDataset Statistics:")
print(f"Image dimensions: {train_generator.image_shape}")
print(f"Number of classes: {len(class_names)}")
total_train = sum(len(os.listdir(os.path.join(train_dir, folder))) for
folder in os.listdir(train_dir))
total_test = sum(len(os.listdir(os.path.join(test_dir, folder))) for
folder in os.listdir(test_dir))
print(f"Total training images: {total_train}")
print(f"Total test images: {total_test}")
```

```
Found 14034 images belonging to 6 classes.

Class Names: ['buildings', 'forest', 'glacier', 'mountain', 'sea',
'street']

Sample count per class:
glacier: 2404 images
sea: 2274 images
mountain: 2512 images
buildings: 2191 images
street: 2382 images
forest: 2271 images

Displaying sample images from each class:
```




```
Dataset Statistics:
Image dimensions: (150, 150, 3)
Number of classes: 6
Total training images: 14034
Total test images: 3000
```

- Purpose: Understand and visualize the dataset before training
- Why This Approach:
- Using ImageDataGenerator for easy data loading and preprocessing
- Visualizing sample images to verify data quality
- Displaying class distribution to check for imbalances
- Normalizing pixel values (0-255 → 0-1) for better training

2. Model Architecture:

o Design a CNN model with at least 3 convolutional layers, followed by pooling layers and fully connected (dense) layers.

o Experiment with different kernel sizes, activation functions (such as ReLU), and pooling strategies (max-pooling or average pooling).

o Experiment with different kernel sizes, activation functions (such as ReLU), and pooling strategies (max-pooling or average pooling). o Implement batch normalization and dropout techniques to improve the generalization of your model.

```
# Task 2: Model Architecture
from tensorflow.keras import layers, models

def create_cnn_model():
    model = models.Sequential([
        # First Convolutional Block
        layers.Conv2D(32, (3, 3), padding='same', input_shape=(150, 
150, 3)),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Second Convolutional Block
        layers.Conv2D(64, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Third Convolutional Block
        layers.Conv2D(128, (3, 3), padding='same'),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.MaxPooling2D((2, 2)),
        layers.Dropout(0.25),

        # Dense Layers
        layers.Flatten(),
        layers.Dense(512),
        layers.BatchNormalization(),
        layers.Activation('relu'),
        layers.Dropout(0.5),
        layers.Dense(6, activation='softmax')
    ])

    return model
```

```python
# Create and display model
model = create_cnn_model()
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ conv2d_3 (Conv2D)                    │ (None, 150, 150, 32)        │
896 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ batch_normalization_4                │ (None, 150, 150, 32)        │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ activation_4 (Activation)            │ (None, 150, 150, 32)        │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 75, 75, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ dropout_4 (Dropout)                  │ (None, 75, 75, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ conv2d_4 (Conv2D)                    │ (None, 75, 75, 64)          │
18,496 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ batch_normalization_5                │ (None, 75, 75, 64)          │
256 │
│ (BatchNormalization)                 │                             │
│
```

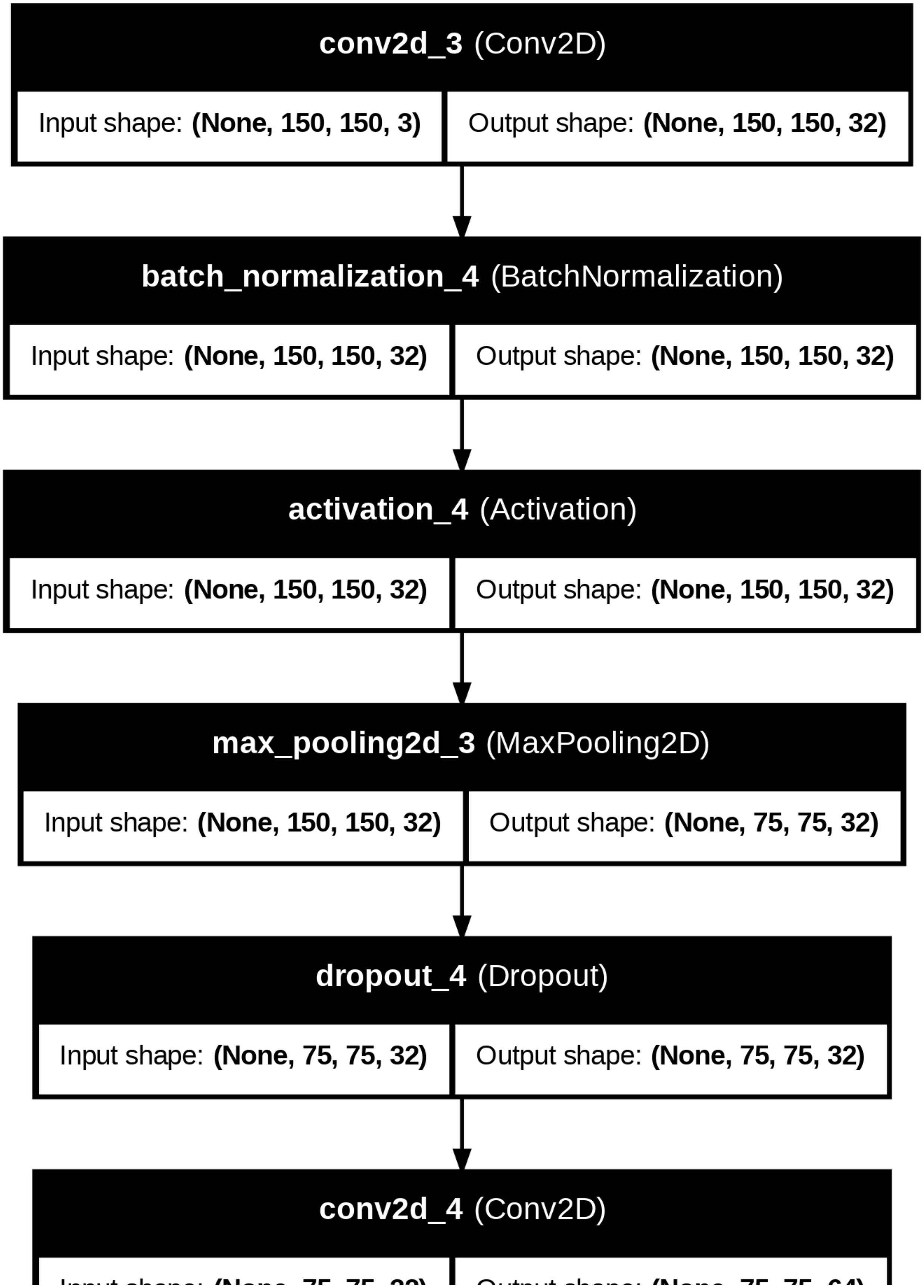
```
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ activation_5 (Activation)               │ (None, 75, 75, 64)             │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ max_pooling2d_4 (MaxPooling2D)          │ (None, 37, 37, 64)             │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ dropout_5 (Dropout)                     │ (None, 37, 37, 64)             │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ conv2d_5 (Conv2D)                       │ (None, 37, 37, 128)            │
73,856 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ batch_normalization_6                   │ (None, 37, 37, 128)            │
512 │
│ (BatchNormalization)                    │                                │
│
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ activation_6 (Activation)               │ (None, 37, 37, 128)            │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ max_pooling2d_5 (MaxPooling2D)          │ (None, 18, 18, 128)            │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ dropout_6 (Dropout)                     │ (None, 18, 18, 128)            │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ flatten_1 (Flatten)                     │ (None, 41472)                  │
0 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ dense_2 (Dense)                         │ (None, 512)                    │
21,234,176 │
├─────────────────────────────────────────┼────────────────────────────────┼
─────────────────┤
│ batch_normalization_7                   │ (None, 512)                    │
2,048 │
│ (BatchNormalization)                    │                                │
│
├─────────────────────────────────────────┼────────────────────────────────┼
```

```
───────────────────┤
│ activation_7 (Activation)            │ (None, 512)                  │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
───────────────────┤
│ dropout_7 (Dropout)                  │ (None, 512)                  │
0 │
├──────────────────────────────────────┼──────────────────────────────┼
───────────────────┤
│ dense_3 (Dense)                      │ (None, 6)                    │
3,078 │
└──────────────────────────────────────┴──────────────────────────────┴
───────────────────┘

 Total params: 21,333,446 (81.38 MB)

 Trainable params: 21,331,974 (81.38 MB)

 Non-trainable params: 1,472 (5.75 KB)
```

- Purpose: Design an effective CNN for image classification
- Why This Approach:
- Three convolutional blocks with increasing filters (32→64→128) to capture hierarchical features
- BtchNormalization for stable training
- MaxPooling to reduce dimensions and computation
- Dropout layers to prevent overfitting
- Dense layers at end for final classification

Visualize model architecture

```
!pip install pydot
!pip install graphviz

Requirement already satisfied: pydot in
/usr/local/lib/python3.10/dist-packages (3.0.2)
Requirement already satisfied: pyparsing>=3.0.9 in
/usr/local/lib/python3.10/dist-packages (from pydot) (3.2.0)
Requirement already satisfied: graphviz in
/usr/local/lib/python3.10/dist-packages (0.20.3)

# Visualize model architecture
from tensorflow.keras.utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=True)
```



conv2d_3 (Conv2D)
Input shape: (None, 150, 150, 3)
Output shape: (None, 150, 150, 32)
batch_normalization_4 (BatchNormalization)
Input shape: (None, 150, 150, 32)
Output shape: (None, 150, 150, 32)
activation_4 (Activation)
Input shape: (None, 150, 150, 32)
Output shape: (None, 150, 150, 32)
max_pooling2d_3 (MaxPooling2D)
Input shape: (None, 150, 150, 32)
Output shape: (None, 75, 75, 32)
dropout_4 (Dropout)
Input shape: (None, 75, 75, 32)
Output shape: (None, 75, 75, 32)
conv2d_4 (Conv2D)

3. Model Training:

o Split the dataset into training and test sets.

The dataset is already organized into train and test directories, so no further splitting is needed. We use ImageDataGenerator to handle both sets.

o Compile the model using an appropriate loss function (categorical cross- entropy) and an optimizer (such as Adam or SGD).

o Train the model for a sufficient number of epochs, monitoring the training and validation accuracy.

```
# Create data generators with augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Create generators
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

# Compile model
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
Found 11230 images belonging to 6 classes.
Found 2804 images belonging to 6 classes.
```

```python
# Train model
history = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=3,
            restore_best_weights=True
        ),
        tf.keras.callbacks.ModelCheckpoint(
            'best_model.keras',
            save_best_only=True,
            monitor='val_accuracy'
        )
    ]
)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset`
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

Epoch 1/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 179s 501ms/step - accuracy: 0.5310 -
loss: 1.3020 - val_accuracy: 0.2874 - val_loss: 2.2031
Epoch 2/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 173s 492ms/step - accuracy: 0.6693 -
loss: 0.8853 - val_accuracy: 0.7143 - val_loss: 0.7392
Epoch 3/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 174s 496ms/step - accuracy: 0.7146 -
loss: 0.7670 - val_accuracy: 0.6138 - val_loss: 1.1346
Epoch 4/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 175s 498ms/step - accuracy: 0.7290 -
loss: 0.7221 - val_accuracy: 0.7429 - val_loss: 0.6727
Epoch 5/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 172s 488ms/step - accuracy: 0.7509 -
loss: 0.6904 - val_accuracy: 0.6416 - val_loss: 1.1989
Epoch 6/20
351/351 ━━━━━━━━━━━━━━━━━━━━ 170s 485ms/step - accuracy: 0.7613 -
loss: 0.6563 - val_accuracy: 0.6983 - val_loss: 0.8041
Epoch 7/20
```

```
351/351 ━━━━━━━━━━━━━━━━━━━━ 171s 487ms/step - accuracy: 0.7714 -
loss: 0.6178 - val_accuracy: 0.5945 - val_loss: 1.3397
```

- Purpose: Train the model effectively with data augmentation
- Why This Approach:
- Data augmentation (rotation, flips, shifts) to increase effective dataset size
- Validation split to monitor overfitting
- Early stopping to prevent wasting computation
- Adam optimizer and categorical
- crossentropy for reliable training
- Model checkpointing to save best weights

4. Evaluation:

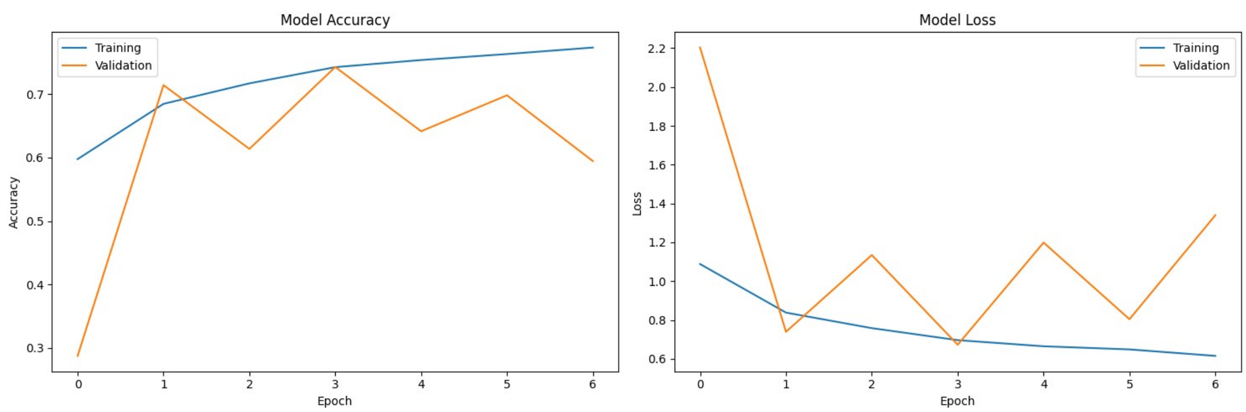
o Evaluate the trained model on the test set and report the accuracy.

o Plot the training and validation accuracy/loss curves to visualize the model's performance.

```
# Task 4: Model Evaluation

# Plot training history
def plot_training_history(history):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    # Accuracy plot
    ax1.plot(history.history['accuracy'], label='Training')
    ax1.plot(history.history['val_accuracy'], label='Validation')
    ax1.set_title('Model Accuracy')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Accuracy')
    ax1.legend()

    # Loss plot
    ax2.plot(history.history['loss'], label='Training')
    ax2.plot(history.history['val_loss'], label='Validation')
    ax2.set_title('Model Loss')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Loss')
    ax2.legend()

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

```python
# Create test generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)

# Evaluate model
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"\nTest Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")

# Plot training history
plot_training_history(history)

# Generate confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get predictions
predictions = model.predict(test_generator)
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes
```

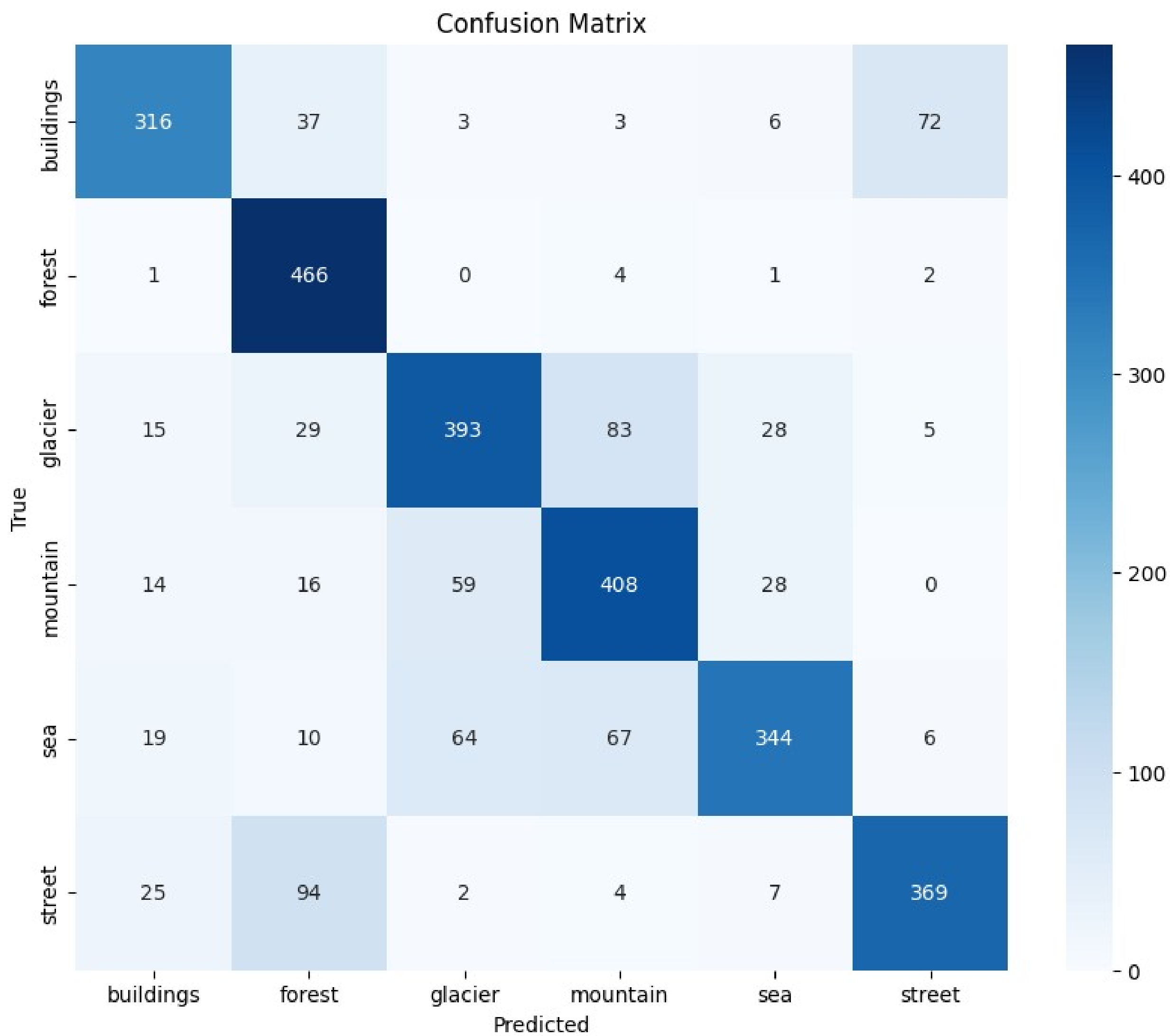
```
Found 3000 images belonging to 6 classes.
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - accuracy: 0.7831 - loss:
0.6021

Test Accuracy: 0.7653
Test Loss: 0.6463
```




```
94/94 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step
```

o Display the confusion matrix for the test set to analyze misclassified samples.

```python
# Create confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```



- Purpose: Assess model performance comprehensively
- Why This Approach:

- Plot training/validation curves to visualize learning progress
- Confusion matrix to identify per-class performance
- Test set evaluation for unbiased performance estimate
- Multiple metrics (accuracy, loss) for thorough evaluation

5. Optimization :

o Experiment with data augmentation techniques (rotation, flipping, zooming) to further improve the model's performance.

```
# Enhanced data augmentation
optimized_train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2
)
```

o Fine-tune hyperparameters like learning rate, batch size, and the number of filters in each layer.

```
# Enhanced data augmentation
optimized_train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2
)

# Create generators with optimized parameters
optimized_train_generator = 
optimized_train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=16,  # Reduced batch size
    class_mode='categorical',
    subset='training'
```

```
)

optimized_validation_generator =
optimized_train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=16,
    class_mode='categorical',
    subset='validation'
)

# Option 1: Using constant learning rate with ReduceLROnPlateau
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),  # Fixed
learning rate
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train with optimized parameters
optimized_history = model.fit(
    optimized_train_generator,
    epochs=30,
    validation_data=optimized_validation_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=5,
            restore_best_weights=True
        ),
        tf.keras.callbacks.ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.2,
            patience=3,
            min_lr=1e-6
        ),
        tf.keras.callbacks.ModelCheckpoint(
            'best_model.keras',
            save_best_only=True,
            monitor='val_accuracy'
        )
    ]
)


# Evaluate optimized model
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"\nOptimized Model Test Accuracy: {test_accuracy:.4f}")
print(f"Optimized Model Test Loss: {test_loss:.4f}")
```

```
# Plot optimized training history
plot_training_history(optimized_history)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-1f855ce87038> in <cell line: 2>()
      1 # Enhanced data augmentation
----> 2 optimized_train_datagen = ImageDataGenerator(
      3     rescale=1./255,
      4     rotation_range=20,
      5     width_shift_range=0.2,

NameError: name 'ImageDataGenerator' is not defined
```

```
# Plot optimized training history
plot_training_history(optimized_history)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-77be26fa653a> in <cell line: 2>()
      1 # Plot optimized training history
----> 2 plot_training_history(optimized_history)

NameError: name 'plot_training_history' is not defined
```

- Purpose: Improve model performance through various techniques
- Why This Approach:
- Enhanced data augmentation for better generalization
- Learning rate scheduling for optimal convergence
- Reduced batch size for better generalization
- Additional callbacks (ReduceLROnPlateau) for adaptive training
- Longer training with patience for finding best model