In [*]:
```
sizes in training a neural network on the MNIST dataset and observe how it affects the convergence rate and final accuracy.
```

In [*]:
```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the CNN model
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Experiment with different batch sizes
batch_sizes = [16, 32, 64, 128, 256]
history_dict = {}

for batch_size in batch_sizes:
    print(f"\nTraining with batch size: {batch_size}")
    model = create_model()
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=10,
                        batch_size=batch_size,
                        verbose=1)
    history_dict[batch_size] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy
for batch_size, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'Train Acc (batch_size={batch_size})')
    plt.plot(history.history['val_accuracy'], label=f'Val Acc (batch_size={batch_size})')

plt.title('Training and Validation Accuracy for Different Batch Sizes')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
# Plot training and validation loss
plt.figure(figsize=(14, 8))
for batch_size, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'Train Loss (batch_size={batch_size})')
    plt.plot(history.history['val_loss'], label=f'Val Loss (batch_size={batch_size})')

plt.title('Training and Validation Loss for Different Batch Sizes')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

In [*]:
```
##. Compare the performance of different learning rates on a simple feed-forward neural network trained on the MNIST dataset.
```

In [*]:
```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the FFNN model
def create_model(learning_rate):
    model = Sequential([
        Flatten(input_shape=(28, 28, 1)),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Experiment with different learning rates
learning_rates = [0.0001, 0.001, 0.01, 0.1]
history_dict = {}

for lr in learning_rates:
    print(f"\nTraining with learning rate: {lr}")
    model = create_model(lr)
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=20,
                        batch_size=64,
                        verbose=1)
    history_dict[lr] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy
for lr, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'Train Acc (learning_rate={lr})')
    plt.plot(history.history['val_accuracy'], label=f'Val Acc (learning_rate={lr})')

plt.title('Training and Validation Accuracy for Different Learning Rates')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(14, 8))
for lr, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'Train Loss (learning_rate={lr})')
    plt.plot(history.history['val_loss'], label=f'Val Loss (learning_rate={lr})')

plt.title('Training and Validation Loss for Different Learning Rates')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

In [*]:
```
sizes in training a neural network on the MNIST dataset and observe how it affects the convergence rate and final accuracy.
```

In [*]:
```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the CNN model
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Experiment with different batch sizes
batch_sizes = [16, 32, 64, 128, 256]
history_dict = {}

for batch_size in batch_sizes:
    print(f"\nTraining with batch size: {batch_size}")
    model = create_model()
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=20,
                        batch_size=batch_size,
                        verbose=1)
    history_dict[batch_size] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy
for batch_size, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'Train Acc (batch_size={batch_size})')
    plt.plot(history.history['val_accuracy'], label=f'Val Acc (batch_size={batch_size})')

plt.title('Training and Validation Accuracy for Different Batch Sizes')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(14, 8))
for batch_size, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'Train Loss (batch_size={batch_size})')
    plt.plot(history.history['val_loss'], label=f'Val Loss (batch_size={batch_size})')

plt.title('Training and Validation Loss for Different Batch Sizes')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

In [*]:
```
ral network (CNN) on a small subset of the MNIST dataset and compare its performance to a basic feed-forward neural network.
```

In [*]:
```python
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the CNN model
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Experiment with different batch sizes
batch_sizes = [16, 32, 64, 128, 256]
history_dict = {}

for batch_size in batch_sizes:
    print(f"\nTraining with batch size: {batch_size}")
    model = create_model()
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=20,
                        batch_size=batch_size,
                        verbose=1)
    history_dict[batch_size] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy
for batch_size, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'Train Acc (batch_size={batch_size})')
    plt.plot(history.history['val_accuracy'], label=f'Val Acc (batch_size={batch_size})')

plt.title('Training and Validation Accuracy for Different Batch Sizes')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure(figsize=(14, 8))
for batch_size, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'Train Loss (batch_size={batch_size})')
    plt.plot(history.history['val_loss'], label=f'Val Loss (batch_size={batch_size})')

plt.title('Training and Validation Loss for Different Batch Sizes')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

In [*]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load and preprocess the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Convert labels to one-hot encoding
y_train_one_hot = tf.keras.utils.to_categorical(y_train, 3)
y_test_one_hot = tf.keras.utils.to_categorical(y_test, 3)

# Define a simple perceptron model
def create_simple_perceptron():
    model = Sequential([
        Dense(3, activation='softmax', input_shape=(4,))
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Define a multi-layer perceptron (MLP) model
def create_mlp():
    model = Sequential([
        Dense(10, activation='relu', input_shape=(4,)),
        Dense(10, activation='relu'),
        Dense(3, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Train and evaluate the simple perceptron
simple_perceptron = create_simple_perceptron()
history_simple = simple_perceptron.fit(X_train, y_train_one_hot, epochs=50, validation_data=(X_test, y_test_one_hot), verbose
simple_perceptron_accuracy = simple_perceptron.evaluate(X_test, y_test_one_hot, verbose=0)[1]

# Train and evaluate the MLP
mlp = create_mlp()
history_mlp = mlp.fit(X_train, y_train_one_hot, epochs=50, validation_data=(X_test, y_test_one_hot), verbose=0)
mlp_accuracy = mlp.evaluate(X_test, y_test_one_hot, verbose=0)[1]

print(f'Simple Perceptron Accuracy: {simple_perceptron_accuracy:.4f}')
print(f'MLP Accuracy: {mlp_accuracy:.4f}')

# Plotting the results
plt.figure(figsize=(14, 5))

# Plot training and validation accuracy for both models
plt.subplot(1, 2, 1)
plt.plot(history_simple.history['accuracy'], label='Simple Perceptron - Train')
plt.plot(history_simple.history['val_accuracy'], label='Simple Perceptron - Val')
plt.plot(history_mlp.history['accuracy'], label='MLP - Train')
plt.plot(history_mlp.history['val_accuracy'], label='MLP - Val')
plt.title('Model Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss for both models
plt.subplot(1, 2, 2)
plt.plot(history_simple.history['loss'], label='Simple Perceptron - Train')
plt.plot(history_simple.history['val_loss'], label='Simple Perceptron - Val')
plt.plot(history_mlp.history['loss'], label='MLP - Train')
plt.plot(history_mlp.history['val_loss'], label='MLP - Val')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

In [*]:
```
erent sizes of hidden layers in a neural network trained on the MNIST dataset and observe how the model performance changes.
```

In [*]:
```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define model architectures with different hidden layer configurations
def create_mlp(hidden_layers):
    model = Sequential()
    model.add(Flatten(input_shape=(28, 28, 1)))
    for layer_size in hidden_layers:
        model.add(Dense(layer_size, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Different hidden layer configurations to experiment with
hidden_layer_configs = {
    'one_hidden_32': [32],
    'one_hidden_64': [64],
    'two_hidden_32_32': [32, 32],
    'two_hidden_64_64': [64, 64],
    'three_hidden_32_32_32': [32, 32, 32]
}

history_dict = {}

# Train and evaluate models with different hidden layer configurations
for config_name, hidden_layers in hidden_layer_configs.items():
    print(f"\nTraining with configuration: {config_name}")
    model = create_mlp(hidden_layers)
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=20,
                        batch_size=64,
                        verbose=1)
    history_dict[config_name] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy for different configurations
plt.subplot(1, 2, 1)
for config_name, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'{config_name} - Train')
    plt.plot(history.history['val_accuracy'], label=f'{config_name} - Val')

plt.title('Model Accuracy for Different Hidden Layer Configurations')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss for different configurations
plt.subplot(1, 2, 2)
for config_name, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'{config_name} - Train')
    plt.plot(history.history['val_loss'], label=f'{config_name} - Val')

plt.title('Model Loss for Different Hidden Layer Configurations')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

In [*]:
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Define the neural network model
def create_model(activation):
    model = Sequential([
        Flatten(input_shape=(32, 32, 3)),
        Dense(512, activation=activation),
        Dropout(0.5),
        Dense(512, activation=activation),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Activation functions to experiment with
activations = ['relu', 'tanh', 'sigmoid']
history_dict = {}

# Train and evaluate the model with different activation functions
for activation in activations:
    print(f"\nTraining with activation function: {activation}")
    model = create_model(activation)
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=20,
                        batch_size=64,
                        verbose=1)
    history_dict[activation] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy for different activation functions
plt.subplot(1, 2, 1)
for activation, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'{activation} - Train')
    plt.plot(history.history['val_accuracy'], label=f'{activation} - Val')

plt.title('Model Accuracy for Different Activation Functions')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss for different activation functions
plt.subplot(1, 2, 2)
for activation, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'{activation} - Train')
    plt.plot(history.history['val_loss'], label=f'{activation} - Val')

plt.title('Model Loss for Different Activation Functions')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

In [*]:
```
nt with different kernel sizes and numbers of filters in the convolutional layers to observe their effect on model accuracy.
```

In [*]:
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Define the CNN model
def create_cnn_model(kernel_size, num_filters):
    model = Sequential([
        Conv2D(num_filters, (kernel_size, kernel_size), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(num_filters * 2, (kernel_size, kernel_size), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Different kernel sizes and filter configurations to experiment with
configurations = [
    {'kernel_size': 3, 'num_filters': 32},
    {'kernel_size': 5, 'num_filters': 32},
    {'kernel_size': 3, 'num_filters': 64},
    {'kernel_size': 5, 'num_filters': 64}
]

history_dict = {}

# Train and evaluate the models with different configurations
for config in configurations:
    kernel_size = config['kernel_size']
    num_filters = config['num_filters']
    config_name = f'kernel_{kernel_size}_filters_{num_filters}'

    print(f"\nTraining with configuration: {config_name}")
    model = create_cnn_model(kernel_size, num_filters)
    history = model.fit(x_train, y_train,
                        validation_data=(x_test, y_test),
                        epochs=20,
                        batch_size=64,
                        verbose=1)
    history_dict[config_name] = history

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy for different configurations
plt.subplot(1, 2, 1)
for config_name, history in history_dict.items():
    plt.plot(history.history['accuracy'], label=f'{config_name} - Train')
    plt.plot(history.history['val_accuracy'], label=f'{config_name} - Val')

plt.title('Model Accuracy for Different Kernel Sizes and Filter Numbers')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss for different configurations
plt.subplot(1, 2, 2)
for config_name, history in history_dict.items():
    plt.plot(history.history['loss'], label=f'{config_name} - Train')
    plt.plot(history.history['val_loss'], label=f'{config_name} - Val')

plt.title('Model Loss for Different Kernel Sizes and Filter Numbers')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

In [*]:
```
Caltech 101) with and without data augmentation. Evaluate the benefits of transfer learning combined with data augmentation.
```

In [*]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Define directories for the dataset (replace with actual paths)
train_dir = 'path_to_caltech101/train'
val_dir = 'path_to_caltech101/val'

# ImageDataGenerators for loading and augmenting data
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical'
)
```

In [*]:
```
def build_model(base_model):
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.5)(x)
    predictions = Dense(101, activation='softmax')(x)  # Caltech-101 has 101 classes
    model = Model(inputs=base_model.input, outputs=predictions)
    return model

# Load pre-trained VGG16 model + higher level layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
model = build_model(base_model)

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [*]:
```
# Training without data augmentation
history_without_aug = model.fit(
    val_generator,  # Using validation data without augmentation
    epochs=10,
    validation_data=val_generator
)

# Unfreeze some layers and fine-tune the model
for layer in base_model.layers[-4:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Training with data augmentation
history_with_aug = model.fit(
    train_generator,
    epochs=10,
    validation_data=val_generator
)

# Plotting the results
plt.figure(figsize=(14, 8))

# Plot training and validation accuracy for both configurations
plt.subplot(1, 2, 1)
plt.plot(history_without_aug.history['accuracy'], label='Without Aug - Train')
plt.plot(history_without_aug.history['val_accuracy'], label='Without Aug - Val')
plt.plot(history_with_aug.history['accuracy'], label='With Aug - Train')
plt.plot(history_with_aug.history['val_accuracy'], label='With Aug - Val')

plt.title('Model Accuracy with and without Data Augmentation')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot training and validation loss for both configurations
plt.subplot(1, 2, 2)
plt.plot(history_without_aug.history['loss'], label='Without Aug - Train')
plt.plot(history_without_aug.history['val_loss'], label='Without Aug - Val')
plt.plot(history_with_aug.history['loss'], label='With Aug - Train')
plt.plot(history_with_aug.history['val_loss'], label='With Aug - Val')


plt.title('Model Loss with and without Data Augmentation')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

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

In [ ]: